User's Manual

Renesas Synergy™ Software Package (SSP) v1.4.0

User's Manual

Renesas Synergy™ Platform

Renesas Electronics
www.renesas.com

Rev.1.03 Apr 2018

Table of Contents

Table of Contents

Table of Contents

Table of Contents

SSP User's Manual

Chapter 1 Renesas Synergy Software Package (SSP)

1.1 Renesas Synergy™ Software Package Introduction

1.1.1 Introduction to the SSP User's Manual

This manual describes how to use the Renesas Synergy Software Package for writing applications for the Synergy microcontroller series. In the figure below, the API Reference components of the SSP User's Manual are indicated in blue. Additional components such as the description of the e^2 studio ISDE and tutorials and example applications are included in this manual to guide you through the steps of programming with the SSP.

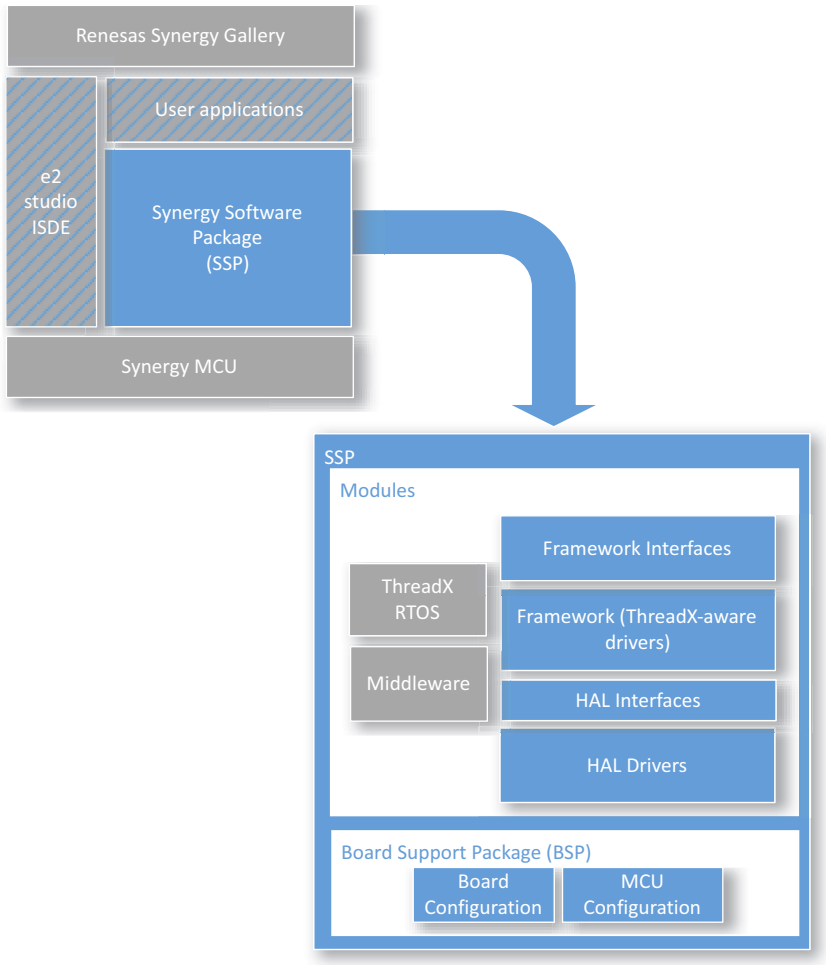


Figure 1: Synergy Software Package (SSP) Documentation

1.1.2 Subjects Covered in this Manual

To learn about the SSP architecture and about board and chip-level support included in the SSP see:

- SSP Architecture
- BSP Architecture

For programming with the SSP and an introduction to the e^2 studio ISDE see:
e^2 studio ISDE User Guide

For introductory tutorials and application examples see:

- Tutorial: Your First Synergy Project - Blinky

- Tutorial: Using HAL Drivers - Programming the WDT

For Module Overviews describing the SSP Modules, see:

- Framework Layer
- HAL Layer

The API reference documentation of the following SSP components is included in this document:

- API Reference: Framework Interfaces for Interfaces to the ThreadX-aware Framework Modules
- API Reference: Framework Layer for ThreadX-aware Framework Driver Modules
- API Reference: HAL Interfaces for Interfaces to the Hardware Abstraction Layer (HAL) Modules
- API Reference: HAL Layer for the RTOS-independent HAL driver Modules
- Board Support Package for the Board Support Package (BSP) which includes board-specific and microcontroller-specific configuration modules

Chapter 2 SSP Overview

Learn how to develop applications with the Synergy Software Package (SSP) using the SSP's module-based architecture and the functional software layers. Integrate SSP applications with multiple boards and Synergy devices using the Board Support Package (BSP).

The following pages describe the fundamental SSP architecture:

- SSP Architecture
- BSP Architecture

2.1 SSP Architecture

2.1.1 SSP Architecture

This section describes the Synergy Software Package (SSP) architecture and how to use the SSP Application Programming Interface (API).

2.1.1.1 Introduction to the SSP

As microcontrollers increase in complexity, so does the breadth of knowledge required to make them operate in the desired way. The Synergy Software Package (SSP) is a software package that we provide to our users for the purpose of allowing them to start their development at a higher level than they have in the past. This does not mean hiding every complexity, but instead presenting it to the user in a way that is easily understandable, and quickly modifiable. Instead of requiring the user to refer to a hardware register at every step, we instead offer a well-documented function call.

2.1.1.2 Prerequisite Information

The following sections provide prerequisite information for using the SSP.

For SSP tools, see e^2 studio ISDE User Guide

2.1.1.3 C99 Use

The SSP uses the ISO/IEC 9899:1999 (C99) C programming language standard. Specific features introduced in C99 that are used include standard integer types (stdint.h), booleans (stdbool.h), designated initializers, and the ability to intermingle declarations and code.

2.1.1.4 Use of const in API parameters

The *const* qualifier is used with API parameters whenever possible. An example case is shown below.

```
ssp_err_t (* open)(flash_ctrl_t * const p_ctrl,

    flash_cfg_t const * const p_cfg);
```

While not fool-proof by any means this does provide some extra checking inside the SSP code to ensure that arguments that should not be altered are treated as such.

2.1.1.5 Doxygen

Doxygen is the default documentation tool used by SSP. You can find Doxygen comments throughout the document when exploring the SSP source.

2.1.1.6 Weak Symbols

Weak symbols are used occasionally in and with the SSP. They are used to ensure that a project builds even when the user has not defined an optional function.

2.1.1.7 SSP Terms

Term	Description	Reference
Module	Modules can be peripheral drivers, purely software, or anything in between. Each Module consists of a folder with source code, documentation, and anything else that the customer needs to use the code effectively. Modules are independent units, but they may depend on other Modules. Example SSP Modules are the UART driver (UART Interface), Audio Framework which relies on timer, DMA, and DAC drivers (Audio Framework Interface), or the Messaging Framework (Messaging Framework Interface). Applications can be built by combining multiple modules to provide the user with the features they need.	SSP Modules
BSP	Short for Board Support Package. In the SSP the BSP provides just enough foundation to allow other SSP modules to work together without issue.	BSP Architecture

Term	Description	Reference
Callback Function	This term refers to a function that is called when an event occurs. For example, the bus error interrupt handler is implemented in the r_bsp. The user will likely want to know when a bus error occurs. To alert the user, a callback function can be supplied to the r_bsp. When a bus error occurs the r_bsp will jump to the provided callback function and the user can handle the error. Interrupt callback functions should be kept short and be handled carefully because when they are called the MCU will still be inside of an interrupt and therefore will be delaying any pending interrupts.	-
Interface	See SSP Interfaces section: SSP Interfaces. All interfaces in the SSP are listed here: API Reference: Framework Interfaces and API Reference: HAL Interfaces	SSP Interfaces
Instance	See SSP Instances section: SSP Instances	SSP Instances
Module Instance	Single and independent configuration of a Module.	-
Application	Code that is owned and maintained by the user. Application code may be based off sample application code provided by Renesas, but is the responsibility of the user.	An example for a simple application is included as tutorial in this manual: Tutorial: Using HAL Drivers - Programming the WDT
Driver	A Driver is a specific kind of Module that directly modifies registers on the MCU.	-
Stacks	The SSP architecture is designed such that Modules work together to form a Stack. Starting with the uppermost Module and going to the bottommost dependency forms a specific Stack.	SSP Stacks

Term	Description	Reference
Layer/Level	Stacks are made of multiple layers of Modules. A Layer can consist of one or multiple Modules depending on the requirements of the next Layer up. Layer and Level are used interchangeably.	SSP Predefined Layers

2.1.1.8 SSP Modules

Modules are the core building block of SSP. Modules can do many different things, but all Modules share the basic concept of providing functionality upwards and requiring functionality from below.

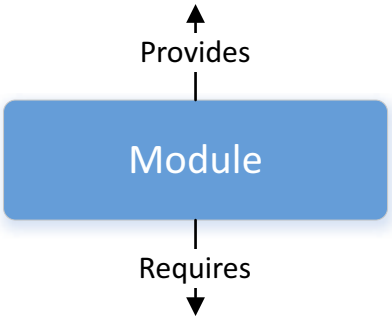


Figure 2: Modules

The amount of functionality provided by a Module is not limited though there are usually points where separation makes sense. If too much functionality is provided, then reuse of the Module can become difficult in the future. If not enough functionality is provided, then unnecessary complexity and overhead may be added in order to make the Modules work as expected.

The simplest SSP application consists of one Module with the user application on top.

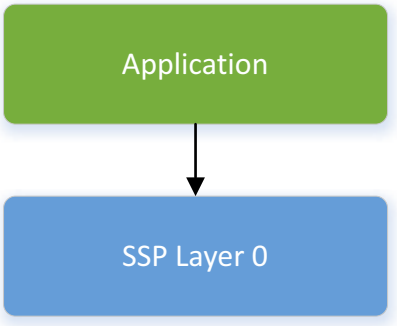


Figure 3: Module with application

For simplicity, ignore the Board Support Package (BSP) for now because it is a requirement of any SSP project. In the picture above, the BSP is located underneath the bottom layer, SSP Layer 0.

2.1.1.9 SSP Stacks

When Modules are layered atop one another, an SSP Stack is formed. The stacking process is performed by matching what one Module provides with what another Module requires. For example, the Audio Playback Framework Module requires a Transfer Interface which can be fulfilled by the Data Transfer Controller (DTC) Driver Module. Instead of including the DTC code in the Audio Playback Module, we split these into two Modules. This allows for reuse of the underlying Modules which has many benefits.

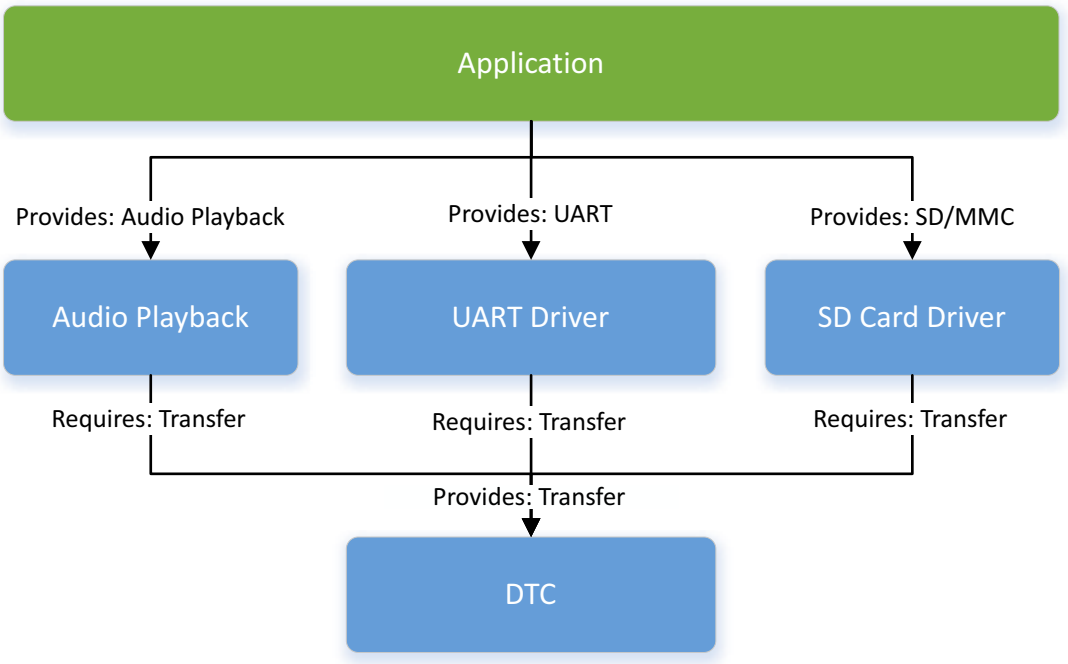


Figure 4: Stacks – Audio playback

By continuing to add layers to the Stack using SSP Modules, you can interface with the Synergy hardware at a high level.

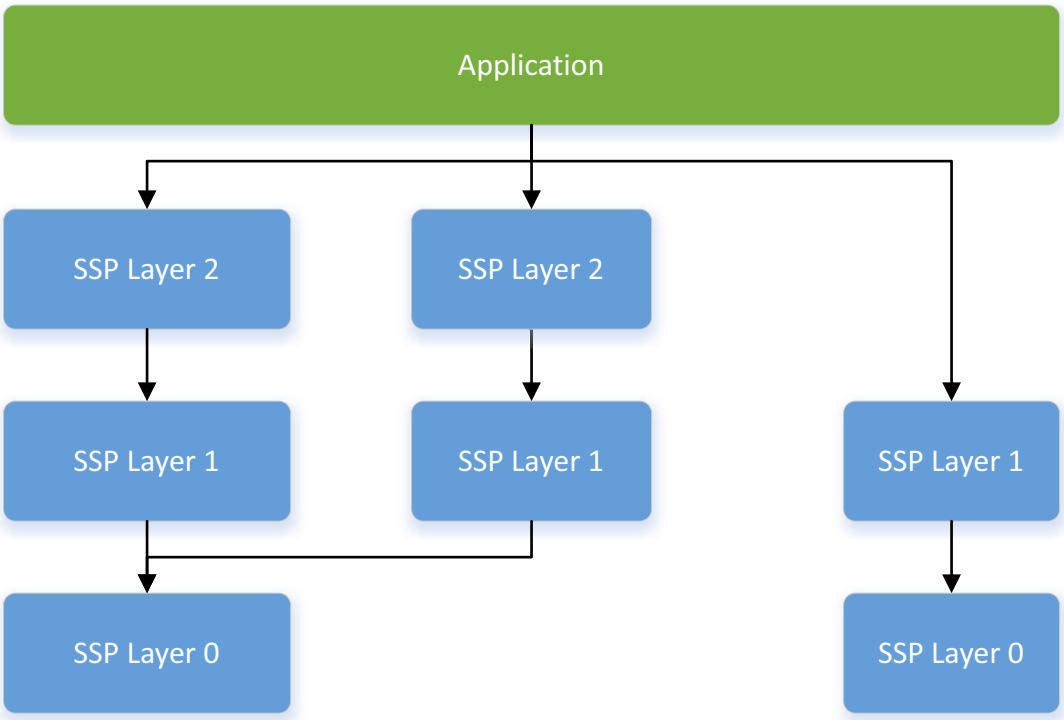


Figure 5: Stacks

The ability to stack modules has great benefit because it ensures the flexibility of the architecture as a whole. If modules are directly dependent upon other modules, then issues arise when application features must work across different user designs. To ensure that modules are reusable the modules must be capable of being swapped out for other modules that provide the same features. The SSP architecture provides this flexibility to swap modules in and out through the use of SSP Interfaces.

2.1.1.10 SSP Interfaces

At the architecture level, Interfaces are the way that Modules provide common features. This commonality allows Modules that adhere to the same Interface to be used interchangeably. Interfaces can be thought of as a contract between two Modules. The Modules agree to work together using the information that was agreed upon in the contract.

On Synergy hardware there is occasionally an overlap of features between different peripherals. For example, I2C communications can be achieved through use of the IIC peripheral or the SCI peripheral in its Simple I2C mode. There is a difference in the level of features provided by both peripherals. In I2C mode the SCI peripheral will only support a subset of the features of the full-featured IIC.

Interfaces aim to provide support for the common features that most users would expect. This means that some of the advanced features of a peripheral, such as the IIC, might not be available in the Interface. In most cases these features are still available through Interface Extensions.

In design, Interfaces are implemented as header files. All Interface header filenames end with '*_api.h'. The following sections detail what makes up an Interface.

SSP Interface Enumerations

Whenever possible, Interfaces use typed enumerations for function parameters and structure members.

```
typedef enum e_i2c_addr_mode

{

    I2C_ADDR_MODE_7BIT = 1, // Use 7-bit addressing mode

    I2C_ADDR_MODE_10BIT     // Use 10-bit addressing mode

} i2c_addr_mode_t;
```

Enumerations remove any uncertainty when deciding what values are available for a parameter. Also note that enumeration options follow a strict naming convention where the name of the type is prefixed on the available options. Combining the naming convention with the autocomplete feature available in e^2 studio provides the benefits of rapid coding while maintaining highly readable code.

SSP Interface Data Structures

At a minimum, all SSP Interfaces include three data structures: a control structure, a configuration structure, and an instance structure.

The control structure is used as a unique identifier for using the module. If SSP modules were only peripheral drivers then this control structure might be replaced with a channel number. All function calls for that module would then take a channel number so that the code could determine which peripheral channel to operate on. SSP modules are not restricted to device drivers and therefore the control structure is used. The user allocates storage for a control structure and then sends a pointer to it into the open() call for a Module. At this point, the Module initializes the structure as needed. The user must then send in a pointer to the control structure for all subsequent Module calls. The contents of the control structure are used by the Module and must not be altered by the user.

The contents of a control structure vary depending on the Module. Modules that implement device drivers may have very small control structures which only identify the channel being used. Usually this means that the Module will be statically allocating memory internally. In this case, the Module knows exactly how many channels are available. Purely software Modules cannot operate that way because the number of instances is typically only limited by amount of memory available. Dynamic memory allocation through use of the malloc() and free() functions are not used in SSP modules.

The configuration structure is used for the initial configuration of a Module during the open() call. The structure consists of members such as: channel, bitrate, and operating mode. The structure is used purely for input into the Module. This structure does not have to be unique and could be discarded by the user after initialization if desired.

```
typedef struct st_i2c_cfg

{

/* Generic configuration */

    uint32_t channel; // Identifier recognizable by underlying instance

    i2c_rate_t rate;  // Device's maximum clock rate in Hz, i.e. 400000

    uint16_t slave; // The address the device will respond to
```

```
    i2c_addr_mode_t addr_mode; // Indicates how slave fields should be interpret
ed


    /* Parameters to control software behavior */

    void (*p_callback)(i2c_callback_args_t * p_args); // Pointer to callback fun
ction

    void const * p_context; // Pointer to the user-provided context

    /* Instance-specific configuration */

    void const * p_extend; // Any configuration data needed by underlying hardwa
re

} i2c_cfg_t;
```

Above is an example configuration structure for the I2C Interface. The last three structure members (p_callback, p_context, and p_extend) are common to almost all module configurations.

The *p_callback* and *p_context* members are described in the SSP Interface Callback Functions section.

The *p_extend* member is used for extending the current Interface for a specific Instance. Interfaces are designed to support the most common features. There are cases where an Instance of an Interface requires extra information to properly configure itself. There are also cases where the extra information is not required, but users might need it to adjust the module for their specific application. When this is the case, the user can provide the underlying Instance with more configuration information by passing it through the *p_extend* member. The information that is passed through this member is defined by the underlying Instance, and therefore the user must adhere to its structure. If invalid information is passed to the underlying driver, then the Instance is not able to successfully use the data and proper operation cannot be guaranteed. Refer to the Interface Extensions section for more information.

It is also important that configuration structures only have members that apply to the current Interface. If multiple layers in the same stack define the same configuration parameters then it becomes difficult to know where to modify the option. For example, the baud rate for a UART is only be defined at the Driver layer. Any layers that use the UART Interface rely on the baud rate being provided at the Driver layer and do not offer it in their own configuration structures.

SSP Interface Callback Functions

Callback functions allow Modules to asynchronously alert the user application when an event has occurred. An example for an event is when a byte has been received over a UART channel. Callbacks are required to allow user application code to react to interrupts. SSP Modules define and handle the interrupts for Synergy MCU peripherals. If the user tries to define the interrupt service routine at the same time as a SSP Module, then the code does not build. Therefore SSP Modules allow the user application to respond to interrupts by registering a function to be called when an interrupt occurs.

Callback functions must be defined in the user application. They always return *void* and take a structure for their one parameter. The structure typedef is provided in the Interface for the Module and is named * <interface>_callback_args_t*. The contents of the structure may vary depending on the Interface, but two members are common: *event* and *p_context*.

The *event* member is used by the application to determine why the callback was called. Using the UART example again, the callback could have been triggered because a byte was received, all bytes had been transmitted, or a framing error has occurred. The *event* member is an enumeration provided by the Interface.

The *p_context* member is used for providing user-specified data to the callback function. In many cases a callback function is shared between multiple channels or Module Instances. When the callback occurs, the code handling the callback needs context information so that it can figure out which Module Instance the callback is for. For example, if the callback wanted

to make a SSP API call in the callback, then at a minimum the callback must use the control structure. To make this easy, the user can provide a pointer to the control structure as the *p_context*. When the callback occurs, the control structure is available as it will be passed in the callback structure.

Callback functions are called from within an interrupt service routine. For this reason callback functions should be kept as short as possible so they do not affect the real time performance of the user's system. An example skeleton function for the Flash Interface callback is shown below.

```
static void flash_callback (flash_callback_args_t * p_args)

{

    /* See what event caused this callback. */

    switch (p_args->event)

    {

        case FLASH_EVENT_ERASE_COMPLETE:

        /* Handle event. */

        break;

        case FLASH_EVENT_WRITE_COMPLETE:

        /* Handle event. */

        break;

        case FLASH_EVENT_BLANK:

        /* Handle event. */

        break;

        case FLASH_EVENT_NOT_BLANK:

        /* Handle event. */

        break;

        case FLASH_EVENT_ERR_DF_ACCESS:

        /* Handle error. */

        break;

        case FLASH_EVENT_ERR_CF_ACCESS:

        /* Handle error. */
```

```
        break;

        case FLASH_EVENT_ERR_CMD_LOCKED:

        /* Handle error. */

        break;

    }

}
```

When a Module is not directly used in the user application (it is not the top layer of the stack) then its callback function will be handled by the Module above. If there is a Console Interface Module that requires a UART Interface Module then the Console Module will control and use the UART's callback function. In this case the user does not need to create a callback function for the UART Module in their application code.

SSP Interface API Structure

All Interfaces include an API structure which contains function pointers for all the supported Interface functions. An example structure, with the comments removed, for the Digital to Analog Convert (DAC) is shown below.

```
typedef struct st_dac_api

{

    ssp_err_t (*open)(dac_ctrl_t *p_ctrl, dac_cfg_t const *const p_cfg);

    ssp_err_t (*close)(dac_ctrl_t *p_ctrl);

    ssp_err_t (*write)(dac_ctrl_t *p_ctrl, dac_size_t *p_value);

    ssp_err_t (*start)(dac_ctrl_t *p_ctrl);

    ssp_err_t (*stop)(dac_ctrl_t *p_ctrl);

    ssp_err_t (*versionGet)(ssp_version_t *p_version);

} dac_api_t;
```

The API structure is what allows for Modules to easily be swapped in and out for other Modules that are Instances of the same Interface. Let's look at an example application using the DAC Interface above.

Synergy MCUs have an internal DAC peripheral. If the DAC API structure in the DAC Interface were not used, then the application could make calls directly into the module. In the example below the application is making calls to the R_DAC_Write function which is provided in the r_dac module.

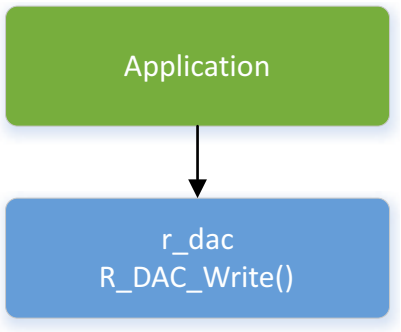


Figure 6: DAC Write example

Now let's assume that the user needs more DAC channels than are available on the MCU by adding a new external DAC module named r_dac_external. The external DAC uses I2C for communications. The application must now distinguish between the two modules, which adds complexity and further dependencies to the application.

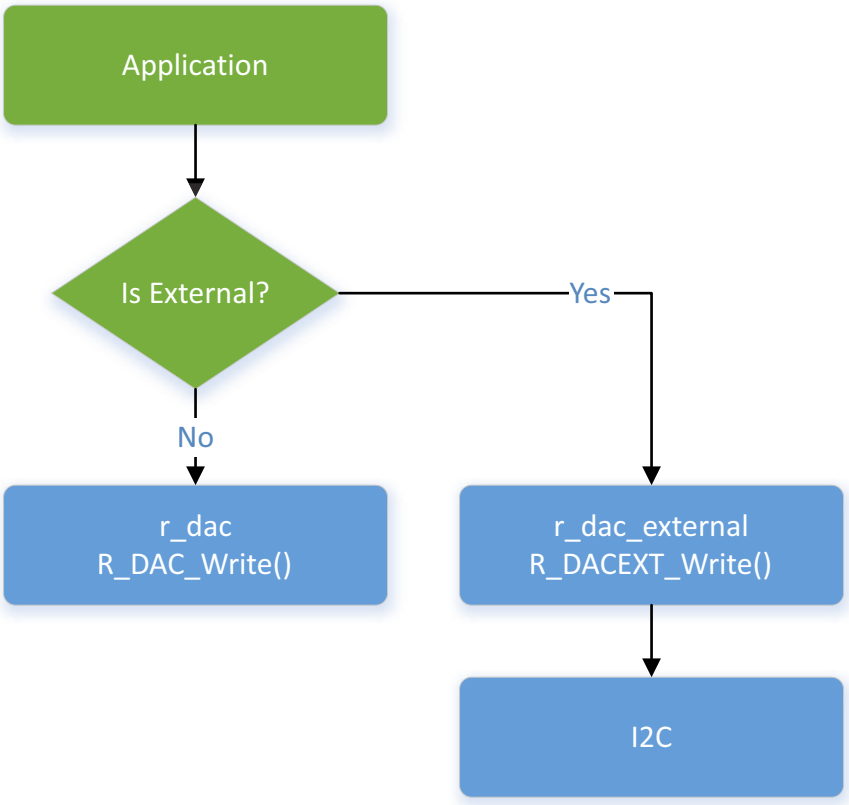


Figure 7: DAC Write with two write modules

The use of Interfaces and the API structure allows for the use of an abstracted DAC. This means that no extra logic is needed and the application no longer depends upon certain hard-coded Modules. Instead the application now depends on the DAC Interface API which can be implemented by any number of Modules.

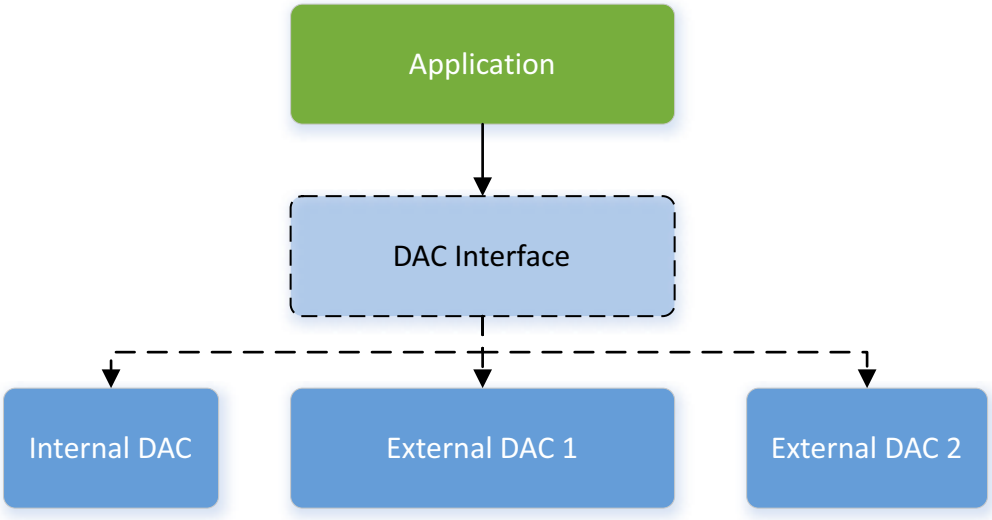


Figure 8: DAC Interface

Functions inside of the API structures follow common names. Most Modules will have a pair of open() and close functions. The open() function must be called before any of the other functions. The only exception is the versionGet() function which is not dependent upon any user provided information.

Other functions that will commonly be found are read(), write(), get(), and set(). Function names are designed to be a noun followed by a verb. Example names include:

- read(), write(), writeRead()
- statusGet()
- calendarAlarmSet(), calendarAlarmGet()
- accessWindowSet(), accessWindowClear()

SSP Interface Version Information

All Interfaces supply a versionGet() function. This function fills in a structure of type *ssp_version_t*. This structure is made up of two versions: one for the Interface (the API) and one for the underlying Instance that is currently being used.

```
typedef union st_ssp_version

{

uint32_t version_id;

struct
```

```
{

    uint8_t code_version_major; // Code major version

    uint8_t code_version_minor; // Code minor version

    uint8_t api_version_major;  // API major version

    uint8_t api_version_minor;  // API minor version

};

} ssp_version_t;
```

The API version ideally never changes, and only rarely if it does. A change to the API may require users to go back and modify their code. The code version, the version of the current Instance, may be updated more frequently. Bug fixes, enhancements, and additional features may all bump the code version. Changes to the code version will only require changes to the user code if the user code is using extended features provided by the Instance.

SSP Instances

While Interfaces dictate the features that are provided, Instances actually implement those features. Each Instance is tied to a specific Interface. Instances use the enumerations, data structures, and API prototypes from the Interface. This allows for an application that uses an Interface to swap out the Instance when needed.

On Synergy MCUs some peripherals will have a one-to-one mapping between the Interface and Instance, while others will have a one-to-many. In the example below the IIC and SPI peripherals map to only one Interface each while the SCI peripheral implements three Interfaces.

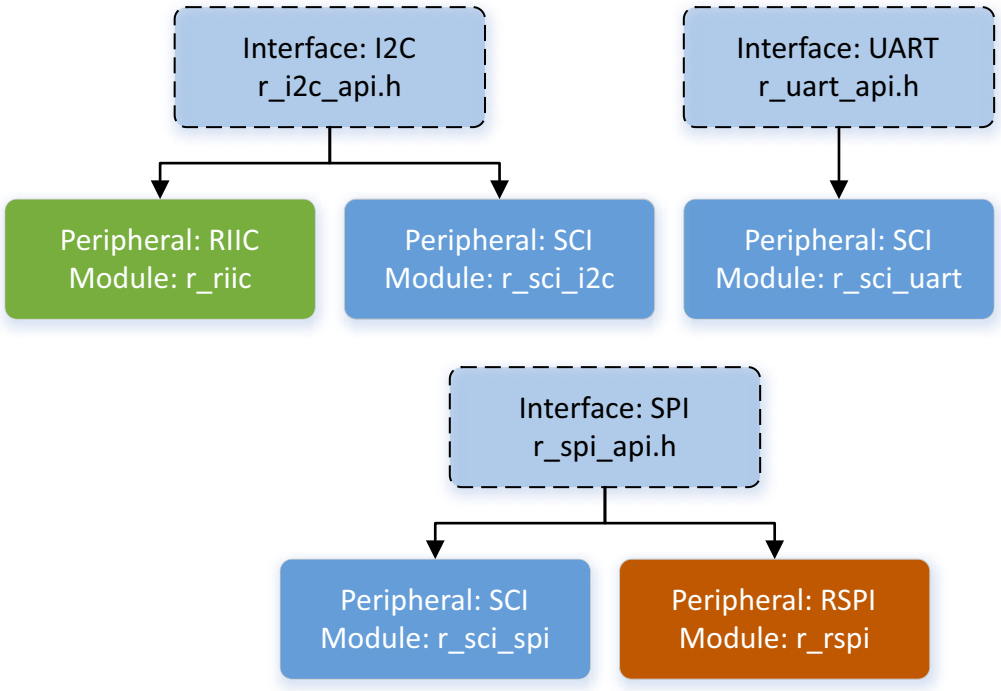


Figure 9: Instances

SSP Instances API Structure

Each Instance includes a constant global structure with its functions that implement the Interface's API. The name of this structure is standardized as g_<*interface*>_on_<*instance*>. Examples include g_spi_on_spi, g_transfer_on_dtc, and g_adc_on_adc. This structure is available to be used through an extern in the instances header file (r_spi.h, r_dtc.h, and r_adc.h respectively).

2.1.1.11 Build Time Configuration

All modules have a build-time configuration header file. Most configuration options are supplied at run time. Some options that are rarely used, or apply to all instances of a module, may be moved to build time. The advantage of using a build-time configuration option is to potentially reduce code size reduction by removing an unused feature.

All modules have at least one build time option, which is whether to enable or disable parameter checking for the module. SSP modules check function arguments for validity when possible. Some users may want to disable this feature when their testing has concluded to save code space and to speed up execution.

2.1.1.12 Interface Extensions

In some cases, Instances require more information than is provided in the Interface. This situation can occur in the following two cases:

- An Instance offers extra features that are not common to most Instances of the Interface.

- An Interface must be very generic out of necessity. As an Interface becomes more generic, the number of possible Instances increases. A prime example of this is the Block Media Interface.

```
typedef struct st_sf_block_media_cfg

{

    uint32_t block_size; // Block size in bytes

    void * p_extend; // Instance dependent configuration

} sf_block_media_cfg_t;
```

The configuration structure for the Block Media Interface is intentionally sparse. This allows for nearly endless Instances. Possible Instances include SD card, SPI Flash, SDRAM, USB, and many more. Different configuration information is needed for each Instance. This is accomplished by supplying the information through the *p_extend* parameter. While the configuration data provided in the *p_extend* is not the same between Instances, the API calls thereafter will be. This means that the change is only required in one place.

Use of Interface extensions is not always necessary. Some Instances do not offer an extension since all functionality is provided in the Interface. In many cases the *p_extend* member can be set to NULL. If NULL is provided and the Instance does offer an extension then the Instance will take this to mean that the default options should be used. The documentation for each Instance indicates whether an Interface extension is provided and whether its use is mandatory or optional.

2.1.1.13 SSP Predefined Layers

The SSP comes with two predefined layers: the Driver layer and the Framework layer. The layers are easily identifiable because the modules reside in different folders and have different prefixes. Driver layer modules are located in the *ssp/src/driver* folder, while Framework level modules are located in the *ssp/src/framework* folder. Modules in the Driver layer start with an *r_* prefix, while Framework level modules start with a *sf_* prefix.

The core difference in the functionality between the layers is that Driver layer modules are restricted to being peripheral drivers that are RTOS aware, but do not use any RTOS objects or make any RTOS API calls. This means that Driver layer modules can be used in applications with, or without, an RTOS.

Framework layer modules are free to use RTOS objects such as semaphores, mutexes, or event flags. Framework modules may also create their own when needed. Framework layer modules that need to access hardware typically do so through a Driver layer Interface. Exceptions can be granted in special cases where multiple peripherals need to be used together in a way that would not be practical through multiple individual Interfaces.

2.1.1.14 SSP File Structure

The high-level file structure of the SSP is shown below.

```
ssp

+---inc

|   +---bsp

|   +---driver
```

```
|   |   +---api
|   |   \---instances
|   \---framework
|       +---api
|       +---el
|       +---instances
|       \---tes
\---src
    +---bsp
    |   +---cmsis
    |   \---mcu
    +---driver
    |   \---r_module
    \---framework
        +---el
        +---sf_module
        \---tes
ssp_cfg
+---bsp
+---driver
\---framework
```

Directly underneath the base *ssp* folder the folders are split into the source and include folders. Include folders are kept separate from the source for easy browsing and easy setup of include paths. The the same set of folders are located in the *ssp/inc* and *ssp/src* folders: *bsp, driver,* and *framework*.

Apart from the BSP, the SSP's two predefined layers, Driver and Framework, are present. Driver layer modules are located in the *ssp/src/driver* folder and Framework layer modules are located in *ssp/src/framework*. Under the include tree, the Driver and Framework layer folders contain two folders each: *api* and *instances*. The *api* folder contains the Interface

header files for that layer. The *instances* folder contains the Instance header files for that layer. Both layers are flat internally which limits the number of include paths required for a project.

The *ssp_cfg* folder is where configuration header files are stored for each module. Its layout is the same as the *ssp* folder where the BSP, Driver, and Framework layers have separate flat directories. See the Build Time Configuration section for information on what is provided in these header files.

2.1.1.15 SSP Connecting Layers

SSP modules are meant to be both reusable and stackable. It is important to remember that modules are not dependent upon other modules, but upon other Interfaces. The user is then free to fulfill the Interface using the Instance that best fits their needs.

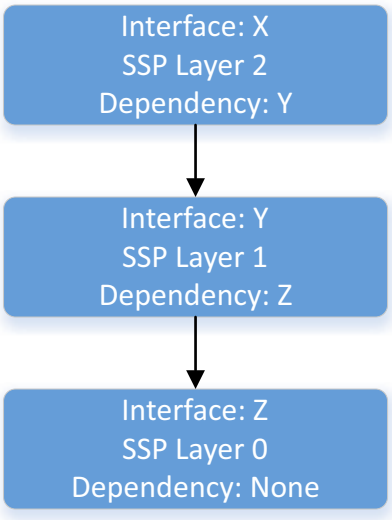


Figure 10: Connecting layers

In the image above Interface Y is a dependency of Interface X and has its own dependency on Interface Z. Interface X only has a dependency on Interface Y. Interface X has no knowledge of Interface Z. This is a requirement for ensuring that layers can easily be swapped out. This is shown in the next diagram.

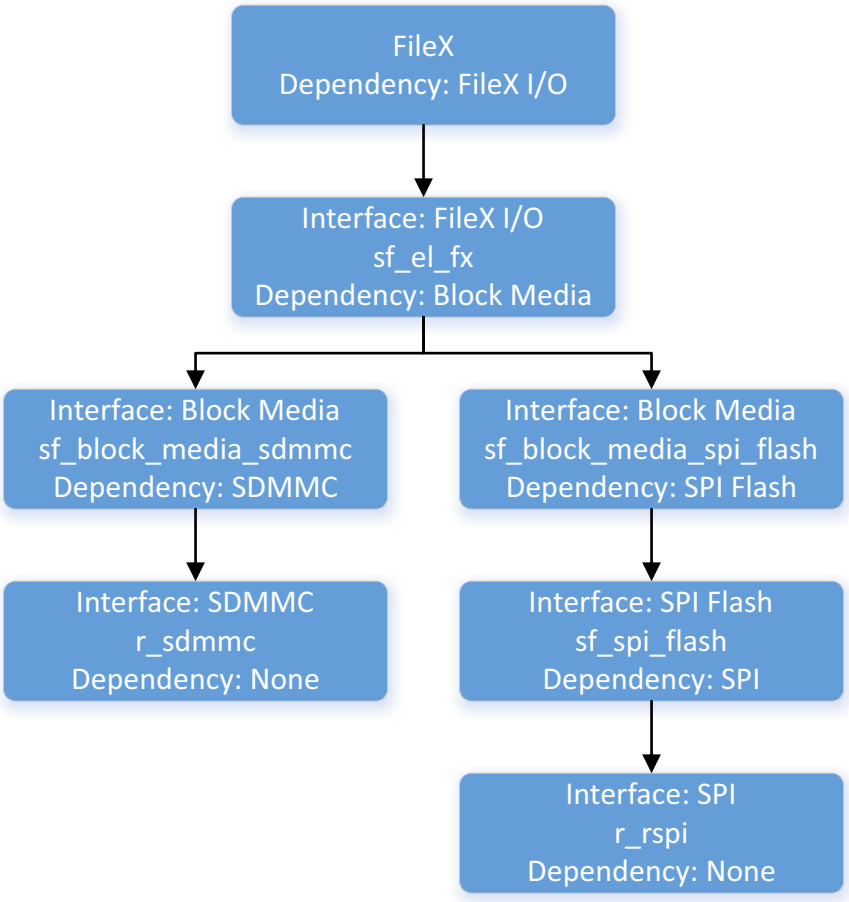


Figure 11: Connecting layers with the FileX interface

In this example we are using the Express Logic, Inc. FileX file system on two storage mediums: SDMMC and SPI Flash. The SPI Flash Interface takes care of the SPI flash protocol but requires a SPI Interface for actual SPI bus communications. The SDMMC Interface takes care of the protocol and the bus communications meaning that it does not have any dependencies.

For details on the SDMMC and FileX Interfaces, see FileX Port Block Media Framework.

2.1.1.16 SSP Architecture In Practice

Each layer in the SSP Stack is responsible for calling the API functions of its dependencies. This can also be described by saying that users are only responsible for calling the API functions at the layer at which they are interfacing. Using the FileX example above, the user is only responsible for calling FileX functions in the application code. Internally, FileX then calls FileX I/O, which in turn calls a Block Media Interface Module. The Block Media Interface can call multiple drivers. At a minimum an upper layer Module calls the *open()* function of the Interface it depends upon.

To write an application using a Module, you must determine the following:

1) Determine which open() function to call. Dependencies are based upon Interfaces which means that a Module must have some way of discerning which Instance to call.
2) Determine the configuration parameters. The Module also needs to know what configuration information to pass down. In some cases the Module requires certain configuration parameters to be set. If this is the case then the module sets these configuration structure members itself before passing on the rest of the structure. The rest of the configuration structure members must be provided outside of the Module.
3) Provide a control structure which is Module Instance specific and therefore can be allocated by the upper layer module.

Putting this all together means that to interact with a Module Instance, the following is needed:

- A pointer to the Instance's API structure
- A pointer to the Module Instance's configuration structure
- A pointer to the Module Instance's control structure

This information is sufficient to use any module. Notice that the API structure is the only structure that is Instance specific; not *Module Instance* specific. This is because the API structure will not vary between multiple uses of the same Instance. If SPI is being used on SCI channels 0 and 2 then both Module Instances will use the same API structure while the configuration and control structures will vary.

To make Module Instances easier to use, all of these pieces are encapsulated in instance structures found in each Interface. These structures have a standardized name of * <interface>_instance_t*. An example from the WDT interface is shown below.

```
typedef struct st_wdt_instance

{

    wdt_ctrl_t *p_ctrl; // Pointer to the control structure for this instance

    wdt_cfg_t const *p_cfg; // Pointer to the configuration structure for this i
nstance

    wdt_api_t const *p_api; // Pointer to the API structure for this instance

} wdt_instance_t;
```

Upper layer modules that have a dependency on an Interface can then use the instance structure to hold everything needed to interact with an Instance of that Interface. Continuing with the WDT example above, below is the Thread Monitor Framework Interface configuration structure. The Thread Monitor Interface is dependent upon the WDT Interface.

```
typedef struct st_sf_thread_monitor_watchdog_type

{

    wdt_instance_t * p_lower_lvl_wdt; // Pointer to lower level watchdog instanc
e
```

```
    bool profiling_mode_enabled;      // Enables or disables profiling mode

    UINT priority;                    // Priority of thread monitor thread

} sf_thread_monitor_cfg_t;
```

The Thread Monitor module has everything it needs to work with the WDT Interface in the *p_lower_lvl_wdt* structure member.

In some cases module dependencies are not be defined in the Interface, but instead in the Instance. An example is the Block Media Interface could be implemented on SDMCC, SPI Flash, or many other Instances (also see). Because of the wide range of implementations, the instance structure for a particular Interface cannot be used directly in the Block Media Interface's configuration structure. The Block Media Interface's configuration structure is shown again below.

```
typedef struct st_sf_block_media_cfg

{

    uint32_t block_size; // Block size in bytes

    void * p_extend; // Instance dependent configuration

} sf_block_media_cfg_t;
```

Notice there are Instance structure pointers provided. The reason for this, as previously mentioned, is that the Block Media Interface is too generic to enforce a dependency upon a particular Interface.

When a Module is an Instance of a generic Interface, such as Block Media, and it has dependencies on other Modules, the module puts the lower-layer pointers in an extension structure that is referenced through the Interface's *p_extend* configuration member. This is required to allow Module stacking while not forcing Interfaces to expand and have many optional configuration members.

```
typedef struct st_block_media_on_sdmmc_cfg

{

    sdmmc_instance_t const * const p_lower_lvl_sdmmc; // Pointer to SDMMC instan
ce structure

} sf_block_media_on_sdmmc_cfg_t;
```

2.1.1.17 Using SSP Modules

This section will give general information on how to use a SSP module.

Pick an Interface

Start by picking an Interface for the functionality that is required. For example, for UART communications use the UART Interface.

Find a suitable Instance of the Interface

After picking an Interface, choose a suitable Instance. The list of known Instances of an Interface is listed in the documentation comments for an Interface. Include the header file of the selected Instance in the source file of the application that uses the Instance.

Allocate control and configuration structures

The e^2 studio ISDE provides a graphical user interface for setting the parameters of the Interface and Instance configuration structures. The ISDE also automatically includes those structures, once they are configured in the GUI, in application-specific header files that you can include into your application code.

To see how the ISDE handles the configuration, see Configuring a Project in the ISDE User's Guide: e^2 studio ISDE User Guide.

The configuration and control structure types follow standard names of <interface>_ctrl_t* and *<interface>_cfg_t* respectively. The ISDE allocates storage for both structures in the application specific header files, which the ISDE creates. Use the ISDE Properties view to set the values for the members of the configuration structure as needed. Many members will be typed enumerations in which case the enumeration can be referenced for available options.

If the Interface has a callback function option then you first need to declare and define the function in their source code. The return value is always of type *void* and the parameter to the function is a typed structure of name *<interface>_callback_args_t*. Once the function has been defined, assign its name to the *p_callback* member of the configuration structure. If any context information is required in the callback then the user can provide a pointer to the *p_context* member. You can assign callback function names through the ISDE Properties window for the selected Module.

Refer to the Instance documentation to see if an Interface extension is provided. If so, then it will be found in the Instance's header file and named *<interface>_on_<instance>_cfg_t*. It may have several members just like the Interface's configuration structure. When you select a driver with a specific Instance, you can select any parameter in the configuration structure of the instance in the ISDE property.

Interact using Interface's Instance Structure

Once the instance structure has been defined, you can interact with the Instance as needed. Below is code that builds up an instance structure for the UART Interface as implemented on SCI. Please note that when using e^2 studio for Synergy, the following code is automatically generated for the user.

```
/* Include the header file of the Instance. */

\#include "r_sci_uart.h" //This will in turn include the r_uart_api.h Interface

/* Allocate control structure. */

uart_ctrl_t my_uart_ctrl;

/* Setup extended UART configuration on SCI. */

uart_on_sci_cfg_t my_uart_extended_cfg =

{

    /* Set extended configuration members... */

};

/* Configure standard UART Interface. */
```

```
uart_cfg_t my_uart_cfg =

{

    .data_bits = UART_DATA_BITS_8,

    /* Continue configuring other members... */

    .p_extend = &my_uart_extended_cfg

};

/* Setup instance structure */

uart_instance_t my_uart = {

    .p_ctrl = &my_uart_ctrl,

    .p_cfg = &my_uart_cfg, //Extended configuration is brought through in p_exte
nd

    .p_api = &g_uart_on_sci //Defined in r_sci_uart.h

};
```

Now that the instance structure is ready, you can interact with the UART Interface. In e^2 studio, the name of the instance structure is the Name* that you provide when configuring the Module Instance in the ISDE Properties window.

```
ssp_err_t err;

/* Initialize UART */

err = my_uart.p_api->open(my_uart.p_ctrl, my_uart.p_cfg);

/* Check return for errors. */

if (SSP_SUCCESS != err)

{

/* Handle error. */

}

/* Use other Interface functions. */

err = my_uart.p_api->write(my_uart.p_ctrl, ...);
```

```
err = my_uart.p_api->read(my_uart.p_ctrl, ...);
```

2.2 BSP Architecture

2.2.1 BSP Architecture

This section describes the BSP or Board Support Package. For the API Reference see Board Support Package. The BSP is board specific and as a result also MCU specific.

2.2.1.1 BSP Related Terminology

Term	Meaning
system_xxxx.c or startup_xxxx.c	The 'xxxx' refers to the MCU type. For example, system_S7G2.c when referencing the S7G2 MCU.
BSP	Short for Board Support Package. BSP's usually have source files related to a specific board.
Callback Function	This term refers to a function that is called when an event occurs. For example, the NMI interrupt handler is implemented in the BSP. The user will likely want to know when an NMI system exception occurs. To alert the user, a callback function can be configured for the group interrupts (a group of exceptions all of which are tied to the NMI). When an NMI occurs the BSP will jump to the provided callback function and the user can handle the error. Interrupt callback functions should be kept short and be handled carefully because when they are called the MCU will still be inside of an interrupt and therefore will be delaying any pending interrupts.

2.2.1.2 What Does the BSP Do?

The BSP is responsible for getting the MCU from reset to the user's application (i.e. the main() function). Before reaching the user's application the BSP sets up the stacks, heap, clocks, interrupts, and C runtime environment. The BSP also configures and sets up the port I/O pins and performs any board specific initializations.

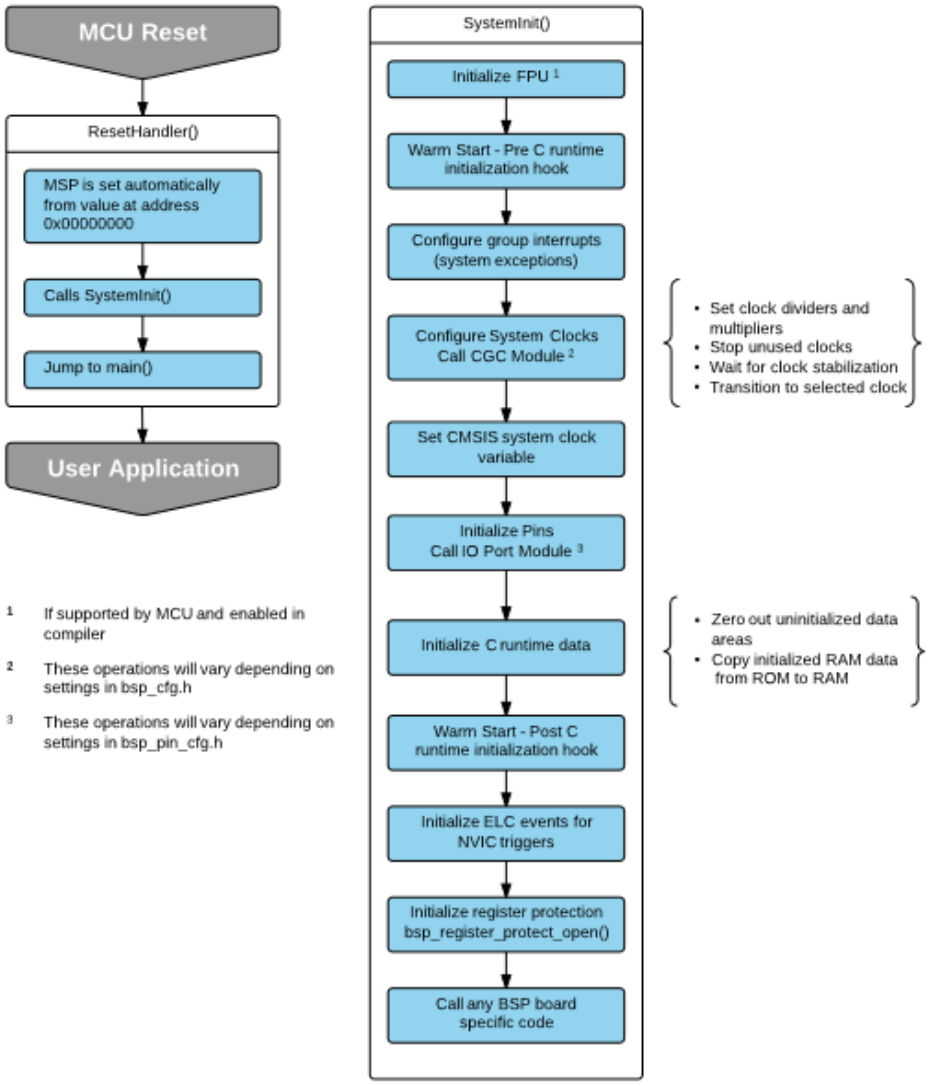


Figure 12: BSP flow

2.2.1.3 BSP Prerequisites

Supported Devices and Boards

The BSP has been tested on all Synergy microcontroller families. It currently supports the following boards:

- PE-HMI1
- DK-S124
- DK-S7G2

- SK-S7G2
- DK-S3A7
- PK-S5D9
- DK-S128
- TB-S3A6
- TB-S5D5
- TB-S3A3
- TB-S3A1
- TB-S1JA

2.2.1.4 BSP Directory Structure

The BSP is organized into folders containing MCU, board specific and CMSIS information.

Synergy is CMSIS-compliant and based on the CMSIS-Core. This requires that we follow CMSIS requirements and naming standards.

- Standardized definitions for processor peripherals
  - NVIC (Nested Vector Interrupt Controller)
  - Systick (System Tick Timer)
  - MPU (Memory Protection Unit)
- Standardized access functions to access processor features
  - NVIC_SetPriority()
  - NVIC_EnableIRQ
- Standardized function names for system exception handlers
  - Reset_Handler()
  - SysTick_Handler()
- Standardized functions for system initialization.
  - SystemInit() – defined in system_S7G2.c for S7G2 MCU's
- Standardized software variables for clock speed information
  - SystemCoreClock

The BSP directory structure is shown below:

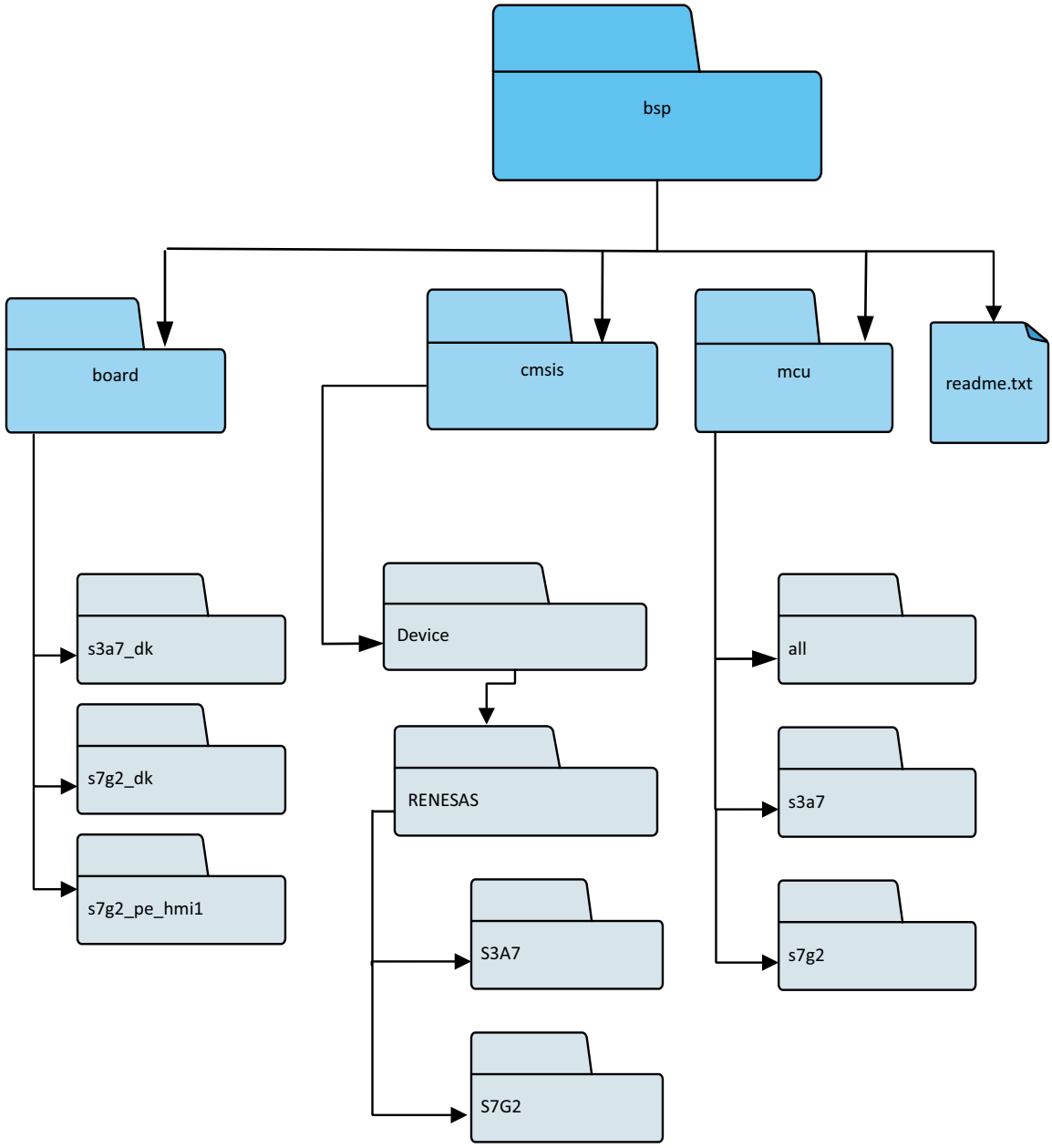


Figure 13: BSP directory structure

2.2.1.5 Configuring the BSP

The BSP is heavily data driven with most features and functionality being configured based on the content from configuration files. Configuration files represent the settings specified by the user and are generated by the ISDE when the Generate Project Content button is clicked.

2.2.1.6 BSP Configuration Settings

The table below describes each of the configurable BSP settings. Many of these settings are MCU specific and there are differences between the settings available for each of the supported MCU's.

Table: BSP Configuration options

BSP Property	Description
Part package	MCU package type and size
Part memory size (ROM/RAM/Data Flash)	Available memory configurations
Core and Frequency	ARM core type and max system frequency
Part series	MCU part series
Main stack size (bytes)	Size of the Main Stack. Must be > 0.
Process stack size (bytes)	Size of the Process Stack. Use of this stack is optional. If 0, then PSP use is disabled
Heap size (bytes)	Size of the heap in bytes. If 0, the heap is disabled.
OFS0 register settings: IWDT Start Mode, IWDT Timeout Period, IWDT Dedicated Clock Frequency Divisor, IWDT Window End Position, IWDT Window Start Position, IWDT Reset Interrupt Request Select, IWDT Stop Control, WDT Start Mode Select, WDT Timeout Period, WDT Clock Frequency Division Ratio, WDT Window End Position, WDT Window Start Position, WDT Reset Interrupt Request, WDT Stop Control	The option-setting memory determines the state of the MCU after a reset. It is allocated to the configuration setting area and the program flash area of the flash memory. See the MCU user manual for details.
OFS1 register settings: Voltage Detection 0 Circuit Start, Voltage Detection 0 Level, HOCO Oscillation Enable. S3 MPU has MPU configuration settings.	See the MCU user manual for details.
MPU - Enable or disable PC Region 0	Start block address for access window protection
MPU - PC0 Start, MPU - PC0 End, MPU - Enable or disable PC Region 1, MPU - PC1 Start, MPU - PC1 End, MPU - Enable or disable Memory Region 0, MPU - Memory Region 0 Start, MPU - Memory Region 0 End, MPU - Enable or disable Memory Region 1, MPU - Memory Region 1 Start, MPU - Memory Region 1 End, MPU - Enable or disable Memory Region 2, MPU - Memory Region 2 Start, MPU - Memory Region 2 End, MPU - Enable or disable Memory Region 3, MPU - Memory Region 3 Start, MPU - Memory Region 3 End	Secure MPU ROM register settings. See user manual for details.

BSP Property	Description
ID code 1, ID code 2, ID code 3, ID code 4	Sets the ID Code for boot mode and debugger access protection.
MCU Vcc (mV)	Some Modules (e.g. LVD) need to know the voltage supplied to the MCU. This information is obtained from here.
Parameter checking	Defines whether the global setting for parameter checking is enabled or disabled. Local modules will take this value by default but can be locally overridden.
RTOS being used	Defines whether or not an RTOS is being used with this BSP.
Assert Failures	Defines what happens when an assertion failure occurs.
Error Log	Defines whether or not errors are logged to ssp_error_log.

2.2.1.7 BSP Configuration Files

Configuration files are used by the BSP to set up ROM registers, clocks, interrupts, ELC events and initial pin configurations. These configuration files can be found in ssp_cfg\bsp.

Bsp_cfg.h

This configuration file represents the values for BSP system settings. These are the settings that can be modified from the ISDE BSP properties tab. They include ROM register settings, stack size, parameter checking and control of error logging.

The BSP tab and the Properties window are shown below.

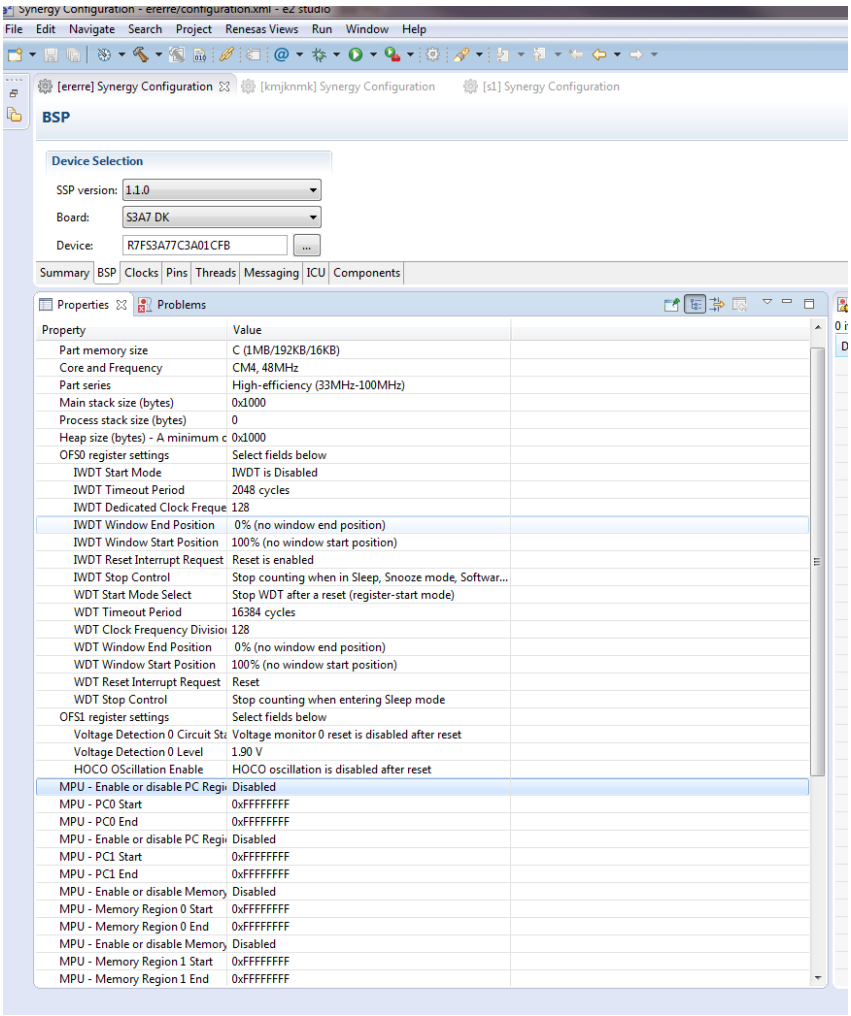


Figure 14: BSP Properties window

Some registers are located in ROM and therefore must be set at compile-time. These include some option-setting memory (OFS) registers as well as certain memory protection registers.

Option-setting memory determines the state of the MCU after a reset. For example, the IWDT can be configured and enabled, voltage detection can be enabled, and HOCO oscillation can be enabled. When these registers are set the operations are completed before the MCU's reset vector is fetched and execution begins.

Some Synergy MCUs include a Memory Protection Unit (MPU). The MPU is a programmable device that can be used to define memory access permissions (i.e. privileged access only or full access) and memory attributes (i.e. bufferable, cacheable) for different memory regions. The MPU can support up to eight programmable memory regions, each with their own programmable starting addresses, sizes and settings.

The ISDE configures these memory areas by setting values for the provided MPU settings. You must be careful when setting these registers. Incorrect settings can prevent access to required memory areas or prevent access to the MCU entirely.

Synergy MCUs have two mechanisms for protecting MCU memory from being read after production:

1) The use of ID codes. The ID code is 16 byte value that can be used to protect the MCU from being connected to a debugger or from connecting in Serial Boot Mode. There are different settings that can be set for the ID code; please refer to the hardware manual for your device for available options.
2) The use of a 4 byte value called ROM Code Protection. This value determines what read and write access parallel programmers have to the MCU.

2.2.1.8 BSP Pin Configuration

You can configure the pins used in your application through the ISDE pin configurator. See Configuring Pins.

Bsp_pin_cfg.h

This configuration file contains an array of pin configurations. During start-up, and before main() is executed, the BSP iterates over this array and initializes the MCU's port pins based on the settings in the array. Initially, before any pin configuration by the user, the ISDE **Pins** tab displays the initial reference configuration defined for the selected board type (see Configuring Pins). Once the user modifies the pin configuration and clicks **Generate Project**, a new bsp_pin_cfg.h file is generated containing the new pin configuration. The BSP always uses the bsp_pin_cfg.h file from ssp_cfg\bsp as the source for its pin configuration information, but the pin information generated by clicking **Generate Project** is written to a bsp_pin_cfg.h file in the hidden folder ssp\cfg\bsp\ .out.

In this way, the user can manually edit the bsp_pin_cfg.h in ssp\bsp without the fear of the file being overwritten by the project generation, while the Pin Configuration information generated by the ISDE also remains available for view or merging with the user's config file.

2.2.1.9 BSP Clock Configuration

All system clocks are set up during BSP initialization based on the settings in bsp_clock_cfg.h. These settings are derived from clock configuration information provided from the ISDE **Clocks** tab setting.

- Clock configuration is performed prior to initializing the C runtime environment to speed up the startup process, as it is possible to start up on a relatively slow (i.e. 32 kHz) clock.
- The BSP implements the required delays to allow the selected clock to stabilize.

Bsp_clock_cfg.h

This configuration file represents the values for system clock settings. These are the settings that can be modified from the ISDE **Clocks** tab. See: Configuring Clocks

2.2.1.10 System Interrupts

As Synergy MCUs are based on the Cortex-M ARM architecture, the NVIC Nested Vectored Interrupt Controller (NVIC) handles exceptions and interrupt configuration, prioritization and interrupt masking. In the ARM architecture, the NVIC handles exceptions. Some exceptions are known as System Exceptions. System exceptions are statically located at the "top" of the vector table and occupy vector numbers 1 to 15. Vector zero is reserved for the MSP Main Stack Pointer (MSP). The remaining 15 system exceptions are shown below:

- Reset
- NMI
- Cortex-M4 Hard Fault Handler

- Cortex-M4 MPU Fault Handler
- Cortex-M4 Bus Fault Handler
- Cortex-M4 Usage Fault Handler
- Reserved
- Reserved
- Reserved
- Reserved
- Cortex-M4 SVCall Handler
- Cortex-M4 Debug Monitor Handler
- Reserved
- Cortex-M4 PendSV Handler
- Cortex-M4 SysTick Handler

NMI and Hard Fault exceptions are enabled out of reset and have fixed priorities. Other exceptions have configurable priorities and some can be disabled.

2.2.1.11 Group Interrupts

Group interrupt is the term used to describe the 12 sources that can trigger the Non-Maskable Interrupt (NMI). When an NMI occurs the NMI Handler examines the NMISR (status register) to determine the source of the interrupt. NMI interrupts take precedence over all interrupts, are usable only as CPU interrupts, and cannot activate the Synergy peripherals Data Transfer Controller (DTC) or Direct Memory Access Controller (DMAC).

Possible group interrupt sources include:

- IWDT Underflow/Refresh Error
- WDT Underflow/Refresh Error
- Voltage-Monitoring 1 Interrupt
- Voltage-Monitoring 2 Interrupt
- VBATT monitor Interrupt
- Oscillation Stop is detected
- NMI pin
- RAM Parity Error
- RAM ECC Error
- MPU Bus Slave Error
- MPU Bus Master Error
- MPU Stack Error

A user may enable notification for one or more group interrupts by registering a callback using the BSP API function R_BSP_GroupIrqWrite. When an NMI interrupt occurs, the NMI handler checks to see if there is a callback registered for the cause of the interrupt and if so calls the registered callback function.

As mentioned earlier, the first 16 slots in the vector table are already accounted for by the system exceptions. Beginning with slot 16 are user configurable interrupts. These may be external, or peripheral generated interrupts.

The size of the NVIC interrupt table varies across Synergy MCU types (shown below).

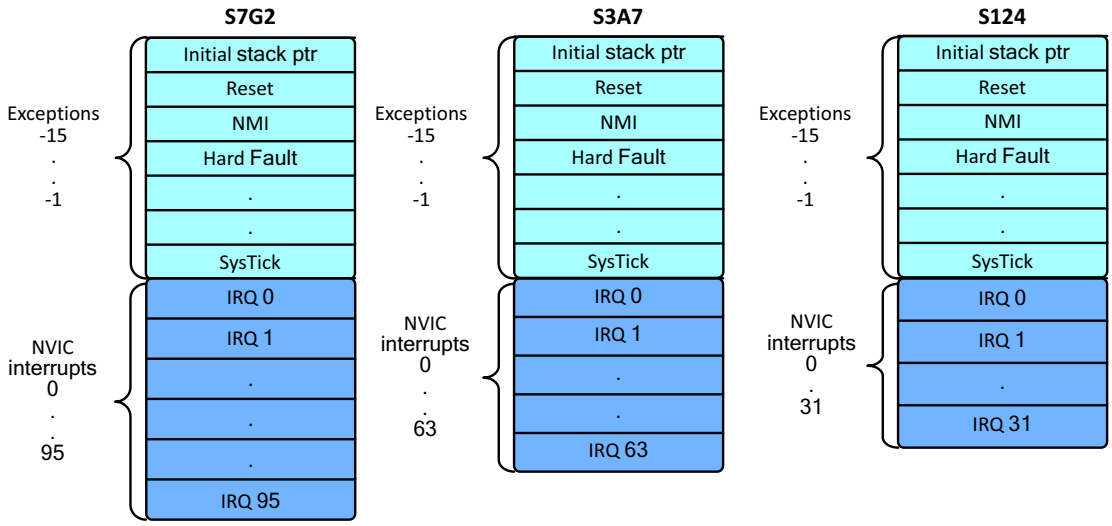


Figure 15: NVIC Interrupt vector table

Although the number of available slots for the NVIC interrupt vector table may seem small, the BSP defines up to 512 events that are capable of generating an interrupt. By using Event Mapping, the BSP maps user enabled events to NVIC interrupts. For an S7G2 MCU, only 96 of these events may be active at any one time, but the user has flexibility by choosing which events generate the active event.

The diagram below shows the interrupt vector table for the S7G2:

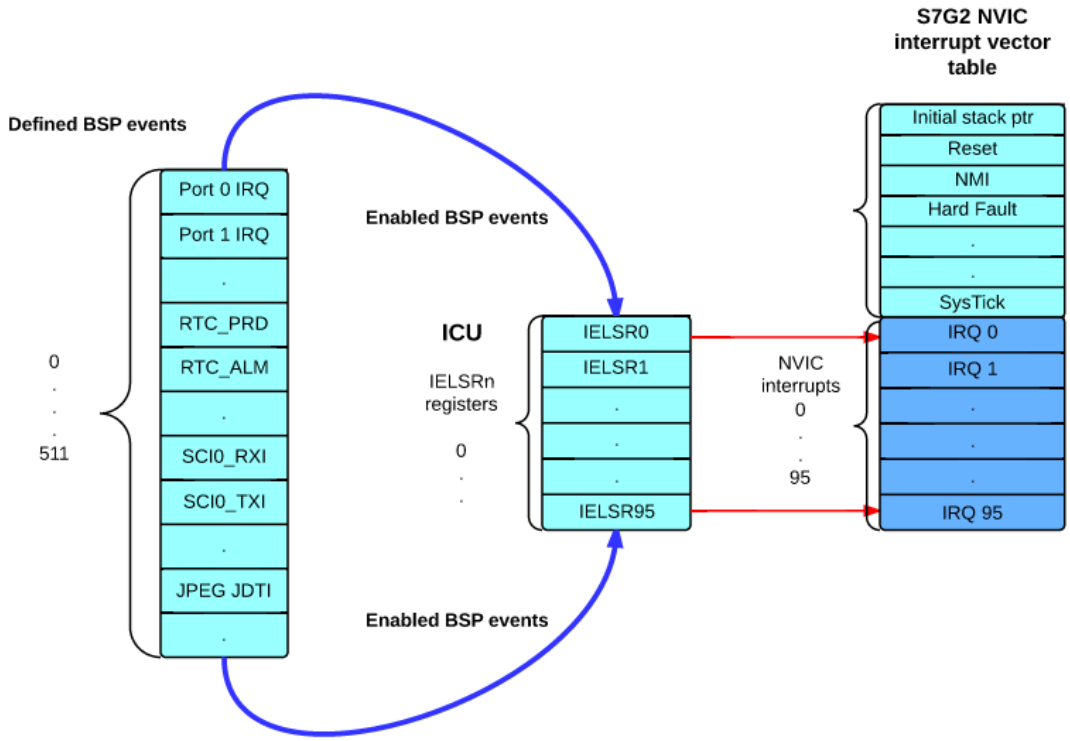


Figure 16: NVIC Interrupt vector table

By allowing the user to select only the events they are interested in as interrupt sources, we are able to provide an interrupt service routine that is fast and event specific.

For example, on other microcontrollers a standard NVIC interrupt vector table might contain a single vector entry for the SCI0 (Serial Communications Interface) peripheral. The interrupt service routine for this would have to check a status register for the 'real' source of the interrupt. In the Synergy implementation there is a vector entry for each of the SCI0 events that we are interested in. The difference between a standard NVIC table and the Synergy S7G2 NVIC table is shown below:

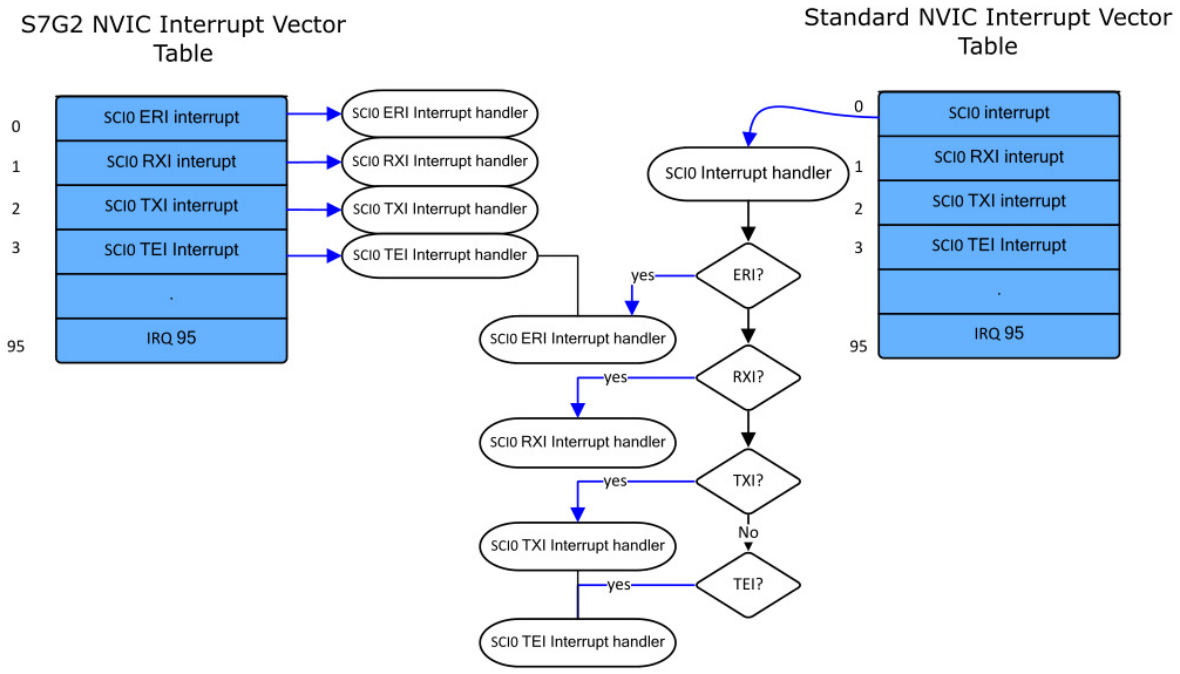


Figure 17: NVIC Interrupt vector table example

Configuration of interrupts is done through the bsp_irq_cfg.h file. For every event given a 'non disabled' priority in the ISDE ICU tab, the BSP generates a corresponding NVIC vector table entry. The BSP iterates over the entire list of possible events, and NVIC slots are assigned in that order with the corresponding event select register updated likewise.

When an interrupt occurs one of the very first operations must be to call R_BSP_IrqStatusClear with the interrupt number corresponding to the NVIC interrupt slot that was assigned by the BSP. The interrupt number is created by the BSP at build time based on the interrupt events that have been enabled in bsp_irq_cfg.h.

R_BSP_IrqStatusClear clears the interrupt status flag (IR) for the given interrupt. When an interrupt is triggered the IR bit is set. If it is not cleared in the ISR then the interrupt will trigger again immediately.

Entries that have been assigned a priority (ie. BSP_IRQ_CFG_ICU_IRQ0) in our example, have their corresponding 'weak handler' address placed in the next available vector slot. All of the possible interrupt sources are iterated over in this manner. Defined interrupts are entered into the vector table.

Bsp_irq_cfg.h

This configuration file contains an entry for every peripheral event that is capable of generating an interrupt. When a peripheral event is given a value other than 'BSP_IRQ_DISABLED' by modifying an peripheral event defined on the ISDE ICU tab, the corresponding entry is updated in bsp_irg_cfg.h. The BSP then iterates through these entries, and peripheral events that have a priority defined (a value other than 'BSP_IRQ_DISABLED') have interrupt service routine entries added to the NVIC Vector table. In addition, the BSP then updates the respective IELSRn register for that NVIC vector number such that it corresponds to the respective peripheral event.

BSP Weak symbols

You might wonder how the BSP is able to place ISR addresses in the NVIC table without the user having explicitly defined one. All that is required by the BSP is that the interrupt event be given a priority.

This is accomplished through the use of the 'weak' attribute. The weak attribute causes the declaration to be emitted as a weak symbol rather than a global. The BSP actually defines an ISR for each of the 512 possible interrupt events in the CMSIS startup_xxxx.c file. A weak symbol is one that can be overridden by an accompanying strong reference with the same name. When the BSP declares a function as weak, user code can define the same function and it will be used in place of the BSP function. By defining all possible interrupt sources as weak, the vector table can be built at compile time and any user declarations (strong references) will be used at runtime.

Weak symbols are supported for ELF targets and also for a.out targets when using the GNU assembler and linker.

Note that in CMSIS system_xxxx.c, there is also a weak definition (and a function body) for the Warm Start callback function R_BSP_WarmStart(). Because this function is defined in the same file as the weak declaration, it will be called as the 'default' implementation. The function may be overridden by the user by copying the body into their user application and modifying it as necessary. The linker identifies this as the 'strong' reference and uses it.

Warm start callbacks

As the BSP is in the process of bringing up the board out of reset, there are two points where the user can request a callback. These are defined as the 'Pre C' and 'Post C' warm start callbacks.

As described above, this function is already weakly defined as R_BSP_WarmStart(), so it is a simple matter of redefining the function or copying the existing body from CMSIS system_xxxx.c into the application code to get a callback. R_BSP_Warmstart() takes an event parameter which describes the type of warm start callback being made.

```
/** Different warm start entry locations in the BSP. */

typedef enum e_bsp_warm_start_event

{

    BSP_WARM_START_PRE_C = 0, // Called almost immediately after reset.

    /* No C runtime environment, clocks, or IRQs. */

    BSP_WARM_START_POST_C // Called after clocks and C runtime environment have
been set up.

} bsp_warm_start_event_t;
```

This function is not enabled/disabled and is always called for both events as part of the BSP startup. Therefore it needs a function body, which will not be called if the user is overriding it. The function body is located in system_xxxx. To use this function just copy this function into your own code and modify it to meet your needs.

Pre C Warm start callback

This callback occurs almost immediately after reset and at this point no C runtime environment, clocks, or IRQs have been setup.

Why would you be interested in a 'Pre C' warm start callback?

Below are a few examples.

- Execution of safety code (ie. destructive memory tests) as part of the startup process.
- Examination of global memory as part of a crash dump investigation.
- Preventing re-initialization of an already running RTC.

Post C Warm start callback

This callback occurs after clocks and the C runtime environment have been setup.

Why would you be interested in a 'Post C' warm start callback?

Below are a few examples.

- Run tests that require that clocks have been setup.
- ADC diagnostics.
- ROM/External memory system checks.

2.2.1.12 Additional BSP Board support

The following Synergy boards are supported:

- DK-S7G2
- PE-HMI1
- DK-S3A7
- DK-S124
- SK-S7G2

2.2.1.13 Custom BSP Board support

What if you're developing your own board based on one of the Synergy MCU's?

There is an external command line utility, the Custom BSP creator tool which can be used to generate a custom BSP that can be used from within e^2 studio. The application note R01AN3044EU0101 describes this process in detail.

2.2.1.14 BSP API functions

The BSP provides public functions, available to any project using the BSP, that allow access to functionality that is common across BSP supported MCU's and boards.

- R_BSP_SoftwareLockInit: The BSP provides API functions to implement atomic locking. These locks can be used to protect critical areas of code as an RTOS semaphore or mutex normally would. This function simply initializes a defined lock structure to BSP_LOCK_UNLOCKED.
- R_BSP_SoftwareLock: Attempt to acquire the lock that has been sent in. The Load-Exclusive and Store-Exclusive instructions are being used to perform an exclusive read-modify-write on the input lock. This process is:
  - Use a load-exclusive (LDREXB) to read the value of the lock.
  - If the lock is available, then modify the lock value so it is reserved. If not available, then issue CLREX.
  - Use a store-exclusive to attempt to write the new value back to memory
  - Test the returned status bit to see if the write was performed or not.
- R_BSP_SoftwareUnlock: Release hold on an existing software lock.
- R_BSP_HardwareLock: Hardware locks are similar to Software locks. In fact the BSP Software lock functions are called by the Hardware lock functions. Hardware locks are specific to a particular peripheral, the list of available

hardware locks being defined in bsp_hw_locks.h. Hardware locks can be used to prevent multiple threads from trying to use a peripheral that is already in use by a process or thread. For example when the Flash API open() function is called it takes the Flash Hardware lock and keeps it until the Flash API close is called.

- R_BSP_HardwareUnlock: Release the hold on an existing hardware lock. In the Flash example above the Flash API close function would call this function.
- R_BSP_GroupIrqWrite: Registers a callback function for one of supported group interrupts. As described earlier, there are 12 of these and they are all mapped to the NMI exception. When an NMI occurs, the NMI_Handler looks at the NMISR (status register) to determine the source of the interrupt. If a callback function has been registered for this group interrupt it will be called. If NULL is passed for the callback argument then any previously registered callbacks are unregistered.
- R_BSP_IrqStatusClear: Clears the interrupt status flag (IR) for a given interrupt. When an interrupt is triggered the IR bit is set. If it is not cleared in the ISR then the interrupt will trigger again immediately.
- R_BSP_SoftwareDelay: Implements a blocking software delay. A delay can be specified in microseconds, milliseconds or seconds. The delay is implemented based on the system clock rate.
- R_BSP_VersionGet: Returns the version of the BSP.
- R_BSP_RegisterProtectEnable: Enables register protection. Registers that are protected cannot be written to. Register protection is enabled by using the Protect Register (PRCR) and the MPC's Write-Protect Register (PWPR). The registers that may be protected are grouped together into one of three groups.
  - BSP_REG_PROTECT_CGC - registers related to the clock generation circuit.
  - BSP_REG_PROTECT_OM_LPC_BATT - registers related to operating modes, low power consumption, and battery backup function.
  - BSP_REG_PROTECT_LVD – registers related to LVD (Low Voltage Detection)

The BSP register protection functions utilize reference counters to ensure that an application which has specified a certain register and subsequently calls another function doesn't have its register protection settings inadvertently modified.

Each time RegisterProtectDisable() is called, the respective reference counter is incremented.

Each time RegisterProtectEnable() is called, the respective reference counter is decremented.

Both functions will only modify the protection state if their reference counter is zero.

As the example below shows, without reference counters MODULE2 would re-protect the registers that MODULE 1 had un-protected, preventing MODULE1 from writing them.

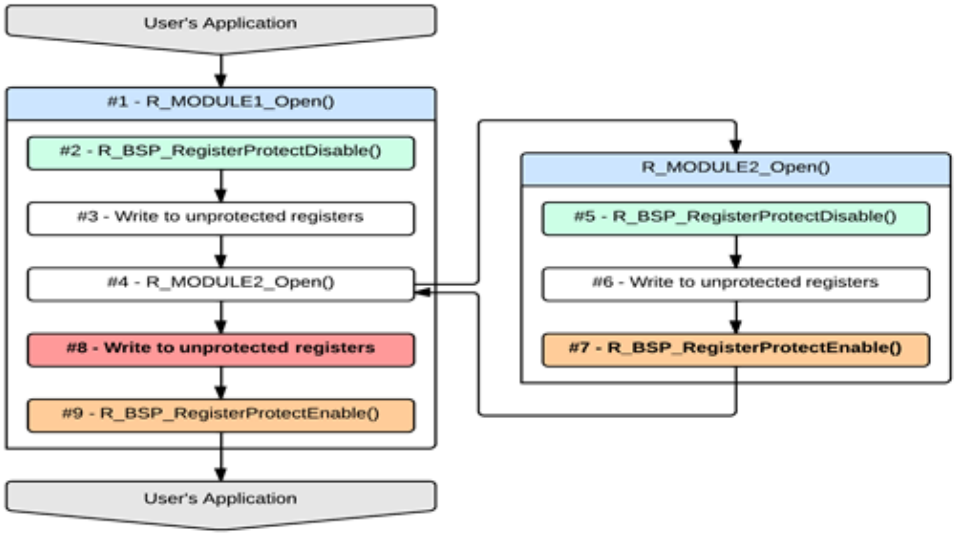


Figure 18: Register protection

- R_BSP_RegisterProtectDisable: Disables register protection. Registers that are not protected can be written to. Register protection is disabled by using the Protect Register (PRCR) and the MPC's Write-Protect Register (PWPR). The register groupings described above still apply.

Chapter 3 Starting Development

To start development with the Renesas Synergy Software Package (SSP), download and install e2 studio, obtain a target Synergy development or evaluation board, and run through the tutorials in this chapter. The e2 studio ISDE user guide and the tutorials include step-by-step instructions for getting started with a simple application. To get started with the SSP, refer to these pages:

- e2 studio ISDE User Guide
- Tutorial: Your First Synergy Project - Blinky
- Tutorial: Using HAL Drivers - Programming the WDT
- IAR Embedded Workbench for Renesas
- What is Synergy Standalone Configurator (SSC)?

3.1 e2 studio ISDE User Guide

This section describes how to use the Renesas e2 studio Integrated Solutions Development Environment (ISDE) to develop applications with the Renesas Synergy Software Package (SSP). The architecture of the SSP directly determines how you use the e2 studio ISDE to develop a Synergy application. See the following documents for details on the SSP architecture included in this manual:

- SSP Architecture
- BSP Architecture

For simple example projects generated and built with e2 studio, see:

- Tutorial: Using HAL Drivers - Programming the WDT
- Tutorial: Your First Synergy Project - Blinky

All User Guides in this manual show how to configure a driver and develop an application using the e2 studio ISDE. See:

- HAL Layer for HAL layer user guides
- Framework Layer for Framework layer user guides

3.1.1 What is the e2 studio ISDE?

The Renesas e2 studio ISDE, or Integrated Solution Development Environment, is a development tool encompassing code development, build, and debug. The ISDE is based on the open-source Eclipse IDE and the associated C/C++ Development Tooling (CDT). Specifically for Synergy MCUs, the ISDE provides a Graphical User Interface (GUI) with numerous wizards for configuring and auto-generating code using the Synergy Software Package (SSP). The ISDE also incorporates a smart manual so that driver and device documentation is available in the form of tooltips right in the code.

Figure 19: e² studio Splash Screen

The e² studio ISDE and the Synergy Project Configurator have been developed to make it as easy as possible to quickly select the SSP modules required for a particular application, include them in a project, and configure them. The ISDE provides a graphical user interface to configure all elements of the SSP for the Synergy MCU applications. In addition to HAL and Framework modules, the ISDE can add and configure RTOS threads, semaphores, mutexes, event flags, and queues. This makes adding RTOS support to an application very straightforward. Once a project has been generated, you can go back and reconfigure any of the modules and settings if required. The ISDE generates the complete and correct configuration code from the selections in the configuration views, so you can focus on writing the application code.

The elements of the SSP are shown in the **Project Explorer** view of the e² studio ISDE. All SSP configuration structures and parameters are mapped to XML files. The XML files enable the ISDE to present a visual list of configurable options that you can select from. In addition to generating code to configure the modules, the XML also provides dependency information for modules.

When you add an SSP module to your project, the e² studio ISDE checks the dependencies of this module and adds all necessary drivers and framework modules to create the appropriate stack. If there is a dependency that requires you to make a choice, this module is highlighted in the Stack window and the ISDE guides your selection by showing the available options.

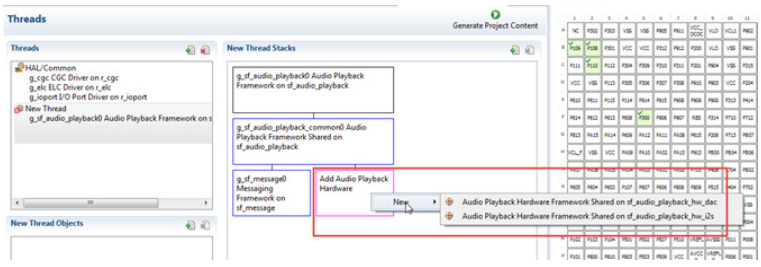


Figure 20: ISDE dependency checking

Errors are flagged next to the Driver name in the HAL/Common Modules or New Thread Modules pane. You can also review errors in the Problems window.

3.1.2 e2 studio ISDE Prerequisites

3.1.2.1 Obtaining a Synergy Kit

To develop applications with the SSP, start with one of the Renesas Synergy Kits. The Renesas Synergy Kits are designed to seamlessly integrate with the e2 studio ISDE. Several types of kits are available:

- Development Kit (DK)
- Starter Kit (SK)
- Product Example (PE)

Ordering information, Quick Start Guides, User Manuals, and other related documents for all Synergy Kits are available at http://renesassynergy.com.

3.1.2.2 PC Requirements

To use the e2 studio ISDE, ensure that your PC meets the following minimum requirements:

- Windows 7 with Intel i5 or i7, or AMD A10-7850K or FX
- Memory: 8 GB DDR3 or DDR4 DRAM (16 GB DDR4/2400 MHz RAM is preferred)
- Minimum 250 GB hard disk

3.1.2.3 Installing e2 studio and the SSP

Detailed installation instructions and installers for the e2 studio ISDE and the SSP are available on the Renesas Synergy Gallery website https://synergygallery.renesas.com. Review the release notes for e2 studio to ensure that the e2 studio version supports the selected SSP version.

See also: Handling SSP Releases

3.1.2.4 Choosing a Toolchain

The e2 studio ISDE can work with several toolchains and toolchain versions such as the GNU ARM compiler and the IAR toolchain. A version of the GNU ARM compiler is included in the e2 studio installer and has been verified to run with the SSP version.

To use the IAR toolchain for ARM, install IAR Embedded Workbench for Renesas Synergy (EWSYN) v7.40.5 (a license from IAR is required). Before starting a Synergy project with IAR, also install the IAR Embedded Workbench for ARM Eclipse plugin (using the IAR Embedded Workbench plugin manager in the 'Help' menu).

3.1.2.5 Licensing

By default, the SSP download includes an evaluation license for the SSP. When prompted for the license file during the Synergy project configuration, refer to the installation instructions for details on licensing. The evaluation license file is located in the following directory: < e2_studio_base_dir >/internal/projectgen/arm/Licenses/.

The evaluation license allows the use of a fully functional version of the SSP. This means that all modules of the SSP can be compiled and linked into your application. However, the source code for modules in the Framework Layer (including Express Logic X-Ware components) is encrypted and cannot be altered. The source code of selected modules is still viewable. This type of encrypted code is called protected code. For details, refer to the license file.

Source code for modules in the HAL Layer is not protected and can be viewed in the e2 studio ISDE by clicking on the respective synergy/ssp/src/< module >/< module>.c file.

Other license types are available which allow you to view protected code in the ISDE during the debug phase with the Secure Debug View or to have full read and write access to selected sources.

The table below shows the available licenses and the functionality they provide.

License Type	Allowed	Not Allowed
Evaluation License	Compiling/Linking	Saving/Editing of protected sources. Not all modules are viewable. For details see the license file.
Development/Production License	Compiling/Linking and viewing of protected sources with the Secure Viewer	Saving/Editing of protected sources
Source License	Compiling/Linking and viewing of protected sources with the Secure Viewer. Selected sources (depending on the source license) can be edited and saved as clear text files.	Saving/editing of protected source files that are not included in the Source License

3.1.2.6 Adding the IAR Embedded Workbench for Renesas Synergy Compiler into e2 studio

The IAR Embedded Workbench for Renesas Synergy compiler (IAR compiler) can now be used from within 2 studio. This allows the developer to have the advantages provided by the IAR compiler without the need to also use the IAR EW for Renesas Synergy ISDE. The installation process involves installing IAR EW for Renesas Synergy, installing e2 studio, and installing the associated IAR plugins for e2 studio. The process is described in the application note found with this search: https://www.renesas.com/en-us/search/keyword-search.html#q=R11AN0272

The application note also includes a description of how to migrate a project that originally used e2 studio and GCC. It also includes a description of how to migrate a project from IAR 7.x to IAR 8.x so it will be successfully opened when using e2 studio 6.2.0.

3.1.3 What is a Project?

In e2 studio, all SSP applications are organized in Synergy projects. Setting up a Synergy project involves:

1) Creating the project
2) Configuring the project

The e2 studio ISDE has many project wizards and configuration windows specifically for Synergy projects. You can create a new Synergy Project with the **Synergy Project Generator** or edit the configuration of an existing project in the **Synergy Project Editor**.

When you launch e2 studio and select a workspace, all projects previously saved in the selected workspace are loaded and displayed in the **Project Explorer** view. Each project has an associated configuration file named configuration.xml which is located in the project's root directory.

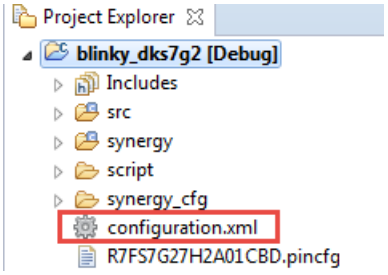


Figure 21: e2 studio Project Configuration file

Double-click on the configuration.xml file to open the Synergy Project Editor and view or modify all configuration settings associated with this project. To edit the project configuration, make sure that the **Synergy Configuration** perspective is selected in the upper right hand corner of the e2 studio window.

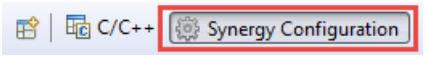


Figure 22: e2 studio Synergy Configuration Perspective

The Synergy Project Editor has a number of tabs. The configuration steps and options for individual tabs are discussed in the following sections.

NOTE: Which tabs are available with the Synergy Project Editor depends on the e2 studio version. Version 5.0 of e2 studio supports an additional tab for the Messaging Configurator.

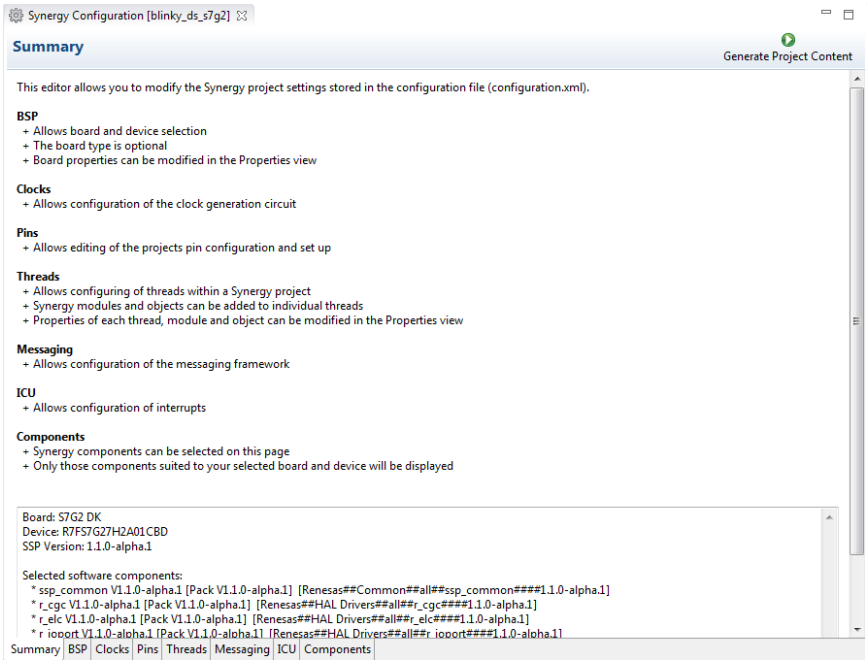


Figure 23: e² studio Project Editor

3.1.4 Creating a Project

This section includes step-by-step instructions for creating a Synergy Project. Once you have created the project, you can easily configure the hardware (clocks, pins, interrupts) and the parameters of all modules that are part of your application.

To create a new Synergy Project with the Synergy Project Generator, select the project name, select the hardware for your application, review or add the license, select the toolchain and choose from preconfigured clock, pin, and MCU related settings by selecting a project template.

3.1.4.1 Creating a New Project

For Synergy applications, always generate a new project as a Synergy Project in the following way:

1) Click on **File > New > Synergy Project**

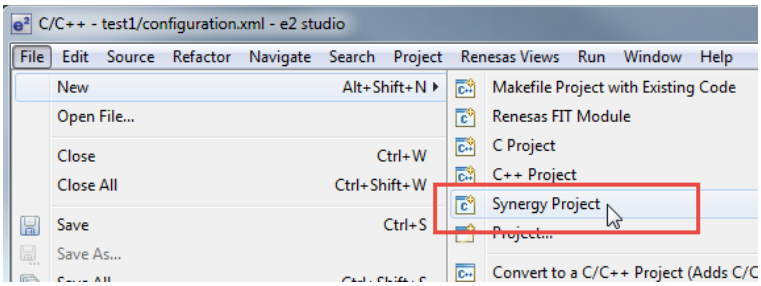


Figure 24: New Synergy Project

2) Select a project name and location.

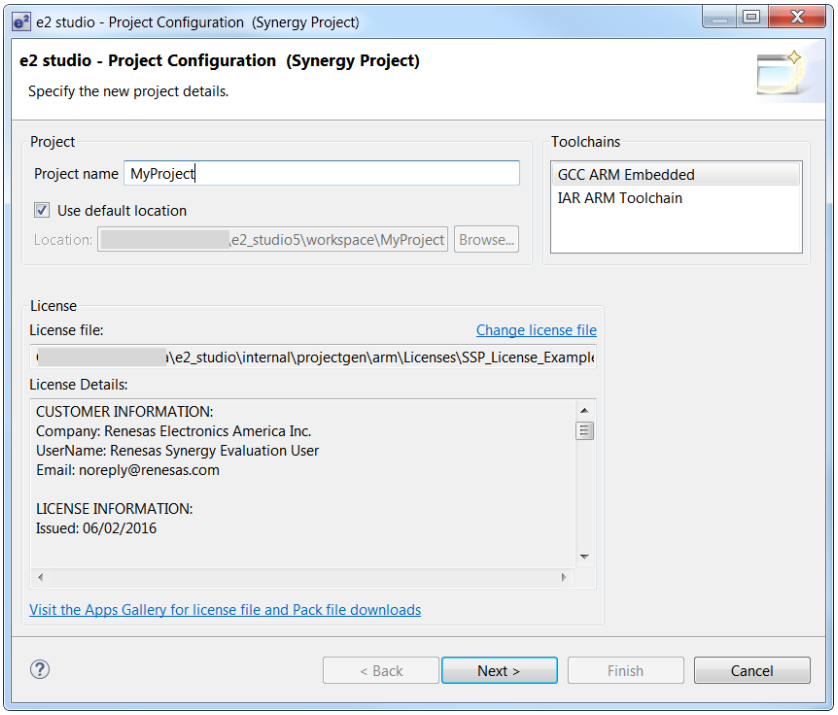


Figure 25: Synergy Project Generator (Screen 1)

3) Ensure that you have the license file for your latest SSP download. An evaluation license is included in the SSP download pack, but other license types such as a Production License and a Source License are available.See the **Installation_Guide** on the Synergy website for instructions how to load the license file. The license pane is blank if you have installed the e2 studio and SSP for the first time. Otherwise, the license pane displays the most recently installed license. You can select from multiple licenses if available. The evaluation license file is located in the following directory: < e2_studio_base_dir >/internal/projectgen/arm/Licenses/See also Licensing.

4) Click **Next**.

3.1.4.2 Selecting a Board and Toolchain

In the next Project Configuration window select the hardware and software environment:

1) Select the **SSP version**.

2) Select the **Board** for your application. You can select an existing Synergy Kit or select Custom User Board for any of the Synergy devices with your own BSP definition.

NOTE: To develop your own BSP, see the following Application Note: "Creating a Custom Board Support Package" at http://renesassynergy.com.

3) Select the **Toolchain version**.
4) Select the **Debugger**. The J-Link ARM Debugger is preselected.
5) Click **Next**.

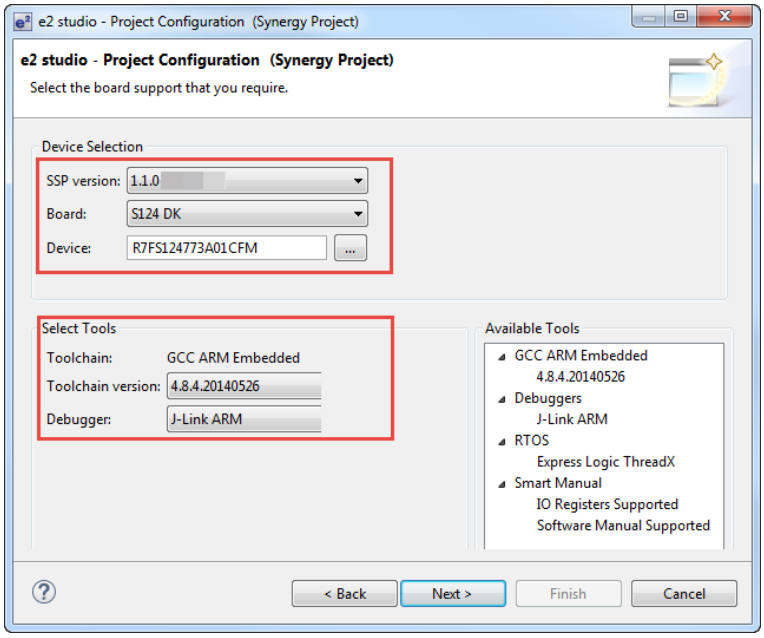


Figure 26: Synergy Project Generator (Screen 2)

3.1.4.3 Selecting a Project Template

In the next window, select a project template from the list of available templates and click **Finish**.

NOTE: If you want to develop your own application, select a basic template for your board, such as S7G2-DK BSP. You can add RTOS support at any time while you configure the modules for your project.

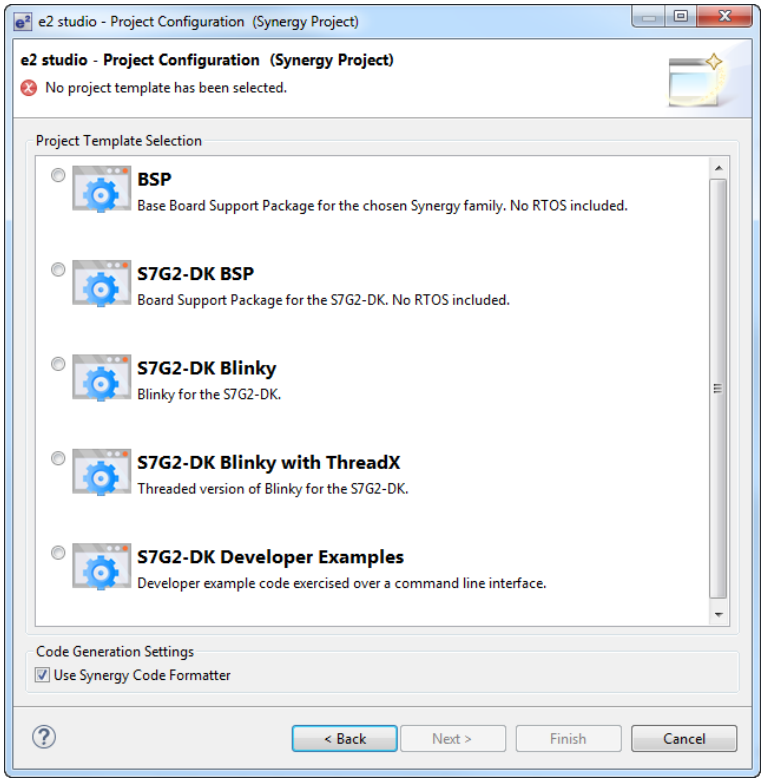


Figure 27: Synergy Project Generator (Screen 3)

By default, this screen shows the templates that are included in your current SSP pack.

When the project is created, the ISDE displays a summary of the current project configuration in the Synergy Project Editor.

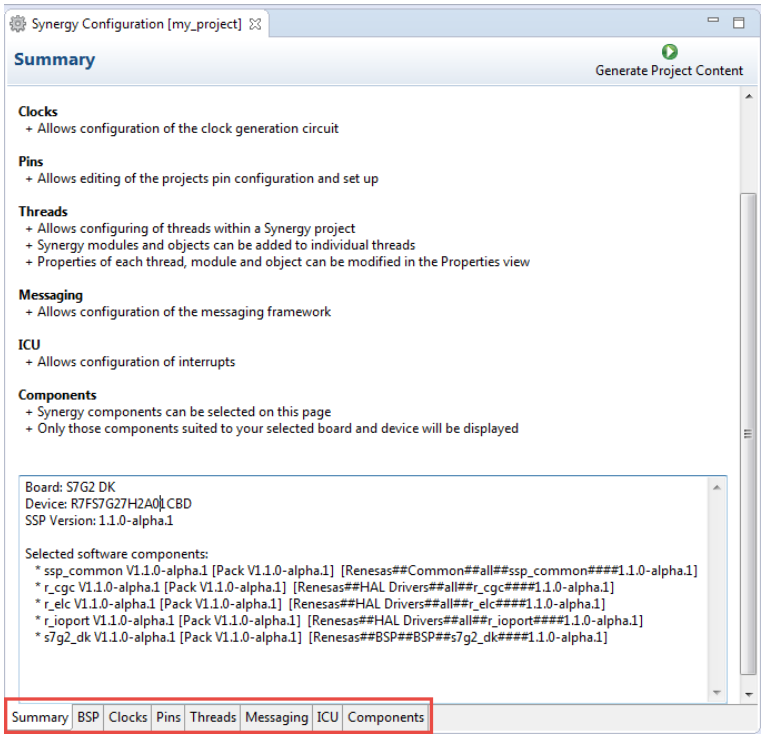


Figure 28: Synergy Project Editor and available editor tabs

On the bottom of the Synergy Project Editor view, you can find the tabs for configuring multiple aspects of your project:

- With the BSP tab, you can change board specific parameters from the initial project selection.
- With the Clocks tab, you can configure the MCU clock settings for your project.
- With the Pins tab, you can configure the electrical characteristics and functions of each port pin.
- With the Threads tab, you can add SSP modules and drivers for RTOS and non-RTOS applications and configure the drivers. For each module or driver selected in this tab, the Properties window provides access to the configuration parameters, interrupt priorities, and pin selections.
- With the Messaging tab, you can configure the Messaging Framework for ThreadX-based projects. The Messaging tab is included in e2 studio version 5.0 and higher.
- With the ICU tab, you can enable interrupts and assign interrupt priorities.
- The Components tab provides an overview of the selected modules. You can also add drivers for specific SSP releases and application sample code here.

3.1.5 Configuring a Project

A project has two levels of configuration.

- The BSP, Clocks, and Pins tabs determine the initial configuration of the MCU after reset and before any user code is executed. By selecting a project template during project creation, the ISDE configures default values as appropriate for the selected board. You can change those default values as needed.
- The Threads allows you to add SSP modules to the project and set the configuration parameters of the module as needed by the application. Because the Messaging Framework is an integral part of many ThreadX-based applications, you can configure the Messaging Framework for each thread requiring messaging in the Messaging tab (for e2 studio versions 5.0 and higher).

3.1.5.1 Configuring the BSP with the ISDE

The **BSP** tab shows the currently selected board (if any) and device. The **Properties** view is located in the lower left of the Project Configurations view as shown below.

NOTE: If the Properties view is not visible, click **Window > Show View > Properties** in the top menu bar.

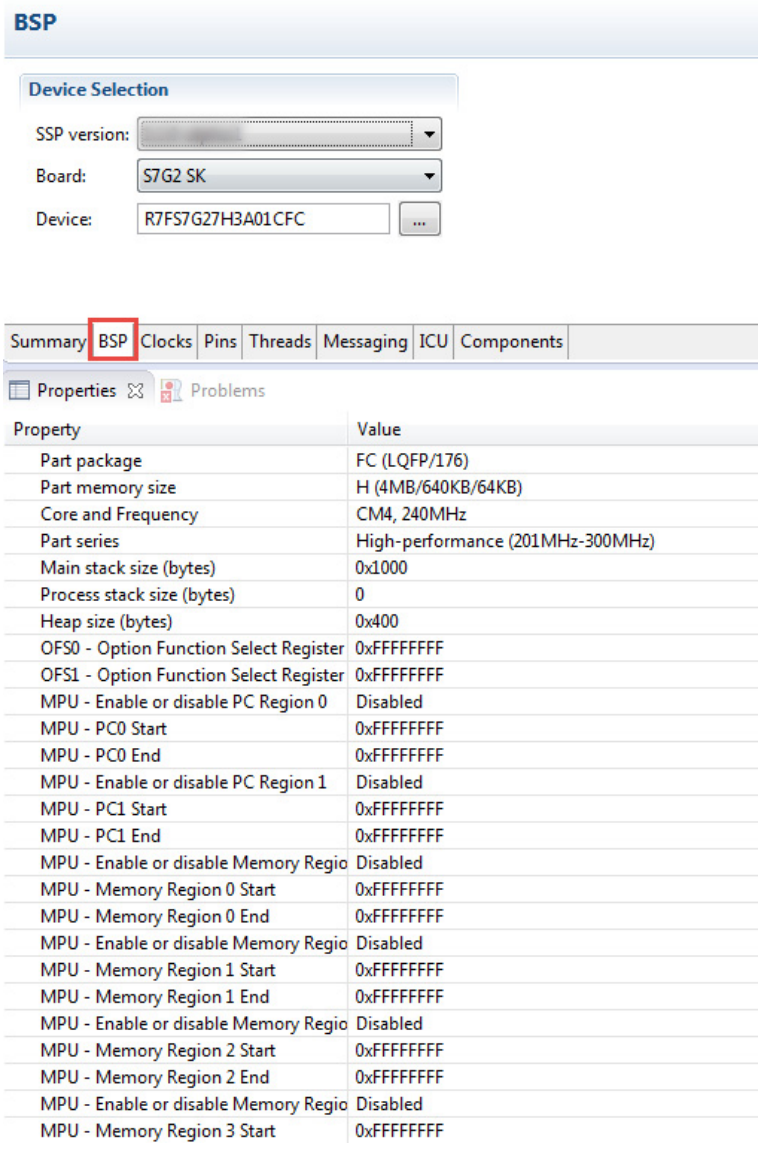


Figure 29: ISDE BSP tab

The Properties view shows the configurable options available for the BSP. These can be changed as required. The BSP is the SSP layer above the MCU hardware. The ISDE checks the entry fields to flag invalid entries. For example, only valid numeric values can be entered for the stack size.

When you press the **Generate Project Content button**, the BSP configuration contents are written to

synergy_cfg/ssp_cfg/bsp/bsp_cfg.h

This file is created if it does not already exist.

ATTENTION: Do not edit this file as it is overwritten whenever the **Generate Project Content** button is pressed.

3.1.5.2 Configuring Clocks

The **Clocks** tab presents a graphical view of the MCU's clock tree, allowing the various clock dividers and sources to be modified. If a clock setting is invalid, the offending clock value is highlighted in red. It is still possible to generate code with this setting, but correct operation cannot be guaranteed. In the figure below, the USB clock UCLK divider has been changed so the resulting clock frequency is 60 MHz instead of the required 48 MHz. This parameter is colored red.

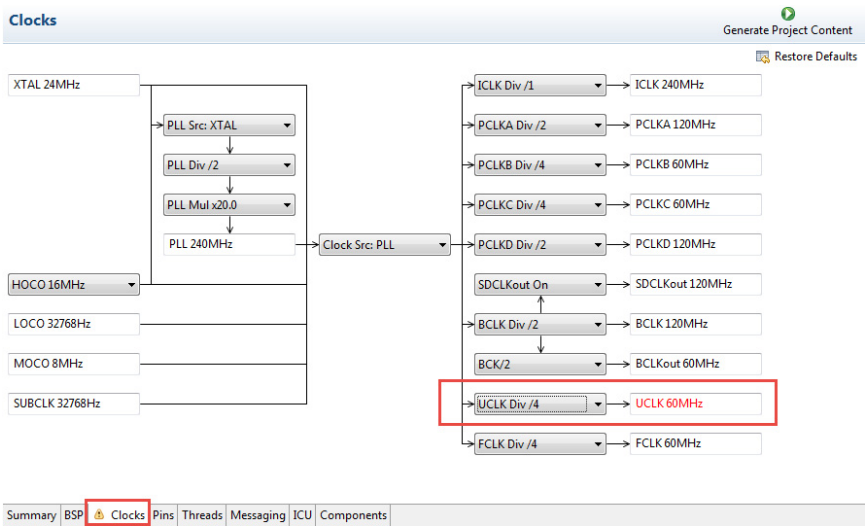


Figure 30: ISDE Clocks tab

When you press the **Generate Project Content** button, the clock configuration contents are written to:

synergy_cfg/ssp_cfg/bsp/bsp_clock_cfg.h

This file will be created if it does not already exist.

ATTENTION: Do not edit this file as it is overwritten whenever the **Generate Project Content** button is pressed.

3.1.5.3 Configuring Pins

The **Pins** tab provides flexible configuration of the MCU's pins. As many pins are able to provide multiple functions, they can be configured on a peripheral basis. For example, selecting a serial channel via the SCI peripheral offers multiple options for the location of the receive and transmit pins for that module and channel. Once a pin is configured, it is shown as green in the **Package** view.

NOTE: If the Package view window is not open in the ISDE, select **Window > Show View > Pin Configurator > Package** from the top menu bar to open it.

The **Pins** tab simplifies the configuration of large packages with highly multiplexed pins by highlighting errors and

presenting the options for each pin or for each peripheral. If you selected a project template for a specific board such as the DK-S7G2, some peripherals connected on the board are preselected.

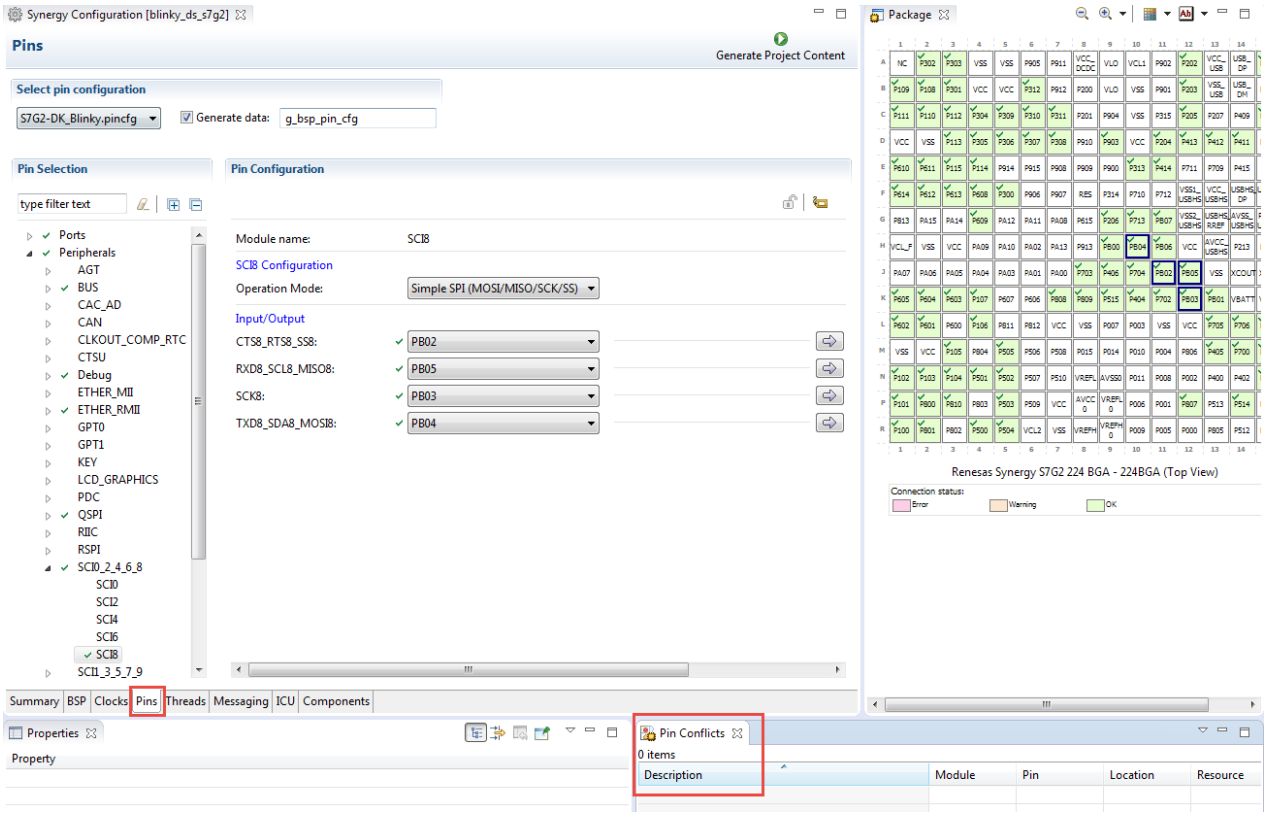


Figure 31: ISDE Pins tab

The pin configurator includes a built-in conflict checker, so if the same pin is allocated to another peripheral or I/O function the pin will be shown as red in the package view and also with white cross in a red square in the **Pin Selection** pane and **Pin Configuration** pane in the main **Pins** tab. The **Pin Conflicts** view provides a list of conflicts, so conflicts can be quickly identified and fixed.

In the example shown below, port P103 is already used by the External Memory peripheral, and the attempt to connect this port to the Serial Communications Interface (SCI) results in a dangling connection error. To fix this error, select another port from the pin drop-down list or disable the External Memory peripheral in the **Pin Selection** pane on the left side of the tab.

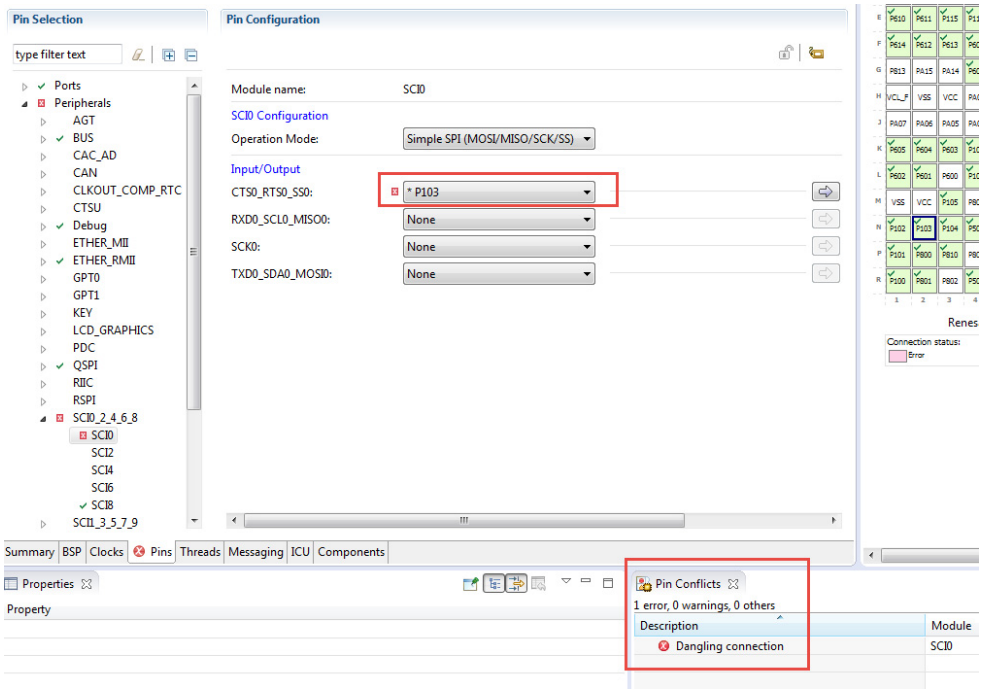


Figure 32: ISDE Pin configurator

The pin configurator also shows a package view and the selected electrical or functional characteristics of each pin.

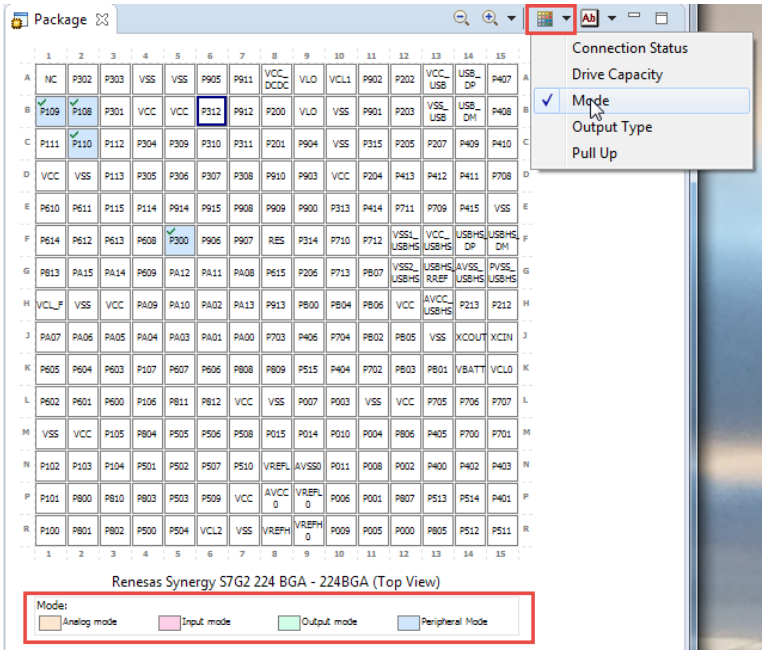


Figure 33: ISDE Pin configurator

When you press the **Generate Project Content** button, the pin configuration contents are written to:

synergy_cfg/ssp_cfg/bsp/bsp_pin_cfg.h

This file will be created if it does not already exist.

ATTENTION: Do not edit this file as it is overwritten whenever the **Generate Project Content** button is pressed.

To make it easy to share pinning information for your project, the ISDE exports your pin configuration settings to a csv format and copies the csv file to synergy_cfg/ssp_cfg/bsp/pincfg_< MCU package>.csv.

3.1.6 Adding Threads and Drivers

Every ThreadX-based Synergy Project includes at least one RTOS Thread and a stack of SSP modules running in that thread. The **Threads** tab is a graphical user interface which helps you to add the right modules to a thread and configure the properties of both the threads and the modules associated with each thread. Once you have configured the thread, the ISDE automatically generates the code reflecting your configuration choices.

For any driver or more generally any module that you add to a thread, the ISDE automatically resolves all dependencies with other modules and creates the appropriate stack. This stack is displayed in the Threads pane, which the ISDE populates with the selected modules and module options for the selected thread. If there is more than one module that can fulfill a dependency requirement, the ISDE prompts you to choose a module from a dropdown menu. For example, when you add the Audio Playback Framework to a thread, you also must pick either the DAC or the I2S framework for playback. The Audio Playback Framework also requires a timer driver, for which you must select either the AGT or GPT driver implementation, and a transfer driver. Whether or not you have a choice of implementations for the transfer driver depends in this example on the selected playback hardware, and the ISDE presents the following:

- If you selected the DAC, you have an option: You can choose between the DMAC and DTC implementations of the transfer driver.
- If you selected the I2S, only the DTC is supported: Two instances of the DTC are required (one for each channel) which are both added by the ISDE.

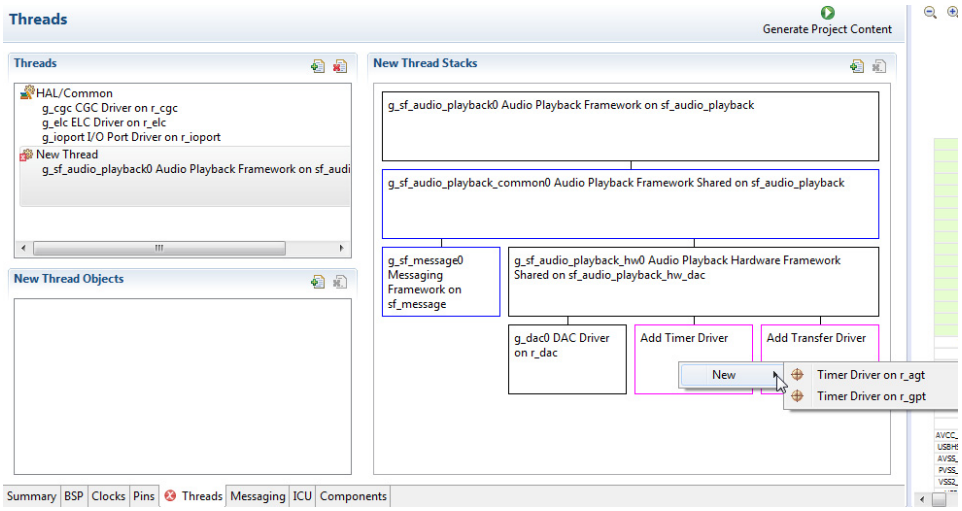


Figure 34: ISDE Project configurator - Overview

The default view of the **Threads** tab includes a Common Thread called **HAL/Common.** This thread includes the drivers for I/O control (IOPORT), clock generation circuit (CGC), and the event link controller (ELC). The default stack is shown

in the **HAL/Common Stacks** pane. The default modules added to the HAL/Common thread are special in that the SSP only requires a single instance of each, which the ISDE then includes in every user-defined thread by default.

In applications that do not use an RTOS or run outside of the RTOS, the HAL/Common thread becomes the default location where you can add additional drivers to your application.

For a detailed description on how to add and configure modules and threads, see the following sections:

- Adding and Configuring HAL Drivers
- Adding Drivers to a Thread and Configuring the Drivers

Once you have added a module either to HAL/Common or to a new thread, you can access the driver's configuration options in the **Properties** view. If you added thread objects, you can access the objects configuration options in the **Properties** view in the same way.

You can find details about how to configure threads here: Configuring Threads

NOTE: Driver and module selections and configuration options are defined in the SSP pack and can therefore change when the SSP version changes.

3.1.6.1 Adding and Configuring HAL Drivers

For applications that run outside or without the RTOS, you can add additional HAL drivers to your application using the HAL/Common thread. To add drivers, follow these steps:

1) Click on the HAL/Common icon in the **Threads** pane. The Modules pane changes to **HAL/Common Stacks**.

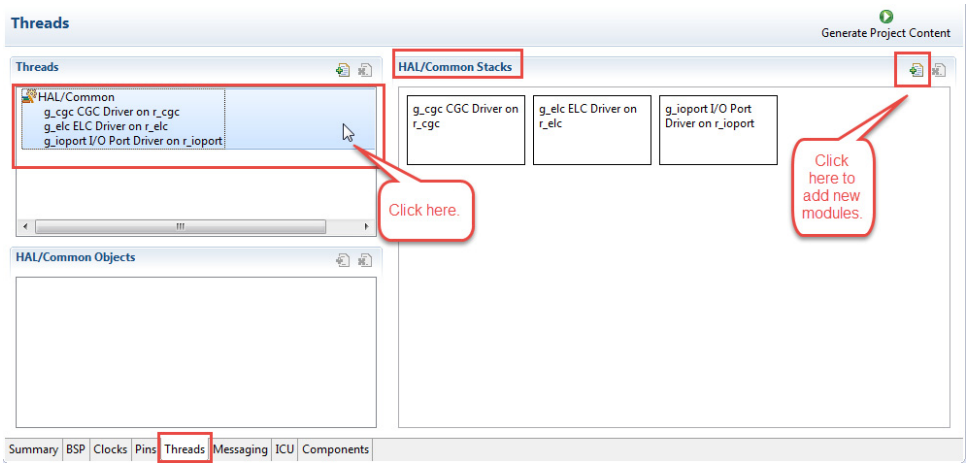


Figure 35: ISDE Project configurator - Adding drivers

2) Click **New Stack** to see a drop-down list of HAL level drivers available in the SSP.

3) Select a driver from the menu **New > Driver**. In addition, you can select the Power Profiles from **New > Framework > Services > Power Profiles Framework** on sf_power_profiles for RTOS independent applications. All other modules can only be added to a thread when ThreadX is present.

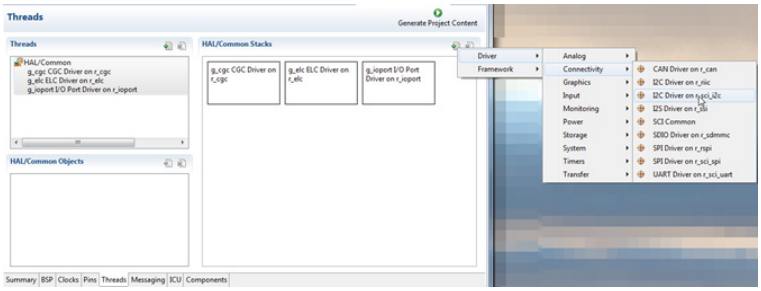


Figure 36: Select a driver

The ISDE creates the stack for the selected driver and alerts you when the driver needs additional resources that must be enabled. In the case below, you can configure the interrupt in the **Properties** view or add missing interrupts using the **ICU** tab later.

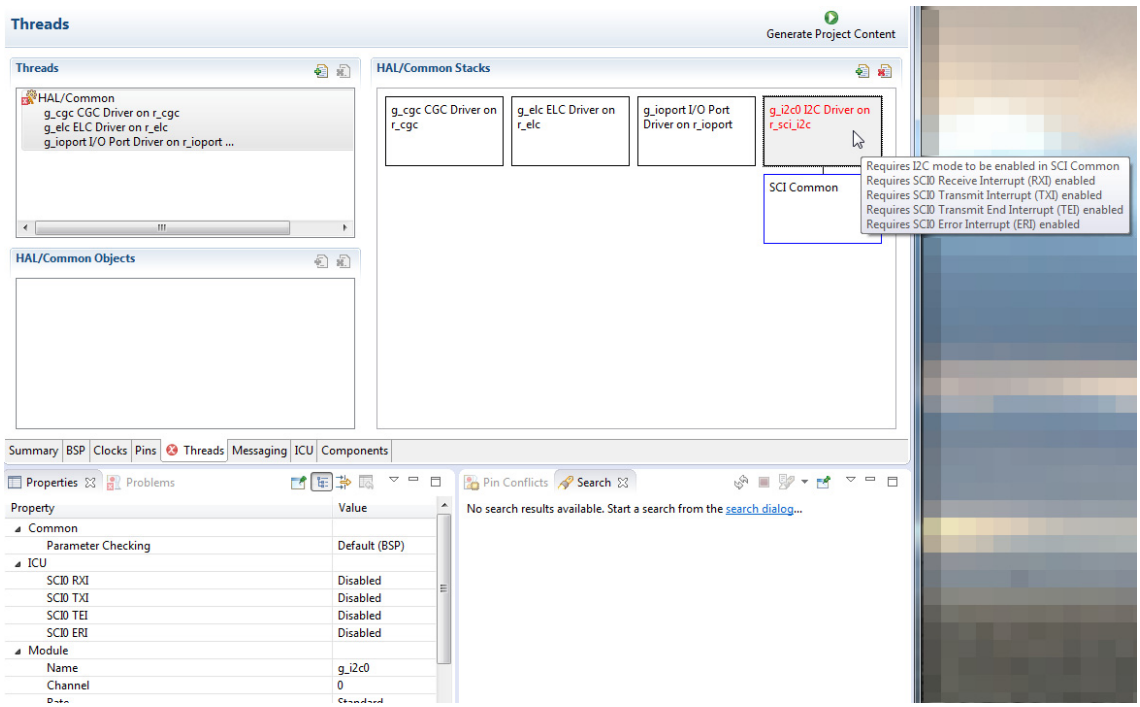


Figure 37: Dependency checking on the Threads tab

4) Select the driver module in the **HAL/Common Modules** pane and configure the driver properties in the **Properties** view.

The ISDE adds the following files when you click the **Generate Project Content** button:

- The selected driver module and its files to the synergy/ssp directory.
- The main() function and configuration structures and header files for your application as shown in the table below.

File	Contents	Overwritten by Generate Project Content?
src/synergy_gen/main.c	Contains main() calling generated and user code. When called, the BSP already has initialized the MCU.	Yes
src/synergy_gen/hal_data.c	Configuration structures for HAL Driver only modules.	Yes
src/synergy_gen/hal_data.h	Header file for HAL driver only modules.	Yes
src/hal_entry.c	User entry point for HAL Driver only code. Add your code here.	No

The configuration header files for all included modules are created or overwritten in this folder:

synergy_cfg/ssp_cfg/driver

3.1.6.2 Adding Drivers to a Thread and Configuring the Drivers

For an application that uses the ThreadX RTOS, you can add one or more threads, and for each thread at least one module that runs in the thread. You can select modules from either the Driver or Framework dropdown menu. To add modules to a thread, follow these steps:

1) In the **Threads** pane, click **New Thread** to add a Thread.

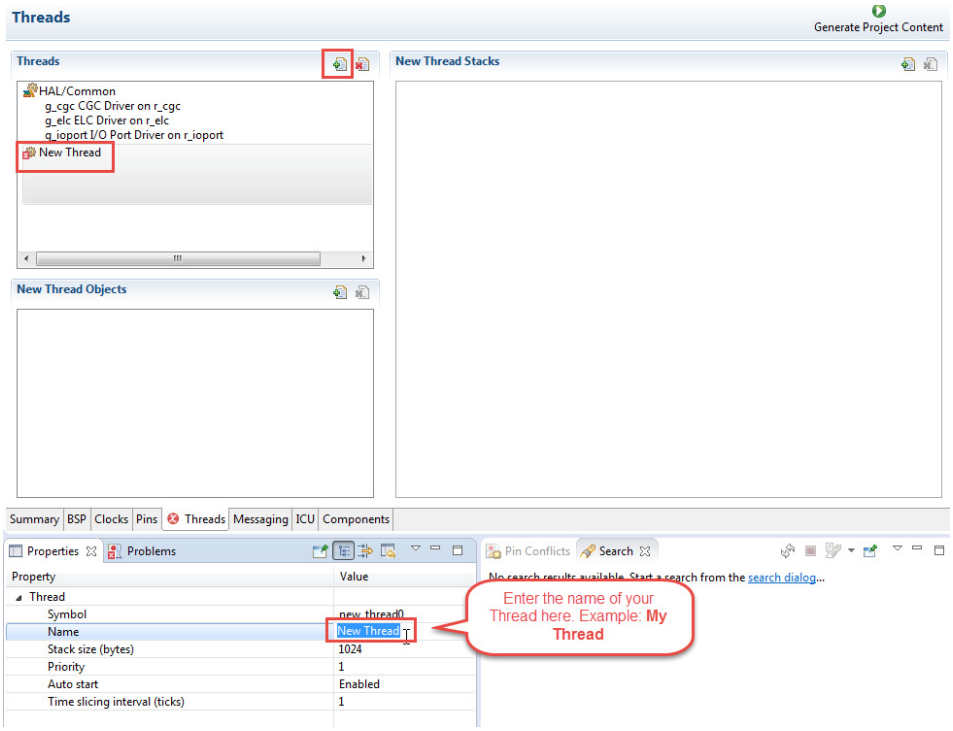


Figure 38: Adding a new RTOS Thread on the Threads tab

2) In the **Properties** view, click on the **Name** and **Symbol** entries and enter a distinctive name and symbol for the new thread.

   NOTE: The ISDE updates the name of the thread stacks pane to **My Thread Stacks**.

3) In the **My Thread Stacks** pane, click on **New** to see a list of modules and drivers. Both Framework-level Modules and HAL-level drivers can be added here.

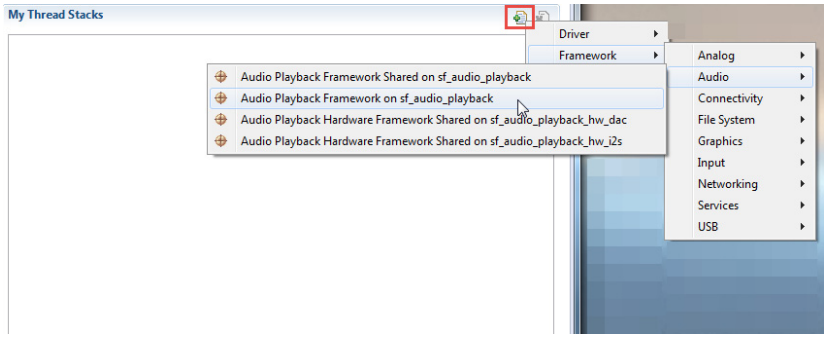


Figure 39: Adding Modules and Drivers to a thread

4) Select a module or driver from the list.

5) If the module or driver indicates a dependency, select the missing resources.

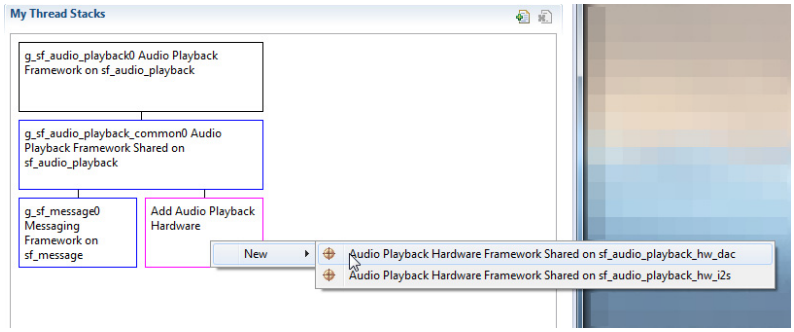


Figure 40: Identifying Module or Driver dependencies on the Threads tab

6) Click on the added driver and configure the driver as required by the application by updating the configuration parameters in the **Properties** view. To see the selected module or driver and be able to edit its properties, make sure the Thread containing the driver is highlighted in the **Threads** pane.

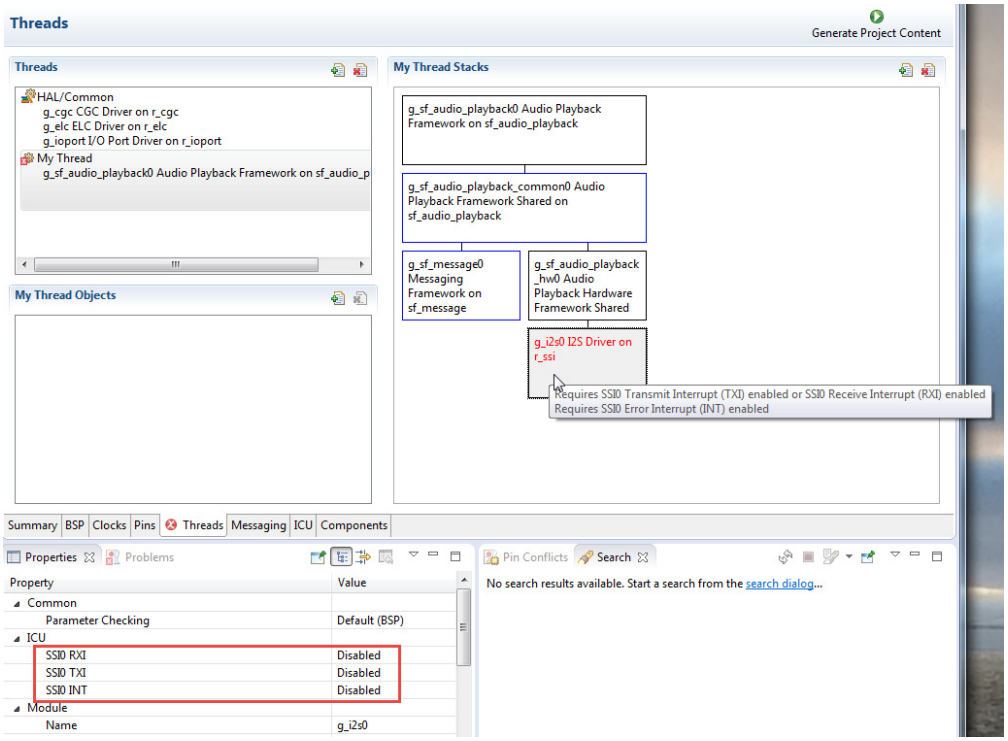


Figure 41: Configuring Module or Driver properties

7) If needed, add another thread by clicking **New** in the **Threads** pane.

When you press the **Generate Project Content** button for the example above, the ISDE creates the files as shown in the following table:

File	Contents	Overwritten by Generate Project Content?
src/synergy_gen/main.c	Contains main() calling generated and user code. When called the BSP will have initialized the MCU.	Yes
src/synergy_gen/my_thread.c	Generated thread "my_thread" and configuration structures for modules added to this thread.	Yes
src/synergy_gen/my_thread.h	Header file for thread "my_thread"	Yes
src/synergy_gen/hal_data.c	Configuration structures for HAL Driver only modules.	Yes

File	Contents	Overwritten by Generate Project Content?
src/synergy_gen/hal_data.h	Header file for HAL Driver only modules.	Yes
src/hal_entry.c	User entry point for HAL Driver only code. Add your code here.	No
src/my_thread_entry.c	User entry point for thread "my_thread". Add your code here.	No

The configuration header files for all included modules and drivers are created or overwritten in the following folders:

synergy_cfg/ssp_cfg/driver

synergy_cfg/ssp_cfg/framework

3.1.6.3 Configuring Threads

If the application uses the ThreadX RTOS, the **Threads** tab can be used to simplify the creation of ThreadX threads, semaphores, mutexes, and event flags.

The components of each thread can be configured from the **Properties** view as shown below.

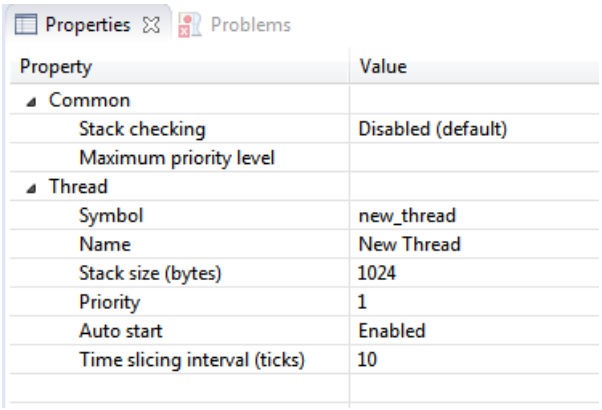


Figure 42: ISDE Thread Properties

The **Properties** view contains settings common for all Threads (**Common**) and settings for this particular thread (**Thread**).

For this thread instance, the thread's name and properties (such as priority level or stack size) can be easily configured. The ISDE checks that the entries in the property field are valid. For example, the ISDE ensures that the field **Priority**, which requires an integer value, only contains numeric values between 0 and 9.

To add ThreadX resources to a Thread, select a thread and click on **New Object** in the Thread Objects pane. The pane takes on the name of the selected thread, in this case **My Thread Objects**.

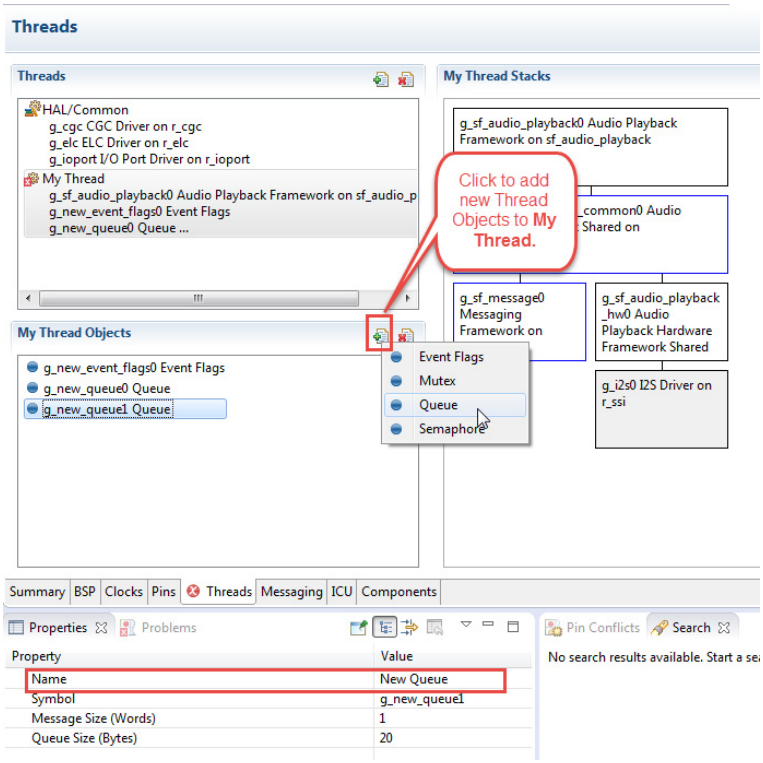


Figure 43: Configuring Thread Object Properties

Make sure to give each thread object a unique name and symbol by updating the **Name** and **Symbol** entries in the **Properties** view.

3.1.6.4 Configuring Interrupts

You can use the **Properties** view in the **Threads** tab or the **ICU** (Interrupt Control Unit) tab to enable interrupts by setting the interrupt priority. Select the thread in the Threads pane to view and edit its properties.

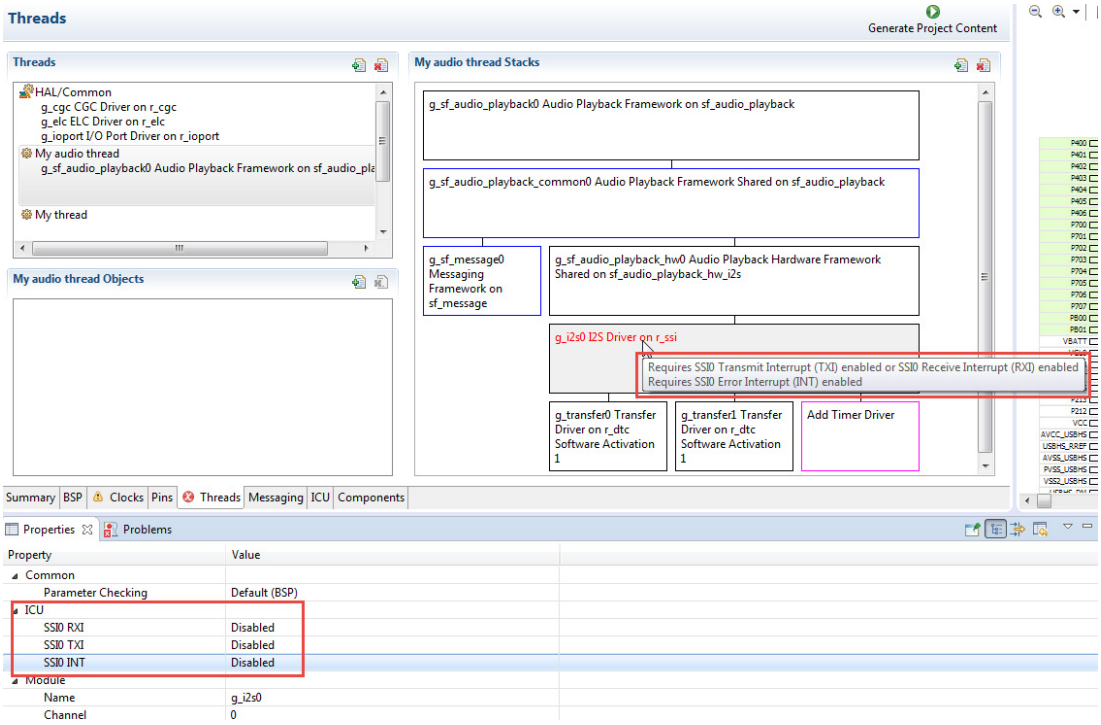


Figure 44: Configuring Interrupt on the Threads tab

3.1.7 Configuring the SSP Messaging Framework

The Messaging Framework extends the ThreadX messaging queue functionality and is one of the most important SSP modules. It provides the mechanism for threads to communicate with each other through exchanging messages. The Messaging Framework allows threads to send (publish) or listen to (pend on) messages when preconfigured or user-configured Events happen. Any thread can publish a message with an attached Event Class that all threads subscribing to this Event class can listen to and act upon. The list of Threads that can listen to a specific Event Class is called the Subscriber List for that Event Class.

To use the Messaging Framework, you first must add one Messaging Framework instance in the **Threads** tab. You may add the Messaging Framework to any thread which is not the HAL/Common thread. All threads in your project can use this instance to communicate with each other. Some modules like the Touch Panel Framework or the Audio Playback Framework require the Messaging Framework and add it automatically to the stack as shown below for the Audio Playback Framework. If your project includes a thread with such a module, you do not need to add another instance of the Messaging Framework even if you add more threads to your application. All threads can share one instance of the Messaging framework.

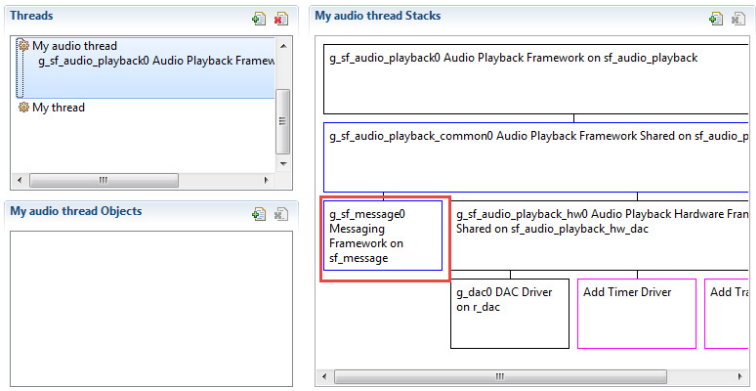


Figure 45: Adding the messaging module to a thread

Once you have added a Messaging Framework instance in the **Threads** tab, you can use the **Messaging** tab to define your own event classes and events and determine which threads can listen to which event class. The SSP contains some predefined event classes and events for the Touch Panel and the Audio Framework modules. If you have added any of these modules, their predefined event classes and events are shown in the Messaging tab as well. The predefined Event Class and Events for the Audio Playback Framework are shown below.

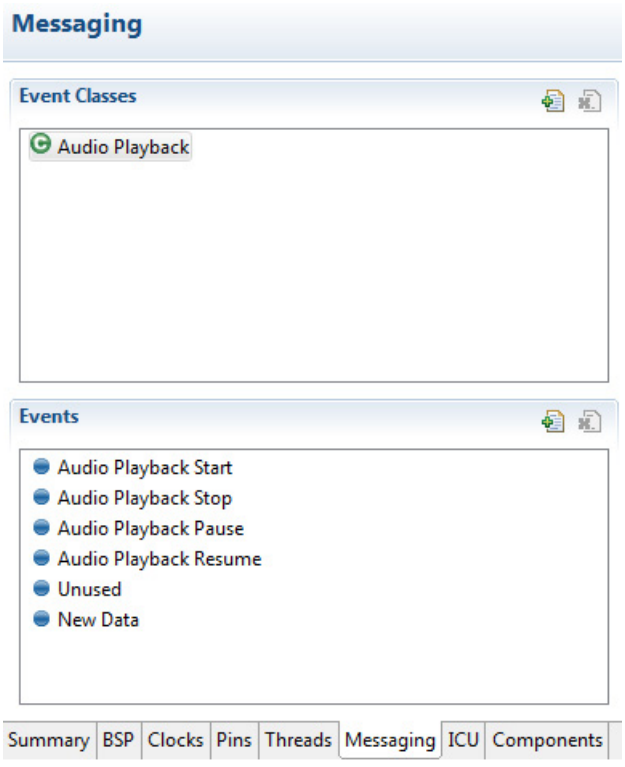


Figure 46: Audio Playback Framework predefined Event Class

3.1.7.1 Adding an Event Class

To add your own user-defined Event Class to the messaging system, follow these steps:

1) In the **Messaging** tab, select the **Event Classes** Pane, and click the add button.
2) Enter a unique name for your event class.
3) Click **OK**.

NOTE: User-defined Event Classes are marked with a golden square on the upper right of the Event Class icon.

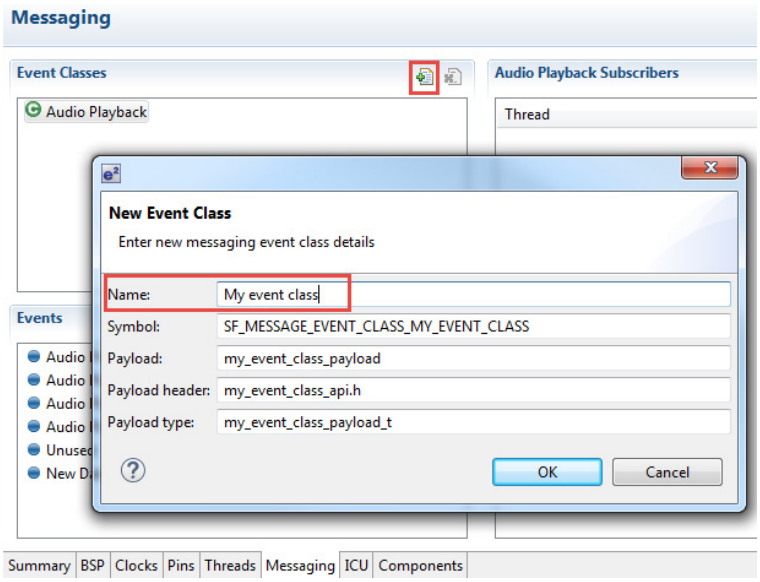


Figure 47: Messaging – Add an Event Class

3.1.7.2 Adding an Event

To add your own user-defined Event to the messaging system, follow these steps:

1) In the **Messaging** tab, select the **Event** Pane, and click the add button.
2) Enter a unique name for your event class.
3) Click **OK**.

NOTE: User-defined Events are marked with a golden square on the upper right of the Events icon.

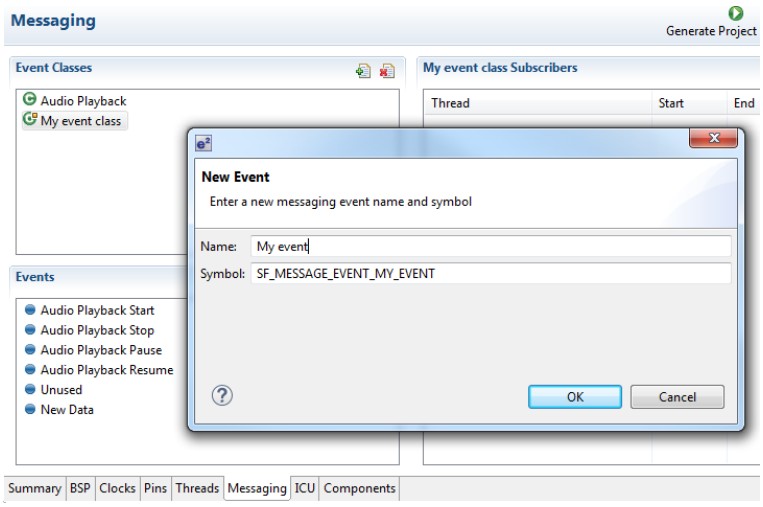


Figure 48: Messaging – Add an Event

3.1.7.3 Configuring the Messaging Subscriber List

In the Subscriber List, you select the threads that are listening to messages from the message publisher. The connection between the publishing thread and the listening thread is established through the Event Class. Therefore you define a subscriber list for each of the Event Classes in your project. All threads in the Subscriber List then can listen and act upon messages belonging to the selected Event Class.

To following assumes that you have two threads defined in the **Threads** tab, one of which uses the Audio Playback Framework:

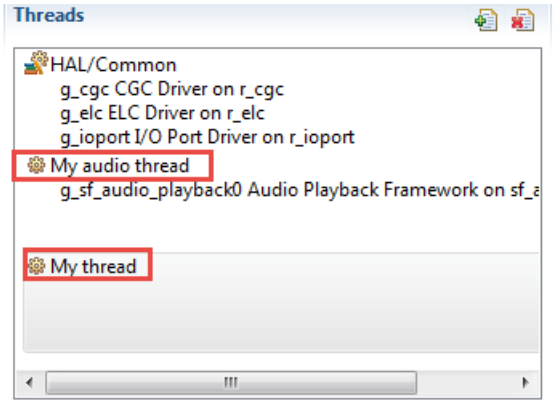


Figure 49: Messaging – Example threads

To configure the Subscriber List for an Event Class, follow these steps:

1) In the **Messaging** tab, select the Event Classes in the Event Classes Pane.The Subscriber List pane takes its name from the selected Event Class.
2) Click the Add Icon.The New Subscriber Dialog box opens.

3) Select the Thread to add to the Subscriber List from the Thread dropdown menu.

4) Fill out the instance range by selecting Start and End.If you only have one instance of an Event Class, keep the Start and End values at their default value (0). See the Messaging Framework User Guide for selecting an instance range if you have more than one Event Class instance. Multiple Event Class instances can be useful in an application that uses the same Event Class multiple times for example for audio streaming on multiple channels.

5) Repeat steps 3 and 4 for each thread that you want to add to the Subscriber List for the selected Event Class.

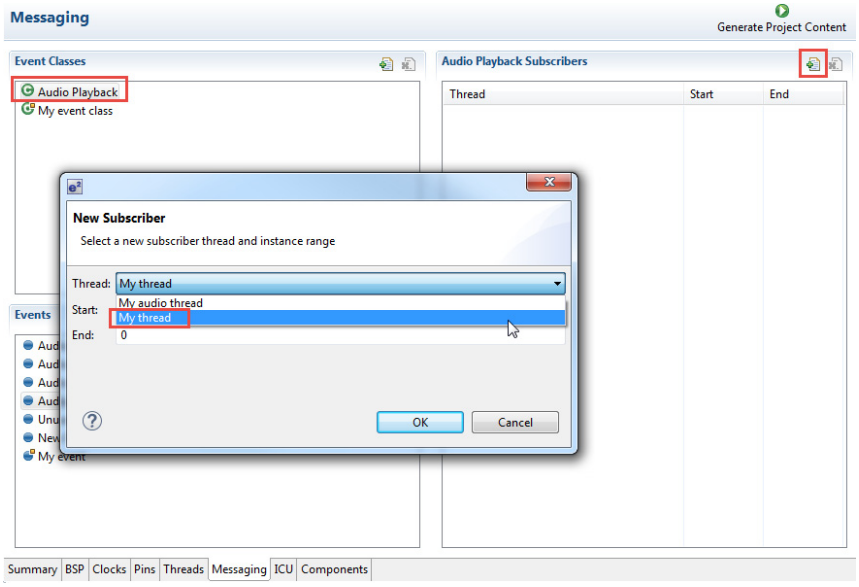


Figure 50: Messaging – Configuring the Subscriber List

3.1.7.4 Generating Files for the Messaging Framework

The ISDE generates the following files for the configured Messaging Framework when you click the Generate Project button:

File	Contents	Overwritten by Generate Project Content?
synergy_cfg/ssp_cfg/framework/sf_message_port.h	Contains the event class and event enumerations	Yes
synergy_cfg/ssp_cfg/framework/sf_message_payloads.h	Contains pointers to the event class payloads.	Yes
synergy_cfg/ssp_cfg/framework/sf_message_payloads.h	Compiler options for the Messaging Framework	Yes

3.1.8 Reviewing and Adding Components

The **Components** tab enables the individual modules required by the application to be included or excluded. Modules common to all Synergy projects are preselected (for example: BSP > BSP > Board-specific BSP and HAL Drivers > all > r_cgc). All modules that are necessary for the modules selected in the **Threads** tab are included automatically. You can include or exclude additional modules by ticking the box next to the required component.

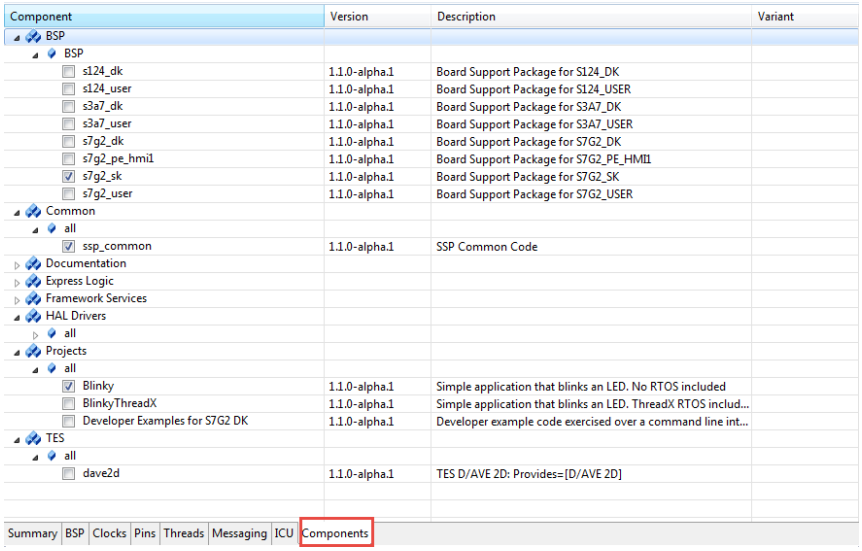


Figure 51: Components Tab

Components at the HAL and Framework layers are available as are components from Express Logic such as the RTOS ThreadX, file system FileX, TCP/IP networking NetX. In addition, you can select documentation to be added to a project or include complete projects.

While the components tab selects modules for a project, you must configure the modules themselves in the other tabs. Pressing the **Generate Project Content** button copies the .c and .h files for each component for a Pack file into the following folders:

synergy/ssp/inc/bsp

synergy/ssp/inc/driver

synergy/ssp/inc/framework

synergy/ssp/src/bsp

synergy/ssp/src/driver

synergy/ssp/src/framework

The ISDE also creates configuration files in the synergy_cfg/ssp_cfg folder with configuration options included from the remaining **Threads** and **ICU** tabs.

3.1.9 Writing the Application

Once you have added Modules and drivers and set their configuration parameters in the Threads tab, you can add the application code that calls the Modules and drivers.

NOTE: To check your configuration, build the project once without errors before adding any of your own application code.

3.1.9.1 RTOS-independent Applications

To write application code:

1) Add all drivers and modules in the **Threads** tab and resolve all dependencies flagged by the ISDE such as missing interrupts or drivers.
2) Configure the drivers in the **Properties** view.
3) In the Project Configuration view, press the **Generate Project Content** button.
4) In the **Project Explorer** view, double-click on the src/hal_entry.c file to edit the source file.

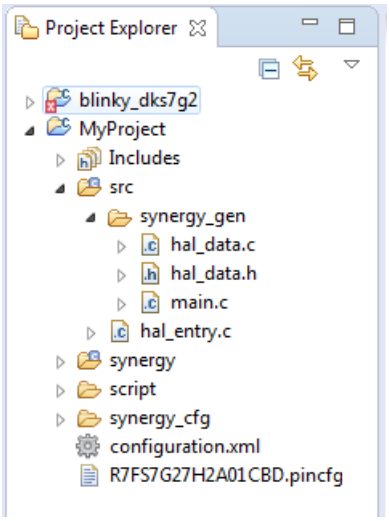


Figure 52: hal_entry.c

NOTE: All configuration structures necessary for the driver to be called in the application are initialized in src/synergy_gen/hal_data.c.

NOTE: Do not modify the files in the directory src/synergy_gen. These files are overwritten every time you push the **Generate Project Content** button.

5) Add your application code here:

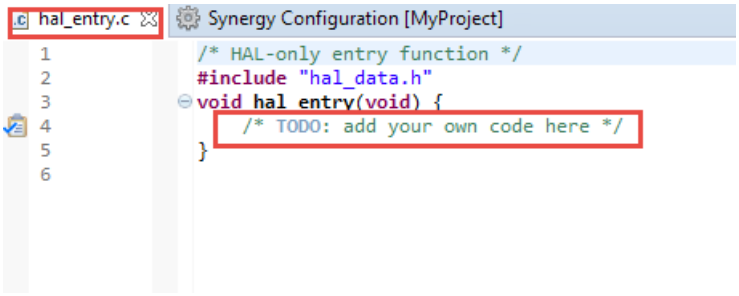


Figure 53: Adding user code to hal_entry.c

6) Build the project without errors by clicking on **Project > Build Project**.

The following tutorial shows how execute the steps above and add application code: Tutorial: Using HAL Drivers - Programming the WDT

The WDT example is a HAL level application which does not use an RTOS. The user guides for each module also include basic application code that you can add to hal_entry.c.

3.1.9.2 ThreadX Applications

To write RTOS-aware application code using ThreadX, follow these steps:

1) Add a thread using the **Threads** tab.
2) Provide a unique name for the thread in the **Properties** view for this thread.
3) Configure all drivers and resources for this thread and resolve all dependencies flagged by the ISDE such as missing interrupts or drivers.
4) Configure the thread objects.
5) Provide unique names for each thread object in the **Properties** view for each object.
6) Add more threads if needed and repeat steps 1 to 5.
7) In the **Synergy Project Editor**, press the **Generate Project Content** button.
8) In the **Project Explorer** view, double-click on the src/my_thread_1_entry.c file to edit the source file.

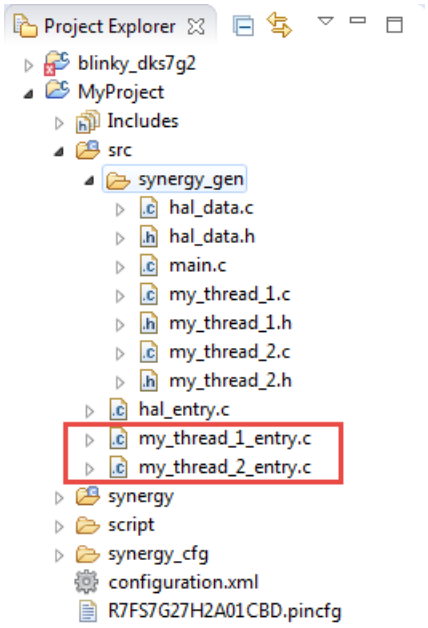


Figure 54: ISDE generated files for an RTOS application

NOTE: All configuration structures necessary for the driver to be called in the application are initialized in synergy_gen/my_thread_1.c and my_thread_2.c

NOTE: Do not modify the files in the directory src/synergy_gen. These files are overwritten every time you push the **Generate Project Content** button.

9) Add your application code here:

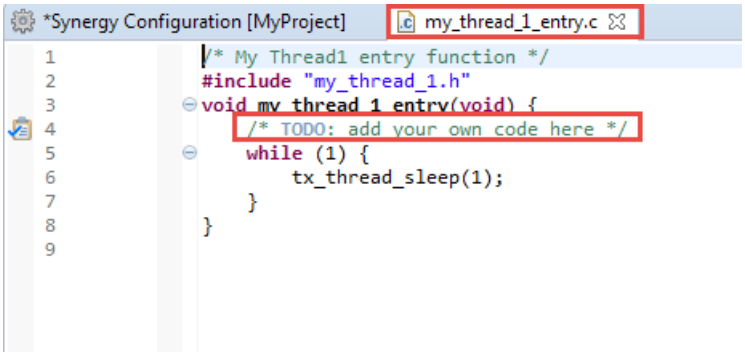


Figure 55: Adding user code to my_thread_1.entry

10) Repeat steps 1 to 9 for the next thread.

11) Build your project without errors by clicking on **Project > Build Project**.

3.1.10 Debugging the Project

Once your project builds without errors, you can use the Debugger to download your application to the board and execute it.

To debug an application follow these steps:

1) On the drop-down list next to the debug icon, select **Debug Configurations**.

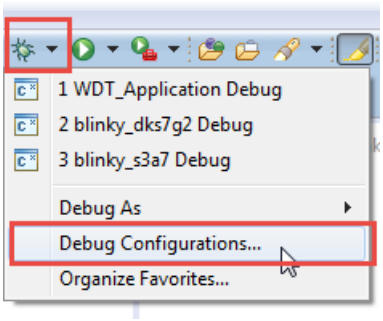


Figure 56: Invoking the Debug Configurations dialog

2) In the **Debug Configurations** view, click on your project listed as **MyProject Debug**.

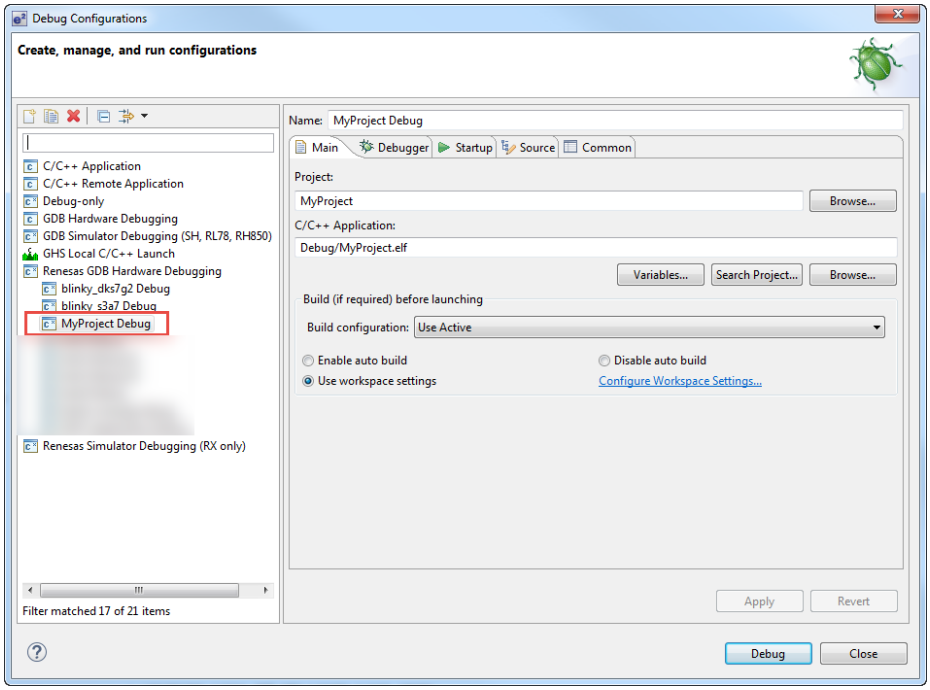


Figure 57: Debug Configuration

3) Connect the board to your PC via either a standalone Segger J-Link debugger or a Segger J-Link On-Board (included on all Synergy DKs and SKs) and click **Debug**.

NOTE: For details on using J-Link and connecting the board to the PC, see the Quick Start Guide included in the Synergy Kit.

3.1.11 Using TraceX with a Synergy Project

NOTE: Before you can use TraceX with your Synergy Project, you must download the TraceX executable file from the Synergy Gallery Website.

TraceX™ is a host-based analysis tool that provides a graphical view of real-time system events. TraceX collects data on the target device and displays the data for inspection and analysis. A TraceX version for Synergy devices is available for downloading on the Synergy Gallery Website and can be used together with the Synergy License.

To use TraceX, do the following:

1) Enable TraceX in the Properties Window of the Thread using the **Threads** tab. Keep the default name for the TraceX buffer as g_tx_trace_buffer.

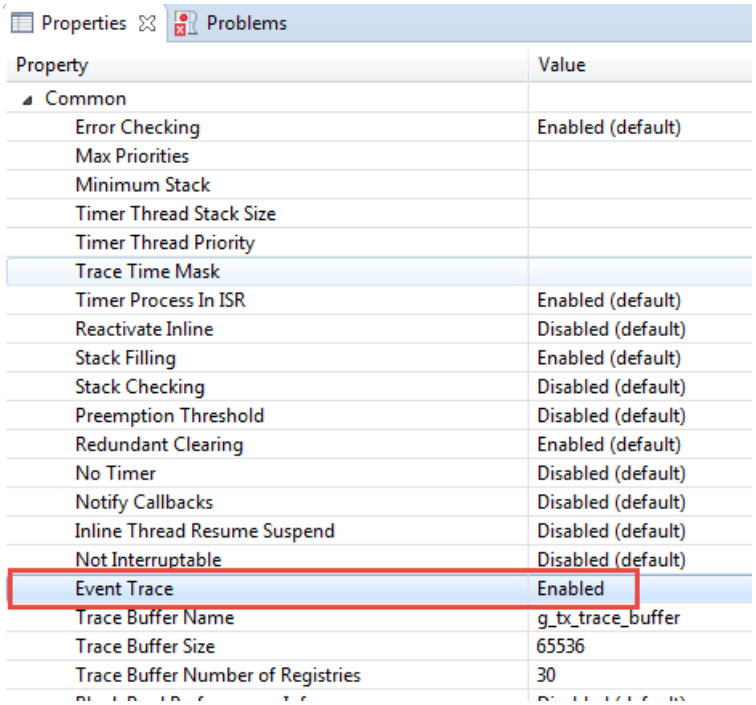


Figure 58: ISDE TraceX Configuration

2) Set the path to the TraceX application in **Window > Preferences > C/C++ > Renesas > TraceX**

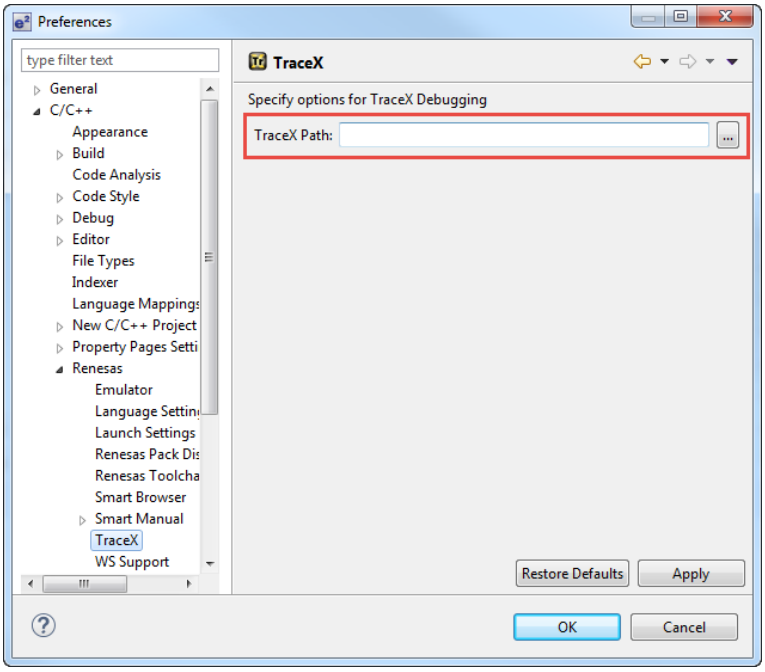


Figure 59: ISDE TraceX Path

3) In **Run > TraceX**, select **Launch TraceX Debugging**

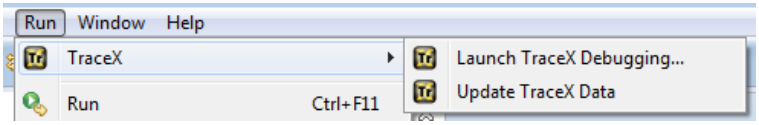


Figure 60: ISDE TraceX Launch

4) In the TraceX Debugging window, set **Buffer Start Address** to &g_tx_trace_buffer.

5) In the TraceX Debugging window, set **Buffer Size (bytes)** to the buffer size selected in the Properties Window in step 1. The default is 65536.

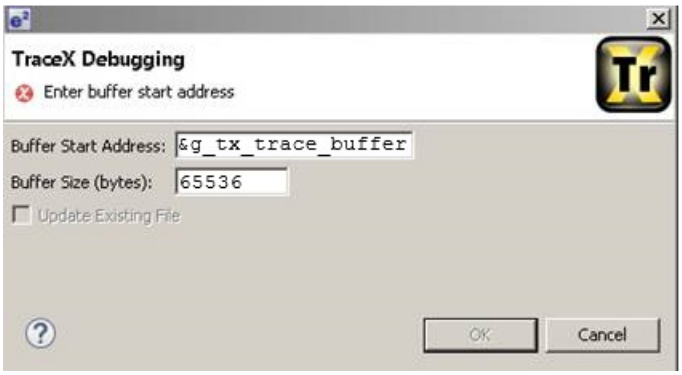


Figure 61: ISDE TraceX Debug

6) Click **OK**.

You can find details on how to use TraceX on the Express Logic website.

3.1.12 Modifying Toolchain Settings

There are instances where it may be necessary to make changes to the toolchain being used (for example, to change optimization level of the compiler or add a library to the linker). Such modifications can be made from within the ISDE through the menu **Project > Renesas Tool Settings** when the project is selected. The following screenshot shows the settings dialog for the GNU ARM toolchain. This dialog will look slightly different depending upon the toolchain being used.

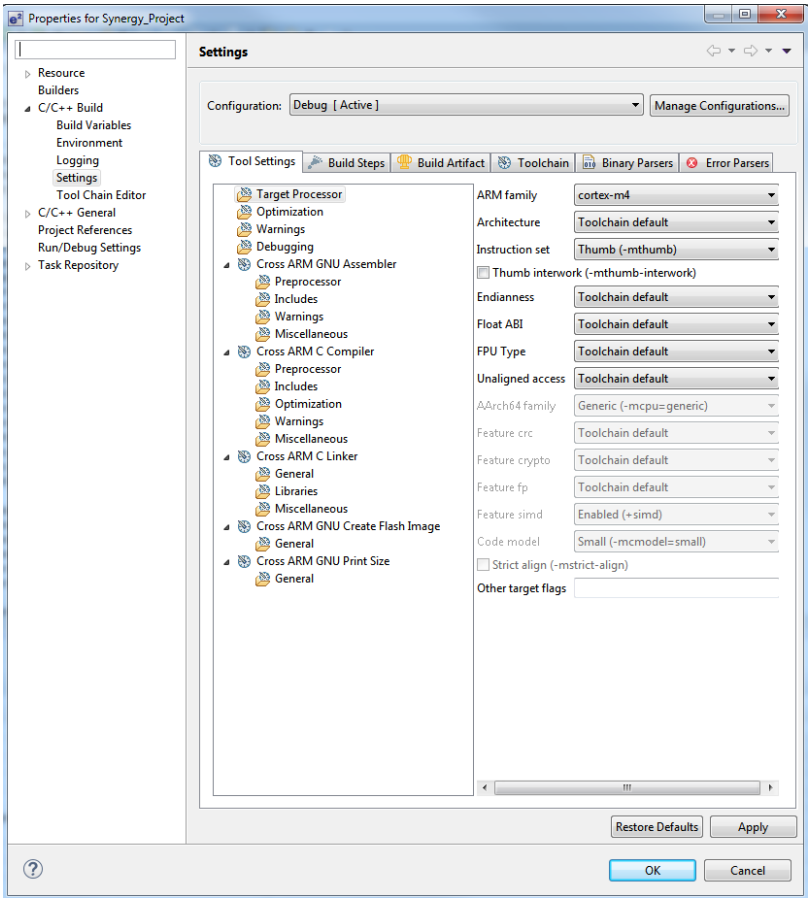


Figure 62: ISDE Project toolchain settings

The scope for the settings is project scope which means that the settings are valid only for the project being modified.

The settings for the linker which control the location of the various memory sections are contained in a script file specific for the device being used. This script file is included in the project when it is created and is found in the script folder (for example, /script/S7G2.ld).

3.1.13 e² studio ISDE Usage Notes

3.1.13.1 Handling SSP Releases

Several release packs of the SSP are available on the Synergy Gallery website.

There are two types of SSP packs:

1) X.Y.0 Gold – Production release with full SQA. Contains only X.Y.0 pack.
2) X.Y.Z – Patch release based on released X.Y.0 version (X=major; Y=minor; Z=patch). Only contains added or updated modules in X.Y.Z pack. (Does not include full X.Y.0 pack.)

You can install multiple X.Y.Z patch releases in addition to the X.Y.0 production release.

For example, you can install the following SSP versions: - SSP v1.0.0: Production release - SSP v1.0.1: First patch release - SSP v1.0.2: Second patch release - SSP v1.0.3: Third patch release

Patch releases may or may not include a Board Support Package (BSP). When you create a new Synergy Project using the Synergy Project Generator, select the highest Patch Release that includes a BSP in the **SSP Version** field. Whether or not an SSP release includes a BSP can be identified by available BSP selections in the **Board** field. If the selected patch release does not contain a BSP (as is the case in patch release v0.91.3), this is indicated by the following message: < No BSP module >

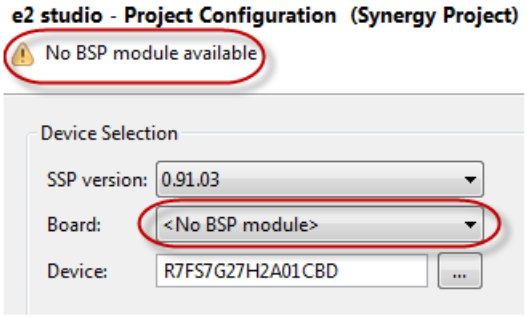


Figure 63: e2 studio installing multiple SSP versions

In this case, select the next-lowest SSP version that includes a BSP. Once you selected an SSP version with a valid BSP, the e2 studio ISDE automatically selects the SSP components from the highest available SSP version to be included in the project. You can verify this by looking up the module in the Components tab.

If any of the SSP components on the Components tab do not match exactly the selected SSP version (including "patch" version), then a warning icon appears next to the SSP Version field on the BSP tab.

In the following example, a driver module has been selected using v0.91.03 on top of SSP version 0.91.02. This selection contains a valid BSP (0.91.02), but the selected component belongs to a newer patch release (0.91.03).

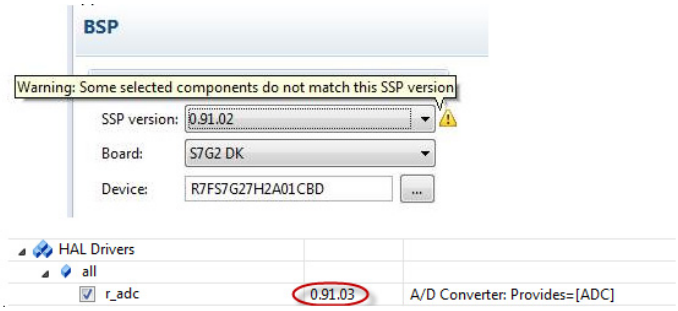


Figure 64: e2 studio Components mismatch when installing multiple SSP versions

ATTENTION: Do not select multiple versions of the same component (from different SSP packs) on the Components tab.

3.1.13.2 Including ThreadX sources

You can use the **Theads** tab to include ThreadX source code in your project as follows:

1) Click on the HAL/Common icon in the **Threads** pane. The Modules pane changes to **HAL/Common Modules**.
2) Select **New** > **Framework** > **RTOS** > **ThreadX Source**.
3) Check the **Properties** window to configure the TheadX RTOS properties.
4) Click **Generate Project**.

The e^2 studio ISDE extracts the ThreadX source code into the following directory: synergy/ssp/src/framework/el/tx

NOTE: You can see and modify the ThreadX source code only with the appropriate license. See Licensing.

NOTE: Extracting the ThreadX sources increases the compile time for your project.

3.2 Tutorial: Your First Synergy Project - Blinky

The goal of this tutorial is to quickly get acquainted with the Synergy Platform by moving through the steps of creating a simple application using e^2 studio and running that application on a Synergy board.

3.2.1 What Does Blinky Do?

The application used in this tutorial is Blinky, traditionally the first program run in a new embedded development environment.

Blinky is the "Hello World" of microcontrollers. If the LED blinks you know that:

- The toolchain is setup correctly and builds a working executable image for your chip.
- The debugger has installed with working drivers and is properly connected to the board.
- The board is powered up and its jumper and switch settings are probably correct.
- The microcontroller is alive, the clocks are running, and the memory is initialized.

The Blinky example application used in this tutorial is designed to run the same way on all boards offered by Renesas that hold the Synergy microcontroller. The code in Blinky is completely board independent. It does the work by calling into the BSP (board support package) for the particular board it is running on. This works because:

- Every board has at least one LED connected to a GPIO pin
- That one LED is always labeled LED1 on the silk screen.
- Every BSP supports an API that returns a list of LEDs on a board, and their port and pin assignments.

3.2.2 Prerequisites

To follow this tutorial, you need:

- Windows based PC
- e^2 studio
- Synergy Software Package
- A Synergy board kit

3.2.3 Create a New Project for Blinky

The creation and configuration of a Synergy project is the first step in the creation of an application. The base SSP pack includes a pre-written Blinky example application that is simple and works on all Renesas Synergy boards.

Follow these steps to create a Synergy project:

1) In e^2 studio ISDE, click **File > New > Synergy Project**
2) Assign a name to this new project. Blinky is a good name to use for this tutorial.
3) Identify the license file if the license window is empty. The license file for this version of the platform can be found in your <ISDE base directory> starting here: ISDE\internal\projectgen\arm\Licenses\SSP_License_Example_EvalLicense_<rev>.xml.
4) Click **Next**. The Project Configuration window shows your selection.

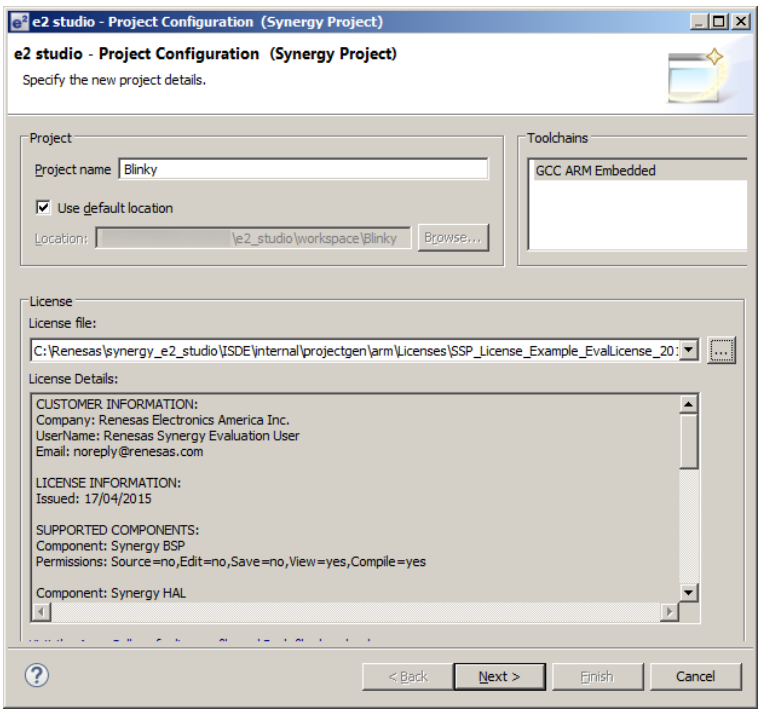


Figure 65: e^2 studio ISDE Project Configuration window (part 1)

5) Select the board support package by selecting the name of your board from the **Device Selection** drop-down list and click **Next**.

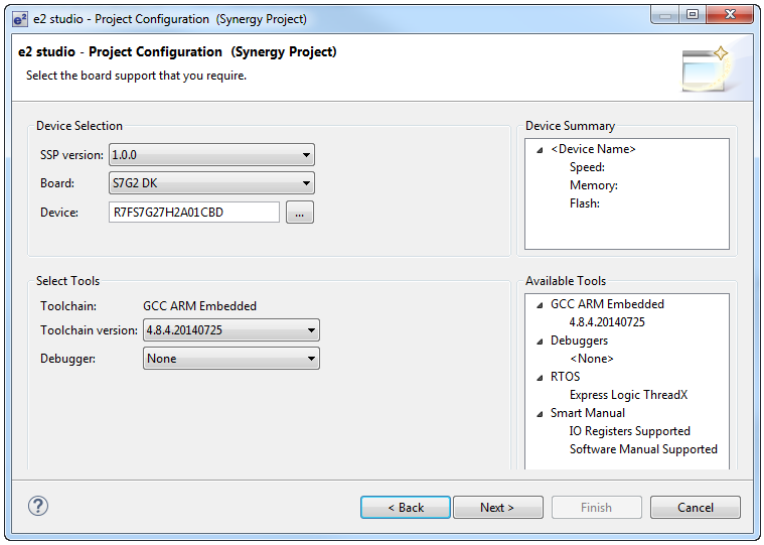


Figure 66: e² studio ISDE Project Configuration window (part 2)

6) Select the Blinky template for your board and click **Finish**.

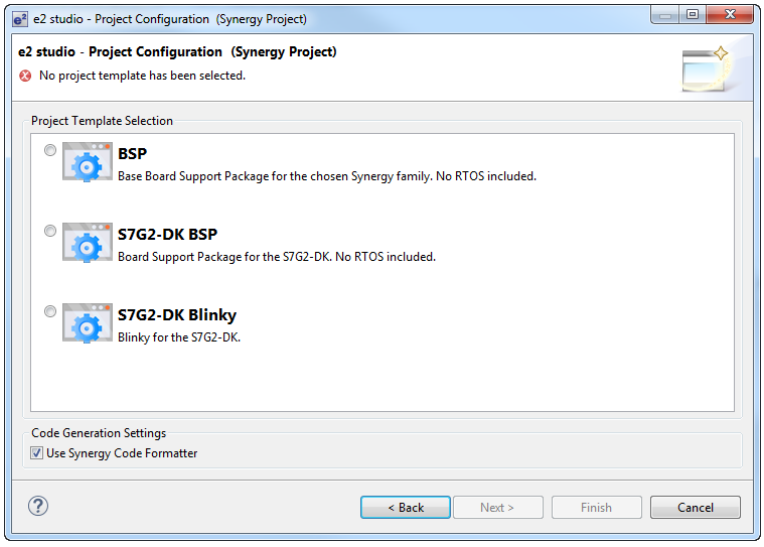


Figure 67: e² studio ISDE Project Configuration window (part 3)

Once the project has been created, the name of the project will show up in the Project Explorer window of the ISDE. Now press the **Generate Project Content** button in the top right corner of the **Project Configuration** window to generate your board specific files.

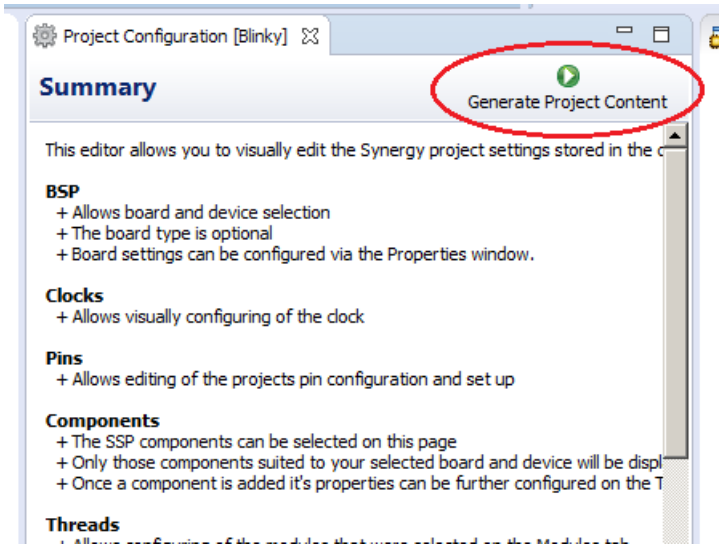


Figure 68: e² studio ISDE Project Configuration tab

Your new project is now created, configured, and ready to build.

3.2.3.1 Details about the Blinky Configuration

The **Generate Project Content** button creates configuration header files, copies source files from templates, and generally configures the project based on the state of the **Project Configuration** screen.

For example, if you check a box next to a module in the **Components** tab and press the **Generate Project Content** button all the files necessary for the inclusion of that module into the project will be copied or created. If that same check box is then unchecked those files will be deleted.

3.2.3.2 Configuring the Blinky Clocks

By selecting the Blinky template, the clocks are configured by the ISDE for the Blinky application. The ISDE clock configuration tab (see Configuring Clocks) shows the Blinky clock configuration. The Blinky clock configuration is stored in the BSP clock configuration file (see BSP Clock Configuration).

3.2.3.3 Configuring the Blinky Pins

By selecting the Blinky template, the GPIO pins used to toggle the LED1 are configured by the ISDE for the Blinky application. The ISDE pin configuration tab shows the pin configuration for the Blinky application (see Configuring Pins). The Blinky pin configuration is stored in the BSP configuration file (see BSP Pin Configuration).

3.2.3.4 Configuring the Parameters for Blinky Components

The Blinky project automatically selects the following HAL components in the ISDE Component:

- r_cgc
- r_elc

- r_ioport

To see the configuration parameters for any of the components, check the Properties tab in the **HAL** window for the respective driver (see Adding and Configuring HAL Drivers).

3.2.3.5 Where is main()?

The main function is located in < project >/src/synergy_gen/main.c. It is one of the files that are generated during the project creation stage and only contains a call to hal_entry(). For more information on generated files Adding and Configuring HAL Drivers .

3.2.3.6 Blinky Example Code

The blinky application is stored in the hal_entry.c file. This file is generated by the ISDE when you select the Blinky Project template and is located in the project's src/ folder.

The application performs the following steps:

1) Get the LED information for the selected board by calling the BSP HAL function R_BSP_LedsGet.
2) Define the output level HIGH for the GPIO pins controlling the LEDs for the selected board.
3) Get the selected system clock speed and scale down the clock, so the LED toggling can be observed.
4) Toggle the LED by writing to the GPIO pin with

```
g_ioport.p_api->pinWrite()
```

3.2.4 Build the Blinky Project

Highlight your new project in the **Project Explorer** window and build it.

There are three ways to build a project:

a. Click on Project in the menu bar and select Build Project.

b. Click on the hammer icon.

c. Right-click on the project and select Build Project.

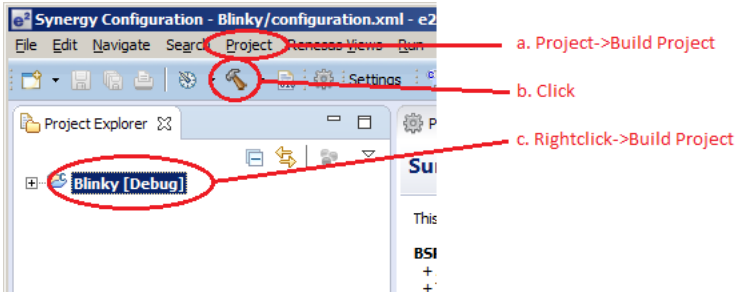


Figure 69: e² studio ISDE Project Explorer window

Once the build is complete a message is displayed in the build **Console** window that displays the final image file name and section sizes in that image.

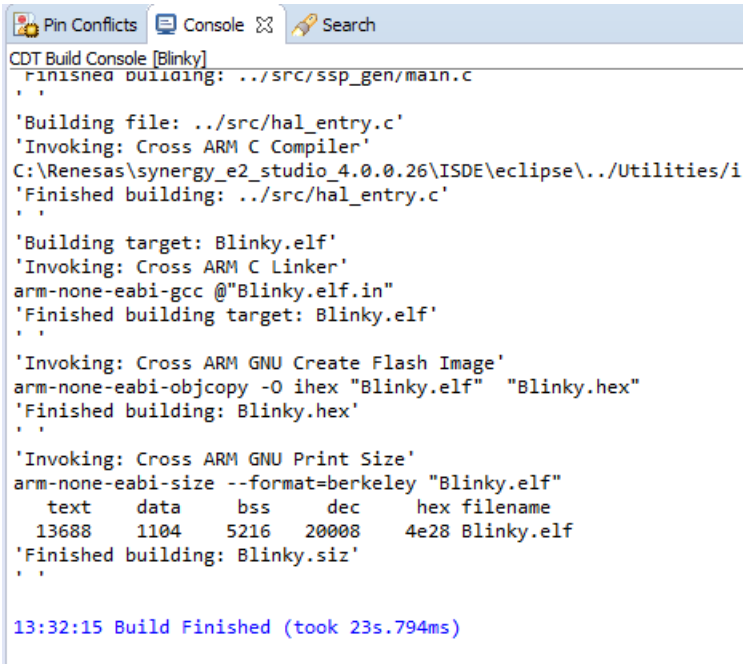


Figure 70: e^2 studio ISDE Project Build console

3.2.5 Debug the Blinky Project

3.2.5.1 Debug prerequisites

To debug the project on a board, you need

- the board to be connected to the ISDE
- the debugger to be configured to talk to the board
- the application to be programmed to the microcontroller

Applications run from the internal flash of your microcontroller. To run or debug the application, the application must first be programmed to the microcontroller's flash. There are two ways to do this:

- JTAG debugger
- Built-in boot-loader via UART or USB

Some boards have an on-board JTAG debugger and others require an external JTAG debugger connected to a header on the board.

Refer to your board's user manual to learn how to connect the JTAG debugger to your ISDE.

3.2.5.2 Debug steps

To debug the Blinky application, follow these steps:

1) Configure the debugger for your project by clicking **Run > Debugger Configurations**...

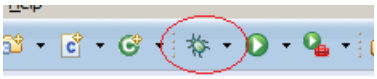


Figure 71: e² studio ISDE Debug icon

… or by selecting the drop-down menu next to the bugicon and selecting **Debugger Configurations**…

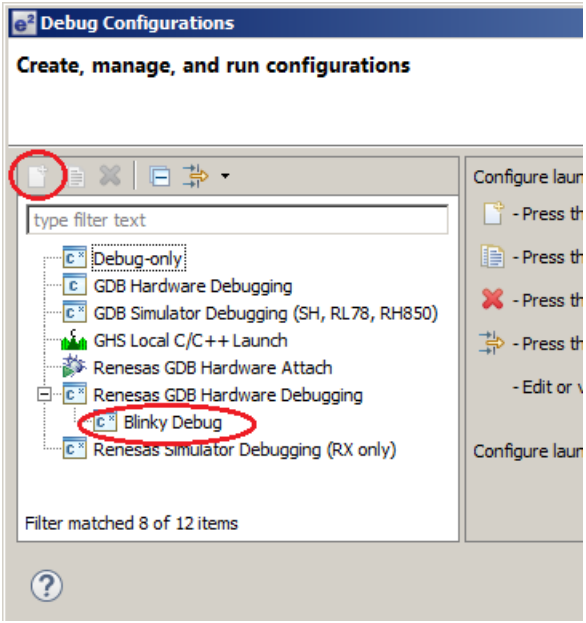


Figure 72: e² studio ISDE Debugger Configurations window

2) Select your debugger configuration in the window. If it is not visible then it must be created by clicking the **New** icon in the top left corner of the window. Once selected, the **Debug Configuration** window displays the Debug configuration for your Blinky project.

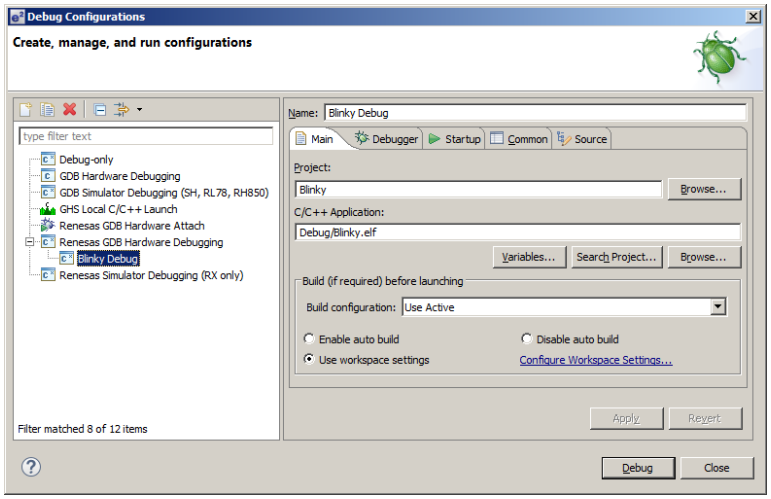


Figure 73: e^2 studio ISDE Debugger Configurations window with Blinky project

3) Press **Debug** to begin debugging the application.

3.2.5.3 Details about the Debug Process

In debug mode, the ISDE executes the following tasks:

1) Downloading the application image to the microcontroller and programming the image to the internal flash memory.
2) Setting a breakpoint at main().
3) Setting the stack pointer register to the stack.
4) Loading the program counter register with the address of the reset vector.
5) Displaying the startup code where the program counter points to.

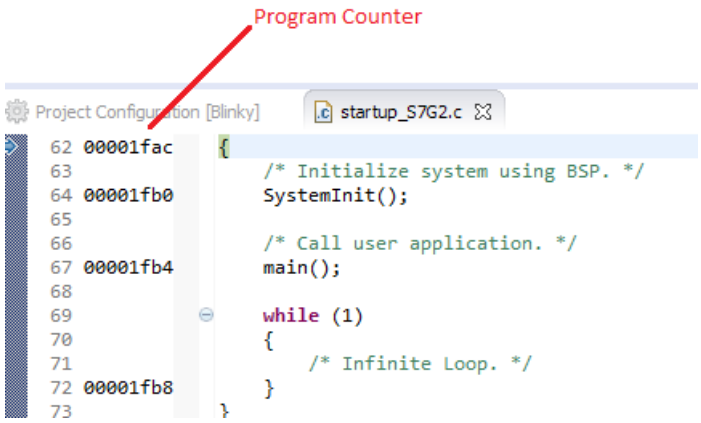


Figure 74: e² studio ISDE Debugger memory window

3.2.6 Run the Blinky Project

While in Debug mode, click **Run > Resume** or click on the Play icon twice.

Figure 75: e² studio ISDE Debugger Play icon

The LED on the board marked LED1 should now be blinking.

3.3 Tutorial: Using HAL Drivers - Programming the WDT

This application uses the WDT Interface implemented by the WDT HAL Driver WDT. This document describes how to use the ISDE and SSP to create an application for the Synergy MCU Watchdog Timer (WDT) peripheral. This application makes use of the following SSP modules:

- Board Support Package (Board Support Package)
- CGC (Clock Generation Circuit)
- WDT (Watchdog Timer)
- IOPORT (GPIO)

3.3.1 Creating a WDT Application Using the Synergy SSP and ISDE

3.3.1.1 Using the SSP and the e^2 studio ISDE

The Synergy Software Package (SSP) from Renesas provides a complete driver library for developing Synergy applications. The SSP provides Hardware Abstraction Layer (HAL) drivers, Board Support Package (BSP) drivers and higher level Framework applications for the developer to use to create applications. The SSP is integrated into the Renesas e^2 studio Integrated Solution Development Environment (ISDE) based on eclipse providing build (editor, compiler and linker) and debug phases with an extended GNU Debug (GDB) interface.

3.3.1.2 The WDT Application

The flowchart for the WDT application is shown below.

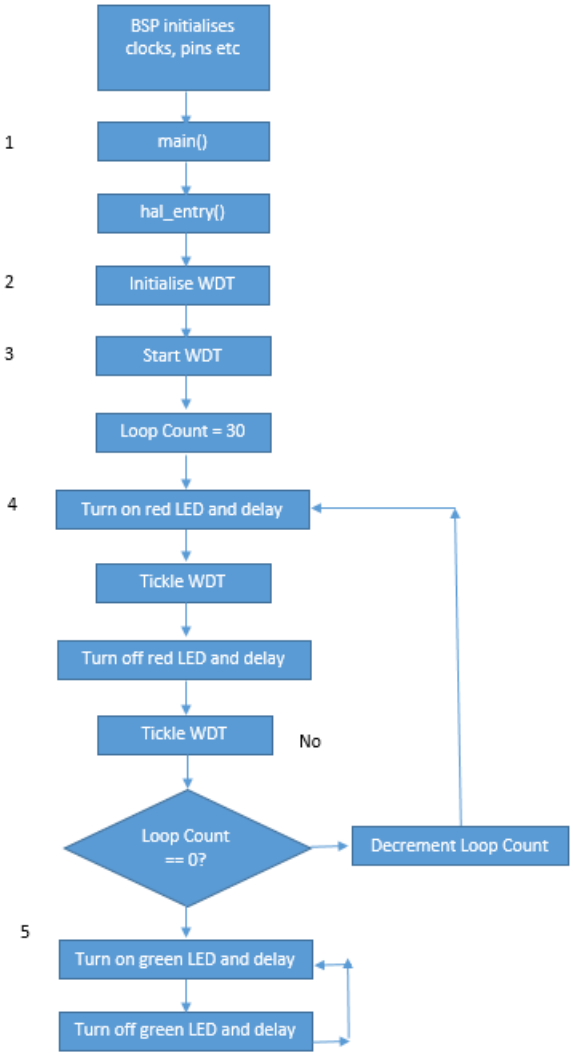


Figure 76: WDT Application flow diagram

3.3.1.3 WDT Application flow

These are the main parts of the WDT application:

1) main() calls hal_entry(). The function hal_entry() is created by the SSP with a placeholder for user code. The code for the WDT will be added to this function.
2) Initialize the WDT, but do not start it.
3) Start the WDT by refreshing it.
4) The red LED is flashed 30 times and refreshes the watchdog each time the LED state is changed.

5) Flash the green LED but DO NOT refresh the watchdog. After the timeout period of the watchdog the device will reset which can be observed by the flashing red LED again as the sequence repeats.

3.3.2 Creating the Project with the ISDE

Start the ISDE and choose a workspace folder in the Workspace Launcher. Configure a new Synergy project as follows.

1) Select **File > New > Synergy Project**

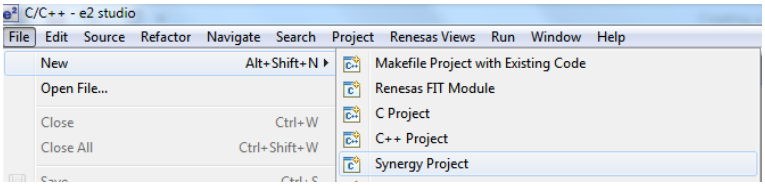


Figure 77: Creating a new project

2) In the ISDE Project **Configuration (Synergy Project)** window enter a project name e.g. WDT_Application. In addition select the toolchain and license file. If you want to choose a new location for the project unselect **Use default location**. Click **Next**.

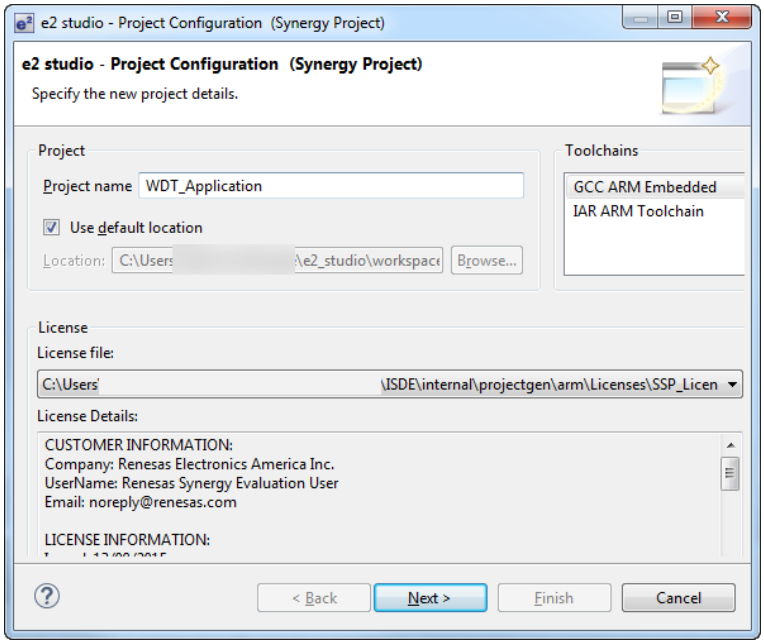


Figure 78: Project configuration (part 1)

3) This application runs on the Synergy S7G2 based DK-S7G2 board. So, for the **Board** select **S7G2 DK**.This will automatically populate the **Device** drop-down with the correct device used on this board. Select the **Toolchain** version. Select **J-Link ARM** as the **Debugger**. No **RTOS** is being used in this application but it can be left at the default **Express Logic ThreadX**. Click **Next** to configure the project.

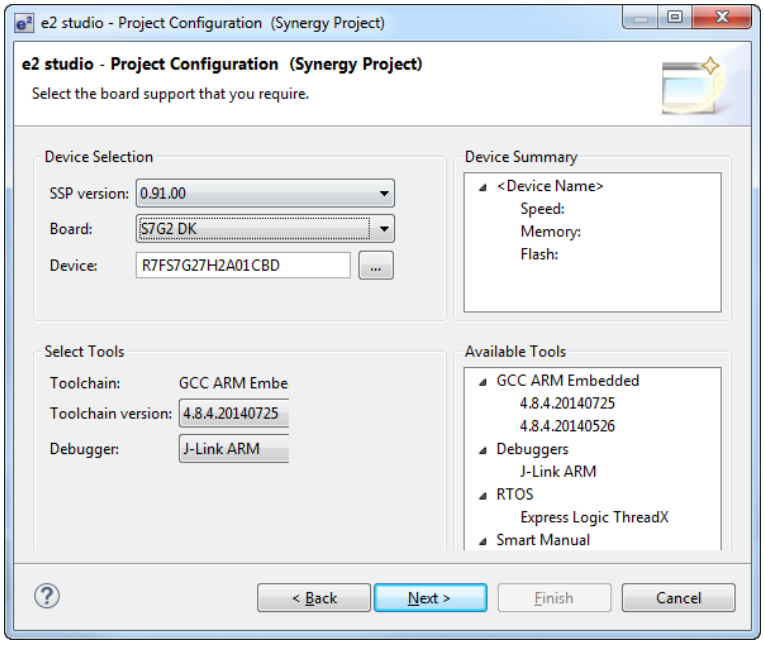


Figure 79: Project configuration (part 2)

The project template is now selected. As no RTOS is required select **S7G2-DK BSP**.

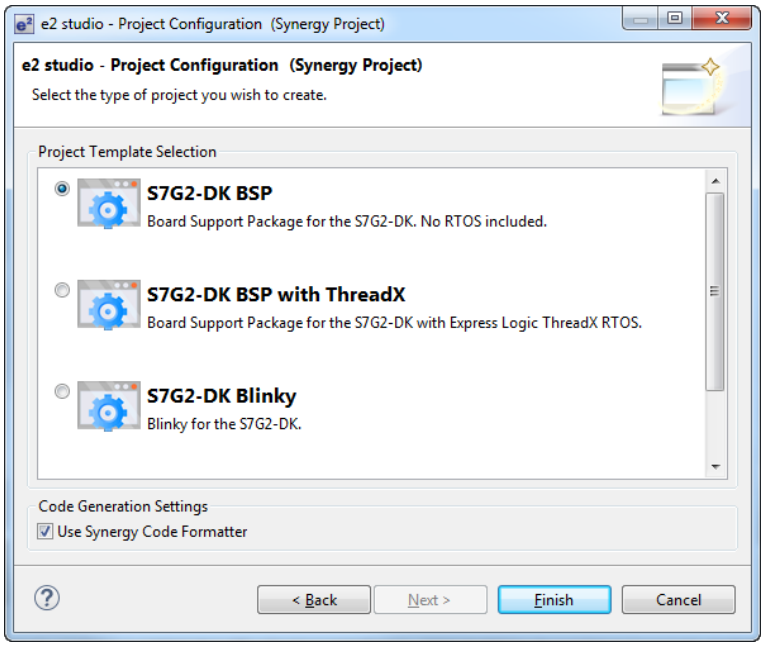


Figure 80: Project configuration (part 3)

4) Click **Finish**.

The ISDE creates the project and opens the **Project Explorer** and Project Configuration Settings** views with the **Summary** page showing a summary of the project configuration.

3.3.3 Configuring the Project with the ISDE

The e^2 studio ISDE simplifies and accelerates the project configuration process by providing a GUI interface for selecting the options to configure the project.

The ISDE offers a selection of perspectives presenting different windows to the user depending on the operation in progress. The default perspectives are **C/C++**, **Synergy Configuration** and **Debug**. The perspective can be changed by selecting a new one from the buttons at the top right of the ISDE.

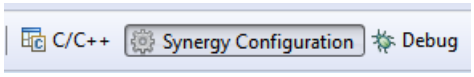


Figure 81: Selecting a perspective

The **C/C++** perspective provides a layout selected for code editing. The **Synergy Configuration** perspective provides elements for configuring a Synergy project, and the **Debug** perspective provides a view suited for debugging.

1) In order to configure the project settings ensure the **Synergy Configuration** perspective is selected.
2) Ensure the **Project Configuration [WDT Application]** is open. It is already open if the Summary information is visible. To open the Project Configuration now or at any time make sure the **Synergy Configuration** perspective is selected and double-click on the **configuration.xml** file in the Project Explorer pane on the right side of the ISDE.

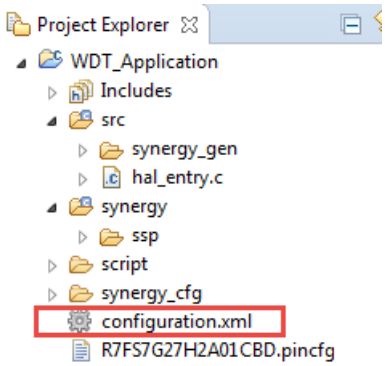


Figure 82: Synergy Project Configuration Settings

At the base of the Project Configuration view there are several tabs for configuring the project. A project may require changes to some or all of these tabs. The tabs are shown below.

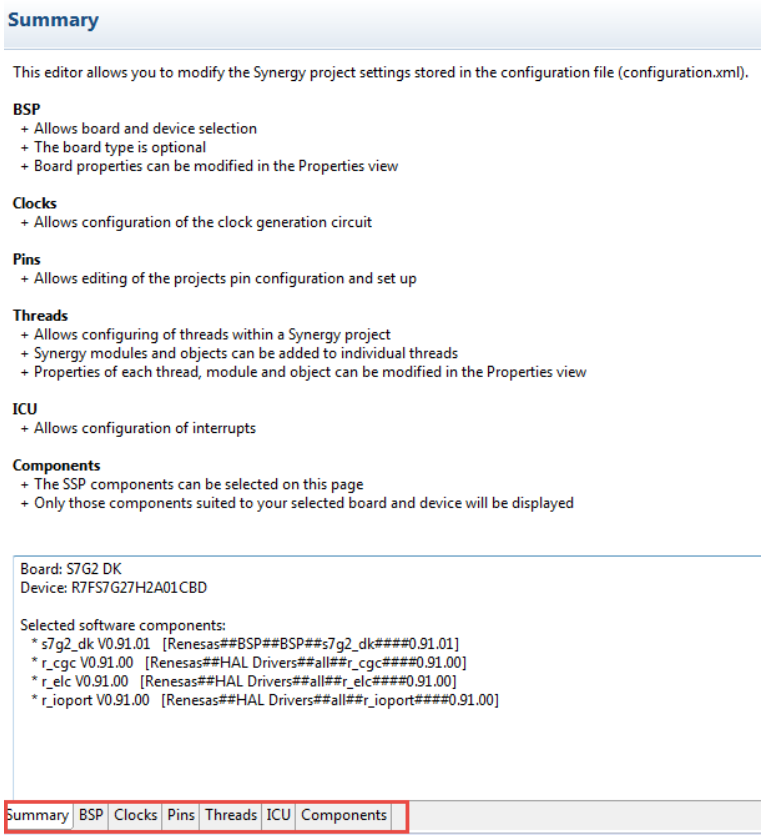


Figure 83: Project Configuration Tabs

3.3.3.1 BSP Tab

The BSP tab allows the Board Support Package (BSP) options to be modified from their defaults. For this particular WDT project no changes are required. However, if you want to use the WDT in auto-start mode, you can configure the settings of the OFS0 (Option Function Select Register 0) register in the BSP tab. See the Synergy Hardware User's Manual for details on the WDT autostart mode.

3.3.3.2 Clocks Tab

The clocks tab presents a graphical view of the clock tree of the device. Using the drop down boxes in the GUI enables configuration of the various clocks. The WDT uses PCLCKB. The default output frequency for this clock is 60 MHz. Ensure this clock is outputting this value.

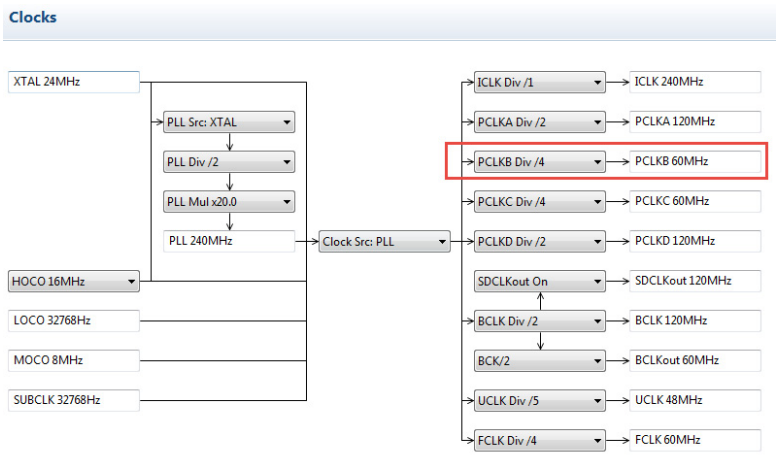


Figure 84: Clock configuration

3.3.3.3 Pins Tab

The Pins tab provides a graphical tool for configuring the functionality of the pins of the device. For the WDT project no pin configuration is required. Although the project uses two LEDs connected to pins on the device, these pins are pre-configured as output GPIO pins by the BSP.

3.3.3.4 Threads Tab

You can add any driver to the project using the Threads tab. The HAL drivers for the Clock Generation Circuit, the Event Link Controller, and the IO port pins are added automatically by the ISDE when the project is configured. The WDT application uses no ThreadX Resources, so you only need to add the HAL WDT driver.

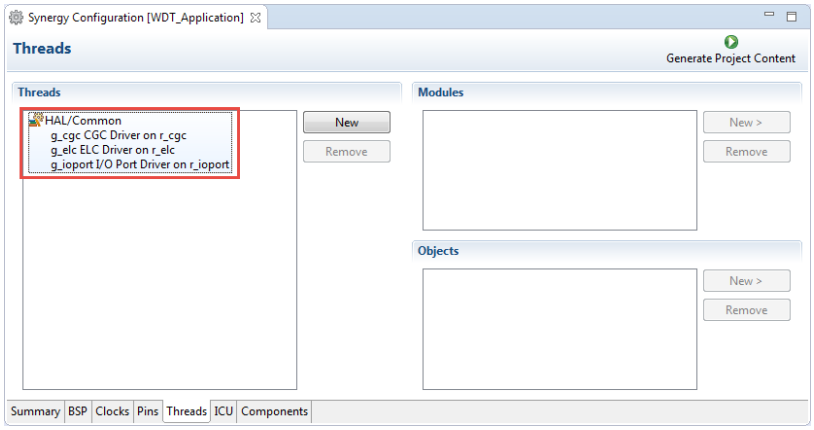


Figure 85: Threads tab

1) Click on the **HAL/Common Panel** in the Threads Window as indicated in the figure above.The Modules Panel becomes a **HAL/Common Modules** panel and is populated with the modules preselected by the ISDE.
2) Click on **New >** to find a pop-up window with the available HAL level drivers.
3) Select **WATCGDOG Driver on WDT**.

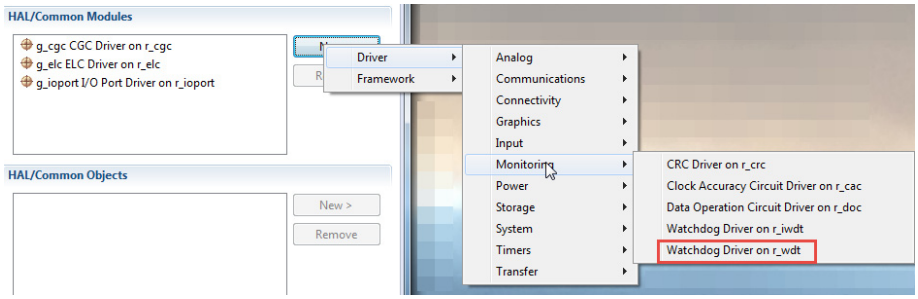


Figure 86: Module Selection

The selected HAL WDT driver is added to the **HAL/Common Modules** Panel and the **Property** Window shows all configuration options for the selected module. The **Property** tab for the WDT should be visible at the bottom left of the screen. If it not visible check that the **Synergy Configuration** perspective is selected.

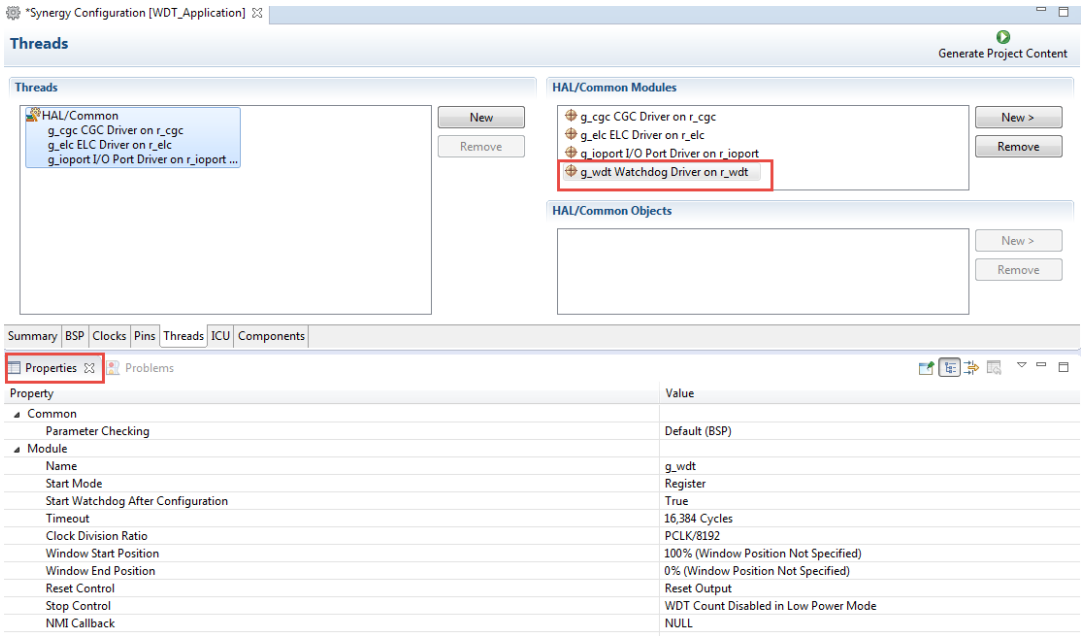


Figure 87: Module Properties

Change parameter **Start Watchdog After Configuration** from **True** to **False**. The other parameters can be left with their default values. Setting **Start Watchdog After Configuration** to **False** instructs the WDT driver (via its open API call) to configure the WDT but not to start it. It will be started later by refreshing it.

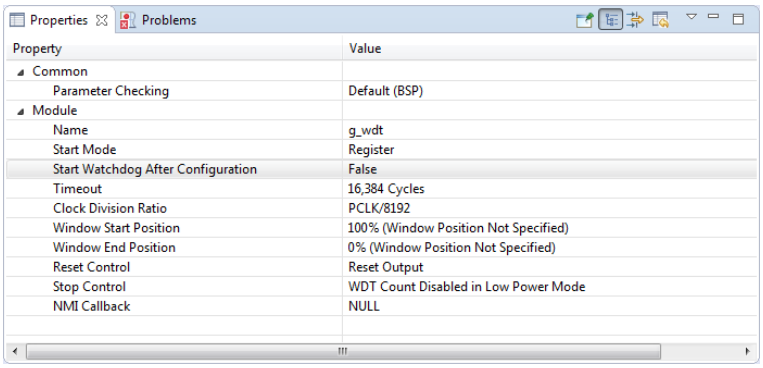


Figure 88: g_wdt WATCHDOG Driver on WDT properties

With PCLKB running at 60 MHz the WDT will reset the device 2.23 seconds after the last refresh.

WDT clock = 60 MHz / 8192 = 7.32 kHz

Cycle time = 1 / 7.324 kHz = 136.53 us

Timeout = 136.53 us x 16384 = 2.23 seconds

Save the **Project Configuration** file and click the **Generate Project Content** button in the top right corner of the **Project Configuration** pane.

Figure 89: Generate Project Content button

The ISDE generates the project files.

3.3.3.5 Components Tab

The components tab is included for reference to see which modules are included in the project. Modules are selected automatically in the Components view when after they are added in the Threads Tab.

For the WDT project ensure that the following modules are selected:

1) HAL_Drivers -> r_cgc
2) HAL_Drivers -> r_elc
3) HAL_Drivers -> r_ioport
4) HAL_Drivers -> r_wdt

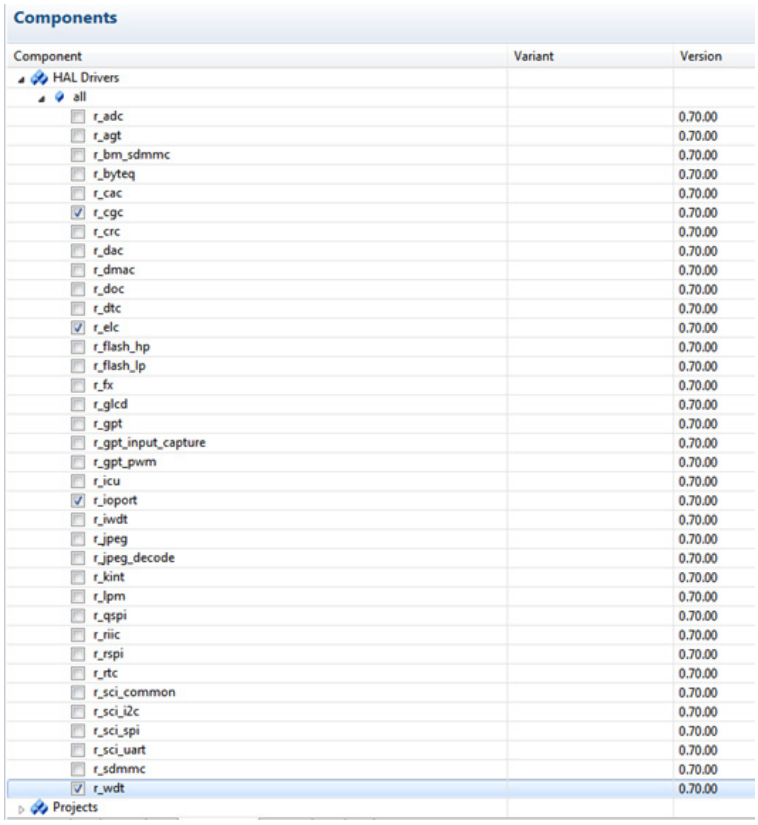


Figure 90: Component Selection

NOTE: The list of modules displayed in the Components tab depends on the installed SSP version.

3.3.4 WDT Generated Project Files

Pressing the Generate Project Content button performs the following tasks.

- r_wdt folder and WDT driver contents created at:
  - synergy/src/driver/
- r_wdt_api.h created in:
  - synergy/inc/api
- r_wdt.h created in:
  - synergy/inc/instances

The above files are the standard files for the WDT HAL module. They contain no specific project contents. They are the driver files for the WDT. Further information on the contents of these files can be found in the documentation for the WDT HAL module.

Configuration information for the WDT HAL module in the WDT project is found in:

- synergy_cfg/driver/r_wdt_cfg.h

The above file's contents are based upon the **Common** settings in the **g_wdt WATCHDOG Driver on WDT Properties** pane.

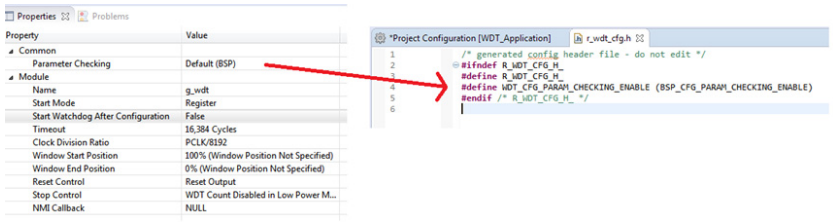


Figure 91: r_wdt_cfg.h contents

ATTENTION: Do not edit any of these files as they are recreated every time the Generate Project Content button is pressed and so any changes will be overwritten.

r_ioport folder is not created at ssp/src/driver as this module is required by the BSP and so already exists. It is included in the WDT project in order to include the correct header file in src/synergy_gen/hal_data.h – see later in this document for further details. For the same reason the other IOPORT header files – synergy/ssp/inc/api/r_ioport_api.h and synergy/ssp/inc/instances/r_ioport.h are not created as they already exist.

In addition to generating the HAL driver files for the WDT and IOPORT files the ISDE also generates files containing configuration data for the WDT and a file where user code can safely be added. These files are shown below.

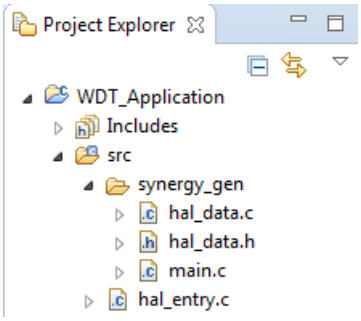


Figure 92: WDT project files

3.3.4.1 WDT hal_data.h

The contents of hal_data.h are shown below.

```
/* generated HAL header file - do not edit */

#ifndef HAL_DATA_H_

#define HAL_DATA_H_

#include <stdint.h>

#include "bsp_api.h"

#include "r_wdt.h"

#include "r_wdt_api.h"

#include "r_ioport.h"

#include "r_ioport_api.h"

#include "r_elc.h"

#include "r_elc_api.h"

#include "r_cgc.h"

#include "r_cgc_api.h"

/** WDT on WDT Instance. */

extern const wdt_instance_t g_wdt;

#ifdef NULL
```

```
#define WATCHDOG_ON_WDT_CALLBACK_USED (0)

#else

#define WATCHDOG_ON_WDT_CALLBACK_USED (1)

#endif

#if WATCHDOG_ON_WDT_CALLBACK_USED

void NULL(wdt_user_cb_data_t * p_args);

#endif

/** IOPORT on IOPORT Instance */

extern const ioport_instance_t g_ioport;

/** Pointer to ELC API */

extern const elc_instance_t g_elc;

#endif /* HAL_DATA_H_ */
```

hal_data.h contains the header files required by the ISDE generated project. In addition this file includes external references to the **g_wdt** instance structure which contains pointers to the configuration, control, api structures used for WDT HAL driver.

ATTENTION: This file is regenerated each time Generate Project Content is pressed and must not be edited.

3.3.4.2 WDT hal_data.c

The contents of hal_data.c are shown below.

```
/* generated HAL source file - do not edit */

#include "hal_data.h"

#if WATCHDOG_ON_WDT_CALLBACK_USED

#if defined(__ICCARM__)

#define NULL_WEAK_ATTRIBUTE

#pragma weak NULL = NULL_internal

#elif defined(__GNUC__)
```

```
#define NULL_WEAK_ATTRIBUTE __attribute__ ((weak,
alias("NULL_internal")))

#endif

void NULL(wdt_user_cb_data_t * p_args) NULL_WEAK_ATTRIBUTE;

#endif

static wdt_ctrl_t g_wdt_ctrl;

static const wdt_cfg_t g_wdt_cfg =

{

    .start_mode = WDT_START_MODE_REGISTER,

    .autostart = false,

    .timeout = WDT_TIMEOUT_16384,

    .clock_division = WDT_CLOCK_DIVISION_8192,

    .window_start = WDT_WINDOW_START_100,

    .window_end = WDT_WINDOW_END_0,

    .reset_control = WDT_RESET_CONTROL_RESET,

    .stop_control = WDT_STOP_CONTROL_DISABLE,

    .p_callback = NULL,

};

/* Instance structure to use this module. */

const wdt_instance_t g_wdt =

{

    .p_ctrl = &g_wdt_ctrl,

    .p_cfg = &g_wdt_cfg,

    .p_api = &g_wdt_on_wdt

};
```

```
#if WATCHDOG_ON_WDT_CALLBACK_USED

/* This is a weak example callback function. It should be */
/* overridden by defining a user callback function  */
/* with the prototype below.  */

/* - void NULL(wdt_user_cb_data_t * p_args) */

/* parameter p_args Callback arguments used to identify what caused the callback
. */

void NULL_internal(wdt_user_cb_data_t * p_args)

{

    /** Do nothing. */

}

#endif

const ioport_instance_t g_ioport =

{

    .p_api = &g_ioport_on_ioport,

    .p_cfg = NULL

};

const elc_instance_t g_elc =

{

    .p_api = &g_elc_on_elc,

    .p_cfg = NULL

};

void g_hal_init(void) {

}
```

hal_data.c contains g_wdt_ctrl which is the control structure for this instance of the WDT HAL driver. This structure should not be initialised as this is done by the driver when it is opened.

The contents of **g_wdt_cfg** are populated in this file using the **g_wdt WATCHDOG Driver on WDT Properties** pane in the **ISDE Project Configuration HAL** tab. If the contents of this structure do not reflect the settings made in the ISDE, ensure the **Project Configuration** settings are saved in the ISDE before pressing the **Generate Project Content** button.

ATTENTION: This file is regenerated each time Generate Project Content is pressed and so should not be edited.

3.3.4.3 WDT main.c

Contains main() called by the BSP start-up code. main() calls hal_entry() which contains user developed code (see next file). Here are the contents of main.c.

```
/* generated main source file - do not edit */

extern void hal_entry(void);

void main(void) {

hal_entry();

}
```

ATTENTION: This file is regenerated each time Generate Project Content is pressed and so should not be edited.

3.3.4.4 WDT hal_entry.c

This file contains the function hal_entry() called from main(). User developed code should be placed in this file and function.

For the WDT project edit the contents of this file to contain the code below. This code implements the flowchart in overview section of this document.

```
/* HAL-only entry function */

#include "hal_data.h"

#define RED_LED_NO_OF_FLASHES 30

#define RED_LED_PIN IOPORT_PORT_08_PIN_08

#define GREEN_LED_PIN IOPORT_PORT_08_PIN_07

#define RED_LED_DELAY_COUNT 1500000

#define GRN_LED_DELAY_COUNT 1200000

volatile uint32_t delay_counter;
```

```
volatile uint16_t loop_counter;

void hal_entry(void) {

    /* TODO: add your own code here */

    /* Open the WDT */

    g_wdt.p_api->open(g_wdt.p_ctrl, (wdt_cfg_t *const)g_wdt.p_cfg);

    /* Start the WDT by refreshing it */

    g_wdt.p_api->refresh(g_wdt.p_ctrl);

    /* Flash the red LED and tickle the WDT for a few seconds */

    for(loop_counter=0; loop_counter<RED_LED_NO_OF_FLASHES; loop_counter++)

    {

        /* Turn red LED on */

        g_ioport.p_api->pinWrite(RED_LED_PIN, IOPORT_LEVEL_HIGH);

        /* Delay */

        for(delay_counter=0; delay_counter<RED_LED_DELAY_COUNT; delay_counter++)
;

        /* Refresh WDT */

        g_wdt.p_api->refresh(g_wdt.p_ctrl);

        /* Turn red off */

        g_ioport.p_api->pinWrite(RED_LED_PIN, IOPORT_LEVEL_LOW);

        /* Delay */

        for(delay_counter=0; delay_counter<RED_LED_DELAY_COUNT; delay_counter++)
;

        /* Refresh WDT */

        g_wdt.p_api->refresh(g_wdt.p_ctrl);

    }
```

```
    /* Flash green LED but STOP tickling the WDT. WDT should reset the
    device */

    while(1)

    {

    /* Turn green LED on */

    g_ioport.p_api->pinWrite(GREEN_LED_PIN, IOPORT_LEVEL_HIGH);

    /* Delay */

    for(delay_counter=0; delay_counter<GRN_LED_DELAY_COUNT; delay_counter++);

    /* Turn green off */

    g_ioport.p_api->pinWrite(GREEN_LED_PIN, IOPORT_LEVEL_LOW);

    /* Delay */

    for(delay_counter=0; delay_counter<GRN_LED_DELAY_COUNT; delay_counter++);

    }

}
```

The WDT HAL driver is called through the interface **g_wdt_on_wdt** defined in **r_wdt.h**. The WDT HAL driver is opened through the open API call using the instance defined in **r_wdt_api.h**:

```
g_wdt.p_api->open(g_wdt.p_ctrl, (wdt_cfg_t *const)g_wdt.p_cfg);
```

The first passed parameter is the pointer to the control structure **g_wdt_ctrl** instantiated in hal_data.c. The second parameter is the pointer to the configuration data **g_wdt_cfg** instantiated in the same hal_data.c file.

The WDT is started and refreshed through the API call:

```
g_wdt.p_api->refresh(g_wdt.p_ctrl);
```

Again the first (and only in this case) parameter passed to this API is the pointer to the control structure of this instance of the driver.

3.3.5 Building and Testing the Project

Build the project in the ISDE **Build > Build Project**. The project should build without errors.

To debug the project

1) Connect the JLink debugger between the target board and host PC. Apply power to the board.
2) In the **Project Explorer** pane on the right side of the ISDE right-click on the WDT project **WDT_Application** and select **Debug As > Debug Configurations**.
3) Under **Renesas GDB Hardware Debugging** select **WDT_Application Debug** as shown below.

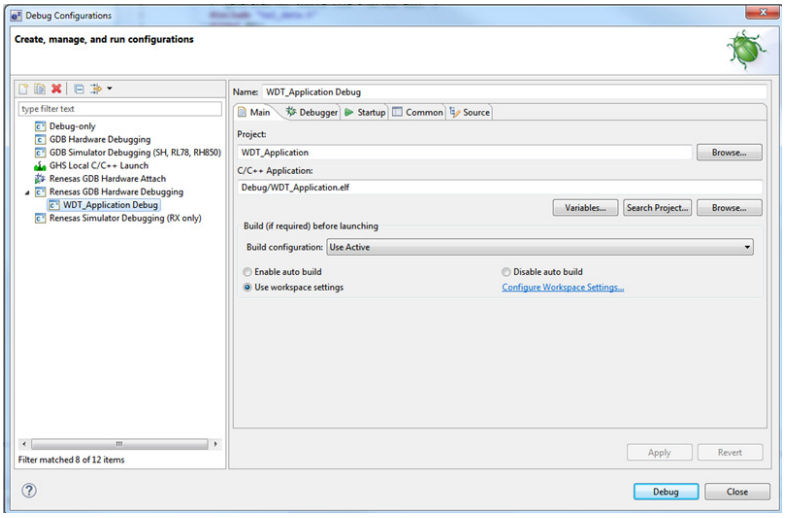


Figure 93: Debug configuration

4) Press the **Debug** button. Switch (Yes) to the Debug perspective if asked.
5) The code should run to the Reset_Handler() function.
6) Resume execution via **Run > Resume**. Execution will stop in main() at the call to **hal_entry()**.
7) Resume execution again.

The red LED should start flashing. After 30 flashes the green LED will start flashing and the red LED will stop flashing.

While the green LED is flashing the WDT will underflow and reset the device resulting in the red LED to flash again as the sequence repeats. However, this sequence does not occur when using the debugger because the WDT does not run when connected to the debugger.

1) Stop the debugger in the ISDE via **Run > Terminate**.
2) Press the reset button on the target board. The LEDs begin flashing.

3.4 IAR Embedded Workbench for Renesas

Synergy User Guide

3.4.1 Using IAR Embedded Workbench for Synergy

This section describes how to use the IAR Embedded Workbench for Renesas Synergy (IAR EW for Synergy) in combination with the Renesas Synergy Standalone Configurator (SSC) to develop applications with the Renesas Synergy Software Package (SSP). The architecture of the SSP directly determines how you use the IAR EW for Synergy and SSC to develop a Synergy application. See the following documents for details on the SSP architecture included in this manual:

- SSP Architecture
- BSP Architecture

3.4.2 What is IAR EW for Synergy?

IAR Embedded Workbench is now completely integrated with the Renesas Synergy Platform. The new product IAR EW for Synergy provides add-on functionality to simplify and accelerate software development, and provide the best performance and smallest code size.

Just like e^2 studio, IAR EW for Synergy offers secure source-level visibility into the Synergy Software Package (SSP) as well as secure source-level debugging. The developer can see protected source code but not modify or save it. Once the application code is developed, IAR EW for Synergy includes IAR C-STAT® and C-RUN® analyzers, tools which help and guide to improve application code quality.

3.4.3 Key Features

- Integrated development environment with project management tools and editor
- Highly optimizing C and C++ compiler and Linker for Renesas Synergy devices
- Integration support for Renesas Synergy Standalone Configurator (SSC)
- C-STAT and C-RUN code analysis tools included
- Extensive HW target system support
- Power debugging to visualize power consumption in correlation with source code
- C-SPY® Debugger with JTAG/SWD support and support for RTOS-aware debugging on hardware
- Support for ETM Trace
- Comprehensive user and reference guides and context-sensitive help function
- Compliant with ARM® Embedded Application Binary Interface (EABI) and ARM Cortex® Microcontroller Software Interface Standard (CMSIS)

For detailed instructions on how to download and install IAR EW for Synergy, see the IAR EW for Synergy Release Notes on the Synergy Gallery.

3.4.4 What is Synergy Standalone Configurator (SSC)?

The Synergy Standalone Configurator (SSC) is an Eclipse Rich Client Platform (RCP) application containing the Synergy Project Generator and the Synergy Project Editor as implemented in the Renesas e^2 studio ISDE. SSC includes configurators like the Clock Configurator, Pin Configurator, RTOS Configurator, SSP Module Selector/Configurator, and Interrupt Control Unit (ICU) Configurator for use with 3rd party IDEs such as IAR EW for Synergy.

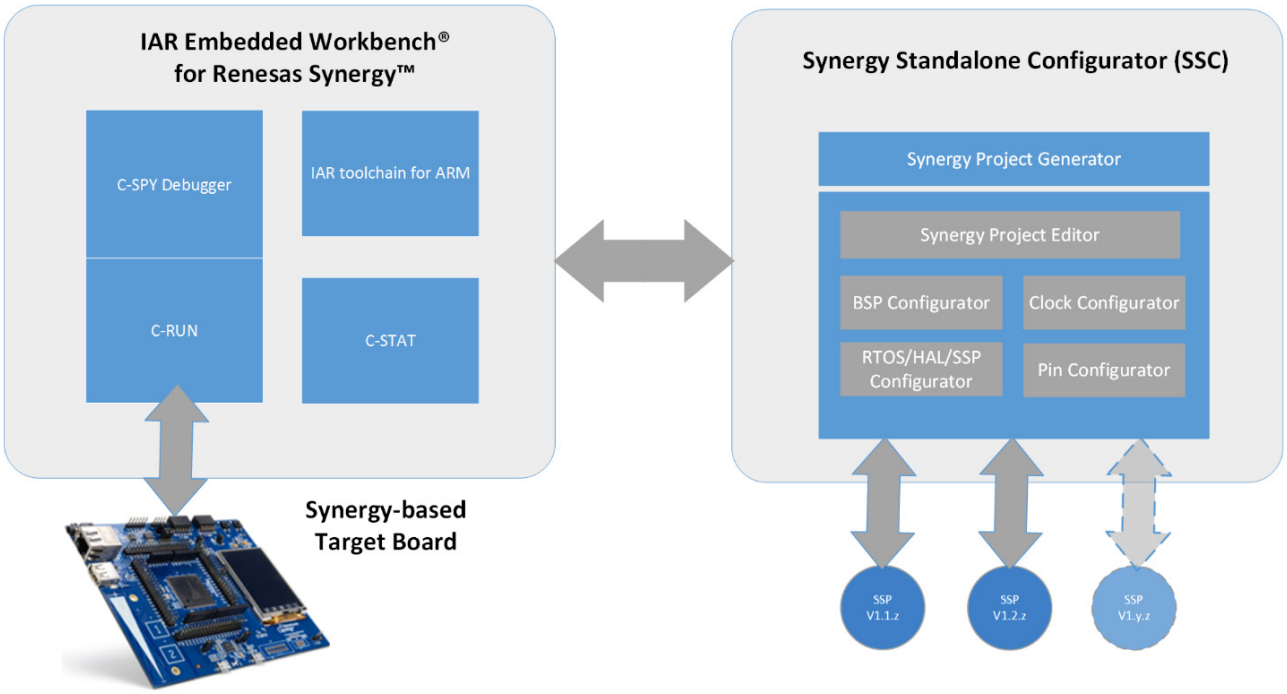


Figure 94: IAR EW for Synergy and SSC functional blockdiagram

Since the functionality of the SSC is identical to the Synergy Project Generator and the Synergy Project Editor as implemented in the Renesas e^2 studio ISDE, refer to e^2 studio ISDE User Guide for information on how to use it.

For detailed instructions on how to download and install the SSC and the SSP to use with IAR EW for Synergy, see the SSC Release Notes and the SSP Release notes on the Synergy Gallery.

3.4.5 Installing the Tools

To install the tools, follow the steps below:

1) Download and install the Renesas Synergy Standalone Configurator (SSC) from the Renesas Synergy Gallery. You can find it under Development Tools. The default installation directory is C:\Renesas\Synergy\SSC_<SSCversion>.
2) Download and install the Renesas Synergy Software Package (SSP) from the Renesas Synergy Gallery. During the installation you will be prompted to specify an installation directory for the SSP. Point the SSP installer to the directory where you just installed the SSC (for example C:\Renesas\Synergy\SSC_<SSCversion>).
3) Download and install IAR Embedded Workbench for Renesas Synergy from the Renesas Synergy Gallery. To install IAR Embedded Workbench:
    a) In your web browser, specify the URL https://synergygallery.renesas.com and download IAR Embedded Workbench for Renesas Synergy from the Renesas Synergy Gallery. You will also find information about how to obtain a license and get a license number.
    b) Execute the installer that is included in the downloaded file.

c) Specify the license number when prompted for in the IAR License Manager.

NOTE: The IAR EW for Synergy license entitles you to use this specific edition of IAR Embedded Workbench, but not the Synergy Standalone Configurator for which separate licenses are required.

3.4.6 Creating a Renesas Synergy Project using IAR EW for Synergy and SSC

To create a Synergy Project using IAR EW for Synergy and SSC, follow the steps below:

1) In the IAR Embedded Workbench IDE, choose **Project>Create New Project**.
2) In the **Create New Project** dialog box, select **Renesas Synergy Project** and click **OK**.

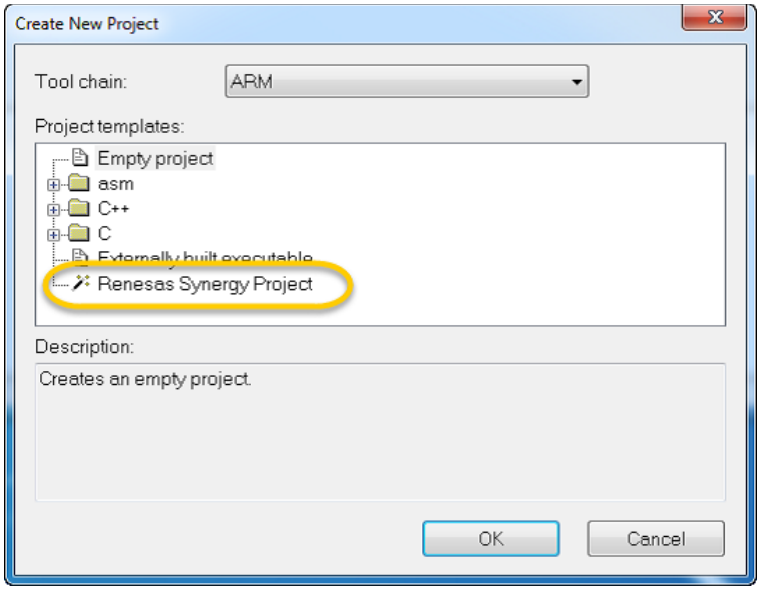


Figure 95: Creating a new Synergy Project using IAR EW for Synergy

3) In the **Save As** dialog box that appears, choose a suitable destination directory for your workspace (the container that holds your project), for example MyWorkspace, and click **Save**.

NOTE: Do not save your workspace in the root directory of your operating system (C:).

4) In the **Renesas Synergy Setting** dialog box that appears, specify:

- The location of your installed Synergy Standalone Configurator (SSC), which by default is installed in C:\Renesas\Synergy\SSC_<SSCversion>.

- The Synergy License File, which you can find in the SSC installation directory, e.g. C:\Renesas\Synergy\SSC_<SSCversion>\internal\projectgen\arm\Licenses.
- Whether or not you would like to replace any encrypted source files with decrypted ones. Note that a Renesas Synergy Source License is required for this function.

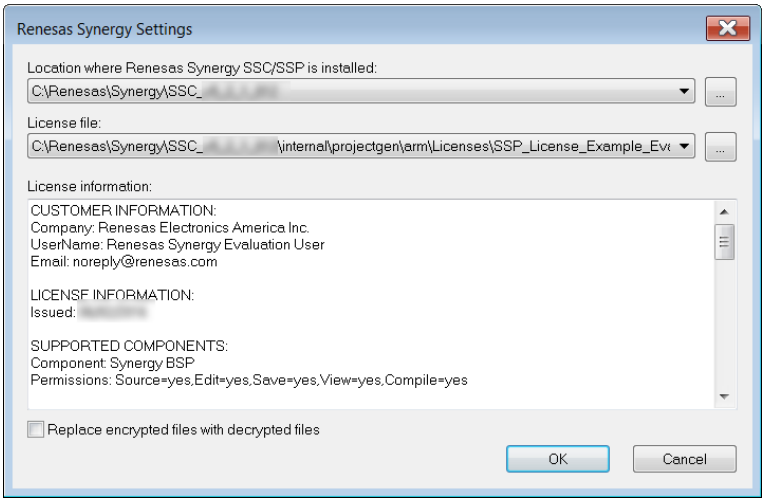


Figure 96: Renesas Synergy Settings in IAR EW for Synergy

5) Click **OK**.
6) In the **Save As** dialog box that appears, specify the name of your project, for example MyProject.

   NOTE: Do not save your project in the root directory of your operating system (C:).

7) The IAR Embedded Workbench IDE now connects with the Renesas Synergy Standalone Configurator (SSC). Specify the board support you require:

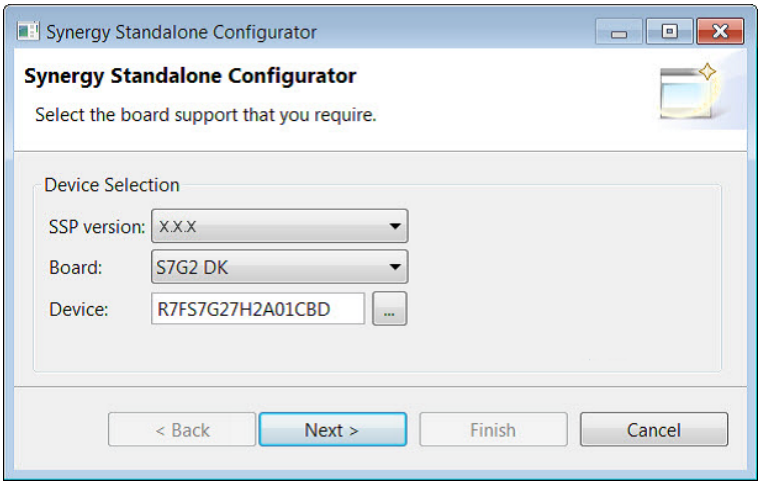


Figure 97: SSC selection dialog for SSP, board, and MCU to be

used"

NOTE: The SSP versions available in the drop-down list correspond to the versions you have previously installed on your computer.

Click **Next**.

8) The Synergy Software Packages come with several example projects, which include source code files, header files, and linker configuration files, adapted for your device. Select the example packages that you want to add to your project:

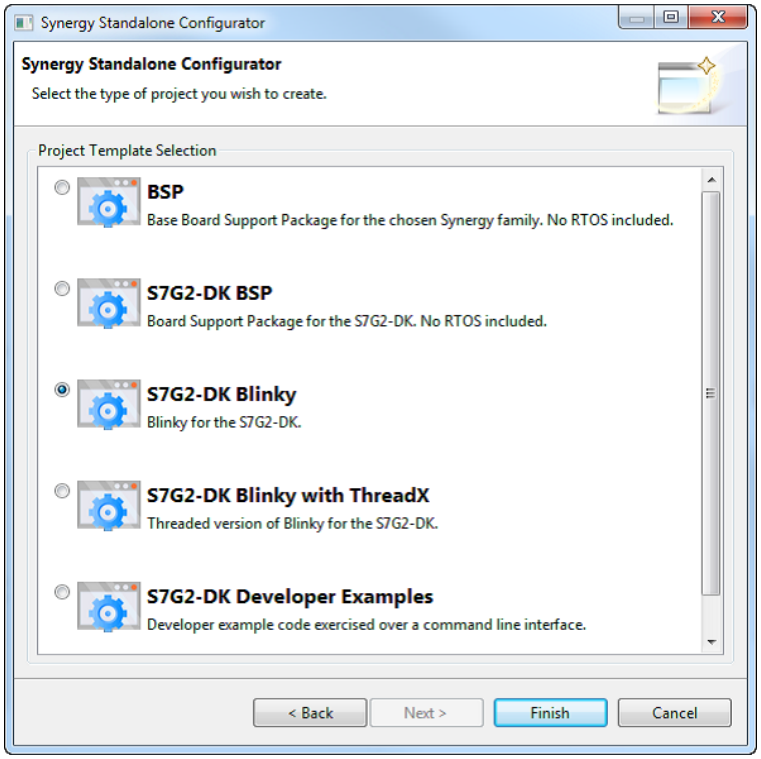


Figure 98: Project Template Selection dialog

Click **Finish**.

9) In the Synergy Project Editor that opens up, you can now configure MCU pin function assignments, clock and peripheral settings, and interrupt source assignments. When finished configuring, click the **Generate Project Content** button. The source code is now generated.

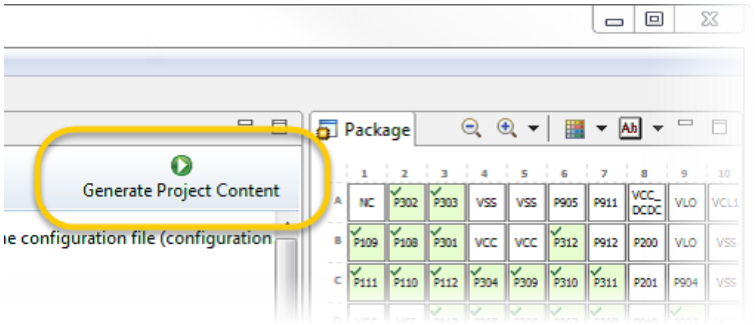


Figure 99: Generate Project Content button

NOTE: You can always add or change the configuration of your Synergy project later on.

10) After a couple of seconds, your Renesas Synergy project is displayed in the IAR EW Workspace window:

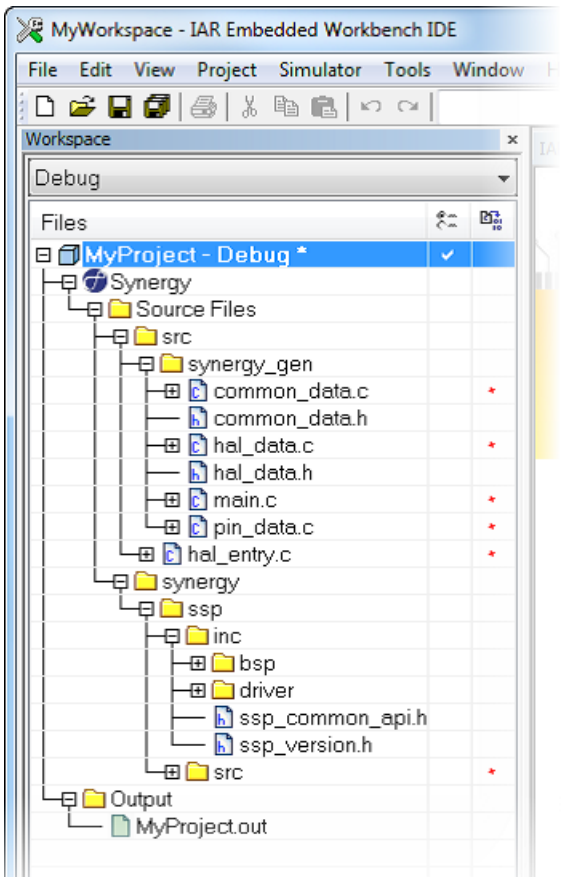


Figure 100:IAR EW for Synergy Workspace and Project window

11) If you close the SSC, you can re-open it again by clicking the Synergy Configuration button

in the toolbar, or by selecting **Renesas Synergy > Configurator** from the menu.

12) Whenever you switch back to the SSC to change configuration settings, click the **Generate Project Content** button when finished. The affected source code files are now re-generated.

13) You can now continue building and debugging according to the standard routines in the IAR Embedded Workbench IDE, see the IAR Embedded Workbench® IDE User Guide on the IAR web site.As the SSC works just like the Synergy Project Editor in e^2 studio, refer to e^2 studio ISDE User Guide for more details on how to use it.Note that the Synergy Project is by default configured for the J-Link debugging probe.If you have another debugging probe, for example I-jet or I-jet Trace, choose **Project>Options>Debugger** in the IDE and select I-jet/JTAGjet from the Driver drop-down list.

Chapter 4 Module Overviews

The Module Overviews for each SSP module have been significantly changed since the last release. The new Module Overviews should provide all the information necessary for a developer to evaluate a specific modules fitness for use in a target application and significantly help with the development process. The intent of these notes is to provide all the information needed to begin develop0ment with the target module in one easy to find location.

Each Module Overview includes the following sections:

1) A short introduction to the module includes a short description, a block diagram of key module components, and a list of features
2) An API table lists all the available APIs, and example use of the API and a short description of the APIs function. A list of some of the key Status Return values is provided to help determine the result of the API call.
3) A functional overview describes key module operations and includes a list of important module limitations.
4) A step-by-step description of how to include the module in an application using the ISDE threads tab and stack selection process.
5) A set of tables showing the configuration parameters for the module and key lower level modules is provided so the developer can easily see the modules key capabilities. Note that these configurable properties vary by MCU series and by SSP Release. Treat the tables in these notes as illustrations and refer to the actual parameters available within the ISDE for your target MCU and for your chosen SSP release. Example pin and clock configuration and selection information is also provided to help guide development.
6) A simple implementation using the target module is provided and shows the steps used in a typical application, the associated flow diagram, and the API use at each step. This helps describe how APIs are commonly used and will give the developer a head start with their implementation.

Module Guide Application Notes

These six sections are also found in the Module Guide Application Notes for a specific module. In the Application Note additional sections provide detailed descriptions of the associated application project that demonstrates the module working in an actual design. Development can be dramatically simplified when the application project is used as a starting point or reference for a new design. Module guide application notes can be found on the Renesas Synergy Tools & Kits page under the sample code tab https://www.renesas.com/en-us/products/synergy/tools-kits.html#sampleCodes More module guide application notes are being added all the time so check back frequently to find when new ones have been released.

Using the Module Guide Module Overviews

The Module Overviews provide sufficient details to begin development, but there will be cases when additional information is useful in implementing a design. The SSP User Manual provides a wealth of information on the details of API implementation, structures, enumerations and more. Simply jump to the API reference section and find your module of interest to find any additional information you might need. Here are some examples for common items and where you would find them:

- API Framework Function Details- API Reference: Framework Interfaces Example- ADC Periodic Framework, chapter 6.2.1.2 provides a list of available APIs. Click on the API in the table to see the API template, description, parameters and instance structures.
- API Framework Function Return Values- API Reference: Framework Example- ADC Periodic Framework, chapter 7.2.1 provides details on defines, return values, and the internal function steps the framework uses. HAL modules have chapters that cover the above topics as well. It is highly recommended that you spend some time looking at the reference material available in ALL the reference chapters so you know where to look when an API

implementation question, not answered in the module guide usage note or associated module guide application note, comes up.

You can find a list of Module Overviews on the following pages:

- Framework Layer
- HAL Layer
- Express Logic Modules

4.1 Framework Layer

ADC Periodic Framework
Audio Playback Framework
Audio Playback DAC Framework
Audio Playback I2S Framework
Audio Record ADC Framework
Audio Record I2S Framework
Block Media QSPI Framework
Block Media RAM Framework
BLE Framework
Cellular Framework
Communications Framework
Communications Framework on NX
Communications Framework on USBX™
Console Framework
Crypto Framework
Capacitive Touch Framework
Capacitive Touch Button Framework
Capacitive Touch Slider Framework
External IRQ Framework
I2C Framework
JPEG Decode Framework
Messaging Framework
Power Profiles Framework
Power Profiles V2 Framework
SPI Framework
Thread Monitor Framework
Touch Panel Framework

UART Communications Framework

Wi-Fi Framework

4.1.1 ADC Periodic Framework

The ADC Periodic Framework provides high-level APIs and is implemented on sf_adc_periodic. The module configures the ADC to sample any of the available channels (using the single-scan mode) at a configurable rate and buffers the data for a configurable number of sampling iterations before notifying the application. The ADC Periodic Framework uses the ADC, GPT or AGT, and DTC peripherals on the Renesas Synergy™ Microcontroller hardware. A user-defined callback can be created to process the data each time a new sample is complete.

4.1.1.1 ADC Periodic Framework Module Features

- 14-Bit A/D Converter (S3A7, S124) or 12-Bit A/D Converter (S7G2, S5D9) for the Synergy MCU Group
- Multiple Operation Modes
- Single Scan
- Group Scan
- Continuous Scan
- Multiple Channels
- 13 channels (unit 0), 12 channels (unit 1) for S7G2 and S5D9 Synergy MCU Groups
- 18 channels for S124
- 28 channels for S3A7

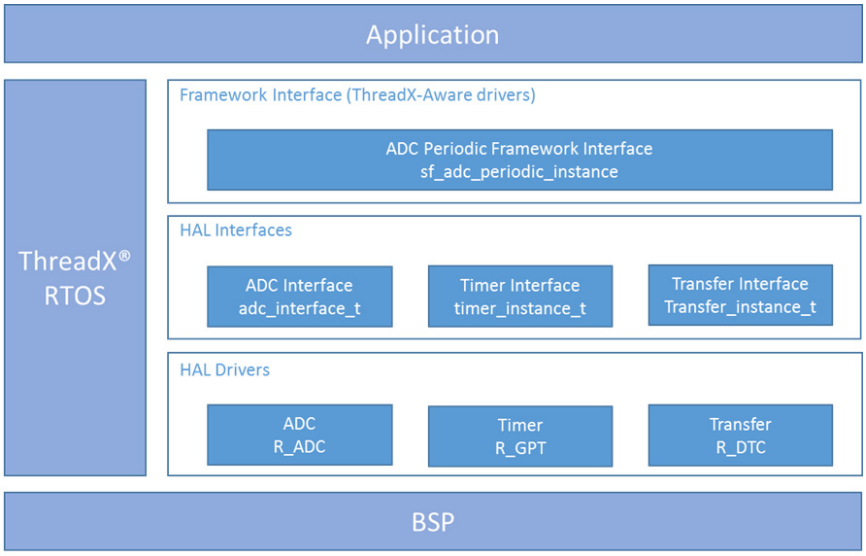


Figure 101: ADC Periodic Framework Module Block Diagram

4.1.1.2 ADC Periodic Framework Module APIs Overview

The ADC Periodic Framework defines APIs for opening, closing, starting, and stopping the ADC scans. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

ADC Periodic Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_adc_periodic.p_api->open(g_sf_adc_periodic.p_ctrl, g_sf_adc_periodic.p_cfg); Acquires mutex, then initializes module at the HAL layer.
start	g_sf_adc_periodic.p_api->start(g_sf_adc_periodic.p_ctrl); Starts the scan.
stop	g_sf_adc_periodic.p_api->stop(g_sf_adc_periodic.p_ctrl); Stops the hardware trigger (timer) from triggering any more ADC scans.
close	g_sf_adc_periodic.p_api->close(g_sf_adc_periodic.p_ctrl); Releases channel mutex and closes channel at HAL layer.
versionGet	g_sf_adc_periodic.p_api->versionGet(&version); Retrieve the API version using the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the *SSP User's Manual* API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful

Name	Description
SSP_ERR_UNSUPPORTED	Command not found in the current menu
SSP_ERR_NOT_OPEN	Driver control block not valid. Call SF_ADC_PERIODIC_Open to configure.
SSP_ERR_ASSERTION	Version get error- p_version was NULL
SSP_ERR_INTERNAL	An internal ThreadX® error has occurred. This is typically a failure to create/use a mutex or to create an internal thread.

NOTE: Lower-level drivers may return common error codes. Refer to the *SSP User's Manual* API References for the associated module for a definition of all relevant status return values.

4.1.1.3 ADC Periodic Framework Module Operational Overview

The ADC Periodic Framework module samples and buffers ADC data. The Framework notifies the application once the configured number of samples are buffered. The ADC Periodic Framework works as follows:

- After initial configuration and the scan process is started, the framework uses a hardware timer to trigger an ADC scan in one-shot mode. Each scan can consist of one or more channels. When each scan is completed, the ADC interrupt is intercepted by the DTC, which moves the result of the scan into the user buffer.
- Each scan is defined as a sampling iteration, and the number of samples generated for each scan is equal to the number of channels. If the channels are sequential (for example, channels 1, 2, 3, 4), the data is captured in order. If the channels are not in sequence (for example, channels 1, 3, 4, 5), then the samples generated by each scan also include data from the unused channels in between. Therefore, in the second example, five samples are stored to the user buffer each time.
- The user specifies the total number of sample iterations that need to occur before being notified. When the specified number of sampling iterations have occurred, and the data for each iteration has been stored into the user buffer, the user is notified via a callback with an index for the valid data in the buffer, and an event indicating that sampling for the specified number of iterations is complete.
- Unless the user stops the scan process, the scan continues to be triggered by the timer (using AGT or GPT), and data will be written into the user buffer, which is treated by the Framework as a circular buffer. The name and length of the buffer are specified via the ISDE configurator.

ADC Periodic Framework Module Operational Notes

1) At least one channel must be chosen while configuring the ADC HAL module to avoid an API return error.
2) When configuring the scan rate for the ADC Periodic Framework (the GPT or AGT timer period), make sure that the period is long enough to accommodate scanning of all selected channels (about 2 microseconds for each channel conversion on a Synergy S7G2 MCU Group device).
3) The ADC Periodic Framework stores data for all the channels from each scan into the user-specified buffer. When the specified number of sample iterations is completed, the user is notified. If five channels are selected (channels

1,2,3,4,5) and the sample count is set to 3, the user will be notified when (5 x 3 =) 15 samples are available. The samples are ordered as follows:

Figure 102: ADC Periodic Operation – Sample Order

When selecting the data buffer length in the ADC Periodic Framework configuration, make sure that the buffer length is at least twice the length of the number of samples that will be generated (15 x 2 = 30 in this example). This is because once the user application is notified that the data is available, the framework keeps buffering in new data at the sample rate. Since the buffer is treated as a circular buffer, you can inadvertently overwrite the data. If the size is not larger than the number of samples generated, the data is overwritten before the application can use it.

The application callback has an index into the appropriate location in the buffer where valid data is present.

ADC Periodic Framework Module Limitations

The ADC Periodic Framework does not currently support the following features:

- The use of group-scan mode
- The use of DMA
- When configuring the ADC channels to be used with this framework, the temperature and voltage sensors must not be selected if any of the other available channels are also selected. It is possible to use only the temperature sensor, only the voltage sensor, or any number of the regular ADC channels.
- Refer to the most recent *SSP Release Notes* for any additional operational limitations for this module.

4.1.1.4 Including the ADC Periodic Framework Module in an Application

This section describes how to include the ADC Periodic Framework module in an application using the SSP configurator.

NOTE: It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the *Getting Started Guide for SSP* given in the References section at the end of this document to learn how to manage each of these important steps in creating SSP-based applications.

To add the ADC Periodic Framework to your application, simply add it to a project thread using the stacks selection sequence given in the following table. (The default name for the ADC Periodic Framework is g_sf_adc_periodic0. This name can be changed in the associated **Properties** window.)

ADC Periodic Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_adc_periodic0 ADC Periopdic Framework	**Threads**	**New Stack> Framework> Analog> ADC Periodic Framework on sf_sdc_periodic**

When the ADC Periodic Framework on sf_adc_periodic is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information will have box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

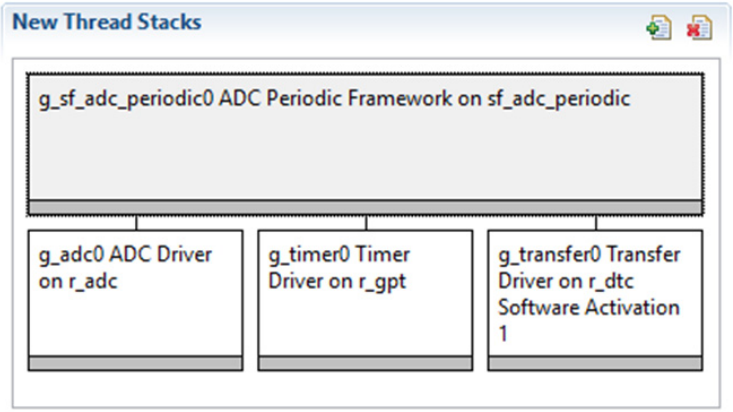


Figure 103: ADC Periodic Framework Module Stack

4.1.1.5 Configuring the ADC Periodic Framework Module

The ADC Periodic Framework module must be configured by you for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only those properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes, and are identified with a **lock** icon for the locked property in the **Properties** window within the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the **Properties** tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority. This configuration setting is available within the **Properties** window of the associated module. Simply select the indicated module and then view the **Properties** window. The interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Note that the interrupt priorities listed in the **Properties** window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE and create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the ADC Periodic Framework Module on sf_adc_periodic

Parameter	Value	Description
Parameter Checking	Enabled, Disabled, BSP (Default: BSP)	Selects if code for parameter checking is to be included in the build

Parameter	Value	Description
Name	Default: g_sf_adc0_periodic0	ADC Periodic Framework module name
Name of the data-buffer to store samples	Default: g_user_buffer	Name of the 16-bit data buffer to store samples
Length of the data-buffer	Default: 128	Length of the buffer to which data is to be stored
Number of sampling iterations	Default: 10	Number of samples captured per iteration
GPT Timer channel used to trigger the scan	Default: Channel 0	Channel used to generate the ELC event entry scan_trigger
Callback	Default: g_adc_framework_user_callback	User function that will be called once the number of sampling iterations of data has been buffered

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select a different ADC unit or enable the averaging function. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

NOTE: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated **Properties** window from the SSP configurator.

The ADC HAL module (r_adc) is used to configure the ADC unit. Most of the default properties are acceptable. The interrupt must be enabled and a name must be assigned to the module. This name is used by the application code to reference the unit.

Configuration Settings for the ADC HAL Module

Configuration Settings for the ADC HAL Module on r_adc

ISDE Property	Value	Description
Parameter Checking	Enabled, Disabled, BSP (Default: BSP)	Selects if code for parameter checking is to be included in the build
Name	Default: g_adc0	Module Name
Unit	Default: 0 (1 for S7G2 MCU)	Specify the ADC Unit to be used. The S7G2 MCU has two units, 0 and 1

ISDE Property	Value	Description
Resolution	8-bit, 10-bit, 12-bit, 14-bit (Select as per the board)	Specify the conversion resolution for this unit
Alignment	Right, Left (Default: Right)	Specify the conversion result alignment
Clear After read	On, Off (Default: On)	Specify if the result register must be automatically cleared after the conversion result is read
Mode	Single Scan, Continuous scan, group scan (Default: Single Scan)	Specify the mode that this ADC unit is used in
Channel Scan Mask Channel 0-27, Temperature sensor, Voltage Sensor	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example, if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Normal/Group A trigger	None, Asynchronous external trigger, ELC event, Software	Specify the trigger type to be used for this unit. If group mode is used mode, then this field is used to set the Group A trigger. NOTE: The only valid option in group mode is the ELC trigger.
Group B trigger	Default: ELC event	Specify the group B trigger. This option is only valid if group mode is chosen in mode.
Add Average Count	Disabled, Enabled Default: Disabled	Specify if addition or averaging needs to be done for any of the channels in this unit. The actual channels are specified by using a channel mask add_mask.

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have

different default values and available configuration settings.

ADC HAL Module Configuration Settings continued

ISDE Property	Value	Description
Group Priority	Default: Group A cannot interrupt Group B, Group A can interrupt group B; Group B scan restarts at next trigger, Group A cannot interrupt Group B, Group A can interrupt group B; Group B scan restarts immediately Group A cannot interrupt Group B, Group A can interrupt group B; Group B scan restarts and scans continuously	Determines if an ongoing group B scan can be interrupted by a group A trigger, if it should abort on a group A trigger, or if it should pause to allow group A scan and restart immediately after group A scan is complete. NOTE: This field is valid only in group mode.
Addition/ Average mask Channel 0-27, Temperature sensor, Voltage Sensor	Disabled, Enabled Default: Disabled	This field is valid only if add_average_count is enabled. This filed determines what channels results are to be averaged or summed.
Sample Hold Mask Channel 0-2	Disabled, Enabled Default: Disabled	Determines which of channels 0, 1 and 2 are using the updated sample-and-hold states value specified in sample_hold_states. This field must only be set if it is desired to modify the default sample and hold count value for channels 0, 1 and 2.

ISDE Property	Value	Description
Sample Hold States	4-255	Specifies the updated sample-and-hold count for the channel dedicated sample-and-hold circuit. This field is valid only if sample_hold_mask is not 0. Only channels 0, 1 and 2 have dedicated sample and hold circuits. NOTE: Use this to modify the default number of states (24) for which the value is sampled. Each state is equal to 1/ADCLK time.
Callback	NULL	The ADC Framework preconfigures and locks this field. User should override by defining a user call back function with prototype sf_adc_periodic_cb
Scan End Interrupt Priority	Disabled, Priority 0-15 (Default: BSP)	Specify interrupt to enable adc module
Scan End Group B interrupt priority.	Disabled, Priority 0-15 (Default: BSP)	Specify interrupt to enable adc module in group B

Configuration Settings for the GPT HAL Module

The configuration parameters for the GPT Timer Driver used with the ADC Periodic Framework module can often be used with the default settings. Typically, only a small number of settings need to be modified from the default settings. Note that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table lists the options that must be set to configure the GPT HAL module.

Configuration Settings for the GPT HAL Module

ISDE Property	Value	Description
Parameter Checking	Enabled, Disabled, BSP (Default: BSP)	Selects if code for parameter checking is to be included in the build

ISDE Property	Value	Description
GPT0 COUNTER OVERFLOW	Disabled, Priority 0-15 (Default: BSP)	Selects if code for parameter checking is to be included in the build
Name	g_timer0	Module name
Channel	0	The ADC Framework preconfigures and locks this field.
Mode	Periodic	The ADC Framework preconfigures and locks this field.
Period Value	Default: 10	Configure Timer Period to trigger ADC scans
Period Unit	Raw Counts, Nanoseconds, Microseconds, Milliseconds, Seconds, Hertz, Kilohertz (Default: Milliseconds)	Units for the Period setting
Duty Cycle Value	Default: 50	Value for the duty cycle setting
Duty Cycle Unit	Raw count, unit %, Unit % x 1000 (Default: Raw Counts)	Units used for duty cycle value
Autostart	False	The ADC Framework preconfigures and locks this field.
GTIOCA Output Enabled	True, False (Default: False)	Output is enabled or disabled by this setting
GTIOCA Stop Level	Pin level High, Low or Retained (Default: Low)	Determines output pin level after a transition
GTIOCB Output Enabled	True, False (Default: False)	Output is enabled or disabled by this setting
GTIOCB Stop Level	Pin level High, Low or Retained (Default: Low)	Determines output pin level after a transition

ISDE Property	Value	Description
Callback	NULL	The ADC Framework preconfigures and locks this field.
Interrupt Priority	Priority 0:15	Interrupt Priority

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

DTC HAL Module Configuration Settings

The DTC HAL module is configured internally by the ADC Periodic framework. Therefore, no user configuration is required (other than providing the module name). Refer to the associated property settings within the SSP configuration window for the details.

ADC Periodic Framework Module Pin Configuration

To access a channel, ADC channels must be set in the **Pins** tab of the ISDE. The following table illustrates the method for selecting the pins within the SSP configuration window.

Pin Selection for the ADC HAL Module

Resource	ISDE Tab	Pin selection Sequence
ADC	**Pins**	Select **Peripherals > Analog ADC > ADC0\1 > AN_XX**

NOTE: In the cases of the internal temperature sensor and the internal voltage sensor, there are no pin configurations required.

4.1.1.6 Using the ADC Periodic Framework Module in an Application

The key elements in constructing a simple ADC Periodic Framework module are selecting and configuring the stack, selecting a timer (GPT or AGT), defining the body of the callback function p_callback, initializing the framework, scanning the defined ADC channels, operating on the captured data as required by the application, rescanning periodically, and then stopping if the ADC measurements have been completed. The typical steps in using the ADC Periodic Framework module in an application are:

1) Initialize the ADC using the open API
2) Start the scan of channels using the start API
3) Stop the scan with the stop API
4) Read the results of the conversion using the read API
5) Close the instance using the close API

Often the scan is repeated periodically or stopped between scan operations. These common steps are illustrated in a typical operational flow diagram in the following figure:

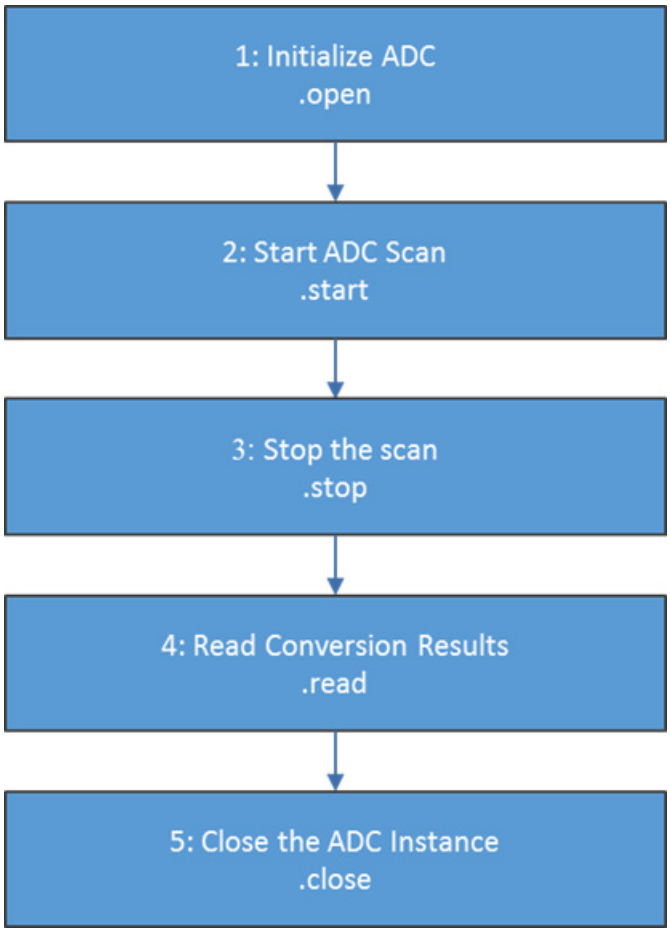


Figure 104: ADC Periodic Framework Module Application Project Flow Diagram

4.1.2 Audio Record ADC Framework

The Audio Record ADC Framework module provides high-level API for audio recording applications and is implemented on sf_adc_periodic. The Audio Record ADC Framework module uses the sf_adc_periodic and its lower layer ADC, GPT and DTC peripherals on the Synergy MCU. A user-defined callback can be created to indicate that the sample count has been completed.

4.1.2.1 Audio Record ADC Framework Module Features

- Records data in 8- or 12-bit PCM
- Uses ADC Periodic Framework to simplify configuration and integration
- Uses ThreadX object, like mutex, to protect hardware from improper access

- APIs for high-level functions simplify coding:
  - open, start
  - stop, infoGet
  - close

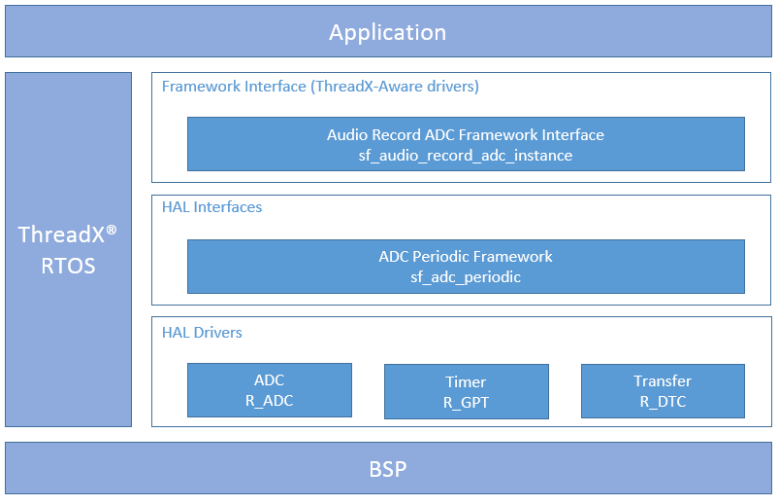


Figure 105: Audio Record ADC Framework Module Organization, Options and Stack Implementations

4.1.2.2 Audio Record ADC Framework Module APIs Overview

The Audio Record ADC Framework module defines APIs for opening, closing, starting, and stopping the record process. A complete list of the available APIs, an example API call, and a short description of each can be found in the table below. A table of status return values follows.

Audio Record ADC Framework Module API Summary

Function Name	Example API Call and Description
.open	g_sf_audio_record_adc.p_api->open(g_sf_audio_record_adc.p_ctrl, g_sf_audio_record_adc.p_cfg); Initialize the module.
.start	g_sf_audio_record_adc.p_api->start(g_sf_audio_record_adc.p_ctrl); Start audio recording.

Function Name	Example API Call and Description
.stop	g_sf_audio_record_adc.p_api->stop(g_sf_audio_record_adc.p_ctrl); Stop audio recording.
.infoGet	g_sf_audio_record_adc.p_api->infoGet(g_sf_audio_record_adc.p_api.*p_ctrl*); Get the channel information (mono or Stereo).
.close	g_sf_audio_record_adc.p_api->close(g_sf_audio_record_adc.p_ctrl); Close the module.
.versionGet	g_sf_audio_record_adc.p_api->versionGet(&version); Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manuals API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_IN_USE	The adc periodic framework mutex may be unavailable for the unit requested. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use a mutex or to create an internal thread.
SSP_ERR_NOT_OPEN	Unit is not open
SSP_ERR_ASSERTION	The parameter p_ctrl or p_sample is NULL.

Name	Description
SSP_ERR_UNSUPPORTED	This function is not supported by the HAL driver (p_ctrl->p_api->close is NULL).

NOTE: Lower level drivers may return Common Error Codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.2.3 Audio Record ADC Framework Module Operational Overview

The Audio Record ADC Framework module samples audio analog data using the ADC Periodic Framework and the data samples captured are stored in the user buffer. The data is made available for further processing as needed by the application. The Audio Record ADC Framework has a configuration parameter that is initialized during the framework initialization, which also initializes the underlying ADC periodic framework for data capture.

The captured data is stored in a user defined buffer and this is done in the callback function as illustrated below:

Assuming the name of the callback has been configured to be sf_audio_record_user_callback.

```
uint16_t * audio_record_buffer;
void sf_audio_record_user_callback (sf_audio_record_callback_args_t *p_args)
{
audio_record_buffer = ((uint16_t *)g_sf_audio_record_adc.p_cfg->
    p_capture_data_buffer + (p_args->buffer_index/2)); }
```

Audio Record ADC Framework Module Important Operational Notes and Limitations

- The Audio Record ADC Framework Module configuration data can specify the length of the data buffer, data width, sampling rate and the number of sampling iterations.
- Currently the Audio Record ADC only supports the ADC Periodic Framework as the lower level and thus recording via I2S is not supported with the framework.
- The framework currently supports 12-bit ADCs (supports 8, 10, and 12 bits) and 14-bit ADCs (supports 14- or 12-bit PCM data).
- Currently the Audio record ADC only supports Mono Channel.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.2.4 Including the Audio Record ADC Framework Module in an Application

This section describes how to include the Audio Record ADC Framework Module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP based applications.

To add the Audio Record ADC framework module to an application, simply add it to a thread using the Stacks Selection Sequence given in the table below. (The default name for the Audio Record ADC Framework Module is g_audio_record_adc0. This name can be changed in the associated Properties window.)

Audio Record ADC Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_audio_record_adc0 Audio Record ADC Framework on sf_audio_record_adc	Threads	New Stack> Driver> Audio> Audio Record ADC Framework on sf_audio_record_adc

When the Audio Record ADC on sf_audio_record_adc is added to the Thread Stack as shown in the figure below, the configurator automatically adds any needed lower level drivers.

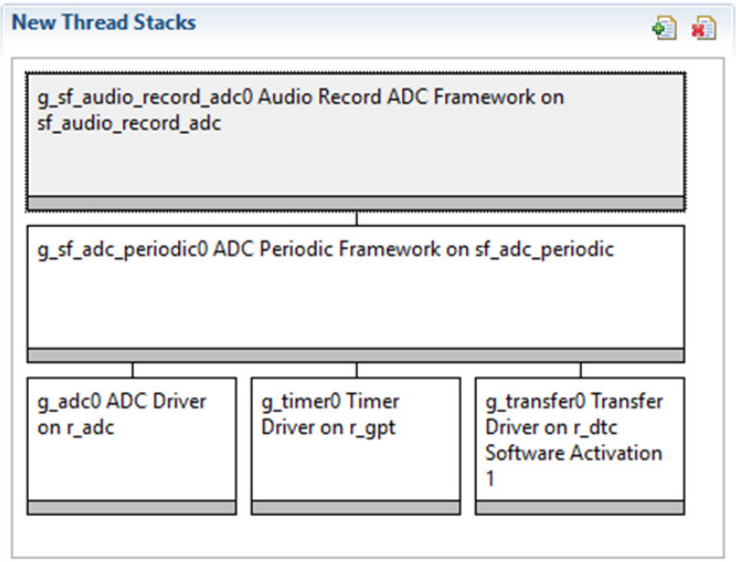


Figure 106: Audio Record ADC Framework Module Stack

4.1.2.5 Configuring the Audio Record ADC Framework Module

The Audio Record ADC Framework Module must be configured by you for the desired operation. The SSP configuration window will automatically identify, by highlighting the block in red, any required configuration selections, such as Interrupts or operating modes, which must be configured for lower level modules, for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP Configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the Interrupt Priority. This configuration setting is available with the Properties window of the associated module. Simply select the indicated module and then view the properties window. The Interrupt settings are often toward the bottom of the properties list, so scroll down until they

become available. Also note that the Interrupt Priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the below configuration properties tables, but is easily visible with the ISDE when configuring Interrupt Priority levels.

NOTE: You may want to open your ISDE and create the module and explore the property settings in parallel with looking over the Configuration Table Settings given below. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for Audio Record ADC Framework Module on sf_audio_record_adc

ISDE Property	Value	Description
Parameter Checking	Enabled, Disabled, BSP Default: BSP	Selects if code for parameter checking is to be included in the build
Name	g_sf_audio_record_adc0	Module name
Name of the data-buffer to store samples	P_capture_data_buffer	Name of the data buffer to store samples
Length of the data-buffer	2048	Length of the buffer to which data is to be stored
Audio Record Data Size	8-bit, 16-bit Default: 8-bit	Data width
Sampling Rate in HZ	8000	Sampling rate
Number of sampling iterations	Default: 256	Number of samples captured per iteration
Callback	g_audio_redord_framework_user_callback	User function that will be called once the number of sampling iterations of data has been buffered.

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for stack modules can be desirable. For example, it might be useful to select different buffer sizes or sample rates. The configurable properties for the lower level stack modules are given in the below sections for completeness and as a reference.

NOTE: Most of the property settings for modules are fairly intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the Audio Record ADC Framework Module Low Level Drivers

Typically, only a small number of settings must be modified from the default for lower level drivers and these are indicated via the red text in the Thread Stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The tables below identify all the settings within the properties section for the module.

Configuration Settings for ADC Periodic Framework on sf_adc_periodic

ISDE Property	Value	Description
Parameter Checking	Enabled, Disabled, BSP Default: BSP	Selects if code for parameter checking is to be included in the build
Name	g_sf_adc_periodic0	Module name
Name of the data-buffer to store samples	g_user_buffer	Name of the data buffer to store samples
Length of the data-buffer	2048	Length of the buffer to which data is to be stored
Number of sampling iterations	256	Number of samples captured per iteration
GPT Timer channel used to trigger scan	Channel 0-12	Channel number
Callback	g_audio_redord_framework_user_callback	User function that will be called once the number of sampling iterations of data has been buffered.

Configuration Settings for the ADC HAL Module on r_adc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: Enabled	If selected code for parameter checking is included in the build.
Name	g_adc0	Module name
Unit	0, 1 (S7G2 Only) Default: 0	Specify the ADC Unit to be used. The S7G2 has two units; 0 and 1.

ISDE Property	Value	Description
Resolution	14-Bit (S3A7/S124 Only), 12-Bit, 10-Bit (S7G2 Only), 8-Bit (S7G2 Only) Default: 8-Bit (S7G2 Only)	Specify the conversion resolution for this unit.
Alignment	Right, Left Default: Right	Specify the conversion result alignment.
Clear after read	Off, On Default: On	Specify if the result register must be automatically cleared after the conversion result is read. NOTE: If this is enabled, then watching the result register using a debugger always results in a 0.
Mode	Single Scan	The ADC Framework preconfigures and locks this field.
Channels 0-6	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channels 7-10 (S3A7/S124 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.

ISDE Property	Value	Description
Channels 11-15 (S3A7 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channels 16-20	Unused, Use in Normal/Group A, Use in Group B (Default: Unused)	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channel 21 (Unit 0 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channel 22 (S3A7/S124 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channels 23-27 (S3A7 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.

ISDE Property	Value	Description
Temperature Sensor	Unused, Use in Normal/Group A, Use in Group B Default: Unused	Temperature sensor use selection for Channel Scan Mask
Voltage Sensor	Unused, Use in Normal/Group A, Use in Group B Default: Unused	Voltage sensor use selection for Channel Scan Mask
Scan Mask Group B	Use #define ADC_MASK_xxx which are defined in r_adc.h. Use (ADC_MASK_xxx / ADC_MASK_xxx) for multiple channels.	Do not use with ADC Framework since the mode is locked to Single Scan Mode.
Normal/Group A Trigger	ELC Event	The ADC Framework preconfigures and locks this field.
Group B Trigger (Valid Only in Group Scan Mode)	ELC Event (The only valid trigger for either group in Group Scan Mode)	The ADC Framework preconfigures and locks this field.
Group Priority (Valid only in Group Scan Mode)	Group A cannot interrupt Group B, Group A can interrupt Group B; Group B scan restarts at next trigger, Group A can interrupt Group B; Group B scan restarts immediately, Group A can interrupt Group B; Group B scan restarts immediately and scans continuously (Default: Group A cannot interrupt Group B)	Do not use with ADC Framework since the mode is locked to Single Scan Mode.
Add/Average Count	Disabled, Add two samples, Add three samples, Add four samples, Add sixteen samples, Average two samples, Average four samples Default: Disabled	Specify if addition or averaging needs to be done for any of the channels in this unit. The actual channels are specified by using a channel mask add_mask.
Channels 0-27	Disabled, Enabled Default: Disabled	This field is valid only if add_average_count is enabled. This field determines what channels results are to be averaged or summed.

ISDE Property	Value	Description
Temperature Sensor	Disabled, Enabled Default: Disabled	Temperature sensor use selection for Addition/Averaging Mask
Voltage Sensor	Disabled, Enabled Default: Disabled	Voltage sensor use selection for Addition/Averaging Mask
Channels 0-2	Disabled, Enabled Default: Disabled	Determines which of channels 0, 1 and 2 are using the updated sample-and-hold states value specified in sample_hold_states. This field must only be set if it is desired to modify the default sample and hold count value for channels 0, 1 and 2.
Sample Hold States (Applies only to the 3 channels selected above)	24	Specifies the updated sample-and-hold count for the channel dedicated sample-and-hold circuit. This field is valid only if sample_hold_mask is not 0. Only channels 0, 1 and 2 have dedicated sample and hold circuits. NOTE: Use this to modify the default number of states (24) for which the value is sampled. Each state is equal to 1/ADCLK time.
Callback	NULL	The ADC Framework uses the callback internally.
Scan End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Scan End Interrupt Priority selection

ISDE Property	Value	Description
Scan End Group B Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Scan End Group B Interrupt Priority selection

Configuration for the GPT HAL Module on r_gpt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_timer0	Module name.
Channel	0	The ADC Framework preconfigures and locks this field based on channel selected in the ADC Framework.
Mode	Periodic	The ADC Framework preconfigures and locks this field.
Period Value	10	Configure timer period to trigger ADC scans.
Period Unit	Raw Counts, Nanoseconds, Microseconds, Milliseconds, Seconds, Hertz, Kilohertz Default: Milliseconds	Configure units of the timer period set above.
Duty Cycle Value	50	Duty cycle value selection
Duty Cycle Unit	Unit Raw Counts, Unit Percent, Unit Percent x 1000 Default: Unit Raw Counts	Duty cycle unit selection

ISDE Property	Value	Description
Auto Start	False	The ADC Framework preconfigures and locks this field.
GTIOCA Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCA Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	Controls output pin level when the timer is stopped.
GTIOCB Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCB Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	Controls output pin level when the timer is stopped.
Callback	NULL	The ADC Framework preconfigures and locks this field.
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Audio Record ADC Framework Module Clock Configuration

The ADC peripheral module uses PCLKC as its clock source.

Audio Record ADC Framework Module Pin Configuration

The ADC peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. ADC pins must be configured as analog pins. The first table below

illustrates the method for selecting the pins within the SSP configuration window and the following table illustrates an example selection for the pins.

NOTE: For some peripherals, the Operation Mode selection mode determines what peripheral signals are available and thus what MCU pins are required.

Pin Selection Sequence for ADC

Resource	ISDE Tab	Pin selection Sequence
ADC	Pins	Select Peripherals > Analog: ADC > ADC/ADC0/ADC1

NOTE: The above selection sequence assumes ADC0 is the desired hardware target for the driver.

Pin Configuration Settings for ADC

Pin Configuration Property	Value	Description
Operation Mode	Disabled, Custom (Default: Custom)	Select operating mode for ADC
ADTRG	None, P407, P102 (Default: None)	ADTRG Pin
AN00-19	None, Pnnn, Pmmm (Default: None)	Analog input pins
PGAVSS0	None, P003 (Default: None)	PGAVSS pin

NOTE: The above example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.1.2.6 Using the Audio Record ADC Framework Module in an Application

The typical steps in using the Audio Record ADC Framework module in an application are:

1) Open the module using the open API
2) Start the recording using the start API
3) Store data in a user buffer with the callback

4) Operate on data as needed
5) Close the module using the close API

The above common steps are illustrated in a typical operational flow diagram in the figure below.

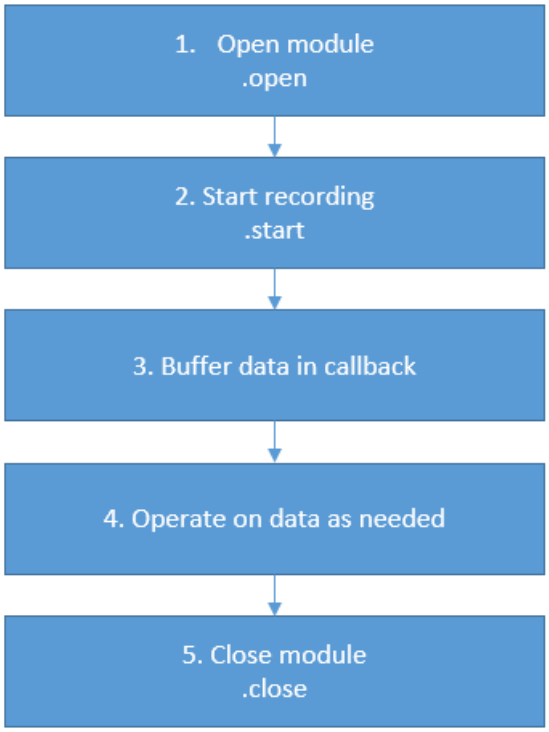


Figure 107: Flow Diagram of a Typical Audio Record ADC Framework Module Application

4.1.3 Audio Record I2S Framework

The Audio Record I2S Framework module provides high-level APIs for audio recording applications and uses the I2S interface. The Audio Record I2S Framework module uses the SSI, GPT and DTC peripherals on the Synergy MCU. A user-defined callback can be created to indicate that new samples are stored in the user buffer..

4.1.3.1 Audio Record I2S Framework Module Features

- Thread safe
- Records data in 8 or 16-bit PCM
- Periodic callback function when new samples are available
- Configurable number of samples (sample count) per callback

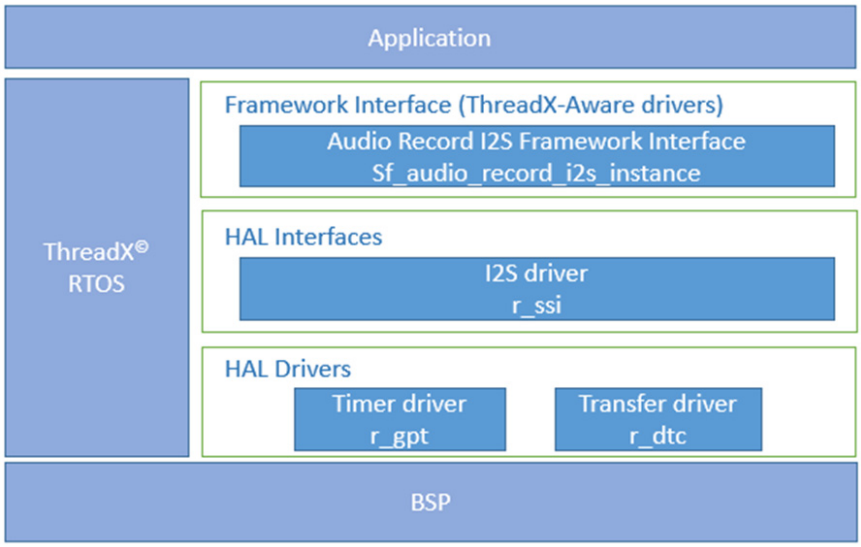


Figure 108: Audio Record I2S Framework Module Block Diagram

4.1.3.2 Audio Record I2S Framework Module APIs Overview

The Audio Record I2S Framework module defines APIs to open, start, stop and close the recording module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Audio Record I2S Framework Module API Summary

Function Name	Example API Call and Description
.open	g_sf_audio_record_i2s0.p_api->open (g_sf_audio_record_i2s0.p_ctrl, g_sf_audio_record_i2s0.p_cfg); Initializes audio recording framework.
.start	g_sf_audio_record_i2s0.p_api->start (g_sf_audio_record_i2s0.p_ctrl); Starts audio recording.

Function Name	Example API Call and Description
.stop	g_ sf_audio_record_i2s0.p_api->stop (g_sf_audio_record_i2s0.p_ctrl); Stops audio recording.
.infoGet	g_sf_audio_record_i2s0.p_api->infoGet (g_sf_audio_record_i2s0.p_ctrl, p_info); Gets channel information (Mono/Stereo).
.close	g_sf_audio_record_i2s0.p_api->close (g_sf_audio_record_i2s0.p_ctrl); Releases channel *mutex* and closes channel at HAL layer.
.versionGet	g_ sf_audio_record_i2s0.p_api->versionGet(&version); Gets version and stores it in provided version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_INTERNAL	An internal TheadX error has occurred.
SSP_ERR_NOT_OPEN	Unit is not open
SSP_ERR_ASSERTION	The parameter p_ctrl or p_sample is NULL.
SSP_ERR_IN_USE	Peripheral is still running in another mode; perform Close first.

Name	Description
SSP_ERR_UNSUPPORTED	Command not supported.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.1.3.3 Audio Record I2S Framework Module Operational Overview

The Audio Record I2S Framework Module uses the I2S HAL layer as the underlying interface for the audio data transfer and the data captured is stored in the user buffer. The data is made available for further processing as needed by the application.

The captured data is stored in a user defined buffer and this is done in the callback function as illustrated below:

Assuming the name of the callback has been configured to be sf_audio_record_user_callback.

```
uint16_t * audio_buffer;
void audio_record_user_callback (sf_audio_record_callback_args_t * p_args)
{
    audio_buffer = ((uint16_t *)sf_audio_record_i2s.p_cfg->p_capture_data_buffer
                    + (p_args->buffer_index));
}
```

Audio Record I2S Framework Module Important Operational Notes and Limitations

The Audio Record I2S Framework Module configuration data can specify the name of the data buffer, length of the data buffer, data size (8-bit or 16-bit samples), sampling iterations and the name of the callback.

- The framework currently supports recording 8 bit or 16 bit PCM data.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.3.4 Including the Audio Record I2S Framework Module in an Application

This section describes how to include the Audio Record I2S Framework module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Audio Record I2S Framework module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the Audio Record I2S Framework module is g_sf_audio_record_i2s0. This name can be changed in the associated Properties window.)

Audio Record I2S Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_audio_record_i2s0 Audio Record I2S Framework on sf_audio_record_i2s	Threads	New Stack> Framework> Audio> Audio Record I2S Framework on sf_audio_record_i2s

When the Audio Record I2S Framework module on sf_audio_record_i2s is added to the thread stack as shown in the following figure, the configurator automatically adds any lower-level drivers that are required. Any drivers that need additional configuration information will be box-text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

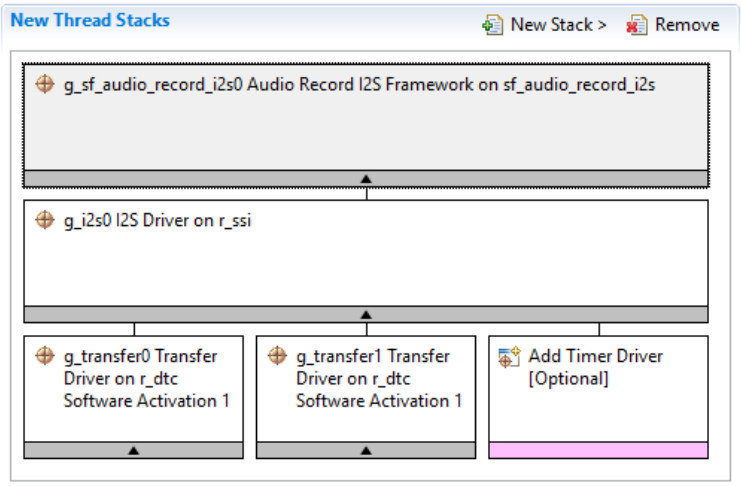


Figure 109: Audio Record I2S Framework Module Stack

4.1.3.5 Configuring the Audio Record I2S Framework Module

The Audio Record I2S Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the Audio Record I2S Framework Module on sf_audio_record_i2s

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_sf_audio_record_i2s0	Module name.
Name of the data-buffer to store samples	p_capture_data_buffer	Data-buffer name.
Length of the data buffer	2048	Length of the data buffer
Audio Record Data Size	8-Bit, 16-Bit Default: 16-Bit	Audio record data size selection
Number of sampling iterations	256	Number of sampling iterations
Callback	g_audio_record_framework_user_callback	Callback name.
Name of generated initialization funcion	sf_audio_record_i2s_init0	Name of generated initialization function.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the I2S HAL Module on r_ssi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_i2s0	Module name.
Channel	0	Physical hardware channel.
Audio Clock Frequency (Hertz)	2822400	Input audio clock frequency, used to generate the I2S clock. Must be a multiple between 1 and 128 of: (sampling_freq_hz * word_length_in_bits)
Sampling Frequecy (Hertz)	44100	Sampling frequency of audio data.
Data Bits	8 bits, 16, 18, 20, 22, 24 Default: 16 bits	Bit depth of audio data, which is the size in bits of one sample of audio data.
Word Length	8 bits, 16, 24, 32 Default: 16 bits	Word length of audio data, must be at least the same size as the bit depth (Data Bits field).
WS Continue Mode	Enabled, Disabled Default: Disabled	Enable WS continue mode to continue to output the word select line when the peripheral is idle. Disable to stop outputting the word select line when the peripheral is idle.
Audio Clock Frequency (Hertz)	External, GTIOC1A Default: External	Select External for external signal to AUDIO_CLK input pin or GTIOCA1.

ISDE Property	Value	Description
Name of I2S callback function to be defined by user	NULL	A user callback function must be registered in open. The callback will be called from the interrupt service routine (ISR) when the transmission FIFO reaches the high watermark point after all data for transmission is transmitted or when reception is complete (the requested number of bytes have been received). Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Transmit interrupt priority selection
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Receive interrupt priority selection
Idle/Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Idle/error interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different

default values and available configuration settings.

Configuration Settings for the DTC HAL Module on r_dtc Software Activation

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer0	Driver name.
Mode	Normal	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	FALSE	Auto enable selection.

ISDE Property	Value	Description
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the DTC HAL Module on r_dtc Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer1	Driver name.
Mode	Normal	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.

ISDE Property	Value	Description
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	FALSE	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the AGT HAL Module on r_agt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection
Name	g_timer0	Module name

ISDE Property	Value	Description
Channel	0	Channel selection
Mode	Periodic	Mode selection
Period Value	2822400 *2	Period value selection
Period Unit	Hertz	Period unit selection
Auto Start	FALSE	Auto start selection
Count Source	PCLKB, PCLKB/8, PCLKB/2, LOCO, AGT0 Underflow, AGT0 fSub Default: PCLKB	Count source selection
AGTO Output Enabled	True, False Default: False	AGTO output selection
AGTIO Output Enabled	True, False Default: False	AGTIO output selection
Output Inverted	True, False Default: False	Output inverted selection
Enable comparator A output pin	True, False Default: False	Enable comparator A output pin selection
Enable comparator B output pin	True, False Default: False	Enable comparator B output pin selection
Callback	NULL	Callback selection

ISDE Property	Value	Description
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the GPT HAL Module on r_gpt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection
Name	g_timer0	Module name
Channel	0	Channel selection
Mode	Periodic	Mode selection
Period Value	2822400 * 2	Period value selection
Period Unit	Hertz	Period unit selection
Duty Cycle Value	50	Duty cycle value selection
Duty Cycle Unit	Unit Raw Counts, Unit Percent, Unit Percent x 1000 Default: Unit Raw Counts	Duty cycle unit selection
Auto Start	FALSE	Auto start selection
GTIOCA Output Enabled	True, False Default: False	GTIOCA output enabled selection

ISDE Property	Value	Description
GTIOCA Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	GTIOCA stop level selection
GTIOCB Output Enabled	True, False Default: False	GTIOCB output enabled selection
GTIOCB Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	GTIOCB stop level selection
Callback	NULL	Callback selection
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Audio Record I2S Framework Module Clock Configuration

The Audio Record I2S Framework module uses the PCLKB as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

Audio Record I2S Framework Module Pin Configuration

To use the Audio Record I2S Framework module, the port pins for the peripheral inputs and outputs must be set in the pin configurator in the ISDE. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection Sequence for the Audio Record I2S Framework Module

Resource	ISDE Tab	Pin selection Sequence
SSI	Pins	Select Peripherals > Peripherals>Connectivity:SSI>SSI/SSI0/SSI1

4.1.3.6 Using the Audio Record I2S Framework Module in an Application

The typical steps in using the Audio Record I2S Framework module in an application are:

1) Initialize the module using the open API
2) Start the recording using the start API
3) Operate on data from the periodic callback function as needed
4) Close the module using the close API

These common steps are illustrated in a typical operational flow in the following figure:

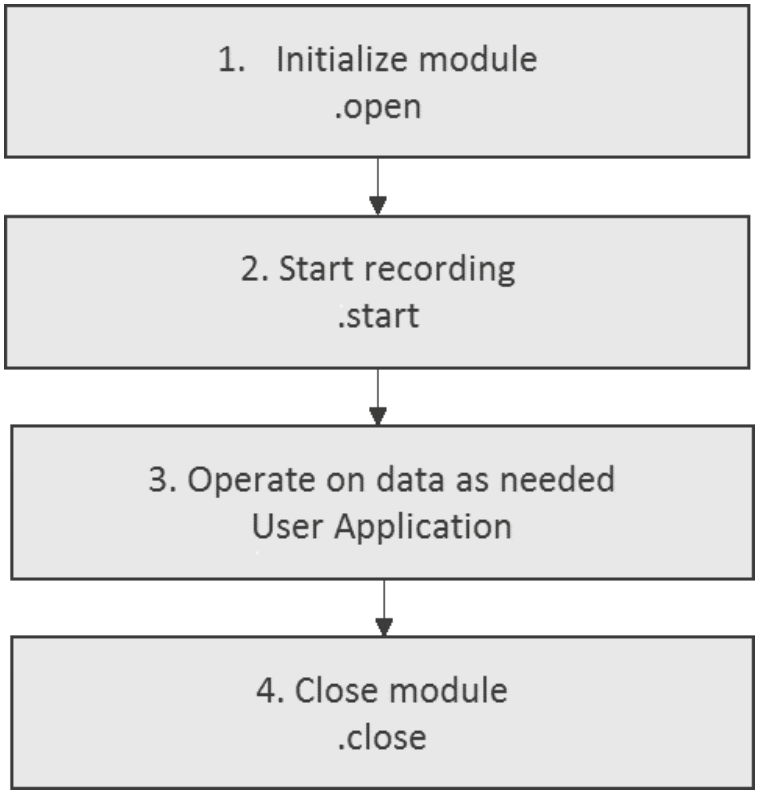


Figure 110: Flow Diagram of a Typical Audio Record I2S Framework Module Application

4.1.4 Audio Playback Framework

The Audio Playback Framework module provides high-level APIs for audio playback applications and is implemented on either sf_audio_playback_hw_dac or sf_audio_playback_hw_i2s. It handles the synchronization needed to play 16-bit and 8-bit pulse-code modulation (PCM) samples. The Audio Playback Framework uses the DAC (DAC12 or DAC8) or I2S, timer (AGT or GPT) and data-transfer (DMA or DTC) peripherals on a Synergy MCU. A user-defined callback can be created to respond to the need for additional data.

4.1.4.1 Audio Playback Framework Module Features

The Audio Playback Framework module supports the following features:

- Plays long buffers by splitting the data into manageable chunks
- Repeats playback until a ThreadX timeout (for repeated audio like sine wave tones or looped background music)
- Requests next data using callback after last buffer playback begins
- Software volume control
- Pause and resume functions
- Scaling, for example to move signed 16-bit PCM data into range of the unsigned 12-bit DAC or unsigned 8-bit DAC8
- Basic mixing for multiple streams.

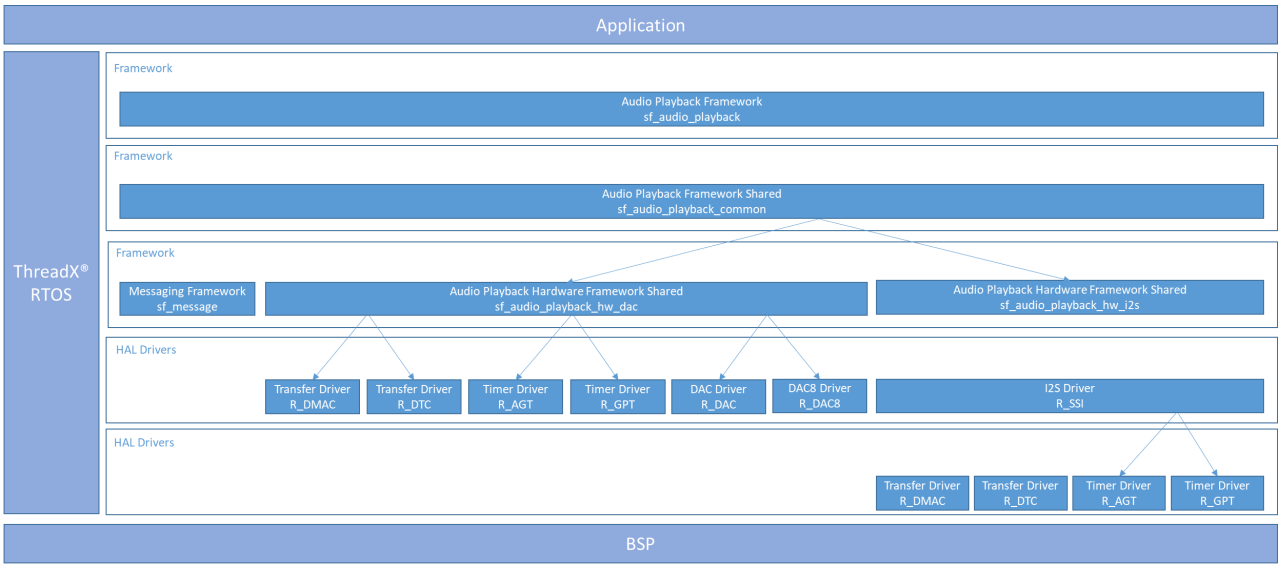


Figure 111: Audio Playback Framework Stack Options

4.1.4.2 Audio Playback Framework Module APIs Overview

The Audio Playback Framework module defines APIs for operations such as opening, starting, playing and stopping. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Audio Playback Framework API Summary

Function Name	Example API Call and Description
.open	g_sf_audio_playback0.p_api->open(g_sf_audio_playback0.p_ctrl, g_sf_audio_playback0.p_cfg); Open a device channel for read/write and control.
.start	g_sf_audio_playback0.p_api->start(g_sf_audio_playback0.p_ctrl, p_data, Timeout); Start Audio Playback Hardware.
.pause	g_sf_audio_playback0.p_api->pause(g_sf_audio_playback0.p_ctrl); Pause Audio Playback Hardware.
.stop	g_sf_audio_playback0.p_api->stop(g_sf_audio_playback0.p_ctrl); Stop Audio Playback Hardware.
.resume	g_sf_audio_playback0.p_api->resume(g_sf_audio_playback0.p_ctrl, &buffer, length); Resume playback.
.volumeSet	g_sf_audio_playback0.p_api->volumeSet(g_sf_audio_playback0.p_ctrl, volume); Sets volume.
.close	g_sf_audio_playback0.p_api->close(g_sf_audio_playback0.p_ctrl); Close the audio module.
.versionGet	g_sf_audio_playback0.p_api->versionGet(&version); Return the version of the module using the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manuals API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_ASSERTION	A pointer is NULL or a parameter is invalid.

Name	Description
SSP_ERR_OUT_OF_MEMORY	The number of streams open at once is limited to SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS. If this number is exceeded, an out of memory error occurs.
SSP_ERR_TIMEOUT	Timeout occurred before playback finished.
SSP_ERR_NOT_OPEN	The stream control block p_ctrl is not initialized.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.4.3 Audio Playback Framework Module Operational Overview

The Audio Playback Framework module creates a thread internally to support audio playback. The following figure shows a flowchart of the audio playback framework thread and its interactions with public Audio Playback Framework APIs.

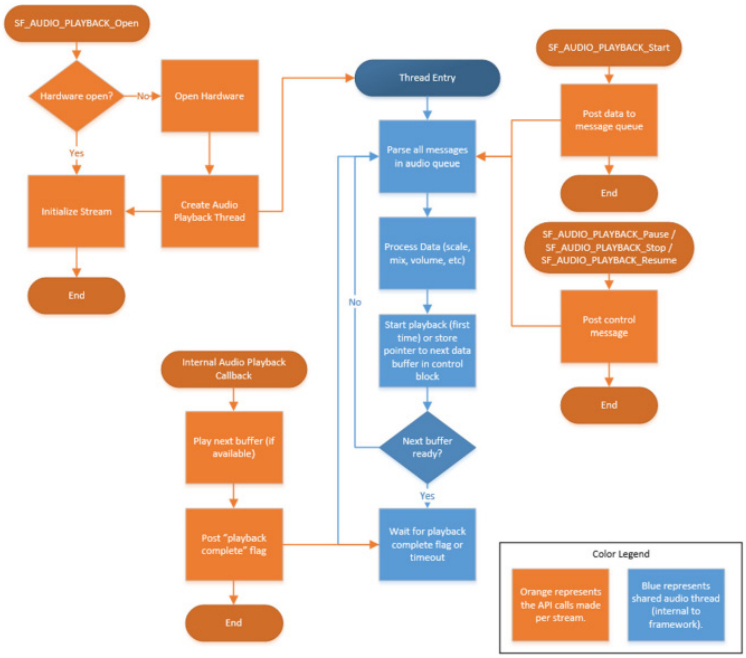


Figure 112: Audio Playback Framework Flowchart

Suggested use of the audio playback framework:

- Create a semaphore (for example, g_sf_audio_playback_semaphore). This can be done on the **Threads** tab. Set the initial value to 2 (the audio playback framework can store up to two data messages per stream).

- Create a callback function (for example, sf_audio_playback_callback). Enter the name of your callback function in the Audio Playback Framework instance. The callback function will be called when the audio playback framework is done with the data. In the callback, put the semaphore created above.
- In your main loop, get the semaphore before playing data. To play data, first acquire a buffer from the messaging framework, then create your audio playback data structure inside the buffer.

The Audio Playback Framework supports multiple audio streams on a single hardware port. A block diagram of the modules required if two streams are used is shown in the following figure:

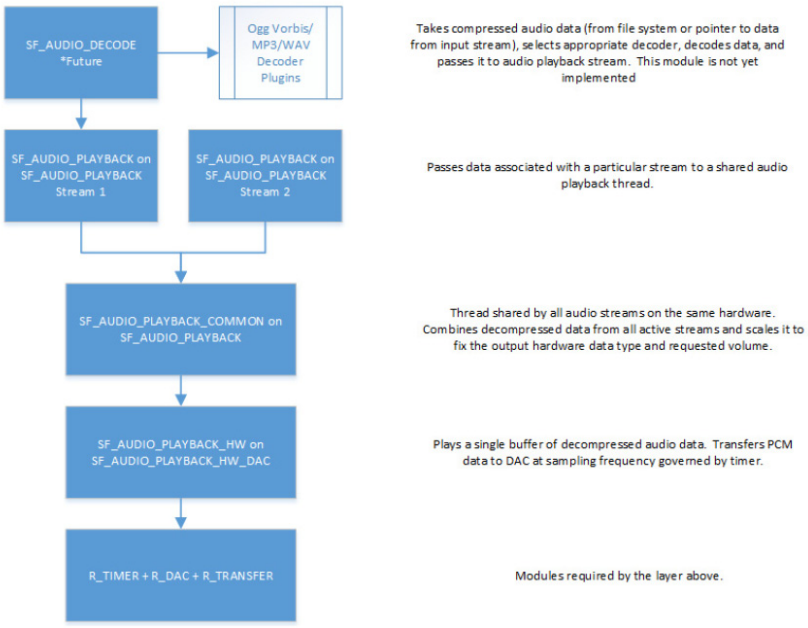


Figure 113: Implementing Multiple Audio Streams

Audio Playback Framework Module Operational Notes

Configuring Messages

Use the Messaging Framework configurator on the Messaging tab to configure the messaging framework:

1) Highlight the Audio Playback event class.
2) Add a new subscriber. Select the following configurations and make sure the Message Class Instance property set in the Properties tab of the Audio Playback Framework on sf_audio_playback module is between the Start and End instance.
   - Thread: Any thread in the application.
   - Start: First audio instance used in application.
   - End: Last audio instance used in application.
3) Highlight the new Subscriber in the Audio Playback Subscribers. Record the Symbol name.
4) Go back to the **Threads** tab.

5) Highlight the Audio Playback Framework Shared module in HAL/Common, and set the Audio Message Queue Name to the Symbol name from the Audio Playback Subscriber.

Using the I2S Implementation

The audio framework I2S hardware port has dependencies on the I2S driver module. The I2S driver module can be accelerated with DTC (recommended).

- Set the ISDE properties for the I2S driver module.
  - Set the Audio Clock Frequency (Hertz) to the frequency of the input audio clock used.
  - Set the Sampling Frequency (Hertz) to the sampling frequency of your audio data.
  - Set the Data Bits and Word Length to 16 bits (audio framework accepts 16 bit samples only).
  - Enable the SSIn TXI and SSIn INT interrupts.
- (Recommended) The Transfer module on r_dtc is added automatically.

Using the DAC Implementation

The audio framework DAC hardware port has dependencies on the Timer, DAC, and Transfer API modules.

- Add a Timer module.
  - Set the Frequency in Hz to the sampling frequency of your audio data.
  - Enable the interrupt if using DTC as the transfer module (recommended).
- Add a DAC (DAC12 or DAC8) module.
- Add a Transfer module on r_dtc.
  - Set Destination pointer to &R_DAC->DADRn[0] if using DAC channel 0 or &R_DAC->DADRn[1] if using DAC channel 1.
  - Set Destination pointer to &R_DAC8->DADRn[2] if using DAC8 channel 2 (DK-S128) or &R_DAC8->DADRn[0] if using DAC8 channel 0 (S1JA).
  - Set the Activation source to the timer interrupt chosen above.

Other Operational Notes

- The Queue used must match the name specified in Properties for Audio Playback Framework Shared on sf_audio_playback (default is g_sf_audio_playback_queue).

Audio Playback Framework Module Limitations

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.4.4 Including the Audio Playback Framework Module in an Application

This section describes how to include the Audio Playback Framework module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few sections of the SSP User's Manual, available as described in the Reference Section at the end of this document to learn how to manage each of these important steps in creating SSP-based applications.

To add the Audio Playback to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Audio Playback is g_sf_audio_playback. This name can be changed in the associated Properties window.)

Audio Playback Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_audio_playback Audio Playback Framework on g_sf_audio_playback	**Threads**	**New Stack> Framework> Audio> Audio Playback Framework on g_sf_audio_playback**

When the Audio Playback Framework module on sf_audio_playback is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information will have box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level drivers; these are either optional or recommended (this is indicated in the block with the inclusion of this text.) If the addition of lower-level drivers is required, the module description will include "Add" in the text. Clicking on any Pink banded modules will bring up the "New" icon and then display the possible choices.

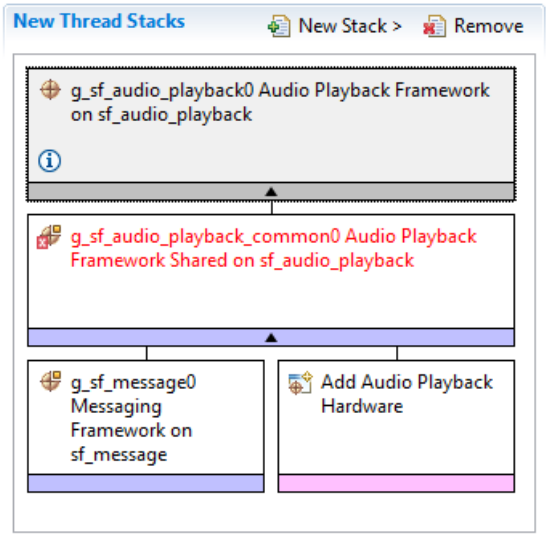


Figure 114: Audio Playback DAC Framework Stack

4.1.4.5 Configuring the Audio Playback Framework Module

The Audio Playback Framework module must be configured by you for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous

'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority: this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for Audio Playback Framework on sf_audio_playback

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking
Buffer Size Bytes	512	Buffer size bytes selection
Maximum Number of Streams	1	Maximum number of streams selection
Thread Stack Size	512	Thread stack size selection
Name	g_sf_audio_playback0	Module name
Message Class Instance	0	Message class instance selection
Callback	NULL	Callback selection.
Name of generated initialization function	sf_audio_playback_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful

to select the DAC Channel based on the target hardware implementation. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

NOTE: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the Audio Playback Framework Low Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

Configuration for Audio Playback Framework Shared on sf_audio_playback_common

ISDE Property	Value	Description
Name	g_sf_audio_playback_common0	Module name
Thread Priority	3	Priority for the thread (do not edit)
Audio Message Queue Name	g_sf_audio_playback_queue	Message Queue name

Configuration for Messaging Framework on sf_message

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Message Queue Depth (Total number of messages to be enqueued in a Message Queue)	16	Specify the size of Thread X Message Queue in bytes for Message Subscribers. This value is applied to all the Message Queues.
Name	g_sf_message0	The name of Messaging Framework module control block instance.

ISDE Property	Value	Description
Work memory size in bytes	2048	Specify the work memory size in bytes. Choosing a small number results a small number of buffers which can be allocated at the same time (You can confirm the total buffer number on: sf_message_ctrl_t::number_of_buffers). If the value is smaller than the peak number of messages to be posted at the same time, the Framework occurs a buffer allocation failure affecting system performance.
Pointer to subscriber list array	p_subscriber_lists	Specify the name of pointer to the Subscriber List array.
name of the block pool internally used in the messaging framework	sf_msg_blk_pool	The name of memory block memory the Framework creates in the control block. This parameter might be useful for debugging purpose but NULL can be specified for saving memory.
Name of generated initialization function	sf_message_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Audio Playback Hardware Framework Shared on sf_audio_playback_hw_dac

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking
DMAC Support	Disabled, Enabled Default: Disabled	DMAC support selection
Name	g_sf_audio_playback_hw0	Module name

Configuration for the Transfer Driver on r_dmac (Transfer Driver Option)

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Name	g_transfer0	Module name
Channel	0	Channel selection
Mode	Normal	Mode selection
Transfer Size	2 Bytes	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source	Software Activation, Peripheral Events Default: Software Activation	Activation source selection
Auto Enable	False	Auto enable selection
Callback	NULL	Callback selection

ISDE Property	Value	Description
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection

Configuration Settings for Transfer Driver on r_dtc (Transfer Driver Option)

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section selection
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	2 Bytes	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection

ISDE Property	Value	Description
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Software Activation 1, Software Activation 2, Peripheral Events Default: Software Activation 1	Activation source selection
Auto Enable	True, False Default: True	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

Configuration Settings for Timer Driver on r_agt (Timer Driver Option)

ISE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables parameter checking
Name	g_timer0	Module name
Channel	0	Physical hardware channel

ISE Property	Value	Description
Mode	Periodic	Warning: One Shot functionality is not available in the GPT hardware, so it is implemented in software by stopping the timer in the ISR called when the period expires. For this reason, ISR's must be enabled for one-shot mode even if the callback is unused.
Period Value	10	See Timer Period Calculation
Period Unit	Hertz	See Timer Period Calculation
Auto Start	FALSE	Set to true to start the timer after configuring or false to leave the timer stopped until start is called
Count Source	PCLKB, PCLKB/8, PCLKB/2, LOCO, AGT0 Underflow, AGT0 fSub Default: PCLKB	The clock source for the AGT counter
AGTO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTO pin). Set to false for no output of the timer signal
AGTIO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTIO pin). Set to false for no output of the timer signal.
Output Inverted	True, False Default: False	Set to false to start the output signal low. Set to true to start the output signal high.
Enable comparator A output pin	True, False Default: False	Enable comparator A output pin selection
Enable comparator B output pin	True, False Default: False	Enable comparator B output pin selection

ISE Property	Value	Description
Callback	NULL	A user callback function can be registered in open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Timer interrupt priority. 0 is the highest priority.

Configuration Settings for Timer Driver on r_gpt (Timer Driver Option)

ISE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_timer0	Module name.
Channel	0	Channel selection.
Mode	Periodic	Warning: One Shot functionality is not available in the GPT hardware, so it is implemented in software by stopping the timer in the ISR called when the period expires. For this reason, ISR's must be enabled for one-shot mode even if the callback is unused.

ISE Property	Value	Description
Period Value	10	See Timer Period Calculation
Period Unit	Hertz	See Timer Period Calculation
Duty Cycle Value	50	Duty cycle value selection
Duty Cycle Unit	Unit Raw Counts, Unit Percent, Unit Percent x 1000 Default: Unit Raw Counts	Duty cycle unit selection
Auto Start	False	Set to true to start the timer after configuring or false to leave the timer stopped until start is called.
GTIOCA Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCA Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	Controls output pin level when the timer is stopped.
GTIOCB Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCB Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	Controls output pin level when the timer is stopped.

ISE Property	Value	Description
Callback	NULL	A user callback function can be registered in open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection

Configuration Settings for DAC Driver on r_dac

Parameter	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_dac0	Module name.
Channel	0	Set to 0 for output DA0 or 1 for output DA1.
Synchronize with ADC	Enabled, Disabled Default: Disabled	Set to true for anti-interference synchronization with the Analog-to-Digital Converter (ADC) Module. Set to false false if power supply interference between the analog modules is not a problem, or if asynchronous conversion by the DAC Module is desired.

Parameter	Value	Description
Data Format	Right Justified	Set to zero, if 12-bit data values are loaded in bits 11 through 0, or right justified. Set to one, if 12-bit data values are loaded in bits 15 through 4, or left justified.
Output Amplifier	Enable, Disable Default: Disable	Set to true, if output amplifier hardware function is desired. Set to false to bypass output amplifier hardware function.

Configuration Settings for the Audio Playback Framework Shared on sf_audio_playback_hw_i2s

Parameter	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_sf_audio_playback_hw0	Module name.

Configuration Settings for the I2S HAL Module on r_ssi

Parameter	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_i2s0	Module name.
Channel	0	Physical hardware channel.
Audio Clock Frequency (Hertz)	2822400	Input audio clock frequency, used to generate the I2S clock. Must be a multiple between 1 and 128 of: (sampling_freq_hz * word_length_in_bits)
Sampling Frequecy (Hertz)	44100	Sampling frequency of audio data.

Parameter	Value	Description
Data Bits	8 bits, 16, 18, 20, 22, 24 Default: 16 bits	Bit depth of audio data, which is the size in bits of one sample of audio data.
Word Length	8 bits, 16, 24, 32 Default: 16 bits	Word length of audio data, must be at least the same size as the bit depth (Data Bits field).
WS Continue Mode	Enabled, Disabled Default: Disabled	Enable WS continue mode to continue to output the word select line when the peripheral is idle. Disable to stop outputting the word select line when the peripheral is idle.
Audio Clock	External, GTIOC1A Default: External	Select External for external signal to AUDIO_CLK input pin or GTIOC1A for internal connection to GPT channel 1 GTIOC1A.
Name of I2S callback function to be defined by user	NULL	A user callback function must be registered in open. The callback will be called from the interrupt service routine (ISR) when the transmission FIFO reaches the high watermark point after all data for transmission is transmitted or when reception is complete (the requested number of bytes have been received). Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Transmit interrupt priority selection

Parameter	Value	Description
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Receive interrupt priority selection
Idle/Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Idle/error interrupt priority selection

Configuration Settings for the DTC HAL module on r_dtc Software Activation 1

Parameter	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer0	Driver name.
Mode	Normal	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.

Parameter	Value	Description
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection

Configuration Settings for the DTC HAL Module on r_dtc Software Activation 1

Parameter	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.

Parameter	Value	Description
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer1	Driver name.
Mode	Normal	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	False	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.

Parameter	Value	Description
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection

Configuration Settings for the AGT HAL Module on r_agt

Parameter	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection
Name	g_timer0	Module name
Channel	0	Channel selection
Mode	Periodic	Mode selection
Period Value	2822400 * 2	Period value selection
Period Unit	Hertz	Period unit selection
Auto Start	False	Auto start selection
Count Source	PCLKB, PCLKB/8, PCLKB/2, LOCO, AGT0 Underflow, AGT0 fSub Default: PCLKB	Count source selection
AGTO Output Enabled	True, False Default: False	AGTO output selection
AGTIO Output Enabled	True, False Default: False	AGTIO output selection

Parameter	Value	Description
Output Inverted	True, False Default: False	Output inverted selection
Enable comparator A output pin	True, False Default: False	Enable comparator A output pin selection
Enable comparator B output pin	True, False Default: False	Enable comparator B output pin selection
Callback	NULL	Callback selection
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection

Configuration Settings for the GPT HAL Module on r_gpt

Parameter	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection
Name	g_timer0	Module name
Channel	0	Channel selection
Mode	Periodic	Mode selection
Period Value	2822400 *2	Period value selection
Period Unit	Hertz	Period unit selection
Duty Cycle Value	50	Duty cycle value selection

Parameter	Value	Description
Duty Cycle Unit	Unit Raw Counts, Unit Percent, Unit Percent x 1000 Default: Unit Raw Counts	Duty cycle unit selection
Auto Start	False	Auto start selection
GTIOCA Output Enabled	True, False Default: False	GTIOCA output enabled selection
GTIOCA Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	GTIOCA stop level selection
GTIOCB Output Enabled	True, False Default: False	GTIOCB output enabled selection
GTIOCB Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	GTIOCB stop level selection
Callback	NULL	Callback selection
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 Group MCU. Other MCUs may have different default values and available configuration settings.

Audio Playback Framework Clock Configuration

The Audio Playback Framework hardware modules use the peripheral clocks available from the Clock configuration window.

Audio Playback Framework Pin Configuration

The DAC or SSI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following illustrate the method for selecting the pins within the SSP configuration window and shows an example selection for the associated pins.

NOTE: For some peripherals, the operation mode selection determines what peripheral signals are available and thus what MCU pins are required.

Pin Selection Sequence for DAC

Resource	ISDE Tab	Pin selection Sequence
DAC	Pins	Select **Peripherals > Analog: DAC12 > DAC120/121/122**
SSI	Pins	Select **Peripherals > Connectivity: SSI > SSI/0/1**

Note: The selection sequence assumes DAC0 or DAC1, or SSI0 or SSI1 is the desired hardware target for the driver.

Pin Configuration Settings for DAC driver on DAC

Pin Configuration Property	Value	Description
Operation Mode	Enabled, Disabled	Operation selection
DAC	None, P014 (Default: P014)	DAC pin selection

Note: The example values are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.1.4.6 Using the Audio Playback Framework Module in an Application

The typical steps in using the Audio Playback Framework module in an application are:

1) Initialize an audio stream using the open API
2) Use the callback function to post to a semaphore with an initial count equal to the number of buffers used in the application. Get this semaphore in the application thread before calling the start API.
3) Acquire a buffer from the Messaging Framework using the bufferAcquire API.
4) Create the Audio Framework Data Structure sf_audio_playback_data_t inside the buffer.

5) Start the Audio Playback Framework using the start API.

6) If multiple streams are desired, repeat steps 1-5 for any additional audio streams. A separate audio stream should only be used if the streams may need to play simultaneously using mixing. If audio sounds are always played in sequence and never overlap, the stream can be reused.

These common steps are illustrated in a typical operational flow diagram in the following figure:

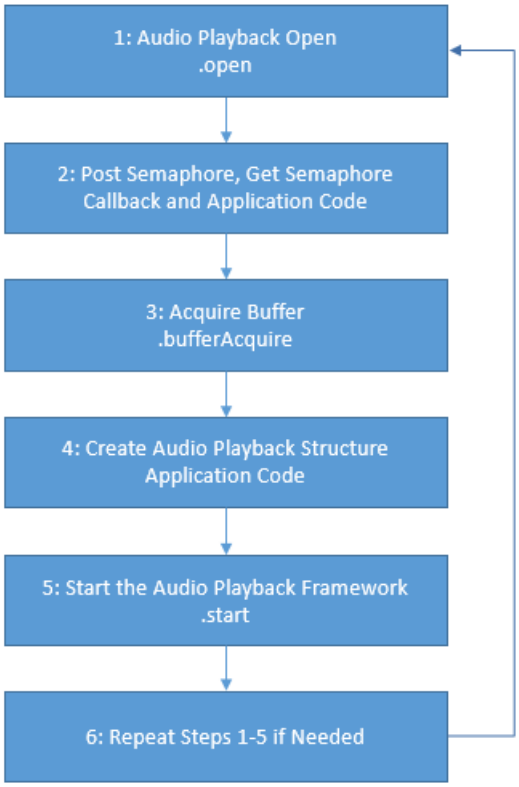


Figure 115: Flow Diagram of a Typical Audio Playback Application

4.1.5 Audio Playback DAC Framework

The Audio Playback DAC Framework module is a high-level API for audio playback applications and is implemented on sf_audio_playback_hw_dac. It handles the synchronization needed to play 16-bit pulse-code modulation (PCM) samples. The Audio Playback DAC Framework uses the DAC, timer (AGT or GPT) and data-transfer (DMA or DTC) peripherals on a Synergy MCU. A user-defined callback can be created to respond to the need for additional data.

4.1.5.1 Audio Playback Hardware DAC Framework Module Features

The Audio Playback Hardware DAC Framework module supports the following features:

- Plays long buffers by splitting the data into manageable chunks.
- Repeats playback until a ThreadX timeout (for repeated audio like sine wave tones or looped background music).

- Requests next data using callback after last buffer playback begins.
- Software volume control.
- Pauses and resumes functions.
- Scaling- for example, to move signed 16-bit PCM data into range of the unsigned 12-bit DAC or unsigned 8-bit DAC8.
- Basic mixing for multiple streams.

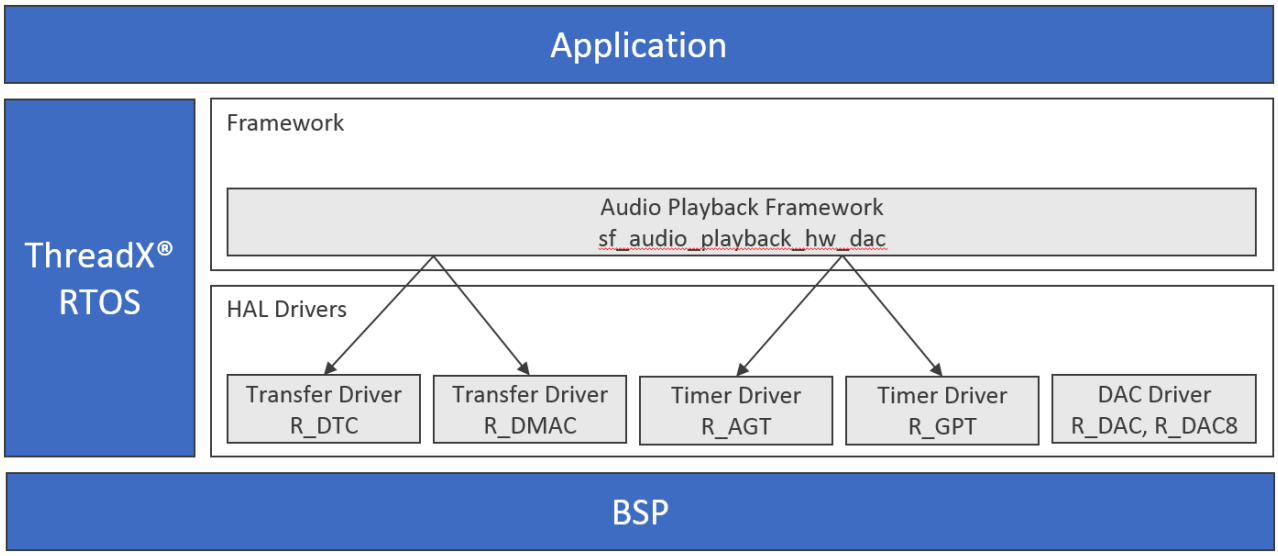


Figure 116: Audio Playback Hardware DAC Framework Module Block Diagram

4.1.5.2 Audio Playback Hardware DAC Framework Module APIs Overview

The Audio Playback DAC Module defines APIs for operations such as opening, starting, playing, and stopping. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Audio Playback Hardware DAC Framework Module API Summary

Function Name	Example API Call and Description
.open	g_sf_audio_playback_hw0.p_api->open(g_sf_audio_playback_hw0.p_ctrl, g_sf_audio_playback_hw0.p_cfg); Open a device channel for read/write and control.

Function Name	Example API Call and Description
.start	g_sf_audio_playback_hw0.p_api->start(g_sf_audio_playback_hw0.p_ctrl, &p_data, Timeout); Start Audio Playback Hardware.
.stop	g_sf_audio_playback_hw0.p_api->stop(g_sf_audio_playback_hw0.p_ctrl); Stop Audio Playback Hardware.
.play	g_sf_audio_playback_hw0.p_api-> stop(g_sf_audio_playback_hw0.p_ctrl, p_buffer, length); Play audio buffer.
. dataTypeGet	g_sf_audio_playback_hw0.p_api->dataTypeGet(g_sf_audio_playback_hw0.p_ctrl, p_data_type); Stores expected data type in provided pointer p_data_type.
.close	g_sf_audio_playback_hw0.p_api->close(g_sf_audio_playback_hw0.p_ctrl); Close the audio module.
.versionGet	g_sf_audio_playback_hw0.p_api->versionGet(&version); Return the version of the module with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.

Name	Description
SSP_ERR_ASSERTION	A pointer is NULL or a parameter is invalid.
SSP_ERR_OUT_OF_MEMORY	The number of streams open at once is limited to SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS. If this number is exceeded, an out of memory error occurs.
SSP_ERR_TIMEOUT	Timeout occurred before playback finished.
SSP_ERR_NOT_OPEN	The stream control block p_ctrl is not initialized.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.5.3 Audio Playback Hardware DAC Framework Module Operational Overview

The Audio Playback Framework module creates a thread internally to support audio playback. The following figure shows a flowchart of the audio playback framework thread and its interactions with public Audio Playback Framework APIs.

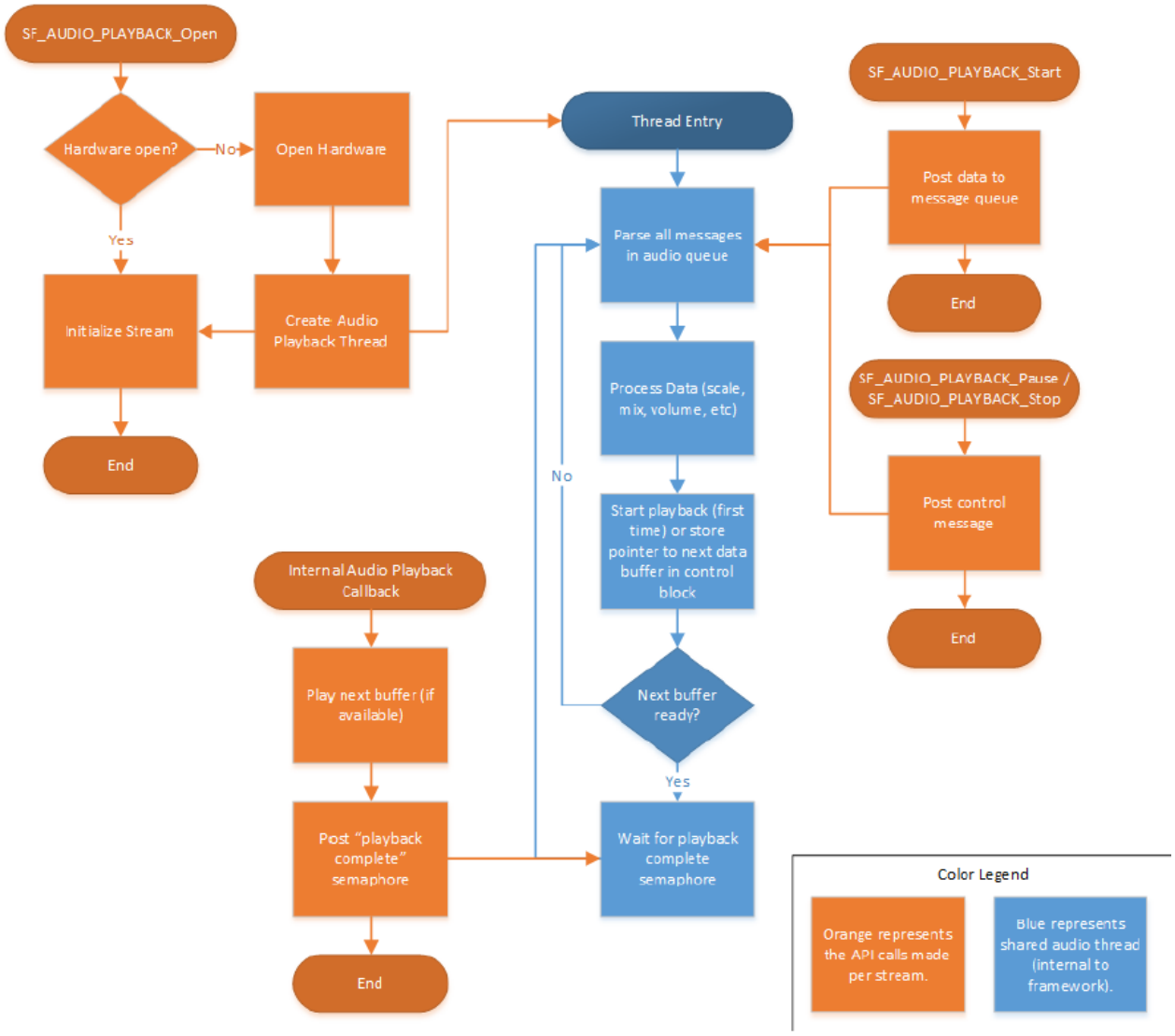


Figure 117: Audio Playback Framework Flowchart

Suggested use of the audio playback framework:

- Create a semaphore (for example g_sf_audio_playback_semaphore). This can be done on the Threads Tab. Set the initial value to 2 (the audio playback framework can store up to two data messages per stream).
- Create a callback function (for example sf_audio_playback_callback). Enter the name of your callback function in the Audio Playback Framework instance. The callback function will be called when the audio playback framework is done with the data. In the callback, put the semaphore created above.
- In your main loop, get the semaphore before playing data. To play data, first acquire a buffer from the messaging framework, then create your audio playback data structure inside the buffer.

The Audio Playback Framework supports multiple audio streams on a single hardware port. A block diagram of the modules required if two streams are used is shown in the following figure:

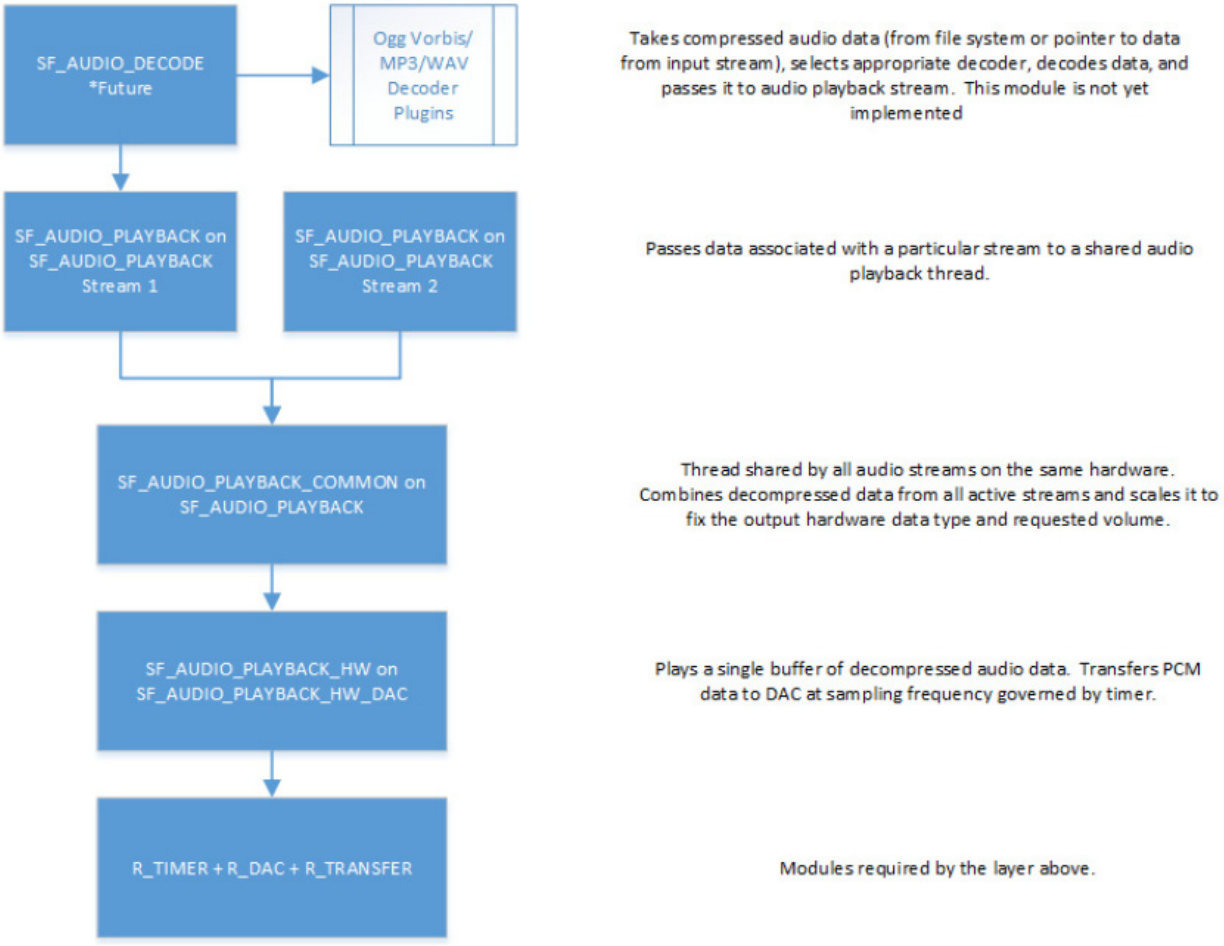


Figure 118: Implementing Multiple Audio Streams

Audio Playback Hardware DAC Framework Module Important Operational Notes and Limitations

- The Audio framework DAC hardware port uses the transfer API to transfer audio data from a playback buffer to DAC at a sampling frequency defined by an internal timer.
- The audio framework DAC hardware port has dependencies on the Timer, DAC, and Transfer API modules.
  - Add a timer module.
  - **Set the Frequency in Hz to the sampling frequency of your audio data.**
  - **Enable the interrupt if using DTC as the transfer module (recommended).**
  - Add a transfer module on r_dtc or r_dmac.

- **For DTC:**
- **Set Destination pointer to &R_DAC->DADRn[0] if using DAC channel 0 or &R_DAC->DADRn[1] if using DAC channel 1.**
- **Set the Activation source to the timer interrupt chosen above.**
- **Set Destination pointer to &R_DAC8->DADRn[2] if using DAC8 channel 2 (DK-S128) or &R_DAC8->DADRn[0] if using DAC8 channel 0 (S1JA).**

```
-   For DMAC:

    -   Enable dmac support

    -   Set Destination pointer to &R\_DAC-&gt;DADRn\[0\] if
        using DAC channel 0 or &R\_DAC-&gt;DADRn\[1\] if using
        DAC channel 1.

    -   Set the Activation source to the timer interrupt chosen
        above.
```

- The Audio Playback Framework is designed to support the following MCU families with no changes to the API:
  - S7G2
  - S5D5, S5D9, S128, S1JA
  - S3A3, S3A6, S3A7, S3D9
  - S124, S128, S1JA
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.5.4 Including the Audio Playback Hardware DAC Framework Module in an Application

This section describes how to include the Audio Playback Hardware DAC Framework module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the Getting Started Guide for SSP given in the References section at the end of this document to learn how to manage each of these important steps in creating SSP-based applications.

To add the Audio Playback Hardware DAC Framework to your application, simply add it to a project thread using the stacks selection sequence given in the following table. (The default name for the Audio Playback Hardware DAC Framework is g_sf_audio_playback_hw0. This name can be changed in the associated **Properties** window.)

Audio Playback Hardware DAC Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_audio_playback_hw0 Audio Playback Hardware Framework on g_sf_audio_playback_hw0	**Threads**	**New Stack> Framework> Audio> Audio Playback Hardware Framework on g_sf_audio_playback_hw_dac**

When the Audio Playback Hardware DAC Framework module on sf_audio_playback_hw_dac is added to the thread stack as shown in the following figure, the configurator automatically adds the needed lower-level drivers. Any drivers that need additional configuration information will be highlighted in red.

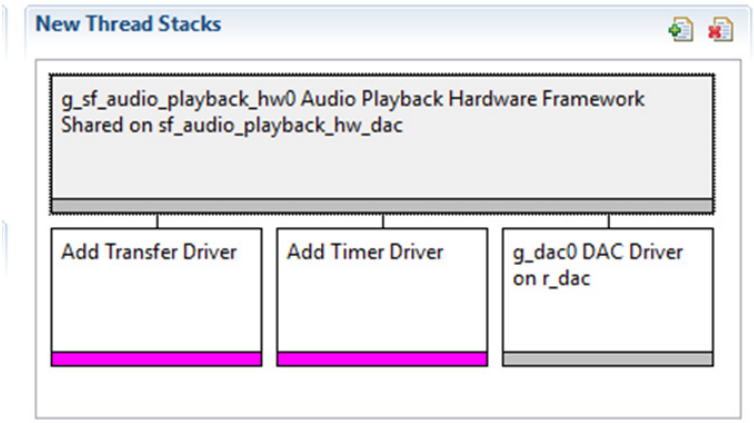


Figure 119: Audio Playback Hardware DAC Framework Module Stack

4.1.5.5 Configuring the Audio Playback Hardware DAC Framework Module

The Audio Playback Hardware DAC Framework module must be configured by you for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manualapproaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the Interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE and create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful hands-on approach to learning

the ins and outs of developing with SSP.

Configuration Settings for the Audio Playback Hardware DAC Framework Module on sf_audio_playback_hw_dac

Parameter	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
DMAC Support	Enabled, Disabled Default: Disabled	DMAC support enabled/disabled.
Name	g_sf_audio_playback_hw0	Module name.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select the DAC Channel based on the target hardware implementation. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Configuration Settings for the Audio Playback DAC Framework Low-Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated with red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

Configuration Settings for the DMAC HAL Module on r_dmac Software Activation

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Name	g_transfer0	Module name
Channel	0	Channel selection
Mode	Normal	Mode selection
Transfer Size	2 Bytes	Transfer size selection

ISDE Property	Value	Description
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source	Software Activation, Peripheral Events Default: Software Activation	Activation source selection
Auto Enable	False	Auto enable selection
Callback	NULL	Callback selection
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection

NOTE: The example settings and defaults are for a project using the Synergy SK-S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the DTC HAL Module on r_dtc Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section selection
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	2 Bytes	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Software Activation 1, Software Activation 2, Peripheral Events Default: Software Activation 1	Activation source selection
Auto Enable	False	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection

ISDE Property	Value	Description
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the AGT HAL Module on r_agt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables parameter checking.
Name	g_timer0	Module name.
Channel	0	Physical hardware channel.
Mode	Periodic	Warning: One Shot functionality is not available in the GPT hardware, so it is implemented in software by stopping the timer in the ISR called when the period expires. For this reason, ISRs must be enabled for one-shot mode even if the callback is unused.
Period Value	10	See Timer Period Calculation.
Period Unit	Hertz	See Timer Period Calculation.
Auto Start	FALSE	Set to true to start the timer after configuring or false to leave the timer stopped until start is called.

ISDE Property	Value	Description
Count Source	PCLKB, PCLKB/8, PCLKB/2, LOCO, AGT0 Underflow, AGT0 fSub Default: PCLKB	The clock source for the AGT counter.
AGTO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTO pin). Set to false for no output of the timer signal.
AGTIO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTIO pin). Set to false for no output of the timer signal.
Output Inverted	True, False Default: False	Set to false to start the output signal low. Set to true to start the output signal high.
Callback	NULL	A user callback function can be registered in open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Timer interrupt priority. 0 is the highest priority.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the GPT HAL Module on r_gpt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_timer0	Module name.
Channel	0	Channel selection.
Mode	Periodic	Warning: One-shot functionality is not available in the GPT hardware, so it is implemented in software by stopping the timer in the ISR called when the period expires. For this reason, ISRs must be enabled for one-shot mode even if the callback is unused.
Period Value	10	See Timer Period Calculation
Period Unit	Hertz	See Timer Period Calculation
Duty Cycle Value	50	Duty cycle value selection
Duty Cycle Unit	Unit Raw Counts, Unit Percent, Unit Percent x 1000 Default: Unit Raw Counts	Duty cycle unit selection
Auto Start	False	Set to true to start the timer after configuring or false to leave the timer stopped until start is called.
GTIOCA Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCA Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	Controls output pin level when the timer is stopped.

ISDE Property	Value	Description
GTIOCB Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCB Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	Controls output pin level when the timer is stopped.
Callback	NULL	A user callback function can be registered in open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the DAC HAL Module on r_dac

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.

ISDE Property	Value	Description
Name	g_dac0	Module name.
Channel	0	Set to 0 for output DA0 or 1 for output DA1.
Synchronize with ADC	Enabled, Disabled Default: Disabled	Set to true for anti-interference synchronization with the Analog-to-Digital Converter (ADC) Module. Set to false if power supply interference between the analog modules is not a problem, or if asynchronous conversion by the DAC Module is desired.
Data Format	Right Justified	Set to zero, if 12-bit data values are loaded in bits 11 through 0, or right justified. Set to one, if 12-bit data values are loaded in bits 15 through 4, or left justified.
Output Amplifier	Enable, Disable Default: Disable	Set to true, if output amplifier hardware function is desired. Set to false to bypass output amplifier hardware function.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Audio Playback Hardware DAC Framework Module Clock Configuration

The Audio Playback Framework hardware modules (DAC) use the peripheral clocks available in the Clocks configuration window.

Audio Playback Hardware DAC Framework Module Pin Configuration

The DAC or SSI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the associated pins.

NOTE: For some peripherals, the operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection for DAC

Resource	ISDE Tab	Pin selection Sequence
ADC	Pins	Select **Peripherals > Analog:ADC > ADC0/ADC1**

NOTE: The selection sequence assumes ADC0 or ADC1, or SSI0 or SSI1 is the desired hardware target for the driver.

Pin Configuration Settings for the DAC HAL Module on r_dac

Pin Configuration Property	Value	Description
Operation Mode	Enabled, Disabled	Operation selection

| DA | None, P014
(Default: P014) | DAC pin selection |

NOTE: The above example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.1.5.6 Using the Audio Playback Hardware DAC Framework Module in an Application

The typical steps in using the Audio Playback Hardware DAC Framework module in an application are:

1) Initialize the Audio Playback DAC Framework using the open API
2) Start the low-level hardware of the Audio Playback DAC framework using the start API.
3) Play the PCM audio samples using the play API.
4) PCM audio samples supported by the hardware is provided by the dataTypeGet API.
5) Stop the low-level hardware of the Audio Playback DAC Framework using stop API.
6) Close the Audio Playback DAC framework using the close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

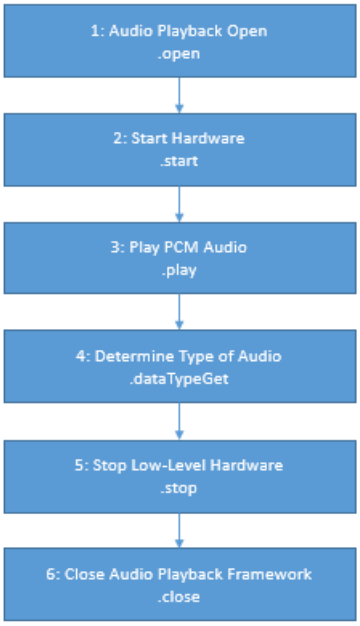


Figure 120: Flow Diagram of a Typical Audio Playback Hardware DAC Framework Module Application

4.1.6 Audio Playback I2S Framework

The Audio Playback I2S Framework module provides high-level APIs for Audio Playback applications and is implemented on sf_audio_playback_hw_i2s. It handles the synchronization needed to play 16-bit pulse-code modulation (PCM) samples. The Audio Playback Framework uses the I2S, Timer (AGT or GPT) and Data Transfer (DMA or DTC) peripherals on a Synergy MCU. A user defined callback can be created to respond to the need for additional data.

This document is divided into the following sections which can be read independently (by a more experienced Synergy developer) or in series (by developers new to the Synergy Platform.)

4.1.6.1 Audio Playback I2S Framework Module Features

The Audio Playback I2S Hardware Framework module supports the following features:

- Play long buffers by splitting the data into manageable chunks.
- Repeat playback until ThreadX timeout (for repeated audio like sine wave tones or looped background music).
- Request next data using callback after last buffer playback begins.
- Software volume control.
- Pause and resume functions.
- Basic mixing for multiple streams.

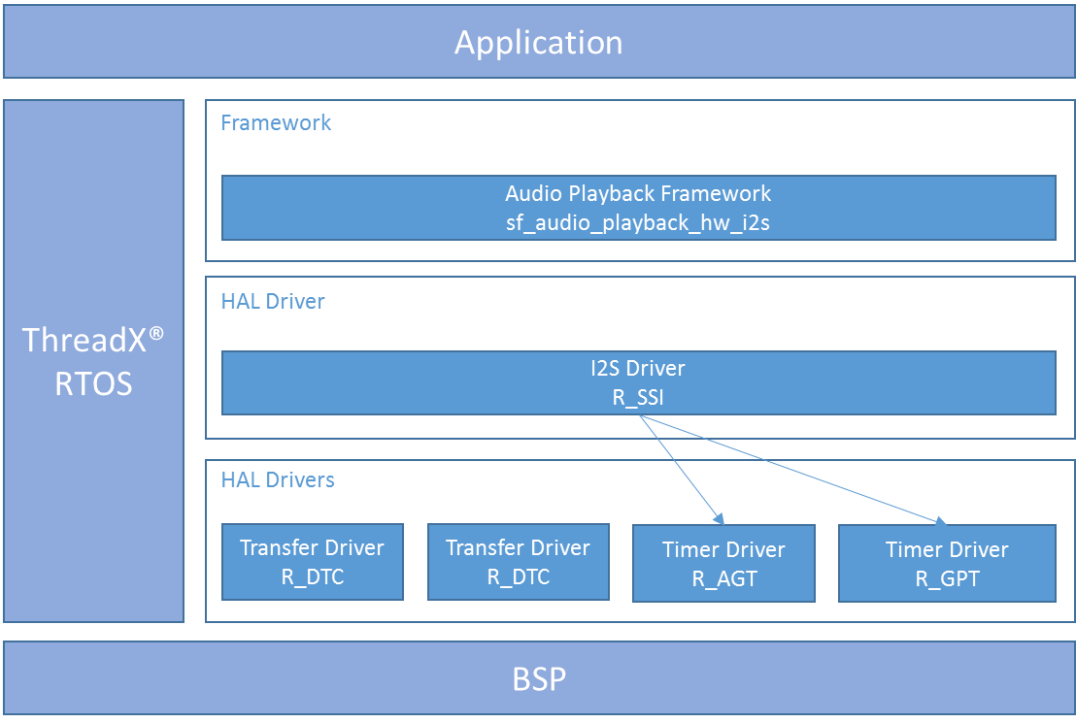


Figure 121: Audio Playback Framework Stack Options

4.1.6.2 Audio Playback I2S Framework APIs Overview

The Audio Playback I2S Framework module defines APIs for operations such as opening, starting, playing, and stopping. A complete list of the available APIs, an example API call, and a short description of each can be found in the table below. A table of status return values follows.

Audio Playback I2S Framework Module API Summary

Function Name	Example API Call and Description
.open	g_sf_audio_playback_hw0.p_api-> open (g_sf_audio_playback_hw0.p_ctrl, g_sf_audio_playback_hw0.p_cfg); Open a device channel for read/write and control.
.start	g_sf_audio_playback_hw0.p_api-> start(g_sf_audio_playback_hw0.p_ctrl); Start Audio Playback Hardware.
.stop	g_sf_audio_playback_hw0.p_api-> stop(g_sf_audio_playback_hw0.p_ctrl); Stop Audio Playback Hardware.

Function Name	Example API Call and Description
.play	g_sf_audio_playback_hw0.p_api-> stop(g_sf_audio_playback_hw0.p_ctrl, p_buffer, length); Play audio buffer.
.dataTypeGet	g_sf_audio_playback_hw0.p_api-> dataTypeGet(g_sf_audio_playback_hw0.p_ctrl, p_data_type); Stores expected data type in provided pointer.
.close	g_sf_audio_playback_hw0.p_api-> close(g_sf_audio_playback_hw0.p_ctrl); Close the audio driver.
.versionGet	g_sf_audio_playback_hw0.p_api-> versionGet(&version); Return the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manuals API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_OUT_OF_MEMORY	The number of streams open at once is limited to SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS. If this number is exceeded, an out of memory error occurs.
SSP_ERR_TIMEOUT	Timeout occurred before playback finished.

NOTE: Lower level drivers may return Common Error Codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.6.3 Audio Playback I2S Framework Module Operational Overview

The Audio Playback I2S Framework module creates a thread internally to support audio playback. The figure below shows a flowchart of the audio playback framework thread and its interactions with public Audio Playback Framework APIs.

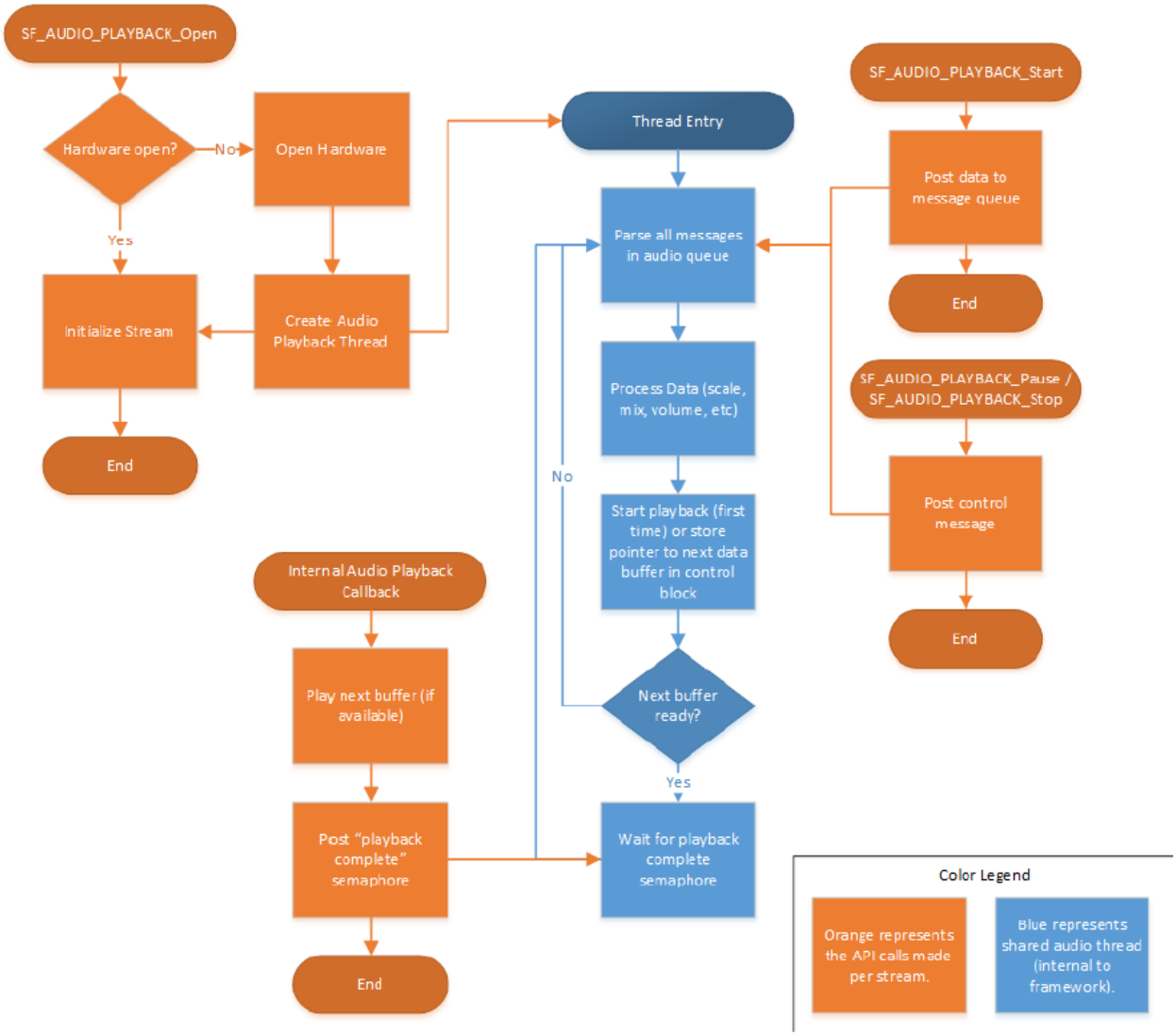


Figure 122: Audio Playback Framework Flowchart

Suggested use of the audio playback framework:

- Create a semaphore (for example, g_sf_audio_playback_semaphore). This can be done on the **Threads** tab. Set the initial value to 2 (the audio playback framework can store up to two data messages per stream).
- Create a callback function (for example, sf_audio_playback_callback). Enter the name of your callback function in the Audio Playback Framework instance. The callback function will be called when the audio playback framework is done with the data. In the callback, put the semaphore created above.
- In your main loop, get the semaphore before playing data. To play data, first acquire a buffer from the messaging framework, then create your audio playback data structure inside the buffer.The Audio Playback Framework

supports multiple audio streams on a single hardware port. A block diagram of the modules required if two streams are used is shown in the figure below.

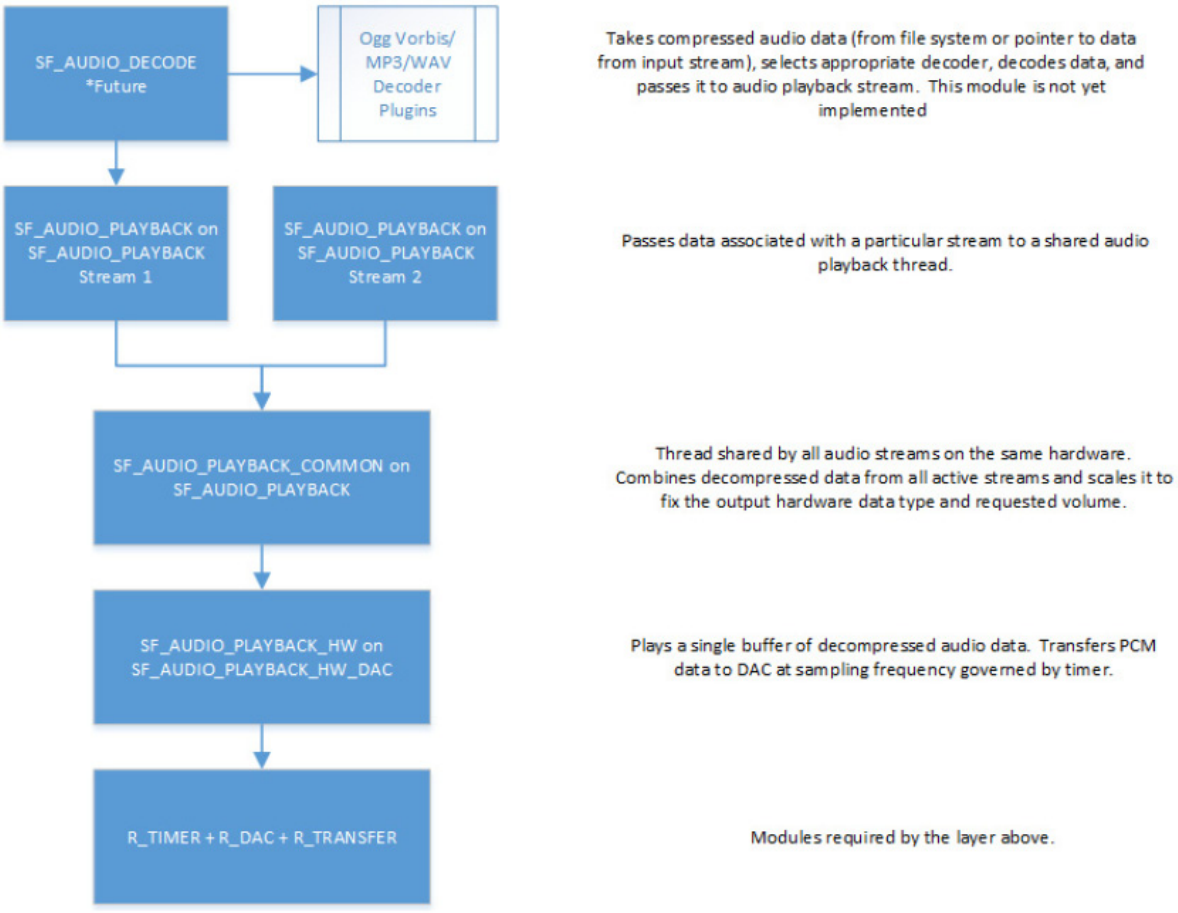


Figure 123: Implementing Multiple Audio Streams

Audio Playback I2S Framework Module Operational Notes

- The Queue used must match the name specified in **Properties** for Audio Playback Framework Shared on sf_audio_playback (default is g_sf_audio_playback_queue).
- The audio framework I2S hardware port has dependencies on the I2S Driver module. The I2S driver module can be accelerated with DTC (recommended).
- I2S driver module.
  - Set the Audio Clock Frequency (Hertz) to the frequency of the input audio clock used.
  - Set the Sampling Frequency (Hertz) to the sampling frequency of your audio data.
  - Set the Data Bits and Word Length to 16 bits (audio framework accepts 16 bit samples only).
  - Enable the SSIn TXI and SSIn INT interrupts.

- Transfer module on r_dtc (recommended).
- Set the Activation source to the SSIn TXI interrupt.
- The Audio Playback I2S Framework is designed to support the following MCU families with no changes to the API:
  - S7G2
  - S5D9, S5D5
  - S3A3, S3A6, S3A7, S3A1

Audio Playback I2S Framework Module Limitations

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.6.4 Including the Audio Playback I2S Framework Module in an Application

This section describes how to include the Audio Playback I2S Framework module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the *Getting Started Guide for SSP* listed in the Reference Section at the end of this document to learn how to manage each of these important steps in creating SSP based applications.

To add the Audio Playback I2S Framework module to an application, simply add it to a thread using the Stacks Selection Sequence given in the table below. (The default name for the Audio Playback is g_sf_audio_playback_hw0. This name can be changed in the associated **Properties** window.)

Audio Playback I2S Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_audio_playback_hw0 Audio Playback Hardware Framework on g_sf_audio_playback_hw0	**Threads**	**New Stack> Framework> Audio> Audio Playback Hardware Framework on g_sf_audio_playback**

When the Audio Playback I2S Framework module on sf_audio_playback_hw_i2s is added to the thread stack as shown in the figure below, the configurator automatically adds any needed lower level modules. Any drivers that need additional configuration information will have text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower level drivers. These are either optional or recommended (this is indicated in the block with the inclusion of this text.) If the addition of lower level drivers is required, the module description includes "Add" in the text. Clicking on any Pink banded modules brings up the "New" icon and then display the possible choices.

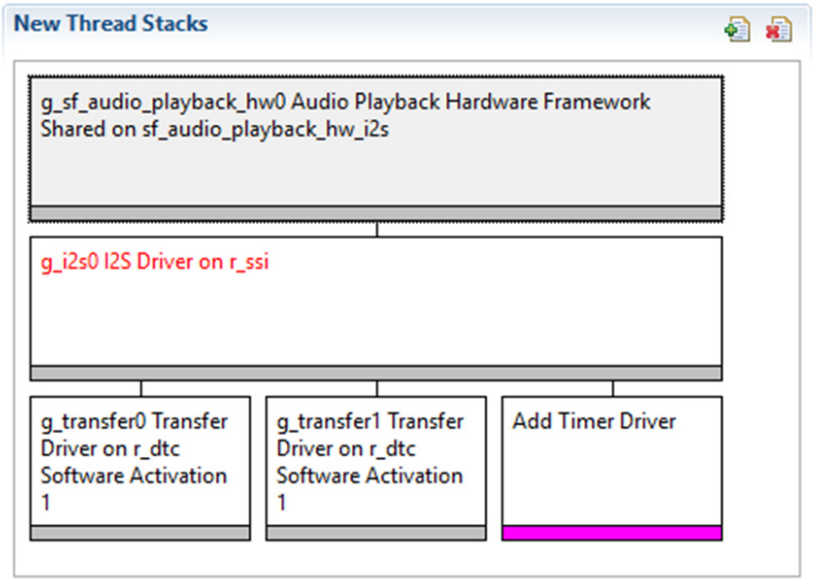


Figure 124: Audio Playback I2S Framework Stack

4.1.6.5 Configuring the Audio Playback I2S Framework Module

The Audio Playback I2S Framework module must be configured by you for the desired operation. The SSP configuration window automatically identifies, by highlighting the block in red, any required configuration selections, such as Interrupts or operating modes, which must be configured for lower level modules, for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the **Properties** tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority. This configuration setting is available with the **Properties** window of the associated module. Simply select the indicated module and then view the **Properties** window. The Interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the Interrupt priorities listed in the **Properties** window in the ISDE includes an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring Interrupt Priority levels.

NOTE: You may want to open your ISDE and create the module and explore the property settings in parallel with looking over the Configuration Table Settings that follow. This will help orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Audio Playback I2S Hardware Framework Module on sf_audio_playback_hw_i2s

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_sf_audio_playback_hw0	Module name.

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower level modules can be desirable. For example, it might be useful to select the DAC or I2S Channel based on the target hardware implementation. The configurable properties for the lower level stack modules are given in the below sections for completeness and as a reference.

NOTE: Most of the property settings for lower level modules are fairly intuitive and can usually be determined by inspection of the associated **Properties** window from the SSP configurator.

4.1.6.6 Configuration Settings for the Audio Playback Framework Low Level Modules

Typically, only a small number of settings must be modified from the default for lower level modules. (These are indicated with red text in the thread stack block.) Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The tables below identify all the settings within the properties section for the module.

Configuration Settings for the I2S HAL Module on r_ssi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_i2s0	Module name.
Channel	0	Physical hardware channel.
Audio Clock Frequency (Hertz)	2,822,400	Input audio clock frequency, used to generate the I2S clock. Must be a multiple between 1 and 128 of: (sampling_freq_hz * word_length_in_bits)

ISDE Property	Value	Description
Sampling Frequecy (Hertz)	44,100	Sampling frequency of audio data.
Data Bits	8 bits, 16, 18, 20, 22, 24 Default: 16 bits	Bit depth of audio data, which is the size in bits of one sample of audio data.
Word Length	8 bits, 16, 24, 32 Default: 16 bits	Word length of audio data, must be at least the same size as the bit depth (Data Bits field).
WS Continue Mode	Enabled, Disabled Default: Disabled	Enable WS continue mode to continue to output the word select line when the peripheral is idle. Disable to stop outputting the word select line when the peripheral is idle.
Name of I2S callback function to be defined by user	NULL	A user callback function must be registered in open. The callback will be called from the interrupt service routine (ISR) when the transmission FIFO reaches the high watermark point after all data for transmission is transmitted or when reception is complete (the requested number of bytes have been received). ATTENTION: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Transmit interrupt priority selection

ISDE Property	Value	Description
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Receive interrupt priority selection
Idle/Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Idle/error interrupt priority selection

Configuration Settings for the DTC HAL Module on r_dtc Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer0	Driver name.
Mode	Normal	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.
Source Address Mode	Incremented	Source address mode selection.

ISDE Property	Value	Description
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activiation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	FALSE	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection

Configuration Settings for the DTC HAL Module on r_dtc Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.

ISDE Property	Value	Description
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer1	Driver name.
Mode	Normal	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activiation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	FALSE	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.

ISDE Property	Value	Description
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection

Configuration Settings for AGT HAL Module on r_agt

ISE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection
Name	g_timer0	Module name
Channel	0	Channel selection
Mode	Periodic	Mode selection
Period Value	2,822,400 *2	Period value selection
Period Unit	Hertz	Period unit selection
Auto Start	FALSE	Auto start selection
Count Source	PCLKB, PCLKB/8, PCLKB/2, LOCO, AGT0 Underflow, AGT0 fSub Default: PCLKB	Count source selection
AGTO Output Enabled	True, False Default: False	AGTO output selection
AGTIO Output Enabled	True, False Default: False	AGTIO output selection

ISE Property	Value	Description
Output Inverted	True, False Default: False	Output inverted selection
Callback	NULL	Callback selection
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection

Configuration Settings for the GPT HAL Module on r_gpt

ISE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection
Name	g_timer0	Module name
Channel	0	Channel selection
Mode	Periodic	Mode selection
Period Value	2,822,400 *2	Period value selection
Period Unit	Hertz	Period unit selection
Duty Cycle Value	50	Duty cycle value selection
Duty Cycle Unit	Unit Raw Counts, Unit Percent, Unit Percent x 1000 Default: Unit Raw Counts	Duty cycle unit selection
Auto Start	FALSE	Auto start selection

ISE Property	Value	Description
GTIOCA Output Enabled	True, False Default: False	GTIOCA output enabled selection
GTIOCA Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	GTIOCA stop level selection
GTIOCB Output Enabled	True, False Default: False	GTIOCB output enabled selection
GTIOCB Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	GTIOCB stop level selection
Callback	NULL	Callback selection
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

4.1.6.7 Audio Playback I2S Framework Module Clock Configuration

The Audio Playback Framework hardware modules (I2S) use the peripheral clocks available from the Clock configuration window.

4.1.6.8 Audio Playback I2S Framework Module Pin Configuration

The SSI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The first table below illustrates the method for selecting the pins within the SSP configuration window and the following table illustrates an example selection for the associated pins.

Pin Selection Sequence for ADC or I2S

Resource	ISDE Tab	Pin selection Sequence
I2S	Pins	Select **Peripherals > Connectivity:SSI > SSI/SSI0/SSI1**

NOTE: The above selection sequence assumes ADC0 or ADC1, or SSI0 or SSI1 is the desired hardware target for the driver.

Pin Configuration Settings for I2S driver on SSI

Pin Configuration Property	Value	Description
Pin Group Selection	_A only, _B only, Mixed	Pin group for I2S port
Operation Mode	Enabled, Custom, Disabled	Operation selection
SSISCK	None, P204 (Default: None)	SSI Serial Clock
SSIWS	None, P205 (Default: None)	SSI Stereo pin selection
SSIDATA	None, P206 (Default: None)	SSI Data pin selection

NOTE: The above example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.1.6.9 Audio Playback I2S Framework Module Messaging Configuration

Use the Messaging Framework configurator on the **Messaging** tab to configure the messaging framework:

1) Highlight the Audio Playback event class.
2) Add a new subscriber.
3) Select the following configurations and make sure the Message Class Instance property set in the **Properties** tab of the Audio Playback Framework on sf_audio_playback module is between the Start and End instance.
    a) Thread: Any thread in the application.
    b) Start: First audio instance used in application.
    c) End: Last audio instance used in application.
4) Highlight the new Subscriber in the Audio Playback Subscribers. Record the Symbol name.
5) Go back to the **Threads** tab.
6) Highlight the Audio Playback Framework Shared module in HAL/Common, and set the Audio Message Queue Name to the Symbol name from the Audio Playback Subscriber.

4.1.6.10 Using the Audio Playback I2S Framework Module in an Application

The typical steps in using the Audio Playback I2S Framework module in an application are:

1) Initialize the Audio Playback I2S Framework using the open API
2) Start the low-level hardware of the Audio Playback I2S framework using the start API.
3) Play the PCM audio samples using the play API.
4) PCM audio samples supported by the hardware is provided by the dataTypeGet API.
5) Stop the low-level hardware of the Audio Playback I2S Framework using stop API.
6) Close the Audio Playback I2S framework using the close API.

These common steps are illustrated in a typical operational flow diagram in the figure below:

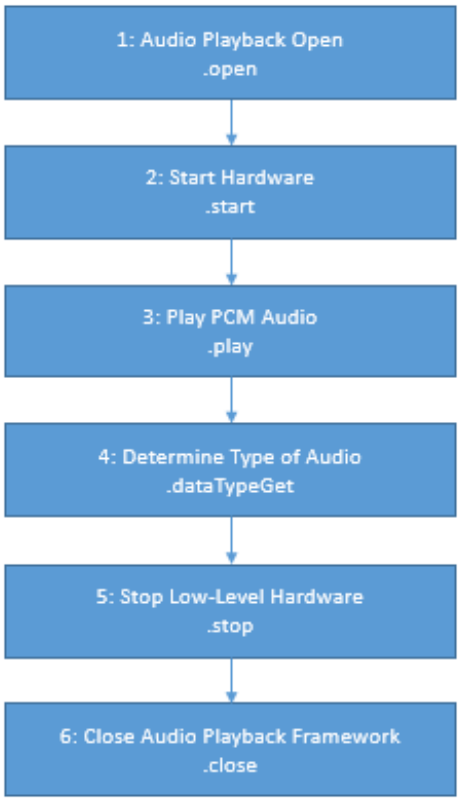


Figure 125: Flow Diagram of a Typical Audio Playback Framework Module Application

4.1.7 Block Media QSPI Framework

The Block Media Framework Module provides high-level APIs for a QSPI channel and supports read, write, and control of the QSPI Flash memory peripheral through the r_qspi driver. The driver has all the functionality needed to interface with a file system through a block media interface.

4.1.7.1 Block Media QSPI Framework Module Features

- Supports the QSPI channel interface for a QSPI flash memory device.
- Supports a file system on QSPI flash memory.

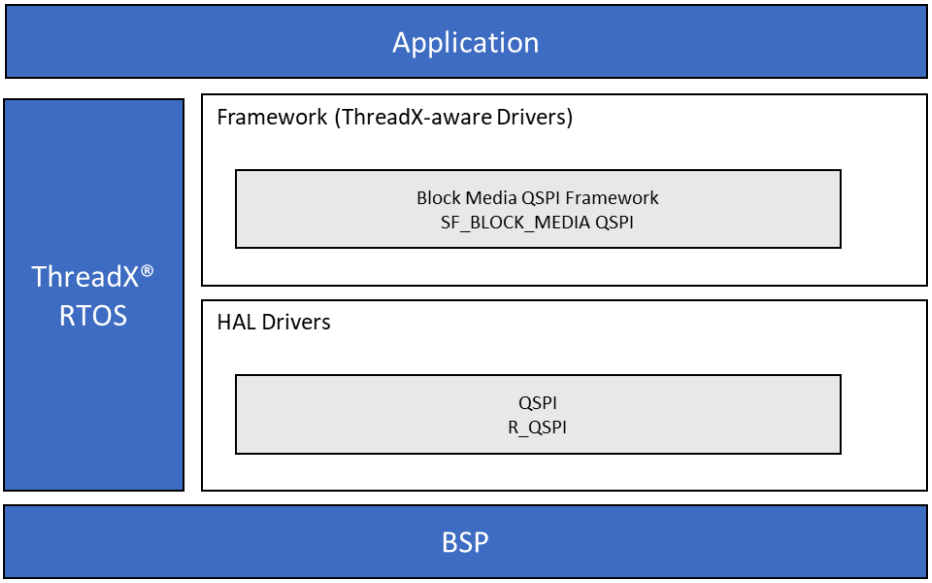


Figure 126: Block Media QSPI Framework Module Block Diagram

4.1.7.2 Block Media QSPI Framework Module APIs Overview

The Block Media QSPI Framework module defines APIs to open, control, read and write to the media. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Block Media QSPI Framework Module API Summary

Function Name	Example API Call and Description
.open	g_sf_block_media_qspi0.p_api->open (g_sf_block_media_qspi0.p_ctrl, g_sf_block_media_qspi0.p_cfg); Open a device channel for read/write and control.
.read	g_sf_block_media_qspi0.p_api->read (g_sf_block_media_qspi0.p_ctrl, p_dest, start_block, block_count); Read data from a media channel.

Function Name	Example API Call and Description
.write	g_sf_block_media_qspi0.p_api->write (g_sf_block_media_qspi0.p_ctrl, p_src, start_block, block_count); Write data to a media channel.
.ioctl	g_sf_block_media_qspi0.p_api->ioctl (g_sf_block_media_qspi0.p_ctrl, command, p_data); Send control commands to and receives the status from the media port.
.close	g_sf_block_media_qspi0.p_api->close (g_sf_block_media_qspi0.p_ctrl); Close the open media channel.
.versionGet	g_sf_block_media_qspi0.p_api->versionGet(&version); Return the version of the driver using the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_INTERNAL	An internal TheadX error has occurred.
SSP_ERR_NOT_OPEN	Unit is not open
SSP_ERR_ASSERTION	The parameter p_ctrl or p_sample is NULL.
SSP_ERR_IN_USE	Peripheral is still running in another mode; perform Close first.
SSP_ERR_UNSUPPORTED	Command not supported.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.1.7.3 Block Media QSPI Framework Module Operational Overview

The Block Media Framework Interface is simply an abstract interface using function pointers instead of direct function calls. Functions are called between FileX and the SSP block media drivers, such as SDMMC, SPI Flash and SDRAM/RAM. The interface remains the same for any media driver so that all media drivers appear functionally identical at file I/O layer and can be interchanged with one another without changing code. Device adaptation drivers, such as sf_block_media_qspi, are accessed through the Block Media Framework Interface and provide device specific code needed to perform media I/O operations. Configuration and control structures passed through block media function calls are generally device specific as well.

Block Media QSPI Framework Module Important Operational Notes and Limitations

The media must be formatted at least once before you can begin creating, writing, and reading files.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.7.4 Including the Block Media QSPI Framework Module in an Application

This section describes how to include the Block Media QSPI Framework module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Block Media QSPI Framework module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the Block Media QSPI Framework module is g_sf_block_media_qspi0. This name can be changed in the associated Properties window.)

Block Media QSPI Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_block_media_qspi0 Block Media Framework on sf_block_media_qspi	Threads	New Stack> Framework> File System> Block Media Framework on sf_block_media_qspi

When the Block Media QSPI Framework module on sf_block_media_qspi is added to the thread stackas shown in the following figure, the configurator automatically adds any lower-level drivers that are required. Any drivers that need additional configuration information will be box-text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

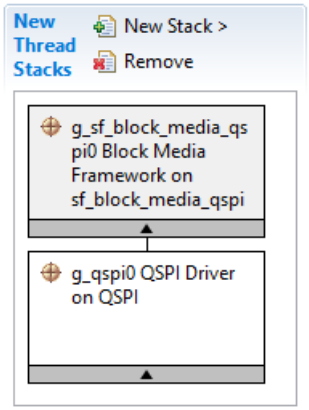


Figure 127: Block Media QSPI Framework Module Stack

4.1.7.5 Configuring the Block Media QSPI Framework Module

The Block Media QSPI Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the Block Media QSPI Framework Module on sf_block_media_qspi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_sf_block_media_qspi	Module name.

ISDE Property	Value	Description
Block size of media in bytes	4096	Block size selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the QSPI HAL Module on r_qspi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_qspi0	Module name.

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Block Media QSPI Framework Module Clock Configuration

The Block Media QSPI Framework module uses the PCLKA as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

Block Media QSPI Framework Module Pin Configuration

To use the Block Media QSPI Framework module, the port pins for the peripheral inputs and outputs must be set in the pin configurator in the ISDE. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection Sequence for the Block Media QSPI Framework Module

Resource	ISDE Tab	Pin selection Sequence
QSPI	Pins	Select Peripherals > Storage:QSPI > QSPI0

4.1.7.6 Using the Block Media QSPI Framework Module in an Application

The typical steps for sf_block_media_qspi framework in an application are:

1) Initialize media using the open API
2) Read media as required using the read API

3) Write media as required using the write API
4) Operate on data as required.
5) Close media as required using the close API

These common steps are illustrated in a typical operational flow in the following figure:

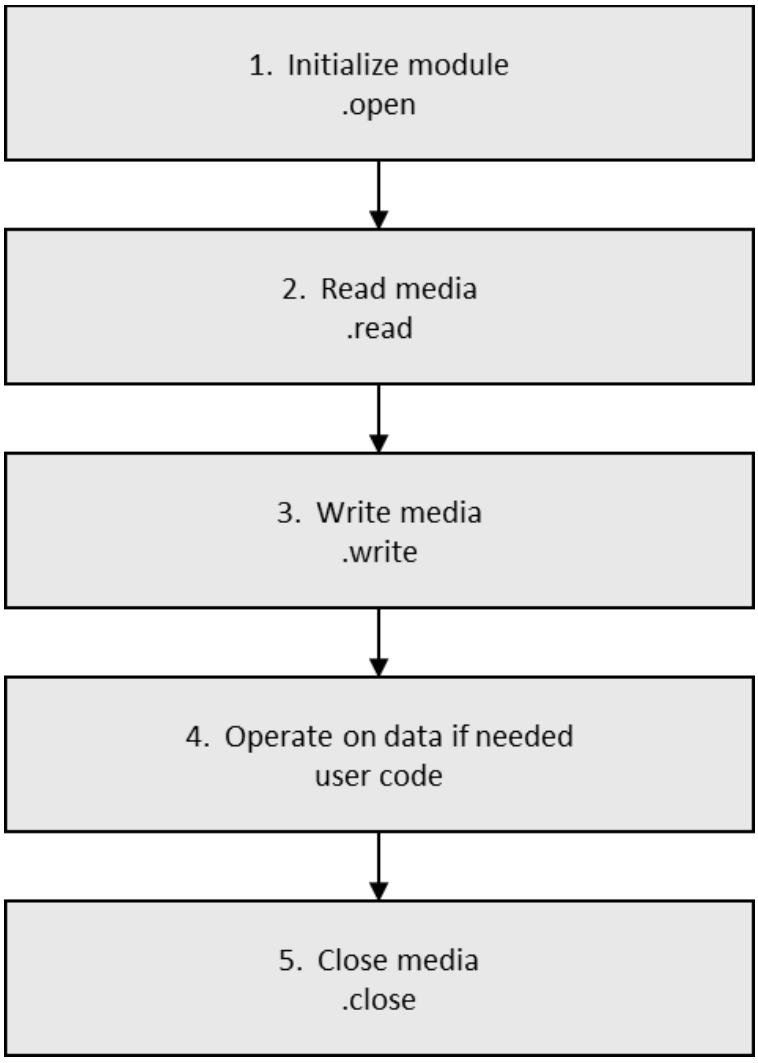


Figure 128: Flow Diagram of a Typical Block Media QSPI Framework Module Application

The typical steps for sf_block_media_qspi using sf_el_fx, in an application are:

1) Initialize media using the FileX API fx_system_initialize (sf_el_fx calls it automatically)
2) Format media using FileX API fx_media_format (sf_el_fx calls it automatically if "Format media during initialization" property is set to Enabled) This Format media is optional if media has already been formatted. Anything on the media will erase while media format.

3) Open the media using FileX API fx_media_open (sf_el_fx opens the media automatically)
4) Read media as required using the read API (Block Media Framework) or one of the FileX API e.g. fx_media_read(), fx_file_read()
5) Write to media as required using the write API (Block Media Framework) or one of the FileX API e.g. fx_media_write(), fx_file_write()

NOTE: After successful fx_media_open call, all FileX APIs can be used, not only read and write.

These common steps are illustrated in a typical operational flow in the following figure:

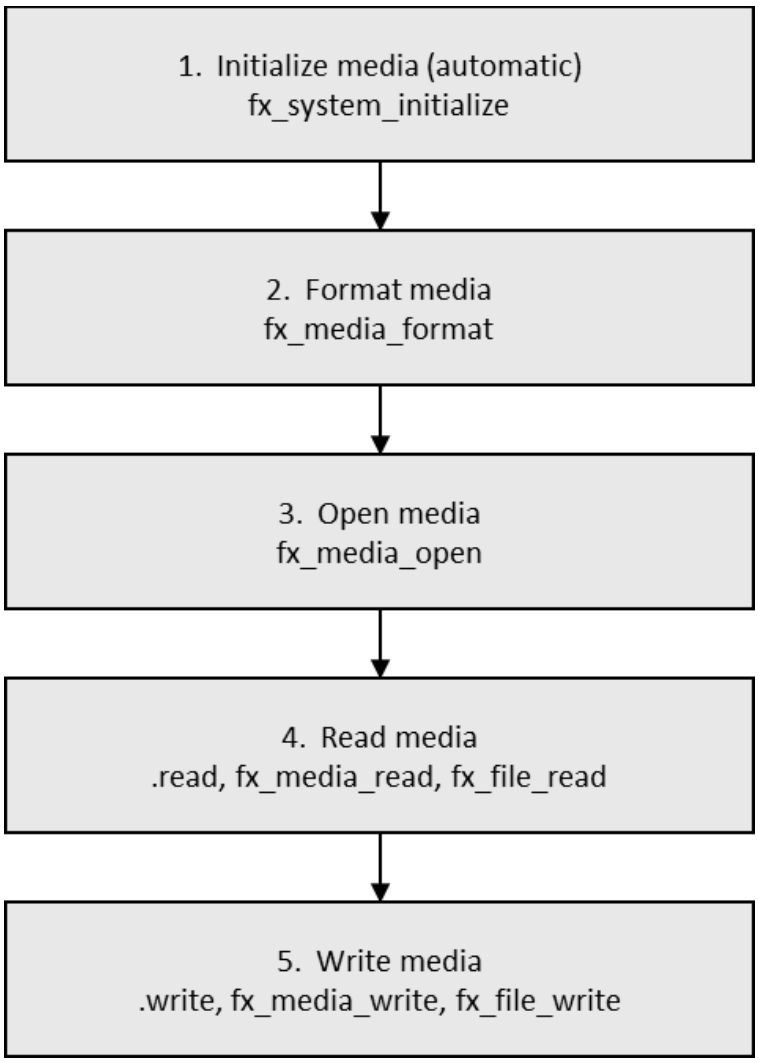


Figure 129: Flow Diagram of a Typical Block Media QSPI Framework Module Application

4.1.8 Block Media RAM Framework

The Block Media Framework Module implements a file system on RAM for read, write, and control of the read/write region of RAM memory. The framework has all the functionality needed to interface with a file system through a block media interface.

4.1.8.1 Block Media RAM Framework Module Features

- Enable FileX to be run on linear memory-mapped devices.
- Temporary and fast storage of data on RAM.

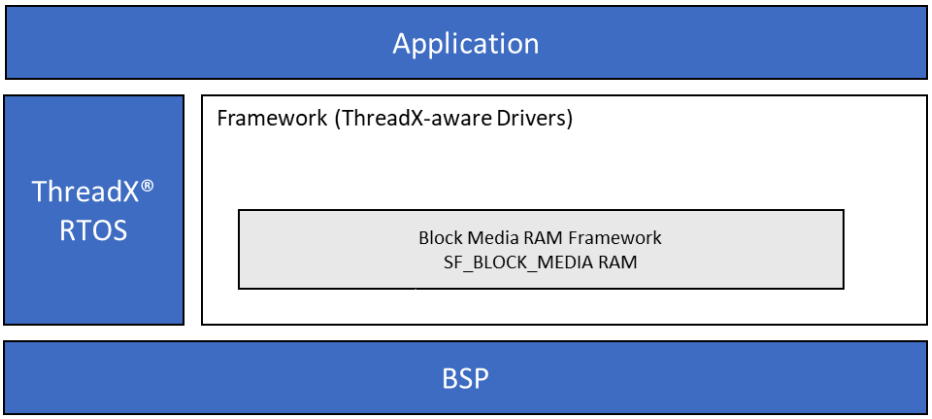


Figure 130: Block Media RAM Framework Module Block Diagram

4.1.8.2 Block Media RAM Framework Module APIs Overview

The Block Media RAM Framework module implements APIs to open, read, write and close the module. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Block Media RAM Framework Module API Summary

Function Name	Example API Call and Description
.open	g_sf_block_media_ram0.p_api->open (g_sf_block_media_ram0.p_ctrl, g_sf_block_media_ram0.p_cfg); Open device for read, write and control.

Function Name	Example API Call and Description
.read	g_sf_block_media_ram0.p_api->read (g_sf_block_media_ram0.p_ctrl, p_dest, start_block, block_count); Read data from RAM buffer.
.write	g_sf_block_media_ram0.p_api->write (g_sf_block_media_ram0.p_ctrl, p_source, start_block, block_count); Write data to RAM buffer.
.ioctl	g_sf_block_media_ram0.p_api->ioctl (g_sf_block_media_ram0.p_ctrl, command, p_data); Send control commands to and receive status of RAM buffer.
.close	g_sf_block_media_ram0.p_api->close (g_sf_block_media_ram0.p_ctrl); Close the framework.
.versionGet	g_sf_block_media_ram0.p_api->versionGet(&version); Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_UNSUPPORTED	Command not supported.
SSP_ERR_NOT_OPEN	Unit is not open
SSP_ERR_ASSERTION	A parameter is NULL.

Name	Description
SSP_ERR_IN_USE	Framework is already open.
SSP_ERR_INVALID_BLOCKS	Invalid block passed.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.1.8.3 Block Media RAM Framework Module Operational Overview

The Block Media Framework Interface is simply an abstract interface using function pointers instead of direct function calls. Functions are called between FileX and the SSP block media drivers, such as SDMMC, SPI Flash and SDRAM/RAM. The interface remains the same for any media driver so that all media drivers appear functionally identical at file I/O layer and can be interchanged with one another without changing code. Device adaptation drivers, such as sf_block_media_ram, are accessed through the Block Media Framework Interface and provide device specific code needed to perform media I/O operations. Configuration and control structures passed through block media function calls are generally device specific as well.

Block Media RAM Framework Module Important Operational Notes and Limitations

The media must be formatted before you can open the media and begin creating, writing, and reading files.

The memory area used must be directly readable and writable by the core. Internal SRAM or SDRAM size should be sufficient for the block media RAM and there will be no persistence of data across the power cycles.

Block count must be assigned by considering the boot record, FAT area, root directory and directory sector. If the File System is used on RAM then the minimum block count must be four, as the first three sectors are reserved for boot record, FAT area and root directory and the fourth sector is used for data sector.

The "Format media during initialization" property needs to be enabled in order to use RAM for FileX.

The "File System is on block media" property needs to be true in order to configure the block count and block size from the block media RAM.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.8.4 Including the Block Media RAM Framework Module in an Application

This section describes how to include the Block Media RAM Framework module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Block Media RAM Framework module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the Block Media RAM Framework module is g_sf_block_media_ram0. This name can be changed in the associated Properties window.)

Block Media RAM Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_ sf_block_media_ram0 Block Media Framework on sf_block_media_ram	Threads	New Stack> Framework> File System> Block Media Framework on sf_block_media_ram

When the Block Media RAM Framework module on sf_block_media_ram is added to the thread stack, as shown in the following figure, the configurator automatically adds any lower-level drivers that are required. Any drivers that need additional configuration information will be box-text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

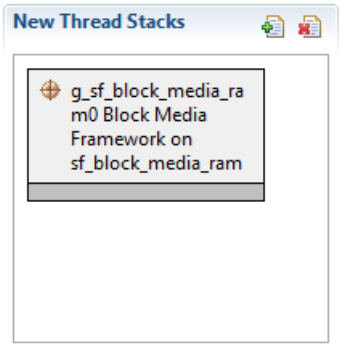


Figure 131: Block Media RAM Framework Module Stack

4.1.8.5 Configuring the Block Media RAM Framework Module

The Block Media RAM Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the Block Media RAM Framework Module on sf_block_media_ram

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: Enabled	If selected code for parameter checking is included in the build.
Name	g_sf_block_media_ram0	Module name
Block size of media in bytes	512	Specify the size of the block used on the target media
Number of blocks to allocate	16	Specify the number of blocks to allocate on the target media. The target must have sufficient storage to support the total number of bytes specified.
Enter the valid section for RAM buffer allocation	noint	Specify the section of RAM to be used as the target media.

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Block Media RAM Framework Module Clock Configuration

The Block Media RAM Framework module uses the ICLK as the source for the internal RAM.

Block Media RAM Framework Module Pin Configuration

The Block Media RAM Framework module uses internal RAM so no external pins are required.

4.1.8.6 Using the Block Media RAM Framework Module in an Application

The "Format media during initialization" property needs to be enabled in order to use RAM for FileX.

The "File System is on block media" property needs to be true in order to configure the block count and block size from the block media RAM.

The typical steps for sf_block_media_ram framework in an application are:

1) Initialize media using the open API
2) Read media as required using the read API
3) Write media as required using the write API
4) Operate on data as required.
5) Close media as required using the close API

These common steps are illustrated in a typical operational flow in the following figure:

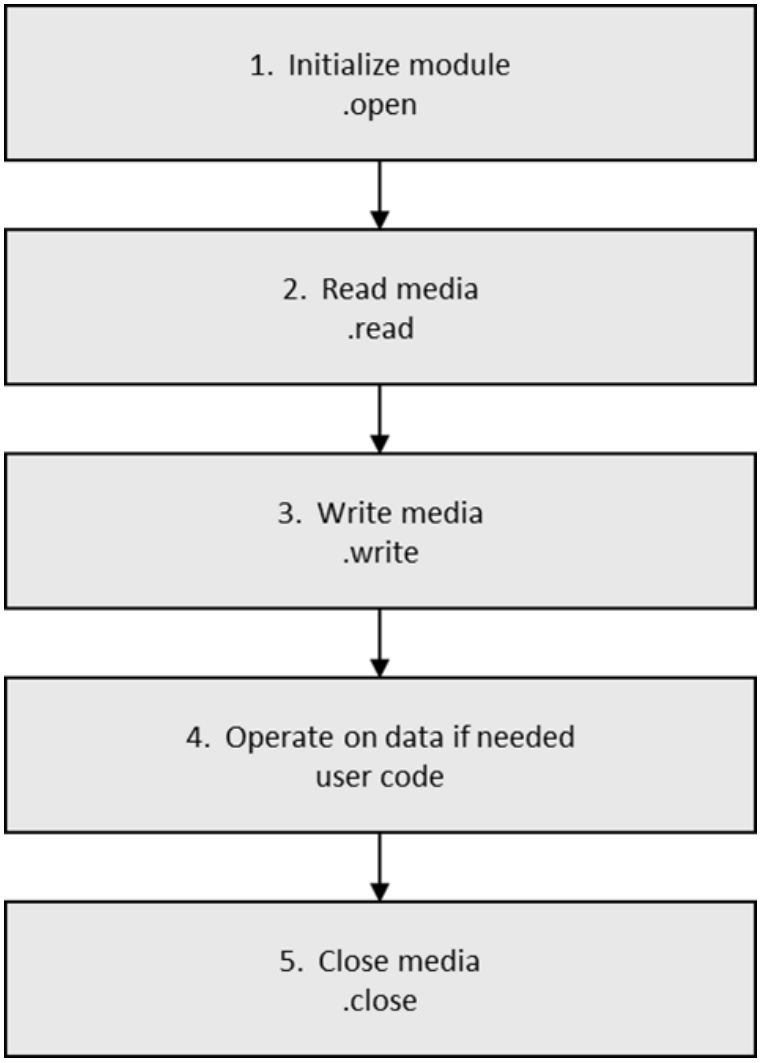


Figure 132: Flow Diagram of a Typical Block Media RAM Framework Module Application

The typical steps for sf_block_media_ram using sf_el_fx, in an application are:

1) Initialize media using the FileX API fx_system_initialize (Synergy generated code calls it automatically)
2) Format media using FileX API fx_media_format (Synergy generated code calls it automatically if "Format media during initialization" property is set to Enabled)
3) Open the media using FileX API fx_media_open (Synergy generated code opens the media automatically if "Auto Initialization" property is set to Enable)
4) Read media as required using the read API (Block Media Framework) or one of the FileX APIs, for example, fx_media_read(), fx_file_read()Write to media as required using the write API (Block Media Framework) or one of the FileX APIs, for example, fx_media_write(), or fx_file_write()

NOTE: After successful fx_media_open call, all FileX APIs can be used, not only FileX read and FileX write.

These common steps are illustrated in a typical operational flow in the following figure:

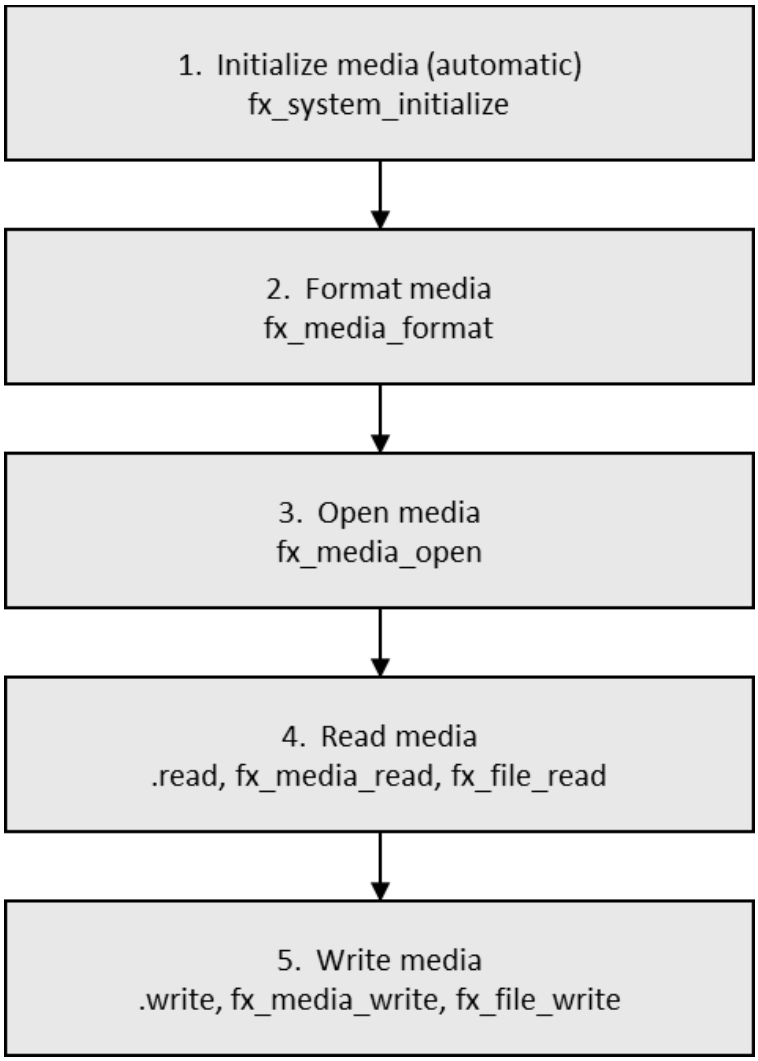


Figure 133: Flow Diagram of a Typical Block Media RAM Framework Module Application for FileX

4.1.9 BLE Framework

Bluetooth® Low Energy (BLE), sometimes referred to as **Bluetooth Smart**, is a light-weight subset of Classic Bluetooth and was introduced as part of the Bluetooth 4.0 core specification. In contrast to Classic Bluetooth, BLE is designed to provide significantly lower power consumption. This allows Internet of Thing (IoT) devices that have stricter power capacity to transfer small amounts of data between nearby devices.

Application developers access the functionality provided by the BLE stack using its APIs. The BLE stack APIs provided by different vendors are not standardized. This results in application developers having to update their code when porting to different BLE stacks.

The Synergy BLE Framework handles this issue by providing a generic interface for the underlying BLE stack provided by various vendors there by preventing coupling between application and vendor-specific BLE stack code. The use of generic APIs makes application development simpler and portable.

The BLE framework provides high-level API for BLE applications, and is implemented as sf_ble_rl78g1d. The BLE framework uses the Synergy Software Package (SSP) communication framework which in turn enables UART driver for communication to the underlying BLE module. It also integrates the generic BLE profile framework (g_sf_ble_onboard_profile) which provides a uniform interface to BLE profiles. For the RL78G1D BLE hardware module, the generic BLE profiles are implemented by the BLE module firmware.

4.1.9.1 Features

The Synergy BLE framework supports the following features:

- ThreadX® RTOS Aware and thread safe
- Bluetooth v4.2 compliant framework.
- Generic Access Profile (GAP) Features
  - User-defined advertising data
  - Security modes 1 and 2
  - Peripheral and central roles
  - White list support up to 6 devices
  - Bonding support
- Generic Attribute Profile (GATT) features
  - GATT client and server
- Generic Attribute Profile (GATT) APIs
- Generic Access Profile (GAP) APIs
- Generic On-board Profiles APIs

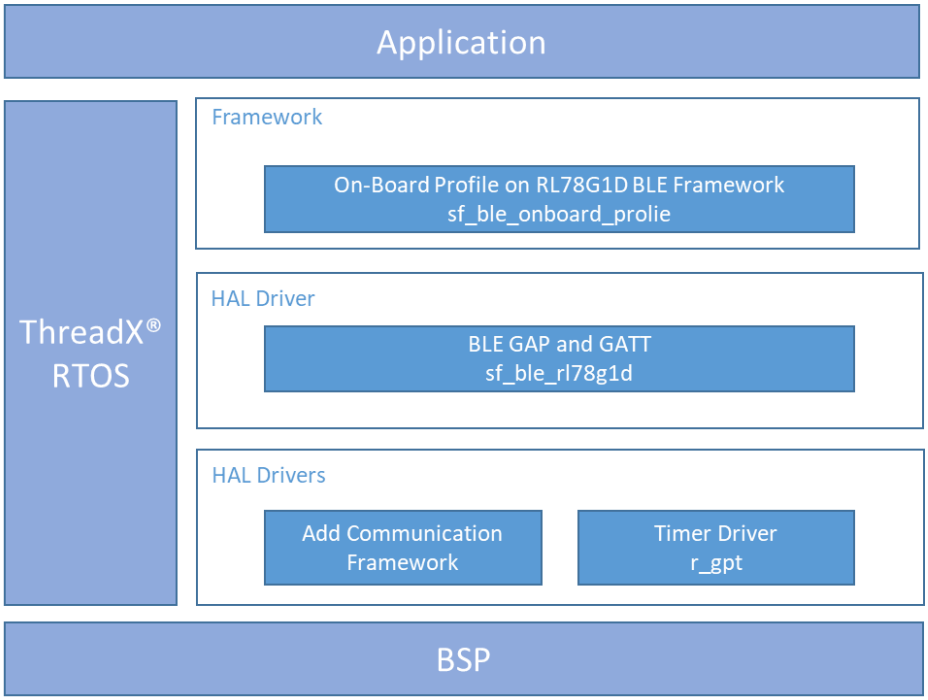


Figure 134: BLE Framework Module Organization, Options and Stack Implementations

4.1.9.2 BLE Framework Module APIs Overview

The BLE Framework defines API's for initializing, setting and getting values and stopping modules. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

BLE Framework Module On-Board Profiles API Summary

Function Name	Example API Call and Description
.open	g_sf_ble_onboard_profile0.p_api->open (g_ sf_ble_onboard_profile0.p_cfg); This API initializes the interface for data transfers.
.close	g_sf_ble_onboard_profile0.p_api->close (g_ sf_ble_onboard_profile0.p_cfg); This API de-initializes the interface and may put it in low power mode or power it off. The API closes the driver, and disables the driver link and interrupt.

Function Name	Example API Call and Description
.onbpEnable	g_sf_ble_onboard_profile0.p_api->onbpEnable (sf_ble_onboard_profile0.p_cfg, p_handle, profile, p_prf_cb, sec); Enables the profile in server mode or client mode.
.onbpServerWriteData	g_sf_ble_onboard_profile0.p_api->onbpServerWriteData (sf_ble_onboard_profile0.p_cfg,p_handle, profile, characteristics, p_data); Updates the value of the characteristic in the local database.
.onbpServerSendNotification	g_sf_ble_onboard_profile0.p_api->onbpServerSendNotification (sf_ble_onboard_profile0.p_cfg, p_handle, profile, characteristics, p_data); Sends notifications.
.onbpServerSendIndication	g_sf_ble_onboard_profile0.p_api->onbpServerSendIndication (sf_ble_onboard_profile0.p_cfg, p_handle, profile, characteristics, p_data); Sends indications.
.onbpClientWriteCCCD	g_sf_ble_onboard_profile0.p_api->onbpClientWriteCCCD (sf_ble_onboard_profile0.p_cfg, p_handle, profile, cccd_char, cccd_val); Sets the Client Configuration Control Descriptor on the remote device.
.onbpDisable	g_sf_ble_onboard_profile0.p_api->onbpDisable (sf_ble_onboard_profile0.p_cfg, p_handle, profile); Disables the profile in server mode and client mode.

Function Name	Example API Call and Description
.onbpClientReadChar	g_sf_ble_onboard_profile0.p_api->onbpClientReadChar (sf_ble_onboard_profile0.p_cfg, p_handle, profile, characteristics); Reads a GATT characteristic associated with the profile or service.
.opbClientWriteChar	g_sf_ble_onboard_profile0.p_api->onbpClientWriteChar (sf_ble_onboard_profile0.p_cfg, p_handle, profile, characteristics); Writes a GATT characteristic associated with the profile or service.
.versionGet	g_sf_ble_onboard_profile0.p_api-> versionGet(&version); Retrieve the API version using the version pointer.

NOTE: 1. All the details related to BLE standard profiles can found in BLE profile specifications.

NOTE: 2. BLE APIs will return "SSP_ERR_UNSUPPORTED" if module does not support that feature.

BLE Framework Module GAP API Summary

Function Name	Example API Call and Description
.open	g_sf_ble0.p_api->open(g_ sf_ble_0.p_cfg); This function initializes and enables the BLE module. It accepts the BLE module configuration as an argument, which has the following parameters: - 48-bit Bluetooth Address - Device scan interval - Device scan window - Device discoverable time - Device connection interval - Slave latency - Supervision timeout - Own address type - Maximum slaves allowed to be connected
.close	g_sf_ble0.p_api->close (g_ sf_ble0.p_cfg); This API de-initialize the interface and may put the BLE module in low power mode or power it off. It also closes the driver, disables the driver link, disable the interrupt in the BLE module driver.

Function Name	Example API Call and Description
.infoGet	g_sf_ble0.p_api->infoGet (g_sf_ble_0.p_cfg, p_handle, p_ble_info); The infoGet API takes the BLE control structure as an argument. It returns the following information obtained from the BLE module: - Chipset/driver information string - RSSI value (unsigned integer 16 bits)
.provisionGet	g_sf_ble0.p_api->provisionGet (g_sf_ble_0.p_cfg, p_ble_provisioning); The provisionGet API gets the BLE GAP provisioning information and takes the BLE control structure as an argument. It returns the following parameters: - Bonding Mode - Security Mode - GAP Role (Central/Master or Peripheral/Slave) - Security Keys

Function Name	Example API Call and Description
.provisionSet	g_sf_ble0.p_api->provisionSet (g_sf_ble_0.p_cfg, p_ble_provisioning); The provisionSet() function provisions BLE module. It takes BLE control structure and provisioning structure as an argument. - Bonding Mode - Security Mode - GAP Role (Central/Master or Peripheral/Slave) - Security Keys - GAP user event callback
.scan	g_sf_ble0.p_api->scan (g_sf_ble_0.p_cfg, p_scan, p_cnt, p_scan_info); The scan() function takes BLE control structure as an argument. Scan() function returns a list of BLE devices scanned by the BLE module with below parameters. - Bluetooth address (48-bits) - RSSI - Scan data - Advertising Event type The scan() function takes device count as an argument which acts as an in/out parameter. It specifies size of scan result array and BLE framework sets it to count indicating number of scan results stored in array. The function takes scan type as argument(Active/Passive).

Function Name	Example API Call and Description
.advertisementStart	g_sf_ble0.p_api->advertisementStart (g_sf_ble_0.p_cfg, p_advt_info); The advertisementStart() function takes following parameters: - Discovery mode (General/Limited) - Filter policy – Support for scan/connect request filtering combinations - Advertisement data - Connection mode - Advertisement intervals - Channel map - Address type - Advertising type - Scan response data
.advertisementStop	g_sf_ble0.p_api->advertisementStop (g_sf_ble_0.p_cfg); Stops advertisement.
.whitelistAdd	g_sf_ble0.p_api->whitelistAdd (g_sf_ble_0.p_cfg, p_bp_addr); The whitelistAdd() function add devices to the whitelist for advertisements, scan requests and connection requests.

Function Name	Example API Call and Description
.whitelistDel	g_sf_ble0.p_api->whitelistDel (g_sf_ble_0.p_cfg, p_bp_addr); The whitelistDel() function deletes devices from the whitelist for advertisements, scan requests and connection requests.
.bondingStart	g_sf_ble0.p_api->bondingStart (g_sf_ble_0.p_cfg, p_handle, p_bp_addr, p_bonding_start); The bondingStart() function start bonding with a remote device.
.bondingResponse	g_sf_ble0.p_api->bondingResponse (g_sf_ble_0.p_cfg, p_handle, p_bp_addr, p_bonding_resp); The bondingResponse () function responds to a bonding request.
.startEncryption	g_sf_ble0.p_api->startEncryption (g_sf_ble_0.p_cfg, p_enc_info); The startEncryption() function begins an encryption operation.
.connect	g_sf_ble0.p_api->connect (g_sf_ble_0.p_cfg, p_handle, p_conn); The connect function connects to a remote device.
.disconnect	g_sf_ble0.p_api->disconnect (g_sf_ble_0.p_cfg, p_handle); The disconnect() function disconnects from a remote device.
.listen	g_sf_ble0.p_api->listen (g_sf_ble_0.p_cfg); The listen function listens for an incoming connect request from a remote device.

Function Name	Example API Call and Description
authorization	g_sf_ble0.p_api->authorization (g_sf_ble_0.p_cfg, &conhandle); The authorization() function authorizes a remote device after connection.
setTxPower	g_sf_ble0.p_api->setTxPower(g_\sf_ble_0.p_cfg, &con_handle, &tx_power_info); The setTxPower() function sets the transmit power for the procedure specified by the connection handle.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables review the SSP Reference Manuals found as described in the Reference Section at the end of this document.

BLE Framework Module GATT API Summary

Function Name	Example API Call and Description
.gattAddCustomProfiles	g_sf_ble0.p_api->gattAddCustomProfiles (g_sf_ble_0.p_cfg, p_handle, p_sf_ble_svc_dscv_req, p_sf_ble_svc_dscv_rsp, p_rsp_cnt); This function adds custom profiles to the GATT database.
.gattServiceDiscovery	g_sf_ble0.p_api->gattServiceDiscovery (g_sf_ble_0.p_cfg, p_handle, p_sf_ble_svc_dscv_req, p_sf_ble_svc_dscv_rsp, p_rsp_cnt); The gattServiceDiscovery() function performs service discovery.
.gattCharDiscovery	g_sf_ble0.p_api->gattCharDiscovery (g_sf_ble_0.p_cfg, p_handle, p_sf_ble_svc_dscv_req, p_sf_ble_svc_dscv_rsp, p_rsp_cnt); The gattCharDiscovery() function performs the Char discovery operation.

Function Name	Example API Call and Description
.gattCharDescDiscovery	g_sf_ble0.p_api->gattCharDescDiscovery (g_sf_ble_0.p_cfg, p_handle, start_handle, end_handle, p_sf_ble_svc_dscv_rsp, p_rsp_cnt); Discovers GATT characteristics descriptor on a remote device.
.gattCharRead	g_sf_ble0.p_api->gattCharRead (g_sf_ble_0.p_cfg, p_handle, start_handle, p_char_read_req, p_char_read_rsp); Reads GATT characteristics on a remote device.
.gattCharWrite	g_sf_ble0.p_api->gattCharWrite (g_sf_ble_0.p_cfg, p_handle, p_char_read_req); Writes GATT characteristics on a remote device.
.gattCharExecuteWrite	g_sf_ble0.p_api->gattCharExecuteWrite (g_sf_ble_0.p_cfg, p_handle, execute_flag); Executes a write (commit) on GATT characteristics on a remote device.
.gattCharWriteLocal	g_sf_ble0.p_api->gattCharWriteLocal (g_sf_ble_0.p_cfg, char_handle, data_length); Update the local GATT database.
.gattSendNotify	g_sf_ble0.p_api->gattSendNotify (g_sf_ble_0.p_cfg, p_handle, char_handle); Sends notifications from local GATT server to remote GATT client.
.gattSendIndicate	g_sf_ble0.p_api->gattSendIndicate (g_sf_ble_0.p_cfg, p_handle, char_handle); Sends indications from local GATT server to remote GATT client.

Function Name	Example API Call and Description
.gattWriteResponse	g_sf_ble0.p_api->gattWriteResponse (g_sf_ble_0.p_cfg, p_handle, char_handle); Responds to the write characteristic value request from the remote GATT client.
.versionGet	g_sf_ble0.p_api->versionGet(&version); Retrieve the API version using the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables review the SSP Reference Manuals found as described in the Reference Section at the end of this document.

Error Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_ASSERTION	Parameter has invalid value.
SSP_ERR_INVALID_PTR	p_version is NULL.

NOTE: Lower level drivers may return Common Error Codes. Refer to the SSP Reference Manual available as described in the Reference Section at the end of this document for a definition of all relevant Error codes.

4.1.9.3 BLE Framework Module Operational Overview

This section provides the Synergy BLE Framework software architecture overview and highlights the major SSP modules used as part of BLE framework along with the operational flow sequence from the user's application level.

NOTE: A more comprehensive description of the operation of the BLE Framework module is available in the BLE Framework- Application Project. The complete project and associated application note can be found by doing a search for r30qan0309eu in the search bar at the top of the http://www.renesas.com home page.

BLE Framework Module Operation Description

The BLE framework provides a common interface for the application. The implementation of the interface is specific for each module. The Synergy BLE framework currently defines an interface implemented for RL78G1D BLE module. Each

implementation interacts with the corresponding BLE device driver. The BLE device driver uses the underlying SSP communication framework (g_sf_comms) which in turn interacts with the SSP HAL components such as Universal Asynchronous Receiver/Transmitter (UART), Data Transfer Controller (DTC), and General PWM Timer (GPT) drivers to communicate with the BLE module. The following figure shows a high-level architectural description of the BLE Framework module.

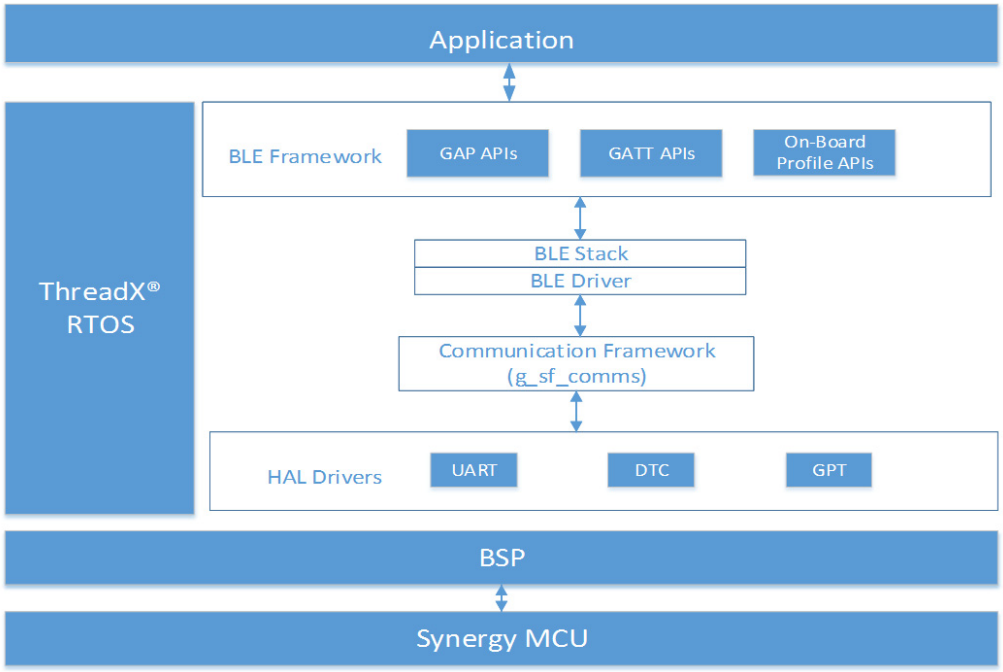


Figure 135: Typical BLE module architecture types

GAP and GATT APIs

The BLE framework provides a generic interface for the application to configure and provision the BLE module. The BLE module has various configuration parameters as specified by the family of Bluetooth Smart standards. It is possible that individual device drivers and/or BLE modules might not support all configuration parameters. At a bare minimum, the provisioning API provides a mechanism to set the operating mode, security mode, security keys, and bonding mode of the BLE interface. It also provides an API for the GAP/GATT layers.

On-board Profiles APIs

The on-board profiles APIs provide a uniform interface to the BLE profiles implemented by the BLE module firmware.

BLE Stack

The BLE module host stack is typically provided by the BLE module vendor. The BLE module typically comes in three different flavors depending on the HW/SW partitioning between the host MCU and BLE module. The RL78G1D BLE module is part of the Network Controller Implementation architecture where the BLE chipset includes all the implementation for the BLE link layer, GAP, GATT, and on-board profiles. The module interfaces with the MCU over sf_comms framework provided by SSP.

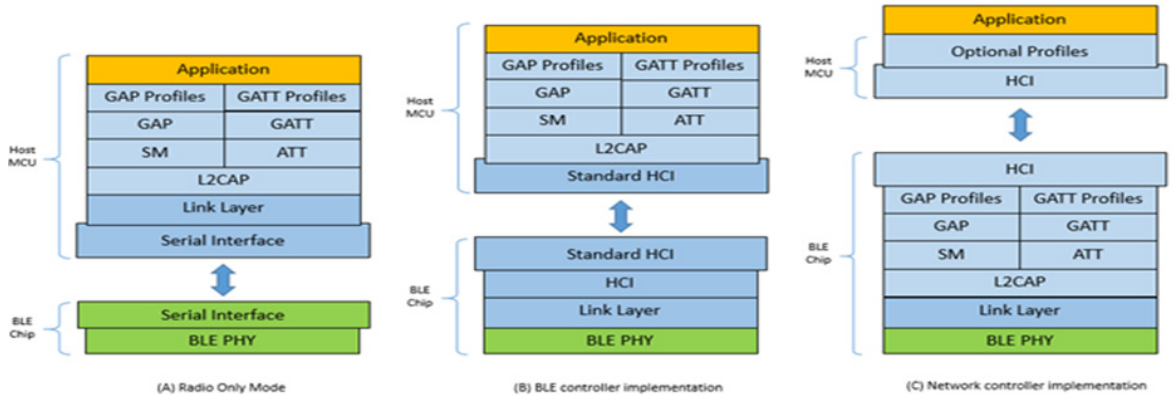


Figure 136: BLE module architecture types

A: BLE radio-only mode

Link layer, L2CAP, GATT, GAP layers, profiles, and application run on the host MCU. Physical layer runs on BLE chipset.

B: BLE controller implementation

Link layer runs on BLE chipset, L2CAP, and higher BLE protocol (GATT, GAP) layers. Profiles and application run on the host MCU.

C: Network controller implementation

Link layer, L2CAP, GATT, GAP layers, and generic profiles run on the BLE chipset. Optional profiles and application run on the host processor.

4.1.9.4 BLE Framework Instances

Application must define the BLE framework instance before using it. The instance is a structure that includes pointers to any of the following:

BLE Framework Control Structure

This structure is used in all BLE framework APIs. This structure includes pointer to driver handle, which is used by framework for storing the required information by the BLE device driver.

BLE Framework Configuration Structure

This structure is passed to open() API and you can use this structure to configure the BLE module. This configuration is applied either during initialization, such as open or provisioning such as provisioningSet. Configuration parameters that are not supported by the BLE module are ignored by the framework.

BLE Framework APIs structure

This structure contains pointers to the BLE Framework APIs that are specific to a given module. See the BLE Framework Module API Overview for more details on these APIs.

BLE Framework Module Operational Flow

The steps for using the BLE Framework module in an application are:

1) Initialize the BLE hardware module.
2) Select the GATT layer role such as GATT client or GATT server. It is most common for the slave (peripheral) device to be the GATT server and the master (central) device to be the GATT client.
3) The application controls operations using generic (on-board) profile APIs or GAP/GATT APIs.

NOTE: The GAP provisioning structure has a BLE user callback that runs in the driver thread context. An application should make sure that callback logic is as minimal as possible without any blocking calls. Print statements or blocking call may introduce delays in BLE driver execution. Make sure that no BLE APIs are called in user callbacks as it may also lead to code failure.

The following BLE module initialization sequence is part of the Synergy auto-generated code:

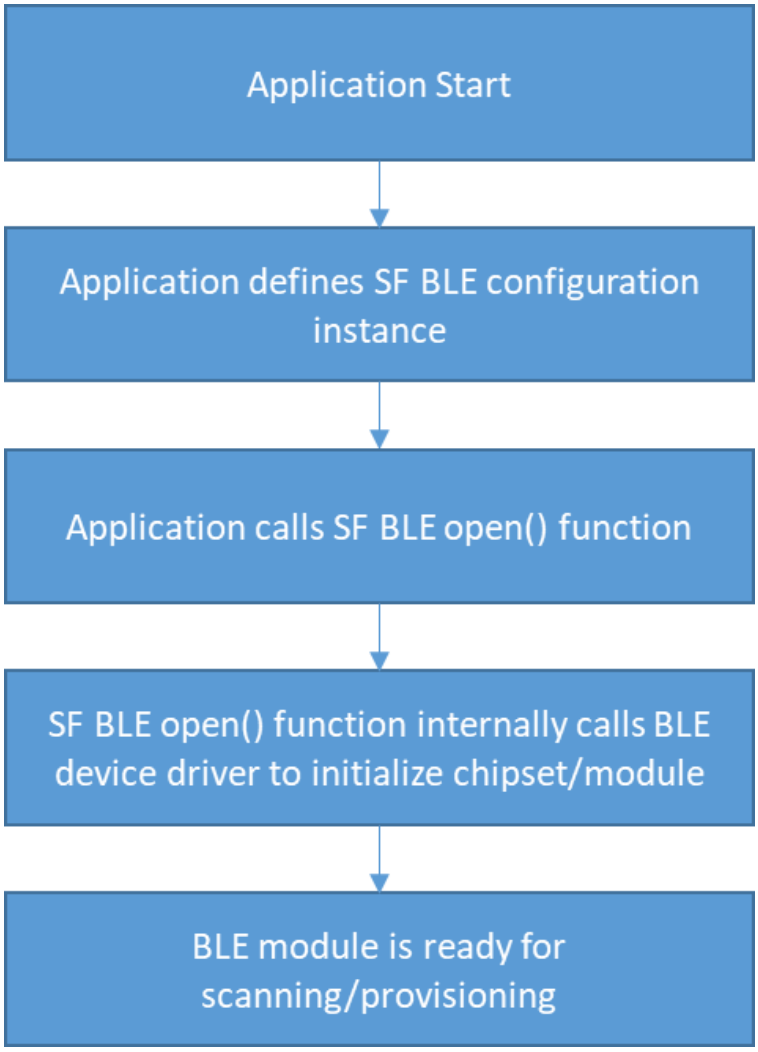


Figure 137: Autogenerated Initialization Sequence Flow Chart

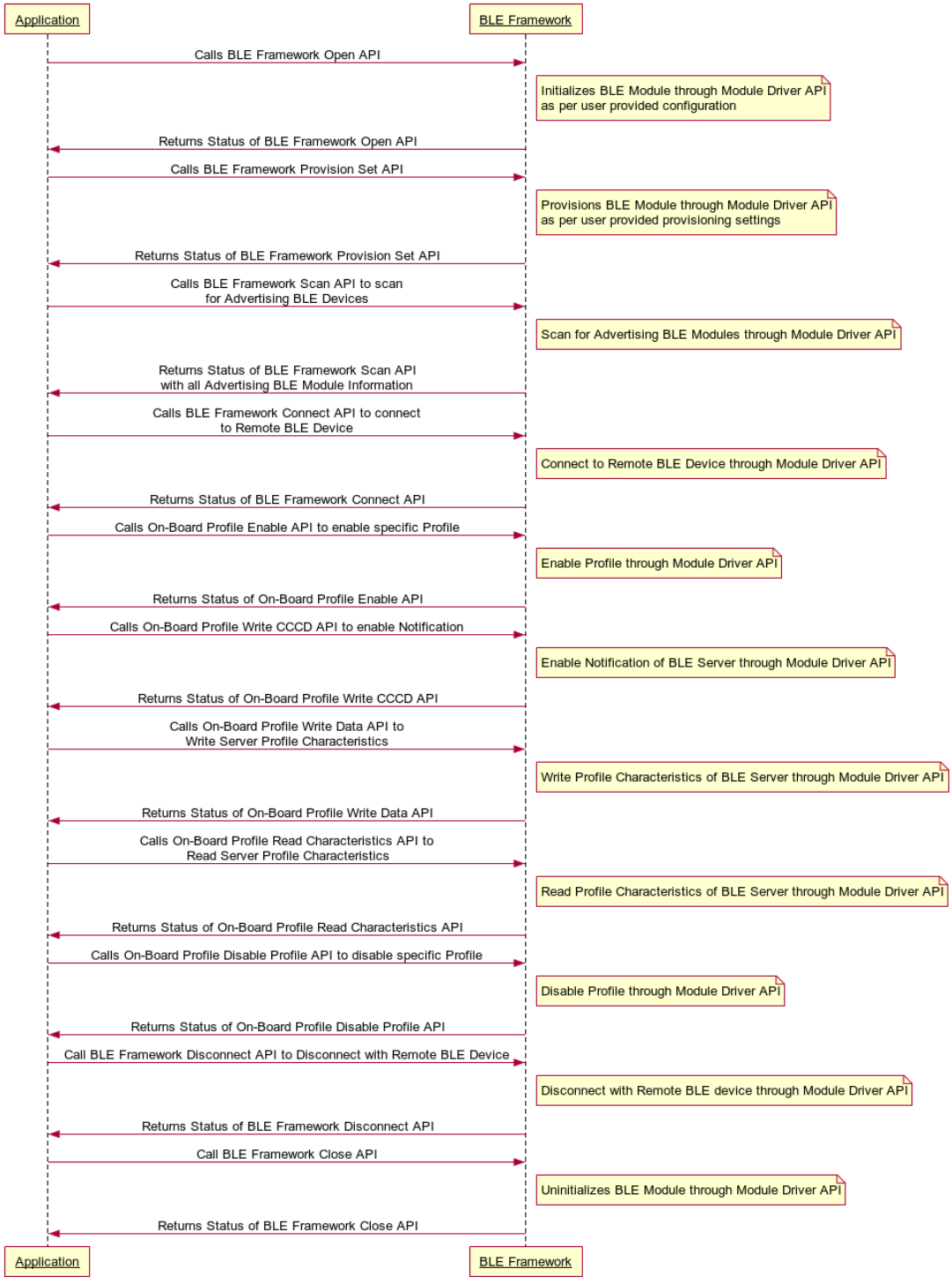


Figure 138: On-Board Profile Client Application Flow

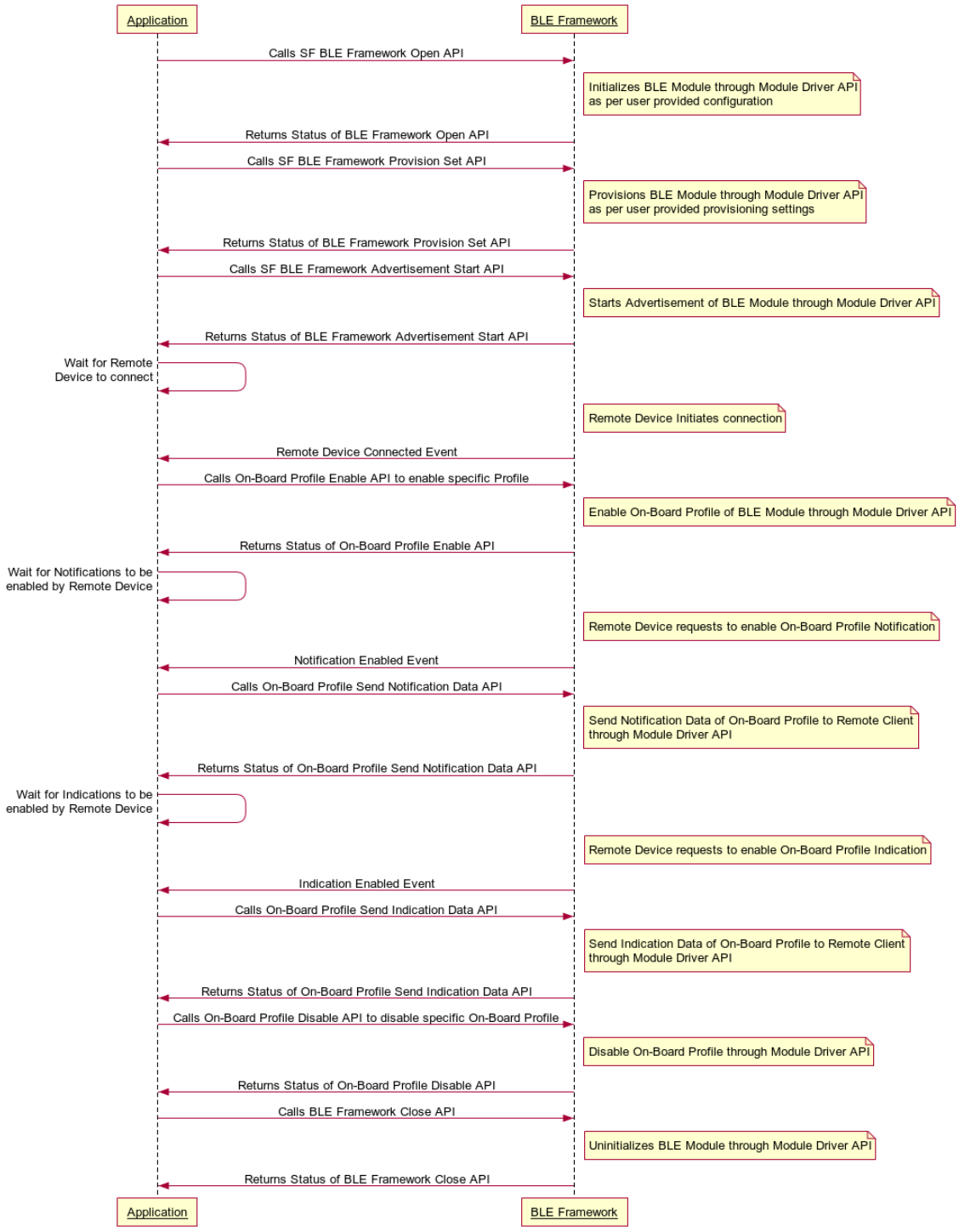


Figure 139: On-Board Profile Server Application Flow

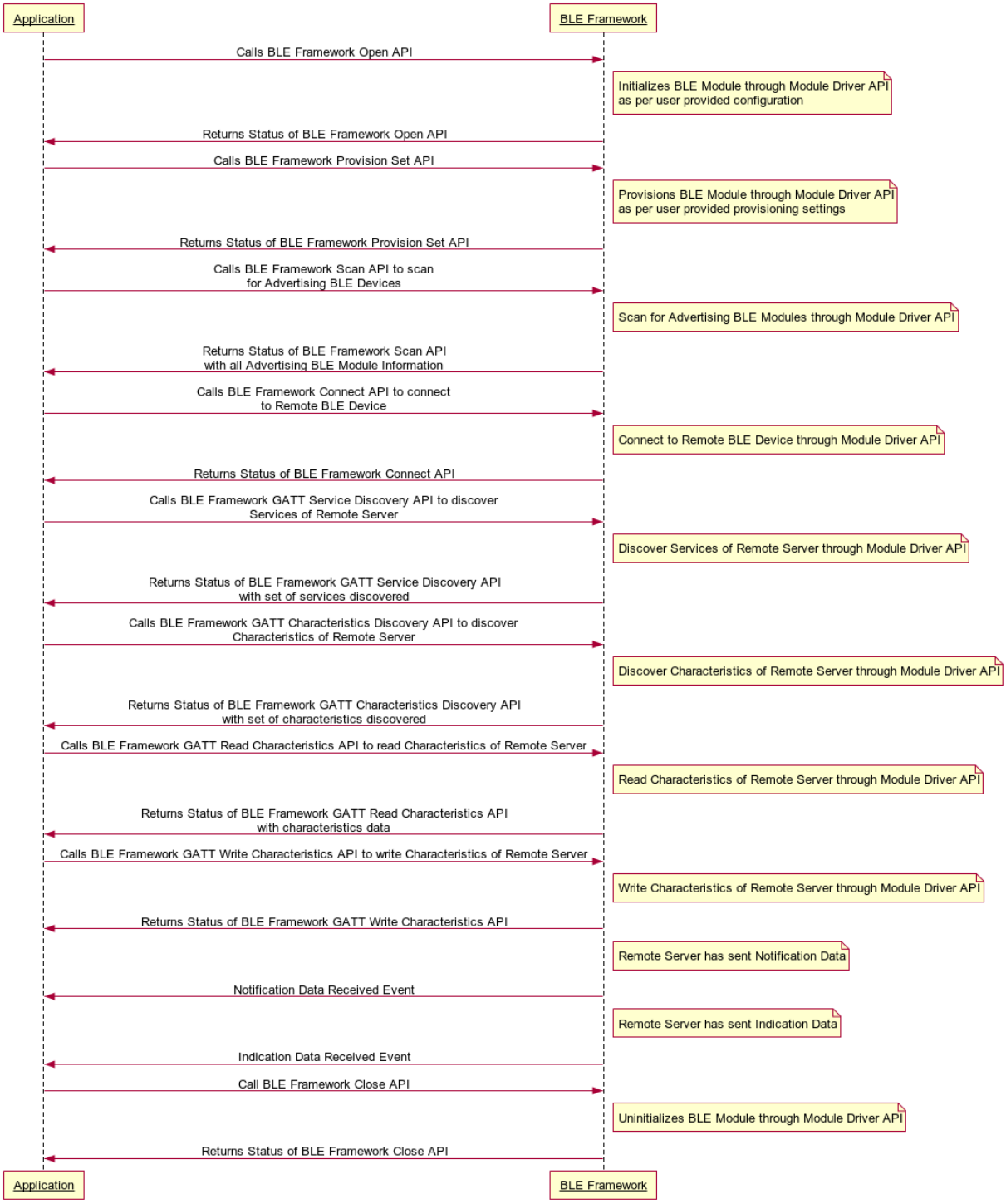


Figure 140: GAP/GATT Client Application Flow

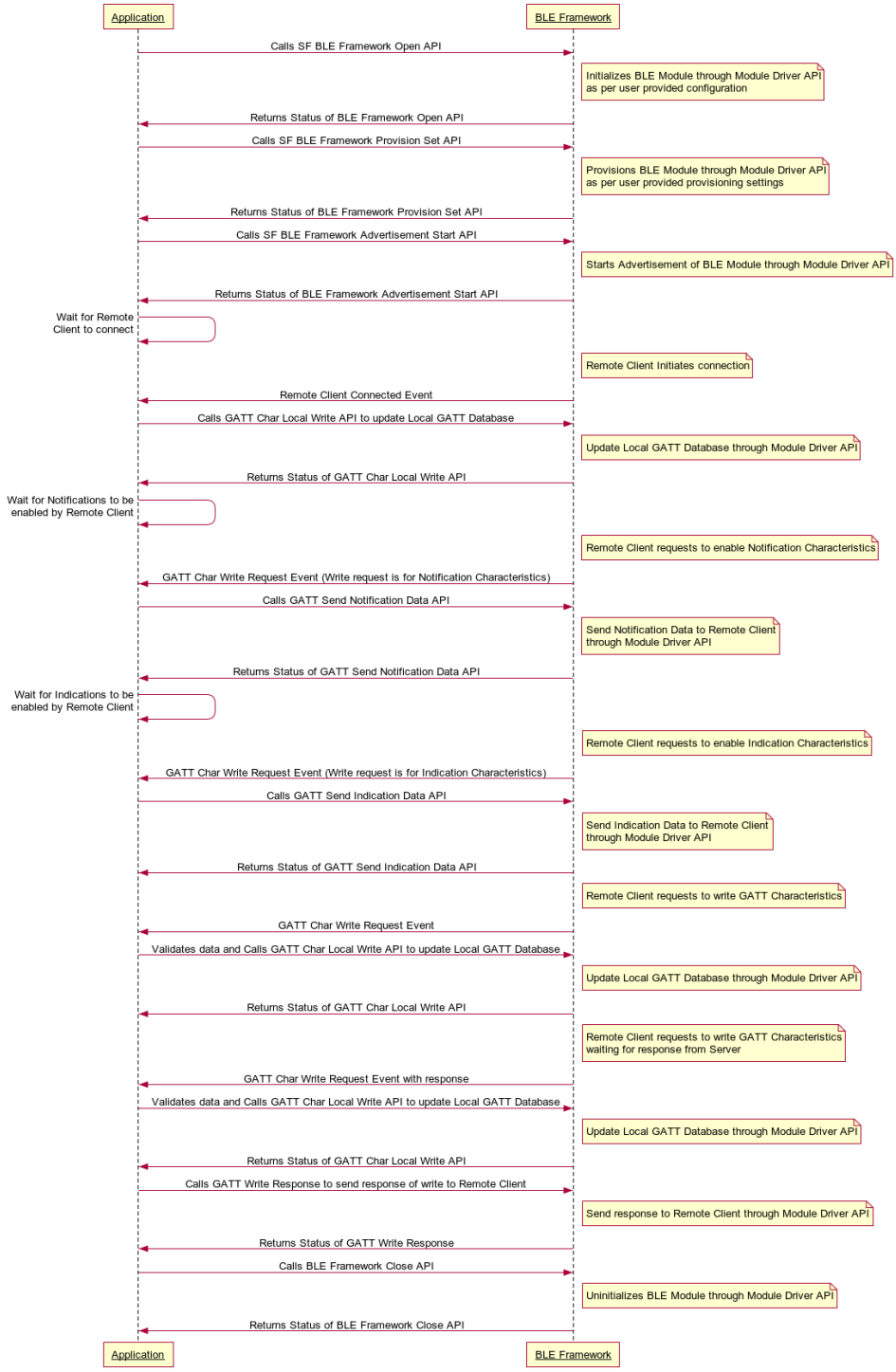


Figure 141: GAP/GATT Server Application Flow

BLE Framework Security

Security Manager provides BLE protocol stack the ability to generate and exchange security keys which is used to encrypt communication link. The Security Manager has two functions:

- **Initiator**
  - This is the GAP Master/Central device
- **Responder**
  - This is the GAP Slave/Peripheral device.

The initiator is the master device that initiate the security procedure, however the slave device can asynchronously request the initiator to begin the security procedure.

BLE Security provides modes with levels associated with each mode. Security mode and level is a combination of support for authenticated or unauthenticated pairing, encryption or data signing. Pairing is required to satisfy various security requirements. Two types of pairing are available:

Authenticated pairing where devices are protected from MITM (Man In The Middle) attacks

Unauthenticated pairing where they are not protected from MITM.

- **Security Mode 1**
  - Security Level 1: No Security
  - Security Level 2: Unauthenticated pairing with encryption
  - Security Level 3: Authenticated pairing with encryption
  - Security Level 4: Authenticated LE secure connections pairing with encryption
- **Security Mode 2**
  - Security Level 1: Unauthenticated pairing with data signing
  - Security Level 2: Authenticated pairing with data signing

NOTE: RL78G1D BLE module does not support Security Mode 1 with Security Level 4

- **On-Board Profile Security Modes**
  - **No Security**: If On-Board Profile security is set to No Security then Profile communication works in unsecured mode regardless of BLE GAP Security Mode and Level.
  - **Unauthenticated**: For this Profile Security method to work any of the BLE GAP Security Method other than Security Mode 1 Level 1 should be used. Module should have completed Pairing with the remote device.
  - **Authenticated**: For this Profile Security method to work any of the following BLE GAP Security Method should be used. Module should have completed Pairing with the remote device.Security Mode 1 Level 3Security Mode 2 Level 2
  - **Authorization**: For this Profile Security method to work the remote device should have been Authorized during GAP connection using BLE Framework authorization API This On-Board profile security parameter is specific to server API only. Refer sample code snippets in section 0 for more details.

– **Encryption**: In this Security procedure Profile will use Encrypted communication. For this Profile Security method to work any of the BLE GAP Security Method other than Security Mode 1 Level 1 should be used. If any security method is used which does not use encryption then Profile works in unencrypted mode.

NOTE: 1. On-Board Profile security can be set in onbpEnable() function. onbpEnable() API enables the profile in server mode or client mode. It is a generic API to enable client and server mode. But, profile security parameter is specific to server API only. Security parameter can be set for server only in driver APIs. So, user needs to set profile security for server enable API.

NOTE: 2. Profile security enums are in bit pattern, so more than one profile security can be set using bitwise operators. There are few points to be kept in mind while setting the profile security. They are listed below:

- If profile security is none, then other security bits should not be set.
- Security bits should not be set to both authenticated and unauthenticated.
-

BLE Security has the following procedures:

Pairing

This procedure is used to generate temporary encryption key to encrypt communication link.

Permanent encryption keys can be shared over this encrypted communication link for additional communication.

Bonding

This is a combination of pairing and storing of permanent keys. After pairing, the permanent keys are stored in a non-volatile memory, which creates a permanent bond between two devices. For subsequent communication, it is not necessary for devices to perform the bonding procedure.

Encryption Establishment

Communication is encrypted using permanent keys.

Pairing creates a secure link that lasts for the lifetime of the connection, whereas bonding creates a permanent association called bond.

BLE Security goes through three phases. Two devices establish connection using the GAP connection procedure, followed by the three phases to establish a secure communication link:

Phase 1 (Pairing Phase, Information Sharing)

Initially in phase 1, all information required to generate the temporary keys are shared between two devices.

Phase 2 (Pairing Phase, Temporary Key Sharing)

In this phase, temporary encryption key (Short Term Key or STK) is generated on both devices. This is used to encrypt the connection. This encrypted link can be used for additional communication. This communication link remains encrypted until the peer devices stay connected.

Phase 3 (Bonding, Sharing and Storage of Permanent keys)

Devices enter this phase if bonding is required. In this phase, permanent keys (Long Term Key or LTK) is exchanged between two devices using the encrypted link which was established in phase 2 using temporary keys. These permanent keys are then stored in non-volatile memory to be made available for the devices over each connection.

4.1.9.5 BLE Framework Limitations

1) The BLE framework is tested only on RL78G1D BLE hardware module. Supported for different BLE modules will be added in later versions.
2) BLE Framework using RL78G1D will see compilation warnings. All the warnings are in the 3rd party RL78G1D driver code. The BLE framework files do not have any warning. These warnings should not impact the user applications.
3) The custom profile support in the BLE framework is limited to RL78G1D type BLE hardware module only.
4) HID profile client mode not supported by RL78G1D BLE hardware module. As a result, the BLE framework implementation of HID profile will also not support HID profile client mode. Applications using BLE framework for RL78G1D will not be able to use the HID profile in client mode.
5) Multiple slave BLE devices cannot be connected to RL78G1D BLE module.
6) The BLE framework is only tested on the following boards:
   - DK-S7G2 Version 3.1
   - DK-S3A7 Version 2.0
   - PK-S5D9 Version 1.0
   - ADK-S3A3
   - TB-S5D5 Version 0.5D
   - TB-S3A6 Version 0.5D
   - DK-S128 Version 0.5b
   - DK-S124 Version 3.1
7) Refer to the most recent SSP release notes for module limitations.

4.1.9.6 Including the BLE Framework Module in an Application

This section describes how to include the BLE Framework Module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the Getting Started Guide for SSP given in the References section at the end of this document to learn how to manage each of these important steps in creating SSP-based applications.

To add the BLE Framework module to an application, simply add it to a project thread using the stacks selection sequence given in the following table. (The default name for the BLE Framework is g_sf_ble0. This name can be changed in the associated Properties window.)

BLE Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_ble0 RL78G1D BLE GAP and GATT on sf_ble_rl78g1d	Threads	New Stack> Framework> Networking> BLE > RL78G1D BLE GAP and GATT on sf_ble_rl78g1d
g_sf_ble_onboard_profile0 On-Board Profile on RL78G1D BLE Framework	Threads	New Stack> Framework> Networking> BLE > On-Board Profile on RL78G1D BLE Framework

When the BLE Framework Module is added to the thread stack as shown in the following figure, the configurator automatically adds the needed lower-level drivers. Any drivers that need additional configuration information will be highlighted in red. Modules with a gray band are individual modules that stand alone. Modules with a blue band are shared or common and need only be added once and can be used by multiple stacks. Modules with a pink band can require the selection of lower-level drivers; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level drivers is required, the module description will include "Add" in the text. Clicking on any pink banded modules will bring up the "New" icon and then display the possible choices.

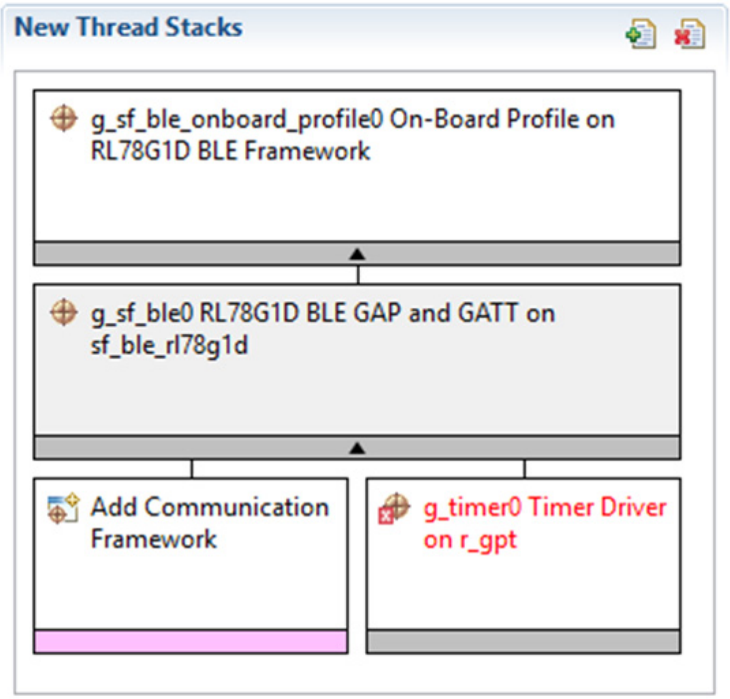


Figure 142: BLE Framework Module Stack

4.1.9.7 Configuring the BLE Framework Module

The BLE Framework Module must be configured by you for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or

operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the Interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the On-Board Profile on RL78G1D BLE Framework

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Heart Rate Profile	Enabled, Disabled Default: Enabled	Heart rate profile selection
Alert Notification Profile	Enabled, Disabled Default: Disabled	Alert notification profile selection
Blood Pressure Profile	Enabled, Disabled Default: Disabled	Blood pressure profile selection
Find Me Profile	Enabled, Disabled Default: Disabled	Find me profile selection
HID Over GATT Profile	Enabled, Disabled Default: Disabled	HID gatt profile selection

ISDE Property	Value	Description
Health Thermometer Profile	Enabled, Disabled Default: Disabled	Health thermometer profile selection
Phone Status Alert Profile	Enabled, Disabled Default: Disabled	Phone alert profile selection
Proximity Profile	Enabled, Disabled Default: Disabled	Proximity proximity profile selection
Scan Parameter Profile	Enabled, Disabled Default: Disabled	Scan parameter profile selection
Time Profile	Enabled, Disabled Default: Disabled	Time time profile selection
Name	g_sf_ble_onboard_profile0	Module name

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the BLE GAP and GATT

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_sf_ble0	Module name
Bluetooth Device Address (Restart Board after first run to see changed Address)	{0x0, 0x0, 0x0, 0x0, 0x0}	Bluetooth device address selection

ISDE Property	Value	Description
Address Type	Public Address, Random Address Default: Public Address	Address type selection
Scan Interval	48	Scan interval selection
Scan Window	48	Scan window selection
Maximum Connection Interval	40	Maximum connection interval selection
Connection Slave Latency	0	Connection slave latency selection
Supervision Timeout	80	Supervision timeout selection
BLE Driver Thread Priority	1	BLE Driver thread priority selection
BLE Serial Thread Priority	1	BLE Serial thread priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the GPT HAL Module on r_gpt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_timer0	Module name.
Channel	0	Channel selection.
Mode	Periodic	Warning: One-shot functionality is not available in the GPT hardware, so it is implemented in software by stopping the timer in the ISR called when the period expires. For this reason, ISR's must be enabled for one-shot mode even if the callback is unused.
Period Value	10	See Timer Period Calculation

ISDE Property	Value	Description
Period Unit	Milliseconds	See Timer Period Calculation
Duty Cycle Value	50	Duty cycle value selection
Duty Cycle Unit	Unit Raw Counts	Duty cycle unit selection
Auto Start	True	Set to true to start the timer after configuring or false to leave the timer stopped until start is called.
GTIOCA Output Enabled	False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCA Stop Level	Pin Level Low	Controls output pin level when the timer is stopped.
GTIOCB Output Enabled	False	Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCB Stop Level	Pin Level Low	Controls output pin level when the timer is stopped.
Callback	RBLE_Timer_cb	A user callback function can be registered in open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

ISDE Property	Value	Description
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

BLE Framework Module Clock Configuration

The BLE Framework module uses the clocks specified by the lower-level modules.

BLE Framework Module Pin Configuration

The BLE Framework module uses the pins specified by the lower-level modules.

4.1.9.8 Using the BLE Framework Module in an Application

NOTE: A more comprehensive example of using the BLE Framework module is available in the BLE Framework-Application Project. This complete project and associated application note can be found by doing a search for r30an0309eu in the search bar at the top of the http://www.renesas.com home page.

The typical steps in using the BLE Framework module in a simple application, using the on-board profile server flow, are:

1) Initialization- registration of callback functions, provisioning, and advertisement using the open API
2) Provision the BLE module using the provisionSet API
3) Advertise the module using the advertisementStart API- wait for remote device to connect
4) Enable the on-board profile using the onbpEnable API- wait for notification to be enabled by Remote device
5) Send notification using the onbpServerSendNotification API- wait for Indication to be enabled by Remote device
6) Send indication using the onbpServerSendIndication API
7) Disable the profile using the onbpDisable API
8) Close the BLE module using the close API

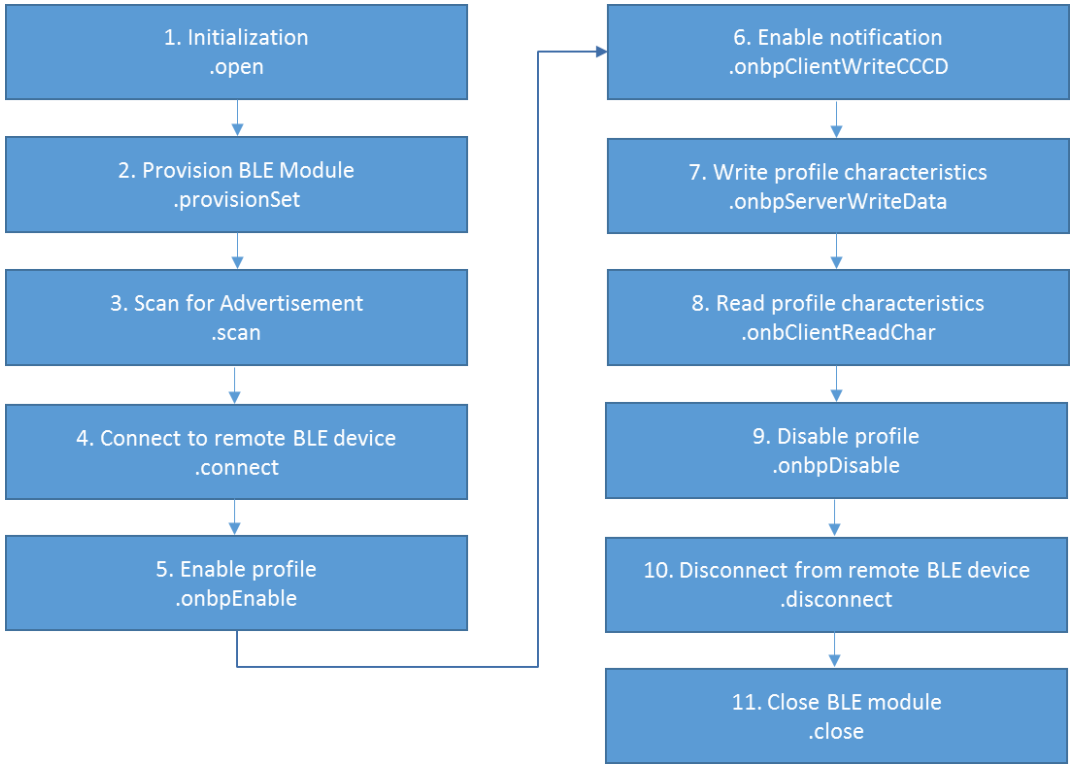


Figure 143: Typical BLE Framework Application Flow for On-Board Profile Client

4.1.10 Cellular Framework

The Cellular Framework module provides a high-level application layer interface for the cellular modem integration in SSP. The Cellular framework provides a common interface for the applications to interface with the Cellular modems from various vendors.

The Cellular Framework provides a set of APIs to provision, configure and communicate with the cellular network for data communication. The Cellular framework uses the Console framework to communicate with Cellular modems over a serial interface by using AT commands. The Cellular framework creates the serial data pipe over serial interface for the data communication, leveraging the PPP WAN protocol provided by NetX™. Data communication using TCP/IP can be established over this Wide Area Network (WAN) link using NetX Application protocols, sockets or IoT protocols such as MQTT.

The Cellular Framework also provides the framework level Socket APIs to communicate with the TCP/IP stack present on-chip (inside cellular hardware module) in certain cellular hardware modules and with the TCP/IP link for the network using socket APIs.

4.1.10.1 Cellular Framework Module Features

- Supports connectivity using:
  - CAT1, CAT3 and CAT M1 Cellular Modems
  - BSD Socket interface for On-Chip stack present on the Cellular Module

  - NetX Stack on Synergy MCU (Host) using NSAL interface
- Supports a common a set of APIs, to interface to the networking stack and generic interface for the different Cellular hardware modules.
- Using generic APIs and abstraction, applications developed for the cellular hardware module can be easily migrated to work with another cellular hardware module.
- Supported Cellular modems:
  - NimbeLink CAT3 (NL-SW-LTE-TSVG) Verizon-US
  - NimbeLink CAT3 (NL-SW-LTE-TEUG) India and Europe
  - NimbeLink CAT1 (NL-SW-LTE-GELS3-B and NL-SW-LTE-GELS3-C) Verizon-US
  - Quectel CAT M1-BG96

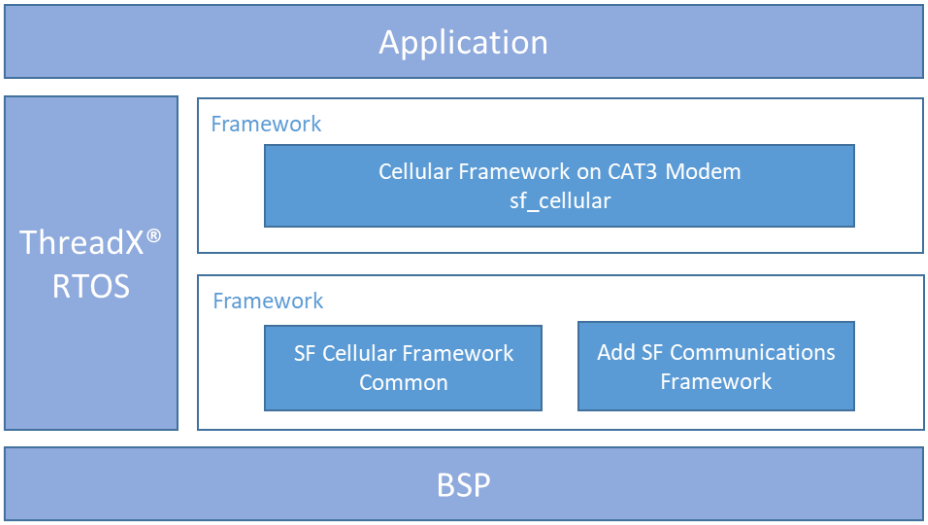


Figure 144: Cellular Framework on CAT3 Modem Module Organization, Options and Stack Implementations

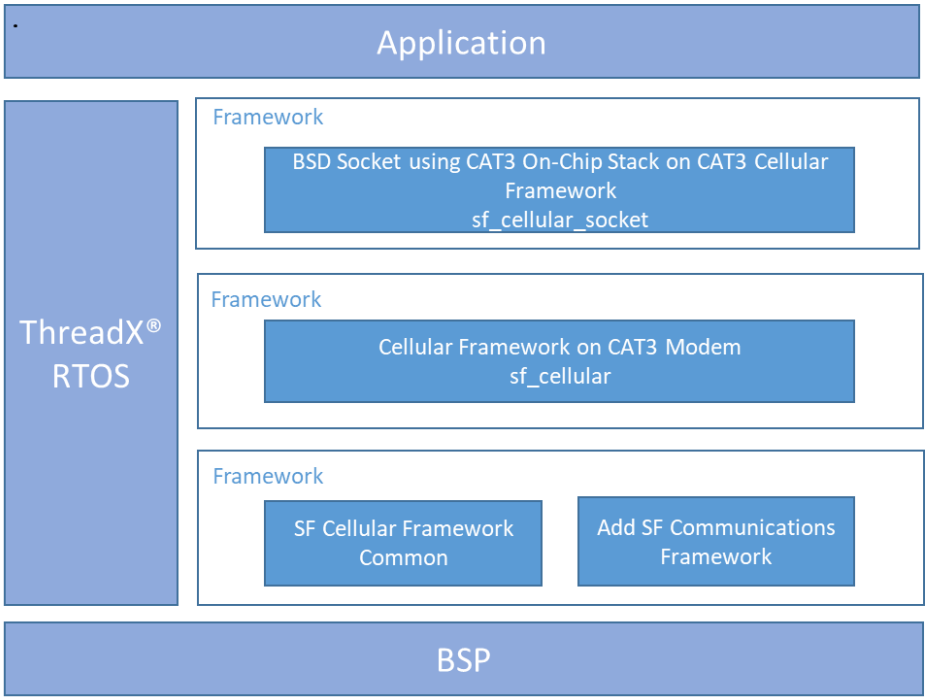


Figure 145: BSD Socket using CAT3 On-Chip Stack on CAT3 Cellular Framework Module Organization, Options and Stack Implementations

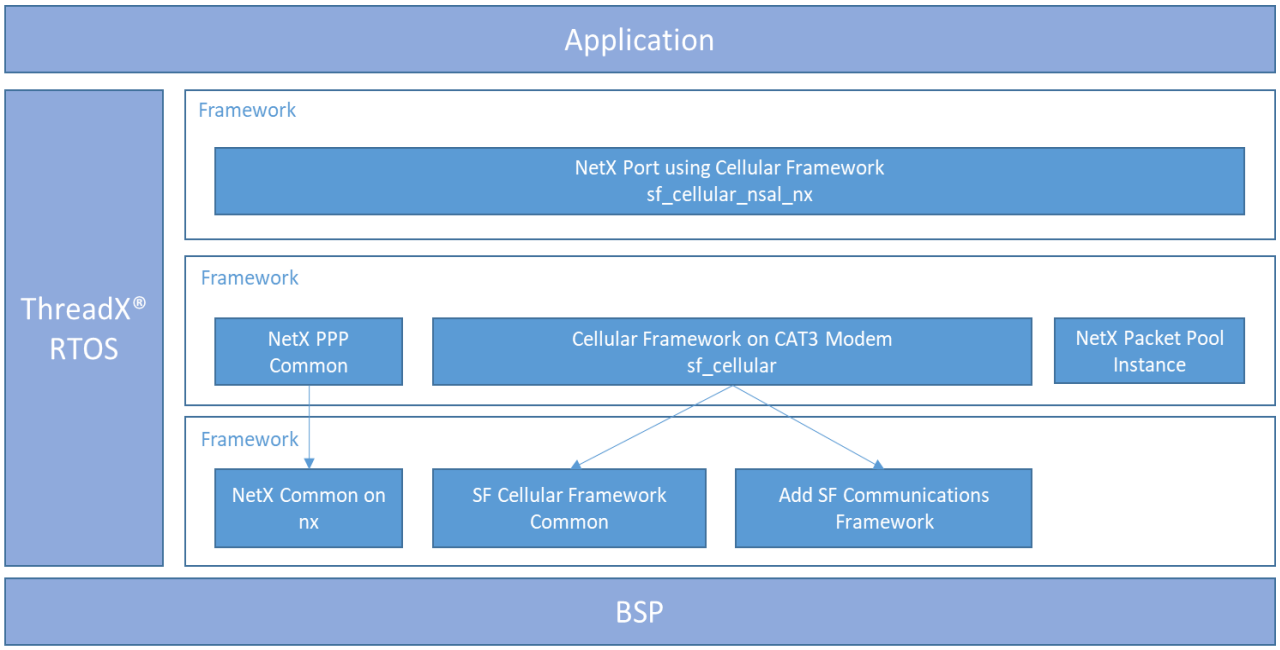


Figure 146: NetX Port using Cellular Framework Module Organization, Options and Stack Implementations

4.1.10.2 Cellular Framework Module APIs Overview

The Cellular Framework defines API's for each of the related modules. The following descriptions explain the operation of each API.

NOTE: A more detailed description of the Cellular framework module APIs are available in the Cellular Application Note downloadable from the Renesas web site. Just search, in the top search bar, for R30AN0311 and the application note and application project will be listed in the search results.

Cellular Framework Module APIs Summary

The Cellular framework provides a generic interface for the network stack and applications. Below are the APIs provided by the framework.

Cellular Framework Module API Summary

Function Name	Example API Call and Description
.open	g_sf_cellular0.p_api->open (g_sf_cellular0.p_ctrl, g_sf_cellular0.p_cfg); This API function initializes and enables the Cellular module. The open function returns the Cellular control structure, uniquely identifying the instance of the Cellular framework. The Cellular framework open function accepts the Cellular module configuration as an argument, with the following parameters: Operator Selection Mode (enumeration) Operator Name Format (enumeration) Operator Name (string) Preferred Operator List (array of structures) Time zone update policy (enumeration)
.close	g_sf_cellular0.p_api->close (g_sf_cellular0.p_ctrl); This API un-initializes the Cellular module and disables it. It takes the Cellular control structure as an argument.

Function Name	Example API Call and Description
.infoGet	g_sf_cellular0.p_api->infoGet (g_sf_cellular0.p_ctrl, p_cellular_info); This API takes the Cellular control structure as an argument and returns the following information obtained from the Cellular module: Chipset/driver information (string) Manufacturer Name (string) Firmware version (string) IMEI Number (string) RSSI value (unsigned integer 16 bits) Bit Error Rate (unsigned integer 16 bits)
.statisticsGet	g_sf_cellular0.p_api->statisticsGet (g_sf_cellular0.p_ctrl, p_stats); This API gets the data statistics from the Cellular module. It takes the Cellular control structure as an argument and returns the following statistics: Received packets (unsigned integer 32 bits) Transmitted packets (unsigned integer 32 bits) Transmit packet errors (unsigned integer 32 bits)

Function Name	Example API Call and Description
.transmit	g_sf_cellular0.p_api->transmit (g_sf_cellular0.p_ctrl, p_buffer, length); This API sends the data/packet out. It takes the Cellular control structure, the data buffer and the data buffer length as an argument. The Cellular framework transmit function passes the data buffer to the PPP driver for transmission.
.provisioningGet	g_sf_cellular0.p_api->provisioningGet (g_sf_cellular0.p_ctrl, p_cellular_provisioninfo); This API takes the Cellular control structure as an argument and returns the following parameters: Authentication type(enumeration) Username (string) Password (string) APN Name(string) PDP Context ID (integer) PDP Context Type (enumeration) Airplane mode (enumeration)

Function Name	Example API Call and Description
.provisioningSet	g_sf_cellular0.p_api->provisioningSet (g_sf_cellular0.p_ctrl, p_cellular_provisioninfo); This API sets the authentication credential information. It takes the Cellular control structure and the following parameters as argument to provision the Cellular module: Authentication type(enumeration) Username (string) Password (string) APN Name(string) PDP Context ID (integer) PDP Context Type (enumeration) Airplane mode (enumeration)
.networkConnect	g_sf_cellular0.p_api-> networkConnect (g_sf_cellular0.p_ctrl); This API establishes the Network connection over Cellular Network using which application can communicate to remote host with the help of Network stack. It takes the Cellular control structure as an argument.
.networkDisconnect	g_sf_cellular0.p_api->networkDisconnect (g_sf_cellular0.p_ctrl); This API terminates the Network connection established using networkConnect API. It takes the Cellular control structure as an argument.

Function Name	Example API Call and Description
.simPinSet	g_sf_cellular0.p_api->simPinSet (g_sf_cellular0.p_ctrl, p_pin); This API allows the application/user to change the PIN required to register on Cellular Network. It takes the Cellular control structure, old PIN and New PIN as arguments.
.simLock	g_sf_cellular0.p_api->simLock (g_sf_cellular0.p_ctrl, p_pin); This API locks the SIM. It takes the Cellular control structure and Sim PIN as arguments.
.simUnlock	g_sf_cellular0.p_api->simUnlock (g_sf_cellular0.p_ctrl, p_pin); This API unlocks the SIM. It takes the Cellular control structure and Sim PIN as arguments.
.simIDGet	g_sf_cellular0.p_api->simIDGet (g_sf_cellular0.p_ctrl, p_sim_id); This API reads the Sim ID from the Cellular module. It takes the Cellular control structure as argument and returns the SIM ID read from the Cellular module.

Function Name	Example API Call and Description
.networkStatusGet	g_sf_cellular0.p_api->networkStatusGet (g_sf_cellular0.p_ctrl, p_status); This API gets Cellular Module Network Status information. It takes the Cellular control structure as argument and returns following parameters: Country code (integer) Operator code (integer) RSSI (integer) Cell ID (string) IMSI (string) Operator name (string) Service Domain (integer) Active Band (integer)
.fotaCheck	g_sf_cellular0.p_api->fotaCheck (g_sf_cellular0.p_ctrl, p_fota_check); This API checks for available firmware Upgrade. It takes the Cellular control structure and fota check specific structure as argument.
.fotaStart	g_sf_cellular0.p_api->fotaStart (g_sf_cellular0.p_ctrl, p_fota_start); This API starts the firmware upgrade. It takes the Cellular control structure and fota start specific structure as argument.

Function Name	Example API Call and Description
.fotaStop	g_sf_cellular0.p_api->fotaStop (g_sf_cellular0.p_ctrl, p_fota_stop); This API stops the firmware upgrade. It takes the Cellular control structure and fota stop specific structure as argument.
.versionGet	g_sf_cellular0.p_api->versionGet (p_version); This API retrieves the version for the API using the version pointer.
.reset	g_sf_cellular0.p_api->reset (g_sf_cellular0.p_ctrl, reset_type); Reset the cellular hardware module.

Cellular Framework Module Socket APIs Summary

The Cellular socket framework provides additional API's as shown in the following table.

Cellular Socket Framework Module API Summary

Function Name	Example API Call and Description
.open	g_sf_cellular_socket0.p_api->open (g_sf_cellular_socket0.p_ctrl, g_sf_cellular_socket0.p_cfg); This API calls the Cellular Framework's lower level Open () API to Initialize the Cellular Device Driver.
.close	g_sf_cellular_socket0.p_api->close (g_sf_cellular_socket0.p_ctrl); This API calls the Cellular Framework's lower level Close () API to close the Cellular Device Driver
.versionGet	g_sf_cellular_socket0.p_api->versionGet (p_version); This API retrieves the version for the API using the version pointer.

Cellular Framework Module Socket APIs Summary

These APIs can be used by application to perform data transfer using sockets. It includes socket APIs which are compliant with BSD APIs:

- socket
- close
- connect
- send
- recv
- sendto
- recvfrom
- select

Cellular Framework Module Error Codes

The table below lists the Cellular Framework specific error codes. These error codes are part of ssp_err_t.

Cellular Framework Error Codes

Error Codes	Description
SSP_SUCCESS	Call successful
SSP_ERR_CELLULAR_CONFIG_FAILED	Configuration failed
SSP_ERR_CELLULAR_INIT_FAILED	Initialization failed
SSP_ERR_CELLULAR_TRANSMIT_FAILED	Transmit failed
SSP_ERR_CELLULAR_FW_UPTODATE	Up to date
SSP_ERR_CELLULAR_FW_UPGRADE_FAILED	Upgrade failed
SSP_ERR_CELLULAR_FAILED	General failure
SSP_ERR_CELLULAR_INVALID_STATE	Invalid state

4.1.10.3 Cellular Framework Module Operational Overview

Cellular Framework Module Operational Introduction

The Cellular framework provides a generic interface for applications to seamlessly communicate with the Cellular hardware module from various vendors without the necessity of changing the applications. The framework mainly consists of a common set of APIs, to interface to the networking stack and different Cellular hardware modules without having to change the applications. This section introduces the Cellular framework's basic blocks and key features that enables you to determine whether the intended Cellular application can be developed using the Cellular framework.

NOTE: Additional operational descriptions of the Cellular framework module are available in the Cellular Application

Note downloadable from the Renesas web site. Just search, in the top search bar, for R30AN0311 and the application note and application project will be listed in the search results.

With the Generic API and abstraction, the applications developed for the cellular hardware module can be easily ported to use another cellular hardware module. The networking stack NetX is also integrated with the framework using the Network Software Abstraction Layer (NSAL).

The Synergy Cellular Framework consists of the following logical blocks:

- Synergy Cellular Framework Application Interface.
- Network Stack Abstraction Layer (NSAL) for NetX TCP/IP stack
- Cellular Device Driver (AT command interface for interacting with the cellular chipset)
- BSD Socket compatible APIs for interfacing with Cellular hardware module that supports on-chip networking stack
- Synergy Software Package(SSP) HAL Interface

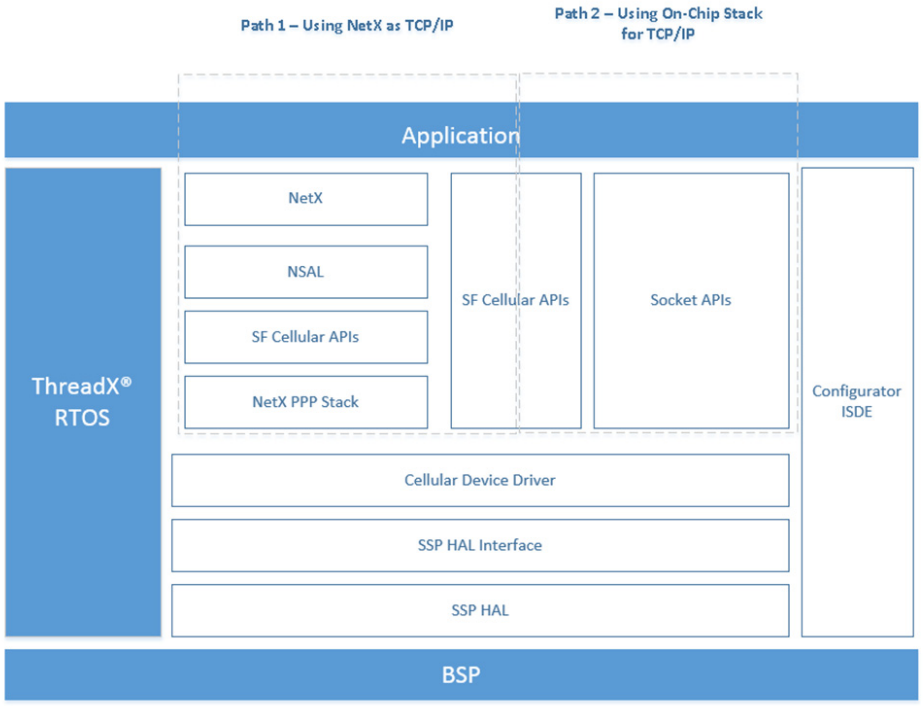


Figure 147: Cellular Framework Module Application Perspective

The Cellular Framework provides a common set of interfaces for the application to configure, provision and to communicate with the Cellular hardware module. By using these Generic interfaces, the user can develop the Cellular based application using Synergy MCUs. The Cellular hardware module has various configuration parameters as specified by the family of 3GPP standards. It is possible that individual device drivers and/or Cellular chipsets/modules will not support configuration of all parameters. At a bare minimum, the network operator, Access Point Name (APN) and security credentials are required to make the module functional.

The Cellular Framework provides a network stack abstraction layer (NSAL). NSAL is layer which connects the NetX and the Cellular driver by using (PPP) stack that is used for the data communication over WAN link.

The Cellular Framework provides a Socket level API for the application to interact with the on-chip networking stack present on the Cellular hardware module. This requires the Cellular hardware module/driver to support an on-chip networking stack and socket interface. When the application uses these APIs, it uses the on-chip networking stack present on the Cellular hardware module and does not use the NSAL or the NetX and its Socket APIs and does not use the Networking stack running on the Synergy MCU Group.

Point to point protocol (PPP) is widely used WAN protocol in the Data communication. PPP Stack is part of NetX available in the SSP. NSAL leverages the PPP stack to communicate over the serial interface to the cellular service provider's network. PPP configuration provides options, for authentication methods like PAP/CHAP for the user. These authentication mechanisms are optional for the link establishment. NSAL makes use of framework APIs to send/receive data from the Cellular hardware module. NSAL allows the cellular device driver to be re-used without any changes specific to the network stack.

Cellular Framework uses the AT command set to interact with the Cellular modem using the serial driver. The serial interface used to interact with the modem is UART. The UART speed used in the framework defaults are up to 115200bits/sec.

Cellular Framework Module Operational Description

From the above figure, illustrating the user application perspective, the application can be used in two different paths for the communication using the framework depending on the support available on the Cellular modems. Some modules provide options to use the TCP/IP stack at the Host end and other modules provide options to use the TCP/IP stack present on the Cellular modem itself. In some cases, cellular hardware module provides both. When the host TCP/IP stack (NetX) is used, the logical layers of NetX, NSAL, PPP are used as described in the Architecture diagram. When the on-chip stack is used, the Socket APIs are used to communicate with the TCP/IP stack present on the Cellular modem. However, the user cannot use both at the same time.

As shown below in the control flow diagram, during the initialization using the configuration supplied by the user as required for the Cellular modem, NetX nx_ip_create is called which internally invokes the NSAL driver entry function that takes care of the link level initialization and initializes the cellular hardware module. In addition, it provisions the module and establishes the Network connection using the PPP interface.

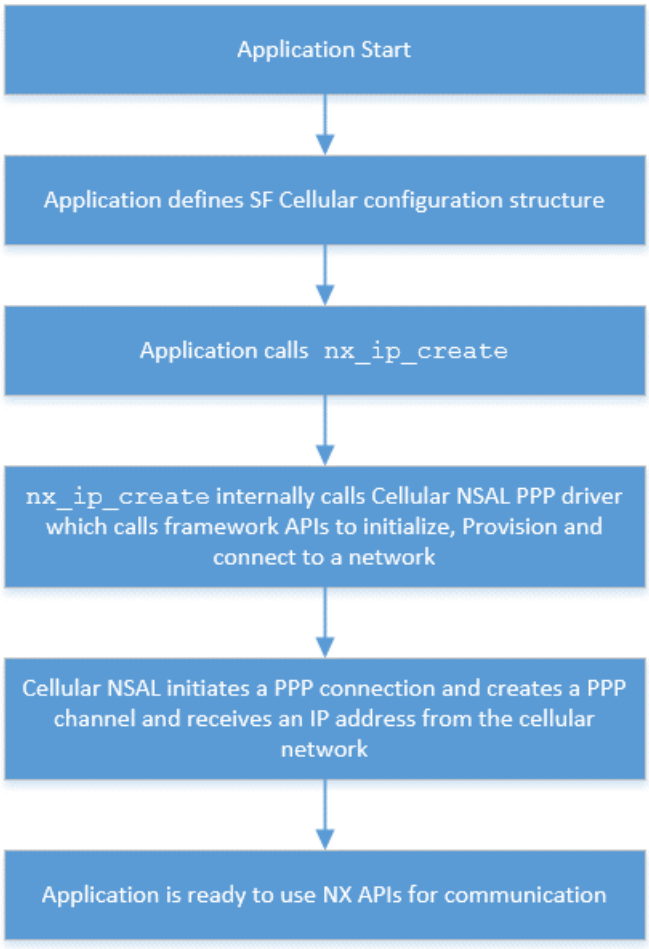


Figure 148: Cellular Framework Module Initialization Sequence

During the provisioning of the Cellular Modem, control structure and user configuration structures are passed as arguments. The details of the user arguments used for provisioning are the authentication, APN, username and password.

The following flow diagram shows the flow for the on-chip stack path usage with the Cellular Socket interface.

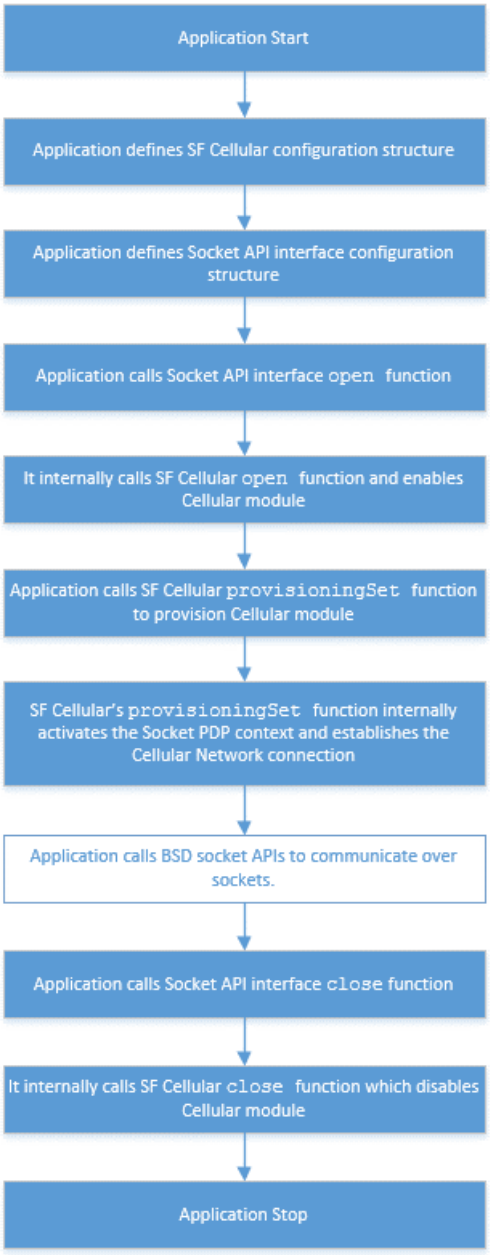


Figure 149: Cellular Framework Module Socket Interface

The following flow diagram shows the sequence of steps that the Packet transmission uses for the NetX application.

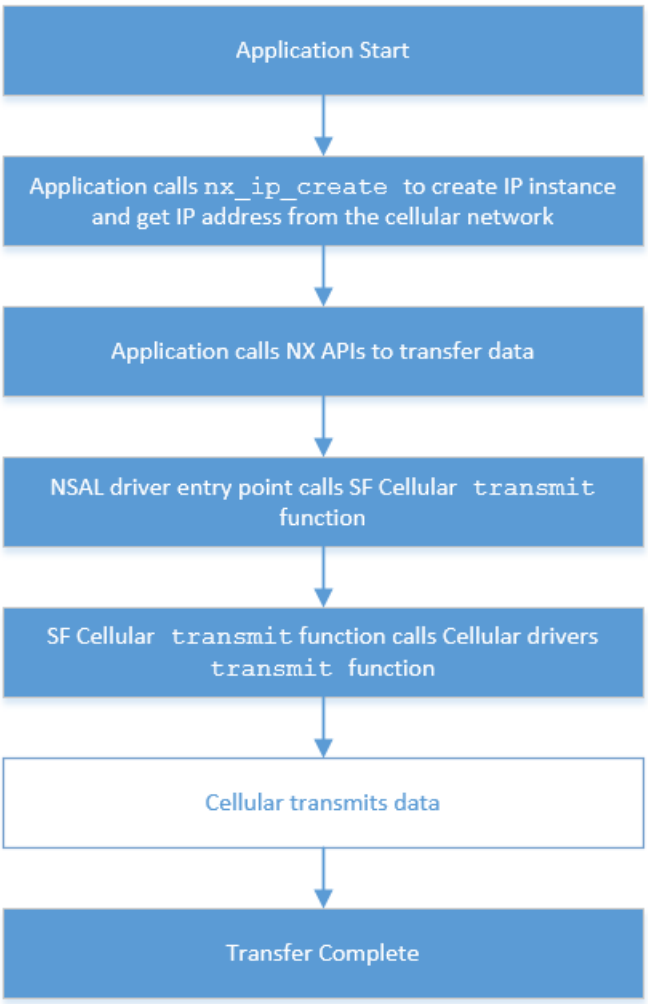


Figure 150: Cellular Framework Packet Transmission Sequence

The following flow diagram shows the Packet reception for the Cellular Framework using NetX. In the case of receive when the data is received on the serial interface, the processing thread triggers the callback function and the callback functions handles the data and sends it to the NetX layers for further processing.

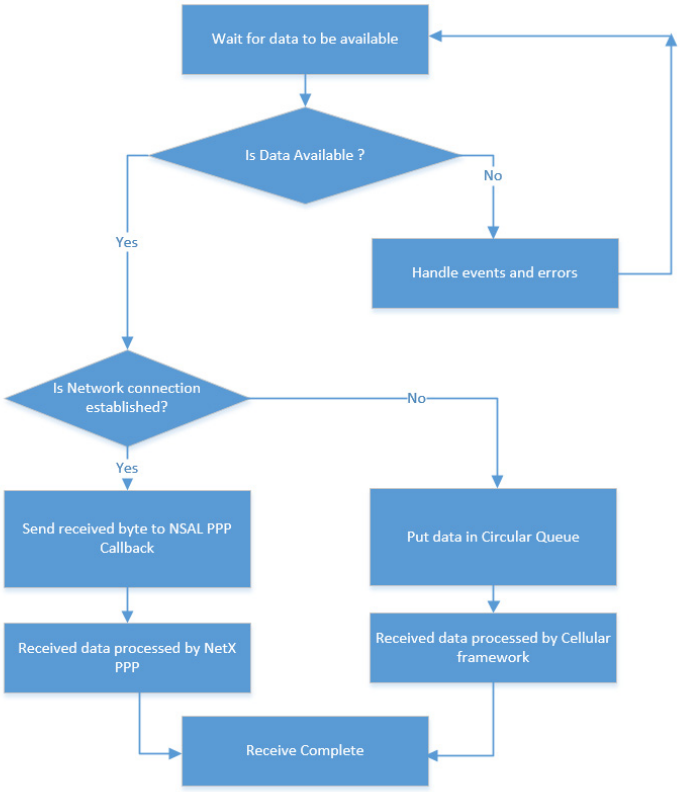


Figure 151: Cellular Framework Packet Reception Sequence

Cellular framework module important operational notes and limitations

- The current framework supports the below Cellular module:
- NimbeLink CAT3 (NL-SW-LTE-TSVG) Verizon-US
- NimbeLink CAT3 (NL-SW-LTE-TEUG) India and Europe
- NimbeLink CAT1 (NL-SW-LTE-GELS3-B and NL-SW-LTE-GELS3-C)
- Quectel CATM1 BG96 Verizon-US
- Firmware upgrade over air (FOTA) is not supported by NimbeLink CAT3 and CAT1 Cellular hardware module.
- Refer to the latest *SSP Release Notes* for any additional limitations for this module.

4.1.10.4 Including the Cellular Framework Module in an Application

This section describes how to include the Cellular Framework module in an application using the SSP configurator.

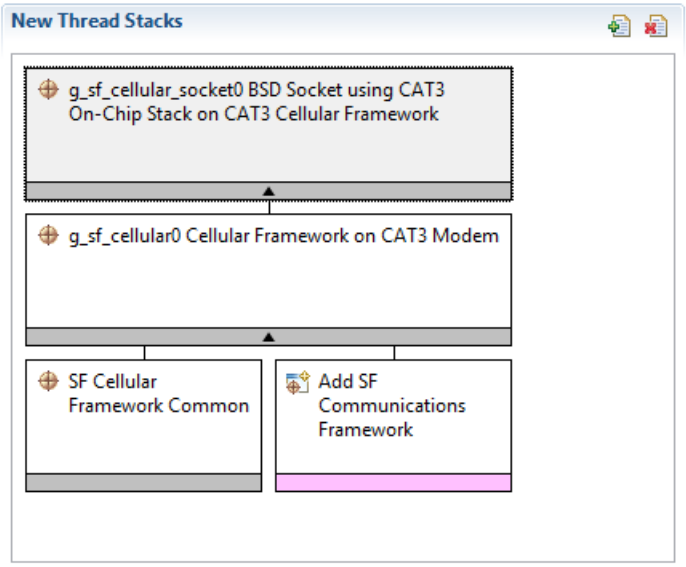
NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. Refer to the first few chapters of the SSP User's Manual if you are unfamiliar with any of these steps.

To add the Cellular Framework module to an application, simply add it to a thread using the stacks selection sequence given in the following table.

Cellular Framework Module Selection Sequences

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_cellular_socket0 BSD Socket using CAT3 On-Chip Stack on CAT3 Cellular Framework	Threads	New Stack> Framework> Networking> Cellular > BSD Socket using CAT3 On-Chip Stack on CAT3 Cellular Framework
g_sf_cellular_0 Cellular Framework on CAT3 Modem	Threads	New Stack> Framework> Networking> Cellular > Cellular Framework on CAT3 Modem
g_sf_el_nx0 NetX Port using Cellular Framework on sf_cellular_nsal_nx	Threads	New Stack> Framework> Networking> Cellular > NetX Port using Cellular Framework on sf_cellular_nsal_nx

When the Cellular framework module is added to the thread stack as shown in the following figure, the configurator automatically adds the needed lower-level drivers. Any drivers that need additional configuration information will be highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Pink band require the selection of an implementation option.



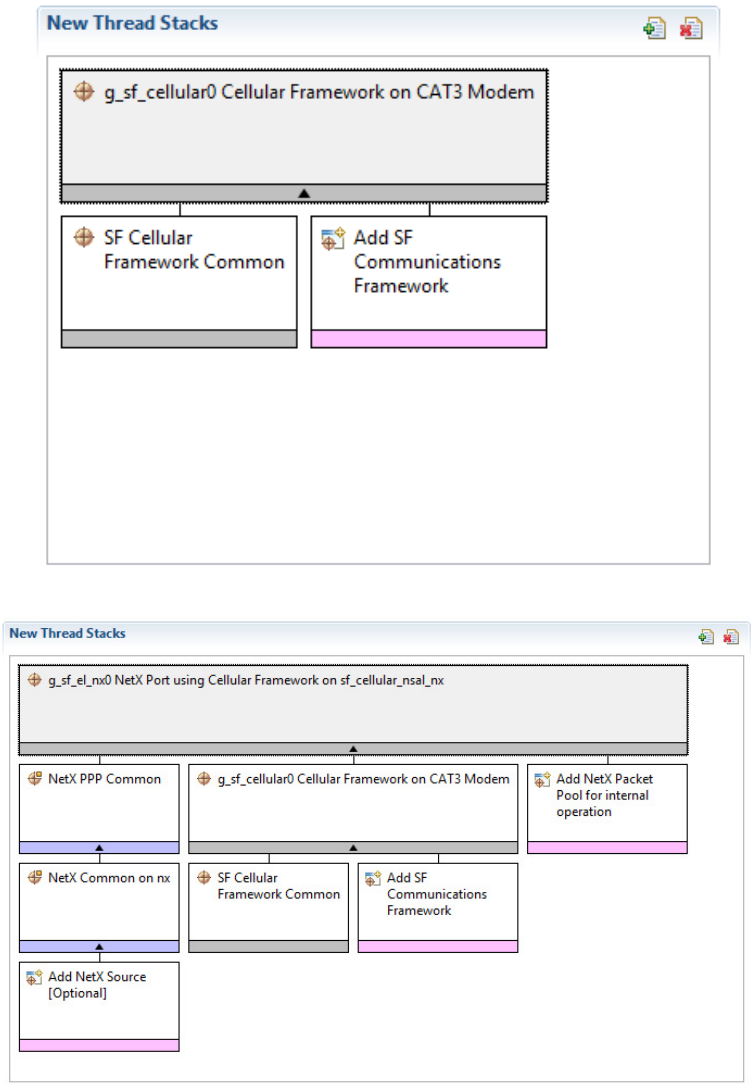


Figure 152: Cellular Framework Module Stack Options

Selecting SF_CELLULAR_NSAL_NX

The sf_cellular_nsal_nx implementation can be used in conjunction with a NetX IP Instance. In this case, it is added as an option underneath the IP Instance stack. Just add the NetX IP instance using the selection steps in the following table:

Resource	ISDE Tab	Stacks Selection Sequence
g_ip0 NetX IP Instance	Threads	New Stack> X-Ware> NetX> NetX IP Instance

Use the available option to add sf_cellular_nsal_nx under the NetX IP Instance as a NetX Network Driver. The resulting stack is shown in the following figure.

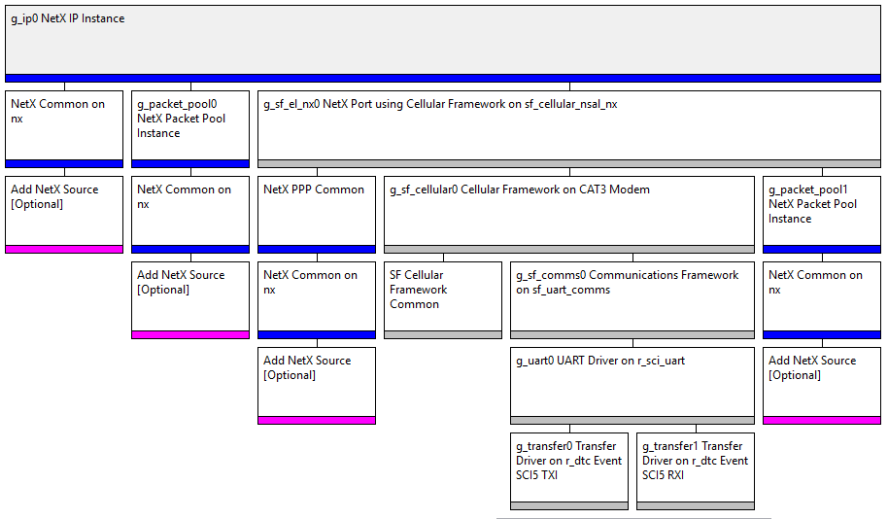


Figure 153: NetX IP Instance use of sf_cellular_nsal_nx

4.1.10.5 Configuring the Cellular Framework Modules

The Cellular framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify, by highlighting the block in red, any required configuration selections, such as Interrupts or operating modes, which must be configured for lower level modules, for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP Configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the Interrupt Priority. This configuration setting is available with the Properties window of the associated module. Simply select the indicated module and then view the properties window. The Interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the Interrupt Priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the below configuration properties tables, but is easily visible with the ISDE when configuring Interrupt Priority levels.

NOTE: You may want to open your ISDE and create the Cellular Framework and explore the property settings in parallel with looking over the Configuration Table Settings given below. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

A separate section is provided below for each of the Cellular framework module implementations.

Configuring the Cellular Framework on CAT3 Modem

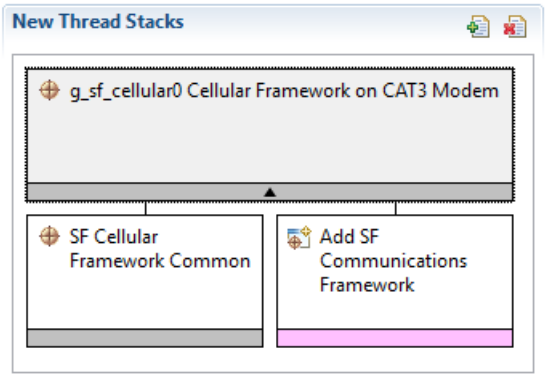


Figure 154: Cellular Framework on CAT3 Modem Thread Stack

Configuration Settings for the Cellular Framework on CAT3 Modem on sf_cellular

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking
On-Chip Stack Support	Enabled, Disabled Default: Disabled	On-chip stack support selection
Modem	TEUG, TSVG Default: TEUG	Modem selection
Name	g_sf_cellular0	Module name
SIM Pin (Used to Unlock SIM)	1111	SIM Pin selection
SIM PUK Pin (Used to Unlock SIM)	12345678	SIM PUK Pin selection
Number of Preferred Operator	0	Number of preferred operator selection
Preferred Operator 1 Name	40422	Preferred operator 1 name selection
Preferred Operator 1 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 1 name format selection

ISDE Property	Setting	Description
Preferred Operator 2 Name	40424	Preferred operator 2 name selection
Preferred Operator 2 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 2 name format selection
Preferred Operator 3 Name	40422	Preferred operator 3 name selection
Preferred Operator 3 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 3 name format selection
Preferred Operator 4 Name	40424	Preferred operator 4 name selection
Preferred Operator 4 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 4 name format selection
Preferred Operator 5 Name	40422	Preferred operator 5 name selection
Preferred Operator 5 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 5 name format selection
Operator Select Mode	Auto, Manual, Deregister, Manual Fallback Default: Auto	Operator select mode selection
Operator Name (Manual Mode Selection)	40422	Operator name selection
Operator Name Format (Manual Mode Selection)	Long, Short, Numeric Default: Numeric	Operator name format selection
Time Zone Update Policy	Enabled, Disabled Default: Enabled	Time zone update policy selection
Receive Data Callback	sf_cellular_nsal_recv_callback	Receive data callback selection

ISDE Property	Setting	Description
Provisioning Callback	celr_prov_callback	Provisioning callback selection
Circular Queue Size in Bytes	256	Circular queue size selection
SF Communications Framework Thread Stack Size	512	SF communications framework thread stack size selection
Numerical priority of SF Communication Framework Thread. Legal values range from 0 through (TX_MAX_PRIORITIES-1), where a value of 0 represents the highest priority.	5	Numerical priority of SF communication framework thread selection
Cellular Module Reset IO Pin	IOPORT_PORT_10_PIN_05	Cellular module reset IO pin selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SF Cellular Framework Common

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuring the BSD Socket using CAT3 On-Chip Stack on CAT3 Cellular Framework

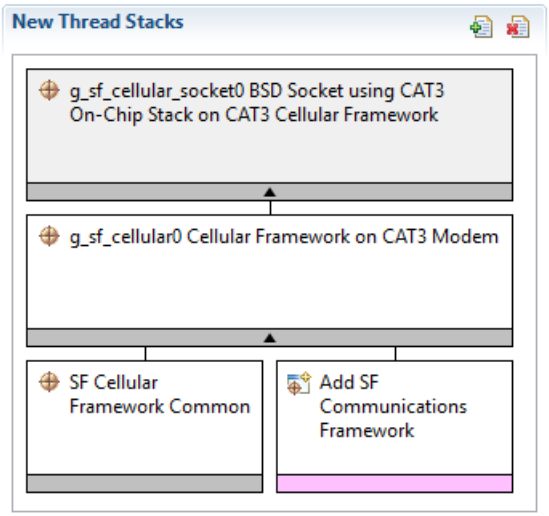


Figure 155: BSD Socket using CAT3 On-Chip Stack on CAT3 Cellular Framework Thread Stack

Configuration Settings for the BSD Socket using CAT3 On-Chip Stack on CAT3 Cellular Framework

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking
Name	g_sf_cellular	Module name

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Cellular Framework on CAT3 Modem

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking

ISDE Property	Setting	Description
On-Chip Stack Support	Enabled, Disabled Default: Disabled	On-chip stack support selection
Modem	TEUG, TSVG Default: TEUG	Modem selection
Name	g_sf_cellular0	Module name
SIM Pin (Used to Unlock SIM)	1111	SIM Pin selection
SIM PUK Pin (Used to Unlock SIM)	12345678	SIM PUK Pin selection
Number of Preferred Operator	0	Number of preferred operator selection
Preferred Operator 1 Name	40422	Preferred operator 1 name selection
Preferred Operator 1 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 1 name format selection
Preferred Operator 2 Name	40424	Preferred operator 2 name selection
Preferred Operator 2 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 2 name format selection
Preferred Operator 3 Name	40422	Preferred operator 3 name selection
Preferred Operator 3 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 3 name format selection
Preferred Operator 4 Name	40424	Preferred operator 4 name selection
Preferred Operator 4 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 4 name format selection
Preferred Operator 5 Name	40422	Preferred operator 5 name selection

ISDE Property	Setting	Description
Preferred Operator 5 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 5 name format selection
Operator Select Mode	Auto, Manual, Deregister, Manual Fallback Default: Auto	Operator select mode selection
Operator Name (Manual Mode Selection)	40422	Operator name selection
Operator Name Format (Manual Mode Selection)	Long, Short, Numeric Default: Numeric	Operator name format selection
Time Zone Update Policy	Enabled, Disabled Default: Enabled	Time zone update policy selection
Receive Data Callback	sf_cellular_nsal_recv_callback	Receive data callback selection
Provisioning Callback	celr_prov_callback	Provisioning callback selection
Circular Queue Size in Bytes	256	Circular queue size selection
SF Communications Framework Thread Stack Size	512	SF communications framework thread stack size selection
Numerical priority of SF Communication Framework Thread. Legal values range from 0 through (TX_MAX_PRIORITIES-1), where a value of 0 represents the hightest priority.	5	Numerical priority of SF communication framework thread selection
Cellular Module Reset IO Pin	IOPORT_PORT_10_PIN_05	Cellular module reset IO pin selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SF Cellular Framework Common

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuring the NetX Port using the Cellular Framework

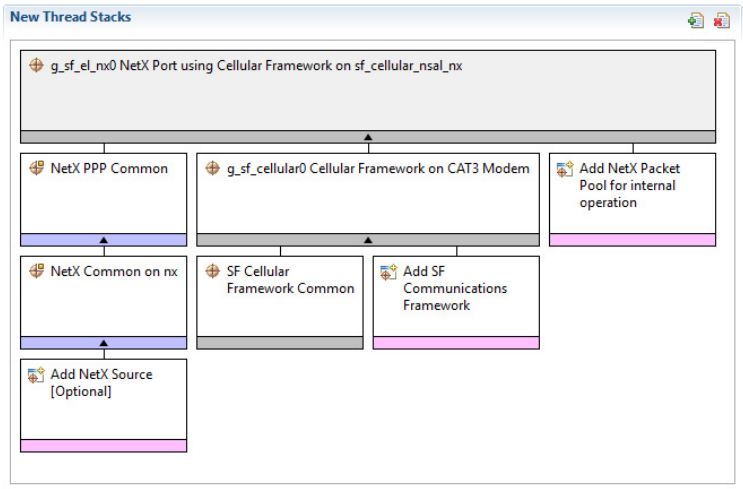


Figure 156: NetX Port using Cellular Framework Thread Stack

Configuration Settings for the Cellular Framework on sf_cellular_nsal_nx

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking
Name	g_sf_el_nx0	Module name
PPP Stack Size in Bytes	2048	PPP stack size selection
Name	g_nx_ppp0	Module name

ISDE Property	Setting	Description
Numerical priority of PPP Thread (Priority must be lower than IP Helper thread). Legal values range from 0 through (TX_MAX_PRIORITIES-1), where a value of 0 represents the highest priority.	3	Numerical priority of PPP thread selection
Authentication Method	None, PAP, CHAP Default: None	Authentication method selection
Invalid Packet Handler Callback	NULL	Invalid packet handler callback selection
Link Down Callback	ppp_link_down_callback	Link down callback selection
Link Up Callback	ppp_link_up_callback	Link up callback selection
PAP Login Callback	NULL	PAP login callback selection
PAP Verify Login Callback	NULL	PAP verify login callback selection
Get Challenges Values Callback	NULL	Get challenges values callback selection
Get Responder Values Callback	NULL	Get responder values callback selection
Get Verification Callback	NULL	Get verification callback selection
Local IPv4 Address (use commas for separation)	0,0,0,0	Local IPv4 address selection
Peer IPv4 Address (use commas for separation)	0,0,0,0	Peer IPv4 address selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX PPP Common

ISDE Property	Setting	Description
Name	g_nx_ppp_common0	Module name

Configuration Settings for the Cellular Framework on CAT3 Modem

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking
On-Chip Stack Support	Enabled, Disabled Default: Disabled	On-chip stack support selection
Modem	TEUG, TSVG Default: TEUG	Modem selection
Name	g_sf_cellular0	Module name
SIM Pin (Used to Unlock SIM)	1111	SIM Pin selection
SIM PUK Pin (Used to Unlock SIM)	12345678	SIM PUK Pin selection
Number of Preferred Operator	0	Number of preferred operator selection
Preferred Operator 1 Name	40422	Preferred operator 1 name selection
Preferred Operator 1 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 1 name format selection
Preferred Operator 2 Name	40424	Preferred operator 2 name selection
Preferred Operator 2 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 2 name format selection
Preferred Operator 3 Name	40422	Preferred operator 3 name selection
Preferred Operator 3 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 3 name format selection
Preferred Operator 4 Name	40424	Preferred operator 4 name selection

ISDE Property	Setting	Description
Preferred Operator 4 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 4 name format selection
Preferred Operator 5 Name	40422	Preferred operator 5 name selection
Preferred Operator 5 Name Format	Long, Short, Numeric Default: Numeric	Preferred operator 5 name format selection
Operator Select Mode	Auto, Manual, Deregister, Manual Fallback Default: Auto	Operator select mode selection
Operator Name (Manual Mode Selection)	40422	Operator name selection
Operator Name Format (Manual Mode Selection)	Long, Short, Numeric Default: Numeric	Operator name format selection
Time Zone Update Policy	Enabled, Disabled Default: Enabled	Time zone update policy selection
Receive Data Callback	sf_cellular_nsal_recv_callback	Receive data callback selection
Provisioning Callback	celr_prov_callback	Provisioning callback selection
Circular Queue Size in Bytes	256	Circular queue size selection
SF Communications Framework Thread Stack Size	512	SF communications framework thread stack size selection
Numerical priority of SF Communication Framework Thread. Legal values range from 0 through (TX_MAX_PRIORITIES-1), where a value of 0 represents the highest priority.	5	Numerical priority of SF communication framework thread selection
Cellular Module Reset IO Pin	IOPORT_PORT_10_PIN_05	Cellular module reset IO pin selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SF Cellular Framework Common

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Packet Pool Instance

ISDE Property	Setting	Description
Name	g_packet_pool0	Module name
Packet Size (bytes)	128	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto initialization	Enable, Disable Default: Enable	Auto initialization selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Common

ISDE Property	Setting	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection

ISDE Property	Setting	Description
Auto initialization	Enable, Disable Default: Enable	Auto initialization selection

Cellular Framework Module Clock Configuration

The Cellular Framework module uses the clocks required for the specific selections of the low-level modules.

Cellular Framework Module Pin Configuration

The Cellular Framework module uses input and output pins depending on the selections of the low-level modules.

4.1.10.6 Using the Cellular Framework Module in an Application

In a typical Cellular application much of the work is done by SSP based on the configured Cellular module stack. When the IP instance along with the Cellular framework is added using the configurator it includes the PPP stack as part of the framework. In addition, it also includes the NSAL and Cellular device driver code. The auto generated code is responsible Cellular initialization.

The user added code is responsible for the data connections and for the ICMP ping. It is responsible for sending the Ping request to the user entered Public IP address and verifying the Ping response. Callback functions can be implemented for PPP link down, PP link up, and cellular provisioning.

The steps in using the Cellular framework module in a simple application are:

Step 0. Initialization from generated code

Step 1. Wait for link to come up using the nx_ip_status_check API

Step 2. Ping the network using the nx_icmp_ping API

Step 3. Check for Event flag using the nx_event_flags_set API

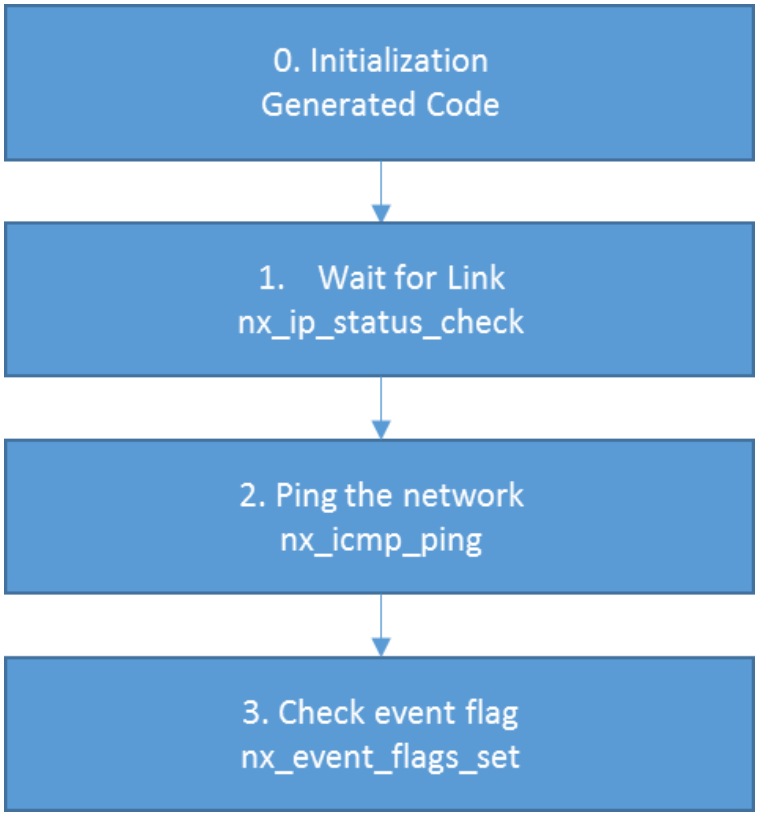


Figure 157: Flow Diagram of a Simple Cellular Framework Module Application

A more detailed description and a complete application project illustrating the typical steps in implementing a Cellular framework application are available in the Cellular Application Note downloadable from the Renesas web site. Just search, in the top search bar, for R30AN0311 and the application note and application project will be listed in the search results.

4.1.11 Communications Framework

The Communications Framework provides high-level APIs for communications applications using the ThreadX RTOS. The Framework is currently implemented on UART (sf_uart_comms), USBX CDC ACM (sf_el_ux_comms), and Telnet (sf_el_nx_comms). This section describes how to configure the Communications Framework using the e^2 studio ISDE and how to include the API functions in your application.

In the project configurator in the e^2 studio ISDE, you can add and configure the Communications Framework Module in the Modules pane of the Threads tab by selecting one of the following modules:

- **New > Framework > Connectivity > Communications Framework on sf_uart_comms**
- **New > Framework > Connectivity > Communications Framework on sf_el_ux_comms**
- **New > Framework > Connectivity > Communications Framework on sf_el_nx_comms**

For details, see: Using e^2 studio to Write an Application with the Communications Framework

You can find the API reference for the Communications Framework in the description of the Communications Framework Interface here: Communications Framework Interface.

To understand how to program with Interfaces in the SSP, see: SSP Architecture

4.1.11.1 What Does the Communications Framework Do?

This module is a ThreadX aware Communications framework. The module uses ThreadX objects like mutex for blocking and synchronization techniques like event flags for the completion of a transaction.

4.1.11.2 Using e^2 studio to Write an Application with the Communications Framework

The driver is integrated into the Synergy Software Package in e^2 studio: e^2 studio ISDE User Guide

In e^2 studio, create and configure a project and add the drivers:

1) Create the project: Creating a Project.
2) Configure the project: Configuring a Project
3) Add the drivers: Adding Drivers to a Thread and Configuring the Drivers

Select one of the sections below depending on which type of communications interface you are using (UART, NetX Telnet Server, or USBX CDC ACM).

Adding the UART Communications Framework

The following resources are required for any application using the UART Framework Interface:

Resource	ISDE Tab	Selection
UART Communications Framework	Threads	**Framework > Connectivity > Communications Framework on sf_uart_comms**

In addition, at least one UART HAL driver is required to create an UART application that can run on the Synergy microcontroller hardware.

Add the following resources when using the UART Framework with the SCI module:

Resource	ISDE Tab	Selection
Driver for UART on SCI	Threads	**UART Driver on r_sci_uart**
SCI Common driver	Threads	SCI Common; enable **Asynchronous Mode(r_sci_uart)** in the Properties window for this driver.
Interrupts	ICU	**SCI > SCIn > SCIn RXI**, **SCIn TXI**, **SCIn TEI**, and **SCIn ERIn** Interrupts for the selected SCI channel n.

Adding the USBX Communications Framework

The following resources are required for any application using the USBX Framework Interface:

Resource	ISDE Tab	Selection
Communications Framework	Threads	**Framework > Connectivity > Communications Framework on sf_el_ux_comms**

Add the following dependent resources when using the USBX CDC ACM Communications Framework with:

Resource	ISDE Tab	Selection
USBX Device Class CDC ACM	Threads	**Framework > USB > USBX Device Class CDC ACM on ux_device_class_cdc_acm**
USBX Stack	Threads	**Framework > USB > USBX on ux**
USBX Port	Threads	**Framework > USB > USBX Port on sf_el_ux**

Highlight the new Communications Framework on sf_el_ux_comms. The properties for this module are described in the table below. In the table, Property refers to the Properties tab name in the ISDE.

USBX Communications Framework configuration

ISDE Property	Setting	Description
Memory Size (Bytes)	Default: 65536	Memory passed to ux_system_initialize(). Recommendation is to start with 65536 if host mode will be used. If device-only macros are defined, recommendation is to start with 3072. Memory size should ultimately be determined by the application.
Read Input Buffer Size (Bytes)	Default: 128	This is the maximum number of bytes that can be received at a time in the read() API.

Adding the Telnet Communications Framework

The following resources are required for any application using the USBX Framework Interface:

Resource	ISDE Tab	Selection
Communications Framework	Threads	**Framework > Connectivity > Communications Framework on sf_el_nx_comms**

Add the following dependent resources when using the USBX CDC ACM Communications Framework with:

Resource	ISDE Tab	Selection
NetX Telnet Server	Threads	**Framework > Networking > NetX Telnet Server on nx**
NetX Stack	Threads	**Framework > Networking > NetX on nx**
NetX Port	Threads	**Framework > Networking > NetX Port on sf_el_nx**

Highlight the new Communications Framework on sf_el_nx_comms. The properties for this module are described in the table below. In the table, Property refers to the Properties tab name in the ISDE.

ISDE Property	Setting	Description
Channel	Default: 0	Underlying channel used by Ethernet driver
IP Address Byte < n >	192.168.0.0	IP address bytes
Subnet Mask Byte < n >	255.255.255.0	Subnet mask bytes

4.1.11.3 Writing a Communications Framework Application

The open call is generated by the ISDE src/synergy_gen/< thread_name >.c file where the module was added. By the time execution reaches src/< thread_name >_entry.c, the module is ready to use, provided the necessary hardware connection is established. To use the communications framework, make calls directly to the API's.

The ISDE configures this instance structure when you generate the project in the ISDE generated source file.

```
/* Instance structure to use this module. */

const sf_comms_instance_t g_sf_comms =

{
```

```
    .p_ctrl = &g_sf_comms_ctrl,

    .p_cfg = &g_sf_comms_cfg,

    .p_api = &g_sf_comms_on_sf_comms

};
```

The following is an example of a write call:

```
char p_src[] = "Hello World!\\r\\n";

g_sf_comms.p_api->write(g_sf_comms.p_ctrl, p_src, sizeof(p_src) - 1, TX_NO_WAIT)
;
```

The following is an example of a blocking read call:

```
char p_dest[10];

g_sf_comms.p_api->read(g_sf_comms.p_ctrl, p_dest, 10, TX_WAIT_FOREVER);
```

4.1.11.4 Limitations for the Communications Framework

There are no know limitations for using this module. For details about the current software version, see the SSP release notes.

4.1.11.5 Supported Devices for the Communications Framework

This communications framework modules has been tested on the Synergy microcontroller family S7G2.

The UART Communications framework is designed to support any Synergy device with an SCI UART and HAL driver with no changes to the API.

The USBX CDC Communications framework is designed to support any Synergy device with a USB controller.

The NetX Telnet Communications framework is designed to support any Synergy device with an Ethernet controller.

4.1.12 Communications Framework on NX

The Communications Framework on NX provides high-level APIs for telnet communications applications and uses the Ethernet peripheral on the Synergy MCU.

4.1.12.1 Express Logic NetX Telnet Communications Framework Module Features

- High level connectivity is supported on Ethernet but is easily changeable to UART and USB connectivity without API modification
- Supports channel locking for exclusive access
- Thread aware implementation uses mutex and event flags internally

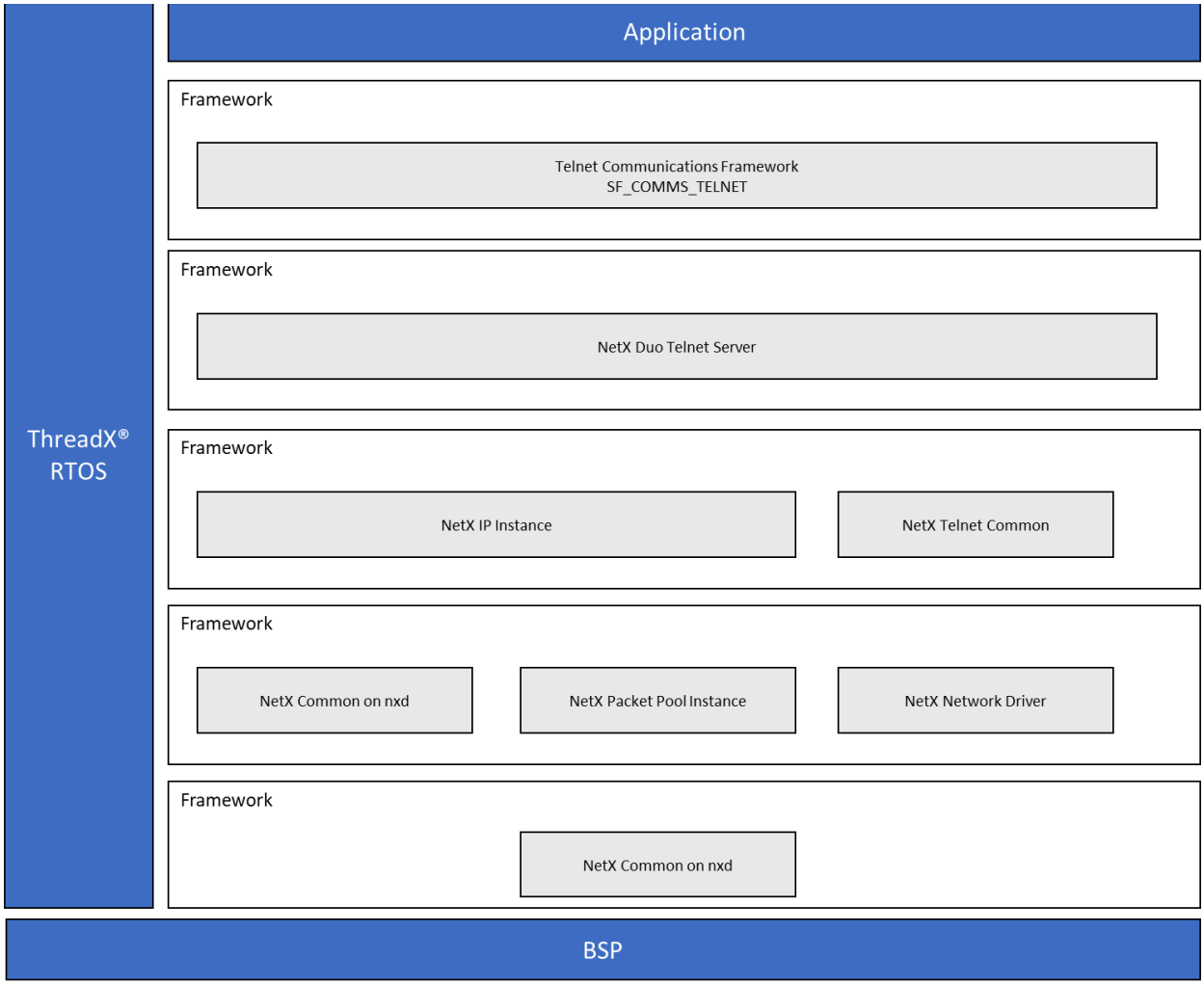


Figure 158: Express Logic NetX Telnet Communications Framework Module Block Diagram

4.1.12.2 Express Logic NetX Telnet Communications Framework Module APIs Overview

The Express Logic NetX Telnet Communications Framework module defines APIs to open, read, write, lock, unlock and close the module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Express Logic NetX Telnet Communications Framework Module API Summary

Function Name	Example API Call and Description
.open	g_sf_comms_telnet0.p_api->open (g_sf_comms_telnet0.p_ctrl, g_sf_comms_telnet0.p_cfg); Initializes module.
.read	g_sf_comms_telnet0.p_api->read (g_sf_comms_telnet0.p_ctrl, p_dest, bytes, timeout); Read a number of bytes of data into the destination.
.write	g_ sf_comms_telnet0.p_api->write (g_sf_comms_telnet0.p_ctrl, P_src, bytes, timeout); Write a number of bytes from the source.
.lock	g_sf_comms_telnet0.p_api->lock (g_sf_comms_telnet0.p_ctrl, locktype, timeout); Acquire lock type for the Telnet comms instance.
.unlock	g_sf_comms_telnet0.p_api->unlock (g_sf_comms_telnet0.p_ctrl, locktype); Release the lock type for the Telnet comms inctance.
.close	g_sf_comms_telnet0.p_api->close (g_sf_comms_telnet0.p_ctrl); Disconnect Telnet server and clean up resources.
.versionGet	g_ sf_comms_telnet0.p_api->versionGet(&version); Gets version and stores it in provided version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_INTERNAL	An internal TheadX error has occurred.
SSP_ERR_NOT_OPEN	Unit is not open
SSP_ERR_ASSERTION	A parameter is NULL.
SSP_ERR_IN_USE	Peripheral is still running in another mode; perform Close first.
SSP_ERR_UNSUPPORTED	Command not supported.
SSP_ERR_OUT_OF_MEMORY	Can't allocate pool memory.
SSP_ERR_TIMEOUT	An event timed out.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.1.12.3 Express Logic NetX Telnet Communications Framework Module Operational Overview

The Communications Framework using Telnet on NX provides an easy-to-use connection over the ethernet port. The high-level APIs are compatible with other connection implementations, such as UART and USB, so that it is easy to switch from one implementation to another without changing APIs. Supported operations include opening the module using the open API and closing the module using the close API. Read is implemented by the read and write by the write API. Unlike read and write, which locks the module only until the called API is in action, the lock API locks the module till the unlock API is called on the same module instance. For the underlying NetX driver, it also lets the user configure the IP address and network mask and choose the Ethernet channel.

Express Logic NetX Telnet Communications Framework Module Important Operational Notes and Limitations

- The Ethernet peripheral can use either RMII or MII depending on MCU capabilities.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.12.4 Including the Express Logic NetX Telnet Communications Framework Module in an Application

This section describes how to include the Express Logic NetX Telnet Communications Framework module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Express Logic NetX Telnet Communications Framework module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the Express Logic NetX Telnet Communications Framework module is sf_comms_telnet0. This name can be changed in the associated Properties window.)

Express Logic NetX Telnet Communications Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
sf_comms_telnet0 Express Logic NetX Telnet Communications Framework on sf_comms_telnet	Threads	New Stack> Framework> Connectivity> Express Logic NetX Telnet Communications Framework on sf_comms_telnet

When the Express Logic NetX Telnet Communications Framework module on sf_comms_telnet is added to the thread stack as shown in the following figure, the configurator automatically adds any lower-level drivers that are required. Any drivers that need additional configuration information will be box-text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

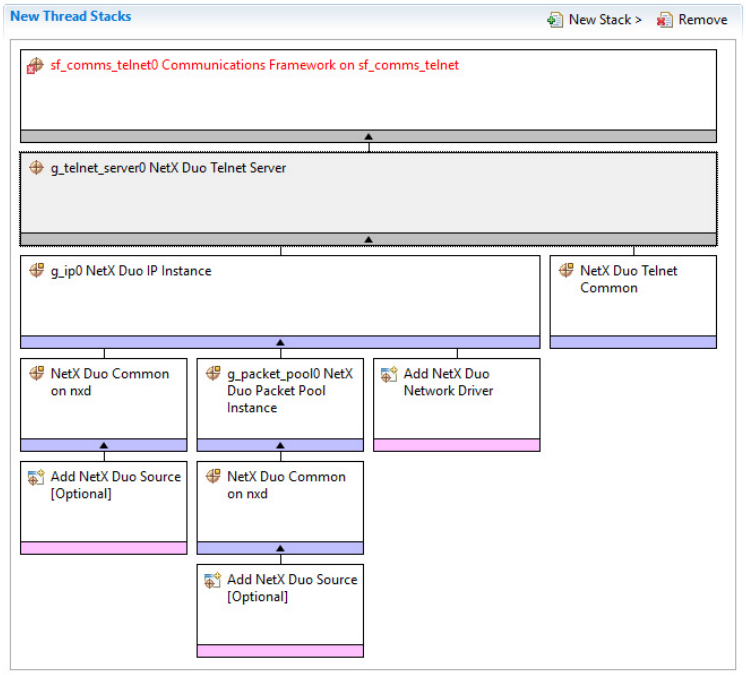


Figure 159: Express Logic NetX Telnet Communications Framework Module Stack

4.1.12.5 Configuring the Express Logic NetX Telnet Communications Framework Module

The Express Logic NetX Telnet Communications Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the Express Logic NetX Telnet Communications Framework Module on sf_comms_telnet

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking
Packet size in pool memory (bytes)	1536	Packet size in pool memory selection
Packets to allocate in pool memory (units)	5	Packets to allocate in pool memory selection
Timeout for internal options (ticks)	10	Timeout for internal options selection
Maximum number of instances	4	Maximum instances that can be open at any given time
Name	sf_comms_telnet0	Module name
Name of generated initialization function	sf_comms_telnet_init0	Name of generated initialization selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX (Duo) Telnet Server on telnet_server

ISDE Property	Value	Description
Internal thread priority	16	Internal thread priority selection
Maximum clients to serve simultaneously	4	Maximum clients to serve simultaneously selection
Socket window size (bytes)	2048	Socket window size selection
Server time out (seconds)	10	Duration internal services will suspend for
Client inactivity timeout (seconds)	600	Client inactivity duration for disconnection

ISDE Property	Value	Description
Timeout check period (seconds)	60	Client activitiy timeout check interval
Option negotiation	Enable, Disable Default: Enable	Option negotiation selection
Use application packet pool	Enable, Disable Default: Disable	Use application packet pool selection
Packet size in the pool (bytes)	300	Telnet Server only creates this packet pool if 'Option negotiation' is enabled
Total packet pool size (bytes)	2048	Telnet Server only creates this packet pool if NX_TELNET_SERVER_OPTION_DISABLE
Name	g_telnet_server0	Module name
Thread Stack Size (bytes)	2048	Thread stack size selection
Name of Client Connect Callback Function	NULL	Name of client connect callback function selection
Name of Receive Data Callback Function	NULL	Name of receive data callback function selection
Name of Client Disconnect Callback Function	NULL	Name of client disconnect callback function selection
Name of generated initialization function	telnet_server_init0	Name of generated initialization function selection
Auto Initialization	Disable	Auto initialization selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX (Duo) IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name

ISDE Property	Value	Description
IPv4 Address (use commas for separation)	0,0,0,0	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable	TCP selection
UDP	Enable, Disable Default: Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection

ISDE Property	Value	Description
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection
Link status change callback	NULL	Link status change callback selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Telnet Common

ISDE Property	Value	Description
Type of Service for TCP requests	Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost Default: Normal	Type of service UDP requests selection
Fragmentation option	Don't fragment, Fragment okay Default: Don't fragment	Fragment option selection
Server TCP port number	23	Server TCP port number selection
Time to live	128	Time to live selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX (Duo) Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	640	Packet size selection

ISDE Property	Value	Description
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Common on nx

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Express Logic NetX Telnet Communications Framework Module Clock Configuration

The Express Logic NetX Telnet Communications Framework module uses the Ethernet peripheral which uses PCLKA as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

Express Logic NetX Telnet Communications Framework Module Pin Configuration

To use the Express Logic NetX Telnet Communications Framework module, the port pins for the peripheral inputs and outputs must be set in the pin configurator in the ISDE. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection Sequence for the Express Logic NetX Telnet Communications Framework Module

Resource	ISDE Tab	Pin selection Sequence
SSI	Pins	Select Peripherals > Peripherals>Connectivity:ETHERC> ETHERC0.RMI or ETHERC1.RMII

4.1.12.6 Using the Express Logic NetX Telnet Communications Framework Module in an Application

The typical steps in using the Communications Framework on NX in an application are:

1) Initialize the Communications Framework on NX using the open API
2) Lock the channel for continuous communications using the lock API if needed
3) Receive data using the read API
4) Send data using the write API
5) Unlock the channel from continuous communication using the unlock command if needed
6) Close the channel using the close API

These common steps are illustrated in a typical operational flow in the following figure:

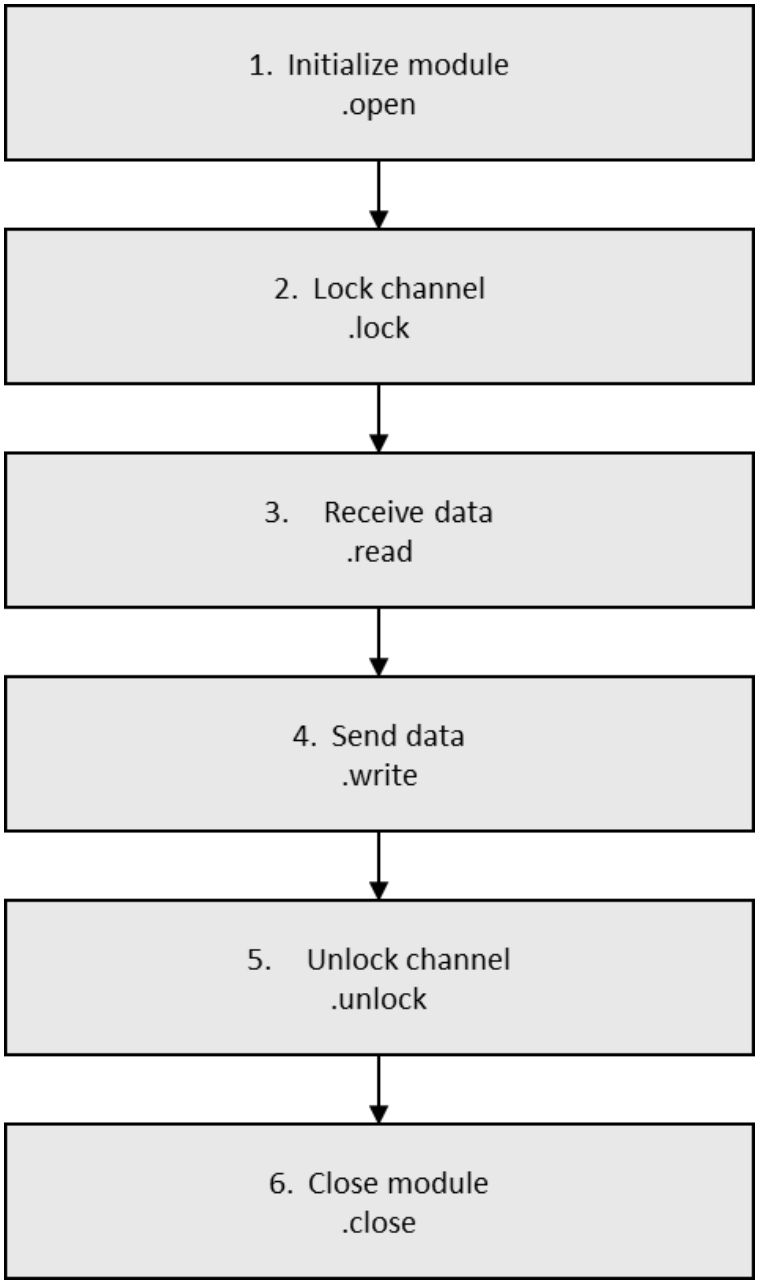


Figure 160: Flow Diagram of a Typical Express Logic NetX Telnet Communications Framework Module Application

4.1.13 UART Communications Framework

The UART communications framework provides an instance of the synergy communication framework APIs on an UART compliant synergy MCU peripheral. It uses r_sci_uart HAL driver, which configures and operates the SCI peripheral of Synergy MCUs to communicate using the UART protocol.

4.1.13.1 UART Communications Framework Module Features

This module is a ThreadX-aware communications framework. It uses ThreadX objects to ensure that the operations are thread safe. Key features include:

- Support for UART Communications Protocol
- Support for locking a channel to reserve exclusive access
- ThreadX aware implementation

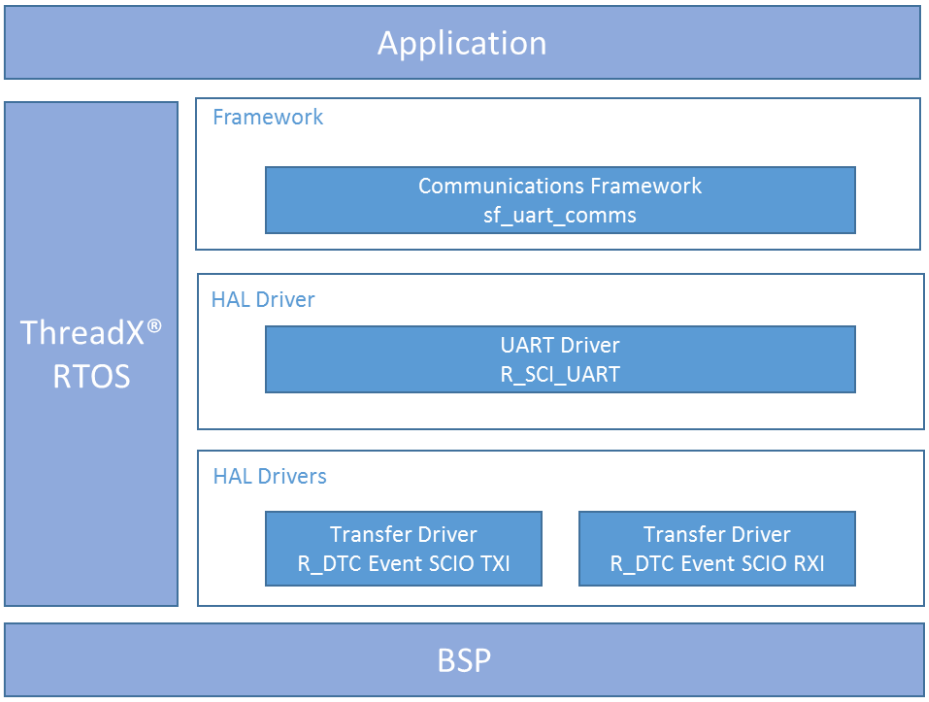


Figure 161: UART Communications Framework Organization, Options and Stack Implementations

4.1.13.2 UART Communications Framework Module APIs Overview

The UART Communications Framework module defines APIs for opening, closing, reading, and writing to the communications channel. A complete list of the available APIs, an example API call, and a short description of each can be found in the table below. A table of status return values follows.

UART Communications Framework API Summary

Function Name	Example API Call and Description
open	g_sf_comms0.p_api->open(g_sf_comms0.p_ctrl, g_sf_comms0.p_cfg); Initialize communications driver.
close	g_sf_comms0.p_api->close(g_sf_comms0.p_ctrl); Clean up communications driver.
read	g_sf_comms0.p_api->read(g_sf_comms0.p_ctrl, &destination, bytes, timeout); Read data from communications driver. This call will return after the number of bytes requested is read or if a timeout occurs while waiting for access to the driver.
write	g_sf_comms0.p_api->write(g_sf_comms0.p_ctrl, &source, bytes, timeout); Write data to communications driver. This call will return after all bytes are written or if a timeout occurs while waiting for access to the driver.

Function Name	Example API Call and Description
lock	g_sf_comms0.p_api->lock(g_sf_comms0.p_ctrl, lock_type, timeout); Lock the communications driver. Reserve exclusive access to the communications driver.
unlock	g_sf_comms0.p_api->unlock(g_sf_comms0.p_ctrl, lock_type); Unlock the communications driver. Release exclusive access to the communications driver.
versionGet	g_sf_comms0.p_api->version(&version); Store the driver version in the provided version.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manuals API References for the associated module.

Error Status Return Values

Name	Description
SSP_SUCCESS	Channel opened successfully.
SSP_ERR_IN_USE	Channel already in use.
SSP_ERR_ASSERTION	Pointer to UART control block or configuration structure is NULL.
SSP_ERR_HW_LOCKED	Channel is locked.
SSP_ERR_INVALID_MODE	Channel is used for non-UART mode or illegal mode is set.

Name	Description
SSP_ERR_INVALID_ARGUMENT	Invalid parameter setting found in the configuration structure.
SSP_ERR_QUEUE_UNAVAILABLE	Cannot open transmit or receive queue or both.
SSP_ERR_INTERNAL	Internal error occurs.
SSP_ERR_TIMEOUT	Timeout error.
SSP_ERR_INSUFFICIENT_DATA	Not enough data in receive circular buffer.
SSP_ERR_RXBUF_OVERFLOW	Receive queue overflow.
SSP_ERR_OVERFLOW	Hardware overflow.
SSP_ERR_FRAMING	Framing error.
SSP_ERR_PARITY	Parity error.
SSP_ERR_INSUFFICIENT_SPACE	Not enough space in transmission circular buffer.

NOTE: Lower level drivers may return Common Error Codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.13.3 UART Communications Framework Module Operational Overview

The UART Framework provides an easy-to-use communication framework using the standard UART protocol. In addition to the high-level APIs to read and write data from the UART device in a thread safe manner, the framework also provides APIs for applications to lock (and unlock) the UART channel to a thread. This is particularly useful when multiple application threads try to communicate with the same UART device and a context switch could upset the high level application protocols and/or state machines implemented on top of the UART (like Kermit, for example).

Important Operational Notes and Limitations for the UART Communications Framework Module

- The UART Framework module is reentrant for any channel.
- The UART framework uses the UART driver on r_sci_uart module for communicating with a UART device. UART driver can be augmented by adding DTC drivers (using synergy configurator in the ISDE) to perform read/write transactions with a UART device without interrupting the CPU. When the UART framework is used to read data from a UART device, it will rely on the UART driver callback feature to read data and **will not use DTC** (Even if the DTC module support is added to the UART driver through the configurator). This is done to avoid any potential timing and synchronization issues that could arise when the driver uses DTC to read data from the device. When using UART framework to write data to a UART device, it will use the DTC to perform the transaction, if the driver is configured to use DTC.
- The ongoing read/write transactions will be aborted when timeout occurs.

- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.13.4 Including the UART Communications Framework Module in an Application

This section describes how to include the UART Communications Framework module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the "Getting Started Guide for SSP" listed in the Reference Section at the end of this document to learn how to manage each of these important steps in creating SSP based applications.

The UART Communications Framework module is implemented on sf_uart_comms. To add the UART Communications Framework module to an application, simply add it to a thread using the Stack Selection Sequence given in the table below. (The default name for the Communications Framework is g_sf_comms0. This name can be changed in the associated Properties window.)

Communications Framework Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_comms0 Communications Framework on sf_uart_comms	Threads	Framework > Connectivity > Communications Framework on sf_uart_comms

When the Communications Framework on Communications Framework is added to the thread stack as shown in the figure below, the configurator automatically adds any needed lower level modules. Any drivers that need additional configuration information will have box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower level drivers; these are either optional or recommended (this is indicated in the block with the inclusion of this text.) If the addition of lower level drivers is required, the module description will include "Add" in the text. Clicking on any Pink banded modules will bring up the "New" icon and then display the possible choices.

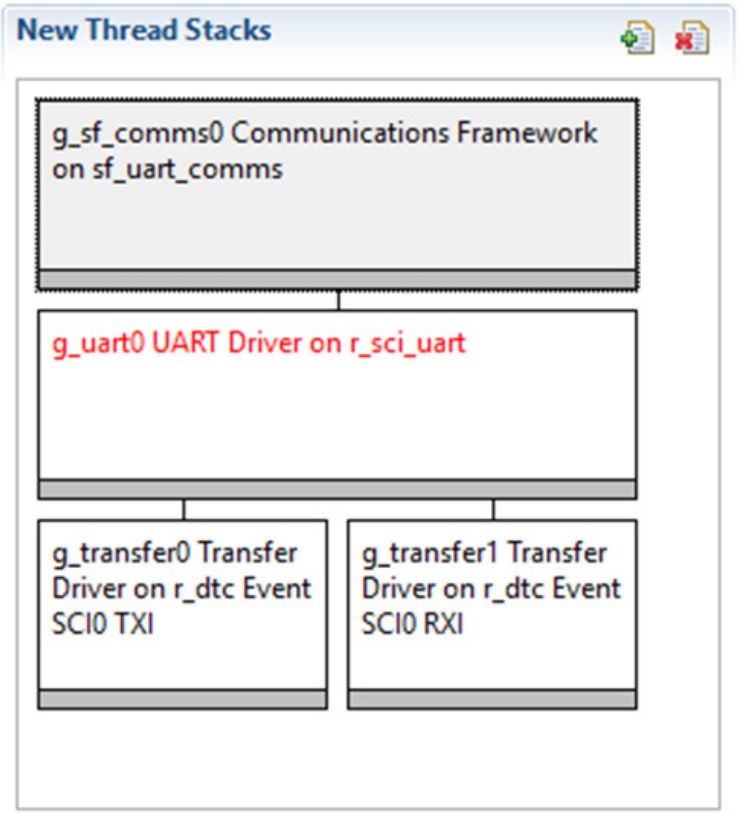


Figure 162: UART Communications Framework Stack Implementation Options

4.1.13.5 Configuring the UART Communications Framework Module

The UART Communications Framework module must be configured by you for the desired operation. The SSP configuration window will automatically identify, by highlighting the block in red, any required configuration selections, such as Interrupts or operating modes, which must be configured for lower level modules, for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority. This configuration setting is available with the properties window of the associated module. Simply select the indicated module and then view the properties window. The Interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the Interrupt priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the below configuration properties tables, but is easily visible with the ISDE when configuring Interrupt Priority levels.

NOTE: You may want to open your ISDE and create the module and explore the property settings in parallel with looking over the Configuration Table Settings given below. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for UART Communications Framework Stack Modules

When the UART Communications Framework stack option is selected for the implementation the following modules are automatically added to the stack. The following tables show the detailed configuration settings available- many of which are populated with default settings that can be used in most common applications.

Configuration Settings for UART Communications Framework (sf_uart_comms)

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if parameter checking is included.
Read Input Queue Size (4-Byte Words)	15	Buffer size for data reception queue. sf_uart_comms utilizes the ThreadX Queue for the queue management.
Name	g_sf_comms0	Name of UART communications framework module.

Configuration Settings for UART Driver (r_sci_uart)

ISDE Property	Value	Description
External RTS Operation	Enable, Disable Default: Disable	Enable an IOPORT pin to be used as RTS signal. For RTS functionality set this configuration parameter to "Enable" and specify the configuration "Name of UART callback function for the RTS external pin control".
Reception	Enable, Disable Default: Enable	Enable or disable UART reception for all UART channels on SCI. Setting this configuration parameter to "Disable" reduces code size because the portion of code for UART reception is not compiled. You cannot set this parameter for individual UART channels.

ISDE Property	Value	Description
Transmission	Enable, Disable Default: Enable	Enable or disable UART transmission for all UART channels on SCI. Setting "Disable" to this configuration allows to get smaller code size due to the portion of code for UART transmission is compiled out, however, you can only set "Disable" to this configuration if any other SCI channels which work as UART ports do not perform the transmission.
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_uart0	The name to be used for UART on SCI module control block instance. This name is also used as the prefix of the other variable instances.
Channel	0-9	SCI channel number.
Baud Rate	9600	Baud rate selection.
Data Bits	7 bits, 8, bits, 9 bits Default: 8 bits	UART data bits.
Parity	None, Odd, Even Default: None	UART parity bits.
Stop Bits	1 bit, 2 bits Default: 1 bit	UART stop bits.

ISDE Property	Value	Description
CTS/RTS Selection	CTS (Note that RTS is available when enabling External RTS Operation mode which uses 1 GPIO pin), RTS (CTS is disabled) Default: RTS (CTS is disabled)	Select CTS or RTS for the CTSn/RTSn pin of SCI channel n. The SCI hardware supports either the CTS or the RTS control signal on this pin but not both. For an application that uses both CTS and RTS, select "CTS" for this configuration parameter and enable the configuration "External RTS Operation" specifying the configuration "Name of UART callback function for the RTS external pin control".
Name of UART callback function to be defined by user	user_uart_callback	Name must be a valid C symbol.
Name of UART callback function for the RTS external pin control to be defined by user	NULL	Name must be a valid C symbol.
Clock Source	Internal Clock, External Clock 8x baudrate, External Clock 16x baudrate Default: Internal Clock	Selection of the clock source to be used in the baud-rate clock generator block.
Baudrate Clock Output from SCK pin	Enable, Disable Default: Disable	Optional setting to output the baud-rate clock on the SCKn pin for the selected channel n.
Start bit detection	Falling Edge, Low Level Default: Falling Edge	Start bit detection mode in the reception, usually set "Falling Edge" to this configuration.
Noise Cancel	Enable, Disable Default: Disable	Enable the digital noise cancellation on RXDn pin. The digital noise filter block in SCI consists of two-stage flip-flop circuits. For detail, refer to the Noise cancellation section in the Renesas Synergy hardware manual.

ISDE Property	Value	Description
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bit rate modulation enable selection.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Transmit interrupt priority selecition.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Transmit end interrupt priority selection.

ISDE Property	Value	Description
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Error interrupt priority selection.

Configuration Settings for TXI Transfer Driver on r_dtc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection

ISDE Property	Value	Description
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 TXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

Configuration Settings for RXI Transfer Driver on r_dtc

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection

ISDE Property	Setting	Description
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SPI0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

UART Communications Framework Module Clock Configuration

The UART Communications Framework has no specific clock configuration requirements.

UART Communications Framework Module Pin Configuration

The UART Communications Framework uses pins on the MCU to communicate to external devices, based on the lower level implementation selected. I/O pins must be selected and configured as required by the external device. The first table below illustrates the method for selecting the pins within the SSP configuration window and the following table illustrates an example selection for the lower level implementation pins.

NOTE: The Operation Mode selection mode determines what peripheral signals are available and thus what MCU pins are required.

Pin Selection Sequence for Communications Framework on UART, USB or Telnet (Receiver)

Resource	ISDE Tab	Pin selection Sequence
UART	Pins	Select **Peripherals > Connectivity: SCI > SCI8**

NOTE: The above selection sequences are examples for selected implementations. Others are also possible depending on the target hardware.

Pin Configuration Settings for UART

Pin Configuration Property	Settings	Description
Pin Group Selection	Mixed, _A Only, _B Only	Pin group selection
Operation Mode	Disabled, Custom, Asynchronous UART, Synchronous UART, Simple I2C, Simple SPI, SmartCard (Default: Disabled)	Select Asynchronous UART as the Operation Mode for a UART Receiver implementation
TXD_MOSI	None, PB05, P105 (Default: None)	TXD Pin P105
RXD_MISO	None, PB05, P104 (Default: None)	RXD Pin P104

Pin Selection Sequence for Communications Framework on UART, USB or Telnet (Transmitter)

Resource	ISDE Tab	Pin selection Sequence
UART	Pins	Select **Peripherals > Connectivity: SCI > SCI3**

NOTE: The above selection sequences are examples for selected implementations. Others are also possible depending on the target hardware.

Pin Configuration Settings for UART

Pin Configuration Property	Settings	Description
Pin Group Selection	Mixed, _A Only, _B Only	Pin group selection
Operation Mode	Disabled, Custom, Asynchronous UART, Synchronous UART, Simple I2C, Simple SPI, SmartCard (Default: Disabled)	Select Asynchronous UART as the Operation Mode for a UART Transmitter implementation
TXD_MOSI	None, P707, P409 (Default: None)	TXD Pin P707
RXD_MISO	None, P706, P408 (Default: None)	RXD Pin P706

NOTE: The above example settings are for a project using the Synergy S7G2 and the DK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.1.13.6 Using the UART Communications Framework Module in an Application

The typical steps in using the UART Communications Framework module in an application are:

1) Initialize the UART Communications Framework using the open API
2) Lock the channel for continuous communications using the lock API if needed
3) Receive data using the read API
4) Send data using the write API
5) Unlock the channel from continuous communication using the unlock API if needed
6) Close the channel using the close API

The above steps are illustrated in a typical operational flow diagram in the figure below:

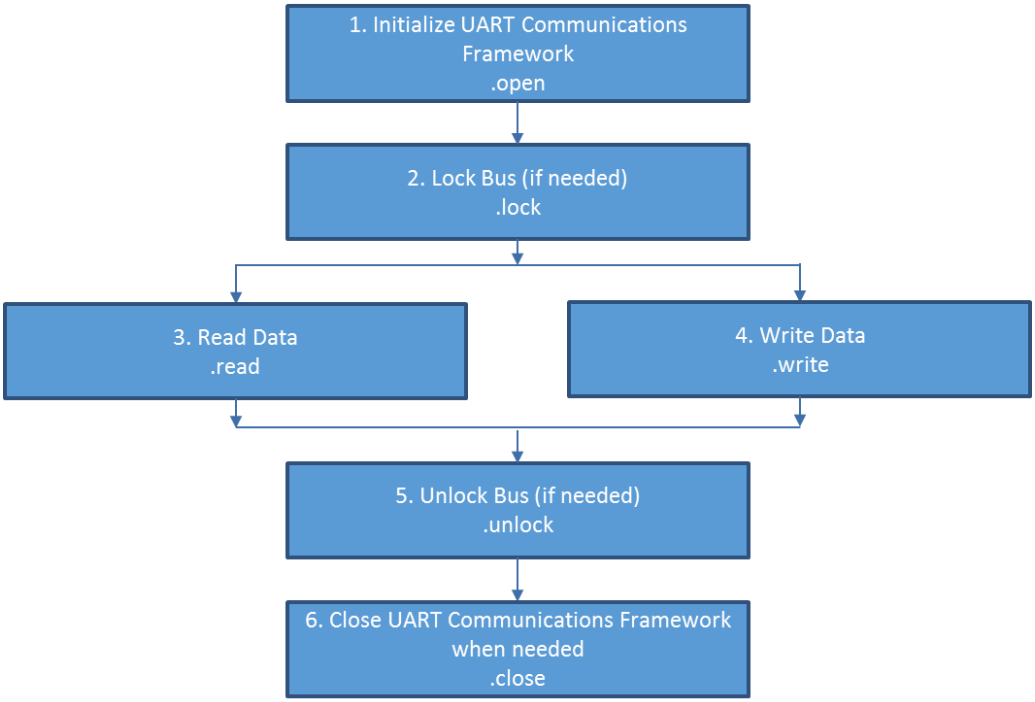


Figure 163: Flow Diagram of a Typical UART Communications Framework Application

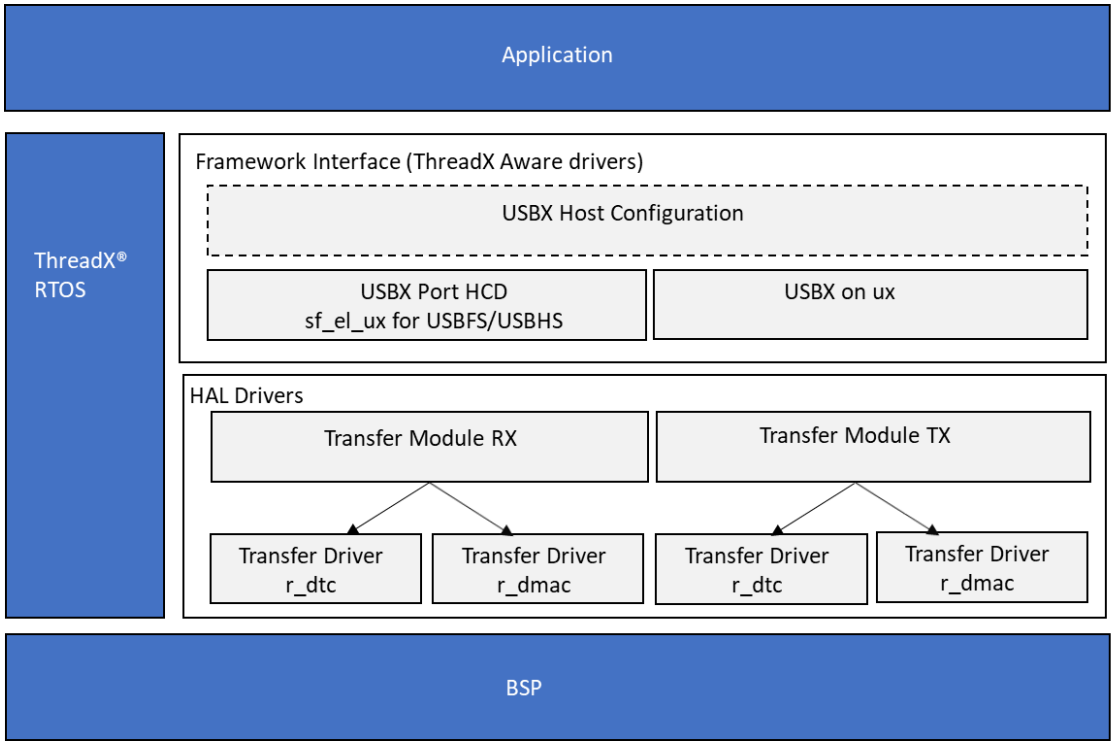
4.1.14 Communications Framework on USBX™

The Express Logic USBX Synergy Port framework module (sf_el_ux) is integrated into the SSP. This driver is meant to be used with Express Logic USBX. For more information about USBX, including API reference, refer to the USBX User Guide.

4.1.14.1 Express Logic USBX Synergy Port Framework Module Features

The Express Logic USBX Synergy Port Framework module supports the following features:

- Implements Express Logic USBX in SSP- support USBX APIs
- Supports the Port Device Controller Driver (DCD) for the USBHS peripheral
- Supports the Port Device Controller Driver (DCD) for the USBFS peripheral
- Supports the Port Host Controller Driver (HCD) for the USBHS peripheral
- Supports the Port Host Controller Driver (HCD) for the USBFS peripheral
- Supports transfer module operation (optional)


Application
ThreadX®
RTOS
Framework Interface (ThreadX Aware drivers)
USBX Host Configuration
USBX Port HCD
sf_el_ux for USBFS/USBHS
USBX on ux
HAL Drivers
Transfer Module RX
Transfer Module TX
Transfer Driver
r_dtc
Transfer Driver
r_dmac
Transfer Driver
r_dtc
Transfer Driver
r_dmac
BSP

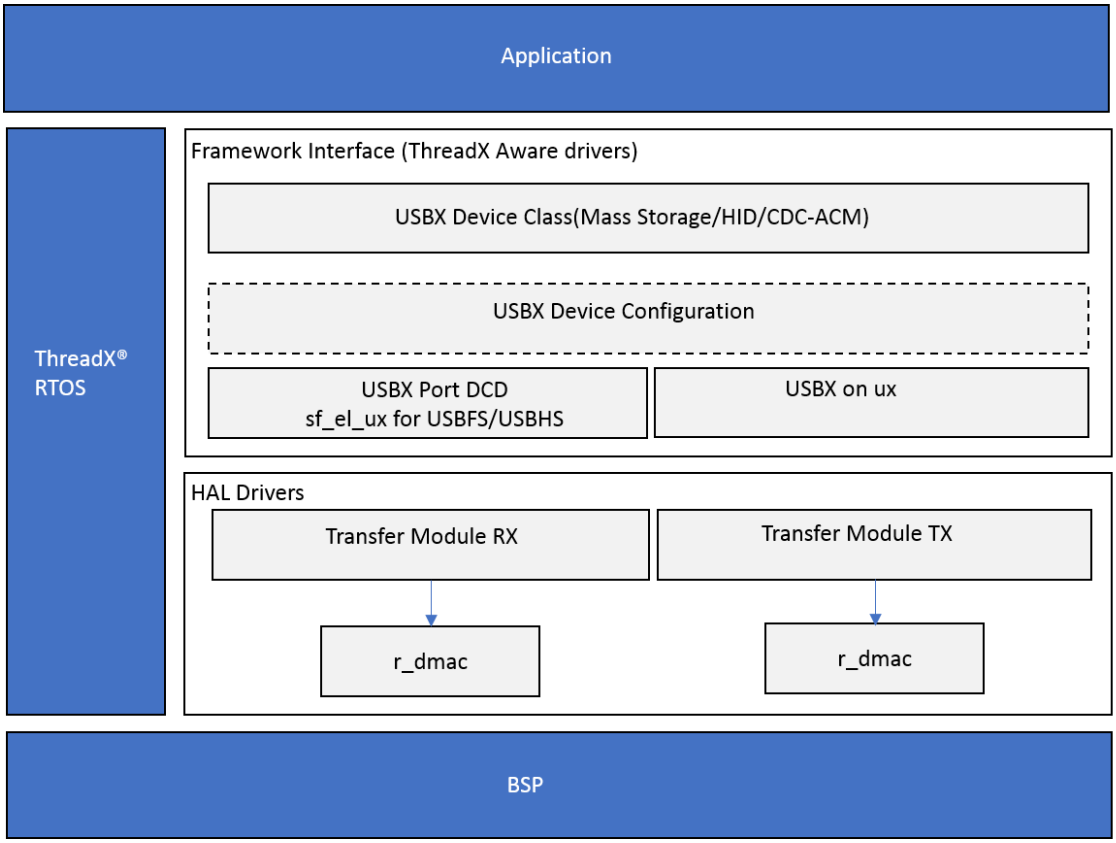


Figure 164: Express Logic USBX Synergy Port Framework Module Organization, Options and Stack Implementations for HCD and DCD

4.1.14.2 Express Logic USBX Synergy Port Framework Module Operational Overview

The Express Logic USBX Synergy Port framework module provides the Synergy USB hardware port functions required to use the USBX stack on Synergy hardware. Application code using this module is expected to use USBX API calls.

Express Logic USBX Synergy Port Framework Module Operational Notes

The Express Logic USBX Synergy Port framework module includes the support for the Express Logic USBX APIs in SSP. Refer to the Express Logic USBX User Manual for a complete description of the available APIs.

The Express Logic USBX Synergy Port framework module supports the Port Device Controller Driver (DCD) on USBHS and USBFS peripherals as well as the Port Host Controller Driver (HCD) for the USBHS on USBFS peripherals.

Users have the option of using the Transfer Module for the USBX Synergy Port framework module to get better USB data throughput by transferring data in the block transfer mode. To enable the Transfer module, just add two instances of transfer components to the USBX Class stack in the Synergy Configuration tool and enable the interrupts in the property. The Synergy Configuration tool auto-generates the driver setup code to enable DMAC or DTC transfer in common_data.c.

NOTE: : The module uses the interrupt of a USB Controller. Set the appropriate interrupt priority level in the Synergy Configuration tool, otherwise it does not work. The module uses the interrupt of a Transfer module (implemented as DMAC or DTC) if it is used. Set the appropriate priority level in the Synergy Configuration tool. The priority level of transfer module must be higher than the priority level for the USB Controller, otherwise it does not work.

Express Logic USBX Synergy Port Framework Module Limitations

- Synergy USB controllers (USBHS and USBFS) have a limited number of PIPEs you can use for the isochronous transfer type (PIPE1 and PIPE2). This will limit the number of UVC devices (two devices) you can connect to.
- The device side driver(sf_el_ux DCD driver) does not support DTC as the transfer interface
- **The isochronous transfer is only supported for USB Host**. The transfer type is not supported for USB Device.
- **The USBFS controller is unlikely to support typical UVC devices.** The maximum packet size of isochronous PIPEs on USBFS controller is limited to 256 bytes. This would impact in the UVC usage.
- **Synergy USB controllers (USBHS and USBFS) do not support high-bandwidth isochronous transfer.** The controllers support 1 transaction per micro-frame if running at High Speed).
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.14.3 Using the Express Logic Synergy Port USBX Framework Module in an Application

Once the Express Logic Synergy Port USBX framework module is added to a thread the Express Logic APIs become available for use.

4.1.15 Console Framework

The Console Framework is a complete API implementation for a menu-driven console command line interface (CLI) using the ThreadX RTOS. The Console Framework module uses a lower-level communications interface which connects to a hardware option for either UART, USB, or Ethernet Telnet connectivity. The Console Framework module has a user-defined menu of commands and various APIs to present a prompt, identify and issue a callback for menu commands, and read, write, and parse input strings.

4.1.15.1 Console Framework Module Features

The console framework supports the following features:

- Creation of a menu-based command-line interface
- Submenus and navigation through multiple menus in a single call
- Menu navigation to go up to the parent menu or back to the root
- A help menu for each menu
- Writing NULL terminated strings and reading until return character is received
- An API to help parse arguments to the command line
- Case-insensitive inputs

The Console Framework module organization, as depicted in the thread stack window in the SSP configurator, is shown in the following figure. Each implementation choice, Ethernet, UART, and USB has its own lower-level modules that are added automatically based on the developer's implementation choice. In most cases, all the needed configuration information is automatically added to the modules leaving the developer with just a few important configuration settings that need to be selected.

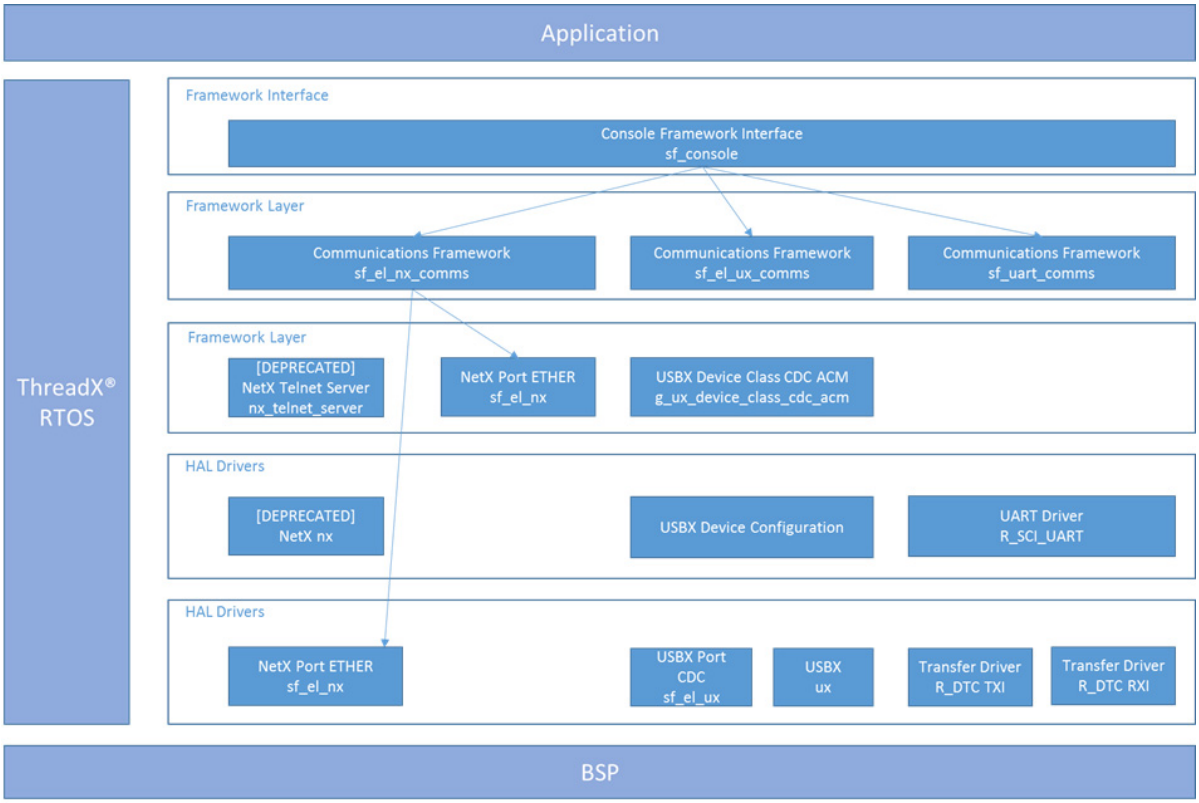


Figure 165: Console Framework Module Block Diagram

4.1.15.2 Console Framework Module APIs Overview

The Console Framework defines APIs for opening, closing, reading, writing, and issuing an input prompt. It also provides some additional functions, such as parse and augumentFind, to assist in processing more complex commands. A complete list of the available APIs, an example API call, a short description of each can be found in the following tables. A table of status return values follows.

Console Framework Module API Summary

Function Name	Example API Call and Definition
open	g_sf_console0.p_api->open(g_sf_console0.p_ctrl, g_sf_console0.p_cfg) The open API configures the console. This function must be called before any other console functions. NOTE: This call is made automatically during system initialization, prior to entering the users thread. Unless the user closes the console, open will not need to be called.

Function Name	Example API Call and Definition
close	g_sf_console0.p_api->close(g_sf_console0.p_ctrl); The close API handles the clean-up of internal driver data.
.prompt	g_sf_console0.p_api->prompt(g_sf_console0.p_ctrl, NULL, TX_WAIT_FOREVER); The prompt API prints the prompt string from the menu, waits for input, parses the input based on the menu, and calls the appropriate callback function if a command is identified.
parse	g_sf_console0.p_api->parse(g_sf_console0.p_ctrl, commands, input, s_length); The parse API looks for an input string in the command menu and, if one is found, calls the appropriate callback function.
read	g_sf_console0.p_api->read(g_sf_console0.p_ctrl, ch, 1, TX_WAIT_FOREVER); The read API puts data into the destination, byte-by-byte and echoes the input to the console. Backspace, delete, and left/right arrow keys are supported. Read completes when a line ending with CR, CR+LF, or CR+NULL is received, or when the input exceeds the number of bytes allowed. If the buffer overflows SF_CONSOLE_MAX_INPUT_LENGTH, the read will return an error code.
write	g_sf_console0.p_api->write(g_sf_console0.p_ctrl, (uint8_t*)data_string, TX_WAIT_FOREVER); The write API gets the buffer mutex object and handles data transmission at the HAL layer. It obtains the event flag to synchronize the completion of a data transfer.
argumentFind	g_sf_console0.p_api->argumentFind("LED", p_args->p_remaining_string, NULL, &led_num); The argumentFind API locates a command line argument in an input string and returns the index of the character immediately following the argument. Any string numbers are converted to integers.
versionGet	g_sf_console0.p_api->versionGet(&version); Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_UNSUPPORTED	Command not found in the current menu.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.15.3 Console Framework Module Operational Overview

The Console Framework module is a ThreadX-aware Command Line Interface (CLI). The module uses ThreadX objects like mutex for blocking and synchronization techniques like event flags for the completion of a transaction. The key operational elements of the Console Framework are initialization and input processing, each of which are described in the following sections.

Console Framework Module Initialization

The open call is automatically generated by the ISDE and is in the src/synergy_gen/toggle_thread.c file where the module was added. The open call requires the application to define a root menu with a variable name that matches the one in the configurator (g_sf_console_root_menu) by default. By the time execution reaches src/ toggle_thread_entry.c, the module is ready to use, provided the necessary hardware connection is established.

Console Framework Module Input Processing

The Console Framework module requires a set of menus, command structures, and callbacks. The Console Framework module typically operates from the prompt, often located within a while loop in the entry thread. The framework prompt API will print the current menu as a prompt, then read input and echo it back to the console (unless echo is disabled in the properties.)

Following operations are performed against the user input:

While the console is accepting input,

- Backspace will remove characters before the cursor.
- Delete will remove characters after the cursor.
- The left and right arrow keys move the cursor.
- The up-arrow key will fill in the last command only when nothing else has been entered.

NOTE: There is no history beyond the last command; if the up-arrow key is pressed twice, the console does not know what command was entered prior to the last command and it will continue to display the last command.

When the console sees a return character on the read input, it parses the input string and calls the associated callback or switches to the next menu if SF_CONSOLE_CALLBACK_NEXT_FUNCTION is used in place of the callback for the command. The console will continue parsing until a callback function is called. If prompt API is called again, it will prompt using the menu that contains the callback function. To navigate up to the parent menu, enter '^'. To navigate to the root menu from any submenu, enter '~'.

Creating Console Framework Module Required Structures – The Menu

The Console Framework requires a menu and it is up to the developer to create the structure that is used by the Console Framework to implement the menu. The Console Menu structure (depicted in the following figure) includes a pointer to the previous menu, (for creating multi-level menus) a name for the menu, the number of commands in the menu, and a pointer to an array of command structures. As seen in the following figure, each entry in the array of commands includes pointers to the command name string, the help command description string, the associated command callback function, and a context parameter provided to the callback.

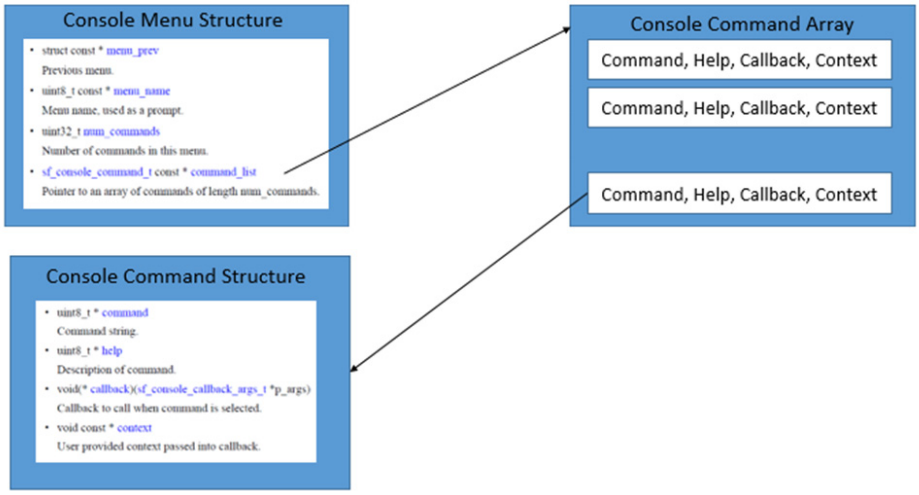


Figure 166: Console Framework Module Menu Structures

The application project example illustrates a Console Framework with a single menu; it controls the toggle of an LED from the CLI. The console command array (g_sf_console_commands, seen on the right in the following figure) stores the array of commands (in this case just a single command.) The command structure defines the command as "LED TOGGLE", the help description as "Toggle an LED", the callback as led_toggle_callback, and the context as NULL, since it is unused for this example.

The root menu, seen on the left side of the following figure, is identified by the g_sf_console_root_menu structure. The structure defines the menu_prev entry as NULL, since there is only the single menu, the menu_name as "Root", the num_commands as the size of the array divided by the size of an entry (to determine the total number of entries) as "1", and the command_list starting address as "address", the location of the first entry in the command array.

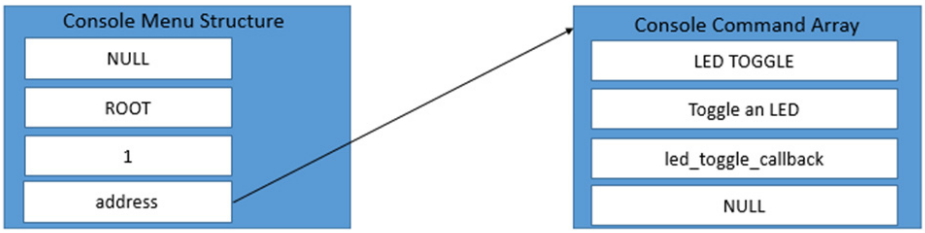


Figure 167: Menu Structure Example Diagram

Console Framework Module Important Operational Notes and Limitations

To use the Console Framework module prompt API, first set up the menu, command structures, and callbacks.

There are no known limitations for using this module.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.15.4 Including the Console Framework Module in an Application

This section describes how to include the Console Framework module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Console Framework module to your application, simply add it to a project thread using the stacks selection sequence provided in the following table. (The default name for the console framework module is g_sf_console0. This name can be changed in the associated Properties window.)

Console Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_console0 Console Framework on sf_console	Threads	New Stack > Framework > Services > Console Framework on sf_console

When the console framework on sf_console is added to the thread stack as shown in the following figure, the configurator reports (via the Add Communications Framework block) that at least one Communications Framework is required to complete the Console Framework. The Communications Framework determines the type of communications interface (and the underlying hardware implementation) the Console Framework will use. Any low-level modules that need additional configuration information will have box text highlighted in red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended (this is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description will include "Add" in the text. Clicking on any Pink banded modules will bring up the "New" icon and then display the possible choices.

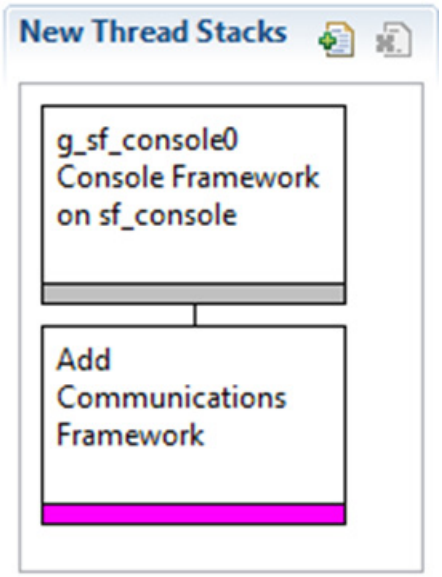


Figure 168: Console Framework Module Stack

Currently there are three possible Communications Frameworks available that the Console Framework can select: UART, USB, or Telnet. Configurations for each are shown in the following thread stack illustrations and they can be easily imported by clicking the Add Communications Framework block and selecting the desired Communications Framework. (Note that the Telnet option uses a deprecated implementation. This will function correctly, but is indicated as deprecated since it will be replaced in a future release. Designs using the deprecated module will need to be updated to the new implementation after that release.) Other Communications Frameworks are being added and may show up when available.

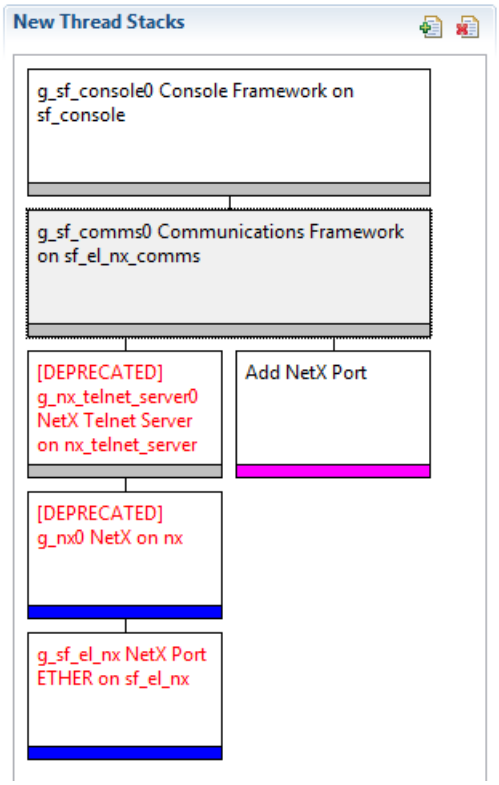


Figure 169: Ethernet

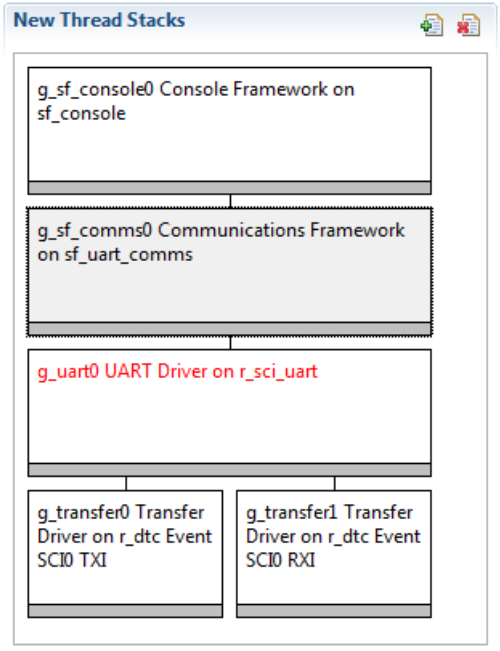


Figure 170: UART

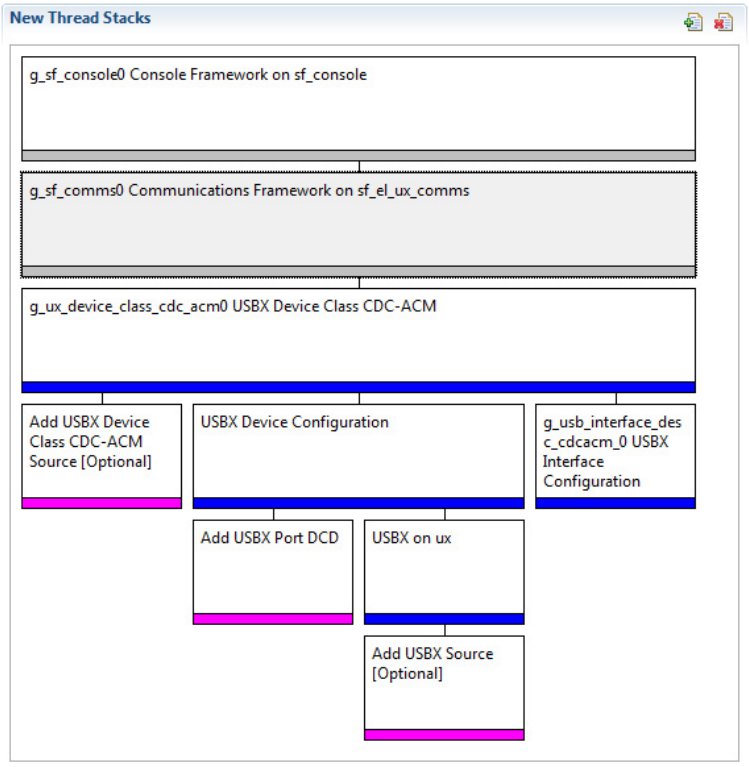


Figure 171: USB

Communications Framework Module Options

4.1.15.5 Configuring the Console Framework Module

The Console Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for-lower level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+.) This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the configuration table settings in the following table. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Console Framework Module on sf_console

Parameter	Value	Description
Parameter Checking	BSP, Enable, Disable Default: BSP	Selects if code for parameter checking is to be included in the build.
Maximum Input String Length	128	Input string length.
Maximum Write String Length	128	Output string length.
Name	g_sf_console0	Console Framework module name.
Name of Initial Menu (Application Defined)	g_sf_console_root_menu	Name of starting menu.
Echo	True, False Default: True	Enable or disable echo to terminal from the prompt.
Autostart	True, False Default: False	If True, the prompt will occur using the top-level menu after initialization. If False, the prompt needs to be called in the application.
Name of the sf_console Initialization Function	sf_console_init0	Name of the sf_console initialization function selection.
Auto sf_console Initialization	Enable, Disable Default: Enable	Auto sf_console initialization selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, the IP address for the Telnet connection or the baud rate of the UART might need to be modified. The configurable properties for the lower-level stack modules are displayed for completeness and as a reference.

NOTE: Most of the property settings for modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the Console Framework Stack Modules

Typically, only a small number of settings must be modified from the default for lower-level modules as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following tables identify all the settings within the properties section for the module.

When the Telnet option is selected for the low-level implementation, the following modules are automatically added to the stack. The following tables show the detailed configuration settings available, many of which are populated with default settings that can be used in most common applications.

Configuration Settings for the Telnet Option (sf_el_nx_comms)

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled	Enable or disable the parameter checking.
	Default: BSP	
Name	g_sf_comms0	Module name.
Channel	0	Underlying channel used by Ethernet driver.
IP Address Byte 1	192	IP Address Byte 1 selection
IP Address Byte 2	168	IP Address Byte 2 selection
IP Address Byte 3	0	IP Address Byte 3 selection
IP Address Byte 4	0	IP Address Byte 4 selection
Subnet Mask Byte 1	255	Subnet Mask Byte 1 selection
Subnet Mask Byte 2	255	Subnet Mask Byte 2 selection
Subnet Mask Byte 3	255	Subnet Mask Byte 3 selection
Subnet Mask Byte 4	0	Subnet Mask Byte 4 selection

NOTE: Information and a description of valid and common settings for IP Addresses and the associated masks are available in the IP Address Limitations Synergy Platform Knowledge Base article available as described at the end of this document. The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for stack modules can be desirable. For example, it might be useful to have a different IP address. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

NOTE: The following deprecated configurations of the NetX Telnet server module and the NetX module are required when using the Communications Framework module on sf_el_nx_comms. The Communications Framework module on sf_el_nx_comms will be updated in a future release to avoid using deprecated configurations.

Configuration Settings for the NetX Telnet Server Module (nx_telnet_server)

ISDE Property	Value	Description
Name	g_nx_telnet_server0	Module name
Show deprecation warning	Enabled, Disabled Default: Enabled	Show deprecation warning selection

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Module (nx)

ISDE Property	Value	Description
Name	Default: g_nx0	NetX module name.
Show deprecation warning	Enabled, Disabled Default: Enabled	Show deprecation warning selection

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for NetX Port ETHER (sf_el_nx)

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking
Channel 0 Phy Reset Pin	IOPORT_PORT_09_PIN_03	Channel 0 Phy reset pin selection
Channel 0 MAC Address High Bits	0x00002E09	Channel 0 MAC address high bits selection
Channel 0 MAC Address Low Bits	0x0A0076C7	Channel 0 MAC address low bits selection

ISDE Property	Value	Description
Channel 1 Phy Reset Pin	IOPORT_PORT_07_PIN_06	Channel 1 Phy reset pin selection
Channel 1 MAC Address High Bits	0x00002E09	Channel 1 MAC address high bits selection
Channel 1 MAC Address Low Bits	0x0A0076C8	Channel 1 MAC address low bits selection
Number of Receive Buffer Descriptors	8	Number of receive buffer descriptors selection
Number of Transmit Buffer Descriptors	32	Number of transmit buffer descriptors selection
Ethernet Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Ethernet interrupt priority selection
Name	g_sf_el_nx	Module name
Channel	0	Channel selection
Callback	NULL	Callback selection

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

When the USB option is selected for low-level implementation, the following modules are automatically added to the stack. The following tables show the detailed configuration settings available, many of which are populated with default settings that can be used in most common applications.

Configuration Settings for USB Communications Framework Module (sf_el_ux_comms)

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Read Input Buffer Size (Bytes)	128	Maximum number of bytes that can be received at a time in the read() API.
Timeout in ticks	1000	Timeout value to suspend a USBX CDC instance creation in the open() API.
Name	g_sf_comms0	Module name.
Name of the sf_comms initilization function	sf_comms_init0	Name of helper function to initialize Communications Framework. The function will be presented in the auto-generated code in the <xxx_thread>.c, where <xxx_thread> is the name of your thread symbol given to the Thread property. The function is to be called in the auto-generated code if Auto sf_comms Initialization property is Enabled. If Disabled, the function can be called in the user application.
Auto sf-comms Intialization	Enable, Disable Default: Enable	Auto Initialization support of Communications Framework. The helper function above will be called in the auto-generated code if this configuration is enabled. Else, the function will not be called automatically and user can call it sometime later.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration for the USBX Device Class CDC ACM Module (g_ux_device_class_cdc_acm0)

ISDE Property	Value	Description
Name	g_ux_device_class_cdc_acm0	Module name
USBX CDC-ACM instance_activate Function Callback	ux_cdc_device0_instance_activate	USBX CDC-ACM instance_activate Function Callback selection
USBX CDC-ACM instance_deactivate Function Callback	ux-cdc_device0_instance_deactivate	USBX CDC-ACM instance_deactivate Function Callback selection

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Device Configuration

ISDE Property	Value	Description
Vendor ID	0x045B	Vendor ID selection
Product ID	0x0000	Product ID selection
Device Release Number	0x0000	Device Release Number selection
Index of Manufacturing String Descriptor	0x00	Index of Manufacturing String Descriptor selection
Index of Product String Descriptor	0x00	Index of Product String Descriptor selection
Index of Serial Number String Descriptor	0x00	Index of Serial Number String Descriptor selection
Class Code	Device, Communications(CDC), HID, Mass Storage, Miscellaneous, Vendor Specific Default: Communications	Class Code selection
Index of String Descriptor describing this configuration	0x00	Index of String Descriptor describing this configuration selection

ISDE Property	Value	Description
Size of USB Descriptor in bytes for this configuration (Modify this value only for Vendor-specific Class, otherwise set zero)	0x00	Size of USB Descriptor in bytes for this configuration (Modify this value only for Vendor-specific Class, otherwise set zero) selection
Number of Interfaces (Modify this value only for Vendor-specific Class, otherwise set zero)	0x00	Number of Interfaces (Modify this value only for Vendor-specific Class, otherwise set zero) selection
Self-Powered	Enable, Disable Default: Enable	Self-Powered selection
Remote Wakeup	Enable, Disable Default: Disable	Remote Wakeup selection
Maximum Power Consumption (in 2mA units)	50	Maximum Power Consumption (in 2mA units) selection
Supported Language Code	0x0409	Supported Language Code selection
Name of USBX String Framework	NULL	Name of USBX String Framework selection
Total index number of USB String Descriptors in USB String Framework	0	Total index number of USB String Descriptors in USB String Framework selection
Name of USBX Language Framework	NULL	Name of USBX Language Framework selection
Number of Languages to support (US English is applied if zero is set)	0	Number of Languages to support (US English is applied if zero is set) selection

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Interface Configuration

ISDE Property	Value	Description
Name	g_usb_interface_desc_cdcacm_0	Module name
Interface Number of Communications Class interface	0x00	Interface Number of Communications Class interface selection
Interrupt Transfer endpoint to use for Communications Class	Endpoint 1-9 Default: Endpoint 3	Interrupt Transfer endpoint to use for Communications Class selection
Polling period for Interrupt Endpoint (in mS/125us units for FS/HS)	0x0F	Polling period for Interrupt Endpoint (in mS/125us units for FS/HS) selection
Interface Number of Data Class interface	0x01	Interface Number of Data Class interface selection
Bulk In Transfer endpoint to use for Data Class	Endpoint 1-9 Default: Endpoint 1	Bulk In Transfer endpoint to use for Data Class selection
Bulk Out Transfer endpoint to use for Data Class	Endpoint 1-9 Default: Endpoint 2	Bulk Out Transfer endpoint to use for Data Class selection
Index of String Descriptor Describing Communications Class interface (Interface Descriptor: Interface)	0x00	Index of String Descriptor Describing Communications Class interface (Interface Descriptor: Interface) selection
Index of String Descriptor Describing Data Class interface (Interface Descriptor: Interface)	0x00	Index of String Descriptor Describing Data Class interface (Interface Descriptor: Interface) selection

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port DCD on sf_el_ux for USBFS

ISDE Property	Value	Description
Full Speed Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Full speed interrupt priority selection.
Name	g_sf_el_ux_dcd_fs_0	Module name.
USB Controller Selection	USBFS	USB controller selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Port DCD on sf_el_ux for USBHS

ISDE Property	Value	Description
High Speed Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	High speed interrupt priority selection.
Name	g_sf_el_ux_dcd_hs_0	Module name.
USB Controller Selection	USBHS	USB controller selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX on ux

ISDE Property	Value	Description
USBX Pool Memory Name	g_ux_pool_memory	USBX pool memory name selection.
USBX Pool Memory Size	18432	USBX pool memory size selection.
User Callback for Host Event Notification (Only valid for USB Host)	NULL	User callback for host event notification (only valid for USB host)

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

When the UART option is selected for low-level implementation, the following modules are automatically added to the stack. The following tables show the detailed configuration settings available, many of which are populated with default settings that can be used in most common applications.

Configuration Settings for the UART Communications Framework Module (sf_uart_comms)

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled	Selects if parameter checking is included.
	Default: BSP	
Read Input Queue Size (4-Byte Words)	15	Buffer size for data reception queue. sf_uart_comms utilizes the ThreadX Queue for the queue management.
Name	g_sf_comms0	Name of UART communications framework module.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the UART Driver Module (r_sci_uart)

ISDE Property	Value	Description
External RTS Operation	Enable, Disable Default: Disable	Enable an IOPORT pin to be used as RTS signal. For RTS functionality set this configuration parameter to "Enable" and specify the configuration "Name of UART callback function for the RTS external pin control".
Reception	Enable, Disable Default: Enable	Enable or disable UART reception for all UART channels on SCI. Setting this configuration parameter to "Disable" reduces code size because the portion of code for UART reception is not compiled. You cannot set this parameter for individual UART channels.
Transmission	Enable, Disable Default: Enable	Enable or disable UART transmission for all UART channels on SCI. Setting "Disable" to this configuration allows to get smaller code size due to the portion of code for UART transmission is compiled out, however, you can only set "Disable" to this configuration if any other SCI channels which work as UART ports do not perform the transmission.
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_uart0	The name to be used for UART on SCI module control block instance. This name is also used as the prefix of the other variable instances.
Channel	0-9	SCI channel number.
Baud Rate	9600	Baud rate selection.

ISDE Property	Value	Description
Data Bits	7 bits, 8, bits, 9 bits Default: 8 bits	UART data bits.
Parity	None, Odd, Even Default: None	UART parity bits.
Stop Bits	1 bit, 2 bits Default: 1 bit	UART stop bits.
CTS/RTS Selection	CTS (Note that RTS is available when enabling External RTS Operation mode which uses 1 GPIO pin), RTS (CTS is disabled) Default: RTS (CTS is disabled)	Select CTS or RTS for the CTSn/RTSn pin of SCI channel n. The SCI hardware supports either the CTS or the RTS control signal on this pin but not both. For an application that uses both CTS and RTS, select "CTS" for this configuration parameter and enable the configuration "External RTS Operation" specifying the configuration "Name of UART callback function for the RTS external pin control".
Name of UART callback function to be defined by user	user_uart_callback	Name must be a valid C symbol.
Name of UART callback function for the RTS external pin control to be defined by user	NULL	Name must be a valid C symbol.
Clock Source	Internal Clock, External Clock 8x baudrate, External Clock 16x baudrate Default: Internal Clock	Selection of the clock source to be used in the baud-rate clock generator block.

ISDE Property	Value	Description
Baudrate Clock Output from SCK pin	Enable, Disable Default: Disable	Optional setting to output the baud-rate clock on the SCKn pin for the selected channel n.
Start bit detection	Falling Edge, Low Level Default: Falling Edge	Start bit detection mode in the reception, usually set "Falling Edge" to this configuration.
Noise Cancel	Enable, Disable Default: Disable	Enable the digital noise cancellation on RXDn pin. The digital noise filter block in SCI consists of two-stage flip-flop circuits. For detail, refer to the Noise cancellation section in the Renesas Synergy hardware manual.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bit rate modulation enable selection.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Transmit interrupt priority selecition.

ISDE Property	Value	Description
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Error interrupt priority selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Transfer Driver Module on r_dtc Event SCI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection
Name	g_transfer0	Module name
Mode	Normal	Mode selection

ISDE Property	Value	Description
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 TXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Transfer Driver Module on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SPI0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection

ISDE Property	Value	Description
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

When the USB option is selected for low-level implementation, the following modules are automatically added to the stack. The following tables show the detailed configuration settings available, many of which are populated with default settings that can be used in most common applications.

Configuration Settings for USB Communications Framework Module (sf_el_ux_comms)

ISDE Property	Value	Description
Parameter Checking	BSP, Enable, Disable Default: BSP	Selects if code for parameter checking is to be included in the build.
Memory Size (Bytes)	65536	Memory size selection.
Timeout in ticks	1000	Timeout value to suspend a USBX CDC instance creation in the open() API.
Read Input Buffer Size (Bytes)	128	This is the maximum number of bytes that can be received at a time in the read() API.
Name	g_sf_comms0	Name for USB communications framework module.
Name of the sf_comms Initialization Function	sf_comms_init0	Name of the sf_comms initialization function selection.
Auto sf_comms Initialization	Enable, Disable Default: Enable	Auto sf_comms initialization selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration for the USBX Device Class CDC ACM Module (g_ux_device_class_cdc_acm0)

ISDE Property	Value	Description
Name	Default: g_ux_device_class_cdc_acm0	USBX device class cdc-acm module name.
Show Deprecation Warning	Enabled, Disabled Default: Enabled	Show deprecation warning selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USXB on ux

ISDE Property	Value	Description
Name	Default: g_ux0	USBX on ux module name.
Show Deprecation Warning	Enabled, Disabled Default: Enabled	Show deprecation warning selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the USBX Ports HS and FS on sf_el_ux

ISDE Property	Value	Description
VBUSEN Pin Signal Logic	Active Low, Active High Default: Active High	VBUSEN pin signal logic selection.
High Speed Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	High speed interrupt priority selection.

ISDE Property	Value	Description
Full Speed Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Full speed interrupt priority selection.
Name	Default: g_sf_el_ux	Name for the USBX Ports HS and FS.
Show deprecation warning	Enabled, Disabled Default: Enabled	Show deprecation warning selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

4.1.15.6 Using the Console Framework Module in an Application

The key elements in constructing a simple Console Framework are selecting and configuring the stack, defining the menu structure, writing the code for the menu command callbacks, and using the API calls to implement the desired CLI functions within the target application. The typical steps in using the Console Framework module in an application are:

1) Create menu and command structures.
2) Implement needed callbacks.
3) Initialize the SF_CONSOLE using open API.
4) Use the prompt API to generate the prompt and process commands.
5) Use other APIs (read, write, parse or argumentFind) as needed to process commands.
6) Use close API to close the module if desired.

These common steps are illustrated in a typical operational flow diagram in the following figure:

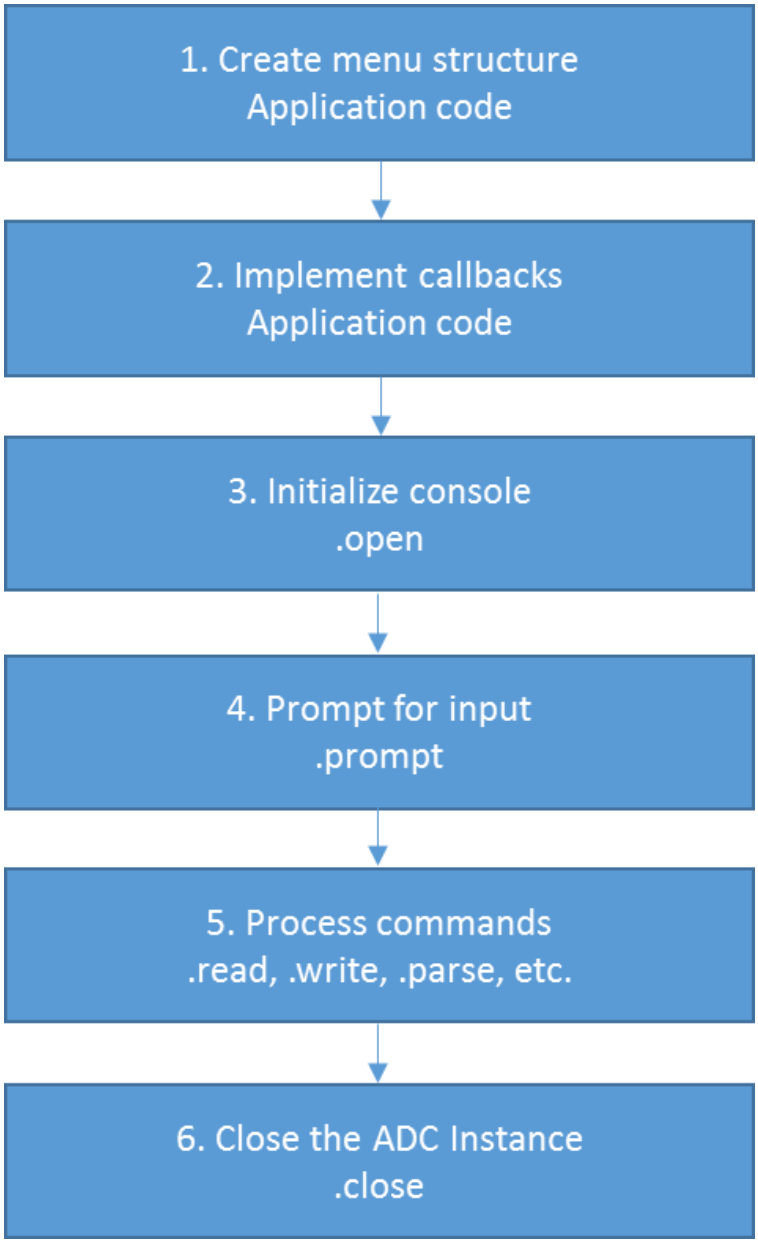


Figure 172: Typical Console Framework Module Application

4.1.16 Crypto Framework

The Crypto Framework layer is composed of multiple Crypto modules providing varied cryptographic services. It includes:

- SF_CRYPTO for resource synchronization between the crypto modules.

- SF_CRYPTO_TRNG for true random number generation.
- SF_CRYPTO_HASH for message digest generation. Provides support for MD5, SHA1, SHA 224, SHA 256 algorithms.
- SF_CRYPTO_KEY for Key Generation services. Provides support for AES, ECC and RSA keys.
- SF_CRYPTO_CIPHER for encryption and decryption services. Provides support for AES and RSA algorithms.
- SF_CRYPTO_SIGNATURE for RSA signature generation and verification services.
- SF_CRYPTO_KEY_INSTALLATION for key installation services. Provides support for AES, ECC and RSA keys.

Refer to SCE HAL module documentation for the cryptographic functions that are supported on different target MCUs.

4.1.16.1 Crypto Framework Module Features

- SF CRYPTO Framework
  - The SF CRYPTO Framework opens the underlying Secure Crypto Engine.
  - Provides services for access to shared resources for other Crypto Framework modules SF_CRYPTO_TRNG, SF_CRYPTO_HASH , SF_CRYPTO_KEY, SF_CRYPTO_CIPHER, SF_CRYPTO_SIGNATURE, and SF_CRYPTO_KEY_INSTALLATION.
- SF CRYPTO TRNG Framework
  - The SF CRYPTO TRNG Framework module uses the TRNG HAL interfaces to the underlying Secure Crypto Engine.
  - Access to shared resources through SF CRYPTO Framework module.
- SF CRYPTO HASH Framework
  - The SF CRYPTO HASH Framework utilizes the underlying Secure Crypto Engine to provide HASH services.
  - This module provides enhancements over the HAL driver such as:
  - Initializing the HASH value
  - **Processing chunks of data through hashUpdate API before finalizing the hash operation.**
  - **Formatting the final block for the final HASH operation.**
- SF CRYPTO KEY Framework Module Features
  - RSA 2048-bit, 1024-bit plain text/raw keys.
  - RSA 2048-bit, 1024-bit standard format wrapped private keys (public keys in plain).
  - AES 128-bit, 192-bit and 256-bit wrapped keys for ECB, CBC, CTR and GCM chaining modes.
  - AES 128-bit and 256-bit wrapped keys for XTS chaining mode.
  - ECC 192-bit, 256-bit wrapped keys.
- SF CRYPTO SIGNATURE Framework Module Features

  - This module currently supports signature generation and signature-verification for RSA algorithm. There is support for both 1024-bits and 2048-bits key lengths for standard format plain-text, standard format wrapped private key and CRT plain-text keys.
  - RSASSA-PKCS1 v1.5 is the supported signature scheme. The input message can be passed as raw message to be signed/verified or can be PKCS1 v1.5 encoded and padded before sign/verify.
  - SHA1, SHA224, SHA256 are the supported hashing algorithms for PKCS1 v1.5 encoding /padding schemes.
  - This module allows signature generation/verification on data which is smaller than a block size.
  - If all of the data is not available at once, update APIs can be used to accumulate the incoming chunks of data and finally sign/verify the message only when all the message is gathered.
  - Allows switching between sign and verify operations with less expensive API calls which are not involved in allocating and deallocating memory.
- SF CRYPTO CIPHER Framework Module Features
  - This module currently supports encryption and decryption for AES and RSA algorithms.
  - This module allows encryption/ decryption of data when available in chunks through the cipherUpdate()API and final() when the last chunk /all the data is gathered.
  - Once the module is opened for a specific key type and key size, the encrypt and decrypt modes can be switched by calling the cipherInit() API.
- SF CRYPTO KEY INSTALLATION Framework Module Features
  - The SF CRYPTO KEY INSTALLATION Framework module uses the KEY INSTALLATION HAL interfaces to the underlying Secure Crypto Engine.
  - Access to shared resources through SF CRYPTO Framework module.
  - This module supports key installation for the following keys:
  - **RSA:**
  - **1024-bit and 2048-bit plain text keys.**
  - **AES:**
  - **128-bit, 192-bit and 256-bit plain text keys for ECB, CBC, CTR and GCM chaining modes.**
  - **128-bit and 256-bit plain text keys for XTS chaining mode.**
  - **ECC:**
  - **192-bit and 256-bit plain text keys.**
  - The user and install keys must be provided to the key installation API in the specified format.
  - Upon installation the key installation service returns a wrapped key to the caller for any future usage of installed key on that device.

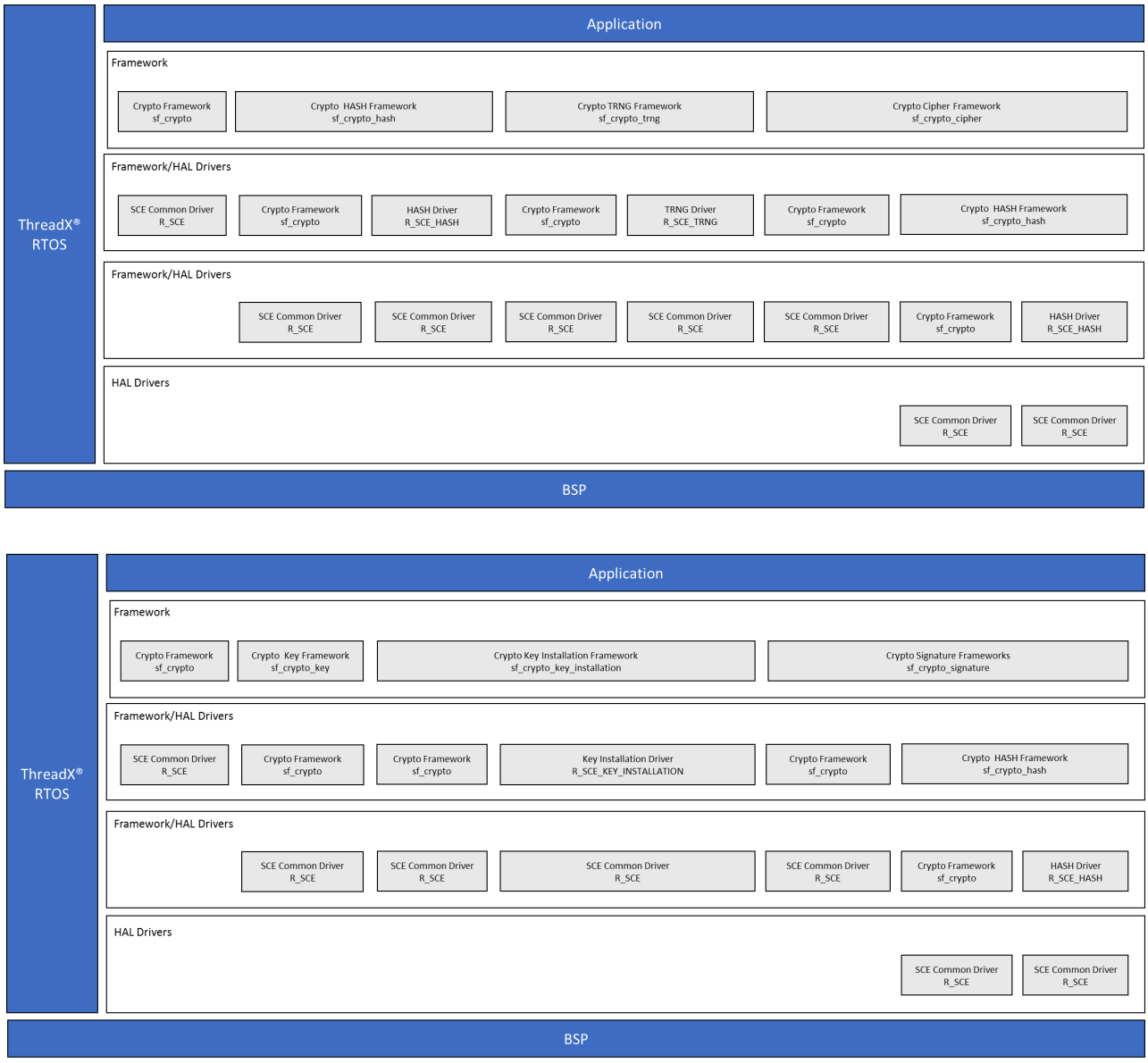


Figure 173: Crypto Framework Module Organization, Options and Stack Implementations

4.1.16.2 Crypto Framework Module APIs Overview

The Crypto Framework Module defines APIs for varied cryptographic services. A complete list of the available APIs, an example API call, and a short description of each can be found in the table below. A table of status return values follows.

Crypto Framework Module API Summary

Function Name	Example API Call and Description
.open	g_sf_crypto0.p_api->open(g_sf_crypto0.p_ctrl, g_sf_crypto0.p_cfg); This function is used to open the crypto framework module and the r_sce HAL module.
.close	g_sf_crypto0.p_api->close(g_sf_crypto0.p_ctrl); This function is used to close the crypto framework module and the r_sce HAL module.
.lock	g_sf_crypto0.p_api->lock(g_sf_crypto0.p_ctrl); This function locks shared resources for Cryptography operations.
.unlock	g_sf_crypto0.p_api->unlock(g_sf_crypto0.p_ctrl); Unlock shared resources for Cryptography operations. .
.getStatus	g_sf_crypto0.p_api->getStatus(g_sf_crypto0.p_ctrl, &p_status); This function gets the status of the Crypto Framework Common Module.
.versionGet	g_sf_crypto0.p_api->versionGet(&version); This function retrieves the API version using the version pointer.

Crypto Framework HASH Module API Summary

Function Name	Example API Call and Description
.open	g_sf_crypto_hash0.p_api->open(g_sf_crypto_hash0.p_ctrl, g_sf_crypto_hash0.p_cfg); This function is used to open the crypto hash framework module.
.close	g_sf_crypto_hash0.p_api->close(g_sf_crypto_hash0.p_ctrl); This function is used to close the crypto hash framework module.
.hashInit	g_sf_crypto_hash0.p_api->hashInit(g_sf_crypto_hash0.p_ctrl); This function initializes the calculation of the hash function. It has to be invoked first and only once, at the beginning of the each new calculation.
.hashUpdate	g_sf_crypto_hash0.p_api->hashUpdate(g_sf_crypto_hash0.p_ctrl, &p_data_in); This function calculates the hash on the data pointed to by the p_data_in pointer.
.hashFinal	g_sf_crypto_hash0.p_api->hashUpdate(g_sf_crypto_hash0.p_ctrl, &p_data_in, &p_size); This function finalizes the hash operation on the data pointed to by the p_data_in pointer and stores the output size in the 32-bit word defined by the p_size pointer.
.versionGet	g_sf_crypto_hash0.p_api->versionGet(&version); This function retrieves the API version using the version pointer.

Crypto Framework TRNG Module API Summary

Function Name	Example API Call and Description
.open	g_sf_crypto_trng0.p_api->open(g_sf_crypto_trng0.p_ctrl, g_sf_crypto_trng0.p_cfg); This function is used to open the crypto TRNG framework module.
.close	g_sf_crypto_trng0.p_api->close(g_sf_crypto_trng0.p_ctrl); This function is used to close the crypto TRNG framework module.
. randomNumberGenerate	g_sf_crypto_trng0.p_api-> randomNumberGenerate (&p_buffer); This function generates the random number and stores it in memory starting at the address defined by the p_buffer pointer.
.versionGet	g_sf_crypto_trng0.p_api->versionGet(&version); This function retrieves the API version using the version pointer.

Crypto Framework KEY Module API Summary

Function Name	Example API Call and Description
.open	g_sf_crypto_key0.p_api->open(g_sf_crypto_key0.p_ctrl, g_sf_crypto_key0.p_cfg); This function is used to open the crypto key framework module.
.keyGenerate	g_sf_crypto_key0.p_api->keyGenerate(g_sf_crypto_key0.p_ctrl, &p_secret_key, &p_public_key); This function is used to generate a key or key pair and store it at the p_secret_key and p_public_key pointers.

Function Name	Example API Call and Description
.close	g_sf_crypto_key0.p_api->close(g_sf_crypto_key0.p_ctrl); This function is used to close the crypto key framework module and the r_sce HAL module.
.versionGet	g_sf_crypto_key0.p_api->versionGet(&version); This function retrieves the API version using the version pointer.

Crypto Framework KEY Installation Module API Summary

Function Name	Example API Call and Description
.open	g_sf_crypto_key_installation.p_api->open(g_sf_crypto_key_installation.p_ctrl, g_sf_crypto_key_installation.p_cfg); This function is used to open the crypto key initialization framework module.
.keyInstall	g_sf_crypto_key_installation.p_api->keyInstall(g_sf_crypto_key_installation.p_ctrl, p_user_key_input, p_install_key_input, p_key_data_out); This function is used to install a key using the p_user_key_input, p_install_key_input and p_key_data_out pointers.
.close	g_sf_crypto_key_installation.p_api->close(g_sf_crypto_key_installation.p_ctrl); This function is used to close the crypto key installation framework module.
.versionGet	g_sf_crypto_key_installation.p_api->versionGet(&version); This function retrieves the API version using the version pointer.

Crypto Framework Cipher Module API Summary

Function Name	Example API Call and Description
.open	g_sf_crypto_cipher.p_api->open(g_sf_crypto_cipher.p_ctrl, g_sf_crypto_cipher.p_cfg); This function is used to open the crypto cipher framework module.
.cipherInit	g_sf_crypto_cipher.p_api->cipherInit(g_sf_crypto_cipher.p_ctrl, mode, p_key, p_params); This function is used to initialize cipher operation. Must be called after open or cipherFinal.
.cipherUpdate	g_sf_crypto_cipher.p_api->cipherUpdate(g_sf_crypto_cipher.p_ctrl, p_data_in, p_data_out); This function is used to encrypt or decrypt input data. Can be called multiple times for additional blocks of data.
.cipherFinal	g_sf_crypto_cipher.p_api->cipherFinal(g_sf_crypto_cipher.p_ctrl, p_data_in, p_data_out); This function is used to encrypt or decrypt all/last block of data.
.cipherAadUpdate	g_sf_crypto_cipher.p_api->cipherAadUpdate(g_sf_crypto_cipher.p_ctrl, p_data_in, p_data_out); This function is used to update AAD (Additional Authentication Data) for AES GCM operation.
.close	g_sf_crypto_cipher.p_api->close(g_sf_crypto_cipher.p_ctrl); This function is used to close the crypto cipher framework module.

Function Name	Example API Call and Description
.versionGet	g_sf_crypto_cipher.p_api->versionGet(&version); This function retrieves the API version using the version pointer.

Crypto Framework Signature Module API Summary

Function Name	Example API Call and Description
.open	g_sf_crypto_signature.p_api->open(g_sf_crypto_signature.p_ctrl, g_sf_crypto_signature.p_cfg); This function is used to open the crypto signature framework module.
.contextInit	g_sf_crypto_signature.p_api->contextInit(g_sf_crypto_signature.p_ctrl, mode, p_params, p_key); This function is used to initialize the crypto signature framework module.
.signUpdate	g_sf_crypto_signature.p_api->signUpdate(g_sf_crypto_signature.p_ctrl, p_message); This function performs the signature update operation. It can be called multiple times to accumulate the message to be signed.
.verifyUpdate	g_sf_crypto_signature.p_api->verifyUpdate(g_sf_crypto_signature.p_ctrl, p_message); This function performs the signature verification update operation. It can be called multiple times to accumulate the message whose signature is to be verified.
.signFinal	g_sf_crypto_signature.p_api->signFinal(g_sf_crypto_signature.p_ctrl, p_message, p_dest); This function generates the signature and writes it to the destination.

Function Name	Example API Call and Description
.verifyFinal	g_sf_crypto_signature.p_api->verifyFinal(g_sf_crypto_signature.p_ctrl, p_signature, p_message); This function performs the signature verify operation.
.close	g_sf_crypto_signature.p_api->close(g_sf_crypto_signature.p_ctrl); This function is used to close the crypto signature framework module.
.versionGet	g_sf_crypto_signature.p_api->versionGet(&version); This function retrieves the API version using the version pointer.

NOTE: For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the *SSP User's Manual* API References for the associated module*._

The Framework APIs return common SSP error codes and in addition may return the following error codes from low level modules. There are also Crypto Framework specific error codes and these are listed at the end of the table.

Status Return Values

Name	Description
SSP_ERR_CRYPTO_CONTINUE	Continue executing function
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Hardware resource busy
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Invalid index
SSP_ERR_CRYPTO_SCE_RETRY	Retry
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Verify failed
SSP_ERR_CRYPTO_SCE_ALREADY_OPEN	Crypto Module is already opened
SSP_ERR_CRYPTO_NOT_OPEN	Hardware module is not initialized
SSP_ERR_CRYPTO_UNKNOWN	Some unknown error occurred

Name	Description
SSP_ERR_CRYPTO_NULL_POINTER	Null pointer input as a parameter
SSP_ERR_CRYPTO_NOT_IMPLEMENTED	Algorithm/size not implemented
SSP_ERR_CRYPTO_RNG_INVALID_PARAM	An invalid parameter is specified
SSP_ERR_CRYPTO_RNG_FATAL_ERROR	A fatal error occurred
SSP_ERR_CRYPTO_INVALID_SIZE	Size specified is invalid
SSP_ERR_CRYPTO_INVALID_STATE	Function used in an valid state
SSP_ERR_CRYPTO_ALREADY_OPEN	Control block is already opened
SSP_ERR_CRYPTO_INSTALL_KEY_FAILED	Specified input key is invalid
SSP_ERR_CRYPTO_AUTHENTICATION_FAILED	Authentication failed
Crypto Framework Specific Error Codes	
SSP_ERR_CRYPTO_COMMON_NOT OPENED	Crypto Framework Common is not opened.
SSP_ERR_CRYPTO_HAL_ERROR	Crypto HAL module returned an error.
SSP_ERR_CRYPTO_KEY_BUF_NOT_ENOUGH	Key buffer size isn't large enough to generate a key
SSP_ERR_CRYPTO_BUF_OVERFLOW	Attempt to write data larger than buffer.
SSP_ERR_CRYPTO_INVALID_OPERATION_MODE	Invalid operation.
SSP_ERR_MESSAGE_TO_LONG	Message for RSA encryption is too long.
SSP_ERR_RSA_DECRYPTION_ERROR	RSA Decryption error.

NOTE: Lower level drivers may return Common Error Codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.16.3 Crypto Framework Module Operational Overview

The following notes apply to SF_CRYPTO, SF_CRYPTO_TRNG, SF_CRYPTO_HASH, SF_CRYPTO_KEY, SF_CRYPTO_KEY_INSTALLATION, SF_CRYPTO_CIPHER and SF_CRYPTO_SIGNATURE modules.

Endianness configuration parameter

- In the Synergy Configurator if *only* the Crypto HAL drivers are selected without the Crypto Framework modules, endianness is a configurable parameter and set to *big endian* by default. This is the mode that was supported in previous releases.
- In the Synergy Configurator, if Crypto Framework modules are selected (the respective HAL component is included automatically), the endianness flag is locked to the *little endian* mode where only the byte array format is supported.
- The Crypto Framework Layer APIs support only byte arrays for input and output data.
- The Crypto HAL APIs support WORD/32-bit (uint32_t) arrays for input and output data.
- Users with data in byte array format should use the Crypto Framework modules.
- For any existing projects using the Crypto HAL API without the respective Framework API, Big Endian data may have been used and in that case, it needs to be converted to little Endian to use framework API or continue to use the existing HAL API directly.
- If using the HAL APIs directly, user needs to make sure that the data cast as (uint32_t) matches the endianness of the HAL module used.

Alignment for data buffers

- All data buffers allocated to be passed in as input buffers must be aligned on WORD boundary.***Blocking calls***
- *All Crypto Framework APIs are blocking calls.*

SF CRYPTO Framework Services

- The SF CRYPTO Framework modules include:
  - SF_CRYPTO for HW initialization and resource synchronization between the crypto modules.
  - SF_CRYPTO_TRNG for true random number generation.
  - SF_CRYPTO_HASH for message digest generation. Provides the ability to process data in chunks before finalizing the hash operation.
  - SF_CRYPTO_KEY for RSA, AES. Provides the option to generate wrapped keys in addition to plain text keys. Only the wrapped key support is present for ECC.
  - SF_CRYPTO_KEY_INSTALLATION provides key installation services for RSA, AES and ECC keys.
  - SF_CRYPTO_CIPHER provides encryption and decryption services for AES and RSA keys.
  - SF_CRYPTO_SIGNATURE provides RSA signature generation and verification services.

VersionGet API

- This API can be called at any time, i.e even before a module is opened.

SF CRYPTO Framework Module Overview Description

The SF CRYPTO Framework provides high-level APIs and is implemented on sf_crypto. The SF CRYPTO Framework provides a foundation for the Framework Crypto services through the Secure Cryptographic Engine (SCE) HAL module.

SF CRYPTO Framework Module Features

- The SF CRYPTO Framework opens/ initializes the underlying HW Secure Crypto Engine.

- Provides services for access to shared resources for other Crypto Framework modules SF_CRYPTO_TRNG, SF_CRYPTO_HASH, SF_CRYPTO_KEY, SF_CRYPTO_CIPHER, SF_CRYPTO_SIGNATURE and SF_CRYPTO_KEY_INSTALLATION.

SF CRYPTO Framework Module Operational Notes

- open API is called during initialization because the "Auto Initialization" configuration option is enabled by default.
- lock API and unlock API are for the SF_CRYPTO_XXX modules to to be used.However, if the application is making a direct call to HAL APIs, use lock API just before making the call to HAL module.Use unlock API just after returning from the call to HAL module.
- statusGet API requires the control block/p_ctrl as input parameter. Hence it should be called only after the open API of SF_CRYPTO module is called.

Configuration Settings for the SF CRYPTO Framework Module

Configuration parameter	Description
uint32_t wait_option	Wait option for RTOS service calls Defines how the service behaves if there is not enough memory available. The wait options are defined as follows: TX_NO_WAIT (0x00000000) T X_WAIT_FOREVER (0xFFFFFFFF) timeout value (0x00000001 through 0xFFFFFFFE) Selecting TX_NO_WAIT results in an immediate return from this service regardless of whether or not it was successful. This is the only valid option if the service is called from initialization. Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until enough memory is available. Selecting a numeric value (1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the memory. Default set through ISDE is TX_WAIT_FOREVER.

Configuration parameter	Description
crypto_instance_t * p_lower_lvl_crypto	Pointer to a low-level Crypto engine HAL driver instance. Configured by Synergy Configurator.
void const * p_extend	Extension parameter for hardware specific settings. Configured by Synergy Configurator.
void const * p_context	Placeholder for user data. For future expansion.
void * p_memory_pool	Byte pool address. Configured by Synergy Configurator.

Configuration parameter	Description
uint32_t memory_pool_size	Byte pool size. Default set by ISDE is 128 bytes. Caller to allocate pool based on the number of SF_CRYPTO_XXX module instances created. Recommended sizes: **In addition to the default byte pool**: NOTE: An additional 12 bytes per instance is needed for RTOS housekeeping. 24 + 12 bytes per instance of SF_CRYPTO_KEY module if RSA Key is required. 264 +12bytes per instance of SF_CRYPTO_KEY module if AES Key is required. SF_CRYPTO_KEY module if AES Key is required. 112 +12 bytes per instance of SF_CRYPTO_HASH module. 1200 bytes per instance of SF_CRYPTO_CIPHER module for RSA algorithm is required. 600 bytes per instance of SF_CRYPTO_CIPHER module for AES algorithm is required. 1450 bytes per instance of SF_CRYPTO_SIGNATURE module.

Configuration parameter	Description
sf_crypto_close_option_t close_option	Close option Selects how the SCE module can be closed. SF_CRYPTO_CLOSE_OPTION_DEFAULT - Close the module only if none of the other SF_CRYPTO_XXX modules are opened. NOTE: This is the default setting. SF_CRYPTO_CLOSE_OPTION_FORCE_CLOSE - Close the module regardless of SF_CRYPTO_XXX modules status. **Note: With either option, It is the responsibility of the caller to ensure that the SF CRYPTO module is not closed when any of the other Crypto Framework modules are open.**

SF CRYPTO Framework Module Limitations

It is the responsibility of the caller to ensure that the SF CRYPTO module is not closed when any of the other Crypto Framework modules are open.

SF CRYPTO TRNG Framework Module Overview Description

The SF CRYPTO TRNG Framework provides high-level APIs and is implemented on sf_crypto_trng. The SF CRYPTO TRNG Framework provides True Random Number Generation services through the Secure Cryptographic Engine (SCE) HAL module.

SF CRYPTO TRNG Framework Module Features

- The SF CRYPTO TRNG Framework module uses the TRNG HAL interfaces to the underlying Secure Crypto Engine.
- Access to shared resources through SF CRYPTO Framework module.

Configuration Settings for the SF SRYPTO TRNG Framework Module

Configuration parameter	Description
sf_crypto_instance_t * p_lower_lvl_common;	Pointer to a low-level Crypto engine HAL driver instance. Configured by Synergy Configurator.
trng_instance_t * p_lower_lvl_instance;	Pointer to a TRNG HAL driver instance. Configured by Synergy Configurator.
void * p_extend;	Extension parameter for hardware specific settings. For future use.

The configuration is set through the ISDE for TRNG HAL driver.

Configuration parameter	Description
uint32_t num_attempts	Number of attempts within which a true random number is to be generated. Set to 2 by default.

SF CRYPTO HASH Framework Module Overview Description

The SF CRYPTO HASH Framework provides high-level APIs and is implemented on sf_crypto_hash. The SF CRYPTO HASH Framework provides hash/message digest services through the Secure Cryptographic Engine (SCE) HAL module.

The hash functions supported are MD5, SHA-1, SHA-224 and SHA-256.

From FIPS Secure Hash Standard:

All of the algorithms are iterative, one-way hash functions that can process a message to produce a condensed representation called a _message digest*. These algorithms enable the determination of a message's integrity: any change to the message will, with a very high probability, result in a different message digest. This property is useful in the generation and verification of digital signatures and message authentication codes, and in the generation of random numbers or bits.

For the supported hash functions, the message size should be $< 2^{64}$ bits.

The message digest sizes are as follows:

MD5 : 16 bytes

SHA-1 : 20 bytes

SHA-224: 28 bytes

SHA-256: 32 bytes

SF CRYPTO HASH Framework Module Features

- The SF CRYPTO HASH Framework utilizes the underlying Secure Crypto Engine to provide HASH services.
- This module provides enhancements over the HAL driver such as:
  - Initializing the HASH value
  - Processing chunks of data through hashUpdate API before finalizing the hash operation.
  - Formatting the final block for the final HASH operation.

SF CRYPTO HASH Framework Module Operational Notes

Configuration Settings for the SF CRYPTO HASH Framework Module

Configuration parameter	Description
sf_crypto_hash_type_t hash_type;	HASH algorithm type. Select MD5, SHA1, SHA 224 or SHA256. Please refer to the header file for the the definitions.
crypto_instance_t * p_lower_lvl_crypto	Pointer to a low-level Crypto engine HAL driver instance. Configured by Synergy Configurator.
hash_instance_t * p_lower_lvl_instance;	pointer to HASH lower-level module instance. Configured by ISDE.
void * p_extend	Extension parameter for hardware specific settings.Optional. Configured by Synergy Configurator.

SF CRYPTO KEY Framework Module Features

The following key types can be generated using the sevices of the SF CRYPTO KEY module.

- RSA 2048-bit, 1024-bit plain text/raw keys.
- RSA 2048-bit, 1024-bit standard format wrapped private keys(public keys in plain).
- AES 128-bit, 192-bit and 256-bit wrapped keys for ECB, CBC, CTR and GCM chaining modes.
- AES 128-bit and 256-bit wrapped keys for XTS chaining mode.
- ECC 192-bit, 256-bit wrapped keys.

SF CRYPTO KEY Framework Module Operational Notes

The format of RSA plain text keys generated by the keyGenerate API is as follows:

RSA Public Key

Byte 0 to Byte 3 : Public key exponent

Byte 4 : Start of RSA modulus

(128 bytes for RSA 1024-bit and 256 bytes for RSA 2048-bit keys)

RSA Private Key in standard format

Byte 0 : Private key exponent (128 bytes for RSA 1024-bit and 256 bytes for RSA 2048-bit keys)

Followed by RSA modulus. (128 bytes for RSA 1024-bit and 256 bytes for RSA 2048-bit keys)

RSA Private key in CRT format

The components are ordered in the following order with exponent2 at byte 0:

exponent2 // the second factor's CRT exponent, a positive integer

prime2 // the second factor, a positive integer

exponent1 // the first factor's CRT exponent, a positive integer

prime1 // the first factor, a positive integer

coefficient // the (first) CRT coefficient, a positive integer

Configuration Settings for the SF SRYPTO KEY Framework Module

Configuration parameter	Description
sf_crypto_key_type_t key_type	Key type to be generated. Plain text or Wrapped keys. Please refer to the header file for the the definitions.
sf_crypto_key_size_t key_size	Key size to be generated. Please refer to the header file for the the definitions.
sf_crypto_data_handle_t domain_params;	Pointer to domain parameters for the requested key type. Structure contains, pointer to the data (contains the domain data) and data length. Structure contains the domain data in the order a, b, p, n for ECC as defined in FIPS186-3 and data length. To be set to NULL for RSA and AES Key types.

Configuration parameter	Description
sf_crypto_data_handle_t generator_point	Pointer to the generator base point of curve in the order Gx, Gy for ECC (where Gx and Gy are x and y coordinates respectively) and data length. To be set to NULL for RSA and AES key types.
sf_crypto_instance_t * p_lower_lvl_crypto_common	Pointer to a Crypto Framework common instance. Configured by Synergy Configurator.
void const * p_extend	Extension parameter for hardware specific settings (Reserved for future use). Configured by Synergy Configurator.

SF CRYPTO KEY Framework Module Limitations

1) For ECC only the wrapped keys are supported at present.

SF Crypto Signature Framework Module Overview Description

The SF CRYPTO Signature Framework provides high-level APIs and is implemented on sf_crypto_signature. This module is in the SSP framework layer that interfaces with Synergy HAL drivers for the hardware level cryptography operations. The module functions specified herein are used to sign/verify message that is provided as input.

SF Crypto Signature Module Features

- This module currently supports signature generation and signature-verification for RSA algorithm. There is support for both 1024-bits and 2048-bits key lengths for standard format plain-text, standard format wrapped private key and CRT plain-text keys.
- RSASSA-PKCS1 v1.5 is the supported signature scheme. The input message can be passed as raw message to be signed/verified or can be PKCS1 v1.5 encoded and padded before sign/verify.
- SHA1, SHA224, SHA256 are the supported hashing algorithms for PKCS1 v1.5 encoding /padding schemes.
- This module allows signature generation/verification on data which is smaller than a block size.
- If all of the data is not available at once, update APIs can be used to accumulate the incoming chunks of data and finally sign/verify the message only when all the message is gathered.
- Module allows to switching between sign and verify operations with less expensive API calls which are not involved in allocating and deallocating memory.

Configuration parameter	Description
sf_crypto_key_type_t key_type;	Type of Key RSA plain text or wrapped. Please refer to the header file for the the definitions.
sf_crypto_key_size_t key_size	Size of Key RSA 1024-bit /2048-bit key. Please refer to the header file for the the definitions.
sf_crypto_hash_instance_t * p_lower_lvl_sf_crypto_hash	Pointer to the Crypto Hash framework module instance structure. Configured by Synergy Configurator.
sf_crypto_instance_t * p_lower_lvl_crypto_common	pointer to Crypto Common module instance. Configured by Synergy Configurator.
void const * p_extend	Extension parameter for hardware specific settings.Optional. Configured by Synergy Configurator.

Configuration Settings for the SF CRYPTO Signature Framework Module

1) Operation modes for Signature Framework Module:
   a) SF_CRYPTO_SIGNATURE_MODE_SIGN
   b) SF_CRYPTO_SIGNATURE_MODE_VERIFY
2) Input Message Format to the Signature Framework APIs to perform sign or verify operation:
   a) SF_CRYPTO_SIGNATURE_MESSAGE_OPERATION_NONE
   b) SF_CRYPTO_SIGNATURE_MESSAGE_OPERATION_RSA_SHA1_PKCS1_1_5
   c) SF_CRYPTO_SIGNATURE_MESSAGE_OPERATION_RSA_SHA224_PKCS1_1_5
   d) SF_CRYPTO_SIGNATURE_MESSAGE_OPERATION_RSA_SHA256_PKCS1_1_5

SF Crypto Cipher Framework Module Overview Description

The SF CRYPTO Cipher Framework provides high-level APIs and is implemented on sf_crypto_cipher. This module is in the SSP framework layer and provides encryption and decryption services through the Secure Cryptographic Engine (SCE) HAL module.

SF Crypto Cipher Module Features

- This module currently supports encryption and decryption for AES and RSA algorithms.
- This module allows encryption/ decryption of data when available in chunks through the cipherUpdate()API and final() when the last chunk /all the data is gathered.
- Once the module is opened for a specific key type and key size, the encrypt and decrypt modes can be switched by calling the cipherInit() API.

AES algorithm support:

- AES 128-bit, 192-bit and 256-bit plain text and wrapped keys for the following chaining modes:
  - ECB (Electronic Code Book)
  - CBC (Cipher Block Chaining)
  - CTR (Counter Mode)
  - GCM (Galois Counter Mode)
- AES 128-bit and 256-bit plain text and wrapped keys for XTS (XEX-based tweaked code-book mode with ciphertext stealing) are supported.
- IV provided for AES GCM operations can be either 96-bits or a 96-bit IV formatted to 128-bits.
  - Example:
  - **96-bit IV: 94c1935afc061cbf254b936f**
  - **96-bit IV formatted to 128-bits: 94c1935afc061cbf254b936f00000001**
- PKCS#7 padding scheme is supported for ECB and CBC modes. This scheme can be used when the data is
- No padding option is supported for all modes where the data is exactly a multiple of the AES block size.
  - For GCM, no padding option is to be selected even when the data is not a multiple of the block size.

RSA algorithm support:

- RSA 1024-bit and 2048-bit plain-text standard format, plain-text CRT format and standard format wrapped keys are supported.
- RSA encrypt operation requires RSA public key and RSA decrypt operation requires RSA private key.

RSAES-PKCS1-v1_5 (RSA Encryption Scheme)

RSA-PKCS1 v1.5 encoding /padding scheme is supported.

- The input raw message is encoded and formatted to be encrypted. It requires that the message be less than k-11 where k is the length of the modulus of the selected key.

No padding option should be selected when an encoded and formatted block (exactly the size as the modulus of the selected key) needs to be encrypted/decrypted.

Configuration settings for SF Crypto Cipher Framework Module

Configuration parameter	Description
sf_crypto_key_type_t key_type	Key type for cipher operation: AES or RSA plain text or wrapped. Please refer to the header file for the the definitions.
sf_crypto_key_size_t key_size	Key size for cipher operation: AES 128-bit/192-bit/256-bit key or RSA 1024-bit /2048-bit key. Please refer to the header file for the the definitions.
sf_crypto_cipher_mode_t cipher_chaining_mode	Applicable only to AES algorithm: ECB, CBC, CTR, XTS or GCM chaining modes Set to ECB for RSA. Please refer to the header file for the the definitions.
sf_crypto_instance_t * p_lower_lvl_crypto_common	Pointer to Crypto Framework Common module instance. Configured by Synergy Configurator.
sf_crypto_trng_instance_t * p_lower_lvl_crypto_trng	Pointer to Crypto Framework TRNG module instance. Configured by Synergy Configurator.
void const * p_extend	Extension parameter for hardware specific settings.Optional. Configured by Synergy Configurator.

SF CRYPTO Cipher Framework Module Limitations

AES XTS mode only supports lengths which are a multiple of 32-bits/4 bytes.

SF CRYPTO KEY INSTALLATION Framework Module Overview Description

The SF CRYPTO KEY INSTALLATION Framework provides high-level APIs and is implemented on sf_crypto_key_installation. The SF CRYPTO KEY INSTALLATION Framework Key Installation services through the Secure Cryptographic Engine (SCE) HAL module.

NOTE:

1) The API returns the wrapped key to user and does not store the key for future use.It is the responsibility of the application to store the wrapped key in non-volatile memory for future use.
2) This Framework module provides the same level of service as the Crypto HAL module without any enhancements. It is provided for thread safe operation and for completeness of Crypto support in the Framework layer.

SF CRYPTO KEY INSTALLATION Framework Module Features

- The SF CRYPTO KEY INSTALLATION Framework module uses the KEY INSTALLATION HAL interfaces to the underlying Secure Crypto Engine.
- Access to shared resources through SF CRYPTO Framework module.
- This module supports key installation for the following keys:
  - RSA:
  - **1024-bit and 2048-bit plain text keys.**
  - AES:
  - **128-bit, 192-bit and 256-bit plain text keys for ECB, CBC, CTR and GCM chaining modes.**
  - **128-bit and 256-bit plain text keys for XTS chaining mode.**
  - ECC:
  - **192-bit and 256-bit plain text keys.**
- The user and install keys must be provided to the key installation API in the specified format.
- Upon installation the key installation service returns a wrapped key to the caller for any future usage of installed key on that device.

Configuration Settings for the SF SRYPTO KEY INSTALLATION Framework Module

Configuration parameter	Description
sf_crypto_key_type_t key_type;	Type of key to be installed. The prepared key will be of the encrypted type. Please refer to the header file for the the definitions.
sf_crypto_key_size_t key_size;	Size of key to be installed. Please refer to the header file for the the definitions.

Configuration parameter	Description
sf_crypto_instance_t * p_lower_lvl_common;	Pointer to a Crypto Framework common instance. Configured by Synergy Configurator.
key_installation_instance_t * p_lower_lvl_instance;	Pointer to Crypto Key Install HAL instance Configured by Synergy Configurator.
void const * p_extend;	Extension parameter for hardware specific settings. For future use.

Crypto Framework Module Important Operational Notes and Limitations

- The lock API and unlock API are for use with the SF_CRYPTO_XXX modules. However, if the application is making a direct call to HAL APIs which are not yet supported in the Framework layer, use the lock API just before making the call to HAL module. Use the unlock API just after returning from the call to HAL module.
- The statusGet API requires the control block/p_ctrl as an input parameter. Hence it should be called only after the open API of SF_CRYPTO module is called.
- All Crypto Framework APIs are blocking calls.
- Refer to the current SSP Release notes for limitations on this module.

4.1.16.4 Including the Crypto Framework Module in an Application

This section describes how to include the Crypto Framework Module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP based applications.

To add the Crypto Framework Module to an application, simply add it to a thread using the sacks selection sequence given in the following table. (The default name for the Crypto Framework Module is g_sf_crypt0 as shown in the following table. This name can be changed in the associated Properties window.)

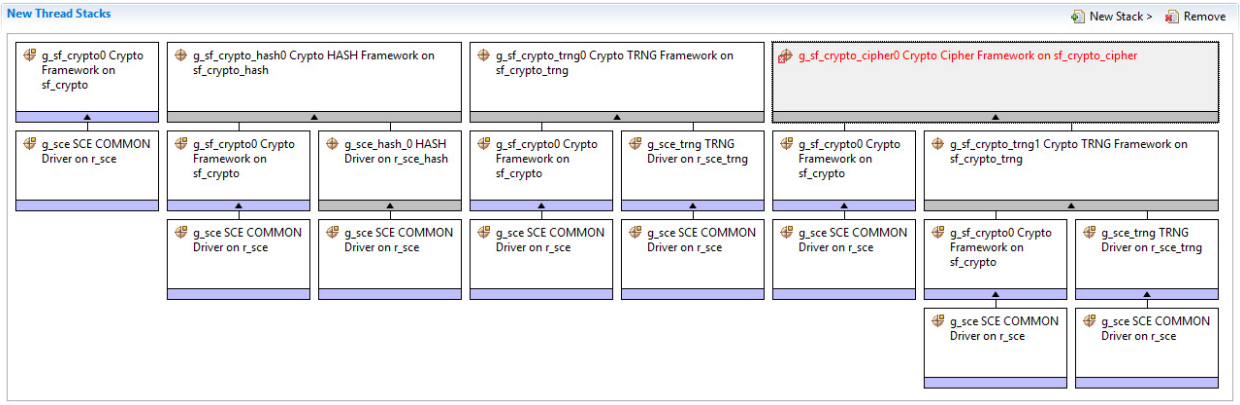
Crypto Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_crypt0 Crypto Framework on sf_crypto	Threads	New Stack> Framework> Crypto > Crypto Framework on sf_crypto

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_crypt_key0 Crypto Key Framework on sf_crypto_key	Threads	New Stack> Framework> Crypto > Crypto Key Framework on sf_crypto_key
g_sf_crypt0 Crypto TRNG Framework on sf_crypto_trng	Threads	New Stack> Framework> Crypto > Crypto TRNG Framework
g_sf_crypt0 Crypto HASH Framework on sf_crypto_hash	Threads	New Stack> Framework> Crypto > Crypto HASH Framework
g_sf_crypt_key0 Crypto Key Installation Framework on sf_crypto_key_installation	Threads	New Stack> Framework> Crypto > Crypto Key Installation Framework on sf_crypto_key_installation
g_sf_crypt0 Crypto cipher Framework on sf_crypto_cipher	Threads	New Stack> Framework> Crypto > Crypto cipher Framework
g_sf_crypt0 Crypto signature Framework on sf_crypto_signature	Threads	New Stack> Framework> Crypto > Crypto signature Framework

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

When the Crypto Framework Module is added to the Thread Stack, as shown in the figure below, the configurator automatically adds the needed lower level drivers. Any drivers that need additional configuration information will be box text highlighted in red. Modules with a gray band are individual modules that stand alone. Modules with a blue band are shared or common and need only be added once, since they can be used by multiple stacks. Modules with a pink band can require the selection of lower level drivers. Sometimes these are optional or recommended and this is indicated in the block with the inclusion of this text. If the addition of lower level drivers is required, the module description will include "Add" in the text. Clicking on any pink banded modules will bring up the "New" icon and then will show the possible choices.



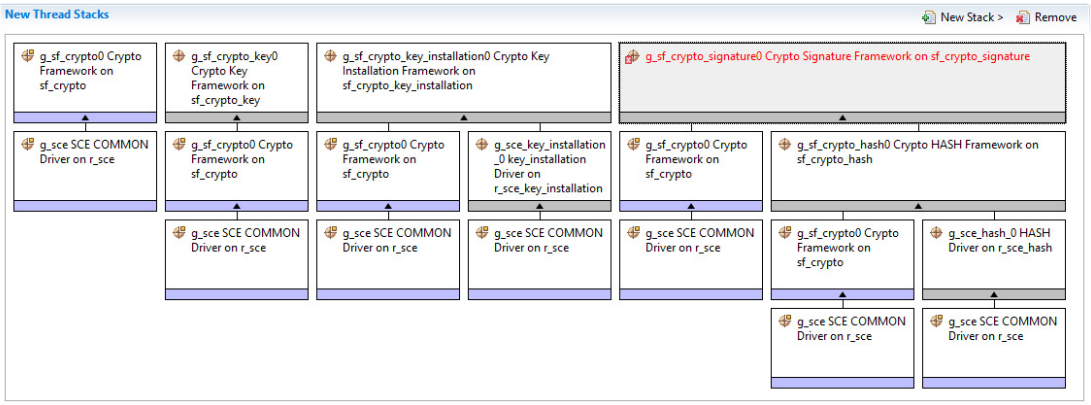


Figure 174: Crypto Framework Modules Stack- Crypto, Hash, Key and TRNG

4.1.16.5 Configuring the Crypto Framework Module

The Crypto Framework Module has configurable properties used to define operation parameters.

NOTE: You may want to open your ISDE and create the Crypto Framework Module and explore the property settings in parallel with looking over the Configuration Table Settings given below. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the SF CRYPTO Framework Module

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_sf_crypto0	Module name.
Wait time	TX_WAIT_FOREVER	Value must be a non-negative integer.
Byte Pool Size	128	Byte pool size selection.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

ISDE Property	Value	Description
Force Closure Support	Default, Force Close Default: Default	Force closure support selection.

Configuration Settings for the SF SRYPTO HASH Framework Module

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Instance Name	g_sf_crypto_hash0	Module name.
Name of generated initialization function	sf_crypto_hash_init0	Name of generated initialization function
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Configuration Settings for the SF CRYPTO KEY Framework Module

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_sf_crypto_key0	Module name.
Key type (For AES, Byte Pool size in sf_crypto mst be > = 280 bytes)	RSA Plain text, RSA CRT Plain text, RSA Wrapped, AES Wrapped, ECC Plain text, ECC Wrapped Default: RSA Plain text	Key type selection.

ISDE Property	Value	Description
Key size	RSA 1024-bits, RSA 2048-bits, AES 128-bits, AES XTS 128-bits, AES 192-bits, AES 256-bits, AES XTS 256-bits, ECC 192-bits, ECC 256-bits Default: RSA 2048-bits	Key size selection.
Name of generated initialization function	sf_crypto_key_init0	Name of generated initialization function
Name of Domain Parameter (Applicable only for ECC)	sf_crypto_key_domain_params0	Name of domain parameter
Name of Generator Point (Applicable only for ECC)	sf_crypto_key_generator_point0	Name of generator point
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Configuration Settings for the SF CRYPTO KEY INSTALLATION Framework Module

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_sf_crypto_key_installation0	Module name.
Key type	Encrypted RSA Key, Encrypted AES Key, Encrypted ECC Key Default: Encrypted RSA Key	Key type selection.
Key size	RSA 1024-bits, RSA 2048-bits, AES 128-bits, AES XTS 128-bits, AES 192-bits, AES 256-bits, AES XTS 256-bits, ECC 192-bits, ECC 256-bits Default: RSA 2048-bits	Key size selection.

ISDE Property	Value	Description
Name of generated initialization function	sf_crypto_key_installation_init0	Name of generated initialization function.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Configuration Settings for the SF CRYPTO TRNG Framework Module

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_sf_crypto_trng0	Module name.
Name of generated initialization function	sf_crypt_trng_init0	Name of generated initalization function
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Configuration Settings for the SF CRYPTO Cipher Framework Module

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_sf_crypto_cipher0	Module name.
Key type	RSA Plain text, RSA CRT Plain text, RSA Wrapped, AES Wrapped Default: RSA Plain text	Key type selection.

ISDE Property	Value	Description
Key size	RSA 1024-bits, RSA 2048-bits, AES 128-bits, AES XTS 128-bits, AES 192-bits, AES 256-bits, AES XTS 256-bits Default: RSA 2048-bits	Key size selection.
Cipher chaining mode	ECB, CBC, CTR, GCM, XTS Default: ECB	Cipher chaining mode selection.
Name of generated initialization function	sf_crypto_cipher_init0	Name of generated initialization function.
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Configuration Settings for the SF CRYPTO Signature Framework Module

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_sf_crypto_signature0	Module name.
Key type (Byte Pool size in sf_crypto must be > = 1450 bytes)	RSA Plain text, RSA CRT Plain text, RSA Wrapped Default: RSA Plain text	Key type selection.
Key size	RSA 1024-bits, RSA 2048-bits Default: RSA 2048-bits	Key size selection.
Name of generated initialization function	sf_crypto_signature_init0	Name of generated initialization function.

ISDE Property	Value	Description
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection.

Configuration Settings for the R_SCE Common Module

ISDE Property	Value	Description
Name	g_sce	Module name
Endian flag	CRYPTO_WORD_ENDIAN_LITTLE	Endian flag selection (locked)

Crypto Framework Module Clock Configuration

The CRYPTO Framework module uses the same clock as the SCE.

Crypto Framework Module Pin Configuration

The CRYPTO Framework module doesn't use any external pins.

4.1.16.6 Using the Crypto Framework Module in an Application

To use the SF CRYPTO module:

- Ensure that the configuration parameters are set as per the needs of the application.
- Use the open API to Initialize the SF_CRYPTO and the SCE HAL module (R_SCE) through the SCE common driver. This is done by Synergy Configurator when the Auto Initialize setting is at default.
- The little endian mode is set by default. It is locked and not configurable.
- The open function cannot be called again until the module is closed.Use the close API to close the Crypto Framework services and the HW SCE.

Using the SF CRYPTO TRNG Framework Module in an Application

The typical steps in using the SF CRYPTO TRNG Framework module in an application are:

1) Use the open API to Initialize the SF_CRYPTO_TRNG and the SCE TRNG HAL module (R_SCE_TRNG). This is done by ISDE when the Auto Initialize setting is at default.

   - NOTE: The little endian mode is set by default. It is locked and not configurable. (Please refer to note on endianness and data format in the Module Operational Notes.)

   - NOTE: The open function cannot be called again until the module is closed.

2) Use the randomNumberGenerate API to generate random numbers. The minimum length of random number bytes that can be requested is 1.
3) Use the close API to close the Crypto Framework services and the SCE when the TRNG services are no longer required.

NOTE: In the rare event that the RandomNumberGenerate API returns an error, all the CRYPTO Framework modules must be closed and the SF CRYPTO module re-opened.

NOTE: There will be only one instance of the HAL TRNG driver. Hence the configuration parameter set of each SF CRYPTO TRNG instance will be overwriting the one configured previously. The last value configured is applicable to all instances.

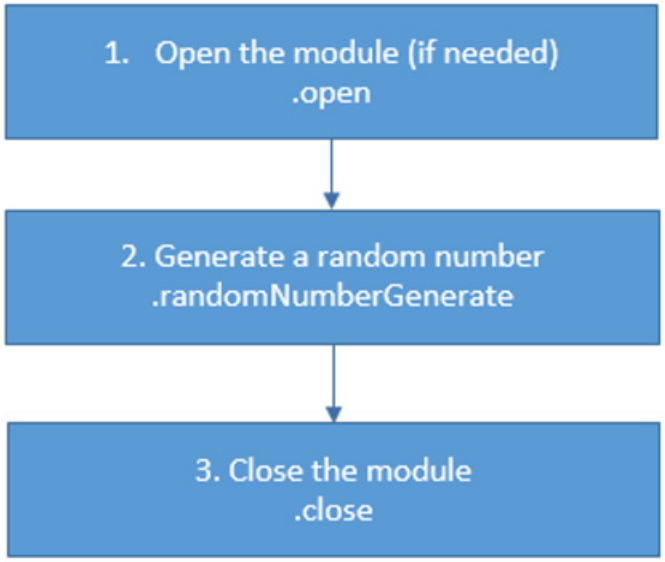


Figure 175: Crypto TRNG Framework Typical

Additional descriptions of some other Crypto functions are given in the following sections. No flow diagrams are provided however in the interest of brevity.

Using the SF CRYPTO HASH Framework Module in an Application

The typical steps in using the SF CRYPTO HASH Framework module in an application are:

Calling the APIs in this order: open->hashInit->hashUpdate->hashFinal->close.

Details:

- Add SF_CRYPTO HASH Framework module in the Synergy Configurator. The HASH algorithm for this instance is set to SHA 256 by default.

- The little endian mode is set by default. It is locked and not configurable. (Please refer to note on endianness and data format in the Module Operational Notes.)
- The SF CRYPTO Module has to be first opened.
  - Synergy Configurator will do this provided the Auto start option set by default is retained.
- Set the chosen HASH algorithm as the configuration parameter for the module open API.
- Use the open API to Initialize the SF CRYPTO HASH Framework module and the SCE HASH HAL module (R_SCE_HASH) through the SCE HASH driver.
  - Synergy Configurator will do this provided the Auto init option set by default is retained.
- The open function cannot be called again until the module is closed.
- Use the hashInit API to initialize the HASH operation for the chosen HASH algorithm.
  - The hashInit can be called after open API or hashUpdate API or hashFinal API to initialize a new operation with the configured HASH algorithm.
- Use the hashUpdate API to hash input data. It can be called multiple times till all the data is completely used up.
- Use the hashFinal API to get the message digest for all the data hashed with the hashUpdate API.
  - Once the hashFinal API is called, either the hashInit can be called to initialize operation for the next set of data using the same HASH algorithm or close API can be called to close the module. It can then be re-opened for a different HASH algorithm.
- Use close API to close the Crypto HASH Framework services.

NOTE: On a particular thread a single instance of SF CRYPTO HASH module can be re-used for different HASH algorithms without having to create multiple instances.

Using the SF CRYPTO KEY Framework Module in an Application

The typical steps in using the SF CRYPTO KEY Framework module in an application are:

Calling the APIs in this order: open->keyGenerate->close.

Details:

- Please refer to note on endianness and data format in the Module Operational Notes.
- The SF CRYPTO Module has to be first opened. ISDE will do this provided the Auto initialize option set by default is retained. Refer to the section Using the SF CRYPTO Framework Module in an Application.
- Set the configuration parameters as per the needs of the application required for the module open API.
- Use the open API to Initialize the SF CRYPTO KEY Framework module (This is done by ISDE when the Auto start option is at default setting) and the SCE HAL modules through the SCE drivers.
- The open function cannot be called again until the module is closed.
- Use the keyGenerate API to generate the cryptographic keys of the type and size set by the configuration parameters at the open call.
- Use close API to close the Crypto Key Framework Module services.

NOTE: On a particular thread a single instance of SF CRYPTO KEY module can be re-used for generating different key

types without having to create multiple instances.

Using the SF CRYPTO SIGNATURE Framework Module in an Application

The typical steps in using the SF CRYPTO Signature Framework module in an application are:

Calling the APIs in this order: open->contextInit->signUpdate->signFinal->close.

Calling the APIs in this order: open->contextInit->verifyUpdate->verifyFinal->close.

Details:

- Please refer to note on endianness and data format in the Module Operational Notes.
- The SF CRYPTO Module has to be first opened. ISDE will do this provided the Auto initialize option set by default is retained. Refer to the section Using the SF CRYPTO Framework Module in an Application.
- Set the configuration parameters as per the needs of the application required for the module open API.
- Use the open API to Open the SF CRYPTO Signature Framework module (This is done by ISDE when the Auto start option is at default setting) and the SCE HAL modules through the SCE drivers.
- The open function cannot be called again until the module is closed.
- Use the contextInit API to initialize the context by assigning appropriate operation mode, key (public key for verify operation mode and private key for sign operation mode), message formatting option.
- If the data is coming in as smaller chunks for sign operation mode, use the Update APIs - signUpdate API to accumulate the data for future signing operation or for verify operation mode use the verifyUpdate API to accumulate the data for future signatureverification.
- In order to complete the sign or verify operation call the Final APIs - signFinal or verifyFinal API respectively. These APIs support accepting last chunk of incoming data. In case all the data has been passed through one of the Update APIs the parameter for last message chunk can be set to NULL.
- In case all the data to be signer or verified is available at once, signFinal or verifyFinal API can be called directly without using any of the Update APIs.
- Use close API to close the Crypto Signature Framework Module services.

NOTE: On a particular thread a single instance of SF CRYPTO Signature module can be re-used for alternately performing sign or verify operations by appropriately setting/re-setting the operation mode and other related parameters. In order to perform sign or verify operations simultaneously two instances of SF Crypto Signature module need to be used.

Using the SF CRYPTO CIPHER Framework Module in an Application

The typical steps in using the SF CRYPTO Cipher Framework module in an application are:

Calling the APIs in this order: open->cipherInit->cipherUpdate->cipherFinal->close.

Details:

- Please refer to note on endianness and data format in the Module Operational Notes.
- The SF CRYPTO Module has to be first opened. ISDE will do this provided the Auto initialize option set by default is retained. Refer to the section Using the SF CRYPTO Framework Module in an Application.
- Set the configuration parameters for the module open API as per the needs of the application.
- Use the open API to open the SF CRYPTO Cipher Framework module (This is done by ISDE when the Auto start option is at default setting) and the SCE HAL modules through the SCE drivers.
- The open function cannot be called again until the module is closed.

- Use the ciphertInit API to initialize the context by assigning appropriate operation mode, key and padding scheme option and algorithm specific parameters.
- If the data for encrypt/decrypt operation is not available all at once but is available in chunks, use the cipherUpdate API. The cipherUpdate API can be called multiple times.

  NOTE: For RSA operations, no data is output from the cipherUpdate API. The message will only be accumulated till the cipherFinal API is called.

- In order to complete the encrypt/decrypt operation call the cipherFinal API. This API accepts last chunk of incoming data. In case all the data has been passed through the cipherUpdate APIs the length for last message chunk can be set to 0.
- In case all the data to be encrypted/decrypted is available at once, cipherFinal API can be called directly without using the cipherUpdate API.
- Use close API to close the Crypto Cipher Framework Module services.

  NOTE: AES GCM operation specifics:

- For AES GCM encrypt/decrypt operations, AAD (Additional Authenticated Data) is optional. If it is to be used, it has to be provided for the cipher operation through the cipherAadUpdate API after the cipherInit is called and prior to calling the cipherUpdate or cipherFinal APIs.
- For AES GCM encrypt operation, the tag (Authentication Tag) will be generated when cipherFinal API is executed. The caller has to supply the buffer to hold the tag through the cipherInit API.
- For AES GCM decrypt operation, the tag has to be provided prior to providing any ciphertext data through the cipherUpdate / cipherFinal APIs. Hence the caller is required to provide the the tag through the cipherInit API.
- The AES GCM decrypt operation may output plain text data through the cipherUpdate/cipherFinal APIs but it should not be consumed if the decrypt operation returns an error code. This is to ensure that the tag is verified before the data is used.

  NOTE: On a particular thread a single instance of SF CRYPTO Cipher module can be re-used for alternately performing encrypt or decrypt operations by appropriately setting/re-setting the operation mode and other related parameters or two instances of SF Crypto Cipher module can be used.

4.1.17 Capacitive Touch Framework

The Capacitive Touch Framework provides high-level APIs for Capacitive Touch applications using the ThreadX RTOS. The Capacitive Touch Framework supports the CTSU peripherals on the Synergy microcontroller hardware and is implemented on sf_touch_ctsu. This framework is designed to be used in conjunction with the configuration data

generated by the **Workbench 6** tool. This section describes how to configure the Framework CTSU Interface using the e2 studio ISDE and how to include the API functions in your application.

In the Project Configurator in the e2 studio ISDE, you can add and configure a Capacitive Touch Framework Module in the Modules pane of the Threads tab by selecting **New > Framework > Input > Cap Touch Framework on sf_touch_ctsu**. For details, see: Using e2 studio to Write an Application with the Capacitive Touch Framework

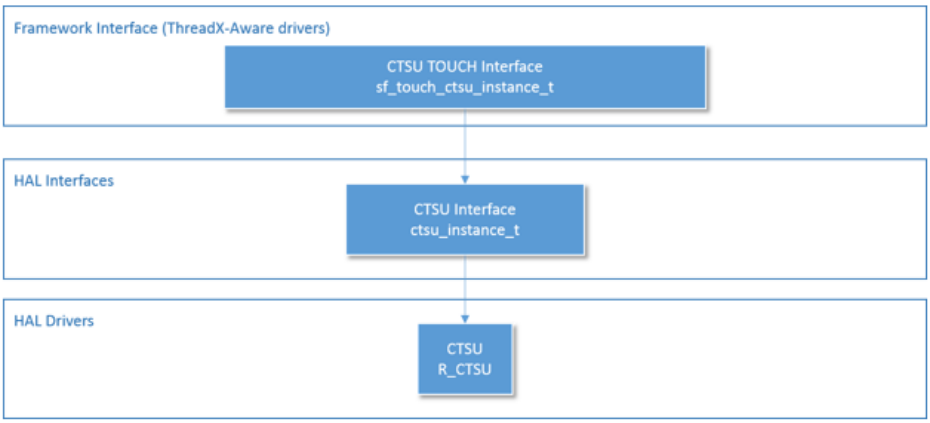


Figure 176: Capacitive Touch Framework – Block Diagram

You can find the API reference for the Framework CTSU Interface here: CTSU Framework Interface.

Also see CTSU Driver.

To understand how to program with Interfaces in the SSP, see: SSP Architecture

4.1.17.1 What Does the Capacitive Touch Framework Do?

The CTSU Framework Interface creates a private thread which drives a hardware scan of a capacitive touch panel and updates the panel at a periodic rate. The Framework Module reads the scanned results using the HAL Layer CTSU driver.

When a scan is completed, the callback registered by the application layer is invoked. If multiple upper layers are using this framework (for example: button, slider, wheel), this layer invokes the callbacks for each of those layers (if not NULL) in the order that they initialized this layer. Note that the frequency of the internal thread (update_hz) is set/modified with each subsequent open() call made to this framework.

4.1.17.2 Using e2 studio to Write an Application with the Capacitive Touch Framework

The driver is integrated into the Synergy Software Package in e2 studio: e2 studio ISDE User Guide

In e2 studio, create and configure a project and add the drivers:

1) Create the project: Creating a Project.
2) Configure the project: Configuring a Project
3) Add the drivers: Adding Drivers to a Thread and Configuring the Drivers

The following resources are required for any application using the CTSU Framework Interface:

Resource	ISDE Tab	Selection
Framework CTSU Driver	Threads	**Framework > Input > Cap Touch Framework on sf_touch_ctsu**
CTSU Interrupts	ICU	See Configuring the Interrupts for the Capacitive Touch Framework

Configuring the Clock for the Capacitive Touch Framework

The CTSU framework uses the same configuration as the CTSU HAL driver: Set the CTSU clock speed to match the the clock speed selected in the capacitive touch tuning tool **Workbench 6**.

See CTSU Driver.

Configuring the Pins for the Capacitive Touch Framework

The CTSU framework uses the same configuration as the CTSU HAL driver: Set the pins as touch sensor pins TSxx and enable the TSCAP function.

See CTSU Driver.

Configuring the Interrupts for the Capacitive Touch Framework

The CTSU framework uses the same configuration as the CTSU HAL driver: Enable the three CTSU interrupts in the ICU tab by setting them all to the same priority. Any value works.

See CTSU Driver.

Configuring the Capacitive Touch Framework Parameters

Use the e^2 studio ISDE to configure the Capacitive Touch Framework parameters: Adding Drivers to a Thread and Configuring the Drivers

The Capacitive Touch Framework is configured by passing a pointer of type sf_ctsu_cfg_t in the SF_CTSU_Open() API.

Capacitive Touch Framework configuration

ISDE Property	Configuration	Setting	Description
Name	-	Arbitrary symbol	Driver name
Lower Level Touch Driver Name	p_ctsu_instance	Name of lower level driver	-
Thread Priority	priority	ThreadX Priority:	User determined based on priority of other threads in system. Recommend high priority.

ISDE Property	Configuration	Setting	Description
Update Hz	update_hz	>10 Hz	Rate at which the CTSU hardware scans should occur
Callback	p_callback	>10 Hz	Name of user callback that will be called when each scan is complete and also when new processed data is available. Can be NULL if not used.

Configuring the CTSU HAL Driver Parameters

Use the e^2 studio ISDE to configure the HAL layer CTSU driver parameters: Adding Drivers to a Thread and Configuring the Drivers

Writing a Capacitive Touch Framework Application

1) Ensure that a folder with the data generated by **Workbench 6** exists in the ${PROJECT_ROOT}\src folder and is included in the build.
2) In the Threads tab create a new thread or add to an existing thread **Cap Touch Framework on sf_touch_ctsu**. Note the thread **Name**
3) In the same thread add a **Touch Driver on r_ctsu**.
4) Using the **Properties** tab for the Touch Driver on CTSU, note the Name** of the module and **CTSU Configuration Used**. (For more information on configuring the CTSU HAL module, refer to the usage notes for the CTSU HAL module).
5) Using the **Properties** tab for the **Cap Touch Framework on sf_touch_ctsu**, set the **Name** for this driver.
6) Ensure that the Lower Level Touch Driver Name provided is the same as the Touch Driver on CTSU Module entry.
7) Set the Thread Priority field to a high priority...e.g. 5.
8) Set the Update Hz field to 100.
9) Click Generate Project Content to generate the required structures.
10) Open **Thread_name.h** file under ssp_gen to see the generated structures that you need to use.
11) Build the Project.
12) Use SF_TOUCH_CTSU **Name.api->read** in the application thread to read the scanned data.
13) Connect the hardware, download the executable and run.
14) Add the data read from **Name.api->read** to a watch window.

Touching your Capacitive Touch Sensors now causes the binary bit mask to change according to configuration.

4.1.17.3 CTSU Framework Limitations

There are no known limitations for using this module. Also see the SSP release notes.

4.1.17.4 CTSU Framework Interface Files

During the project configuration, the ISDE extracts the files shown in the following table to the /ssp directory.

Module	Directory
CTSU Framework Interface	synergy/ssp/inc/framework/api/sf_touch_ctsu_api.h
CTSU Framework Instance	synergy/ssp/inc/framework/instances/sf_touch_ctsu.h
CTSU Framework Driver	synergy/ssp/src/framework/sf_touch_ctsu
CTSU HAL Interface	synergy/ssp/inc/driver/spi/r_ctsu_api.h
CTSU HAL Instance	synergy/ssp/inc/driver/instances/r_ctsu.h
CTSU HAL Driver	synergy/ssp/src/driver/r_ctsu

4.1.17.5 CTSU Framework Supported Devices

The driver is designed to support the following families with no changes to the API:

- S7G2
- S3A7
- S124

The CTSU Framework is designed to support any Synergy device with a CTSU peripheral and CTSU Driver with no changes to the API.

4.1.18 Capacitive Touch Button Framework

The Capacitive Touch Button Framework module provides high-level ThreadX-aware APIs for Capacitive Touch Button applications and is implemented on sf_touch_ctsu using the ThreadX RTOS. The Capacitive Touch Button Framework module uses the CTSU peripheral on the Synergy MCU. A user-defined callback can be created to respond to each button in the order in which they are present.

4.1.18.1 Capacitive Touch Button Framework Module Features

The Capacitive Touch Button Framework module is used to interpret the CTSU data for all the buttons that are present in the system. Some of the key features are:

- Works in conjunction with the Capacitive Touch Workbench for Renesas Synergy (CTW) tool which generates configuration data.

  - Supports S124, S128, S3A6, S3A7, S5D5, S5D9 and S7G2
- Provides a callback function to events.
  - Performs de-bouncing
  - Supports multiple types of events including Press, Release, and LongTouch
  - Calls the callback for each button in the order in which they are present in the button configuration table.

The Capacitive Touch Button Framework requires the Capacitive Touch framework. In most cases, all the needed configuration information is automatically added to the modules. The ISDE configuration options are described further in this document.

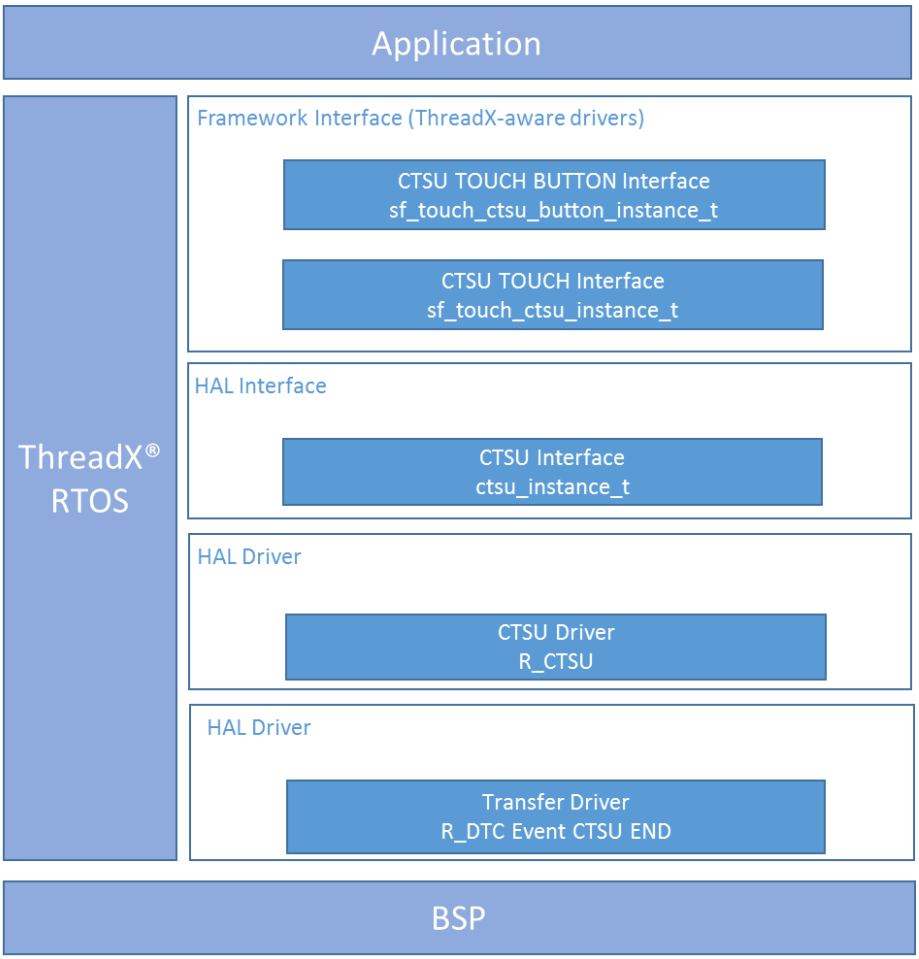


Figure 177: Capacitive Touch Button Organization, Options and Stack Implementations

4.1.18.2 Capacitive Touch Button Framework Module APIs Overview

The Capacitive Touch Button defines APIs for open, enable, disable and close. A complete list of the available APIs, an example API call, and a short description of each can be found in the table below. A table of status return values follows.

Capacitive Touch Button Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_touch_button0.p_api->open(g_sf_touch_button0.p_ctrl, g_sf_touch_button0.p_cfg); Initialize the Touch CTSU Button Framework; configures the lower level hardware and registers callback functions for all the buttons.
enable	g_sf_touch_button0.p_api->enable(g_sf_touch_button0.p_ctrl, button_id); Enable callback notification for a configured button.
disable	g_sf_touch_button0.p_api->disable(g_sf_touch_button0.p_ctrl, button_id); Disable callback notification for a configured button.
close	g_sf_touch_button0.p_api->open(g_sf_touch_button0.p_ctrl); Close the Touch CTSU Button Framework; Closes the button framework and lower layers if no other modules are using it.
versionGet	g_sf_touch_button0.p_api->versionGet(&p_version); Get version and store it in provided pointer p_version.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables review the SSP Users Manuals API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_ASSERTION	Assertion error.

Name	Description
SSP_ERR_IN_USE	The framework has already been initialized by this particular widget.
SSP_ERR_NOT_OPEN	Device not open.
SSP_ERR_INTERNAL	Internal error.

NOTE: Lower level drivers may return Common Error Codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.18.3 Capacitive Touch Button Framework Module Operational Overview

The Capacitive Touch Button Framework is used to interpret the CTSU data for all the buttons that are present in the system. It also initializes the CTSU framework layer which in turn initializes the hardware layer. The Capacitive Touch Button Framework registers a callback with the CTSU Framework layer which will be called each time processed data is available. The Capacitive Touch Button Framework then uses this processed data (binary) to perform de-bouncing and to determine which of the configured events (Press, Release, LongTouch and so forth) are valid for each button.

The Framework calls the callback for each button in the order in which they are present in the button configuration table. Unless the user stops the scan process, the scan continues to be triggered by the timer and data will be written into the user buffer, which is treated by the Framework as a circular buffer. The name and length of the buffer are specified via the ISDE configurator.

Capacitive Touch Button Framework Module Operational Notes

This framework is designed to be used in conjunction with the configuration data generated by the Capacitive Touch Workbench for Renesas Synergy tool which you can download from the Renesas Synergy Gallery in the SSP Utilities tab.

Capacitive Touch Button Framework Module Limitations

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.18.4 Including the Capacitive Touch Button Framework Module in an Application

This section describes how to include the Capacitive Touch Button in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP based applications.

To add the Capacitive Touch Button to an application, simply add it to a thread using the Stacks Selection Sequence given in the table below. (The default name for the Capacitive Touch Button is g_sf_touch_button0. This name can be changed in the associated **Properties** window.)

Capacitive Touch Button Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_touch_button0 Capacitive Touch Button Framework on sf_touch_ctsu_button	**Threads**	**New Stack > Framework > Input > Capacitive Touch Button Framework on sf_touch_ctsu_button**

When the Capacitive Touch Button Framework module on sf_touch_ctsu_button is added to the thread stack as shown in the figure below, the configurator automatically adds any needed lower level modules. Any drivers that need additional configuration information will have box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower level drivers. These are either optional or recommended (this is indicated in the block with the inclusion of this text.) If the addition of lower level drivers is required, the module description will include "Add" in the text. Clicking on any Pink banded modules will bring up the "New" icon and then display the possible choices.

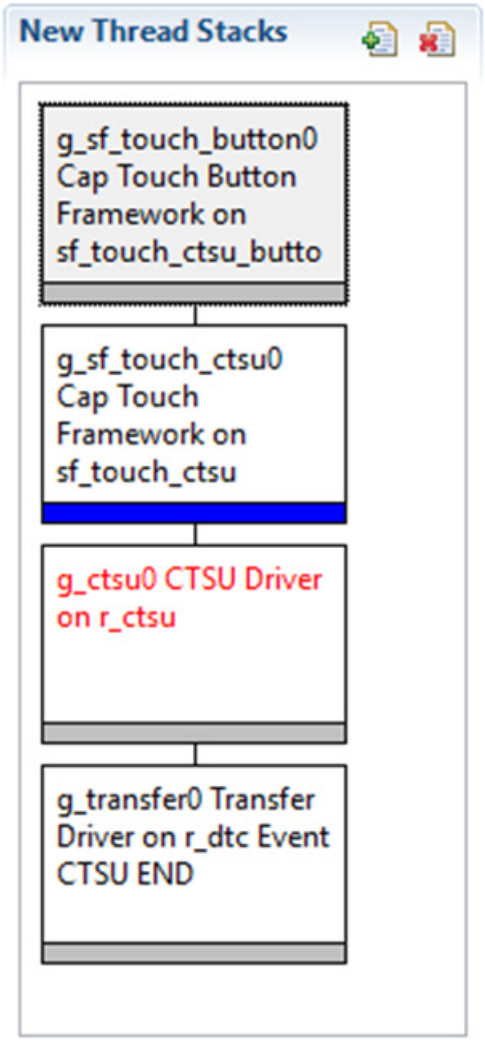


Figure 178: Capacitive Touch Button Framework Module Stack

4.1.18.5 Configuring the Capacitive Touch Button Framework Module

The Capacitive Touch Button module must be configured by you for the desired operation. The SSP configuration window will automatically identify, by highlighting the block in red, any required configuration selections, such as Interrupts or operating modes, which must be configured for lower level modules, for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the **Properties** tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority. This configuration setting is available with the properties window of the associated module. Simply select the indicated module and then view the properties window. The Interrupt settings are often toward the bottom of the properties list, so scroll down until they

become available. Also note that the Interrupt priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the below configuration properties tables, but is easily visible with the ISDE when configuring Interrupt Priority levels.

NOTE: You may want to open your ISDE and create the module and explore the property settings in parallel with looking over the Configuration Table Settings given below. This will help orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Capacitive Touch Button Framework on sf_touch_ctsu_button

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Controls whether to include code for API parameter checking
Number of Buttons	1	Number of buttons configured in CTW
Short hold debounce multiplier	1	Specify the multiplier for a button short-hold event
Long hold debouce multiplier	5	Specify the multiplier for a button long-hold event
Stuck in debounce multiplier	10	Multiplier used in debounce function
Multi touch enable	Enabled, Disabled Default: Disabled	Enables or disables multi-touch detection
Enable stuck at condition detection	Enabled, Disabled Default: Disabled	Specify the multiplier for a button stuck-at event
Name	g_sf_touch_button0	Module name
Button Configuration Structure Name	touch_buttons	Name of button configuration structure generated by CTW
Callback	g_button_framework_user_callback	Name of callback function created by user application

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower level modules can be desirable. The configurable properties for the lower level stack modules are given in the below sections for completeness and as a reference.

NOTE: Most of the property settings for lower level modules are fairly intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the Capacitive Touch Button Framework Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers and these are indicated via the red text in the Thread Stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The table below identifies all the settings within the properties section for the module.

Configuration Settings for the Capacitive Touch Framework on sf_touch_ctsu

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter error checking.
Name	g_sf_touch_ctsu0	Module name.
Thread Priority	3	User determined based on priority of other threads in system. Recommend high priority.

ISDE Property	Value	Description
Update Hz	50	Rate at which the CTSU hardware scans should occur. The maximum scan rate should be less than the tick rate set for ThreadX. The total time to complete a hardware scan is determined by the number of channels used. Each channel takes approximately 600 usec to complete a scan. Thus if 10 channels are used for 10 buttons in self-capacitance mode, the total hardware scan time is ~6 milliseconds or 166 Hz. Similarly, if 7 channels are used for 5 buttons in mutual-capacitance mode (5x2 layout), the total hardware scan time is ~4.2 milliseconds or 250 Hz.The software processing time is additional and will vary depending on the core clock and the software filter depth used by the CTSU. Thus the maximum scan rate should be lesser than the time required to scan and process all the channels used in the configuration and lesser than the RTOS tick rate.
Callback	NULL	Name of user callback that will be called when each scan is complete and also when new processed data is available. Can be NULL if not used.

Configuration Settings for the CTSU HAL Module on r_ctsu

ISDE Property	Value	Description
Parameter Checking Enable	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Offset Adjustment	Enabled, Disabled Default: Enabled	Used to enable or disable offset adjustment. Leave as enabled unless you are tuning the board. (Rate is determined by *Runtime rate of tuning of sensor values*.)

ISDE Property	Value	Description
Drift Compensation	Enabled, Disabled Default: Enabled	Used to enable or disable drift compensation. Leave as enabled unless you are tuning the board. Drift compensation allows the baseline values used to determine the presence or absence of a touch to change over time as the board or surface ages or with changes in temperature and environment. Drift compensation rate is determined by *Steady state drift compensation rate*, *Startup drift compensation rate*, and *Channel release compensation rate*.
Drift Compensation Method (valid only if Drift Compensation is enabled above)	Alternate Method 1	Drift Compensation algorithms provided. Only Alternate 1 is currently supported. Other options may be supported in future releases. Drift compensation method Alternate 1 allows for different rates of drift compensation for the following events: steady state, startup, and channel release.
Steady state drift compensation rate, drift compensation will be applied per n scans	500	Determines the rate of drift compensation (applied every n scans) when the channel is in steady state. (Dependent on the scan rate.)
Startup drift compensation rate (Should be less than the steady state drift compensation)	5	Determines the rate of drift compensation (applied every n scans) when the driver is performing its initialization. (Should be less than the *Steady state drift compensation rate*.)
Channel release compensation rate (Should be less than the steady state drift compensation rate)	500	Determines the rate of drift compensation (applied every n scans) when a channel transitions from pressed to released. (Should be less than or equal to the *Steady state drift compensation rate*.)

ISDE Property	Value	Description
Default filter depth (used in sensor count filter provided by driver)	1	Used in the software sensor count filter provided by driver. This default filter is a modification of the relatively common "leaky integrator" software filter. A depth of 4 equates to an approximate filter constant of 16. When the input is a constant value of 16 or greater, the filter output will reach 68-69 % of the constant input value after 16 cycles/iterations of the filter.
Runtime rate of tuning of sensor values (if drift compensation is used this value is overridden to be twice the rate of the steady state drift compensation)	800	Rate of offset adjustment. (If drift compensation is used this value is set to twice the rate of the steady state drift compensation.)
Perform auto-tune and drift compensation only when all channels are untouched	True, False Default: True	Perform auto-tune and drift compensation only when all channels are untouched.
Max. active channels	1	Specify the maximum number of channels that are to be used by the application. In Self Capacitance Mode, this is the number of channels used. In Mutual Capacitance Mode, this is the multiplication result of the Rx and Tx channels. For example: 4 Rx and 3 Tx channels = 12 Maximum Active Channels.
Name	g_ctsu0	Module name.
CTSU configuration used	g_ctsu0_config	Name of the configuration structure generated by CTW.
Callback	NULL	The user callback function that will be invoked each time the scan is complete as well as when newly processed data is available. Can be set to NULL if not used.

ISDE Property	Value	Description
Data Processing Option	Default Processing (recommended), No Processing (tuning only) Default: Default Processing	Can be used to specify if scans should start after the data from the previous one is completed, if auto-calibration should be run between scans. Use the Default Processing unless you are tuning.
Write Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Write interrupt priority selection.
Read Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Read interrupt priority selection.
End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	End interrupt priority selection.

Configuration Settings for the DTC HAL Module on r_dtc EVENT CTSU END

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build

ISDE Property	Value	Description
Software Start	Enabled, Disabled Default: Enabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table
Name	g_transfer0	Module name
Mode	Block	Mode selection
Transfer Size	2 Bytes	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode)	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	1	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	1	Number of blocks selection
Activation Source (Must enable IRQ)	Event CTSU END	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Capacitive Touch Button Framework Module Clock Configuration

The CTSU Button framework uses the same configuration as the CTSU HAL driver: Set the CTSU clock speed to match the clock speed selected in the capacitive touch tuning tool Capacitive Touch Workbench for Renesas Synergy.

Capacitive Touch Button Framework Module Pin Configuration

The CTSU framework uses the same configuration as the CTSU HAL driver: Set the pins as touch sensor pins TSxx and enable the TSCAP function as required by the external device. The first table below illustrates the method for selecting the pins within the SSP configuration window and the following table illustrates an example selection for the I2C pins.

NOTE: The Operation Mode selection mode determines what peripheral signals are available and thus what MCU pins are required.

Pin Selection Sequence for Capacitive Touch Sensing Unit (CTSU)

Resource	ISDE Tab	Pin selection Sequence
CTSU	**Pins**	Select **Peripherals > Input: CTSU > CTSU0**

Pin Configuration Settings for CTSU

Pin Configuration Property	Value	Description
Operation Mode	Disabled, Enabled (Default: Disabled)	Select Enabled as the Operation Mode for CTSU
TS00 to TS11	None, Pn, Pm (Default: P204, P206, P207, None, P408, P409, None, None, None, P414, P415)	TS Pins
TSCAP	None, Pn, Pm (Default: None)	TSCAP Pin

NOTE: The above example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.1.18.6 Using the Capacitive Touch Button Framework Module in an Application

Before using the Capacitive Touch Button framework, add the framework to your project in ISDE, make sure that the configuration data from CTW is added to the project and update the Properties of the framework including the number of buttons, the name of the configuration structure generated by CTW and the callback to be used. Once this is accomplished the typical steps in using the Capacitive Touch Button Framework module in an application are:

1) Initialize the Capacitive Touch Button Framework module using the open API.
2) Detect and Handle Touch Button events with the Callback Function.
3) Disable the Callback for a Button using the disable API (Optional).
4) Enable the Callback for a Button using the enable API (Optional).
5) Close the Capacitive Touch Button framework module using the close API.

These common steps are illustrated in a typical operational flow diagram in the figure below:

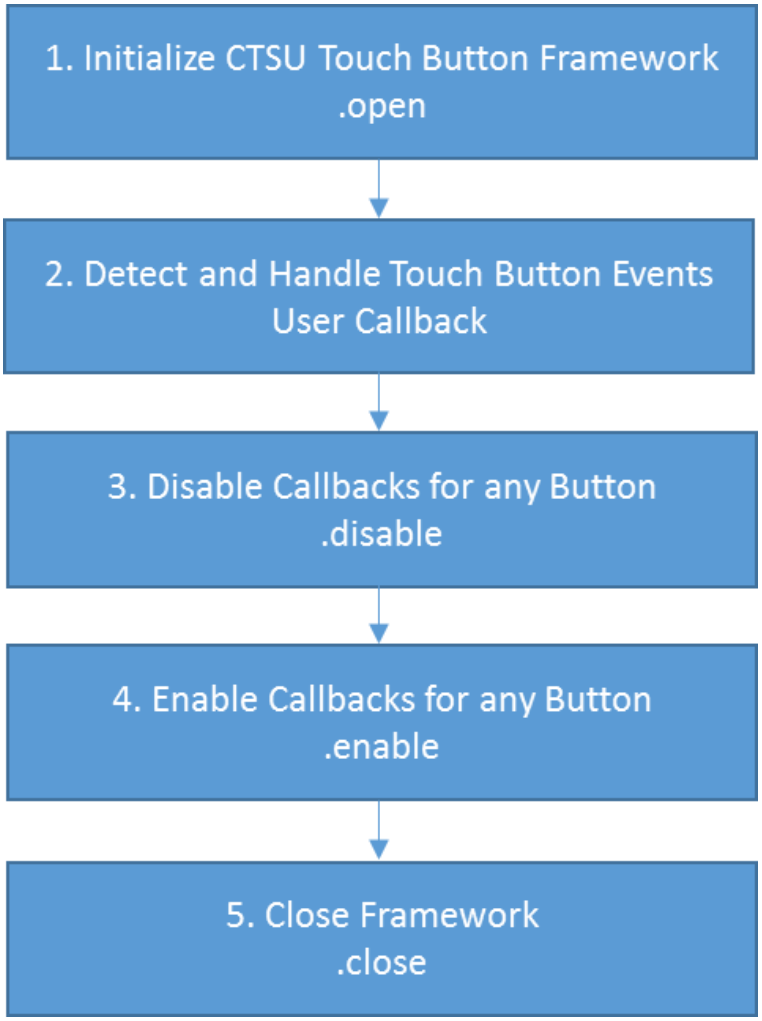


Figure 179: Flow Diagram of a Typical Capacitive Touch Button Framework Module Application

4.1.19 Capacitive Touch Slider Framework

The Capacitive Touch Slider Framework module provides high-level ThreadX-aware APIs for Capacitive Touch slider and wheel applications and is implemented on sf_touch_ctsu_slider. The Capacitive Touch Slider Framework module uses the CTSU on the Synergy MCU. This framework is designed to be used with the configuration data generated by the Capacitive Touch Workbench for Renesas Synergy (CTW) tool. Sliders, wheels, and channels used, are configured in the tool. A user-defined callback can be created to process data when it is available from the hardware after a scan.

This document is divided into the following sections which can be read independently (by a more experienced Synergy developer) or in series (by developers new to the Synergy Platform.)

4.1.19.1 Capacitive Touch Slider Framework Module Features

The Capacitive Touch Slider Framework module is used to interpret the CTSU data for all the slider configurations initialized by the system. Key features include the following:

- Support for slider and wheel
- Support for multiple instances of sliders and wheels
- Callbacks are used to simplify touch processing
  - Use configuration data generated from CTW
  - When a state changes a callback is generated
  - Callbacks are associated with each slider and include the event and position
  - Callbacks are called in the order they appear in the configuration table
- Supports multi-touch detection (can be optionally disabled at build-time)
- Supports S124, S128, S3A6, S3A7, S5D5, S5D9 and S7G2

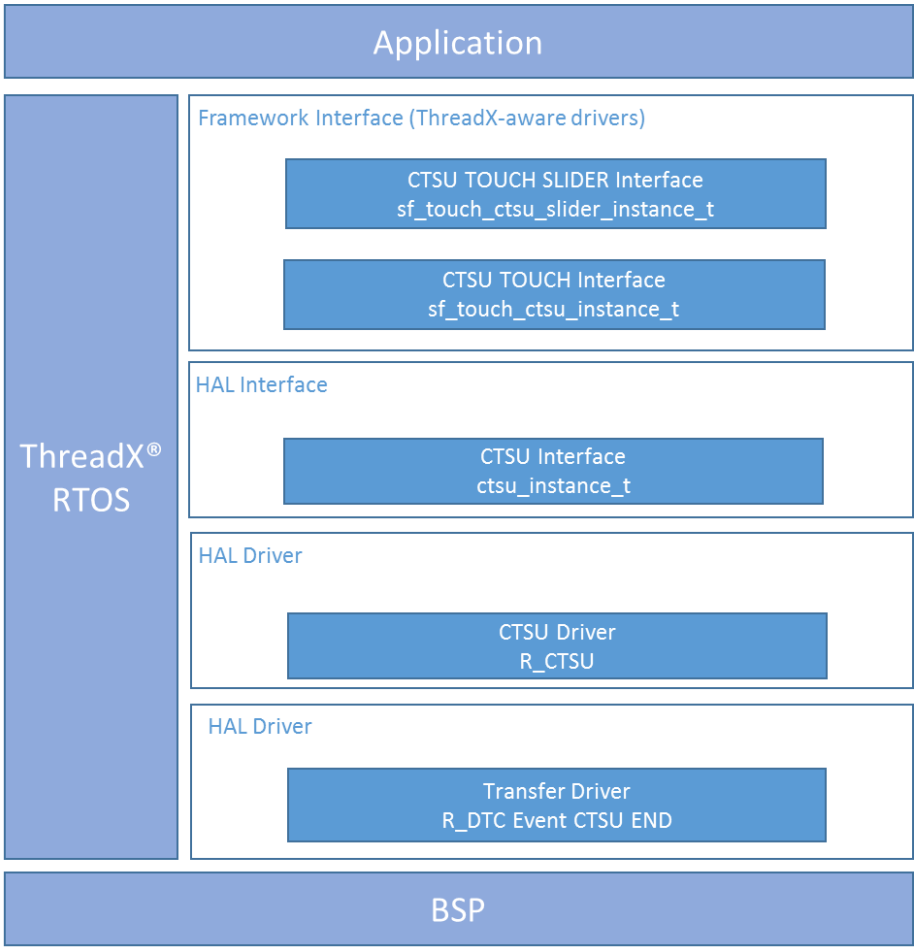


Figure 180: Capacitive Touch Slider Framework Organization, Options and Stack Implementations

4.1.19.2 Capacitive Touch Slider Framework Module APIs Overview

The Capacitive Touch Slider Framework defines APIs for open, close, enable and disable. A complete list of the available APIs, an example API call, and a short description of each can be found in the table below. A table of status return values follows.

Capacitive Touch Slider Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_touch_slider0.p_api->open(g_sf_touch_slider0.p_ctrl, g_sf_touch_slider0.p_cfg); Initialize the Touch CTSU Button Framework; configures the lower level hardware and registers callback functions for all the buttons.
enable	g_sf_touch_slider0.p_api->enable(g_sf_touch_slider0.p_ctrl, slider_id); Enable callback notification for a configured button.
disable	g_sf_touch_slider0.p_api->disable(g_sf_touch_slider0.p_ctrl, slider_id); Disable callback notification for a configured button.
close	g_sf_touch_slider0.p_api->close(g_sf_touch_slider0.p_ctrl); Close the Touch CTSU Button Framework; Closes the button framework and lower layers if no other modules are using it.
versionGet	g_sf_touch_slider0.p_api->versionGet(&p_version); Get version and store it in provided pointer p_version.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables review the SSP Users Manuals API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_ASSERTION	Assertion error.
SSP_ERR_IN_USE	Framework already in use.

Name	Description
SSP_ERR_NOT_OPEN	Device not open.
SSP_ERR_INTERNAL	Internal error.
SSP_ERR_UNSUPPORTED	Device unsupported.
SSP_ERR_INVALID_ARGUMENT	Invalid argument.

NOTE: Lower level drivers may return Common Error Codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.19.3 Capacitive Touch Slider Framework Module Operational Overview

The Capacitive Touch Slider Framework module is used to interpret the CTSU data for all the slider configurations initialized by the system. It also initializes the CTSU Framework layer. The Capacitive Touch Slider Framework registers a callback with the CTSU Framework layer, which will be called each time processed data is available. The Slider Framework then uses this data (raw values) to determine if a touch or release occurred and, if so, where it occurred. If there is a state change, the Framework calls the callback for each slider, in the order in which they are present in the slider configuration table, with the event and the position. The slider framework executes the callback at the update rate (sf_touch_ctsu configuration update_hz) between the touch and release events. The application code should use these callbacks to track the position along the slider. The slider framework executes the callback at the update rate (update_hz) between the touch and release events. This feature can be optionally disabled at build-time.

Capacitive Touch Slider Framework Module Operational Notes

This framework is designed to be used in conjunction with the configuration data generated by the Capacitive Touch Workbench for Renesas Synergy tool which you can download from the Renesas Synergy Gallery in the SSP Utilities tab.

Capacitive Touch Slider Framework Module Limitations

The Framework only supports capacitive touch sliders/wheels that are laid out for self-capacitance mode of operation. There are no other known limitations for using this module. See the most recent SSP release notes for additional limitations.

4.1.19.4 Including the Capacitive Touch Slider Framework Module in an Application

This section describes how to include the Capacitive Touch Slider Framework module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP based applications.

To add the Capacitive Touch Slider Framework module to an application, simply add it to a thread using the Stacks Selection Sequence given in the table below. (The default name for the Capacitive Touch Slider Framework module is g_sf_touch_slider0. This name can be changed in the associated **Properties** window.)

Capacitive Touch Slider Framework Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_touch_slider0 Capacitive Touch Slider/Wheel Framework on sf_touch_ctsu_slider	**Threads**	**New Stack> Framework> Input> Capacitive Touch Slider Framework on sf_touch_ctsu_slider**

When the Capacitive Touch Slider Framework module on sf_touch_ctsu_slider is added to the thread stack as shown in the figure below, the configurator automatically adds any needed lower level modules. Any drivers that need additional configuration information will have box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower level drivers; these are either optional or recommended (this is indicated in the block with the inclusion of this text.) If the addition of lower level drivers is required, the module description will include "Add" in the text. Clicking on any Pink banded modules will bring up the "New" icon and then display the possible choices.

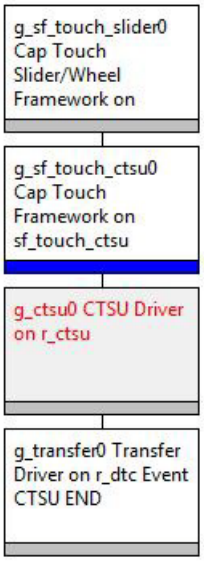


Figure 181: Capacitive Touch Slider Framework Module Stack

4.1.19.5 Configuring the Capacitive Touch Slider Framework Module

The Capacitive Touch Slider Framework module must be configured by you for the desired operation. The SSP configuration window will automatically identify, by highlighting the block in red, any required configuration selections, such as Interrupts or operating modes, which must be configured for lower level modules, for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the **Properties** tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority. This configuration setting is available with the properties window of the associated module. Simply select the indicated module and then view the properties window. The Interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the Interrupt priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the below configuration properties tables, but is easily visible with the ISDE when configuring Interrupt Priority levels.

NOTE: You may want to open your ISDE and create the module and explore the property settings in parallel with looking over the Configuration Table Settings given below. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Capacitive Touch Slider Framework Module on sf_touch_ctsu_slider

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Controls whether to include code for API parameter checking
Number of Sliders/Wheels	1	Number of sliders configured in CTW
Multi touch enable	Enabled, Disabled Default: Enabled	Enables or disables multi-touch detection
Name	g_sf_touch_slider0	Framework name
Slider/Wheel Configuration Structure Name (generated by Workbench)	all_sliders	Name of slider configuration structure generated by CTW
Callback	g_slider_framework_user_callback	Name of callback function created by user application

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Since lower level properties are configured by Capacitive Touch Workbench for Renesas Synergy (CTW), it is best not to make any configuration changes without a detailed knowledge of CTSU operation. The configurable properties for the lower level stack modules are given in the below sections for completeness and as a reference.

NOTE: Most of the property settings for lower level modules are fairly intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the Capacitive Touch Slider Framework Module

Typically, only a small number of settings must be modified from the default for lower level modules. (These are indicated via the red text in the thread stack block.) Notice that some of the configuration properties must be set to a certain value

for proper framework operation and will be locked to prevent user modification. The tables below identify all the settings within the properties section for the module.

Configuration Settings for the Capacitive Touch Framework on sf_touch_ctsu

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter error checking.
Name	g_sf_touch_ctsu0	Module name.
Thread Priority	3	User determined based on priority of other threads in system. Recommend high priority.
Update Hz	50	Rate at which the CTSU hardware scans should occur. The maximum scan rate should be less than the tick rate set for ThreadX. The total time to complete a hardware scan is determined by the number of channels used. Each channel takes approximately 600 usec to complete a scan. Thus if 10 channels are used for 10 buttons in self-capacitance mode, the total hardware scan time is ~6 milliseconds or 166 Hz. Similarly, if 7 channels are used for 5 buttons in mutual-capacitance mode (5x2 layout), the total hardware scan time is ~4.2 milliseconds or 250 Hz. The software processing time is additional and will vary depending on the core clock and the software filter depth used by the CTSU. Thus, the maximum scan rate should be lesser than the time required to scan and process all the channels used in the configuration and lesser than the RTOS tick rate.
Callback	NULL	Name of user callback that will be called when each scan is complete and also when new processed data is available. Can be NULL if not used.

Configuration Settings for the CTSU HAL Module on r_ctsu

ISDE Property	Value	Description
Parameter Checking Enable	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Offset Adjustment	Enabled, Disabled Default: Enabled	Used to enable or disable offset adjustment. Leave as enabled unless you are tuning the board. (Rate is determined by *Runtime rate of tuning of sensor values*.)
Drift Compensation	Enabled, Disabled Default: Enabled	Used to enable or disable drift compensation. Leave as enabled unless you are tuning the board. Drift compensation allows the baseline values used to determine the presence or absence of a touch to change over time as the board or surface ages or with changes in temperature and environment. Drift compensation rate is determined by *Steady state drift compensation rate*, *Startup drift compensation rate*, and *Channel release compensation rate*.
Drift Compensation Method (valid only if Drift Compensation is enabled above)	Alternate Method 1	Drift Compensation algorithms provided. Only Alternate 1 is currently supported. Other options may be supported in future releases. Drift compensation method Alternate 1 allows for different rates of drift compensation for the following events: steady state, startup, and channel release.
Steady state drift compensation rate, drift compensation will be applied per n scans	500	Determines the rate of drift compensation (applied every n scans) when the channel is in steady state. (Dependent on the scan rate.)
Startup drift compensation rate (Should be less than the steady state drift compensation)	5	Determines the rate of drift compensation (applied every n scans) when the driver is performing its initialization. (Should be less than the *Steady state drift compensation rate*.)

ISDE Property	Value	Description
Channel release compensation rate (Should be less than the steady state drift compensation rate)	500	Determines the rate of drift compensation (applied every n scans) when a channel transitions from pressed to released. (Should be less than or equal to the *Steady state drift compensation rate*.)
Default filter depth (used in sensor count filter provided by driver)	1	Used in the software sensor count filter provided by driver. This default filter is a modification of the relatively common "leaky integrator" software filter. A depth of 4 equates to an approximate filter constant of 16. When the input is a constant value of 16 or greater, the filter output will reach 68-69% of the constant input value after 16 cycles/iterations of the filter.
Runtime rate of tuning of sensor values (if drift compensation is used this value is overrriden to be twice the rate of the steady state drift compensation)	800	Rate of offset adjustment. (If drift compensation is used this value is set to twice the rate of the steady state drift compensation.)
Perform auto-tune and drift compensation only when all channels are untouched	True, False Default: True	Perform auto-tune and drift compensation only when all channels are untouched.
Max. active channels	1	Specify the maximum number of channels that are to be used by the application. In Self Capacitance Mode, this is the number of channels used. In Mutual Capacitance Mode, this is the multiplication result of the Rx and Tx channels. For example: 4 Rx and 3 Tx channels = 12 Maximum Active Channels.
Name	g_ctsu0	Module name.
CTSU configuration used	g_ctsu0_config	Name of the configuration structure generated by CTW.

ISDE Property	Value	Description
Callback	NULL	The user callback function that will be invoked each time the scan is complete as well as when newly processed data is available. Can be set to NULL if not used.
Data Processing Option	Default Processing (recommended), No Processing (tuning only) Default: Default Processing	Can be used to specify if scans should start after the data from the previous one is completed, if auto-calibration should be run between scans etc. Use the Default Processing unless you are tuning.
Write Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Write interrupt priority selection.
Read Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Read interrupt priority selection.
End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	End interrupt priority selection.

Configuration Settings for the DTC HAL Module on r_dtc EVENT CTSU END

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Enabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table
Name	g_transfer0	Module name
Mode	Block	Mode selection
Transfer Size	2 Bytes	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	1	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	1	Number of blocks selection
Activation Source (Must enable IRQ)	Event CTSU END	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection

ISDE Property	Value	Description
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Capacitive Touch Slider Framework Module Clock Configuration

The CTSU Slider framework uses the same configuration as the CTSU HAL driver. Set the CTSU clock speed to match the clock speed selected in the capacitive touch tuning tool.

Capacitive Touch Slider Framework Module Pin Configuration

The CTSU framework uses the same configuration as the CTSU HAL driver. Set the pins as touch sensor pins TSxx and enable the TSCAP function as required by the external device. The first table below illustrates the method for selecting the pins within the SSP configuration window and the following table illustrates an example selection for the CTSU pins.

NOTE: The Operation Mode selection mode determines what peripheral signals are available and thus what MCU pins are required.

Pin Selection Sequence for Capacitive Touch Sensing Unit (CTSU)

Resource	ISDE Tab	Pin selection Sequence
CTSU	Pins	Select **Peripherals > CTSU> Cap_Touch_Sensing_Unit**

Pin Configuration Settings for CTSU

Pin Configuration Property	Value	Description
Operation Mode	Disabled, Enabled (Default: Disabled)	Select Enabled as the Operation Mode for CTSU

Pin Configuration Property	Value	Description
TS00 to TS11	None, Pn, Pm (Default: None)	TS Pins
TSCAP	None, P205, P203 (Default: P205)	TSCAP Pin

NOTE: The above example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.1.19.6 Using the Capacitive Touch Slider Framework Module in an Application

Before using the Capacitive Touch Slider framework, add the framework to your project in ISDE, make sure that the configuration data from CTW is added to the project and update the Properties of the framework including the number of sliders, the name of the configuration structure generated by CTW and the callback to be used. Once this is accomplished the typical steps in using the Capacitive Touch Slider Framework module in an application are:

1) Initialize the Capacitive Touch Slider Framework module using the open API.
2) Detect and Handle Touch Slider events through the Callback Function.
3) Disable Callback for the Slider with the disable API (optional).
4) Enable Callback for the Slider with the enable API (optional).
5) Close Capacitive Touch Button Framework module using the close API.

These common steps are illustrated in a typical operational flow diagram in the figure below:

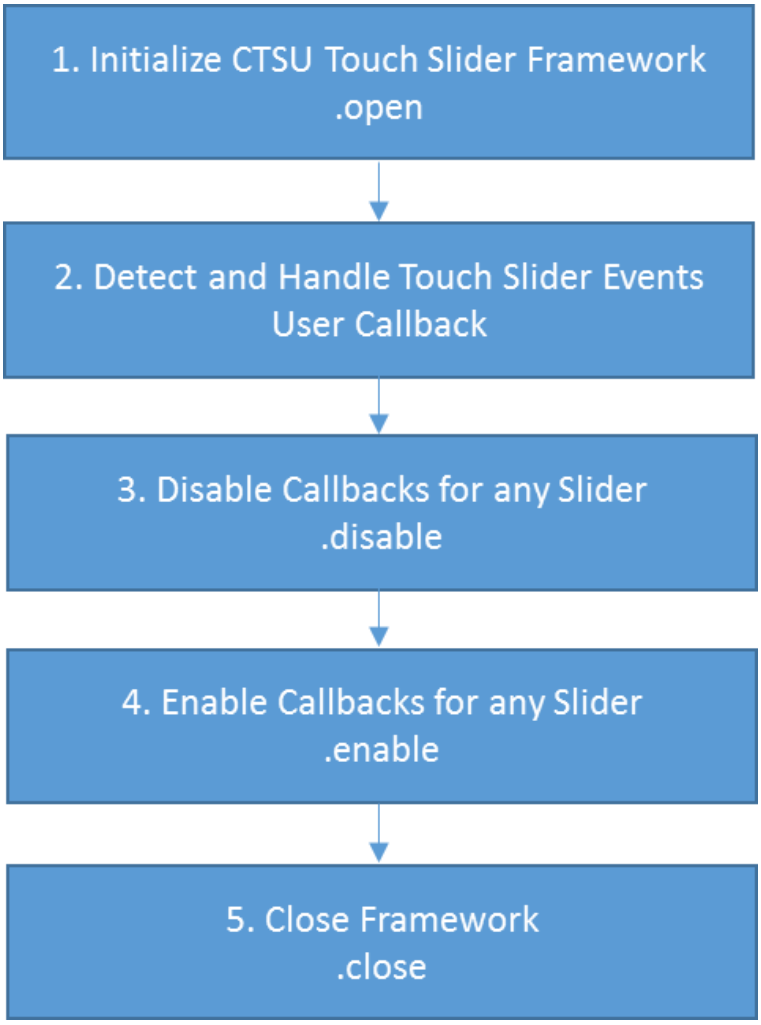


Figure 182: Flow Diagram of a Typical Capacitive Touch Slider Framework Application

4.1.20 External IRQ Framework

The External IRQ Framework provides high-level APIs for applications using the external pin interrupts with the ThreadX RTOS. The Framework is implemented on sf_external_irq and supports the external IRQ pins on the Synergy microcontroller. A callback function (sf_external_irq_callback) is available that will be called from the interrupt service routine (ISR) each time the IRQn triggers.

4.1.20.1 External IRQ Framework Module Features

- Responds to external interrupt inputs
- RTOS aware implementation using an internal semaphore for thread synchronization
  - Can signal internal threads

  - Can trigger transfers via the Event Link Controller (ELC)
- Uses the port pins available on Synergy MCUs
  - Pins may differ between MCUs so refer to MCU User's Manuals for specifics
- Supports several hardware features such as
  - Channel selection
  - Trigger conditions
  - Digital filtering
  - Auto start

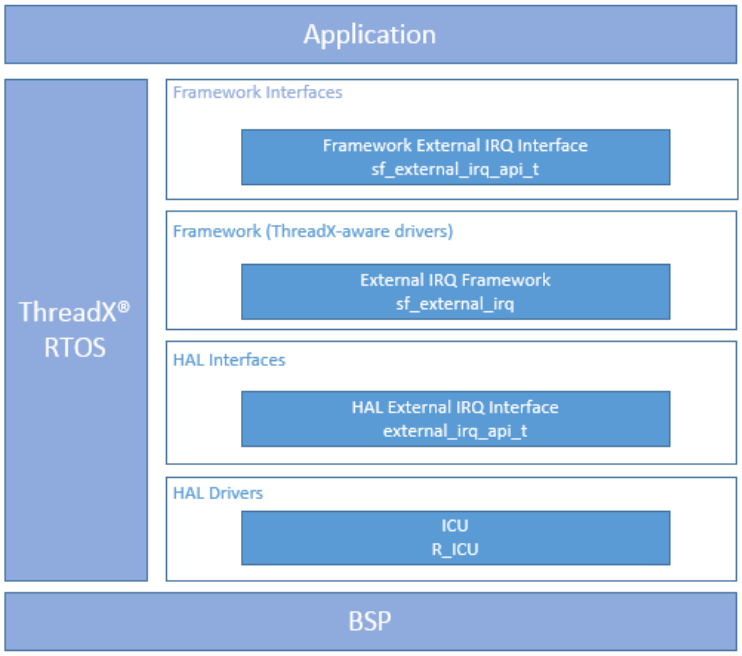


Figure 183: External IRQ Framework Organization, Options, and Stack Implementations

4.1.20.2 External IRQ Framework Module APIs Overview

The External IRQ framework module defines APIs for opening, waiting or closing the module. A complete list of the available APIs, an example API call, and a short description of each can be found in the table below. A table of status return values follows.

External IRQ Framework API Summary

Function Name	Example API Call and Description
open	g_sf_external_irq.p_api->open(g_sf_external_irq.p_ctrl, g_sf_external_irq.p_cfg); Acquire mutex, then handle driver initialization at the HAL layer.
wait	g_sf_external_irq.p_api->wait(g_sf_external_irq.p_ctrl, TX_WAIT_FOREVER); Wait for the next external interrupt expiration, then return.
versionGet	g_sf_external_irq.p_api->versionGet(&version); Retrieve the API version and store it in the version pointer.
close	g_sf_external_irq.p_api->close(g_sf_external_irq.p_ctrl); Release channel mutex and close channel at HAL layer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manuals API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_ASSERTION	Assertion error.
SSP_ERR_IN_USE	Device in use.
SSP_ERR_NOT_OPEN	Device unopened.
SSP_ERR_TIMEOUT	Timeout error.
SSP_ERR_WAIT_ABORTED	Suspension aborted.
SSP_ERR_UNSUPPORTED	Function unsupported by the HAL driver.

NOTE: Lower level drivers may return Common Error Codes. Refer to the API Reference section for the associated module for a definition of all relevant status return values.

4.1.20.3 External IRQ Framework Module Operational Overview

The External IRQ framework is a set of ThreadX aware framework APIs. With the External IRQ Framework external inputs such as switches can signal, via an internal semaphore, threads or trigger transfers via the Event Link Controller (ELC). Both the External IRQ framework module and the External IRQ HAL module need to be configured for proper operation. The HAL configuration settings allow control over hardware options such as triggering level and digital filtering settings.

External IRQ Framework Module Important Operational Notes and Limitations

- Refer to the Datasheet for the Synergy device to be programmed to find the port pins which support the external interrupt functions and to obtain the External IRQ number for a given port pin.
- The External IRQ number corresponds to the Channel setting in the ISDE Properties window for the External IRQ driver.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.20.4 Including the External IRQ Framework Module in an Application

This section describes how to include the External IRQ Framework in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP based applications.

To add the External IRQ Framework to your application simply add it to a project Thread using the Stacks Selection Sequence given in the table below. (The default name for the External IRQ Framework is g_sf_cexternal_irq0 and this is shown in the below table. This name can be changed in the associated **Properties** window).

External IRQ Framework Configuration Settings

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_external_irq0 External IRQ Framework on sf_external_irq	**Threads**	**New Stack> Framework> Input> External IRQ Framework on sf_external_irq**

When the External IRQ Framework on sf_external_irq is added to the thread stack as shown in the figure below, the configurator automatically adds any needed lower level modules. Any drivers that need additional configuration information will have box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

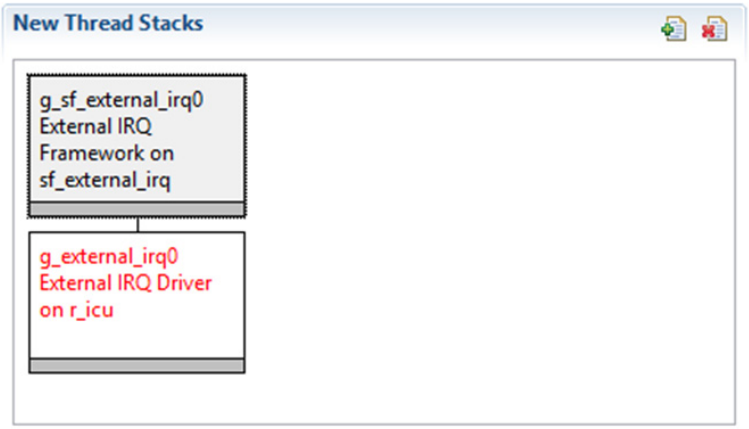


Figure 184: External IRQ Framework Thread Stack

4.1.20.5 Configuring the External IRQ Framework Module

The External IRQ framework module must be configured by you for the desired operation. The SSP configuration window will automatically identify, by highlighting the block in red, any required configuration selections, such as Interrupts or operating modes, which must be configured for lower level modules, for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the **Properties** tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority. This configuration setting is available with the **Properties** window of the associated module. Simply select the indicated module and then view the **Properties** window. The Interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the Interrupt priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring Interrupt Priority levels.

NOTE: You may want to open your ISDE and create the module and explore the property settings in parallel with looking over the Configuration Table Settings given below. This will help orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for External IRQ Framework on sf_external_irq

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Parameter selection.
Name	g_sf_external_irq0	Framework name.

ISDE Property	Value	Description
Event	Semaphore Put, None (Default: Semaphore Put)	Event selection.

NOTE: The above value examples and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

In some cases, values other than the defaults for stack modules can be desirable. For example, it might be useful to select the triggering setting or digital filtering value. The configurable properties for the lower level stack modules are given in the below sections for completeness and as a reference.

NOTE: Most of the property settings for modules are fairly intuitive and can usually be determined by inspection of the associated properties window from the SSP configurator.

4.1.20.6 Configuration Settings for the External IRQ Framework Module Low Level Drivers

Typically, only a small number of settings must be modified from the default for lower level modules. (These are indicated with red text in the thread stack block.) Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The tables below identify all the settings within the properties section for the module.

Configuration Settings for External IRQ Driver on r_icu

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Parameter selection.
ICU IRQ0	Priority 0 (highest), 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 (lowest, not valid if using Thread X), Disabled (Default: Disabled)	ICU selection.
Name	g_external_irq0	Driver name.
Channel	0	Specifies the hardware IRQ channel used.

ISDE Property	Setting	Description
Trigger	Falling, Rising, Both Edges, Low Level (Default: Rising)	Trigger selection.
Digital Filtering	Enabled, Disabled (Default: Disabled)	Digital filter enable/disable.
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCLK/1, PLCK/8, PLCK/32, PCLK/64 (Default: PCKL/64)	Sets noise filter sampling period.
Interrupt enabled after initialization	True, False (Default: True)	Interrupt enable selection.
Callback	NULL	Callback selection.

NOTE: The preceding value examples and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

4.1.20.7 Using External IRQ Framework Module in an Application

The typical steps in using the External IRQ framework module in an application are:

1) Open the External IRQ framework module with the open API
2) Wait for an interrupt using the wait API
3) Process External IRQ event
4) Close the module using the close API

The above common steps are illustrated in a typical operational flow diagram in the figure below.

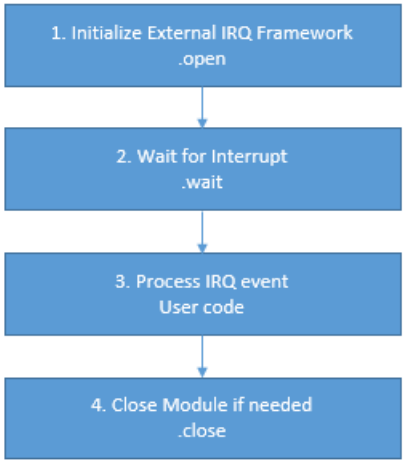


Figure 185: Flow Diagram of a Typical External IRQ Framework Module Application

4.1.21 I2C Framework

The I2C Framework module provides high-level ThreadX®-aware APIs for I2C Framework applications and is implemented on sf_i2c. The I2C HAL module configures the I2C peripheral to enable serial communication to be used by the framework. The I2C Framework module uses the I2C and SCI peripherals on the Synergy MCU.

4.1.21.1 I2C Framework Module Features

- ThreadX-aware framework
- Handles integration and synchronization of multiple I2C peripherals on the I2C bus
- Provides a single interface to access both SCI I2C and RIIC drivers
- The I2C framework module configures I2C communication in master mode
- The I2C framework module supports three data rates: 100 kbps, 400 kbps, and 1 Mbps
- The I2C framework module supports both 7-bit addressing and 10-bit addressing
- The I2C framework module also provides support for callbacks internally. User defined callback is not used. The callback functions are called with the following events i2c_event_t:
  - Transfer aborted
  - Transmit complete
  - Receive complete
- The callback structure i2c_callback_args_t also provides the number of bytes that were sent or received
- Implemented by:
  - Simple I2C on SCI
  - RIIC

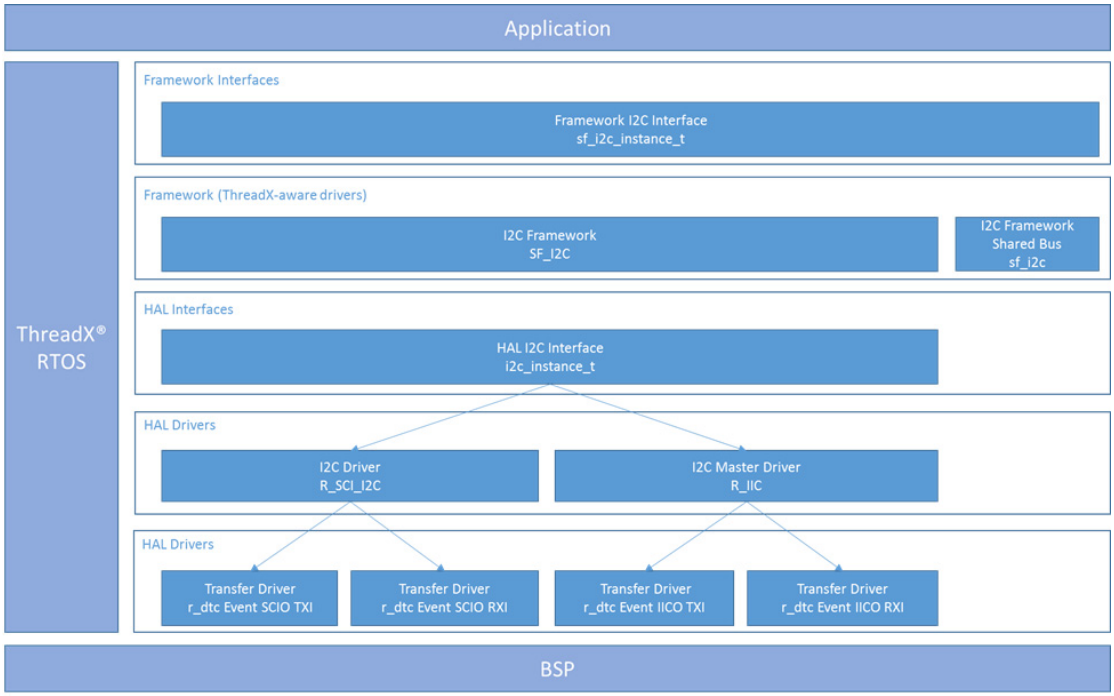


Figure 186:I2C Framework Module Block Diagram

4.1.21.2 I2C Framework Module APIs Overview

The I2C Framework interface defines APIs for opening, closing, reading, writing, locking, unlocking, and resetting the bus using the I2C Framework. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows.

I2C Framework Module API Summary

Function Name	Example API Call and Definition
open	g_sf_i2c_device.p_api->open(g_sf_i2c_device.p_ctrl, g_sf_i2c_device.p_cfg) Opens a designated I2C device on the bus.
close	g_sf_i2c_device.p_api->close (g_sf_i2c_device.p_ctrl); Disables I2C device designated by control handle. Closes the RTOS services used by the bus if no devices are connected to the bus.
read	g_sf_i2c_device.p_api->read (g_sf_i2c_device.p_ctrl, &reg, no_of_bytes_to_read, 0, TX_WAIT_FOREVER); Receives data from I2C device.

Function Name	Example API Call and Definition
write	g_sf_i2c_device.p_api->write (g_sf_i2c_device.p_ctrl, command, no_of_bytes_to_write , false, TX_WAIT_FOREVER); Transmits data to I2C device.
lock	g_sf_i2c_device.p_api->lock (g_sf_i2c_device.p_ctrl); Lock sthe bus for a device. Locking reserves the bus until unlocking and allows several reads and writes without interrupt.
unlock	g_sf_i2c_device.p_api->unlock (g_sf_i2c_device.p_ctrl); Unlocks the bus from a particular device and makes it available for other devices.
reset	g_sf_i2c_device.p_api->reset (g_sf_i2c_device.p_ctrl, TX_NO_WAIT); Aborts any in-progress transfer and forces the I2C peripheral into ready state.
version	g_sf_i2c_device.p_api->version(version); Retrieves the version information using the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the *SSP User's Manual API References* for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	I2C function performed successfully
SSP_ERR_INVALID_MODE	Illegal I2C mode is specified
SSP_ERR_INVALID_CHANNEL	Omitted I2C channel is specified
SSP_ERR_IN_USE	I2C channel has already been opened
SSP_ERR_INVALID_ARGUMENT	Argument is not one of the predefined values
SSP_ERR_INVALID_POINTER	Null pointer(s) is (are) given
SSP_ERR_INTERNAL	Internal error has occurred
SSP_ERR_ASSERTION	A critical assertion has failed
SSP_ERR_NOT_OPEN	Device instance not opened
SSP_ERR_TRANSFER_ABORTED	The data transfer was aborted

Name	Description
SSP_ERR_INVALID_RATE	The requested rate cannot be set
SSP_ERR_TIMEOUT	Timeout error occurs.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.21.3 I2C Framework Module Operational Overview

The I2C Framework module complies with the layered driver architecture of the SSP. It uses the lower-level I2C HAL modules to communicate with I2C peripherals and Controls the I2C-capable peripherals on a Synergy microcontroller, as configured by a user. With the I2C Framework module, one or more I2C buses can be created and multiple I2C peripherals can be connected to each I2C bus. The I2C Framework module APIs use a ThreadX-Mutex to acquire and release the shared bus for I2C Slave devices. Acquire and release are implemented by lock and unlock APIs respectively in the I2C Framework module.

As the I2C framework module configures I2C communication in master mode, this allows the user to:

- Initialize the driver
- Read from a slave device
- Write to a slave device
- Reset the I2C peripheral
- Lock the I2C Bus
- Unlock the I2C Bus

The I2C Framework module works with the Synergy MCU I2C hardware modules; the RIIC and SCI HAL modules. Both I2C modules support the I2C fast-mode with bit rates of up to 400 kbps. The IIC peripheral and the RIIC HAL module support fast-mode plus with 1-Mbps bit-rates. The module supports only master mode for both implementations.

The I2C Framework module uses a bus and device on bus architecture. Every device is linked to the bus to which it will be connected. The user must configure the framework shared-bus and the lower-level I2C HAL layer for each framework device connecting to the bus. The user can add the existing framework shared-bus when configuring multiple devices on the same bus. A common start and stop procedure is used for all I2C data-transfer operations. Only one device is configured to the lower level and the remaining devices perform read or write operations by switching the device address within the framework.

All I2C Framework devices on the same bus must use the same lower-level configuration settings, (for example, the I2C HAL module) except for the slave address and addressing mode. The framework will use the configuration of the first device that it opens in the application to configure the bus; this means that all I2C Framework devices on the same bus must have the same lower-level configuration settings (except for the slave address and addressing mode). If different configurations are used, proper operation cannot be guaranteed.

The I2C Framework supports bus-locking functionality, meaning the bus can be locked for a given peripheral. The locking allows devices to reserve a bus to themselves for a given period of time (between lock and unlock.) This allows devices to complete several reads and writes on the bus without interruption (which is required in some instances.)

I2C Framework Module Important Operational Notes and Limitations

- The closest possible baud rate that can be achieved (less than or equal to the requested rate) at the current PCLKB settings is calculated and used. If a valid clock rate could not be calculated, an error is returned.
- The I2C can trigger the start of other peripherals available from the ELC. See the ELC Module Guide for further information.
- The I2C Framework can support multiple I2C devices on the same bus if the clock rate remains the same for all the devices. That means multiple devices can be opened in the same bus if they are of the same clock rate. If devices have different clock rates, only one device can be opened at a time.
- SDA and SCL output pin type should be n-channel open drain when using I2C on SCI.
- In the I2C Framework configuration, the channel number given to this bus overrides the channel number given in the HAL module.
- Shared bus can be used by multiple slave devices with the respective configuration. The framework also handles mutual exclusion in lock and unlock APIs when multiple devices are using the same I2C channel.
- To configure multiple I2C devices on the same bus, add and configure the following modules for each device connecting to the bus:
  - I2C Framework device module
  - Configure the I2C shared bus module for the first device being configured, then use the same bus for the remaining devices.
  - I2C HAL module
  - DTC module(Optional)
- Lock functionality will be effective for devices from different threads. If multiple devices connected to the bus are from the same thread, the I2C bus will be locked for all devices from that thread. In such cases even if bus is locked all devices from same thread can access the bus.

NOTE: Each I2C Framework device must be configured with a unique name in the ISDE configurator.

NOTE: Provide the same configuration settings for all the devices connected on the same bus (except the slave address and addressing modes.)

The I2C framework module does not currently support the following features:

- The use of any timer other than the GPT
- The use of DMA
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.21.4 Including the I2C Framework Module in an Application

This section describes how to include the I2C Framework module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and

configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.*

To add the I2C framework module to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the I2C framework module is g_sf_i2c_device0. This name can be changed in the associated Properties window.)

I2C Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_i2c_device0 I2C Framework on sf_i2c	Threads	New Stack> Framework> Connectivity> I2C Device Framework on sf_i2c

When the I2C Framework module on sf_i2c is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information will have box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks.

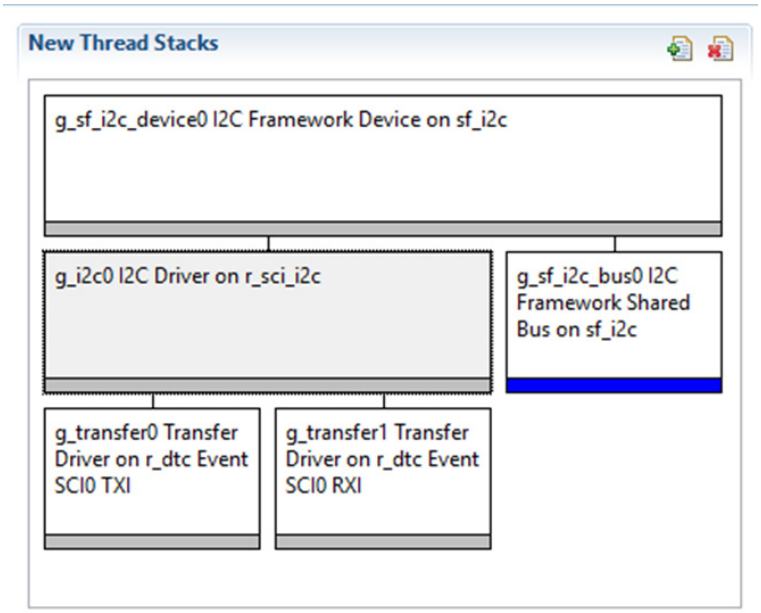


Figure 187:I2C Framework Module Stack

4.1.21.5 Configuring the I2C Framework Module

The I2C framework module must be configured by you for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches

to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority. This configuration setting is available with the Properties window of the associated module. Simply select the indicated module and then view the Properties window. The interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE and create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the I2C Framework Module on sf_i2c

Parameter	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: Enabled	Enable or disable parameter error checking.
Name	g_sf_i2c_device0	Give a name to identify the I2C Framework device. API, Config and Control instances will be created based on this name.

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select different byte ordering or pixel-color format. The configurable properties for the lower-level modules are given in the following sections for completeness and as a reference.

NOTE: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated Properties window with the SSP configurator.

Configuration Settings for the I2C Framework Lower-Level Drivers

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated by the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the I2C Master HAL Module on r_sci_i2c

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.
Name	g_i2c0	Module name.
Channel	0 to 9	Specify the SCI channel to be used with this configuration. SCI has channels as follows: Series S7: 0 1 2 3 4 5 6 7 8 9; Series S3: 0 1 2 3 4 - - - - 9; Series S1: 0 1 - - - - - - - 9.
Rate	Standard, Fast-mode Default: Standard	Standard and Fast.
Slave Address	0x00	Address of the slave device.
Address Mode	7-Bit, 10-Bit Default: 7-Bit	Only 7-bit addresses are currently supported.
SDA Output Delay (nano seconds)	300	SDA output delay in nanoseconds.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Enables bitrate modulation function.

ISDE Property	Value	Description
Callback	NULL	A user callback function can be registered in open. If this callback function is provided, it will be called from the interrupt service routine (ISR) for each of the conditions defined in i2c_event_t. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection.

ISDE Property	Value	Description
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit end interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the I2C Master HAL Module on r_riic

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.
Name	g_i2c0	Module name.
Channel	0 to 9	Specify the SCI channel to be used with this configuration. SCI has channels as follows: Series S7: 0 1 2 3 4 5 6 7 8 9; Series S3: 0 1 2 3 4 - - - - 9; Series S1: 0 1 - - - - - - - 9.
Rate	Standard, Fast-mode Default: Standard	Standard and Fast.
Slave Address	0x00	Address of the slave device.

ISDE Property	Value	Description
Address Mode	7-Bit, 10-Bit Default: 7-Bit	Only 7-bit addresses are currently supported.
SDA Output Delay (nano seconds)	300	SDA output delay in nanoseconds.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Enables bitrate modulation function.
Callback	NULL	A user callback function can be registered in open. If this callback function is provided, it will be called from the interrupt service routine (ISR) for each of the conditions defined in i2c_event_t. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Receive interrupt priority selection.

ISDE Property	Value	Description
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit end interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the I2C Framework Shared Bus on sf_i2c

ISDE Property	Value	Description
Name	g_sf_i2c_bus0	Give a name to identify the bus. This bus name is used in the Framework configuration to link the I2C peripheral to a bus.
I2C Implementation	SCI I2C, RIIC Default: RIIC	Choose any low-level interface to use in the Framework.

ISDE Property	Value	Description
Channel	0-9	SCI or RIIC Channel number to which the device has been connected. In framework, the channel number given in this bus configuration will overrides the channel number given in the HAL driver.

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Enabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	

ISDE Property	Value	Description
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 TXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build

ISDE Property	Value	Description
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection

ISDE Property	Value	Description
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event IIC0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Enabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection

ISDE Property	Value	Description
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event IICI0 TXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid)	ELC software event interrupt priority selection.

Configuration Settings for the Transfer Driver on r_dtc Event IIC0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build

ISDE Property	Value	Description
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event IIC0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection

ISDE Property	Value	Description
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection.

Configuration Settings for the Transfer Driver on r_dtc Event IIC0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection

ISDE Property	Value	Description
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event IIC0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

I2C Framework Module Clock Configuration

The SCI peripheral module uses PCLKB as its clock source. The PCLKB frequency is set by using the SSP configurator clock tab prior to a build, or by using the CGC Interface at run-time. During configuration, the I2C transfer rate is calculated and set internally by the driver, based on the user-selected PCLB rate and the user-selected transfer rate. If the PCLKB is configured in such a manner that the user-selected rate cannot be achieved, an error will be returned when initializing the driver.

I2C Framework Module Pin Configuration

The SCI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the pins.

Note: For some peripherals, the operation-mode selection determines what peripheral signals are available and thus what MCU pins are required.

Pin Selection Sequence for SCI1

Resource	ISDE Tab	Pin Selection Sequence
SCI	Pins	Select Peripherals > SCI1_3_5_7_9 > SCI1

NOTE: The selection sequence assumes SCI1 is the desired hardware target for the driver.

Pin Configuration Settings for the I2C Framework Module on SCI

Pin Configuration Property	Value	Description
Operation Mode	Disabled, Asynchronous UART, Synchronous UART, Simple I2C, Simple SPI, SmartCard Default: Disabled	Select Simple I2C as the Operation Mode for I2C on SCI
RXD1_SCL1_MISO1	None, P212, P708 (Default: None)	SCL Pin
TXD1_SDA1_MOSI1	None, P213, P709 (Default: None)	SDA Pin

NOTE: The example values are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

I2C Framework Module Other Settings

In addition to the SCL and SDA pins, an I2C RESET signal may be required to reset the I2C slave device. If this is the case, the RESET signal can be added using a GPIO pin and must be controlled directly by the application program. The external device reset function is not supported within the r_sci_i2c module.

4.1.21.6 Using the I2C Framework Module in an Application

Once the module has been configured and the files generated and I2C pins are configured, the I2C Framework module is ready to be used in an application. The typical steps in using the I2C module in an application are as follows:

1) Initialize the I2C module using the open API
2) Reset the module using reset API (if needed)
3) Lock the bus using the lock API (if needed)
4) Configure the channel using the write API

5) Unlock the bus using the unlock API (if needed)
6) If a read is desired, lock the bus with the lock API (if needed)
7) The program starts a read from the buffer using the read API
8) Unlock the bus using the unlock API (if needed)
9) The module can be closed if desired using the close API.

These common steps are illustrated in a typical operational flow diagram in the figure below:

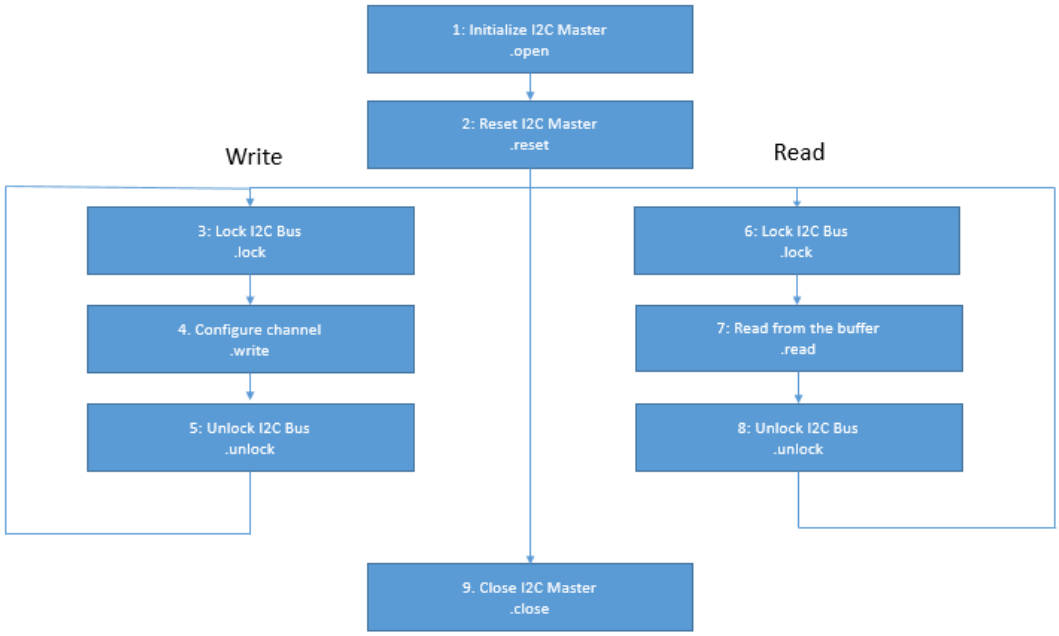


Figure 188: Flow Diagram of a Typical I2C Framework Module Application

4.1.22 JPEG Decode Framework

The JPEG Decode HAL module provides high-level APIs for JPEG decode processing implemented on r_jpeg. (The JPEG Decode HAL module supports the JPEG Codec peripheral.) The JPEG Decode Framework Module is a ThreadX®-aware high-level API for JPEG framework module applications and is implemented on sf_jpeg_decode; it provides thread-safe access to the Synergy JPEG hardware on a Synergy MCU Group. A user-defined callback can be created to detect hardware supported events.

4.1.22.1 JPEG Decode Framework Module Features

- Provides thread-safe access to the Synergy JPEG hardware.
- Supports JPEG decompression using the JPEG Decode HAL module.
- Supports a polling mode that allows an application to wait for the JPEG Decoder to complete.
- Supports an interrupt mode with user-supplied callback functions.

- Configures parameters such as horizontal and vertical subsample values, horizontal stride, decoded pixel format, input and output data format, and color space.
- Obtains the size of the image prior to decoding it.
- Supports putting coded data in an input buffer and an output buffer to store the decoded image frame.
- Supports streaming coded data into the JPEG Decoder module. This feature allows an application to read a coded JPEG image from a file or from a network without buffering the entire image.
- Configures the number of image lines to decode. This feature enables the application to process the decoded image on the fly without buffering the entire frame.
- Supports the input decoded formats YCbCr444, YCbCr422, YCbCr420 and YCbCr411.
- Supports the output formats ARGB8888 and RGB565.
- Returns an error when the JPEG image's size, height, and width don't meet the requirements.
- Supports the wait API function to suspend/resume the thread for synchronizing with the JPEG hardware supported events.

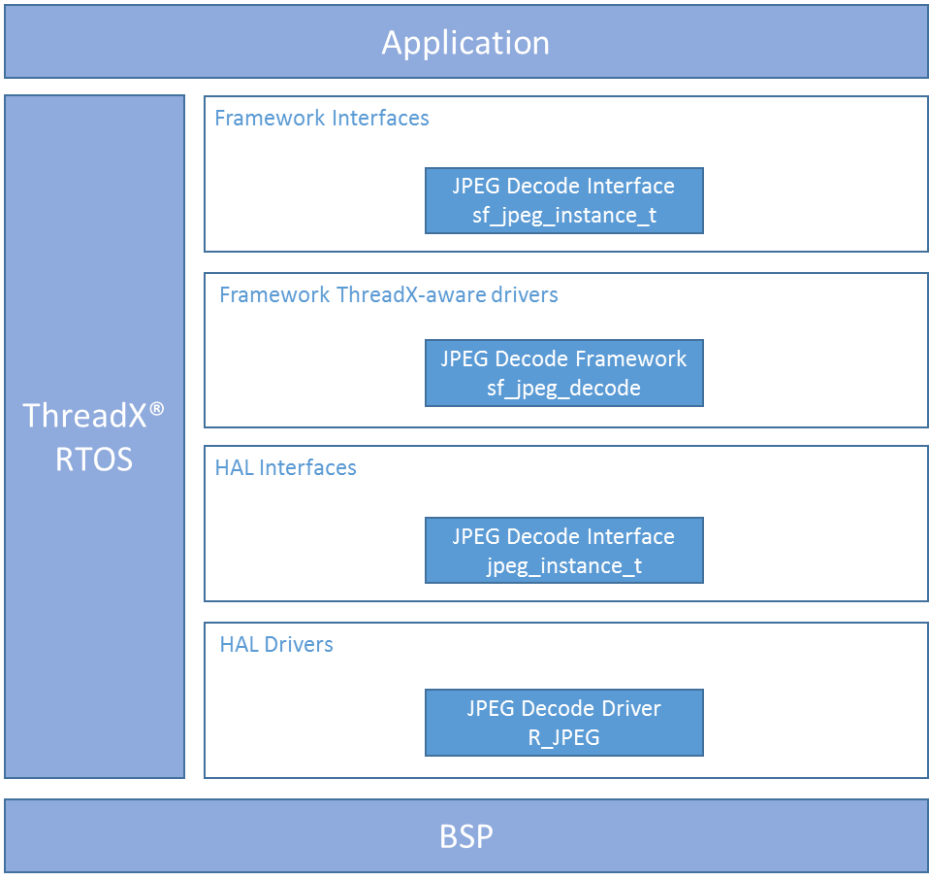


Figure 189: JPEG Decode Framework Module Block Diagram

4.1.22.2 JPEG Decode Framework Module APIs Overview

The JPEG Decode Framework module defines APIs for opening, closing, setting alarms and starting and stopping RTC operations. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

JPEG Decode Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_jpeg_decode.p_api->open(g_sf_jpeg_decode.p_ctrl , g_sf_jpeg_decode.p_cfg); Open the JPEG Decode Framework.
close	g_sf_jpeg_decode.p_api->close(g_sf_jpeg_decode.p_ctrl); Close the JPEG Decode Framework.
inputBufferSet	g_sf_jpeg_decode.p_api->inputBufferSet(g_sf_jpeg_dec ode.p_ctrl, &buffer, size); Feed data into JPEG codec.
outputBufferSet	g_sf_jpeg_decode.p_api->outputBufferSet(g_sf_jpe\g_de code.p_ctrl, &buffer, size); Read processed data from JPEG codec.
linesDecodedGet	g_sf_jpeg_decode.p_api->linesDecodedGet(g_sf_jpeg_d ecode.p_ctrl, &lines); Obtain number of lines decoded.
horizontalStrideSet	g_sf_jpeg_decode.p_api->horizontalStrideSet(g_sf_jpeg_ decode.p_ctrl, stride); Configure stride value.
imageSubsampleSet	g_sf_jpeg_decode.p_api->imageSubsampleSet(g_sf_jpe g_decode.p_ctrl, h_sub, v_sub); Start the calendar counter.
wait	g_sf_jpeg_decode.p_api->wait(g_sf_jpeg_decode.p_ctrl, timeout); Wait for current operation.
statusGet	g_sf_jpeg_decode.p_api->statusGet(g_sf_jpeg_decode.p _ctrl, &status); Obtain JPEG codec status.
versionGet	g_sf_jpeg_decode.p_api->versionGet(&version); Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the *SSP User's Manual*, **API References** for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	JPEG Decode driver is successfully opened.
SSP_ERR_ASSERTION	Assertion error.
SSP_ERR_IN_USE	Module already in use.
SSP_ERR_TIMEOUT	The wait operation times out, the underlying driver did not respond in time.
SSP_ERR_WAIT_ABORTED	System internal error occurred.

NOTE: Lower-level drivers may return common error codes. See *SSP User's Manual*, **API References** for the associated module for a definition of all relevant status return values.

4.1.22.3 JPEG Decode Framework Module Operational Overview

The JPEG Decode Framework module implements the standard JPEG decode operation. It takes the data in an input buffer and applies the defined JPEG decode algorithm to the buffer; the output is then delivered to the defined output buffer location. A wait API function can be used to suspend/resume the thread for synchronization with JPEG hardware supported events.

JPEG Decode Framework Module Important Operational Notes and Limitations

- Start decoding JPEG-encoded data by calling the open API. To open the module, use the JPEG Decode Framework module instance, that includes the API function pointer, the pointer to the control block and static configuration that is generated through the Synergy Project configurator in the e^2 studio for ISDE.
- Stop the JPEG Decode Framework module by calling the close API.
- An input buffer-streaming mode is available when an input-centric function is needed.
- An output buffer-streaming mode is available when an output-centric function is needed.
- Supports RGB565 and ARGBB888 output data-color formats.
- The JPEG Decode Framework module has a status flag in the control block; you can get the current status of the module through statusGet API. The status is also reported through a user-callback function when specific events occur in the module.
- The JPEG Decode Framework module supports buffer-streaming mode for the input buffer in cases when the input buffer is smaller than the source image size. Set the next input frame as an input buffer every time there is a hardware-generated INPUT_PAUSE interrupt.
- The JPEG Decode Framework module supports buffer-streaming mode for the output buffer in cases when the resultant image is larger than the output buffer-size. Read and store data from the output buffer to make space for upcoming data every time there is a hardware generated OUTPUT_PAUSE interrupt.

- The input and output buffers should be 8-bytes aligned for the JPEG Decode Framework module to be successful. Otherwise, APIs will return error codes that indicate unsuccessful execution.
- The JPEG Framework module does not support JPEG-encode processing.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.22.4 Including the JPEG Decode Framework Module in an Application

This section describes how to include the JPEG Decode Framework module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the *SSP User's Manual* to learn how to manage each of these important steps in creating SSP-based applications.

To add the JPEG Decode Framework module to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the JPEG Decode Framework module is g_sf_jpeg_decode0. This name can be changed in the associated **Properties** window.)

RTC Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_jpeg_decode0 JPEG Framework	Threads	New Stack> Framework> Graphics> JPEG Decode Framework on sf_jpeg_decode

When the JPEG Decode Framework module on sf_jpeg_decode is added to the thread stack as shown in the following figure, the configurator automatically adds the needed lower-level drivers. Any drivers that need additional configuration information will be highlighted in Red. The specific settings required can be viewed by hovering the cursor over the highlighted text or suggested in the highlighted stack frame. For this stack, the reported requirements for the JPEG HAL module are to enable the decompression-interrupt priority and the data-transfer interrupt-priority interrupts.

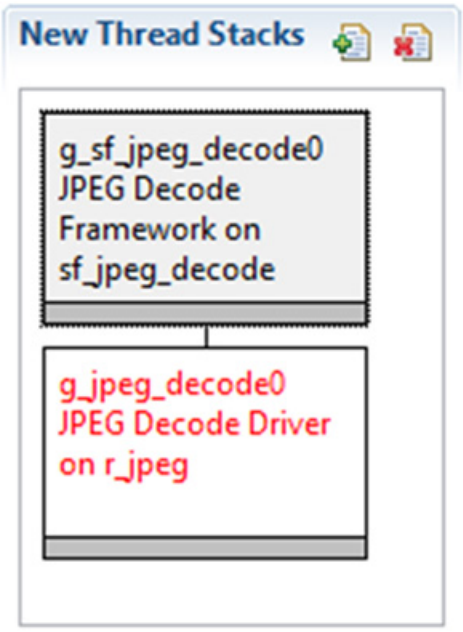


Figure 190: JPEG Decode Framework Module Stack

Decompression Process Interrupt (JEDI)

The JPEG decompression-process interrupt occurs when:

- The current decompression process is successfully completed.
- An error happens in the decompression process.
- Image size and pixel format are successfully read out.

Data Transfer Interrupt (JDTI)

The JPEG data-transfer interrupt occurs when:

- All the JPEG-coded data has successfully completed.
- The number of output image-data lines specified by linesDecodedGet has been transferred.
- The number of input image-data lines specified by inputBufferSet has been transferred.

4.1.22.5 Configuring the JPEG Decode Framework Module

The JPEG Decode Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, that must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are locked and not available for changes and are identified with a 'lock' icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the **Properties** tab within the SSP Configurator and are shown in the following tables for easy reference.

NOTE: You may want to open your ISDE and create the JPEG Decode Framework and explore the property settings in parallel with looking over the configuration table settings in the following table. This will help orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration for the JPEG HAL Module on r_jpeg

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enable or disable the parameter error checking.
Name	g_jpeg_decode0	The name to be used for a JPEG Decode module instance.
Byte Order for Input Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2) (Default: Normal Byte order)	Specify the byte order for input data. The order is swapped as specified in every 8-byte.
Byte Order for Output Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2) (Default: Normal Byte order)	Specify the byte order for output data. The order is swapped as specified in every 8-byte.
Output Data Color Format	Pixel Data RGB565 format, Pixel Data ARGBB888 format (Default: Pixel Data RGB565 format)	Specify the output data format.

ISDE Property	Value	Description
Alpha value to be applied to decoded pixel data (only valid for ARGB8888 format)	255	Specify the alpha value for the output data format (only valid for ARGB8888 format).
Name of user callback function	NULL	Specify the name of user callback function.
Decompression Interrupt Priority	Priority 0 (highest), 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 (lowest, not valid if using Thread X), Disabled (Default: Disabled)	JPEG JEDI Interrupt selection
Data Transfer Interrupt priority	Priority 0 (highest), 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 (lowest, not valid if using Thread X), Disabled (Default: Disabled)	JPEG JDTI Interrupt selection

JPEG Decode Framework Module Clock Configuration

The JPEG Framework module uses the peripheral module clock A (PCLKA) to run the internal logic.

JPEG Decode Framework Module Interrupt Configuration

To enable interrupts, set the priority of the decompression interrupt and the data-transfer interrupt in the Properties window of the JPEG Decode Framework module in the ISDE.

JPEG Decode Framework Module Pin Configuration

The JPEG Decode Framework module does not use any pins.

4.1.22.6 Using the JPEG Decode Framework Module in an Application

The typical steps in using the JPEG Decode Framework module in an application are:

1) Initialize the JPEG Decode peripheral using the open API.
2) Set Image Subsample using the imageSubSampleSet API.
3) Set Horizontal stride using the horizontalStrideSet API.
4) Set output buffer using the outputBufferSet API.
5) Set Input buffer using the inputBufferSet API.
6) Wait for decode to complete with the wait API.
7) Check decode status with the statusGet API.
8) Close the instance with the close API (if needed).

These common steps are illustrated in a typical operational flow diagram in the following figure:

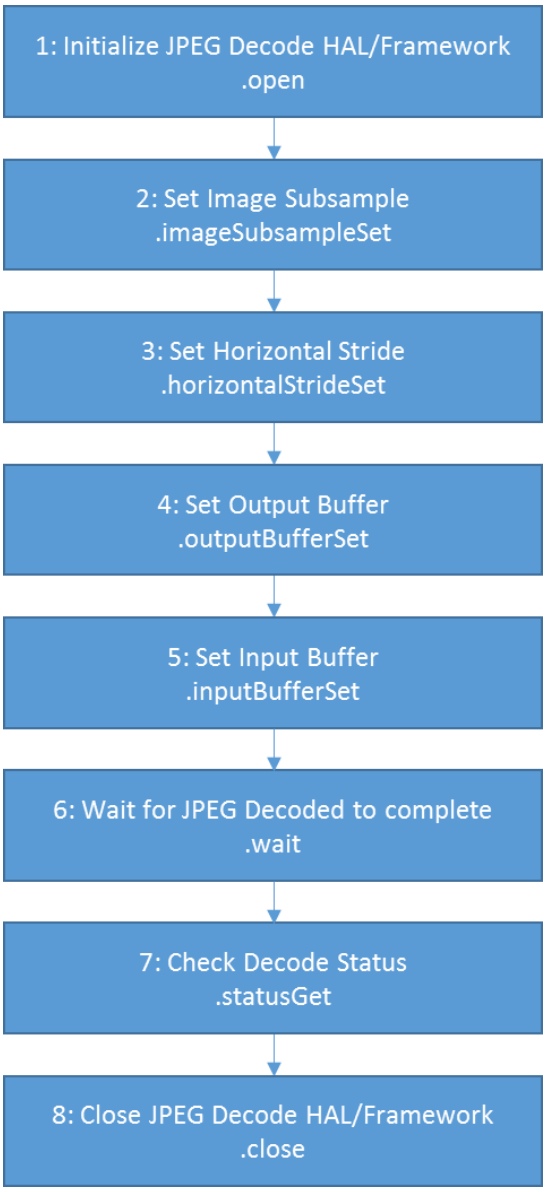


Figure 191: Flow Diagram of a Typical JPEG HAL/Framework Module ApplicationS

4.1.23 Messaging Framework

The Messaging Framework module is implemented on sf_message and provides a lightweight and event-driven framework API for passing messages between threads. The Messaging Framework module allows applications to communicate messages between two or more threads. The framework uses the ThreadX® message-queue primitive for message passing and provides more benefits than the ThreadX RTOS message-queue services alone. The Messaging Framework API is purely a software API and does not access any hardware peripherals. The Messaging Framework

callback is used to allow an event-producer thread and a message-subscriber thread to handshake after the message passing is done.

You can use the messaging tab to either create your own custom event classes, events, and subscribers for the Messaging Framework module or to customize preconfigured events such as the touch event used by the Touch Panel Framework module.

4.1.23.1 Messaging Framework Module Features

The Messaging Framework module supports the following functions:

- Inter-Thread Communication - The framework allows application threads which control disparate devices or manage subsystems to communicate with each other.
- Publishing/Subscribe scheme - The framework design is based on the loosely-coupled messaging paradigm. The design allows multiple threads to listen to an event class. The message producer thread does not need to know who is subscribing to a message for the event class. Subscribers do not need to know who produces the message.
- Message management - The framework supports buffer control blocks to manage each message including flags to control the buffer and a callback function pointer for handshaking.
- Message buffering - The framework manages buffer allocation and release for messaging. An application can make use of the allocated buffer to write a message and discard the message if it is no longer needed.
- Synchronous communication - The framework supports asynchronous messaging by using the ThreadX message-queue but also provides an option to create a handshake between a message producer and a subscriber thread. The handshake is implemented by invoking a user-callback function of the producer thread from a subscriber thread.
- Message formatting - The framework provides a predefined common message header. It also provides some typical payload structure templates as examples.
- Message Priority - The framework can send a high-priority message so that a subscriber thread can retrieve the message prior to other messages which are located in the message queue.

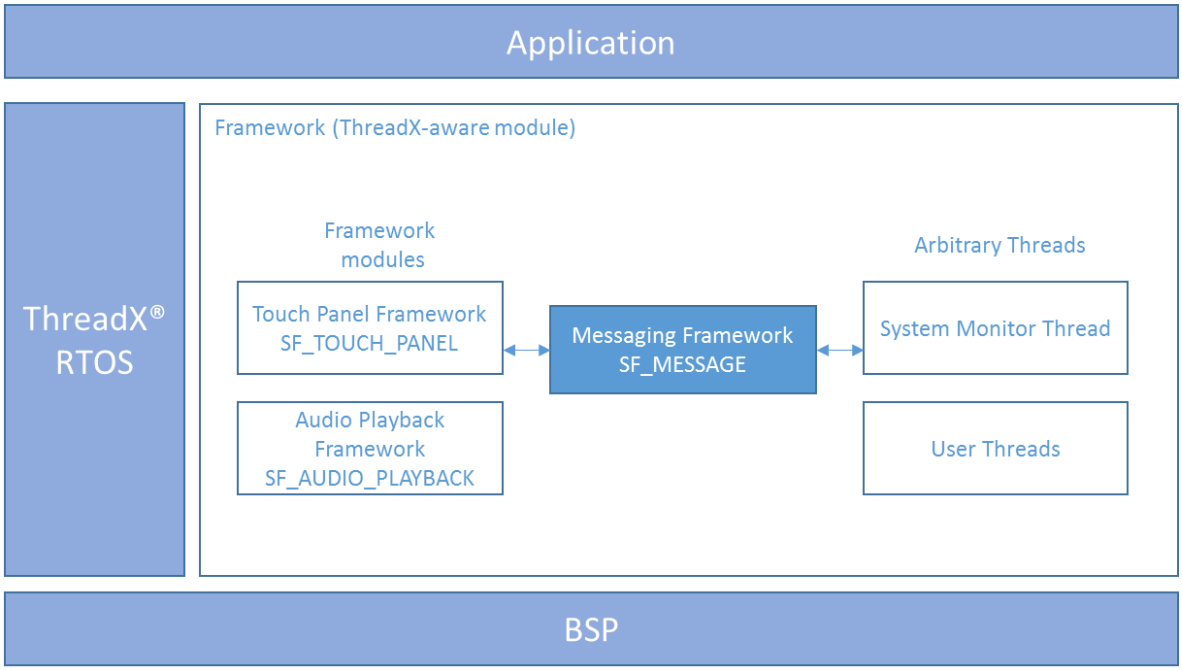


Figure 192: Messaging Framework Module Organization and Use Example

4.1.23.2 Messaging Framework Module APIs Overview

The Messaging Framework module defines APIs for opening and closing the framework, acquiring and releasing buffers, and posting messages to subscribers. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows.

Messaging Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_message.p_api->open (g_sf_message.p_ctrl, g_sf_message.p_cfg); Initialize message framework. Initiate the messaging framework control block, configure the work memory corresponding to the configuration parameters.
close	g_sf_message.p_api->close (g_sf_message.p_ctrl); Finalize message framework.

Function Name	Example API Call and Description
bufferAcquire	g_sf_message.p_api->bufferAcquire (g_sf_message.p_ctrl, &p_buffer, &acquire_cfg, wait_option); Acquire buffer for message passing from the block.
bufferRelease	g_sf_message.p_api->bufferRelease (g_sf_message.p_ctrl, &p_buffer, option); Release buffer obtained from bufferAcquire.
post	g_sf_message.p_api->post (g_sf_message.p_ctrl, (sf_message_header_t *) p_payload, &post_cfg, &err_post, wait_option); Post message to the subscribers.
pend	g_sf_message.p_api->pend (g_sf_message.p_ctrl, &my_queue, &p_buffer, wait_option); Pend message.
versionGet	g_sf_message.p_api->versionGet (&version); Retrieve the API version with the version pointer.

NOTE: For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the *SSP User's Manual***API References** for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API call successful.
SSP_ERR_ASSERTION	Required pointer is NULL.
SSP_ERR_BUFFER_RELEASED	The buffer is released.
SSP_ERR_ILLEGAL_SUBSCRIBER_LISTS	Message subscriber lists is illegal.

Name	Description
SSP_ERR_IN_USE	The messaging framework is in use.
SSP_ERR_INTERNAL	OS service call fails.
SSP_ERR_INVALID_MSG_BUFFER_SIZE	Message buffer size is invalid.
SSP_ERR_INVALID_WORKBUFFER_SIZE	Invalid work buffer size.
SSP_ERR_MESSAGE_QUEUE_EMPTY	Queue is empty. (Timeout occurs before receiving a message if timeout option is specified.)
SSP_ERR_MESSAGE_QUEUE_FULL	Queue is full. (Timeout occurs before sending a message if timeout option is specified.)
SSP_ERR_NO_MORE_BUFFER	No more buffer found in the memory block pool.
SSP_ERR_NO_SUBSCRIBER_FOUND	No subscriber found.
SSP_ERR_NOT_OPEN	Message framework module has yet to be opened.
SSP_ERR_TIMEOUT	OS service call returns timeout.
SSP_ERR_TOO_MANY_BUFFERS	Too many message buffers.

NOTE: Lower-level drivers may return common error codes. Refer to the *SSP User's Manual***API References** for the associated module for a definition of all relevant status return values.

4.1.23.3 Messaging Framework Module Operational Overview

The following figure shows the overview of the messaging data flow between a message producer thread and subscriber thread(s) in the system making use of the Messaging Framework module.

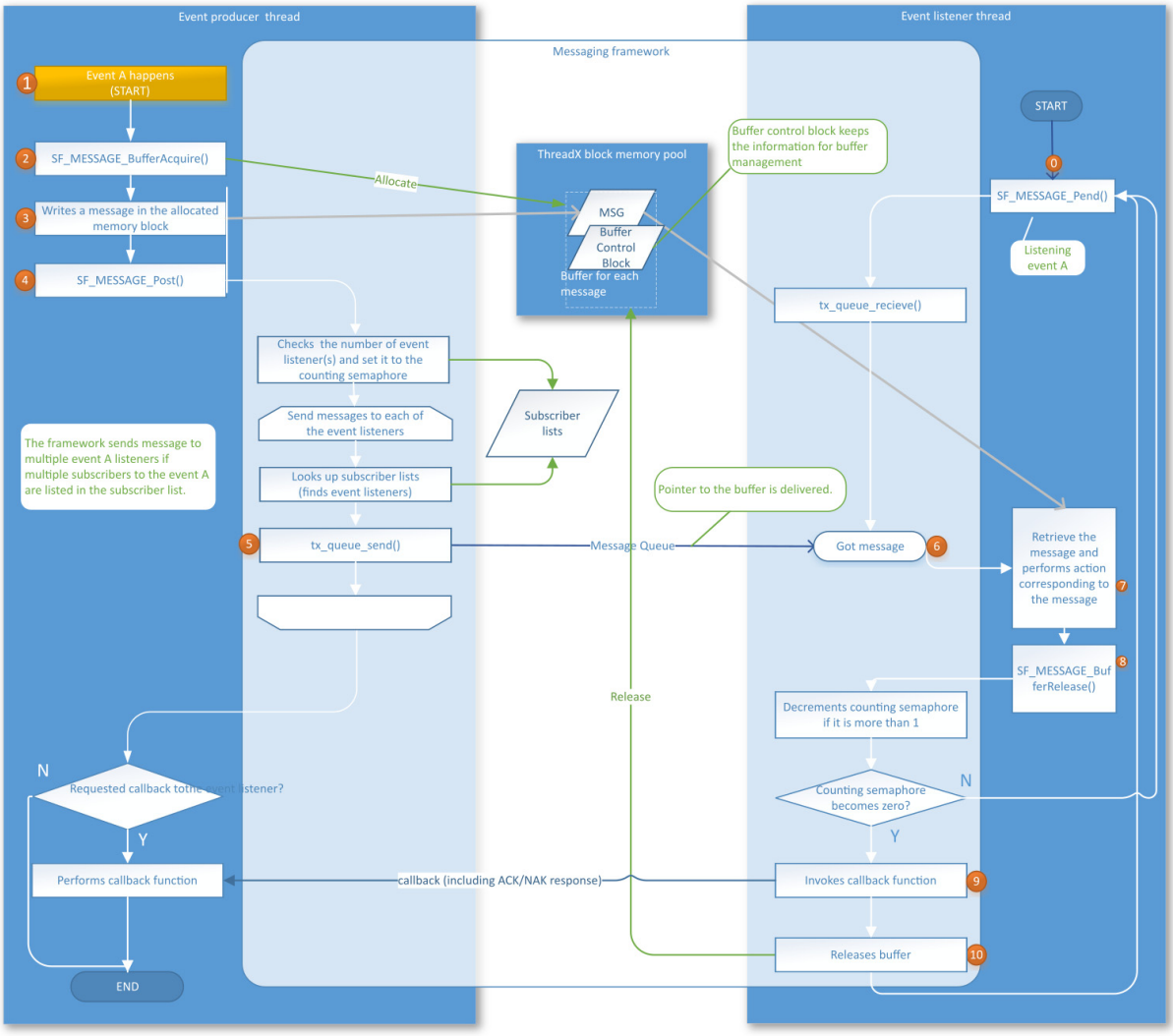


Figure 193: Messaging Framework - Data Flow

The following is a description for each stage of the message passing procedure:

NOTE: A thread in the system has been called using the open API and message subscriber threads have called the pend API to pend on a message for the event class.

1) An event (Event A) happens on a message producer thread.
2) A message producer thread calls bufferAcquire to acquire a buffer from the ThreadX memory pool managed by the Messaging Framework module; bufferAcquire returns the address of allocated buffer.
3) A message producer writes the message to the allocated buffer.
4) A message producer calls post to post the message.
5) The Messaging Framework module looks up the event subscriber list and sends a message to the message queue of the message subscriber threads using the ThreadX message-queue primitive. The framework just sends the pointer to the buffer but does not send the entire message, thereby performing lightweight message passing.
6) The message reaches the message queue of the message subscriber threads and the message subscriber threads return from pend. The API function returns the buffer address where the message is stored to message subscriber threads.
7) The message subscriber threads receive the message and perform an action corresponding to the event.
8) The message subscriber threads call bufferRelease to try to release the allocated buffer for the message. If the message subscriber thread is not the last one subscribing to the message, the framework does not release the buffer as the message has to be kept in the buffer until all subscribers have received the message.
9) The Messaging Framework module invokes a user-callback function which is specified by an event producer thread if the message subscriber thread is the last one in the message subscriber group.
10) The Messaging Framework module releases the buffer if the message subscriber thread is the last one in the message subscriber group. (There is an option 'SF_MESSAGE_ACQUIRE_OPTION_KEEP' for not to release the buffer.)

Messaging Framework Module Message Producer and Subscribers

The Message Framework module is an inter-thread messaging system based on the publish/subscribe model. A message is posted with an event class code by an event producer thread. The message subscriber threads can check for pending messages which subscribe to the event class. Subscribers are registered in the subscriber list, which is referred to by the framework. The subscriber list allows the framework to deliver a message to multiple subscribers.

Every thread which joins the Messaging Framework module system network can send a message, and all threads in the network can listen to the message.

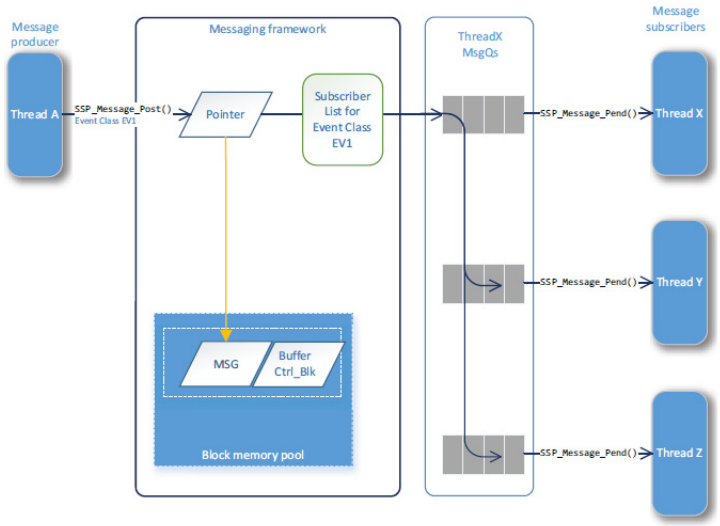


Figure 194: Messaging Framework — Subscribing

Messaging Framework Module Events, Subscribers, and Messages

The event class code is the most important definition for the Messaging module. The Messaging module uses the event class code as the mechanism to connect a message producer with subscribers; the event class code is the class definition of the events which occur in the application. The classification of the event class relies on the user definition, but it is intended to be the group name of the particular events which can occur in a subsystem. For example, you can use the following event classes:

- The 'touch' event class which is part of the touch subsystems. The Touch Event Class is automatically loaded into the event classes' window. This window is available on the **Messaging** tab the Project Configurator when you add the Touch Panel Framework to your Synergy project.
- The 'time' event class is part of the subsystem that manages time-related applications.

The event class code is defined in the sf_message_event_class_t enumeration and has a prefix SF_MESSAGE_EVENT_CLASS_XXX. Since the definition of the event class code is different for each system, the framework does not provide a concrete event class code but instead provides a set of event class codes as examples. (See the Configuring the Messaging Framework Module section.) The maximum number for the event class is 255.

An application can use the event class code as follows:

- The message producer thread sets the event class code to the event_b.class_code bit fields in the sf_message_header_t type common message header before posting the message.
- The message subscriber-thread branches to the event processing corresponding to the event class code which is set to the message header after receiving the message.
- The subscribers for the event class code must be grouped and registered in the subscriber list so that the Message Framework can deliver the message to the subscribers.

The following figure shows how you can configure an event class using the ISDE.

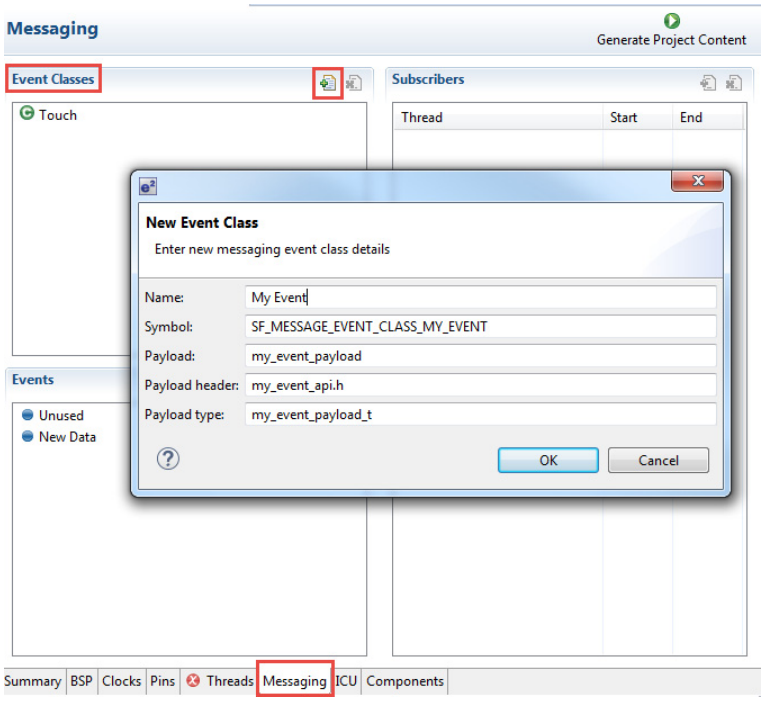


Figure 195: Messaging — ISDE Event Class Configuration

Messaging Framework Module Event Class Instance Number

The event class instance number is used when an application needs to have different event class instances. For example, the audio streaming event class can have instance *N* which represents the streaming channel N. Message subscribers can receive a message only if the event class instance number in the message common header matches to the number it owns.

In other words, messages for which the event class instance number is out of range for the subscriber are filtered out and not delivered to the subscriber even though it is the event class subscriber. The maximum for the event class instance number is 255.

NOTE: The Touch Panel Framework typically only has one instance, and therefore the event class instance number is 0 with the start and end values of the subscriber list set to 0.

An application can use the event class instance number as follows:

- The message producer thread sets the event class instance number to the event_b.class_instance bit fields in the sf_message_header_t type common message header before posting the message.
- Each subscriber instance in the Subscriber List has to specify the range of the event class instance numbers to receive the message (sf_message_subscriber_t:: instance_range.start and sf_message_subscriber_t::instance_range.end).
- If there is no need for multiple instances for an event class, just specify zero to sf_message_header_t::event_b.class_instance, sf_message_subscriber_t::instance_range.start, and sf_message_subscriber_t::instance_range.end in the subscriber instance for the subscriber list.

Messaging Framework Module Event Code

The event code includes the details of the event definition. For instance, the event codes for the audio playback event class are "playback start" and "playback stop." Another example is 'set' or 'get' for the 'time' Event Class. The event code is enumerated in the sf_message_event_t and has a prefix SF_MESSAGE_EVENT_XXX. The definition of the event code relies on the user code as well as the event class code. The framework provides some code as examples. See Configuring the event class code and event code for configuring the event code. The maximum for the event class instance number is 65535.

The Touch Panel Framework uses new data as the only event code.

An application can use the event code as follows:

- The message producer thread sets the event code to the event_b.code bit fields in the sf_message_header_t type common message header before posting the message.
- The message subscriber thread performs an action corresponding to the event code which is set to the message header after receiving the message.

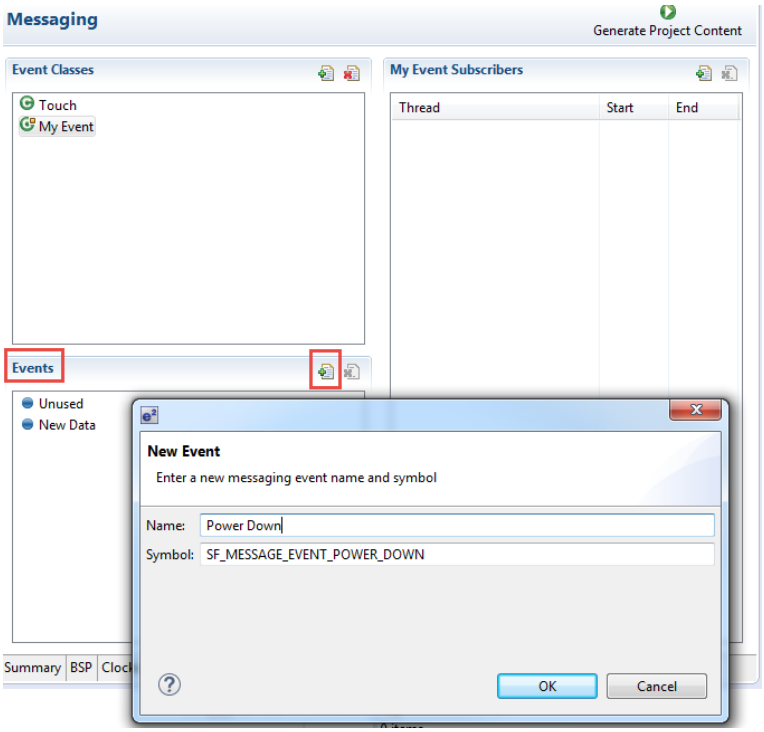


Figure 196: Messaging — ISDE Event Configuration

Messaging Framework Module Subscriber List

The subscriber list is used for message delivery and is looked up by the framework.

The framework starts to look up the message queues of each subscriber thread from the subscriber group listed in the head of the pointer array to the sf_message_subscriber_list_t instance. The important point of the subscriber list is that it is grouped by event class code (event_class).

When the framework looks up the subscriber list in the post API function at runtime, it compares the Event Class code in the message header (sf_message_header_t::event_b.class_code), which is included in the message payload data, with the one in the subscriber group instance (event_class). If there is a match, the Framework goes to the next level to get the message queue instance (sf_message_subscriber_t::queue) until the iterations reach number_of_nodes). If there is no

match, the framework looks up the next subscriber group and continues until encountering a NULL in the pointer array to the sf_message_subscriber_list_t instance.

In the look-up procedure, the subscriber group listed at the head of the Subscriber List gets the highest throughput for messaging, but lower subscriber groups encounter a penalty and get lower throughput for messaging.

The subscriber list is the look-up table for all message subscribers. The subscriber list is configured at compile time. It is statically mapped to the memory and looked up by the framework when the post API function is called. The subscriber list allows the framework to determine message queues to deliver a message to. The subscriber list consists of two structures sf_message_subscriber_list_t and sf_message_subscriber_t as shown in the following figure:

- A queue for a subscriber thread is registered in sf_message_subscriber_t instance.
- The instances above for the same event class code are grouped and listed in a pointer array.
- The pointer array for a subscriber group is registered in a sf_message_subscriber_list_t instance.
- The subscriber list is the pointer array to the subscriber group structures. Subscribers are grouped by the event class code.
- The pointer array must be terminated by NULL.

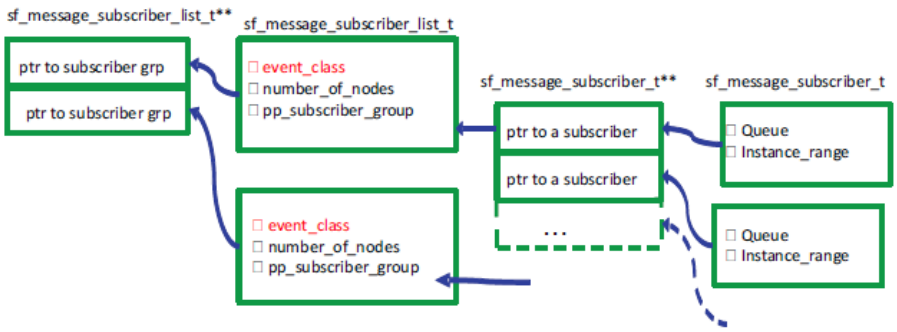


Figure 197: Message Framework — Subscriber List

In the ISDE, you can configure a subscriber grouped by the event class for the thread you named in the **Threads** tab. In the following example, the thread is named "My Thread" in the **Threads** tab. The start and end values reflect the event class instance numbers this thread accepts.

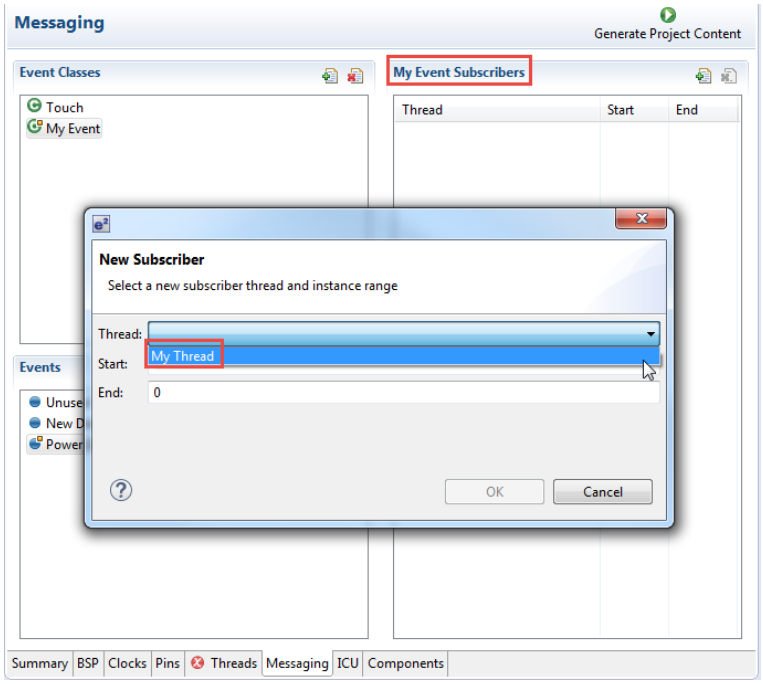


Figure 198: Messaging — ISDE Subscriber Configuration

Messaging Framework Module Message Payload

The message payload is structured data used by the message producer and the message subscribers to communicate with each other. The message payload contains event class and event code in the common header (sf_message_header_t type data, see event class and event code) so that a message producer can post a message to the subscribers to inform the subscribers which event happened.

You must define a system specific message payload structure except for modules for which the SSP provides predefined structures such as the Touch Panel Framework and the audio playback modules. The message payload can contain additional data, which are required for the event processing, in addition to the common header.

The SSP contains the following predefined message payload structures:

- sf_touch_panel_payload_t type for the Touch Panel Framework module
- sf_audio_playback_data_t type for the Audio Playback Framework module

These framework modules use the Messaging Framework internally and define suitable message payload structures. An application thread which subscribes to touch event messages from the Touch Panel Framework module can use the predefined message payload by including the header file sf_touch_panel_api.h. Likewise, an application thread which posts audio event messages to the Audio Playback Framework module can use sf_audio_playback_api.h.

You must define a message payload structure for each event class code; exceptions include the SSP predefined payloads described previously or payloads that require only the common header. To create a new message payload structure, add a common message header (sf_message_header_t type structure) at the head in the user-specific message payload structure. The size of the header is 4 bytes.

NOTE: The payload size must not be greater than the buffer size.

The buffer size limit is critical; oversized data written beyond the buffer may destroy data in the block memory pool, which

is required by ThreadX kernel. Violating the size limit results in a hard fault exception. The buffer size can be configured in sf_message_ctrl_t::buffer_size.

Messaging Framework Module Important Operational Notes and Limitations

Messaging Framework and OS Message Queue Service

The Messaging Framework module uses the ThreadX primitive-message queue and kernel services and supports some enhancement over the ThreadX RTOS features. For this reason, the Messaging Framework module does not work exactly the same as the ThreadX message-queue service. However, a messaging system with the Messaging Framework module can work simultaneously with the ThreadX message queue services in an application if the two messaging systems are separated.

API Calls Contexts

- The open API can only be called from a thread; it can be called only once per the message framework control block instance. The behavior is undefined if the function is called twice.
- The close API can only be called from a thread.
- The bufferAcquire API can be called from a thread and an ISR.
- The bufferRelease API can only be called from a thread.
- The post API can be called from a thread and an ISR.
- The pend API can be called from a thread and an ISR.

Estimating the Number of Buffers

The number of buffers available to be allocated in the work memory should be estimated properly when designing the messaging system. The number of buffers is estimated as follows:

$$N \approx \frac{W_m}{M_b + B_{cb} + T_x} = \frac{W_m}{M_b + 12\ bytes + 4\ bytes}$$

Figure 199: Estimating Buffers Available in Work Memory

where:

N – number of buffers,

Wm – work memory size (in bytes),

Mb – message buffer size (in bytes),

Bcb = 12 bytes – size of buffer control block,

Tx = 4 bytes – reserved bytes for ThreadX.

The maximum number possible for buffers allocated at the same time equals the total amount of depth of message queues in the system. Ideally, the number of buffers for a robust system should be the sum of the depths of the message queues in the system.

Message Queue Size and Depth Setting

The Messaging Framework module needs a 4-byte memory block on the message queue as it delivers the pointer to the buffer which contains a message payload. For this reason, the size of the message queue should be fixed to 4 bytes. A size greater than 4 bytes negatively affects the performance, as the extra memory copy resides in the ThreadX message queue service, which is internally invoked by the Messaging Framework API functions.

The depth of the message queue is arbitrary, but it should accommodate the number of queued messages at runtime. As a guideline, estimate the value as follows:

$$D \approx \frac{P_{avg}}{S_{avg}}$$

Figure 200: Estimating Message Depth Value

where:

D – queue depth,

*P*avg – average message delivery rate from producers,

*S*avg – average event loop completion time in the subscriber.

Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.1.23.4 Including the Messaging Framework Module in an Application

This section describes how to include the Messaging Framework module in an application using the SSP configurator. The Messaging Framework can be used by all threads when added to a single thread.

NOTE: It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the *SSP User's Manual* to learn how to manage each of these important steps in creating SSP-based applications.

To add the Messaging Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Messaging Framework is g_sf_message0. This name can be changed in the associated Properties window.)

Messaging Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_message0 Messaging Framework on sf_message	Threads	New Stack> Framework> Services> Messaging Framework on sf_message

When the Messaging Framework module on sf_message is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information will have box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks.

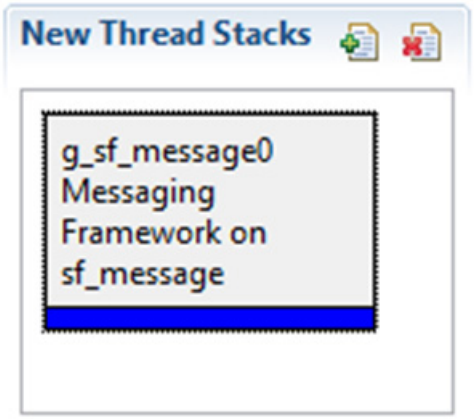


Figure 201: Messaging Framework Module Stack

4.1.23.5 Configuring the Messaging Framework Module

The Messaging Framework module must be configured by you for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules, for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available with the Properties window of the associated module. Simply select the indicated module and then view the Properties window. The interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE and create the module and explore the property settings in parallel with looking over the following configuration table settings. This helps to orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Messaging Framework Module on sf_message

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.

ISDE Property	Value	Description
Message Queue Depth (Total number of messages to be enqueued in a Message Queue)	16	Specify the size of Thread X Message Queue in bytes for Message Subscribers. This value is applied to all the Message Queues.
Name	g_sf_message0	The name of Messaging Framework module control block instance.
Work memory size in bytes	2048	Specify the work memory size in bytes. Choosing a small number results a small number of buffers which can be allocated at the same time (You can confirm the total buffer number on: sf_message_ctrl_t::number_of_buffers). If the value is smaller than the peak number of messages to be posted at the same time, the Framework occurs a buffer allocation failure affecting system performance.
Pointer to subscriber list array	p_subscriber_lists	Specify the name of pointer to the Subscriber List array.
name of the block pool internally used in the messaging framework	sf_msg_blk_pool	The name of memory block memory the Framework creates in the control block. This parameter might be useful for debugging purpose but NULL can be specified for saving memory.

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Messaging Framework Module Creating a Messaging Queue

The messaging configurator automatically creates the message queue for the subscribers.

Messaging Framework Module Configuring an Event Class and Event

To use the Messaging Framework with your own event class, use the **Threads** tab and the **Messaging** tab of the project configurator in the ISDE.

In the **Threads** tab, do the following:

1) Add a new thread in the **Threads** window and give it a unique name.
2) Add the **Messaging Framewor** component in the **Thread Stacks** panel of the **Threads** window.
3) In the **Messaging** tab (see event class code), do the following:A. In the **Event Class** window, add a class event.B. Enter the name of the event class for your thread to subscribe to in the **New Event** Class dialog box.

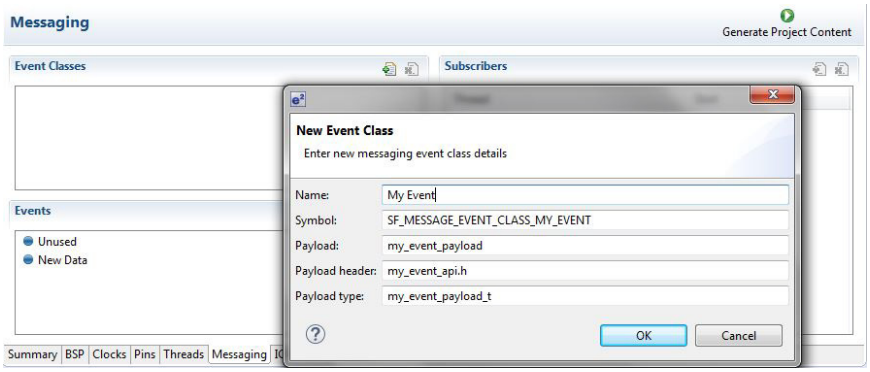


Figure 202: Messaging — e² studio New Event Class Configuration

4) In the **Events** window, add any events that your application may support (see event code).

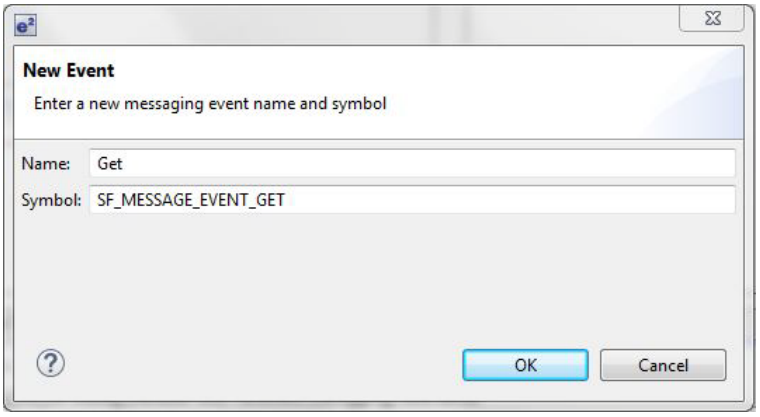


Figure 203: Messaging – e² studio New Event Configuration

Your custom event class code and event code are stored in a file named sf_message_port.h. The audio playback and the touch event classes are two predefined event classes in the SSP. The Touch Event Class only uses the new data event, SF_MESSAGE_EVENT_NEW_DATA.

Messaging Framework Module Configuring the Subscriber List

In the **Messaging** tab (see also Subscriber List), do the following:

1) Select the event class in the **Event Classes** window and configure a thread for the subscriber list in the event class **Subscribers** window.
2) Select your thread from the drop-down list in the **Threads** dialog box.
3) Next to start, enter the start number of the event class instance(s). If your system does not use multiple event class instances for the event class, or you are not sure what number to specify, just keep the default number (0.) Allowed values range from 0 to 255.

4) Next to End, enter the last number of the event class instance(s). If your system does not use multiple event class instances for the event class, or you are not sure what number to specify, just keep the default number (0.) Allowed values range from 0 to 255.
5) Click OK. A subscriber for your specified event is added in the subscriber list.
6) Repeat these steps for all event class instances if there are more than one.

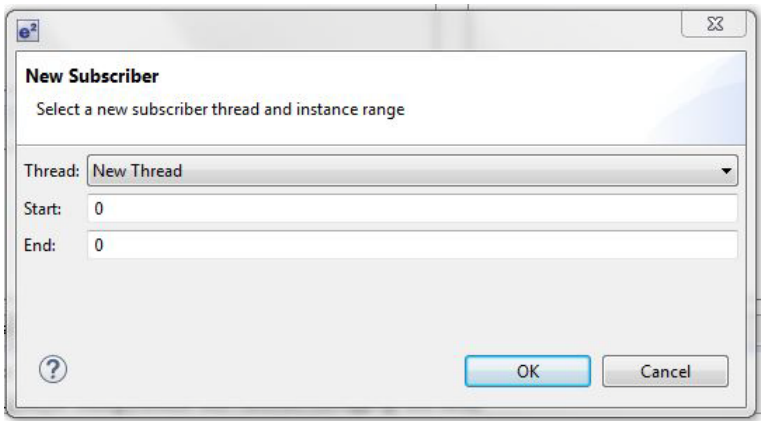


Figure 204: Messaging — e² studio New Subscriber Configuration

Messaging Framework Module Configuring the Event Class Code and Event Code

You can define your own message payload structure. Every user-defined message structure must include the sf_message_header_t type structure as one of the members, but the other members are entirely user-definable. The Messaging Framework does not care where the message payload structures are defined. You can include the file which defines your own message payload structure in the source file for your message producer and subscriber threads.

Configure the sf_message_cfg_t type configuration parameters to match your system. You can generate code for the configuration structure through the Synergy Configuration tool. Add a Messaging Framework component to the thread stacks in the **Threads** tab and modify the properties for the Messaging Framework module in the Properties window. When you press the **Generate Project Content** button, the code for the Messaging Framework module is generated on the thread code.

Before posting a message, an event producer thread must acquire a buffer for the message from the Messaging Framework module. An event producer thread can acquire the buffer by calling bufferAcquire.

When the API function returns SSP_SUCCESS, the buffer with message buffer size in bytes configured on Synergy Configuration tool is allocated in the memory pool managed by the Messaging Framework. The maximum number allowed to be allocated depends on the configuration **Work memory size in bytes** specified on the Synergy Configuration tool. For the estimation of the maximum number, see Estimating the Number of Buffers.

The bufferAcquire API has several options to change the message passing behavior:

- **buffer keep:** This option allows the application thread to hold the buffer not to be released by the API function bufferRelease if set to true. Typically, the buffer is to be released by bufferRelease when the message passing is done; however, in a scenario to have periodical or repeated message passing between threads, we can reuse the same buffer for the messaging without allocating and releasing the buffer each time. Enabling this option reduces the overhead in the buffer allocation/release operation and improve the system throughput.
- **wait_options:** This is the wait time option which is valid if all buffers have been acquired. Any arbitrary thread tick count, TX_WAIT_FOREVER, TX_NO_WAIT can be set for this option.

After message subscriber threads receive a message posted by an event producer, the message subscriber threads must release the buffer to the framework. Buffer releasing is performed by calling bufferRelease. Since the API function can be called multiple times if there are multiple event subscribers in the system, the actual buffer release is performed only by the last message subscriber thread in the event subscribers. For instance, if there were three subscribers in the subscriber group for the event class, the first and second thread which call bufferRelease do not release the buffer. Only the third thread releases the buffer. Note, if the**buffer keep** option is specified by bufferAcquire, the buffer is never being released except when option SF_MESSAGE_RELEASE_OPTION_FORCED_RELEASE is passed to the API function argument option. (Also see Messaging Framework callbacks for bufferRelease API function usage.)

The API is also used for invoking a user-callback function to create a handshake between an event producer thread and a message subscriber thread.

Posting a message

1) After getting a buffer by bufferAcquire, an event producer can write the message payload data to the buffer location.
2) **ATTENTION:** Writing data to the buffer is the user's responsibility and writing more data than the buffer size causes a fatal error in the Messaging Framework module.
3) Write an event class code to the sf_message_header_t::event_b.class_code in the payload structure.
4) Write an event code to the sf_message_header_t::event_b.code in the payload structure. It is not mandatory to specify this but necessary in most cases.
5) Write an event class instance number to sf_message_header_t::event_b.class_instance. Specify a number from 0 to 255 if your system has multiple instances for an event class. Specify 0 if your system simply uses single event class instance.
6) Post the message by the post API. Note that the pointer to the buffer needs to be casted with sf_message_header_t * type when given to the API. The message will be delivered to the message subscribers which are registered in the message subscriber list. The post API has several options to change the message passing behavior.
    - **Message priority:** Message can take two level message priority, SF_MESSAGE_PRIORITY_NORMAL or SF_MESSAGE_PRIORITY_HIGH. When SF_MESSAGE_PRIORITY_HIGH is specified, the message is queued at the front of the message queue of the message subscriber. This is typically used for the emergency message to make the message subscribers handle the event prior to the events which might have been queued in the message queue.
    - **User-callback function:** This function is registered in the buffer control block of the Messaging Framework module. The callback function is invoked by bufferRelease. This function can be used for handshaking between an event producer thread and a message subscriber thread.
    - **Wait_options:** This is the wait time option which is valid if a message queue of the message subscriber thread is full. Any arbitrary ThreadX tick count, TX_WAIT_FOREVER and TX_NO_WAIT can be set to this option.

Checking for a Pending Message

1) After the Messaging Framework module is opened, the message subscriber threads can wait for a message by calling pend. In general use, the second API argument specifies the pointer to a message queue, which you configured for the message subscriber thread in the Thread Subscribers pane in the **Messaging** tab, but you can specify the other message queues instead if required.
2) When a message is delivered from an event producer, the thread returns from pend.
3) The API returns the pointer to the buffer which contains the message to the thread through the third argument of the API.

4) The message subscriber casts the pointer above with a pointer type for the user custom message payload structure and does the event processing corresponding to the Event Class code sf_message_header_t::event_b.class_code, Event code sf_message_header_t::event_b.code and the user defined arbitrary data in the message.

NOTE: Pend has the wait_option to change the behavior of the API function:

The fourth argument of **pend** is the wait time option, which is only valid if the message queue of the message subscriber thread is empty. Any arbitrary ThreadX tick count, TX_WAIT_FOREVER, and TX_NO_WAIT can be set to this option. For details, see tx_queue_receive() ThreadX service call in the ThreadX User Guide.

Messaging Framework Module Interrupts

The Messaging Framework module does not use any interrupts.

4.1.23.6 Using the Messaging Framework Module in an Application

The typical steps in using the Messaging Framework module in an application are to first configure all the required settings as follows:

- Create a Message Queue
- Configure the Event Class and Event
- Configure the Subscriber List
- Configure the Event Class Code and Event Code

Once configuration is complete, the module's APIs can be used in the target application as follows:

1) Initialize the Messaging Framework with the open API
2) Acquire a buffer with the bufferAcquire API
3) Post a message with the post API
4) Check for a pending message with the pend API
5) Release a buffer using the bufferRelease API
6) Close the Messaging Framework with the close API

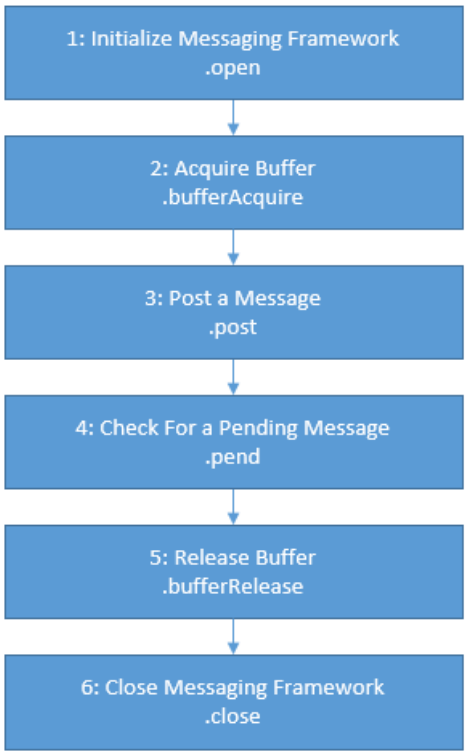


Figure 205: Flow Diagram of a Typical Messaging Framework Module Application

4.1.24 Power Profiles Framework

The Power Profiles Framework provides high-level APIs that can be configured to allow the MCU to be placed in lower power Software Standby mode. The module is implemented on sf_power_profiles.

This module is marked as [DEPRECATED]. We highly recommend that it not be used for development. The module is provided in this release only for backward compatibility with existing projects, which were developed with previous SSP releases. For all new projects we highly recommend the use of the replacement module, Power Profiles V2, since it is expected to be supported in future releases.

This document is divided into the following sections which can be read independently by a more experienced Synergy developer or in series by developers new to the Synergy Platform.

Power Profiles Framework Module Features

- Supports Software Standby operations
- Supports multiple operating modes
  - Run
  - RTC
  - External Interrupt
- Selects clocks and peripherals to disable during Software Standby

- Selects output pin state prior to and after exiting Software Standby

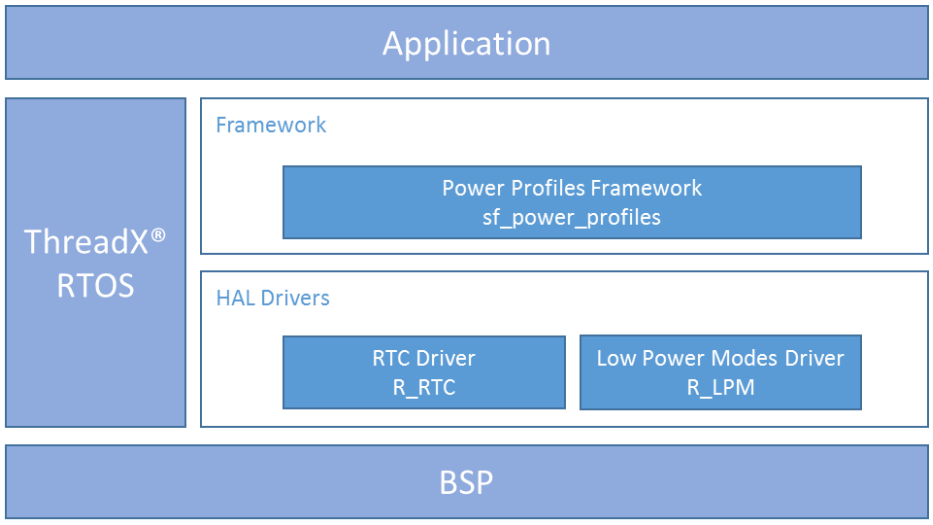


Figure 206: Power Profiles Organization, Options, and Stack Implementations

4.1.24.1 Power Profiles Framework Module APIs Overview

The Power Profiles section defines APIs for common functions such as open, sleep, and close. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values is also provided.

Power Profiles Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_power_profiles0.p_api->open(g_sf_power_profiles0 .p_ctrl, g_sf_power_profiles0.p_cfg); Initialize the Power Profiles Framework.
sleep	g_sf_power_profiles0.p_api->sleep(g_sf_power_profiles0 .p_ctrl); Places the MCU in software standby mode.
close	g_sf_power_profiles0.p_api->close(g_sf_power_profiles0 .p_ctrl); Close the PowerProfiles Framework.

Function Name	Example API Call and Description
versionGet	g_sf_power_profiles0.p_api->versionGet(&p_version); Get version and store it in provided pointer p_version.

NOTE: For more detailed descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables review the SSP Users Manuals API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_ASSERTION	Assertion error.
SSP_ERR_IN_USE	The framework has already been initialized.
SSP_ERR_INVALID_HW_CONDITION	Incompatible system clock configuration.
SSP_ERR_NOT_OPEN	Device not open.
SSP_ERR_UNSUPPORTED	The function is not supported by the module.
SSP_ERR_INTERNAL	Internal error.

NOTE: Lower level drivers may return Common Error Codes. Refer to the API Reference section for the associated module for a definition of all relevant status return values.

4.1.24.2 Power Profiles Framework Module Operational Overview

The Power Profile Framework uses the RTC, LPM, IOPORT and CGC peripherals on the Synergy microcontroller hardware and provides an easy-to-use software interface to access the low power operating modes. The Power Profiles Framework configures the system such that it enters and exits Software Standby mode in a predefined state. You can use the Power Profiles framework in both a threaded and non-threaded environment.

The Power Profiles module works as follows: the initial configuration, provided by the user determines the operating mode (RTC or External interrupt). In the Power Profiles module, you can define two pin configuration tables (Software Standby and wakeup). The configuration tables define the state of the MCU port pins while in Software Standby mode and subsequent to wake up. Both tables can be generated by using a bsp_pin_cfg.h file as a base and modifying the entries accordingly, including the name of the table.

Pointers to those tables are provided on the Power Profiles properties for Wakeup and Software Standby pin configuration. It is not a requirement to define the pin configuration tables. Providing NULL for one or both of these pointers prevents any pin re-configuration for the respective Sleep/Wake state.

Power Profiles Framework Module Operational Notes

With the Power Profiles Framework, you can place the MCU in Software Standby mode where it uses very low power while still retaining all the RAM and operating parameters. External Interrupt mode is the mode using the least amount of power, primarily because the RTC and both of its possible clock sources (LOCO or Sub-clock) are off.

In RTC mode, the RTC is allowed to run and as a result uses more power. Any peripherals or clocks that the MCU normally allows to remain operating in Software Standby mode are turned off by the Power Profiles module before entering Software Standby mode and subsequently restored upon waking up.

If you need a different operation outside of this behavior (for example, keeping LOCO running during Software Standby), you can configure this behavior by using either the SF_POWER_PROFILES_EVENT_PRE_SLEEP or SF_POWER_PROFILES_EVENT_POST_SLEEP callback events to include this specific behavior.

NOTE: The Power Profiles module does not enable wake up by numbered IRQs (IRQ0-IRQ15...), so extra work is required. To wake the MCU using a numbered IRQ, the application must use the r_lpm API functions wupenGet and wupenSet to enable wake up from standby mode by a numbered IRQ.

Power Profiles Framework Supported Operating Systems

When used with ThreadX, this framework uses ThreadX intrinsic objects like mutexes. Operation with ThreadX is optional.

Entering and Exiting Sleep Mode

The following block diagram illustrates the process flow for entering Sleep (Software Standby) mode.

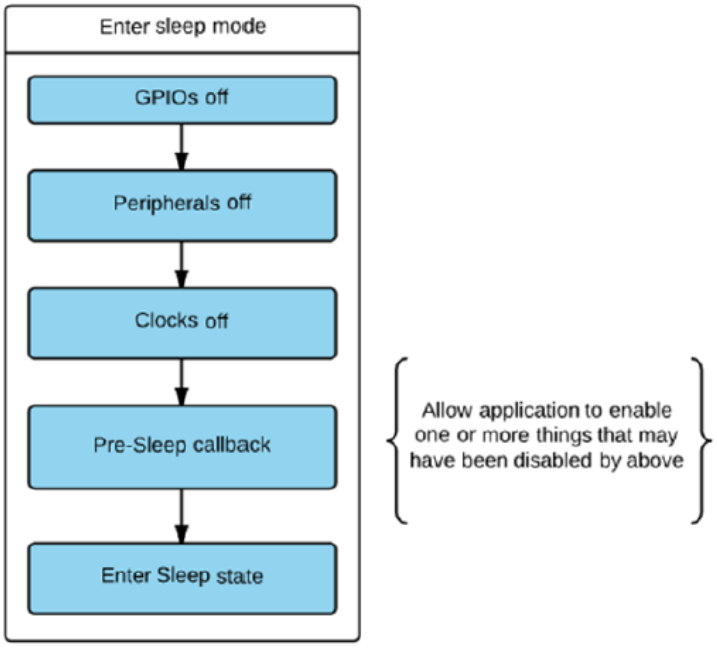


Figure 207: Entering Sleep Mode

The following block diagram illustrates the process flow for waking from Sleep (Software Standby) mode.

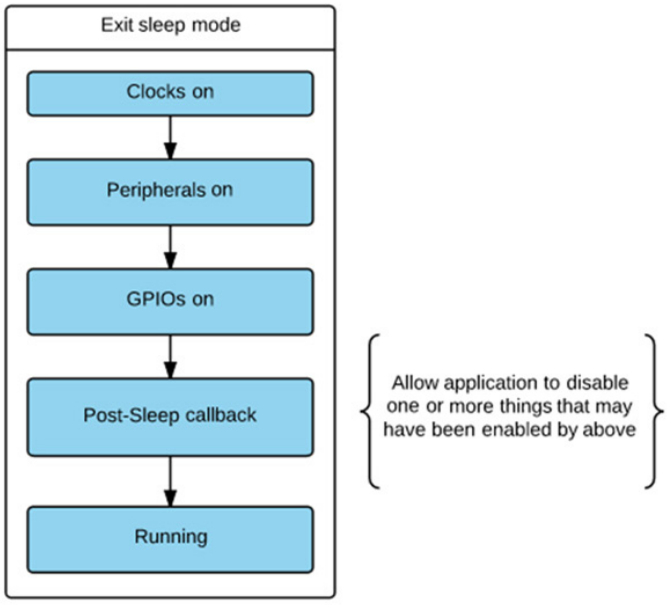


Figure 208: Exiting Sleep Mode

Power Profiles Framework Module Limitations

The Power Profiles Framework does not currently support any sleep mode other than Software Standby.

SF_POWER_PROFILES framework is superseded by SF_POWER_PROFILES_V2 framework. Projects that use SF_POWER_PROFILES can continue to do so, however it is recommended to update to SF_POWER_PROFILES_V2. Any new projects should be using SF_POWER_PROFILES_V2 instead of SF_POWER_PROFILES. SF_POWER_PROFILES has to be used with R_LPM whereas SF_POWER_PROFILES_V2 has to be used with R_LPM_V2. See the SF_POWER_PROFILES V2 usage notes for differences and features.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.24.3 Including the Power Profiles Framework Module in an Application

This section describes how to include the Power Profiles in an application using the SSP configurator.

NOTE: This section assumes that you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP based applications.

To add the Power Profiles framework module to an application, simply add it to a thread using the Stacks Selection Sequence given in the following table. (The default name for the Power Profiles is g_sf_touch_button0. This name can be changed in the associated **Properties** window.)

Power Profiles Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_power_profiles0 Power Profiles Framework on sf_power_profiles	**Threads**	**New Stack > Framework > Services > Power Profiles Framework on sf_power_profiles**

When the Power Profiles Framework module on sf_power_profiles is added to the thread stack as shown in the figure below, the configurator automatically adds any needed lower level modules. Any drivers that need additional configuration information will have a text box highlighted in Red. Modules with a Gray band are individual modules that stand alone.

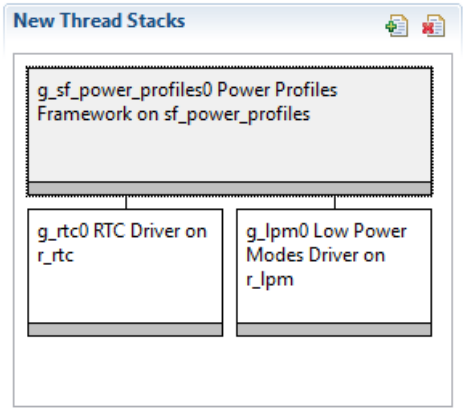


Figure 209: Power Profiles Framework Module Stack

4.1.24.4 Configuring the Power Profiles Framework Module

You must configure the Power Profiles module for the desired operation. The SSP configuration window automatically identifies any required configuration selections, such as Interrupts or operating modes, which must be configured for lower level modules, for successful operation, by highlighting the block in red. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the **Properties** tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority. This configuration setting is available with the **Properties** window of the associated module. Simply select the indicated module and then view the **Properties** window. The Interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the Interrupt priorities listed in the **Properties** window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring Interrupt Priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the Configuration Table Settings given below. This will help orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Power Profiles Framework on sf_power_profiles

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enables or disables the parameter error checking
RTC Support	Disabled, Enabled (Default: Disabled)	Enable or disable RTC support
Name	g_sf_power_profiles0	Module name
Callback	3	User determined based on priority of other threads in system. Recommend high priority.
Wakeup pin config table	NULL	Location of Wake-up pin config table
Sleep pin config table	NULL	Location of Sleep pin config table
Operating mode	Run, RTC, External (Default: Run)	Operating mode

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2. MCU Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower level modules can be desirable. The configurable properties for the lower level stack modules are given in the following sections for completeness, and as a reference.

NOTE: Most of the property settings for lower level modules are fairly intuitive and can usually be determined by the inspection of the associated properties window from the SSP configurator.

Configuration Settings for the Power Profiles Framework Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers and these are indicated with red text in the Thread Stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The table below identifies all the settings within the properties section for the module.

Configuration Settings for the RTC Driver on r_rtc

ISDE Property	Value	Description
Parameter Checking Enable	BSP, Enabled, Disabled (Default: BSP)	Enable or disable the parameter error checking
Name	r_rtc0	Name for the module
Clock Source	Sub Clock, LOCO, (Default: LOCO)	Select clock source for RTC module
Error Adjustment Value [DEPRECATED]	0	Select error adjustment value
Error Adjustment Type [DEPRECATED]	None	Select adjustment type
Callback	NULL	Callback name
Alarm Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) (Default: Disabled)	Set priority level for alarm event
Period Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) (Default: Disabled)	Set priority level for period event

ISDE Property	Value	Description
Carry Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) (Default: Disabled)	Set priority level for carry event

Configuration Settings for the Low Power Modes Driver on r_lpm

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Selects if code for parameter checking is to be included in the build
Name	g_lpm0	Module name
Operating power mode	High speed operating mode, Middle speed operating mode, Low speed operating mode, low voltage operating mode (Default: High speed operating mode)	Select operating mode
Sub oscillator mode	Sub oscillator mode not enabled, Sub oscillator mode enabled (Default: Sub oscillator mode not enabled)	Select sub oscillator mode

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Power Profiles Framework Module Clock Configuration

The Power Profiles framework does not require any specific clock settings.

Power Profiles Framework Module Pin Configuration

The application may optionally provide Software Standby and wakeup pin configuration tables to be used in setting the port pin states before entering Software Standby and after waking from Software Standby.

4.1.24.5 Using the Power Profiles Framework Module in an Application

Once you configure the modules and the ISDE has generated the files, follow the steps below to use the Power Profiles framework in a thread. Optionally, you can use the Power Profiles in a standalone (non-thread) application.

Define the body of the callback function configured the Power Profiles Framework field p_callback. The callback function notifies the application when the MCU is about to enter Software Standby mode and when the MCU just woke up from Software Standby mode. Using a callback is optional, but if you define a callback in the Power Profiles properties, then there must be a definition for it.

NOTE: Power Profiles does not enable wake up by numbered IRQs (IRQ0-IRQ15...). To wake the MCU using a numbered IRQ, the application must use the r_lpm API functions wupenGet and wupenSet to enable wake from standby mode by a numbered IRQ.

1) In the application thread, initialize the framework using the open function call.
2) Enter Software Standby mode by the sleep function.
3) Close the framework by calling the close function.

These common steps are illustrated in a typical operational flow diagram in the figure below:

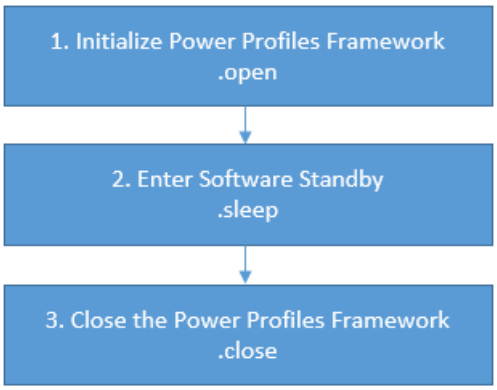


Figure 210: Flow Diagram of a Typical Power Profiles Framework Module Application

4.1.25 Power Profiles V2 Framework

The Power profiles V2 (PPv2) Framework provides more control for users to set a power control mode and the LPM mode in the Synergy MCU. The PPv2 Framework supports all the features of the Power Profiles v1 available in the current release of the Synergy Software Program (SSP) Framework using the LPM v1 driver. The Power Profiles v1 and PPv2 Framework are not compatible. Power Profiles v1 and PPv2 Framework cannot be used in the same project. For all new projects, it is recommended that applications use PPv2 Framework.

4.1.25.1 Power Profiles V2 Framework Module Features

- Configurable options to set different MCU power control modes with customizable clock domains.
- Configurable options to set different MCU low power modes with different IO port or pin configurations.
- Supports both threaded and non-threaded operations

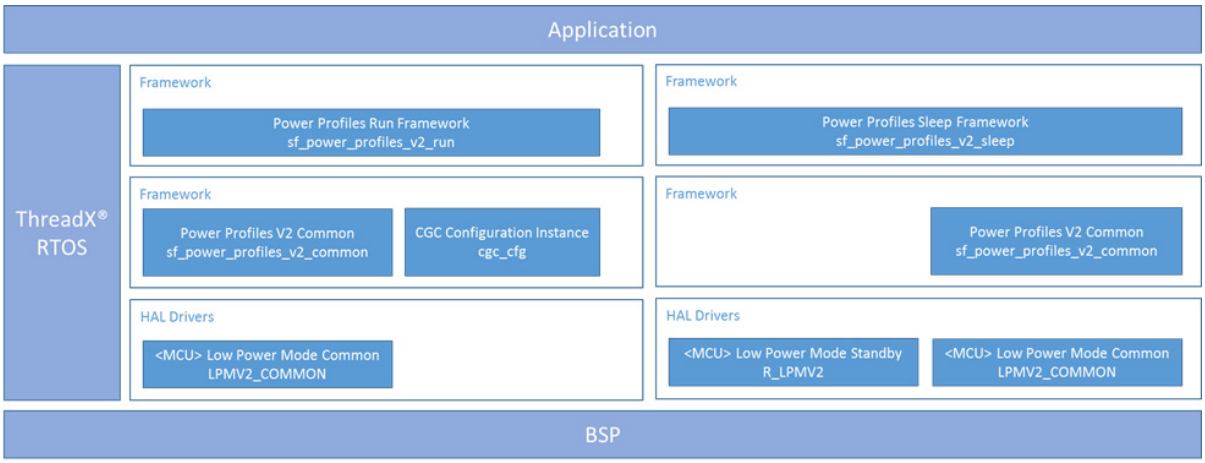


Figure 211: Power Profiles V2 Framework Organization, Options and Stack Implementations

Note that because there are many options for lower level LPMV2 HAL modules, since they are available for different MCUs, they are not all shown on the above diagram. The short-hand <MCU> indicator is replaced with the CPU name (like S7G2) in the actual framework implementation.

4.1.25.2 Power Profiles V2 Framework Module APIs Overview

There are different low-level LPM V2 HAL modules used in the framework depending on the target MCU. These are shown in the below table.

MCU	Driver
S124	S124 Low Power Mode Sleep on r_lpmv2
S124	S124 Low Power Mode Standby on r_lpmv2
S128	S128 Low Power Mode Sleep on r_lpmv2
S128	S128 Low Power Mode Standby on r_lpmv2
S3A3	S3A3 Low Power Mode Sleep on r_lpmv2
S3A3	S3A3 Low Power Mode Standby on r_lpmv2
S3A7	S3A7 Low Power Mode Sleep on r_lpmv2

MCU	Driver
S3A7	S3A7 Low Power Mode Standby on r_lpmv2
S5D9	S5D9 Low Power Mode Sleep on r_lpmv2
S5D9	S5D9 Low Power Mode Standby on r_lpmv2
S5D9	S5D9 Low Power Mode Deep Standby on r_lpmv2
S7G2	S7G2 Low Power Mode Sleep on r_lpmv2
S7G2	S7G2 Low Power Mode Standby on r_lpmv2
S7G2	S7G2 Low Power Mode Deep Standby on r_lpmv2

Because of the large number of lower level LPM V2 HAL modules available it would be difficult to identify separate APIs and configuration settings. This module guide only goes into details for the S7G2 MCU. All the APIs and configuration settings are the same (except for those MCUs that don't support Deep Standby) it should be easy to extrapolate to any target MCU.

The Power Profiles defines APIs for common functions such as open, sleep, and close. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values is also provided.

Power Profiles V2 Framework Module API Summary

Function Name	Example API Call and Description
.open	g_sf_power_profiles_v2_common.p_api->open(g_sf_power_profiles_v2_low_power_0.p_ctrl, g_sf_power_profiles0.p_cfg); Initialize the Power Profiles V2 Framework.
.runApply	g_sf_power_profiles_v2_common.p_api->runApply(g_sf_power_profiles_v2_common.p_ctrl, &g_sf_power_profiles_v2_run_0); Apply the selected run power profile.
.lowPowerApply	g_sf_power_profiles_v2_common.p_api->lowPowerApply (g_sf_power_profiles_v2_common.p_ctrl, &g_sf_power_profiles_v2_low_power_0); Apply the selected low power profile.

Function Name	Example API Call and Description
.close	g_sf_power_profiles_v2_common.p_api->close(g_sf_power_profiles_v2_common.p_ctrl); Close the module.
.versionGet	g_sf_power_profiles_v2_common.p_api->versionGet(&p_version); Get version and store it in provided pointer p_version.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables review the SSP Users Manuals API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_ASSERTION	Assertion error.
SSP_ERR_IN_USE	The framework has already been initialized.
SSP_ERR_INVALID_HW_CONDITION	Incompatible system clock configuration.
SSP_ERR_NOT_OPEN	Device not open.
SSP_ERR_UNSUPPORTED	The function is not supported by the module.
SSP_ERR_INTERNAL	Internal error.

NOTE: Lower level drivers may return Common Error Codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.25.3 Power Profiles V2 Framework Module Operational Overview

The PPv2 Framework provides a generic API for supporting low level power profiles in the Synergy MCU, when used with the LPM v2 Driver, CGC Driver, and IO Port Driver. It can be considered as an advanced control interface over the power consumption of the MCU, and can be used both in an application with or without ThreadX RTOS. Internally, it relies on the LPM v2, IOPORT, and CGC Drivers of the SSP, and provides an easy-to-use software interface to control the power modes of the MCU.

PPv2 Framework Run Profile

The **Run Profile** uses a CGC Clocks configuration and an IO Port pin configuration to set the system clocks and IO Port pins of the MCU in the normal running modes. Its function is implemented in the RunApply() API to perform the following tasks in the specified order.

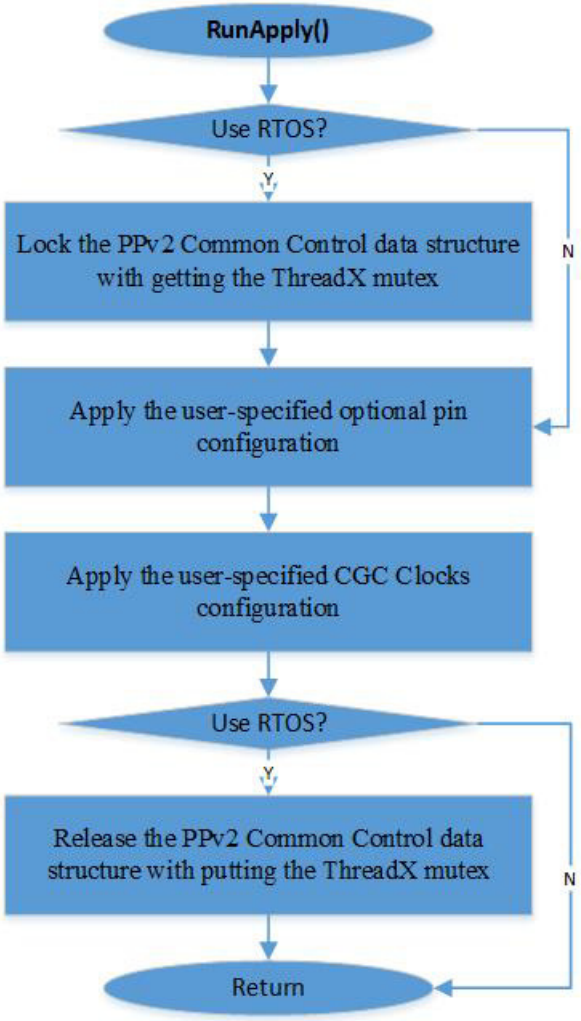


Figure 212: PPv2 RunApply() Process

PPv2 Framework Low Power Profile

The **Low Power Profile** uses an LPM v2 configuration and a pre-LPM and a post-LPM IO port configuration to set the low power mode and IO port pins before entering the configured Low Power mode and after waking up from the Low Power mode. See the *SSP User's Manual* and the corresponding Synergy Microcontroller Group, User's Manuals for details on the available Low Power modes.

The internal function LowPowerApply()API performs the following tasks in order.

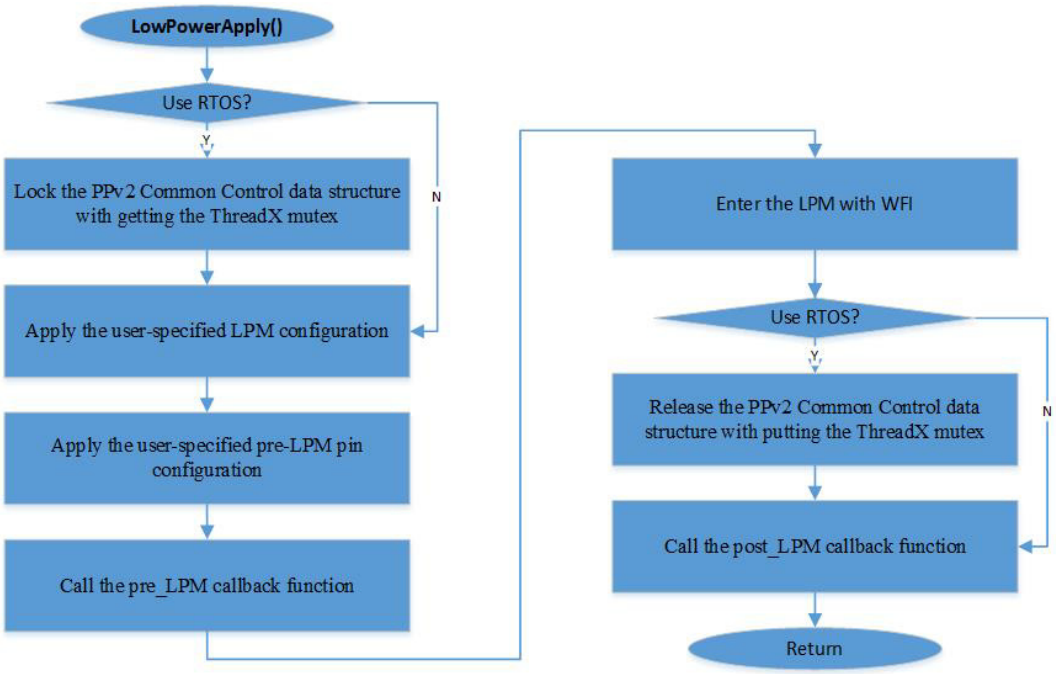


Figure 213: The PPv2 LowPowerApply() process

PPv2 Framework Operational Notes

The following operational notes are based on observations of the current release of the PPv2 framework and driver. It will be updated as more user feedback is received, and new versions of the power profiles package are released.

Power Profile v1 and PPv2

The Power Profiles v1 and PPv2 framework are not compatible, so do not use PPv1 and PPv2 framework in the same project. For all new projects, it is recommended that applications use PPv2 Framework.

An LPM v2 driver instance is added to PPv2 applications by default

This MCU specific LPM v2 driver defines configurations and APIs for configuring, enabling and disabling LPM operations in structures of lpmv2_mcu_cfg_t and lpmv2_api_t, then will be instantiated in the automatically generated file common_data.c. So, the PPv2 Framework instance is based on an LPM v2 Driver instance.

Additional CGC driver instance for using PPv2 runApply() function

A default CGC driver is included in all Synergy application projects, and instantiated in the file common_data.c, that will not increase the code size of a project. Another instance of the CGC Driver is required for a CGC Clocks configuration of a PPv2 runApply() function.

I/O Port Driver instance for different pin configurations

Different pin configuration tables can be defined in the PPv2 power profiles, but only one I/O port driver instance ioport_instance_t is instantiated in the automatically generated file common_data.c.

Operation with ThreadX

As shown in the above RunApply() and LowPowerApply() processes, the PPv2 Framework APIs uses a ThreadX intrinsic objects like a mutex for multithread applications when used with ThreadX.

Special Consideration on Multithreads Applications

In following the LPM modes: Software Standby, Deep Software Standby, or Snooze modes; the source clock for Systick may be configured to be disabled. Special consideration is needed when implementing a multithread RTOS project with the PPv2 LPM modes.

Debugger Usage in LPM Modes

By default, debuggers will prevent MCUs from entering LPM modes since Arm® Core is accessed by CoreSight™ debugging elements. There are two approaches to disconnect their attachment.

- Perform a full power-on reset cycle by unplugging and plugging the USB connection for JTAG;
- Enable Low Power Handling in the Synergy e^2 studio debugger and utilize a provided J-LINK script in **Debugger tab > Connection Settings**, as shown below.

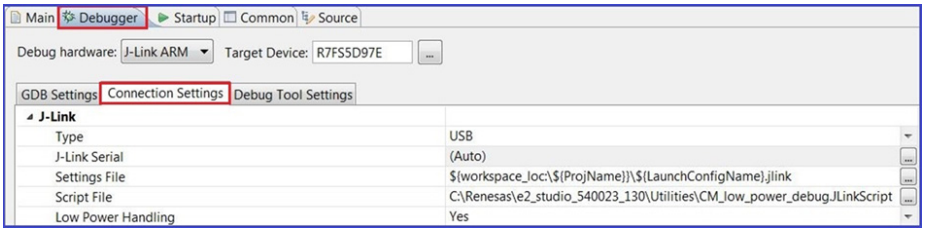


Figure 214: Enabling Low Power Handling in Synergy e^2 studio Debugger

Where the script CM_low_power_debug.JLinkScript will be provided per your request.

PPv2 Framework Limitations

- The Power Profiles V2 Framework open function will not be called automatically prior to the main if the project does not use ThreadX. The initialization must be done explicitly by calling g_common_init() or by explicitly calling the API function open(). This is not a PPv2 limitation, but a general approach of initializing a Framework module without using ThreadX RTOS.
- The PPv2 Framework does not handle starting or stopping MCU peripherals, since no property view is available to select which peripherals to be stopped in a LPM mode, so users must stop them manually.
- Current version of PPv2 Framework only support following Synergy MCUs: S124, S128, S3A7, S5D9, and S7G2.
- Current version of PPv2 Framework doesn't support the transition from the Snooze to the Normal in a LPM Standby mode.

4.1.25.4 Including the Power Profiles V2 Framework Module in an Application

This section describes how to include the Power Profiles in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP based applications.

To add the Power Profiles framework module to an application, simply add it to a thread using the Stacks Selection Sequence given in the table below. (The default name for the Power Profiles is g_sf_power_profiles_low_power_0 . This name can be changed in the associated Properties window.)

Power Profiles V2 Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_power_profiles_v2_low_power_0 Power Profiles V2 Low Power Profile	Threads	New Stack > Framework > Services > Power Profiles V2 Low Power Profile
g_sf_power_profiles_v2_run_0 Power Profiles V2 Run Profile	Threads	New Stack > Framework > Services > Power Profiles V2 Run Profile
g_sf_power_profiles_v2_common Power Profiles V2 Common	Threads	New Stack> Framework> Services> Power Profiles V2 Common

When the Power Profiles V2 low power or run module is added to the thread stack as shown in the figure below, the configurator automatically adds any needed lower level modules. Any drivers that need additional configuration information will have box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

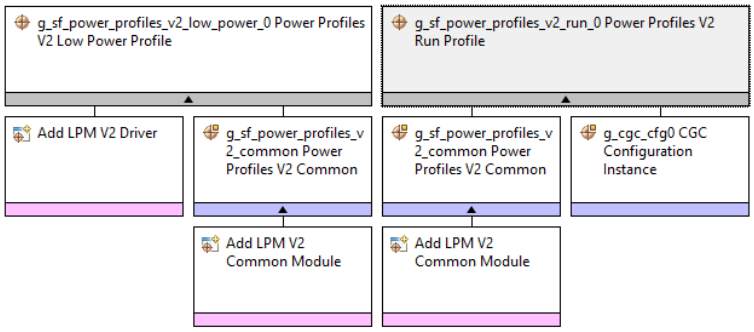


Figure 215: Power Profiles V2 Framework Module Stack

4.1.25.5 Configuring the Power Profiles V2 Framework Module

The Power Profiles V2 run and low power modules must be configured by you for the desired operation. The SSP configuration window will automatically identify, by highlighting the block in red, any required configuration selections, such as Interrupts or operating modes, which must be configured for lower level modules, for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority. This configuration setting is available with the properties window of the associated module. Simply select the indicated module and then view the properties window. The Interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the Interrupt priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the below configuration properties tables, but is easily visible with the ISDE when configuring Interrupt Priority levels.

NOTE: You may want to open your ISDE and create the module and explore the property settings in parallel with looking over the Configuration Table Settings given below. This will help orient you and can be a useful 'hands-on' approach to

learning the ins and outs of developing with SSP.

There are two different stack selections for the Power Profiles V2 Framework, the Run Profile and the Low Power Profile. Their respective configuration settings will be covered separately in the following sections.

Configuration Settings for the Power Profiles V2 Run Profile

Typically, only a small number of settings must be modified from the default for lower level drivers and these are indicated via the red text in the Thread Stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The table below identifies all the settings within the properties section for the module.

Configuration Settings for the Power Profiles V2 Run Profile

ISDE Property	Value	Description
Name	g_sf_power_profiles_v2_run_0	Module name
Pin configuration table	NULL	Pin configuration table selection

A pin configuration table can be created for a power mode, and linked into the Run profile property. Otherwise, the power mode uses the default pin assignment given by g_bsp_pin_cfg.

Basic steps to create a custom pin configuration table are as follows:

- Copy a given board pin configuration file, such as S5D9-PK.pincfg by clicking the right mouse button and selecting the **Copy** operation, then **Paste** it into the same project.
- Rename this new pin configuration file, S5D9-PK_RUN.pincfg.

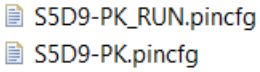


Figure 216: Create a New Pin Configuration File with Copy and Paste

- Select the new pin configuration from the pull-down menu on the Pins tab of SSP Configurator, then specify a pin configuration name to be generated as shown below.

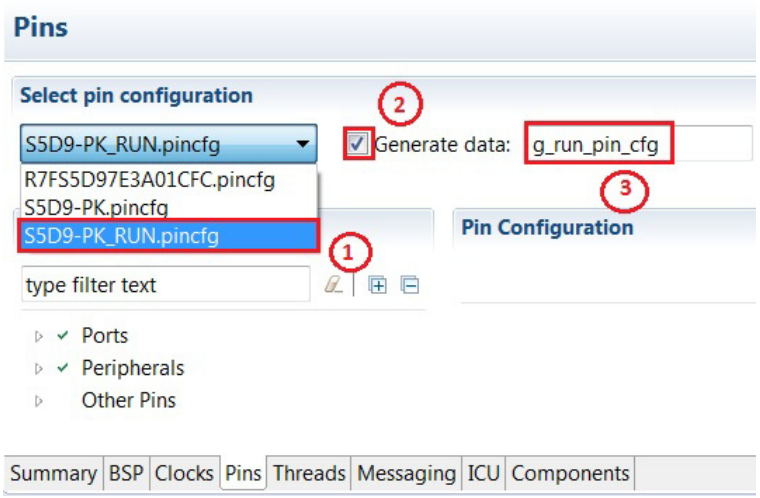


Figure 217: Select a New Pin Configuration File to be Assigned

- Configure the I/O functions on each pin, then generate a new pin configuration when you press the button **Generate Project Content** on the SSP Configurator as specified in the *Synergy SSP User's Manual.

A power control mode of this Run Profile can be defined using the property view of a CGC Configuration Instance.

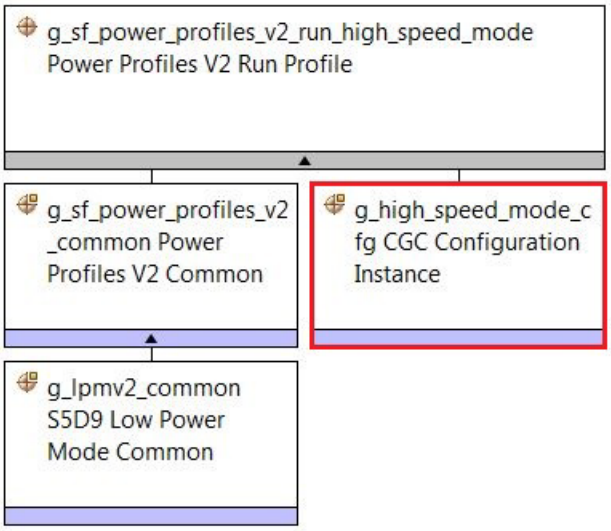


Figure 218: CGC Configuration Instance in the Run Profile Module

The parameters of the CGC configuration property view are shown as follows:

CGC Configuration Settings on a Run Profile Module

ISDE Property	Value	Description
Name	g_cgc_cfg0	Module name
System Clock	HOCO, MOCO, LOCO, Main Oscillator, Sub Clock, PLL Default: HOCO	Set the system clock source
LOCO State Change	None, Start, Stop Default: None	LOCO state change selection
MOCO State Change	None, Start, Stop Default: None	MOCO state change selection
HOCO State Change	None, Start, Stop Default: None	HOCO state change selection
Sub-Clock State Change	None, Start, Stop Default: None	Sub-clock state change selection
Main Clock State Change	None, Start, Stop Default: None	Main clock state change selection
PLL State Change	None, Start, Stop Default: None	PLL source clock selection
PLL Source Clock	HOCO, MOCO, LOCO, Main Oscillator, Sub Clock, PLL Default: HOCO	Set the PLL source

ISDE Property	Value	Description
PLL Divisor	1, 2, 3, 4 Default: 1	PLL Output Frequency Division
PLL Multiplier	10.0, 10.5, 11.0, 11.5, 12.0, 12.5, 13.0, 13.5, 14.0, 14.5, 15.0, 15.5, 16.0, 16.5, 17.0, 17.5, 18.0, 18.5, 19.0, 19.5, 20.0, 20.5, 21.0 , 21.5, 22.0, 22.5, 23.0, 23.5, 24.0, 24.5, 25.0, 25.5, 26.0, 26.5, 27.0, 27.5, 28.0, 28.5, 29.0, 29.5, 30.0, 31.0 Default: 10.0	PLL Output Frequency Multiplication
PCLKA Divisor	1, 2, 4, 8, 16, 64 Default: 1	Peripheral Clock A Division
PCKLB Divisor	1, 2, 4, 8, 16, 64 Default: 1	Peripheral Clock B Division
PCLKC Divisor	1, 2, 4, 8, 16, 64 Default: 1	Peripheral Clock C Division
PCLKD Divisor	1, 2, 4, 8, 16, 64 Default: 1	Peripheral Clock D Division
BCLK Divisor	1, 2, 4, 8, 16, 64 Default: 1	External Bus Clock Division
FCLK Divisor	1, 2, 4, 8, 16, 64 Default: 1	Flash Clock Division

ISDE Property	Value	Description
ICLK Divisor	1, 2, 4, 8, 16, 64 Default: 1	System Clock Division

NOTE: The assignments on these CGC parameters must be satisfied with oscillator availability for each power control mode, and the expected frequencies of each operating clock. Their relationships are illustrated in a CGC Block Diagram in the corresponding Synergy MCU User's Manual.

The maximum operating frequency range for these clocks should not be outside the given ranges in the corresponding MCU User's Manual. The *S5D9 Microcontroller Group User's Manual* lists this specification in the following table.

Maximum Operating Frequency Range for Synergy S5D9 MCU Group Internal Clocks

Parameter	Clock sources	Clock supply	Specifications
Peripheral module clock A (PCLKA)	MOSC, SOSC, HOCO, MOCO, LOCO, PLL	Peripheral modules (ETHERC, EDMAC, USBHS, QSPI, SPI, SCI, SCE7, GLCDC, SDHI, CRC, JPEG engine, DRW, irDA, GPY bus-clock)	Up to 120 MHz*2 Division ratios: 1, 2, 4, 8, 16, 32, 64
Peripheral module clock B (PCLKB)	MOSC, SOSC, HOCO, MOCO, LOCO, PLL	Peripheral modules (IIC, SSIE, SRC, DOC, CAC, CAN, DAC12, POEG, CTSU, AGT, Standby SCRAM, ELC, I/O Ports, RTC, WDT, IWDT, ADC12, KINT, USBFS, ACMPHS, TSN, PDC)	Up to 60 MHz Division ratios: 1, 2, 4, 8, 16, 32, 64
Peripheral module clock C (PCLKC)	MOSC, SOSC, HOCO, MOCO, LOCO, PLL	Peripheral module (ADC12 conversion clock)	Up to 60 MHz Division ratios: 1, 2, 4, 8, 16, 32, 64
Peripheral module clock D (PCLKD)	MOSC, SOSC, HOCO, MOCO, LOCO, PLL	Peripheral module (GPT count-clock)	Up to 120 MHz Division ratios: 1, 2, 4, 8, 16, 32, 64

Parameter	Clock sources	Clock supply	Specifications
Flash interface clock (FCLK)	MOSC, SOSC, HOCO, MOCO, LOCO, PLL	Flash interface	Up to 60 MHz (P/E) Up to 60 MHz (read) *1 Division ratios: 1, 2, 4, 8, 16, 32, 64
External bus clock (BCLK)	MOSC, SOSC, HOCO, MOCO, LOCO, PLL	External bus	Up to 120 MHz Division ratios: 1, 2, 4, 8, 16, 32, 64
EBCLK pin output (EBCLK)	BCLK or ½ BCLK	EBCLK pin	Up to 60 MHz Division ratios: 1, 2
SDCLK pin output (SDCLK)	BCLK	SDCLK pin	Up to 120 MHz
USB clock (CABMCLK)	PLL	USB	48 MHz Division ratios: 3, 4, 5,
USB-PHY clock (USBMCLCK)	MOSC	USB-PHY	12, 20, 25 MHz
Can CLOCK (CANMCLK)	MOSC	CAN	8 to 24 MHz
LCD_CLCK pin output (LCD_CLK) and graphic LCD pixel clock (PXCLK)	LCD_EXTCLK, PLL output	LCD_CLK pin, peripheral module (Graphics LLCD Controller)	Up to 54 MHz (parallel RGB) Up to 60 MHz (serial RGB) LCD_CLK : PXCLK = 1.1 (parallel RGB)

As the current property view does not validate your assignments automatically, you can use the CGC panel of the SSP Configurator to check your assignments first. The following screen shot of the CGC configuration is an example of the High-Speed mode.

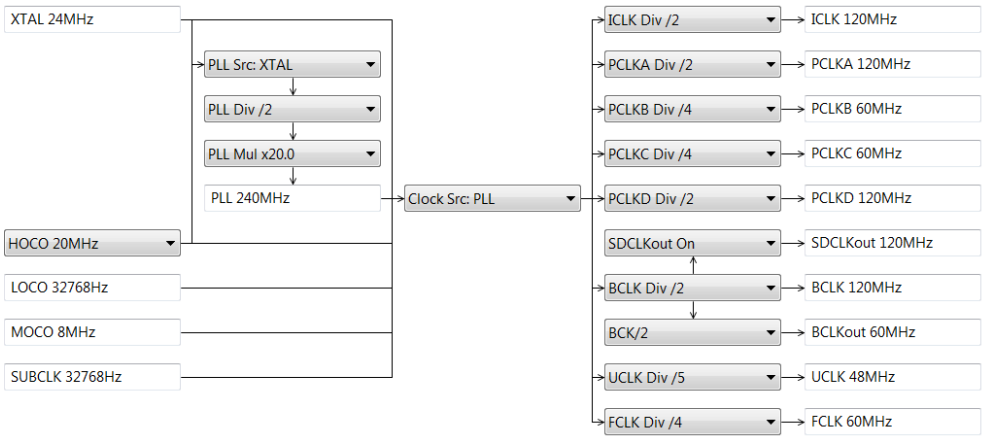


Figure 219: Checking the clock setting for a power control mode

Similarly, you can define the CGC configuration for other power control modes.

Configuration of PPv2 Framework Low Power Profile

Depending on the selected Synergy MCU Group, you may have 3 LPM modes: Sleep, Software Standby, and Deep Software Standby modes, available in the pull-down menu of the Low Power Profile module.

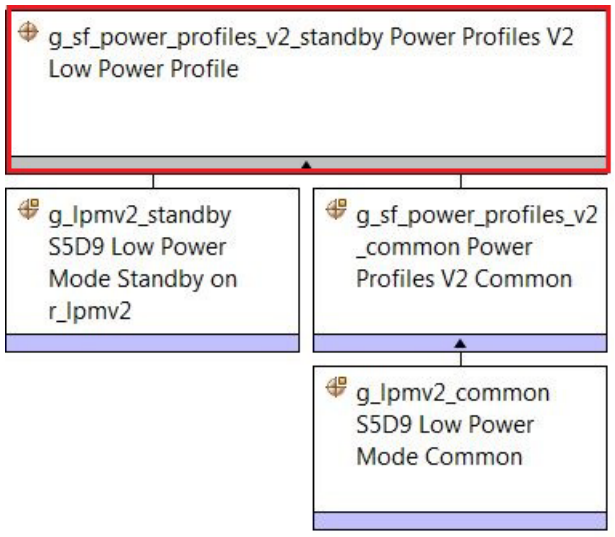


Figure 220: PPv2 Low Power Profile Standby Module

Assuming that a Low Power Profile module is already added into a thread, it will have the following configurations.

Configuration settings on the PPv2 Low Power Profile module

ISDE Property	Value	Description
Name	g_sf_power_profiles_v2_low_power_0	Module name
Callback (Low Power Exit Event N/A when using Deep Software Standby)	NULL	Callback selection
Low power entry pin configuration table	NULL	Low power entry pin configuration table selection
Low power exit pin configuration table	NULL	Low power exit pin configuration table selection

The callback function can be used for handling the following events:

- SF_POWER_PROFILES_V2_EVENT_PRE_LOW_POWER
- SF_POWER_PROFILES_V2_EVENT_POST_LOW_POWER

You can change the IO port functionality with the following two pin configuration tables:

- Low power entry pin configuration table
- Low power exit pin configuration table

They will be used internally in the PPv2 API function LowPowerApply().

The configuration of the LPM Sleep mode is simple, since any interrupt wakes the MCU from the Sleep mode. So only the module name can be renamed.

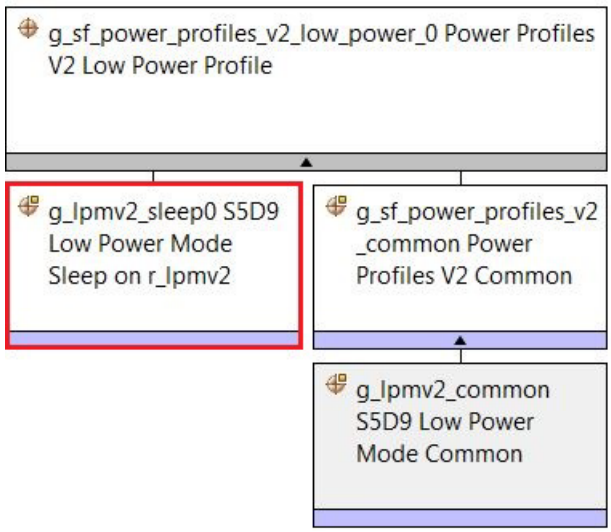


Figure 221: PPv2 Low Power Profile Sleep Driver r_lpmv2

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking
Name	g_lpmv2_standby0	Module name
Choose the low power mode	Standby, Standby with snooze Enabled Default: Standby	Low power mode selection
Output port state in standby and deep standby, applies to address output, data output, and other bus control output pins	High impedance state, No change Default: No change	Output port state selection
IRQ1-15	Enabled, Disabled Default: Disabled	IRQ1-15 selection
IWDT	Enabled, Disabled Default: Disabled	IWDT selection
Key Interrupt	Enabled, Disabled Default: Disabled	Key Interrupt selection
LVD1 Interrupt	Enabled, Disabled Default: Disabled	LVD1 Interrupt selection
LVD2 Interrupt	Enabled, Disabled Default: Disabled	LVD2 Interrupt selection
Analog Comparator High-speed 0 Interrupt	Enabled, Disabled Default: Disabled	Analog Comparator High-speed 0 Interrupt selection

ISDE Property	Value	Description
RTC Alarm	Enabled, Disabled Default: Disabled	RTC Alarm selection
RTC Period	Enabled, Disabled Default: Disabled	RTC Period selection
USB High-speed	Enabled, Disabled Default: Disabled	USB High-speed selection
USB Full-speed	Enabled, Disabled Default: Disabled	USB Full-speed selection
AGT1 underflow	Enabled, Disabled Default: Disabled	AGT1 underflow selection
AGT1 Compare Match A	Enabled, Disabled Default: Disabled	AGT1 Compare Match A selection
AGT1 Compare Match B	Enabled, Disabled Default: Disabled	AGT1 Compare Match B selection
12C 0	Enabled, Disabled Default: Disabled	12C 0 selection
Snooze Entry Source	RXD0 falling edge, IRQ0-IRQ15, KINT, ACMPHS0, RTC Alarm, RTC Period, AGT1 Underflow, AGT1 Compare Match A, AGT1 Compare Match B Default: RXD0 falling edge	Snooze Entry Source selection

ISDE Property	Value	Description
AGT1 Underflow	Enabled, Disabled Default: Disabled	AGT1 Underflow selection
DTC Transfer Completion	Enabled, Disabled Default: Disabled	DTC Transfer Completion selection
DTC Transfer Completion Negated Signal	Enabled, Disabled Default: Disabled	DTC Transfer Completion Negated Signal selection
ADC0 Compare Match	Enabled, Disabled Default: Disabled	ADC0 Compare Match selection
ADC0 Compare Mismatch	Enabled, Disabled Default: Disabled	ADC0 Compare Mismatch selection
ADC1 Compare Match	Enabled, Disabled Default: Disabled	ADC1 Compare Match selection
ADC1 Compare Mismatch	Enabled, Disabled Default: Disabled	ADC1 Compare Mismatch selection
SCI0 Address Match	Enabled, Disabled Default: Disabled	SCI0 Address Match selection
DTC state in Snoooze Mode	Enabled, Disabled Default: Disabled	DTC state in Snoooze Mode selection

Configuration Settings on the PPv2 Profile Sleep module

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking
Name	g_lpmv2_sleep0	Module name

The Standby mode configuration sets up the exit triggers, and some transition conditions between the Standby and the Snooze mode, which are treated as a special case of the Standby mode in the current release of the PPv2. More discussion on the Snooze configuration is provided in the next section.

As an example, a S5D9 MCU Standby module is shown in the following figure.

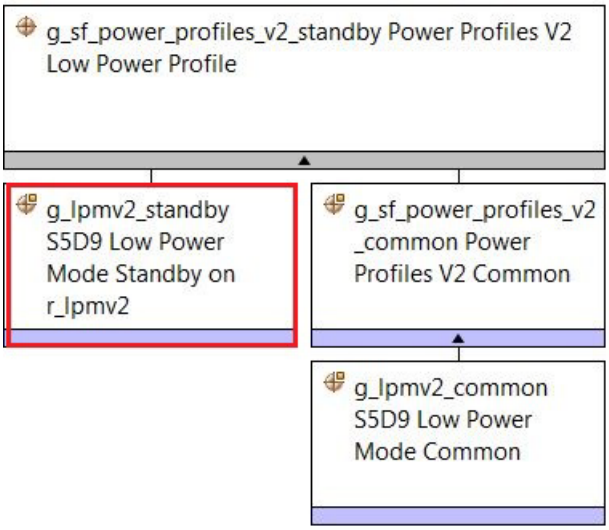


Figure 222: PPv2 Low Power Profile Standby Driver r_lpmv2

Configuration Settings on the PPv2 Profile Standby Module

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking
Name	g_lpmv2_standby0	Module name

ISDE Property	Value	Description
Choose the low power mode	Standby, Standby with snooze Enabled Default: Standby	Low power mode selection
Output port state in standby and deep standby, applies to address output, data output, and other bus control output pins	High impedance state, No change Default: No change	Output port state selection
IRQ1-15	Enabled, Disabled Default: Disabled	IRQ1-15 selection
IWDT	Enabled, Disabled Default: Disabled	IWDT selection
Key Interrupt	Enabled, Disabled Default: Disabled	Key Interrupt selection
LVD1 Interrupt	Enabled, Disabled Default: Disabled	LVD1 Interrupt selection
LVD2 Interrupt	Enabled, Disabled Default: Disabled	LVD2 Interrupt selection
Analog Comparator High-speed 0 Interrupt	Enabled, Disabled Default: Disabled	Analog Comparator High-speed 0 Interrupt selection
RTC Alarm	Enabled, Disabled Default: Disabled	RTC Alarm selection

ISDE Property	Value	Description
RTC Period	Enabled, Disabled Default: Disabled	RTC Period selection
USB High-speed	Enabled, Disabled Default: Disabled	USB High-speed selection
USB Full-speed	Enabled, Disabled Default: Disabled	USB Full-speed selection
AGT1 underflow	Enabled, Disabled Default: Disabled	AGT1 underflow selection
AGT1 Compare Match A	Enabled, Disabled Default: Disabled	AGT1 Compare Match A selection
AGT1 Compare Match B	Enabled, Disabled Default: Disabled	AGT1 Compare Match B selection
12C 0	Enabled, Disabled Default: Disabled	12C 0 selection
Snooze Entry Source	RXD0 falling edge, IRQ0-IRQ15, KINT, ACMPHS0, RTC Alarm, RTC Period, AGT1 Underflow, AGT1 Compare Match A, AGT1 Compare Match B Default: RXD0 falling edge	Snooze Entry Source selection
AGT1 Underflow	Enabled, Disabled Default: Disabled	AGT1 Underflow selection

ISDE Property	Value	Description
DTC Transfer Completion	Enabled, Disabled Default: Disabled	DTC Transfer Completion selection
DTC Transfer Completion Negated Signal	Enabled, Disabled Default: Disabled	DTC Transfer Completion Negated Signal selection
ADC0 Compare Match	Enabled, Disabled Default: Disabled	ADC0 Compare Match selection
ADC0 Compare Mismatch	Enabled, Disabled Default: Disabled	ADC0 Compare Mismatch selection
ADC1 Compare Match	Enabled, Disabled Default: Disabled	ADC1 Compare Match selection
ADC1 Compare Mismatch	Enabled, Disabled Default: Disabled	ADC1 Compare Mismatch selection
SCI0 Address Match	Enabled, Disabled Default: Disabled	SCI0 Address Match selection
DTC state in Snoooze Mode	Enabled, Disabled Default: Disabled	DTC state in Snoooze Mode selection

Currently, the Snooze mode is enabled in the property view of the PPv2 Software Standby mode by selecting **Standby with Snooze Enabled** in the field of the **Choose the low power mode** as shown in the following figure.

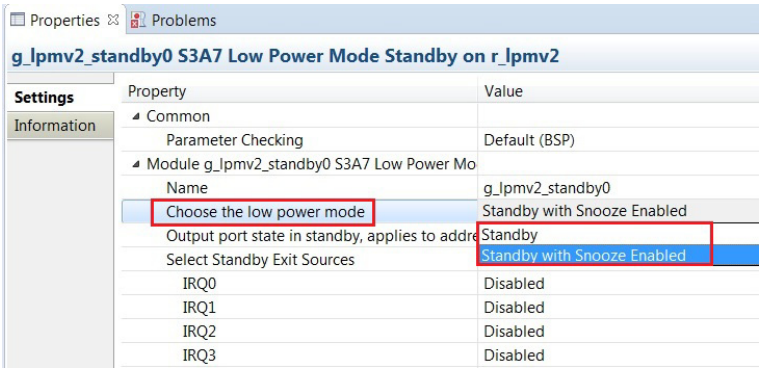


Figure 223: Create a Snooze mode in the Software Standby mode

The configuration on the Deep Software Standby mode is to set the exit triggers of the Deep Software Standby mode, and the internal power supply options. Use the PPv2 Framework on the S5D9 as an example to show the possible configurations as follows.

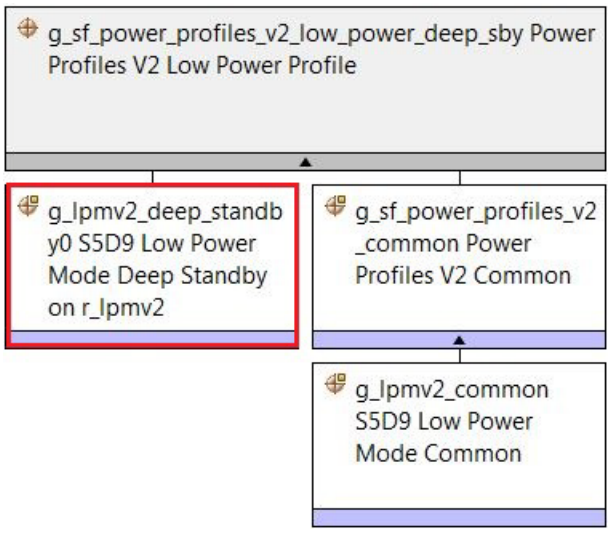


Figure 224: PPv2 Low Power Profile Deep Software Standby driver r_lpmv2

Configuration settings on the PPv2 Profile Deep Software Standby module

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking
Name	g_lpmv2_deep_standby	Module name

ISDE Property	Value	Description
Output port state in standby and deep standby, applies to address output, data output, and other bus control output pins	High impedance state, No change Default: No change	Output port state setting in Standby and Deep Software Standby
Maintain or reset the IO port states on exit from deep standby mode	Maintain the IO port states, Reset the IO port states Default: Maintain the IO port states	Output port state setting exit
Internal power supply control in deep standby mode	Maintain the internal power supply, Cut the power supply to standby RAM, low-speed on-chip oscillator, AGTn, and USPFS/HS resume detecting unit, Cut the power supply to LVDn, standby RAM, low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit Default: Maintain the internal power supply	Internal power supply control in Deep Software Standby mode setting
IRQ0-15	Enabled, Disabled Default: Disabled	IRQ0-15 selection
IRQ0-15 Edge	Disabled, Rising Edge, Falling Edge Default: Disabled	IRQ0-15 Edge selection
LVD1	Enabled, Disabled Default: Disabled	LVD1 selection
LVD1 Edge	Disabled, Rising Edge, Falling Edge Default: Disabled	LVD1 Edge selection
LVD2	Enabled, Disabled Default: Disabled	LVD2 selection

ISDE Property	Value	Description
LVD2 Edge	Disabled, Rising Edge, Falling Edge Default: Disabled	LVD2 Edge selection
RTC Interval	Enabled, Disabled Default: Disabled	RTC Interval selection
RTC Alarm	Enabled, Disabled Default: Disabled	RTC Alarm selection
NMI	Enabled, Disabled Default: Disabled	NMI selection
NMI Edge	Disabled, Rising Edge, Falling Edge Default: Disabled	NMI Edge selection
USBFS	Enabled, Disabled Default: Disabled	USBFS selection
UBSHS	Enabled, Disabled Default: Disabled	UBSHS selection
AGT11	Enabled, Disabled Default: Disabled	AGT11 selection

NOTE: The property dialog of the PPv2 Framework provides two predefined configurations for internal power supply control in Deep Software Standby mode. More selections for stopping power supply to internal components are listed in the *Synergy Microcontroller Group User's Manual*, such as in Table 11.2 of the *Synergy S5D9 Microcontroller Group User's Manual*.

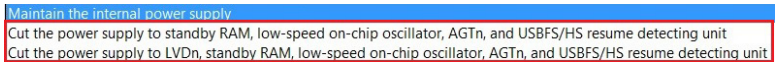


Figure 225: Predefined internal power supply options in the PPv2 Framework Deep Software Standby mode

Configuration of PPv2 Framework Common Modules

There are two Common modules shared among PPv2 Framework Run/Low Power Profile modules:

- **Automatically Generated Power Profile V2 Common Module**

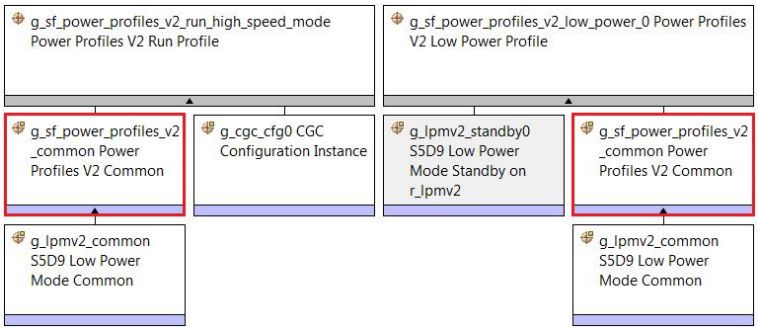


Figure 226: PPv2 Common module of PPv2 Run/Low Power Profile modules

Their configurations are displayed as follows:

Configuration settings for the Power Profiles V2 Common

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_sf_power_profiles_v2_common	Module name

- **Users Added MCU Specific Low Power Mode Common Module**

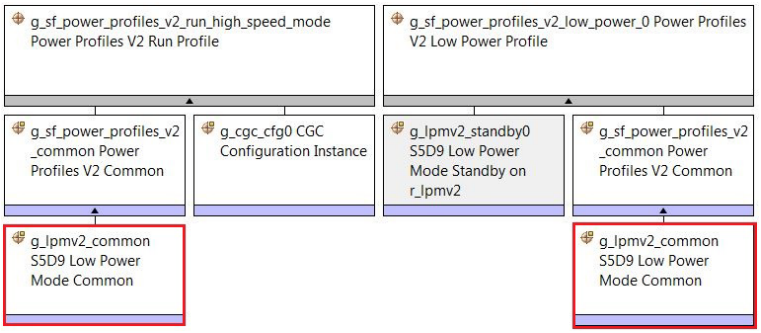


Figure 227: LPM Common Module of PPv2 Run/Low Power Profile Modules

Their configurations are displayed as follows:

Configuration Settings for the Low Power Mode Common

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_lpmv2_common	Module name.

NOTE: The Synergy MCU S5D9 LPM module is already added in Figure 13 when adding a PPv2 Framework Run profile module.

The Power Profiles V2 framework does not require any specific clock settings.

The application may optionally maintain the Io Port state during a low power mode.

4.1.25.6 Using the Power Profiles V2 Framework Module in an Application

Once you have configured the modules and the ISDE has generated the files, follow the steps below to use the Power Profiles V2 Framework in a thread. The steps below assume that the user defined name for the Power Profiles V2 Framework common instance is g_sf_power_profiles_v2_common. The steps below assume that the user defined names for the Power Profiles V2 framework Low Power and Run profiles are g_sf_power_profiles_v2_low_power_0 and g_sf_power_profiles_v2_run_0.

Before using any APIs, you must define the body of the callback function configured in the Low Power profile. The callback function notifies the application when the MCU is about to enter a low power mode and when the MCU just woke

up from a low power mode. Using a callback is optional, but if you define a callback in the Power Profiles V2 properties, then there must be a definition for it.

1) If ThreadX is being used, the Power Profiles V2 Framework open API function will be called by the Synergy generated code before the user application code is reached. If ThreadX is not used, the application must call the open function.

2) Apply a Run Profile at any time using the runApply() API. The runApply() API accepts a Run Profile as its second parameter. The parameter can be any valid Run Profile, allowing the application to easily switch between Run Profiles.

```
g_sf_power_profiles_v2_common.p_api->runApply(g_sf_power_profiles_v2_common.p_ct
rl,
&g_sf_power_profiles_v2_run_0);
```

3) Apply a Low Power Profile using the lowPowerApply() API. The lowPowerApply() API accepts a Low Power Profile as its second parameter. The parameter can be any valid Low Power Profile, allowing the application to easily switch between Low Power Profiles.

```
g_sf_power_profiles_v2_common.p_api->lowPowerApply(g_sf_power_profiles_v2_common
.p_ctrl,
&g_sf_power_profiles_v2_low_power_0);
```

4) *Optional*: Close the framework by calling the close function.

```
g_sf_power_profiles_v2_common.p_api->close(g_sf_power_profiles_v2_common.p_ctrl)
;
```

These common steps are illustrated in a typical operational flow diagram in the figure below:

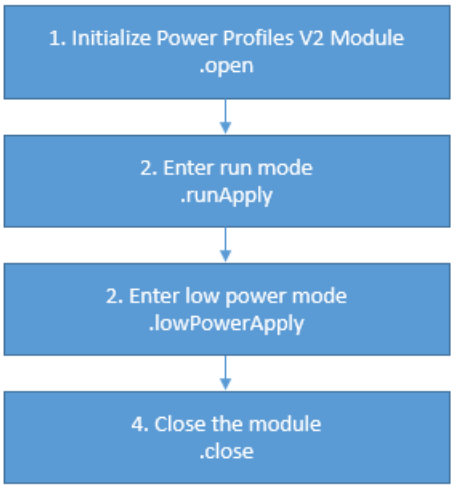


Figure 228: Flow Diagram of a Typical Power Profiles V2 Framework Module Application

4.1.26 SPI Framework

The SPI Framework module provides set of ThreadX-aware framework APIs and is implemented on SF_SPI. The SPI Framework module handles the integration and synchronization of multiple SPI peripherals on an SPI bus, including chip-select handling and its level activation. With the SPI Framework, one or more SPI buses can be created and multiple SPI peripherals can be connected to the SPI bus. The SPI Framework module uses a single interface to access both SCI SPI and RSPI drivers. The SPI Framework module uses the SCI and RSPI peripherals on the Synergy MCU.

4.1.26.1 SPI Framework Module Features

The SPI Framework module uses either the SCI in SPI mode (together with the SCI common lower-level modules) or the RSPI lower-level driver module to communicate with the SPI peripherals on the Synergy microcontroller.

- Supports multiple devices on a bus
- Provides high-level APIs for initialization, transfers, and closing the module
- Supports synchronized transfers
- Supports chip-select operations
- Supports bus-locking

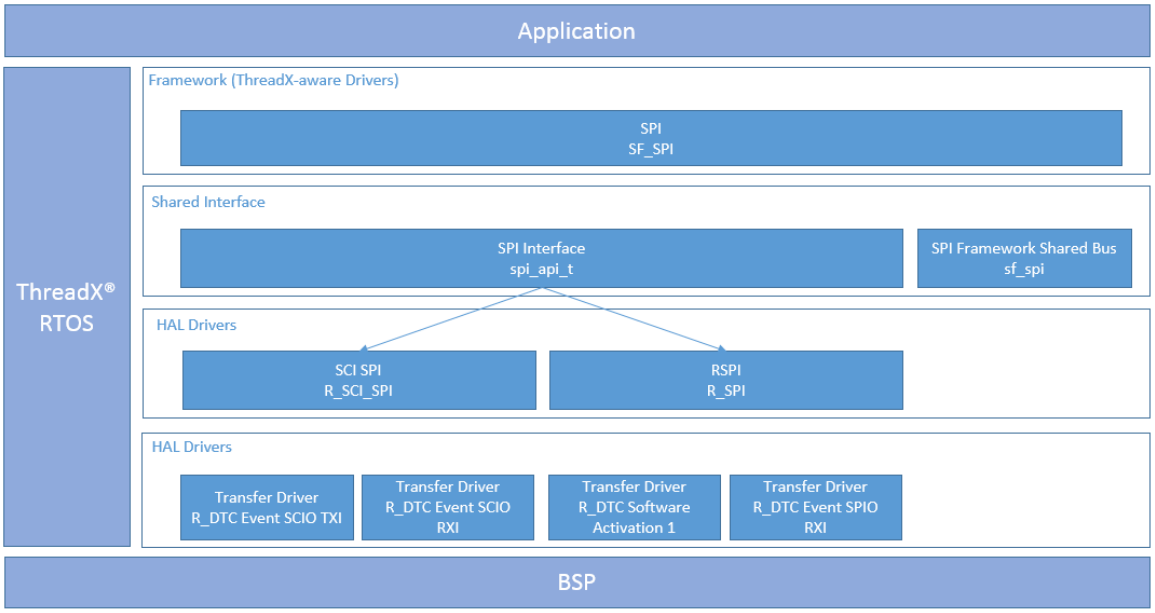


Figure 229: SPI Framework Module Block Diagram

4.1.26.2 SPI Framework Module APIs Overview

The SPI Framework module defines APIs for opening, closing, reading, writing, and other useful functions. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

SPI Framework Module API Summary

Function Name	Example API Call and Description
.open	g_sf_spi_device.p_api->open(g_sf_spi_device.p_cntl, g_sf_spi_device.p_cfg); Open a designated SPI device on a bus.
.read	g_sf_spi_device.p_api->read(g_sf_spi_device.p_cntl, dst8, length, SPI_BIT_WIDTH_8_BITS, TX_WAIT_FOREVER); Receive data from SPI device.
.write	g_sf_spi_device.p_api->write(g_sf_spi_device.p_cntl, src8, length, SPI_BIT_WIDTH_8_BITS, TX_WAIT_FOREVER); Transmit data to SPI device.
.writeRead	g_sf_spi_device.p_api->writeRead (g_sf_spi_device.p_cntl, &source, &destination, length, SPI_BIT_WIDTH_8_BITS, TX_WAIT_FOREVER); Simultaneously transmits data to an SPI device while receiving data from an SPI device (full duplex). The writeread API gets a mutex object, handles the SPI data transmission at SPI HAL layer, and receives data from the SPI HAL layer. The API uses the event flag wait to synchronize to completion of data transfer.
.close	g_sf_spi_device.p_api->close(g_sf_spi_device.p_cntl); Disable the SPI device designated by the control handle and close the RTOS services used by the bus, if no devices are connected to the bus. This function removes power to the SPI channel designated by the handle and disables the associated interrupts.

Function Name	Example API Call and Description
.lock	g_sf_spi_device.p_api->lock(g_sf_spi_device.p_cntl); Lock the bus for a device. The locking allows devices to reserve a bus to themselves for a given period of time (such as between lock and unlock). This allows devices to complete several reads and writes on the bus without an interrupt.
.unlock	g_sf_spi_device.p_api->unlock(g_sf_spi_device.p_cntl); Unlock the bus for a particular device and make the bus usable for other devices.
.versionGet	g_sf_spi_device.p_api->versionGet(&version); Retrieve the API version with the version pointer.

NOTE: Details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the *SSP User's Manual* API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function completed successfully
SSP_ERR_INVALID_MODE	Invalid mode
SSP_ERR_INVALID_CHANNEL	Invalid channel
SSP_ERR_IN_USE	In-use error
SSP_ERR_INVALID_ARGUMENT	Invalid argument
SSP_ERR_QUEUE_UNAVAILABLE	Queue unavailable
SSP_ERR_INVALID_POINTER	Invalid pointer
SSP_ERR_INTERNAL	Internal error
SSP_ERR_TRANSFER_ABORTED	Transfer aborted
SSP_ERR_MODE_FAULT	Mode fault

Name	Description
SSP_ERR_READ_OVF	Read overflow
SSP_ERR_PARITY	Parity error
SSP_ERR_OVERRUN	Overrun error
SSP_ERR_UNDEF	Unknown error
SSP_ERR_TIMEOUT	Timeout error
SSP_ERR_NOT_OPEN	Device not opened

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual* API References for the associated module for a definition of all relevant status return values.

4.1.26.3 SPI Framework Module Operational Overview

The SPI Framework module complies with the layered-driver architecture of the SSP. It uses either the SCI on SPI module or the RSPI module to communicate with the SPI peripherals on the Synergy microcontroller.

The framework-driver architecture uses a "bus" and "device on bus" architecture. Only one device is configured to the lower level at a time, and remaining devices are reconfigured upon read or write operation if required. The Device can be reconfigured only when the bus is not locked. Every device is linked to the bus to which it will be connected. The user must configure the bus, the framework, and the lower-level module for each device connecting to the bus. The user can add the existing framework shared-bus when configuring multiple devices on the same bus. Each SPI framework must be configured with a unique name in the ISDE configurator.

A common start and stop procedure is used for all SPI data-transfer operations (read, write and writeRead). During the start process, the Framework module checks whether reconfiguration is required. Chip select is asserted during the transfer-start process and de-asserted during the transfer-end process if bus is not locked. The user must configure the chip-select IO pin and chip-select active level.

The SPI Framework module supports the bus-locking functionality, meaning that the user can lock the bus for a given peripheral. The locking allows devices to reserve a bus to themselves for a given period of time (between lock and unlock.) This allows devices to complete several reads and writes on the bus without any intervention from other devices on the same bus, which is required in some instances. The chip select becomes active during lock and becomes inactive when unlocked. Writes and reads in between the lock and unlock do not alter the chip-select line.

SPI Framework Module Important Operational Notes and Limitations

- Multiple SPI devices can be configured to share a common bus. Once the SPI Framework bus module is configured, different SPI peripherals (devices) can be connected to that bus.
- For each SPI device connected to the bus, one SPI HAL module (new or shared) and one SPI Framework device module must be added.
- User-defined callback is not required as it has been internally taken care of by the framework.
- Setting the interrupts to different priority levels could result in improper operation.

- Refer to the MCU specification manual for identifying SPI bus compatibility. Device compatibility with the SPI bus is not checked in the framework hence incompatible SPI device may result in improper operation.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.26.4 Including the SPI Framework Module in an Application

This section describes how to include the SPI Framework module in an application using the SSP configurator.

NOTE: It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the *SSP User's Manual* to learn how to manage each of these important steps in creating SSP-based applications.

To add the SPI Framework module to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the SPI Framework is g_sf_spi_device0. This name can be changed in the associated **Properties** window.)

SPI Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_spi_device0 on sf_spi	**Threads**	**New Stack> Framework> Connectivity> SPI Framework Device on sf_spi**

When the SPI Framework module on sf_spi is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information will have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower level drivers; these are either optional or recommended (this is indicated in the block with the inclusion of this text.) If the addition of lower-level drivers is required, the module description will include **Add** in the text. Clicking on any Pink banded modules brings up the **New** icon and displays possible choices.

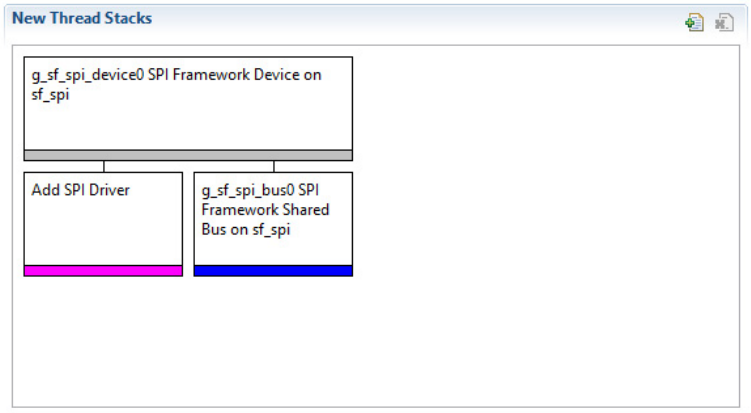


Figure 230: SPI Framework Module Stack (No Driver Selected)

Once the desired lower-level module has been selected, the module stack displays as following figures illustrate.

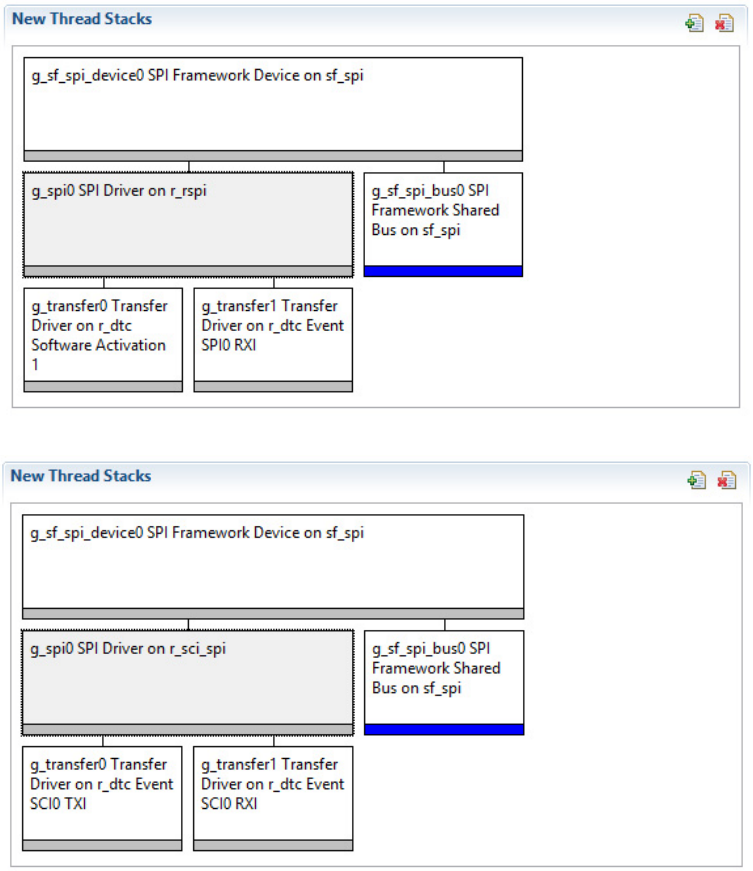


Figure 231: SPI Framework Implementations

4.1.26.5 Configuring the SPI Framework Module

The SPI Framework module must be configured by you for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes, and these are identified with a lock icon for the 'locked' property in the Properties** window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties** tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties** window of the associated module. Simply select the indicated module and then view the **Properties** window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the **Properties** window in the ISDE indicate the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE and create the module and explore the property settings in parallel with looking over the configuration table settings in the following table. This helps to orient you, and can be a useful hands-on approach

to learning the ins and outs of developing with SSP.

Configuration Settings for the SPI Framework Module on sf_spi

Parameter	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Parameter checking setting determines if code is added to check that parameters are within appropriate limits
Name	g_sf_spi_device0	Module name
Chip Select Port	00 thru 11 (Default: 00)	Select GPIO port used for the chip select.
Chip Select Pin	00 thru 15 (Default: 00)	Select GPIO pin used for the chip select.
Chip Select Active Level	Low, High (Default: Low)	Polarity of the chip select signal, active high or low.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, values other than the defaults for stack modules can be desirable. For example, it might be useful to select different chip-select GPIOs or levels. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

NOTE: Most of the property settings for modules are fairly intuitive and usually can be determined by inspection of the associated **Properties** window from the SSP configurator.

SPI Framework Module Configuration Settings for Lower-Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module:

Configuration Settings for the SPI HAL Module on r_rspi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Parameter checking setting determines if code is added to check that parameters are within appropriate limits
Name	g_spi0	Module name
Channel	0	Channel number
Operating Mode	Master, Slave (Default: Master)	Operating mode selection
Clock Phase	Data sampling on odd edge/ data variation on edge, Data sampling on even edge, data variation on odd edge (Default: odd/even)	Clock phase selection
Clock Polarity	Low when idle, high when idle (Default: Low when idle)	Clock polarity selection
Mode Fault Error	Enable, Disable (Default: Disable)	Mode fault error selection
Bit Order	MSB First, LSB First (Default: MSB First)	Bit order selection
Bitrate	500000	Bit rate selection
Callback	NULL	Callback function name
SPI Mode	SPI Operation, Clock Synchronous operation (Default: SPI Operation)	SPI mode selection
SPI Communication Mode	Full Duplex, Transmit Only (Default: Full Duplex)	SPI communication mode selection

ISDE Property	Value	Description
Slave Select Polarity (SSL0)	Active Low, Active High (Default: Active Low)	Slave select polarity selection 0
Slave Select Polarity (SSL1)	Active Low, Active High (Default: Active Low)	Slave select polarity selection 1
Slave Select Polarity (SSL2)	Active Low, Active High (Default: Active Low)	Slave select polarity selection 2
Slave Select Polarity (SSL3)	Active Low, Active High (Default: Active Low)	Slave select polarity selection 3
Select Loopback 1	Normal, Inverted (Default: Normal)	Loopback 1 selection
Select Loopback 2	Normal, Inverted (Default: Normal)	Loopback 2 selection
Enable MOSI Idle	Enable, Disable (Default: Disable)	Enable MOSI idle selection
MOSI Idle State	MOSI Low, MOSI High (Default: MOSI Low)	Enable MOSI idle state selection
Enable Parity	Enable, Disable (Default: Disable)	Enable parity selection
Parity Mode	Parity Odd, Parity Even (Default: Parity Odd)	Enable parity mode selection

ISDE Property	Value	Description
Select SSL (Slave Select)	SSL0, SSL1, SSL2, SSL3 (Default: SSL0)	Select SSL selection
Select SSL Level After Transfer	SSL Level Keep, SSL Level Do Not Keep (Default: Do Not Keep)	Select SSL level after transfer selection
Clock Delay Enable	Enable, Disable (Default: Disable)	Clock delay enable selection
Clock Delay Count	Clock Delay 1 RSPCK thru Clock Delay 8 RSPCK (Default: Clock Delay 1 RSPCK)	Clock delay count selection
SSL Negation Delay Enable	Enable, Disable (Default: Disable)	SSL Negation Delay Enable selection
Negation Delay Count	Negation Delay 1 RSPCK thru Negation Delay 8 RSPCK (Default: Negation Delay 1 RSPCK)	Negation Delay Count selection
Next Access Delay Enable	Enable, Disable (Default: Disable)	Next Access Delay Enable selection
Next Access Delay Count	Next Access Delay 1 RSPCK thru Next Access Delay 8 RSPCK (Default: Next Access Delay 1 RSPCK)	Next Access Delay Count selection

ISDE Property	Value	Description
Receive Interrupt Priority	Priority 0 (highest), 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 (lowest, not valid if using Thread X), Disabled (Default: Priority 2)	Receive Interrupt Priority selection.
Transmit Interrupt Priority	Priority 0 (highest), 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 (lowest, not valid if using Thread X), Disabled (Default: Priority 2)	Transmit Interrupt Priority selection.
Error Interrupt Priority	Priority 0 (highest), 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 (lowest, not valid if using Thread X), Disabled (Default: Priority 2)	Error Interrupt Priority selection.

NOTE: The example settings and defaults are for a project using the Synergy SK-S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the DTC HAL Module on r_dtc Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled (Default: Disabled)	Software start selection
Link section to keep DTC vector table	.ssp_dtc_vector_table	Link section to keep DTC vector table selection
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	4 Bytes	Transfer size selection

ISDE Property	Value	Description
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Software Activation 1	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection

NOTE: The example settings and defaults are for a project using the Synergy S7 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the DTC HAL Module on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled (Default: Disabled)	Software start selection
Link section to keep DTC vector table	.ssp_dtc_vector_table	Link section to keep DTC vector table selection
Name	g_transfer0	Module name
Mode	Normal	Mode selection

ISDE Property	Value	Description
Transfer Size	1 Bytes	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode)	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SPI0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SPI Framework Shared Bus on sf_spi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Selects if code for parameter checking is to be included in the build
Name	g_sf_spi_bus0	Module name
SPI Implementation	SCI SPI	SPI implementation selection
Channel	0	Channel selection

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SPI HAL Module on r_sci_spi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Parameter checking setting determines if code is added to check that parameters are within appropriate limits
Name	g_spi0	Module name
Channel	0	Channel number
Operating Mode	Master, Slave (Default: Master)	Operating mode selection
Clock Phase	Data sampling on odd edge/ data variation on even edge, Data sampling on even edge, data variation on odd edge (Default: odd/even)	Clock phase selection
Clock Polarity	Low when idle, high when idle (Default: Low when idle)	Clock polarity selection
Mode Fault Error	Enable, Disable (Default: Disable)	Mode fault error selection
Bit Order	MSB First, LSB First (Default: MSB First)	Bit order selection
Bitrate	100000	Bit rate selection
Callback	NULL	Callback function name
Receive Interrupt Priority	Priority 0 (highest), 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 (lowest, not valid if using Thread X), Disabled (Default: Priority 2)	Receive Interrupt Priority selection.
Transmit Interrupt Priority	Priority 0 (highest), 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 (lowest, not valid if using Thread X), Disabled (Default: Priority 2)	Transmit Interrupt Priority selection.

ISDE Property	Value	Description
Transmit End Interrupt Priority	Priority 0 (highest), 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 (lowest, not valid if using Thread X), Disabled (Default: Priority 2)	Transmit End Interrupt Priority selection.
Error Interrupt Priority	Priority 0 (highest), 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 (lowest, not valid if using Thread X), Disabled (Default: Priority 2)	Error Interrupt Priority selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

DTC HAL Module on r_dtc Event SCI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled (Default: Disabled)	Software start selection
Link section to keep DTC vector table	.ssp_dtc_vector_table	Link section to keep DTC vector table selection
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection

ISDE Property	Value	Description
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 TXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

DTC HAL Module on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled (Default: Disabled)	Software start selection
Link section to keep DTC vector table	.ssp_dtc_vector_table	Link section to keep DTC vector table selection
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	1 Bytes	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection

ISDE Property	Value	Description
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SPI0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

SPI Framework Module Clock Configuration

The SPI peripheral module uses PCLKB as its clock source. The PCLKB frequency is set by using the SSP configurator clock tab prior to a build, or by using the CGC Interface at run-time.

SPI Framework Module Pin Configuration

The SPI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the SPI pins.

NOTE: For some peripherals, the operation mode selection determines what peripheral signals are available and thus what MCU pins are required.

Pin Selection for the SPI Framework Module

Resource	ISDE Tab	Pin selection Sequence
SCI	Pins	Select **Peripherals > Connectivity: SCI> SCI1**
RSPI	Pins	Select **Peripherals > Connectivity: SPI > SPI0**

NOTE: The top selection sequence assumes SCI1 and SPI0 are the desired hardware targets for the driver and the bottom

selection sequence assumes SPI0 is the desired target.

Pin Configuration Settings for the SPI Framework Module

Property	Settings	Description
Operation Mode	Disabled, Asynchronous UART, Synchronous UART, Simple I2C, Simple SPI, SmartCard (Default: Disabled)	Select Simple SPI as the Operation Mode for SPI on SCI
CTS0_RTS0_SS0	None, P103, P413 (Default: None)	SS0 Pin selection
RXD0_SCL0_MISO0	None, P100, P410 (Default: None)	MISO0 Pin selection
SCK0	None, P102, P412 (Default: None)	SCK0 Pin selection
TXD1_SDA1_MOSI0	None, P213, P709 (Default: None)	MOSI0 Pin selection

Other Settings for the SPI Framework Module

If external chip selects are being used, configure the chip select pins as GPIO outputs.

4.1.26.6 Using the SPI Framework Module in an Application

The typical steps in using the SPI Framework module in an application are:

1) Initialize the SPI Framework module using the open API
2) Lock the bus for continuous transfer using the lock API if needed
3) Read data using read API
4) Write data using write API
5) Write/read data simultaneously using writeRead API
6) Unlock the bus from continuous transfer using the unlock API if needed

7) Close the module using the close API

These common steps are illustrated in a typical operational flow diagram in the figure below:

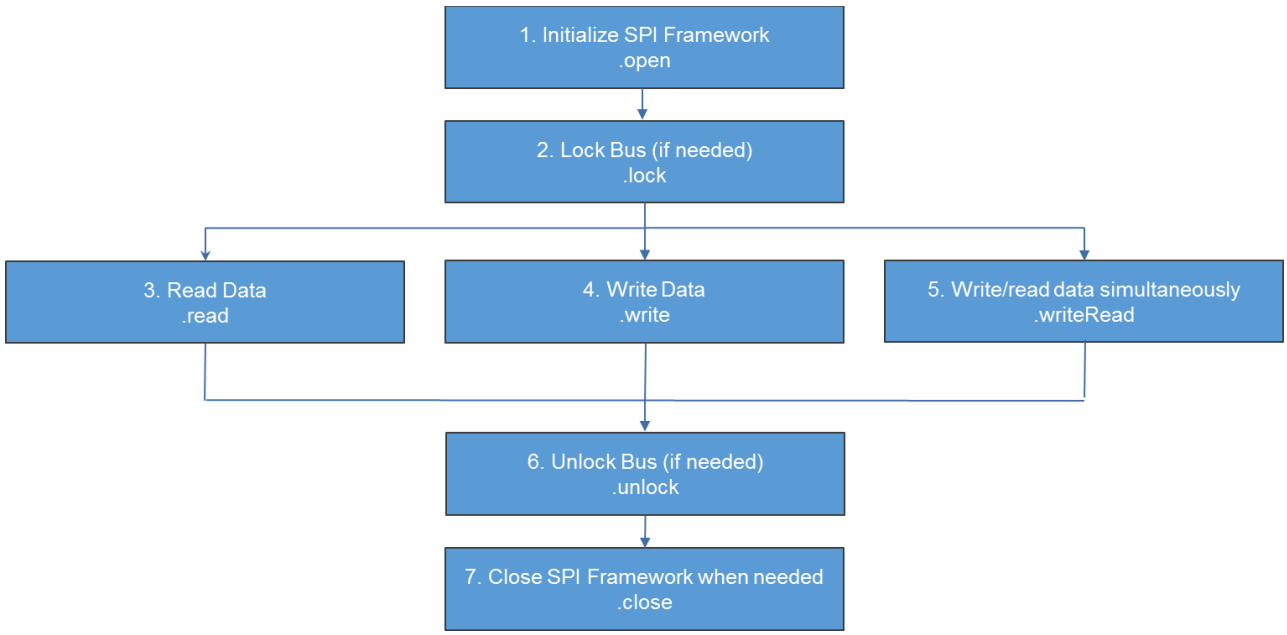


Figure 232: Flow Diagram of a Typical SPI Framework Application

4.1.27 Thread Monitor Framework

The Thread Monitor Framework provides high-level APIs for thread monitor framework applications and is implemented on sf_thread_monitor. The framework configures the watchdog timer (WDT) or independent watchdog timer (IWDT). The Thread Monitor Framework uses WDT or IWDT peripherals on the Synergy MCU device.

4.1.27.1 Thread Monitor Framework Module Features

The Thread Monitor Framework supports the following features:

- The Thread Monitor Framework interface monitors RTOS threads using a watchdog timer. The Thread Monitor forces a watchdog reset of the microcontroller when any of the monitored threads do not behave as expected.
- The Thread Monitor is designed to support any Synergy device with either a WDT or IWDT peripheral, and a HAL module with no changes to the API.
- In profiling mode, the minimum and maximum counter values for registered threads can be determined. When in profiling mode, the watchdog timer is always refreshed and does not reset the device.
- Both the WDT and IWDT HAL modules are supported by this framework module.

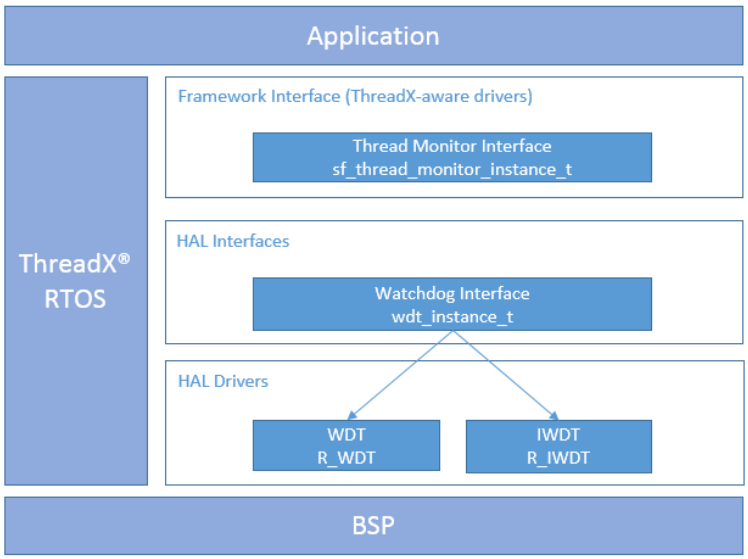


Figure 233: Thread Monitor Framework Module Block Diagram

4.1.27.2 Thread Monitor Framework Module APIs Overview

The Thread Monitor Framework defines API's for opening and closing the framework and registering and unregistering threads for monitoring. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of return status values follows the API summary table.

Console Framework Module API Summary

Function Name	Example API Call and Definition
open	g_sf_thread_monitor.p_api->open (g_sf_thread_monitor.p_ctrl,g_sf_thread_monitor.p_cfg); Configures the WDT or IWDT module. From the configuration data, the timeout period of the WDT/IWDT is determined. A thread created to monitor registered threads.
close	g_sf_thread_monitor.p_api->close (g_sf_thread_monitor.p_ctrl); Suspends the thread monitoring thread. The watchdog peripheral no longer refreshes.

Function Name	Example API Call and Definition
threadRegister	g_sf_thread_monitor.p_api-> threadRegister (g_sf_thread_monitor.p_ctrl, &p_min_max_struct); Registers a thread for monitoring.
threadUnregister	g_sf_thread_monitor.p_api-> threadUnregister (g_sf_thread_monitor.p_ctrl); Removes a thread from monitoring.
countIncrement	g_sf_thread_monitor.p_api-> countIncrement (g_sf_thread_monitor.p_ctrl); Safely increments a monitored thread's count value.
versionGet	g_sf_thread_monitor.p_api-> versionGet(&version); Retrieves the API version and stores it in the version pointer.

NOTE: For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the *SSP User's Manual***API References** for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_ASSERTION	Pointer is null.
SSP_ERR_IN_USE	Thread monitor has already been opened.
SSP_ERR_INVALID_MODE	Low-level watchdog peripheral returns an error when opened.
SSP_ERR_UNSUPPORTED	Data structure could not be allocated.
SSP_ERR_INVALID_ARGUMENT	One or more configuration options is invalid for the low-level driver.

Name	Description
SSP_ERR_NOT_OPEN	SF_THREAD_MONITOR_Open has either not been called or it was not called successfully.
SSP_ERR_INSUFFICIENT_SPACE	Not enough entries in threads-to-be-monitored array to add to this thread. Increases value of THREAD_MONITOR_CFG_MAX_NUMBER_OF_THREADS in sf_thread_moinitor_cfg.h

NOTE: Lower-level drivers may return common error codes. See *SSP User's Manual***API References** for the associated module for a definition of all relevant status return values.

4.1.27.3 Thread Monitor Framework Module Operational Overview

The Thread Monitor works as follows: A thread registers a counter variable with the Thread Monitor along with minimum and maximum expected values for this counter variable. The thread which is monitored increments the counter variable while it runs. At a period of half the watchdog timeout period, the Thread Monitor checks the counter variables of registered threads. If any fall outside of the minimum and maximum values, the watchdog timer is allowed to reset the microcontroller. If all fall within their expected range, the watchdog timer is refreshed and the counter variables are cleared to zero.

Thread Monitor Framework Module Important Operational Notes and Limitations

The following figure shows a flowchart for the operation of the Thread Monitor Framework module.

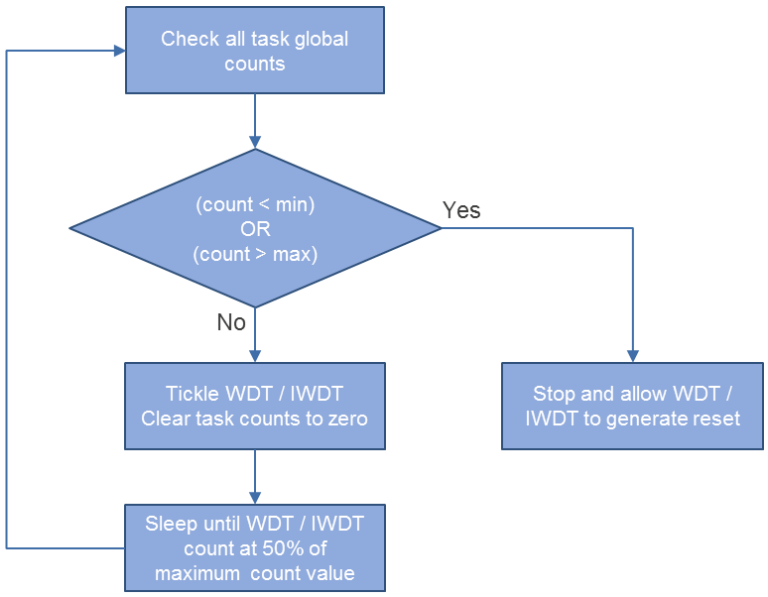


Figure 234: Thread Monitor Framework Operation Diagram

The following figure shows when the WDT/IWDT refreshes. Note that the valid refresh period is the central 50% of the count period, 25% on either side of the 50% count value.

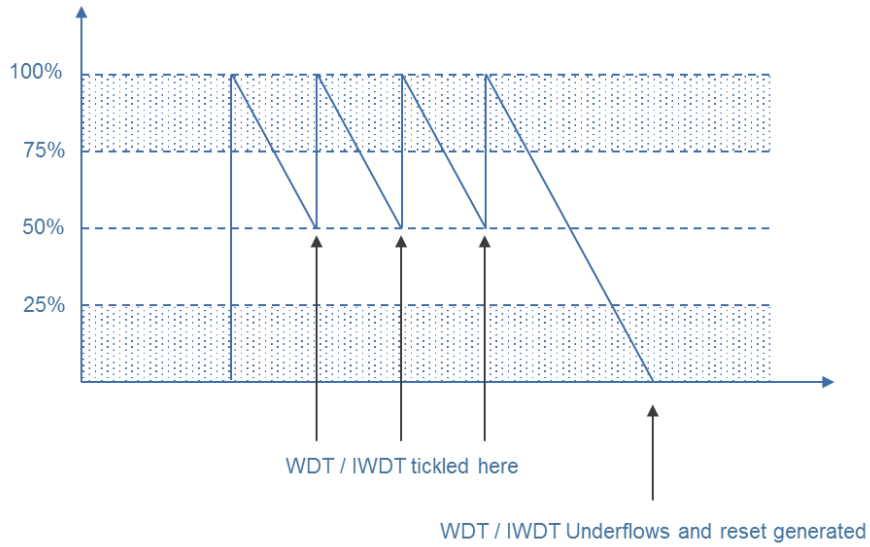


Figure 235: WDT/IWDT Operation Diagram

- The IWDT has its own clock source to improve safety.
- The WDT can be started from the application.
- When the thread is not executing sleep mode, set the stop-control property of the WDT to >WDT Count Enabled in Low Power Mode. When the thread is not executing (asleep), the ThreadX® executes a WFI instruction effectively putting the device into soft sleep, which causes the WDT to stop counting.

NOTE: Do not open or refresh the WDT in a monitored thread file; it is done automatically by the Thread Monitor Framework.

- Internally, the Thread Monitor Framework runs at the rate of half of the watchdog timer reset period. This rate ensures the Thread Monitor Framework runs at 50% of the watchdog count value—well within the valid refresh window. The Thread Monitor calculates the reset period internally by querying the lower-level watchdog driver.
- The Thread Monitor Framework has a close API call. When WDT and IWDT are being used, it is not possible to stop them. If the Thread Monitor Framework is closed, some other provision for refreshing the watchdog must be made or the device resets.
- Debugging mode-support is required when running with a JLink on some devices; WDT/IWDT does not count when using JLink debugging hardware. The Thread Monitor Framework thread typically synchronizes to the WDT/IWDT counter, but skips this synchronization step when it is running with JLink.
- Assign a high priority (low number in ThreadX) to the Thread Monitor Framework thread; any delays in running the Thread Monitor Framework could refresh the watchdog outside of the valid refresh window, causing the microcontroller to reset. The Thread Monitor Framework thread does not run for long and does not impact the performance of the system.

- Refer to the latest *SSP Release Notes* for any additional operational limitations for this module.

4.1.27.4 Including the Thread Monitor Framework Module in an Application

This section describes how to include the Thread Monitor Framework in an application using the SSP configurator.

NOTE: It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these tasks, refer to the first few chapters of the *SSP User's Manual* to learn how to manage each of these important steps in creating SSP-based applications.

To add the Thread Monitor Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Thread Monitor Framework is g_thread_monitor0. This name can be changed in the associated Properties window.)

Thread Monitor Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_thread_monitor0 Thread Monitor Framework	Threads	New Stack> Framework> Services> Thread Monitor Framework on sf_thread_monitor

When the Thread Monitor Framework on sf_thread_monitor is added to the thread stack as shown in the following figure, the configurator reports (via the **Add WDT** block) that at least one watch dog timer is required to complete the stack. Any lower-level modules that need additional configuration information have box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended (this is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description includes **Add** in the text. Clicking on any Pink banded modules brings up the **New** icon and displays possible choices.

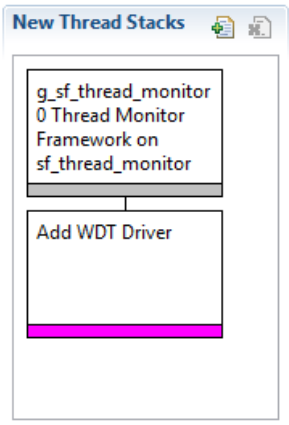


Figure 236: Thread Monitor Framework Module Stack

4.1.27.5 Configuring the Thread Monitor Framework Module

The Thread Monitor Framework module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only those properties that can be changed without causing conflicts are available for modification. Other properties are locked and are not available for changes, and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP Configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available.

Also, note that the interrupt priorities listed in the Properties window in the ISDE includes the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not in the following configuration properties table, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel while looking over the following configuration table settings. This helps to orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Thread Monitor Framework Module on sf_thread_monitor

Parameter	Value	Description
Parameter Checking	Enabled, Disabled, BSP (Default: BSP)	Controls whether to include code for API parameter checking.
Maximum Number of Monitored Threads	5	Maximum number of threads that can be monitored.
Name	g_sf_thread_monitor0	Module name.
Profiling Mode	Enabled, Disabled, (Default: Disabled)	Whether profiling mode should be enabled.
Thread Monitor Thread Priority	1	Priority of thread monitor internal thread.

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Settings other than the defaults for lower-level modules can be desirable in some cases. For example, it might be useful to set the maximum number of monitored threads to a desired value. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

NOTE: Most of the property settings for modules are fairly intuitive and usually can be determined by inspection of the associated Properties window from the SSP configurator.

Configuration Settings for the Thread Monitor Framework Low Level Modules

Typically, only a small number of settings must be modified from the default for lower-level modules as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following tables identify all the settings within the properties section for the module.

Configuration Settings for the Independent Watchdog Timer on r_iwdt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Include parameter checking code
Name	g_wdt0	Module Name
NMI Callback	NULL	Callback Name

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Watchdog Timer on r_wdt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Include parameter checking code
Name	g_wdt0	Module Name
Start Mode	Register, Auto (Default: Register)	Configures the start mode as register start or auto-start.
Start Watchdog After Configuration	True, False (Default: True)	Controls whether WDT is started during initialization
Timeout	1024 cycles, 4096 cycles, 8192 cycles, 16384 cycles (Default: 16384 cycles)	WDT timeout period.
Clock Division Ratio	PCLK/4, PCLK/64, PCLK/128, PCLK/512, PCLK/2048, PCLK/8192 (Default: PCLK/8192)	WDT clock divider

ISDE Property	Value	Description
Window Start Position	100% (Window Position Not Specified); 75%, 50%, 25% (Default: 100%)	Permitted refresh period start position.
Window End Position	0% (Window Position Not Specified; 25%, 50%, 75% (Default: 0%)	Permitted refresh period end position.
Reset Control	Reset Output, NMI Generated (Default: Reset Output)	Select whether WDT should reset the MCU or generate an NMI.
Stop Control	WDT Count Enabled in Low Power Mode, WDT Count Disabled in Low Power Mode (Default: Disabled)	Select whether the WDT should stop counting in low power modes.
NMI Callback	NULL	Callback. A user callback function can be registered in open. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the IRQn triggers. ATTENTION: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Most of the property settings for modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Thread Monitor Framework Module Clock Configuration

Use the ISDE to configure the WDT clock using the **Clocks** tab.

The WDT is initially based on the PCLKB frequency. Set the PCLKB frequency in e^2 studio Integrated Solution Developer Environment (ISDE) using the clock configurator Configuring Clocks or the CGC Interface at run time.

With PCLKB running at 60 MHz, the WDT maximum timeout period is approximately 2.24 seconds.

The IWDT clock runs at 15 kHz resulting in a maximum possible timeout period of just under 35 seconds.

Thread Monitor Framework Module Pin Configuration

The Thread Monitor Framework does not require any pins for its operation.

Threads for the Thread Monitor Framework Module Application

Any thread monitored by the Thread Monitor Framework must be instrumented to work with the monitoring module. When a thread to be monitored is registered with the Thread Monitor, the expected minimum and maximum count values are passed via a pointer to a structure of type sf_thread_monitor_counter_min_max_t containing the values.

The thread's counter and minimum and maximum values must be registered with the Thread Monitor Framework by calling g_sf_thread_monitor.p_api->threadRegister().

Each time round the monitored thread's loop, the counter value should be updated by calling g_sf_thread_monitor.p_api->countIncrement().

Other Thread Monitor Framework Module Settings

The Thread Monitor Framework does not use any interrupts; the WDT/IWDT must be configured to generate a reset.

4.1.27.6 Using the Thread Monitor Framework Module in an Application

The typical steps in using the Thread Monitor Framework in an application are:

1) Initialize the Thread Monitor using the open API
2) Register thread that needs to be monitored with the threadRegister API
3) Use the countIncrement API to increment the count every time the registered thread is executed.
4) Unregister the thread with the threadUnregister API when there is no longer a need to monitor the registered thread.
5) Close the Thread Monitor with the close API (only if the Thread Monitor is no longer required in an application).

Often, the increment count step is repeated periodically. The following figure illustrates these common steps in a typical operational flow.

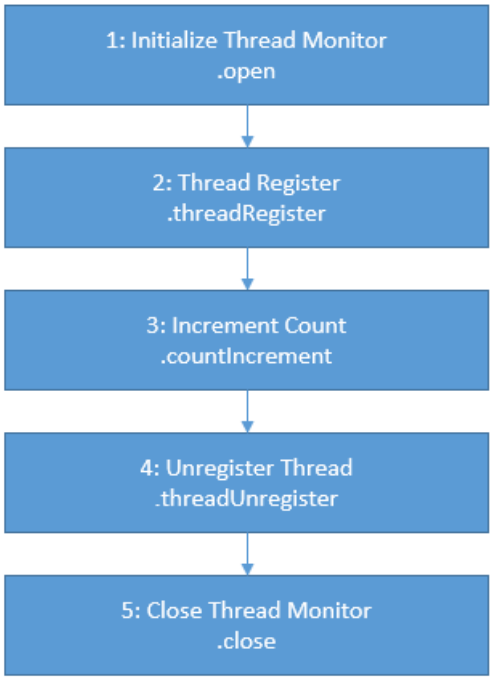


Figure 237: Typical Thread Monitor Framework Module Application

4.1.28 Touch Panel Framework

The Touch Panel Framework module provides a high-level API for reading messages from a touch controller and is implemented using an I2C port. The Touch Panel Framework module uses the IIC or SCI peripherals on the Synergy MCU.

4.1.28.1 Touch Panel Framework Module Features

The Touch Panel Framework supports the following features:

- Reads data from a touch controller and publishes touch messages to any queue subscribed to touch events in the messaging framework
- Provides position data (X and Y coordinates)
- Provides the touch event type (down, up, move, hold, or invalid)
- Supports I2C-based touch panel implementations with external interrupts

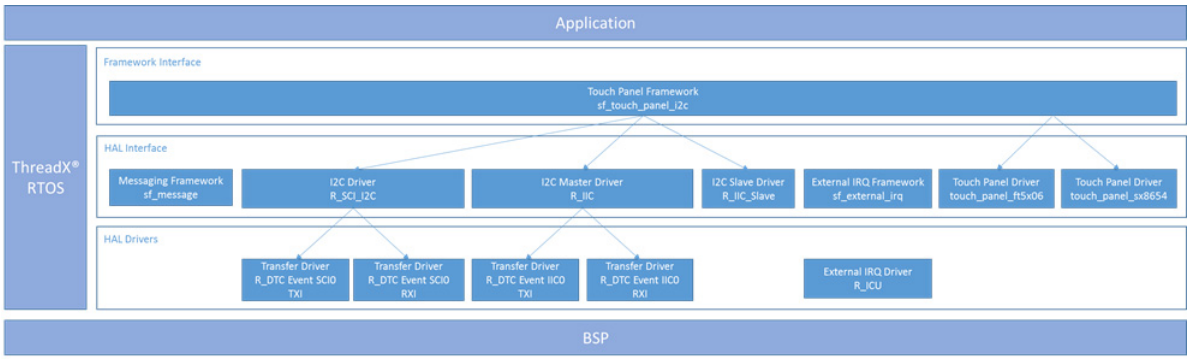


Figure 238: Touch Panel Framework Module Block Diagram

4.1.28.2 Touch Panel Framework Module APIs Overview

The Touch Panel Framework module defines APIs for key functions such as opening, calibrating, starting, stopping or closing. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows.

Touch Panel Framework Module API Summary

Function Name	Example API Call and Description
open	g_sf_touch_panel_i2c0.p_api->open(g_sf_touch_panel_i2c0.p_ctrl, g_sf_touch_panel_i2c0.p_cfg); Create required RTOS objects, call lower level module for hardware specific initialization, and create a thread to post touch data to a message queue.

Function Name	Example API Call and Description
calibrate	g_sf_touch_panel_i2c0.p_api->calibrate(g_sf_touch_panel_i2c0.p_ctrl, &expected, &actual, timeout); Begin calibration routine based on provided expected coordinates. Returns SSP_SUCCESS only if the tolerance is longer than the distance from the expected touch point to the actual touch point (using the followng formula: p_calibrate->tolerance_pixels \^ 2 > (p_calibrate->x - x_measured) \^ 2 + (p_calibrate->y - y_measured) \^ 2
start	g_sf_touch_panel_i2c0.p_api->start(g_sf_touch_panel_i2c0.p_ctrl); Start scanning for touch events.
stop	g_sf_touch_panel_i2c0.p_api->stop (g_sf_touch_panel_i2c0.p_ctrl); Stop scanning for touch events.
reset	g_sf_touch_panel_i2c0.p_api->reset(g_sf_touch_panel_i2c0.p_ctrl); Reset touch controller if reset pin is provided, and resets the I2C bus.
close	g_sf_touch_panel_i2c0.p_api->close(g_sf_touch_panel_i2c0.p_ctrl); Terminate touch thread and close channel at HAL layer.
versionGet	g_sf_touch_panel_i2c0.p_api->versionGet(&version); Retrieves API version and stores it in the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API call successful.
SSP_ERR_ASSERTION	A pointer parameter was NULL, or a lower level driver reported this error.
SSP_ERR_INTERNAL	The touch panel thread or event flags could not be created, or a lower level driver reported this error.
SSP_ERR_CALIBRATE_FAILED	Actual touch value was not in expected range.
SSP_ERR_NOT_OPEN	Touch panel is not configured. Call SF_TOUCH_PANEL_I2C_Open.
SSP_ERR_IN_USE	Mutex was not available, or a lower level driver reported this error.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.1.28.3 Touch Panel Framework Module Operational Overview

The Touch Panel Framework module reads data from a touch controller and publishes touch messages to any queue subscribed to touch events in the messaging framework. The touch event contains position data (X and Y coordinates) and the touch event type (down, up, move, hold, or invalid). The Touch Panel Framework supports I2C-based touch panel implementations with external interrupts. The Touch Panel Framework creates a thread internally to read the touch controller.

The typical flow when using the Touch Framework is:

1) Pend on a touch message.
2) Parse the touch message.

Touch Panel Framework Module Important Operational Notes and Limitations

Initial Configuration

The e^2 studio ISDE automatically generates the Touch Event Class and the New Data Event in the Messaging tab of the Project Configurator when you add the Touch Panel Framework to your project. Use the following steps to configure Touch Event message subscriber(s) in your project:

- Select the Touch Event Class on the Event Classed pane in e^2^ studio Synergy Configurator Messaging tab.
- Check the check box of subscriber thread on the Touch Subscribers Pane in Messaging tab. (If no thread is shown in the pane, add the thread by clicking the "New" icon located at the top right of the Touch Subscriber pane.)

Create a Custom Touch Panel Chip Driver

To create your custom Touch Chip Driver, refer to the existing touch driver code in SSP which is located at synergy/ssp_supplemental/ touch_drivers (Note that this directory will only be visible if any existing touch chip driver is selected in synergy configurator, to do this go to synergy configurator select New > Framework > Input > Touch Panel Framework on sf_touch_panel_i2c> Add Touch Driver and select any existing touch driver for reference and generate project content), the following instructions and pseudo code to connect it to the Touch Panel Framework:

- Implement Touch Chip Driver Instance to enable the Touch Panel Framework to attach the driver.
- Implement the payloadGet function to obtain touch events and touch coordinates from a Touch Panel Controller device through the I2C interface.
- Implement a reset function to reset a Touch Panel Controller device by an associated GPIO pin and I2C interface

Configure the Custom Touch Panel Chip Driver

To configure a custom touch chip driver to a project, use the following steps:

1) Create a **Synergy C** project.
2) Update existing touch driver XML for the custom touch driver by following these steps to modify any existing touch XML.A. Go to .module_descriptions folder under the project root folderB. Edit touch driver XML: Renesas##HAL Drivers##touch panel##touch_panel_i2c_ft5x06####<version> OR Renesas##HAL Drivers##touch panel##touch_panel_i2c_sx8654####<version>C. Change name of the instance structure in the **value** field with the name of custom driver instance structure.D. Add the new instance declaration after the property tabE. Rename all touch chip number in the XML with custom touch chip number.F. Save and close the XML file
3) Open project configurator and add touch panel framework, by selecting New > Framework > Input > Touch Panel Framework on sf_touch_panel_i2c (If the configurator was already opened, close and open it again).
4) Configure all components.
5) Add the custom touch panel chip driver to the **Add Touch Driver** box. (The custom touch driver open will be visible if XML is modified correctly as described above).
6) Generate Project Content and add new directory (for ex. touch_panel_i2c_xxxxxx) structure under synergy/ ssp_-supplemental/ touch_drivers, the directory structure will look like this: synergy/ ssp_supplemental/ touch_drivers/ touch_panel_i2c_xxxxxx (replace xxxxxx with touch controller part no).
7) Add the custom driver code into this directory (code created as described in the above section **Create a custom Touch Panel Chip Driver**, simply copy and paste to ssp_supplemental/ touch_drivers/ touch_panel_i2c_xxxxxx).
8) Exclude existing driver from build if any (right-click the .c file > Exclude from build… > Select all > OK).
9) Build the code.

- User callback is not available in the I2C-bus communication procedure.
- The reset API is only available after the following conditions are satisfied: (1) The stop API is called. (2) A touch event happened and the Touch Panel Framework posts a touch event message.
- The add-on directory for the Touch Panel driver was changed from renesas_sybd (in SSPv1.2.0-b.1) to ssp_supplemental (SSPv1.2.0). If both SSPv1.2.0-b.1 and SSPv1.2.0 are installed in the development environment, both touch panel modules will be selected in the component tab. In your project, exclude the renesas_sybd folder from the build. Right click on renesas_sybd folder -> Exclude from build->Select all->OK**.**
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.28.4 Including the Touch Panel Framework Module in an Application

This section describes how to include the Touch Panel Framework module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the "Getting Started Guide for SSP" listed in the References section at the end of this document to learn how to manage each of these important steps in creating SSP-based applications.

To add the Touch Panel Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Touch Panel Framework is sf_touch_panel_i2c. This name can be changed in the associated Properties window.)

Touch Panel Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_touch_panel_i2c0 Touch Panel Framework on sf_touch_panel_i2c	Threads	New Stack> Framework> Input> Touch Panel Framework on sf_touch_panel_i2c

When the Touch Panel Framework on sf_touch_panel_i2c is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower level modules. Any drivers that need additional configuration information will have box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower level drivers; these are either optional or recommended (this is indicated in the block with the inclusion of this text.) If the addition of lower level drivers is required, the module description will include "Add" in the text. Clicking on any Pink banded modules will bring up the "New" icon and then display the possible choices.

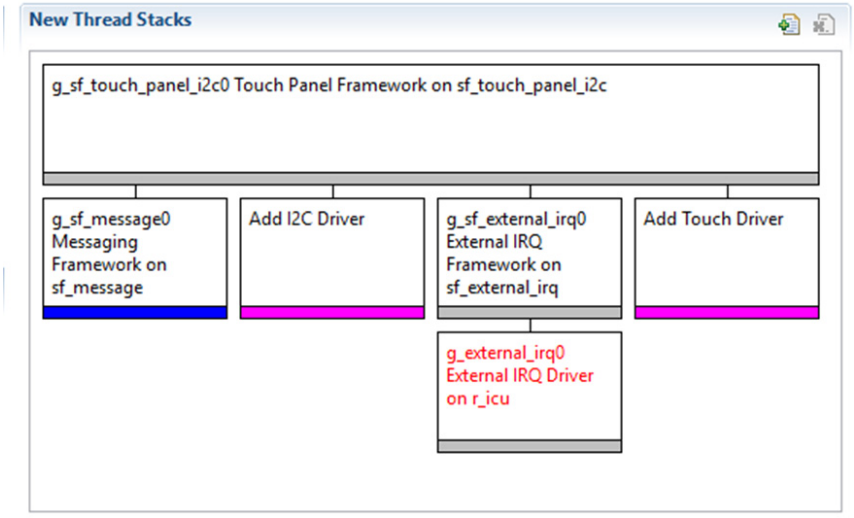


Figure 239: Touch Panel Framework Module Stack

4.1.28.5 Configuring the Touch Panel Framework Module

The Touch Panel Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify, by highlighting the block in red, any required configuration selections, such as Interrupts or operating modes, which must be configured for lower level modules, for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are **locked** and not available for changes and are identified with a lock icon for the **locked** property in the Property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous **manual** approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP Configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the Interrupt Priority. This configuration setting is available with the Properties window of the associated module. Simply select the indicated module and then view the properties window. The Interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the Interrupt Priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring Interrupt Priority levels.

NOTE: You may want to open your ISDE and create the Touch Panel Framework and explore the property settings in parallel with looking over the following Configuration Table Settings. This will help orient you and can be a useful **hands-on** approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Touch Panel Framework Module on sf_touch_panel_i2c

Parameter	Value	Description
Parameter Checking	BSP, Enabled, Disabled ;Default: BSP	Enable or disable the parameter checking.
Name	g_sf_touch_panel_i2c0	Module name.
Thread Priority	Default: 3	Thread priority selection.
Hsize Pixels	Default: 800	Hsize pixels selection.
Vsize Pixels	Default: 480	Vsize pixels selection.
Update Hz	Default: 10	Update hz selection.
Reset Pin	IOPORT_PORT_10_PIN_02	Reset pin selection.
Touch Event Class Instance Number	0	Touch event class instance number selection.
Touch Coordinate Rotation Angle (Clockwise)	0, 90 (select this if 'Screen Rotation Angle' in GUIX Port is '270'), 180, 270 (select this if 'Screen Rotation Angle' in GUIX Port is '90') Default: 0	Touch coordinate rotation angle selection.

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults can be desirable. For example, it might be useful to select different screen sizes. The configurable properties for the lower level stack modules are given in the following sections for completeness and as a reference.

NOTE: Most of the property settings for lower level modules are fairly intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the Touch Panel Framework Low Level Modules

Typically, only a small number of settings must be modified from the default for lower level drivers and these are indicated via the red text in the Thread Stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The Touch Panel Framework consists of the following low level drivers:

- Messaging Framework on sf_message
- I2C Driver: I2C Master Driver on riic or I2C Master Driver on r_sci_i2c
- External IRQ Framework on sf_external_irq
- Touch Panel Driver

The following table identifies all the settings for the lower level drivers within the properties section for the module.

Configuration for the Messaging Framework on sf_message

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Message Queue Depth (Total number of messages to be enqueued in a Message Queue)	16	Specify the size of Thread X Message Queue in bytes for Message Subscribers. This value is applied to all the Message Queues.
Name	g_sf_message0	The name of Messaging Framework module control block instance.

ISDE Property	Value	Description
Work memory size in bytes	2048	Specify the work memory size in bytes. Choosing a small number results a small number of buffers which can be allocated at the same time (You can confirm the total buffer number on: sf_message_ctrl_t::number_of_buffers). If the value is smaller than the peak number of messages to be posted at the same time, the Framework occurs a buffer allocation failure affecting system performance.
Pointer to subscriber list array	p_subscriber_lists	Specify the name of pointer to the Subscriber List array.
name of the block pool internally used in the messaging framework	sf_msg_blk_pool	The name of memory block memory the Framework creates in the control block. This parameter might be useful for debugging purpose but NULL can be specified for saving memory.

Configuration for the I2C Driver on r_sci_i2c

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.
Name	g_i2c0	Module name.
Channel	0 to 9 Default: 0	Specify the SCI channel to be used with this configuration. SCI has channels as follows: Series S7 : 0 1 2 3 4 5 6 7 8 9; Series S3 : 0 1 2 3 4 - - - - 9; Series S1 : 0 1 - - - - - - - 9 .
Rate	Standard, Fast-mode Default: Standard	Standard and Fast.
Slave Address	0x00	Address of the slave device.

ISDE Property	Value	Description
Address Mode	7-Bit, 10-Bit Default: 7-Bit	Only 7-bit addresses are currently supported.
SDA Output Delay (nano seconds)	300	SDA output delay in nanoseconds.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Enables bitrate modulation function.
Callback	NULL	A user callback function can be registered in open. If this callback function is provided, it will be called from the interrupt service routine (ISR) for each of the conditions defined in i2c_event_t. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection.

ISDE Property	Value	Description
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit end interrupt priority selection.

Configuration for the DTC HAL Module on r_dtc Event SCI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled / /Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled / /Default: Enabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection

ISDE Property	Value	Description
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 TXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection.

Configuration for the DTC HAL Module on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection

ISDE Property	Value	Description
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection.

Configuration for the I2C Master Driver on r_riic

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.
Name	g_i2c0	Module name.
Channel	0, 1, or 2	Specify the IIC channel to be used with this configuration.

ISDE Property	Value	Description
Rate	Standard, Fast-mode, Fast-mode Plus Default: Standard	Standard, Fast, and Fast-plus. (See IIC Rate Calculation.)
Slave Address	0x00	Set the address of the slave device the I2C master will be communicating with.
Address Mode	7-Bit, 10-Bit Default: 7-Bit	Only 7-bit addresses are currently supported.
Callback	NULL	A user callback function can be registered in open. If this callback function is provided, it will be called from the interrupt service routine (ISR) for each of the conditions defined in i2c_event_t. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection.

ISDE Property	Value	Description
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Error interrupt priority selection.

Configuration for the DTC HAL Module on r_dtc IIC0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Enabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection

ISDE Property	Value	Description
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event IIC0 TXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection.

Configuration for the DTC HAL Module on r_dtc IIC0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.

ISDE Property	Value	Description
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event IIC0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection.

Configuration for the External IRQ Framework on sf_external_irq

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Controls whether to include code for API parameter checking.
Name	g_sf_external_irq0	Framework name.
Event	None, Semaphore Put Default: Semaphore Put	Event selection.

Configuration for the External IRQ HAL Module on r_icu

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter checking setting enables or disables the addition of parameter checking code.
Name	g_external_irq0	Module name.
Channel	0	Specifies the hardware IRQ channel used.
Trigger	Falling, Rising, Both Edges, Low Level Default: Rising	Configures edge or level triggering.
Digital Filtering	Enabled, Disabled Default: Disabled	Digital filter enable/disable.
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCLK/1, PLCK/8, PLCK/32, PCLK/64 Default: PCKL/64	Sets noise filter sampling period.

ISDE Property	Value	Description
Interrupt enabled after initialization	True, False Default: True	Determines if the interrupt is enabled immediately after initialization.
Callback	NULL	A user callback function can be registered in open. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the IRQn triggers. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	An Interrupt priority can be registered in irq_ipl.

Configuration for the Touch Panel Driver on touch_panel_ft5x06

ISDE Property	Value	Description
Name	g_sf_touch_panel_i2c_chip_ft5x06	Module name.

Configuration for the Touch Panel Driver on touch_panel_sx8654

ISDE Property	Value	Description
Name	g_sf_touch_panel_i2c_chip_sx8654	Module name.

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

NOTE: Most of the property settings for modules are fairly intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Clock Configuration for the Touch Panel Framework

An example implementation of the I2C interface uses the SCI peripheral and is described here. Other implementation choices might have different selections, but can be inferred from the following example. The SCI peripheral module uses PCLKB as its clock source. The PCLKB frequency is set by using the SSP configurator clock tab, prior to a build, or by using the CGC Interface at run-time. During configuration, the I2C transfer rate is calculated and set internally by the driver, based on the user selected PCLB rate and the user selected transfer rate. If the PCLKB is configured in such a manner that the user selected rate cannot be achieved, an error will be returned when initializing the driver.

Pin Configuration for the Touch Panel Framework

An example implementation of the I2C interface uses the SCI peripheral and is described here. Other implementation choices might have different selections, but can be inferred from the following example. Synergy Kit specific settings for pins are shown in the following section. The SCI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table lists the method for selecting the pins within the SSP configuration window and the subsequent table lists an example indicating selection of the I2C pins. The following example settings demonstrate a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings

NOTE: Operation Mode selection mode determines what peripheral signals are available and thus what MCU pins are required.

Pin Selection Sequence for I2C Master Driver on SCI

Resource	ISDE Tab	Pin selection Sequence
SCI0	Pins	Select **Peripherals > Connectivity:SCI > SCI0**

NOTE: The above selection sequence assumes SCI0 is the desired hardware target for the driver.

Pin Configuration Settings for I2C Master Driver on SCI

Pin Configuration Property	Value	Description
Operation Mode	Disabled, Asynchronous UART, Synchronous UART, Simple I2C, Simple SPI, SmartCard (Default: Disabled) Simple I2C	Select Simple I2C as the Operation Mode for SPI on SCI
RXD1_SCL1_MISO1	None, P212, P708 (Default: None)	SCL Pin
TXD1_SDA1_MOSI1	None, P213, P709 (Default: None)	SDA Pin

NOTE: The above example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

Touch Panel Framework Module Synergy Kit Specific Settings

In addition to the above selections some kits specific settings may be needed, based on the touch chip used. These settings are summarized in the following examples of how you may need to configure these for your own custom touch controller device.

Configure the following External IRQ Settings.

For the DK-S7G2 (Touch Chip SX8654), select:

- Channel: 7
- Trigger: Falling

For the SK-S7G2 (Touch Chip SX8654), select:

- Channel: 9
- Trigger: Falling

For the PE-HMI1-S7G2 (Touch Chip FT5x06), select:

- Channel: 12
- Trigger: Falling

Common setting for all boards:

- Digital Filter settings: Any
- Interrupt enabled after initialization: True
- Callback: NULL

Configure the following I2C driver settings.

For the DK-S7G2, select r_sci_i2c:

- Channel: 7
- Rate: Standard
- Slave Address: 0x48
- Address Mode: 7-bit

For the SK- S7G2, select r_riic:

- Channel: 2
- Rate: Standard
- Slave Address: 0x48
- Address Mode: 7-bit

For the PE-HMI1-S7G2, select r_riic:

- Channel: 1

- Rate: Fast-Mode
- Slave Address: 0x38
- Address Mode: 7-bit

Common setting for all boards:

- Callback: NULL

Configure the following Touch Panel Framework module settings.

For the DK-S7G2:

- Touch Chip: g_sf_touch_panel_i2c_chip_sx8654
- H size Pixels: 480
- V size Pixels: 272
- Reset Pin: IOPORT_PORT_07_PIN_11

For the SK-S7G2:

- Touch Chip: g_sf_touch_panel_i2c_chip_sx8654
- H size Pixels: 240
- V size Pixels: 320
- Reset Pin: IOPORT_PORT_06_PIN_09

For the PE-HMI1-S7G2:

- Touch Chip: g_sf_touch_panel_i2c_chip_ft5x06
- H size Pixels: 800
- V size Pixels: 480
- Reset Pin: IOPORT_PORT_10_PIN_02

Common setting for boards:

- Thread Priority: Any

The Touch Chip Drivers for SX8654 and FT5x06 are built-in drivers in the SSP and can be selected when configuring the module.

4.1.28.6 Using the Touch Panel Framework Module in an Application

Once the module has been configured and the files generated, the touch panel framework is ready to be used in an application. Touch panel framework open automatically from the synergy generated file ones the application starts.

Suggested use of the Touch Panel Framework module:

1) Start the touch panel framework using start API of touch panel framework.
2) Pend on a touch message using the pend API
3) Parse the touch message with application code
4) Operate on the received data as needed by the application. Go to step 2.

These common steps in a typical operational flow diagram are shown in the following figure.

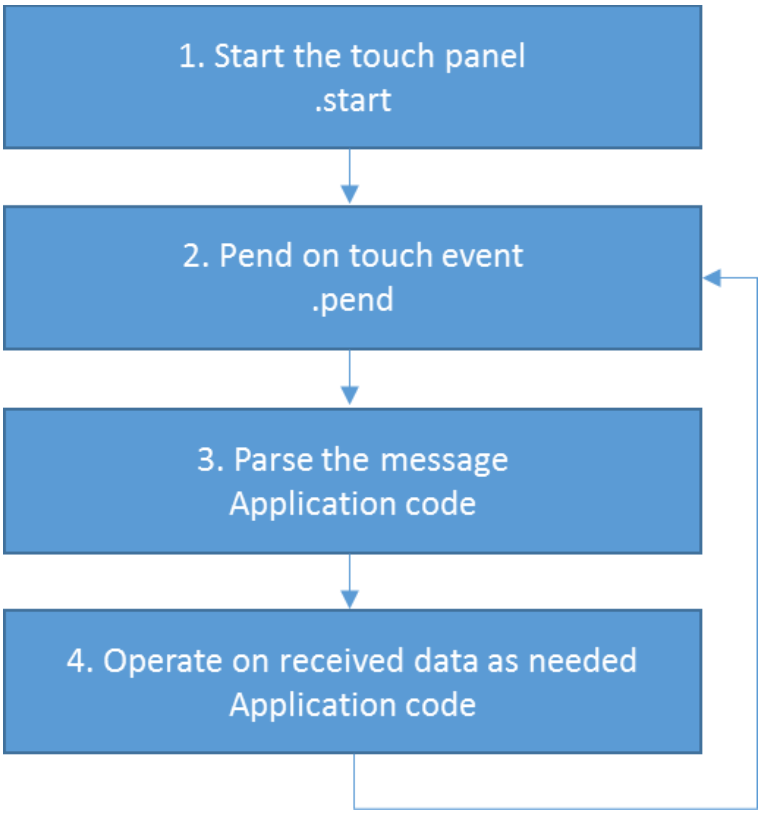


Figure 240: Flow Diagram of a Typical Touch Panel Framework Module Application

4.1.29 Wi-Fi Framework

The Wi-Fi framework provides high-level APIs for configuring and provisioning Wi-Fi modules as well as perform data transfers with or without on-chip networking capability. Currently, only the Qualcomm GT202 module is supported. The Wi-Fi framework communicates through SPI with the underlying GT202 module.

4.1.29.1 Wi-Fi Framework Module Features

The WiFi framework module has the following key features:

- Provides high-level APIs to configure and provision a WiFi module
- Provides four different implementations for:
  - A Wi-Fi device driver stack using the sf_wifi_gt202 framework
  - An on-chip stack using the sf_wifi_onchip_stack framework
  - A BSD socket stack using the sf_wifi_onchip_stack framework
  - A NetX and NetX Duo port using the sf_wifi_nsal_nx framework

- Using NetX and NASL:
  - Allows the same application code to be used across different Wi-Fi modules
  - Allows easy migration of the Ethernet-based application to a Wi-Fi based application
  - Allows for debugging and fine-tuning the application and TCP/IP stack as required by the application.
  - The current NSAL implementation only provides NetX NSAL. Adding support for a new network stack requires implementing the appropriate NSAL.
- Using the On-chip networking stack:
  - Is beneficial when using MCUs with a small memory footprint
  - Provides a BSD sockets interface to create socket-based applications with the On-chip TCP/UDP
  - Provides an option to integrate 3rd-party application protocols on top of TCP/IP such as MQTT and COAP without using the NetX stack.

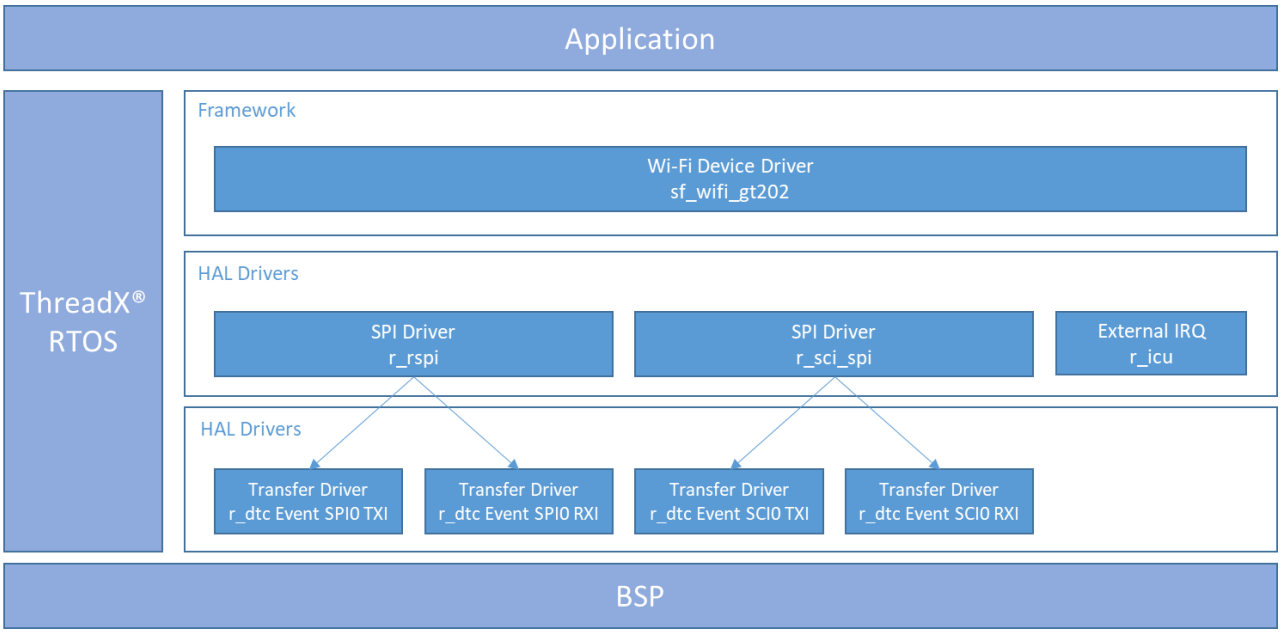


Figure 241:Wi-Fi Device Driver Implementation

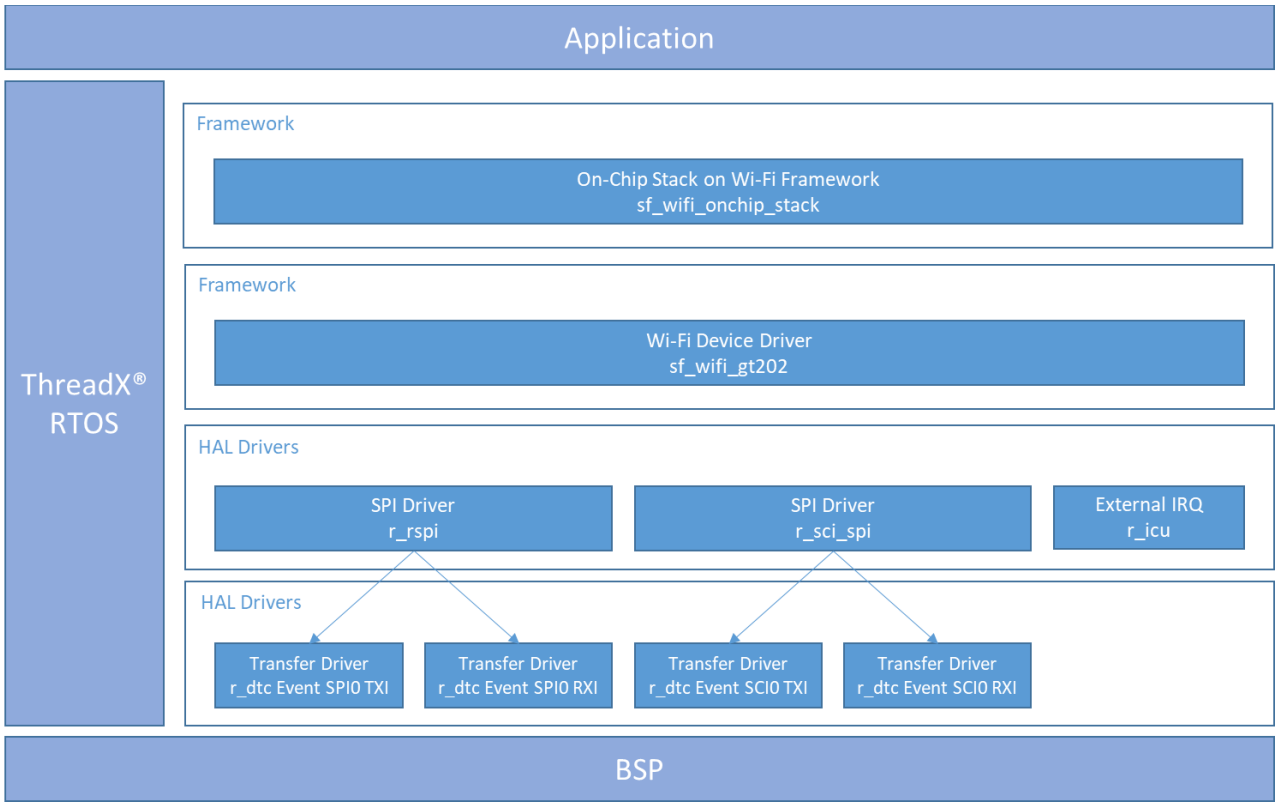


Figure 242: Wi-Fi On-Chip Stack Implementation

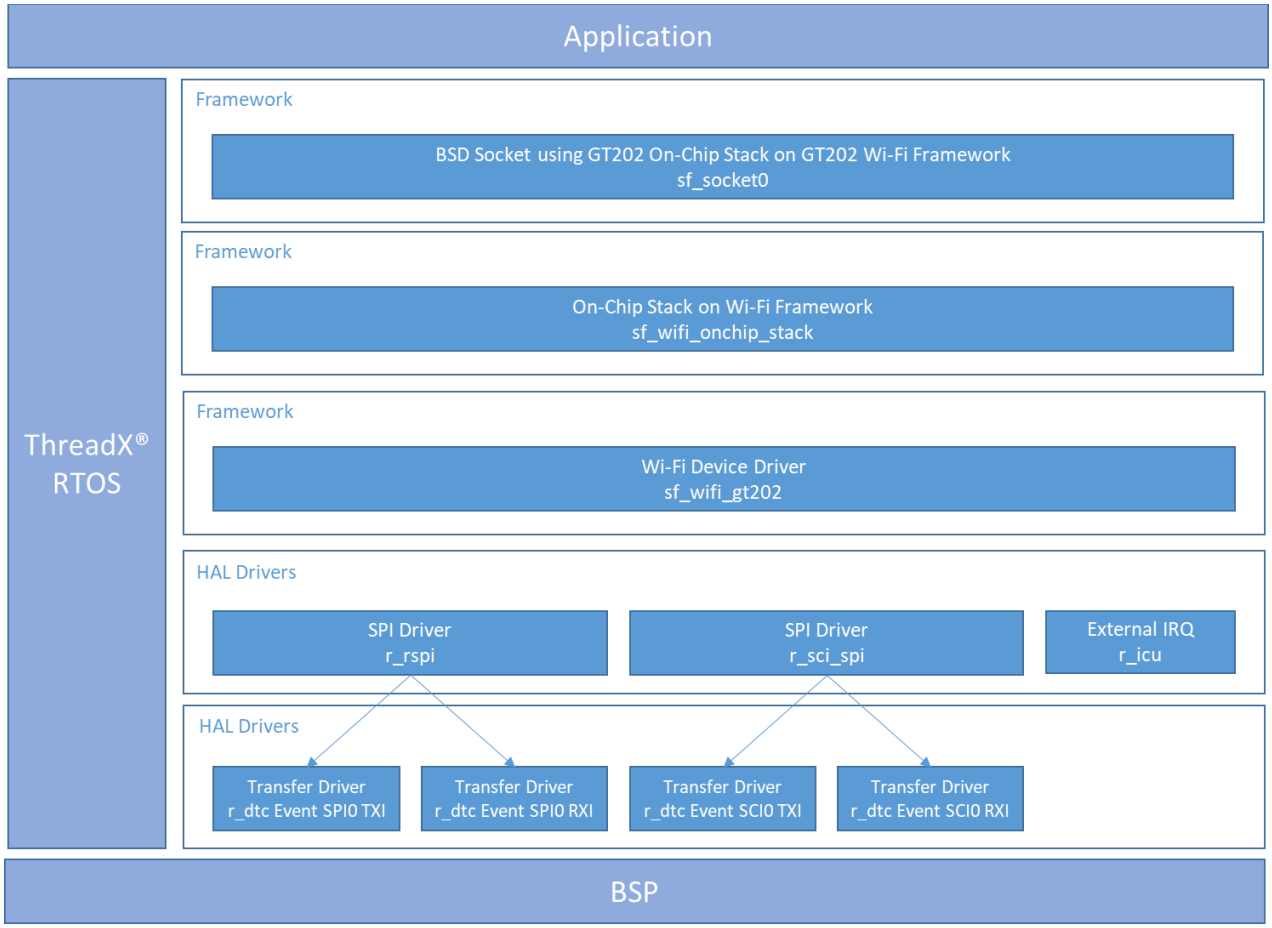


Figure 243: Wi-Fi BSD Socket Implementation

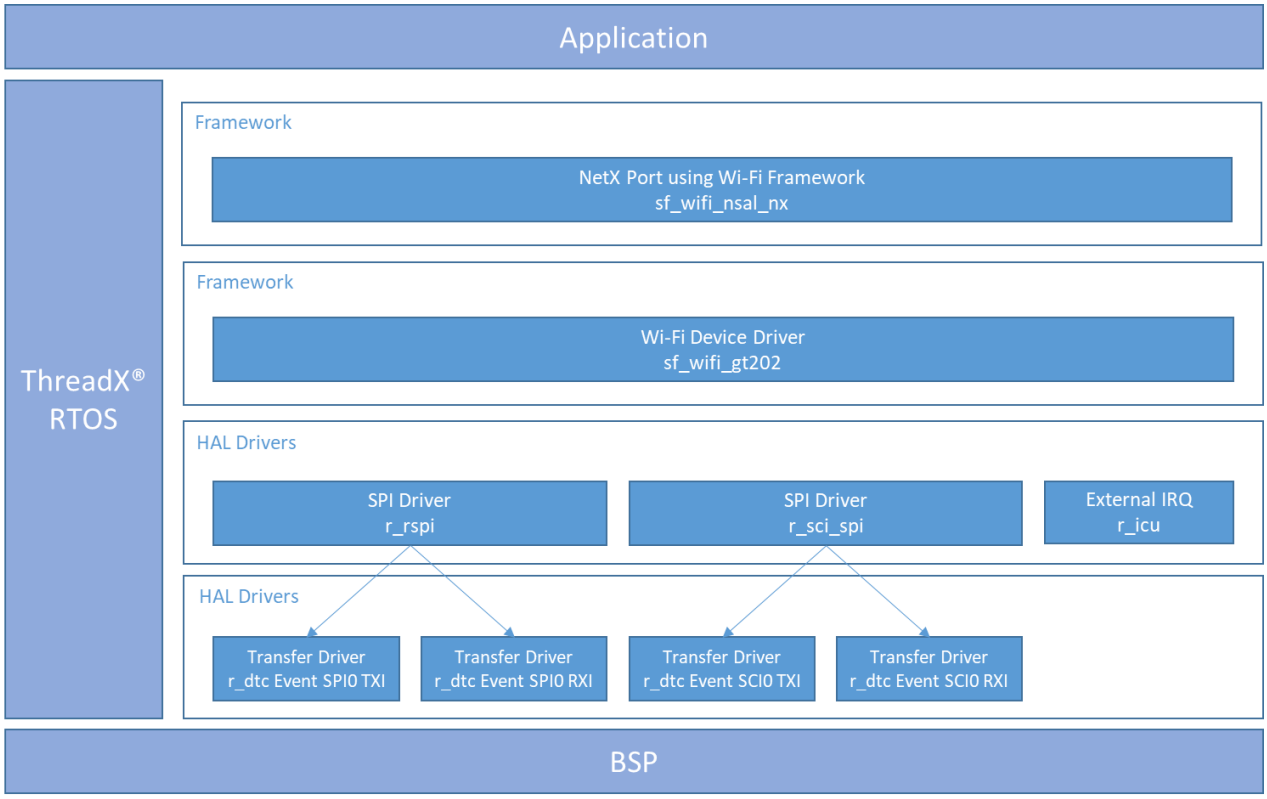


Figure 244:Wi-Fi NetX Port Implementation with NSAL

4.1.29.2 Wi-Fi Framework Module APIs Overview

The Wi-Fi Framework defines APIs for each of the related modules. The following descriptions explain the operation of each API. A table of common return codes follows at the end of the section. Refer to the API reference section for the associated module for additional details.

Additionally, a more detailed API description, along with a working example application (which is too lengthy to include in this document), is available on the Renesas web site. Just search for the associated application note document number, r11an0226eu, in the top page search bar on http://www.Renesas.com. It is highly recommended that you use the application note to augment the summary descriptions found in this document.

Wi-Fi Framework Module APIs

The Wi-Fi framework module provides a high-level interface for the network stack and applications irrespective of the Wi-Fi module. Below are the APIs exposed by the framework.

Wi-Fi Framework Module API Summary

Function Name	Example API Call and Description
.open	g_sf_wifi0.p_api->open (g_sf_wifi0.p_ctrl, g_sf_wifi0.p_cfg); This API initializes and enables the Wi-Fi module. The open function returns the Wi-Fi control structure, uniquely identifying the instance of the Wi-Fi framework.
.close	g_sf_wifi0.p_api->close(g_sf_wifi0.p_ctrl); This API un-initializes the Wi-Fi module and powers it off.
.infoGet	g_sf_wifi0.p_api->infoGet (g_sf_wifi0.p_ctrl, wifi_info); This API takes the WiFi control structure as an argument and returns the following information obtained from the WiFi module: Chipset/driver information string RSSI value (unsigned integer 16 bits) Noise level (unsigned integer 16 bits) Link Quality (unsigned integer 16 bits)
.statisticsGet	g_sf_wifi0.p_api->statisticsGet (g_sf_wifi0.p_ctrl, p_stats); This API gets the data statistics from the Wi-Fi module. It takes the Wi-Fi control structure as an argument and returns the following statistics: Received packets (unsigned integer 32 bits) Transmitted packets (unsigned integer 32 bits) Transmit packet errors (unsigned integer 32 bits)

Function Name	Example API Call and Description
.transmit	g_sf_wifi0.p_api->transmit (g_sf_wifi0.p_ctrl, p_buffer, length); This API sends the data/packet out. This function takes the Wi-Fi control structure as an argument. It takes the network packet buffer, and the network packet buffer length as arguments. The Wi-Fi framework transmit function passes the packet buffer to the Wi-Fi driver for transmission.
.receiveCallback	This is a callback API, which gets called when the data/packet is received by the Wi-Fi module. This function receives packet buffer and packet length as arguments.
.provisioningGet	g_sf_wifi0.p_api->provisioningGet (g_sf_wifi0.p_ctrl, &sf_wifi_provisioninfo); This API takes the Wi-Fi control structure as an argument and returns the following parameters: Mode (enumeration, that is, AP or client) Channel (unsigned integer 8 bits) SSID (string) Security type (enumeration) Encryption type (enumeration) Security key (string)

Function Name	Example API Call and Description
.provisioningSet	g_sf_wifi0.p_api->provisioningSet (g_sf_wifi0.p_ctrl, &g_sf_wifi_provisioninfo); This API sets the Wi-Fi module in the given mode AP/Station. The provisioningSet function uses the following parameters to provision the WiFi module: Mode (enumeration, that is, AP or client) Channel (unsigned integer 8 bits), only used in AP mode SSID (string) Security type (enumeration) Encryption type (enumeration) Security key (sting). Connection Status Notification Callback function: Used to get connection status change notification *See note at the end of this table.

Function Name	Example API Call and Description
.scan	g_sf_wifi0.p_api->scan (g_sf_wifi0.p_ctrl, p_scan, p_count); This API scans the available SSIDs (that is, access points) in range. This function takes the Wi-Fi control structure as an argument and returns a list of SSIDs scanned by the WiFi module with the following parameters: HW mode (enumeration a/b/g/n) RSSI (unsigned integer 16 bits) SSID (string) BSSID (byte array of size 6 bytes) Channel (unsigned integer 8 bits) Security type (enumeration) Encryption type(enumeration) BSS type (enumeration) The Wi-Fi framework scan function takes the SSID count as an argument, which acts as an in/out parameter. It specifies the size of the scan result array and the Wi-Fi framework sets it to count the indicating number of scan results stored in the array.
.ACLAdd	g_sf_wifi0.p_api->ACLAdd (g_sf_wifi0.p_ctrl, peer_device_mac); This API adds the given MAC address to the access control list. This function takes the Wi-Fi control structure and the MAC address as arguments.

Function Name	Example API Call and Description
.ACLDelete	g_sf_wifi0.p_api->ACLDelete (g_sf_wifi0.p_ctrl, peer_device_mac); This API deletes the given MAC address from the access control list. This function takes the Wi-Fi control structure and MAC address as arguments.
.multicastListAdd	g_sf_wifi0.p_api->multicastListAdd (g_sf_wifi0.p_ctrl, p_mac_addr); This API adds the given Multicast IP address to the multicast filer list. This function takes the Wi-Fi control structure and MAC address as arguments.
.multicastListDelete	g_sf_wifi0.p_api->multicastListDelete (g_sf_wifi0.p_ctrl, p_mac_addr); This API deletes the given Multicast IP address from the multicast filer list. This function takes the Wi-Fi control structure and MAC address as arguments.
.macAddressGet	g_sf_wifi0.p_api->macAddressGet (g_sf_wifi0.p_ctrl, p_mac); This API reads MAC address from WiFi module. This function takes the Wi-Fi control structure as argument and returns MAC address read from Wi-Fi module.
.macAddressSet	g_sf_wifi0.p_api->macAddressSet (g_sf_wifi0.p_ctrl, p_mac); This API sets Wi-Fi module's MAC address. This function takes the Wi-Fi control structure and MAC address as arguments.

Note on Provisioning: When device is provisioned in client mode then callback will be called when connection with AP is lost and established back. When device is provisioned in AP mode, this callback is called when any client connects/disconnects with AP. When device is provisioned in client mode then arguments passed to callback will have only below valid field,

- event = SF_WIFI_EVENT_AP_CONNECT or SF_WIFI_AP_DISCONNECT

When device is provisioned in AP mode then arguments passed to callback will have only below valid fields,

- event = SF_WIFI_EVENT_CLIENT_CONNECT or SF_WIFI_CLIENT_DISCONNECT

- mac_addr = MAC address of client.

NOTE: While calling the provisioningSet() API to provision the device in client mode, the framework will not call the callback function on successful association with AP or on failure.

When the device is provisioned in AP mode, if the client tries to connect with the AP using wrong password then callback will be called twice, first with connected event and then immediately after this with disconnected event.

On chip Networking Stack Support APIs

These APIs can be used to configure the Wi-Fi module when using an on-chip networking stack, which helps to configure the IP address for the interface, and start/stop DHCP server (when configured in the AP mode).

On Chip Networking Stack Support Wi-Fi Framework Module API Summary

Function Name	Example API Call and Description
.open	g_sf_wifi_onchip_stack0.p_api->open (g_sf_wifionchip_stack0.p_ctrl, g_sf_wifi_onchip_stack0.p_cfg); This API calls the WiFi framework open API which initializes the Wi-Fi module.
.close	g_sf_wifi_onchip_stack0.p_api->close(g_sf_wifi_onchip_ stack0.p_ctrl); This API calls the Wi-Fi framework close API which un-initializes the Wi-Fi module.
.ipAddressCfg	g_sf_wifi_onchip_stack0.p_api->ipAddressCfg (g_sf_wifi_onchip_stack0.p_ctrl, p_cfg); This API configures the IP address of the interface using an on-chip networking stack. It provides facility to configure static IP address or using DHCP.
.dhcpServerStart	g_sf_wifi_onchip_stack0.p_api->dhcpServer (g_sf_wifi_onchip_stack0.p_ctrl, p_start_ip, p_end_ip); This API starts the DHCP server on the interface (when configured in AP mode) using on-chip networking stack. It takes the range of IP addresses to be used by DHCP server.

Function Name	Example API Call and Description
.dhcpServerStop	g_sf_wifi_onchip_stack0.p_api->dhcpServerStop (g_sf_wifi_onchip_stack0.p_ctrl); This API stops the DHCP server.
.versionGet	g_sf_wifi_onchip_stack0.p_api->versionGet (&version); Retrieves the API version with the version pointer.

BSD Socket APIs

These APIs can be used for BSD socket support using the GT202 on-chip stack implementation.

BSD Socket using GT202 On-Chip Stack Wi-Fi Framework Module API Summary

Function Name	Example API Call and Description
.open	g_sf_socket0.p_api->open (g_sf_socket0.p_ctrl, g_sf_socket0.p_cfg); This API initializes the networking interface.
.close	g_sf_socket0.p_api->close(g_sf_socket0.p_ctrl); This API closes the network interface.
.versionGet	g_sf_socket0.p_api->versionGet (&version); Retrieves the API version with the version pointer.

Additionally, this implementation includes socket APIs which are compliant with BSD APIs. These APIs can be used by the application to perform data transfer using sockets. The following APIs are available:

- socket
- close
- bind
- listen
- accept
- connect
- send

- recv
- recvfrom
- sendto
- setsockopt
- getsockopt
- select

NOTE: While using on chip networking stack, application can use all BSD Socket APIs, all on chip Networking Stack support APIs and few Synergy Wi-Fi framework APIs. The Synergy Wi-Fi framework APIs which application can use are provisioningSet(), provisioningGet(), scan(), macAddressGet(), macAddressSet() and infoGet()

More information is available for these APIs as described in the NetX BSD 4.3 Sockets API Compliancy Wrapper for NetX User Guide which can be found on the Synergy Gallery on the SSP page under the documentation tab in the X-Ware ™ Component Documents for Renesas Synergy zip file.

Wi-Fi NSAL

The Synergy Wi-Fi framework supports the NetX/NetX-Duo Network Services Abstraction Layer. This includes the NetX/NetX-Duo driver, packet transmit and receive callback functions implementation.

NetX/NetX-Duo Driver Function

The NetX/NetX-Duo driver function takes the NetX IP instance, WiFi framework instance and NSAL configuration as arguments. The NSAL configuration controls the behavior of transmit and receive callback functions. The NSAL configuration includes flags which indicates zero-copy support is enabled or disabled in transmit and receive path. The NetX/NetX-Duo driver functions implement various IP driver commands used by NetX/NetX-Duo. The interface attach command calls the WiFi framework open API to initialize the Wi-Fi module. The initialize command calls the Wi-Fi framework macAddressGet API to read MAC address from the WiFi module. The un-initialize command calls the WiFi framework close API which de-initializes the WiFi module. The multicast-join command calls the Wi-Fi framework multicastListAdd API to add the given MAC address to multicast list. The multicast-leave command calls the Wi-Fi framework multicastListDelete API to delete the given MAC address from the multicast list. The Send/Broadcast command calls the WiFi framework transmit API to transmit a packet.

NSAL Transmit Function

The NSAL transmit function takes the NetX IP instance, the NetX packet, the Wi-Fi framework instance and the NSAL configuration as arguments. If zero-copy support is enabled then the same NetX packet is transferred from NetX to the WiFi driver. If zero-copy is not supported then it copies data from the NetX packet to the driver buffer. It calls the Wi-Fi framework transmit API, which passes the buffer/packet to the Wi-Fi driver for further transmission.

NSAL Receive Callback

The NSAL receive callback function takes the NetX IP instance, the packet buffer, the packet buffer length and the NSAL configuration as arguments. This callback is called from the Wi-Fi device driver. If zero-copy support is enabled then the same NetX packet is transferred from the WiFi driver to NetX. If zero-copy is not supported then it copies data from the driver buffer to the NetX packet and then passes the NetX stack for further processing. It calls the Wi-Fi framework transmit API, which passes the buffer to the Wi-Fi driver for further transmission.

More information is available for these APIs as described in the NetX User Guide which can be found on the Synergy Gallery on the SSP page under the documentation tab in the X-Ware ™ Component Documents for Renesas Synergy zip file.

Wi-Fi Framework Error Codes

The table below lists the Wi-Fi Framework specific error codes. These error codes are part of ssp_err_t.

WiFi framework Error Codes

Error Codes	Description
SSP_ERR_WIFI_CONFIG_FAILED	Configuration failed
SSP_ERR_WIFI_INIT_FAILED	Initialization failed
SSP_ERR_WIFI_TRANSMIT_FAILED	Transmission failed
SSP_ERR_WIFI_INVALID_MODE	Invalid mode specified
SSP_ERR_WIFI_FAILED	Wifi failed

4.1.29.3 Wi-Fi Framework Module Operational Overview

The following operational overview is a summary of the operation of the Wi-Fi module. A more detailed description, along with a working example application (which is too lengthy to include in this document), is available on the Renesas web site. Just search for the associated application note document number, r11an0226eu, in the top page search bar on http://www.Renesas.com. It is highly recommended that you use the application note to augment the summary descriptions found in this document.

Wi-Fi Framework Module Operational Introduction

The Wi-Fi framework provides a high-level interface for the application to configure the Wi-Fi module, provision the Wi-Fi module and perform data transfers. This simplifies application development and allows the same application code to be used across different Wi-Fi modules.

The following figure provides an overview of the Synergy Wi-Fi framework layered architecture:

- The Wi-Fi framework includes the enclosed 5 blocks in the middle of the architecture graph: NSAL, Wi-Fi Framework API, Wi-Fi on-chip Stack API, BSD Socket API, and the Wi-Fi Device Driver Interface.
- The vendor-provided Wi-Fi device drivers are included in the SSP package under SSP_Supplemental.

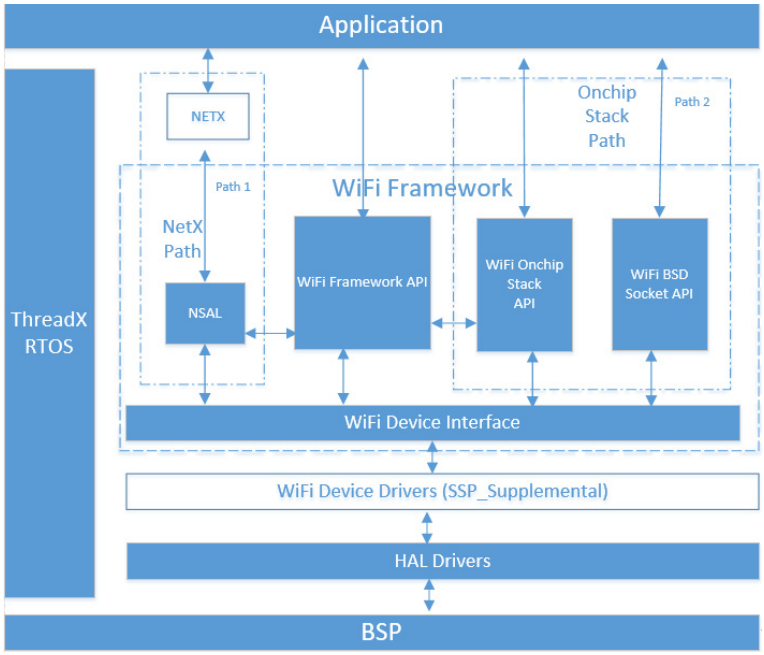


Figure 245: Wi-Fi framework organization, options and stack implementation

The Wi-Fi framework implementation allows Wi-Fi modules with or without On-chip networking stack support to be integrated with the SSP low-level support blocks.

- Path 1: Using NetX™/NetX Duo™, NSAL in addition to the Wi-Fi framework APIs blocks as shown in in the previous figure. To simplify the description, throughout this document, NetX refers to both NetX and NetX-Duo when the comment applies to both.
- Path 2: Using On-chip networking stack support API and the BSD Socket APIs as shown in the previous figure.

Wi-Fi Framework Module Operational Notes and Limitations

- The Wi-Fi module has various parameters as specified by 802.11 standards. It is possible that individual device drivers and Wi-Fi chipsets might not support the configuration of all the functions.
- For the Wi-Fi interface to become active, at least the channel, Service Set Identifier (SSID), security scheme, and security credentials must be configured.
- The current NSAL implementation includes support for NetX (IPv4) and NetX-Duo (IPv6). NetX and NetX Duo support IPv4, however NetX Duo also supports IPv6. Adding support for a new network stack requires implementing the appropriate NSAL.
- For the security setting, WEP keys can be entered in either ASCII or Hex format and can be configured to use either 40 or 104 bit keys. A WEP key has a 24-bit initialization vector in addition to the secret key. Because of this and depending on the vendor, 64-bit WEP keys can be referred to as 40 bit keys and 128-bit WEP keys can be referred to as 104 bit keys. The Wi-Fi framework accepts 1 to 4 WEP keys of a specific format and type. In the provisioning structure, you must fill in the security type as SF_WIFI_SECURITY_TYPE_WEP and at least one (maximum is four) WEP key in the key buffer.

Wi-Fi Framework Module APIs

- OpenWhen using the Wi-Fi framework module with NSAL i.e. with NetX/NeXDuo, the application should not call the Wi-Fi framework module open() API directly, instead it should call the NetX nx_ip_create() API which internally calls open() from the NetX driver.When using the On chip networking stack, the application should call the open() API from the BSD socket interface, which internally calls the Wi-Fi framework open() API.
- CloseWhen using the Wi-Fi framework module with NSAL i.e. with NetX, the application should not call the Wi-Fi framework module close API directly, instead it should call the NetX nx_ip_delete() API which internally calls close from the NetX driver.When using the On chip networking stack, the application should call the close API from the BSD socket interface, which internally calls the Wi-Fi framework module close API.
- ProvisioningSetWhen the device is provisioned in client mode the callback will be called when the connection with the AP is lost and re-established. When the device is provisioned in AP mode, this callback is called when any client connects/disconnects with the AP. When the device is provisioned in client mode the arguments passed to the callback will only have the below valid field:

  – event = SF_WIFI_EVENT_AP_CONNECT or SF_WIFI_AP_DISCONNECT

  When the device is provisioned in AP mode the arguments passed to the callback will only have the below valid fields:

  – event = SF_WIFI_EVENT_CLIENT_CONNECT or SF_WIFI_CLIENT_DISCONNECT

  – mac_addr = MAC address of client.

  While calling the provisioningSet() API to provision the device in client mode, the framework will not call the call-back function on successful association with the AP or on a failure.When the device is provisioned in AP mode, if the client tries to connect with the AP using the wrong password the callback will be called twice, first with the connected event and then immediately after this with the disconnected event.

BSD Socket APIs

While using on chip networking stack, application can use all BSD Socket APIs, all on chip Networking Stack support APIs and few Synergy Wi-Fi framework APIs. The Wi-Fi framework APIs which application can use are provisioningSet(), provisioningGet(), scan(), macAddressGet(), macAddressSet() and infoGet().

- The Wi-Fi framework does not support the Synergy S1 MCU Series due to memory constraints when working with GT202.
- Due to memory constraints S3A6 and S128 MCUs will support only on-chip networking stack (i.e. Path 2 for Wi-Fi use, where networking stack runs on Wi-Fi chipset)
- When RSPI is used for the Wi-Fi module driver, the DTC components that are auto-filled for the dependencies, must be removed because of the limitation in the WiFi module SPI driver. The limitation is that when DTC is used with RSPI, 32-bit transfers are required but GT202 vendor code (that is, WiFi module SPI driver) uses 8-bit transfers only.
- Synergy Wi-Fi Framework APIs implemented for GT-202 are not re-entrant. All these APIs make call to driver APIs to do the requested operation. If GT-202 driver is working on behalf any Wi-Fi framework API and other Wi-Fi framework API is called, then it will return SPP_ERR_IN_USE error until the ongoing operation is finished.
- Synergy Wi-Fi framework provisioningSet() API for GT-202, may fail if peripheral and IO pins' drive capacity is not set to Medium.
- While configuring GT-202 with SPI driver on r_rspi, set drive capacity of slave select pin, reset pin and SPI pins (i.e. MISO, MOSI and RSPCK) to medium. While configuring GT-202 with SPI driver on r_sci_spi, set drive capacity of slave select pin, reset pin and SCI pins (i.e. TXD_MOSI and RXD_MISO) to medium. Do not change the drive capacity of SCK pin when r_sci_spi driver is used.

4.1.29.4 Including the Wi-Fi Framework Module in an Application

This section describes how to include the Wi-Fi framework module in an application using the SSP configurator.

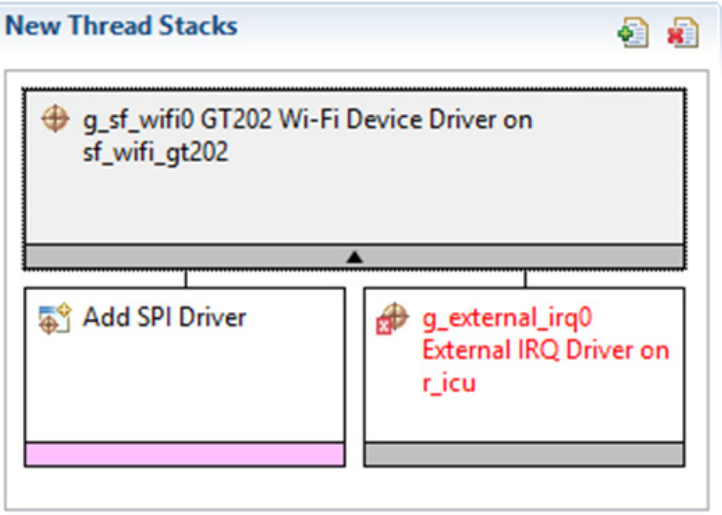
NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to perform these important steps when creating SSP-based applications.

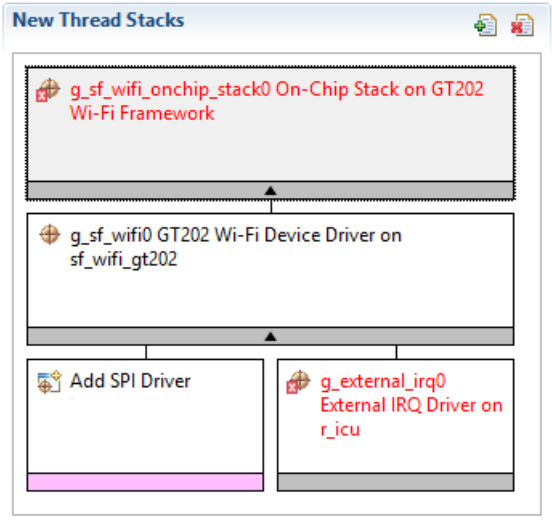
To add the Wi-Fi framework module to an application, simply add it to a thread using the stacks selection sequence given in the following table. There are four different options so four sequences are shown.

Wi-Fi Framework Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_wifi0 GT202 Wi-Fi Device Driver on sf_wifi_gt202	Threads	New Stack> Framework> Networking> WiFi> GT202 Wi-Fi Device Driver on sf_wifi_gt202
g_sf_wifi_onchip_stack0 On-Chip Stack on Gt202 Wi-Fi Framework	Threads	New Stack> Framework> Networking> WiFi> On-Chip Stack on Gt202 Wi-Fi Framework
g_sf_socket0 BSD Socket using On-Chip Stack on Gt202 Wi-Fi Framework	Threads	New Stack> Framework> Networking> WiFi> BSD Socket using On-Chip Stack on Gt202 Wi-Fi Framework
g_sf_el_nx0 NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx	Threads	New Stack> Framework> Networking> WiFi> NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

When the Wi-Fi framework module is added to the thread stack as shown in the following figure, the configurator automatically adds the needed lower-level drivers. Any drivers that need additional configuration information will be highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Pink band require the selection of an implementation option.


New Thread Stacks
g_sf_wifi0 GT202 Wi-Fi Device Driver on sf_wifi_gt202
Add SPI Driver
g_external_irq0 External IRQ Driver on r_icu


New Thread Stacks
g_sf_wifi_onchip_stack0 On-Chip Stack on GT202 Wi-Fi Framework
g_sf_wifi0 GT202 Wi-Fi Device Driver on sf_wifi_gt202
Add SPI Driver
g_external_irq0 External IRQ Driver on r_icu

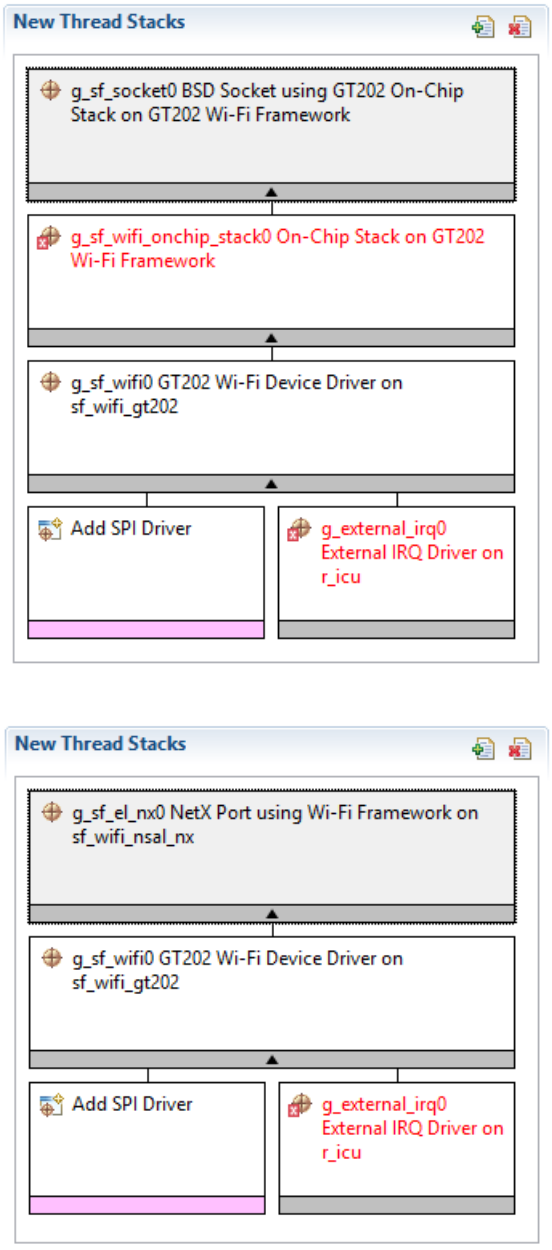


Figure 246: The Four Wi-Fi Framework Module Stack Options

Selecting SF_WIFI_NSAL_NX

The sf_wifi_nsal_nx implementation can be used in conjunction with a NetX IP Instance. In this case, it is added as an option underneath the IP Instance stack. Just add the NetX IP instance using the selection steps in the following table.

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_wifi0 GT202 Wi-Fi Device Driver on sf_wifi_gt202	Threads	New Stack> X-Ware> NetX> NetX IP Instance

Use the available option to add sf_wifi_nsal_nx under the NetX IP Instance as a NetX Network Driver.

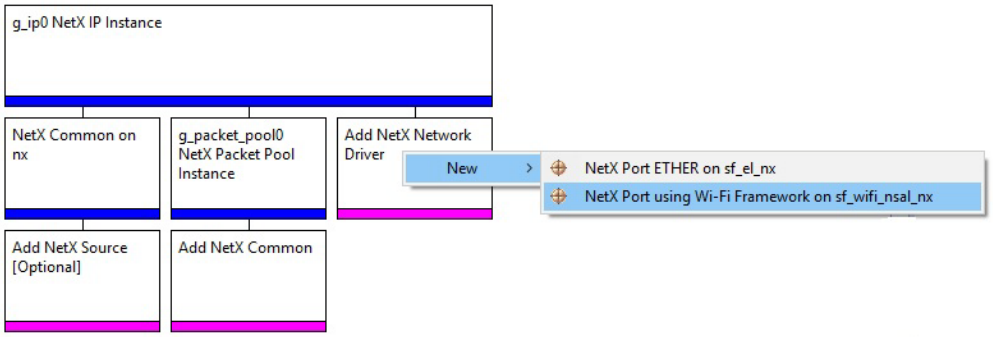


Figure 247: NetX IP Instance use of sf_wifi_nsal_nx

The lower level modules are filled in automatically as shown in the following figure.

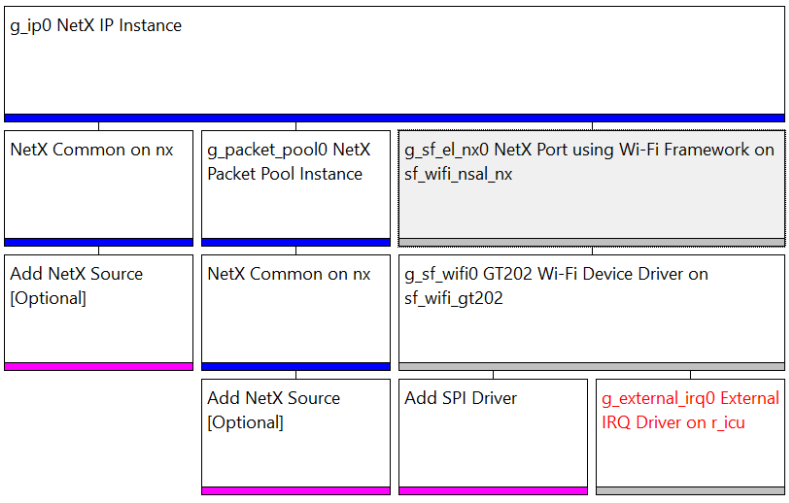


Figure 248: NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

4.1.29.5 Configuring the Wi-Fi Framework Modules

The Wi-Fi framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify, by highlighting the block in red, any required configuration selections, such as Interrupts or operating modes, which must be configured for lower level modules, for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP Configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the Interrupt Priority. This configuration setting is available with the Properties window of the associated module. Simply select the indicated module and then view the properties window. The Interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the Interrupt Priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the below configuration properties tables, but is easily visible with the ISDE when configuring Interrupt Priority levels.

NOTE: You may want to open your ISDE and create the Low Power Mode Driver and explore the property settings in parallel with looking over the Configuration Table Settings given below. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration properties for each of the four WiFi implementations supported by the WiFi framework is provided in the following four sections.

Configuring the GT202 Wi-Fi Device Driver

This section shows the configuration settings for the on-chip stack on the GT202 implementation of the Wi-Fi framework.

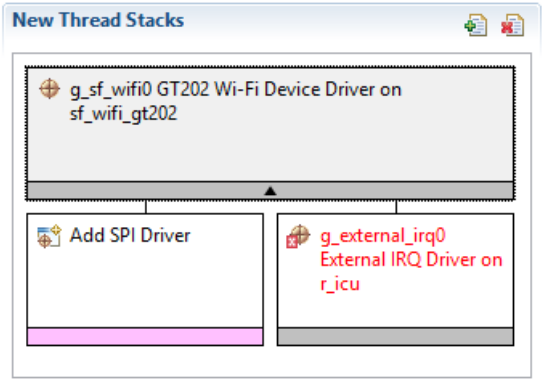


Figure 249: GT202 Wi-Fi Device Driver Thread Stack

Configuration Settings for the Wi-Fi Device Driver on sf_wifi_gt202

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
On-Chip Stack Support	Enabled, Disabled Default: Disabled	On-chip stack support selection
Driver Heap Size in Bytes (Minimum 8192 bytes)	8192	Driver heap size selection
Name (Must be a valid C symbol)	g_sf_wifi0	Module name
Hardware Mode	802.11a, 802.11b, 802.11g, 802.11n Default: 802.11n	Hardware mode selection
Transmit (TX) Power (Valid Range 1-17)	10	Transmit power selection
Ready/Clear to Send (RTS/CTS) Flag	Enabled, Disabled Default: Enabled	Ready/Clear to send selection
Delivery Traffic Indication Message (DTIM) Interval (Valid Range: 1-255)	3	Delivery traffic indication message interval selection
Broadcast SSID (AP mode only)	Enabled, Disabled Default: Enabled	Broadcast SSID selection
Beacon Interval in Microseconds (AP mode only and must be greater than 1023)	1024	Beacon interval in microseconds selection
Station inactivity timeout in seconds (AP mode only and must be greater than 0)	100	Station inactivity timeout selection

ISDE Property	Setting	Description
Requested High Throughput	Enabled, Disabled Default: Disabled	Requested high throughput selection
Reset Pin (must be a valid C symbol)	IOPORT_PORT_06_PIN_00	Reset pin selection
Slave Select Pin (SSL)(Must be a valid C symbol)	IOPORT_PORT_01_PIN_03	Slave select pin selection
GT202 Driver Task Thread Priority (Modifying Task Thread Priority may cause Driver to malfunction)	5	GT202 driver task thread priority selection
Callback	NULL	Callback selection
Support NetX Packet Chaining	Enabled, Disabled Default: Enabled	Support NetX packet chaining selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SPI HAL Driver on r_rspi

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_spi0	Module name
Channel	0	SCI or SPI Channel number to which the device has been connected.

ISDE Property	Setting	Description
Operating Mode	Master	Configure as a Master or Slave device . NOTE: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on even edge, data variation on odd edge	Data sampling on odd or even clock edge.
Clock Polarity	High when idle	Clock level when idle.
Mode Fault Error	Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First	Select transmit order MSB/LSB first
Bitrate	500000	Transmission or reception rate. Bits per second.
Callback	NULL	Optional Callback function pointer.
SPI Mode	Clock synchronous operation	Select spi or clock syn mode operation.
SPI Communication Mode	Full Duplex	Select full-duplex or transmit-only communication.
Slave Select Polarity(SSL0)	Active Low	Select SSL0 signal polarity
Slave Select Polarity(SSL1)	Active Low	Select SSL1 signal polarity
Slave Select Polarity(SSL2)	Active Low	Select SSL2 signal polarity
Slave Select Polarity(SSL3)	Active Low	Select SSL3 signal polarity
Select Loopback1	Normal	Select loopback1
Select Loopback2	Normal	Select loopback2
Enable MOSI Idle State	Disable	Select MOSI idle fixed value and selection
MOSI Idle State	MOSI Low	Select mosi idle fixed value and selection

ISDE Property	Setting	Description
Enable Parity	Disable	Enable/disable parity
Parity Mode	Parity Even	Select parity
Select SSL(Slave Select)	SSL0	Select which slave to use; 0-SSL0; 1-SSL1; 2-SSL2; 3-SSL3
Select SSL Level After Transfer	SSL Level Do Not Keep	Select SSL level after transfer completion; 0-negate; 1-keep
Clock Delay Enable	Clock Delay Disable	Clock delay enable selection
Clock Delay Count	Clock Delay 1 RSPCK	Clock delay count selection
SSL Negation Delay Enable	Negation Delay Disable	SSL negation delay enable selection
Negation Delay Count	Negation Delay 1 RSPCK	Negation delay count selection
Next Access Delay Enable	Next Access Delay Disable	Next access delay enable selection
Next Access Delay Count	Next Access Delay 1 RSPCK	Next access delay count selection
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Receive interrupt priority selection
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection

ISDE Property	Setting	Description
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Error interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SPI0 TXI

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	4 Bytes	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	
Destination Pointer	NULL	Destination pointer selection

ISDE Property	Setting	Description
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SPI0 TXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SPI0 RXI

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	4 Bytes	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection

ISDE Property	Setting	Description
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SPI0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SPI HAL Module on r_sci_spi

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_spi0	Module name

ISDE Property	Setting	Description
Channel	0	SCI or SPI Channel number to which the device has been connected.
Operating Mode	Master	Configure as a Master or Slave device . NOTE: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on even edge, data variation on odd edge	Data sampling on odd or even clock edge.
Clock Polarity	High when idle	Clock level when idle.
Mode Fault Error	Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First	Select transmit order MSB/LSB first
Bitrate	100000	Transmission or reception rate. Bits per second.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bitrate Modulation Function enable or disable
Callback	NULL	Optional Call back function pointer.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Bitrate Modulation Function enable or disable. NOTE: This is applicable only for SCI SPI.

ISDE Property	Setting	Description
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit end interrupt priority selection
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Error interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 TXI

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_spi0	Module name
Channel	0	SCI or SPI Channel number to which the device has been connected.

ISDE Property	Setting	Description
Operating Mode	Master	Configure as a Master or Slave device . NOTE: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on even edge, data variation on odd edge	Data sampling on odd or even clock edge.
Clock Polarity	High when idle	Clock level when idle.
Mode Fault Error	Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First	Select transmit order MSB/LSB first
Bitrate	100000	Transmission or reception rate. Bits per second.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bitrate Modulation Function enable or disable
Callback	NULL	Optional Call back function pointer.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Bitrate Modulation Function enable or disable. NOTE: This is applicable only for SCI SPI.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection.

ISDE Property	Setting	Description
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit end interrupt priority selection
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Error interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 RXI

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection

ISDE Property	Setting	Description
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Softare Event interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the External IRQ HAL Module on r_icu

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter checking setting enables or disables the addition of parameter checking code.

ISDE Property	Setting	Description
Name	g_external_irq0	Module name.
Channel	0	Specifies the hardware IRQ channel used.
Trigger	Falling	Selection for trigger event mode
Digital Filtering	Disabled	Digital filter enable/disable.
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCKL/64	Sets noise filter sampling period.
Interrupt enabled after initialization	TRUE	Determines if the interrupt is enabled immediately after initialization.
Callback	custom_hw_irq_isr	A user callback function can be registered in open. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the IRQn triggers. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuring the On-Chip Stack on GT202 Wi-Fi Framework

This section shows the configuration settings for the on-chip stack on the GT202 implementation of the Wi-Fi framework.

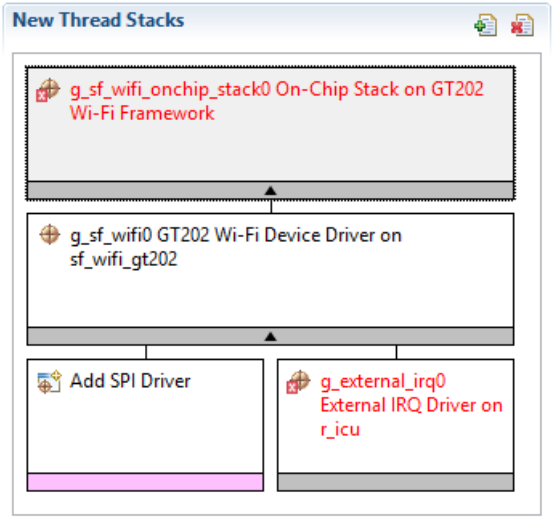


Figure 250: On-Chip Stack on GT202 Wi-Fi Framework Thread Stack

Configuration Settings for the On-Chip Stack on GT202 Wi-Fi Framework

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name (Must be a valid C Symbol)	g_sf_wifi_onchip_stack0	Module name

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Wi-Fi Device Driver

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.

ISDE Property	Setting	Description
On-Chip Stack Support	Enabled, Disabled Default: Disabled	On-chip stack support selection
Driver Heap Size in Bytes (Minimum 8192 bytes)	8192	Driver heap size selection
Name (Must be a valid C symbol)	g_sf_wifi0	Module name
Hardware Mode	802.11a, 802.11b, 802.11g, 802.11n Default: 802.11n	Hardware mode selection
Transmit (TX) Power (Valid Range 1-17)	10	Transmit power selection
Ready/Clear to Send (RTS/CTS) Flag	Enabled, Disabled Default: Enabled	Ready/Clear to send selection
Delivery Traffic Indication Message (DTIM) Interval (Valid Range: 1-255)	3	Delivery traffic indication message interval selection
Broadcast SSID (AP mode only)	Enabled, Disabled Default: Enabled	Broadcast SSID selection
Beacon Interval in Microseconds (AP mode only and must be greater than 1023)	1024	Beacon interval in microseconds selection
Station inactivity timeout in seconds (AP mode only and must be greater than 0)	100	Station inactivity timeout selection
Requested High Throughput	Enabled, Disabled Default: Disabled	Requested high throughput selection
Reset Pin (must be a valid C symbol)	IOPORT_PORT_06_PIN_00	Reset pin selection
Slave Select Pin (SSL)(Must be a valid C symbol)	IOPORT_PORT_01_PIN_03	Slave select pin selection

ISDE Property	Setting	Description
GT202 Driver Task Thread Priority (Modifying Task Thread Priority may cause Driver to malfunction)	5	GT202 driver task thread priority selection
Callback	NULL	Callback selection
Support NetX Packet Chaining	Enabled, Disabled Default: Enabled	Support NetX packet chaining selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SPI HAL Driver on r_rspi

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_spi0	Module name
Channel	0	SCI or SPI Channel number to which the device has been connected.
Operating Mode	Master	Configure as a Master or Slave device . NOTE: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on even edge, data variation on odd edge	Data sampling on odd or even clock edge.
Clock Polarity	High when idle	Clock level when idle.
Mode Fault Error	Disable	Indicates Mode fault error (master/slave conflict) flag.

ISDE Property	Setting	Description
Bit Order	MSB First	Select transmit order MSB/LSB first
Bitrate	500000	Transmission or reception rate. Bits per second.
Callback	NULL	Optional Callback function pointer.
SPI Mode	Clock synchronous operation	Select spi or clock syn mode operation.
SPI Communication Mode	Full Duplex	Select full-duplex or transmit-only communication.
Slave Select Polarity(SSL0)	Active Low	Select SSL0 signal polarity
Slave Select Polarity(SSL1)	Active Low	Select SSL1 signal polarity
Slave Select Polarity(SSL2)	Active Low	Select SSL2 signal polarity
Slave Select Polarity(SSL3)	Active Low	Select SSL3 signal polarity
Select Loopback1	Normal	Select loopback1
Select Loopback2	Normal	Select loopback2
Enable MOSI Idle State	Disable	Select MOSI idle fixed value and selection
MOSI Idle State	MOSI Low	Select mosi idle fixed value and selection
Enable Parity	Disable	Enable/disable parity
Parity Mode	Parity Even	Select parity
Select SSL(Slave Select)	SSL0	Select which slave to use; 0-SSL0; 1-SSL1; 2-SSL2; 3-SSL3
Select SSL Level After Transfer	SSL Level Do Not Keep	Select SSL level after transfer completion; 0-negate; 1-keep
Clock Delay Enable	Clock Delay Disable	Clock delay enable selection
Clock Delay Count	Clock Delay 1 RSPCK	Clock delay count selection
SSL Negation Delay Enable	Negation Delay Disable	SSL negation delay enable selection
Negation Delay Count	Negation Delay 1 RSPCK	Negation delay count selection

ISDE Property	Setting	Description
Next Access Delay Enable	Next Access Delay Disable	Next access delay enable selection
Next Access Delay Count	Next Access Delay 1 RSPCK	Next access delay count selection
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Receive interrupt priority selection
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Error interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SPI0 TXI

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build

ISDE Property	Setting	Description
Software Start	Enabled, Disabled Default: Disabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	4 Bytes	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SPI0 TXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SPI0 RXI

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	4 Bytes	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SPI0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection

ISDE Property	Setting	Description
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SPI HAL Module on r_sci_spi

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_spi0	Module name
Channel	0	SCI or SPI Channel number to which the device has been connected.
Operating Mode	Master	Configure as a Master or Slave device . NOTE: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on even edge, data variation on odd edge	Data sampling on odd or even clock edge.
Clock Polarity	High when idle	Clock level when idle.
Mode Fault Error	Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First	Select transmit order MSB/LSB first

ISDE Property	Setting	Description
Bitrate	100000	Transmission or reception rate. Bits per second.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bitrate Modulation Function enable or disable
Callback	NULL	Optional Call back function pointer.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Bitrate Modulation Function enable or disable. NOTE: This is applicable only for SCI SPI.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit end interrupt priority selection
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Error interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 TXI

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_spi0	Module name
Channel	0	SCI or SPI Channel number to which the device has been connected.
Operating Mode	Master	Configure as a Master or Slave device . NOTE: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on even edge, data variation on odd edge	Data sampling on odd or even clock edge.
Clock Polarity	High when idle	Clock level when idle.
Mode Fault Error	Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First	Select transmit order MSB/LSB first
Bitrate	100000	Transmission or reception rate. Bits per second.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bitrate Modulation Function enable or disable
Callback	NULL	Optional Call back function pointer.

ISDE Property	Setting	Description
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Bitrate Modulation Function enable or disable. NOTE: This is applicable only for SCI SPI.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit end interrupt priority selection
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Error interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 RXI

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection

ISDE Property	Setting	Description
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Softare Event interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the External IRQ HAL Module on r_icu

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter checking setting enables or disables the addition of parameter checking code.
Name	g_external_irq0	Module name.
Channel	0	Specifies the hardware IRQ channel used.
Trigger	Falling	Selection for trigger event mode
Digital Filtering	Disabled	Digital filter enable/disable.
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCKL/64	Sets noise filter sampling period.
Interrupt enabled after initialization	TRUE	Determines if the interrupt is enabled immediately after initialization.

ISDE Property	Setting	Description
Callback	custom_hw_irq_isr	A user callback function can be registered in open. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the IRQn triggers. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuring the BSD Socket using GT202 On-Chip Stack on GT202 Wi-Fi Framework

This section shows the configuration settings for the BSD socket using the GT202 on-chip stack implementation of the Wi-Fi framework.

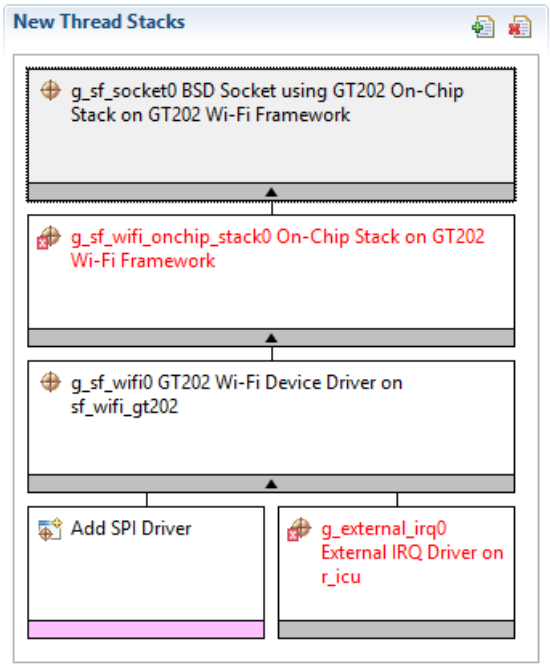


Figure 251: BSD Socket using GT202 On-Chip Stack on GT202 Wi-Fi Framework

Configuration Settings for the BSD Socket using GT202 On-Chip Stack on GT202 Wi-Fi Framework

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name (Must be a valid C Symbol)	g_sf_socket0	Module name

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the On-Chip Stack on GT202 Wi-Fi Framework

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name (Must be a valid C Symbol)	g_sf_wifi_onchip_stack0	Module name

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Wi-Fi Device Driver

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
On-Chip Stack Support	Enabled, Disabled Default: Disabled	On-chip stack support selection
Driver Heap Size in Bytes (Minimum 8192 bytes)	8192	Driver heap size selection
Name (Must be a valid C symbol)	g_sf_wifi0	Module name
Hardware Mode	802.11a, 802.11b, 802.11g, 802.11n Default: 802.11n	Hardware mode selection
Transmit (TX) Power (Valid Range 1-17)	10	Transmit power selection
Ready/Clear to Send (RTS/CTS) Flag	Enabled, Disabled Default: Enabled	Ready/Clear to send selection
Delivery Traffic Indication Message (DTIM) Interval (Valid Range: 1-255)	3	Delivery traffic indication message interval selection

ISDE Property	Setting	Description
Broadcast SSID (AP mode only)	Enabled, Disabled Default: Enabled	Broadcast SSID selection
Beacon Interval in Microseconds (AP mode only and must be greater than 1023)	1024	Beacon interval in microseconds selection
Station inactivity timeout in seconds (AP mode only and must be greater than 0)	100	Station inactivity timeout selection
Requested High Throughput	Enabled, Disabled Default: Disabled	Requested high throughput selection
Reset Pin (must be a valid C symbol)	IOPORT_PORT_06_PIN_00	Reset pin selection
Slave Select Pin (SSL)(Must be a valid C symbol)	IOPORT_PORT_01_PIN_03	Slave select pin selection
GT202 Driver Task Thread Priority (Modifying Task Thread Priority may cause Driver to malfunction)	5	GT202 driver task thread priority selection
Callback	NULL	Callback selection
Support NetX Packet Chaining	Enabled, Disabled Default: Enabled	Support NetX packet chaining selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SPI HAL Driver on r_rspi

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.

ISDE Property	Setting	Description
Name	g_spi0	Module name
Channel	0	SCI or SPI Channel number to which the device has been connected.
Operating Mode	Master	Configure as a Master or Slave device . NOTE: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on even edge, data variation on odd edge	Data sampling on odd or even clock edge.
Clock Polarity	High when idle	Clock level when idle.
Mode Fault Error	Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First	Select transmit order MSB/LSB first
Bitrate	500000	Transmission or reception rate. Bits per second.
Callback	NULL	Optional Callback function pointer.
SPI Mode	Clock synchronous operation	Select spi or clock syn mode operation.
SPI Communication Mode	Full Duplex	Select full-duplex or transmit-only communication.
Slave Select Polarity(SSL0)	Active Low	Select SSL0 signal polarity
Slave Select Polarity(SSL1)	Active Low	Select SSL1 signal polarity
Slave Select Polarity(SSL2)	Active Low	Select SSL2 signal polarity
Slave Select Polarity(SSL3)	Active Low	Select SSL3 signal polarity
Select Loopback1	Normal	Select loopback1
Select Loopback2	Normal	Select loopback2

ISDE Property	Setting	Description
Enable MOSI Idle State	Disable	Select MOSI idle fixed value and selection
MOSI Idle State	MOSI Low	Select mosi idle fixed value and selection
Enable Parity	Disable	Enable/disable parity
Parity Mode	Parity Even	Select parity
Select SSL(Slave Select)	SSL0	Select which slave to use; 0-SSL0; 1-SSL1; 2-SSL2; 3-SSL3
Select SSL Level After Transfer	SSL Level Do Not Keep	Select SSL level after transfer completion; 0-negate; 1-keep
Clock Delay Enable	Clock Delay Disable	Clock delay enable selection
Clock Delay Count	Clock Delay 1 RSPCK	Clock delay count selection
SSL Negation Delay Enable	Negation Delay Disable	SSL negation delay enable selection
Negation Delay Count	Negation Delay 1 RSPCK	Negation delay count selection
Next Access Delay Enable	Next Access Delay Disable	Next access delay enable selection
Next Access Delay Count	Next Access Delay 1 RSPCK	Next access delay count selection
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Receive interrupt priority selection
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection

ISDE Property	Setting	Description
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Error interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SPI0 TXI

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	4 Bytes	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	
Destination Pointer	NULL	Destination pointer selection

ISDE Property	Setting	Description
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SPI0 TXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SPI0 RXI

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	4 Bytes	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection

ISDE Property	Setting	Description
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SPI0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SPI HAL Module on r_sci_spi

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_spi0	Module name

ISDE Property	Setting	Description
Channel	0	SCI or SPI Channel number to which the device has been connected.
Operating Mode	Master	Configure as a Master or Slave device . NOTE: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on even edge, data variation on odd edge	Data sampling on odd or even clock edge.
Clock Polarity	High when idle	Clock level when idle.
Mode Fault Error	Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First	Select transmit order MSB/LSB first
Bitrate	100000	Transmission or reception rate. Bits per second.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bitrate Modulation Function enable or disable
Callback	NULL	Optional Call back function pointer.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Bitrate Modulation Function enable or disable. NOTE: This is applicable only for SCI SPI.

ISDE Property	Setting	Description
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit end interrupt priority selection
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Error interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 TXI

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_spi0	Module name
Channel	0	SCI or SPI Channel number to which the device has been connected.

ISDE Property	Setting	Description
Operating Mode	Master	Configure as a Master or Slave device . NOTE: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on even edge, data variation on odd edge	Data sampling on odd or even clock edge.
Clock Polarity	High when idle	Clock level when idle.
Mode Fault Error	Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First	Select transmit order MSB/LSB first
Bitrate	100000	Transmission or reception rate. Bits per second.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bitrate Modulation Function enable or disable
Callback	NULL	Optional Call back function pointer.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Bitrate Modulation Function enable or disable. NOTE: This is applicable only for SCI SPI.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection.

ISDE Property	Setting	Description
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit end interrupt priority selection
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Error interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 RXI

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection

ISDE Property	Setting	Description
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Softare Event interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the External IRQ HAL Module on r_icu

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter checking setting enables or disables the addition of parameter checking code.

ISDE Property	Setting	Description
Name	g_external_irq0	Module name.
Channel	0	Specifies the hardware IRQ channel used.
Trigger	Falling	Selection for trigger event mode
Digital Filtering	Disabled	Digital filter enable/disable.
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCKL/64	Sets noise filter sampling period.
Interrupt enabled after initialization	TRUE	Determines if the interrupt is enabled immediately after initialization.
Callback	custom_hw_irq_isr	A user callback function can be registered in open. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the IRQn triggers. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuring the NetX Port using Wi-Fi Framework

This section shows the configuration settings for the NetX Port implementation of the Wi-Fi framework.

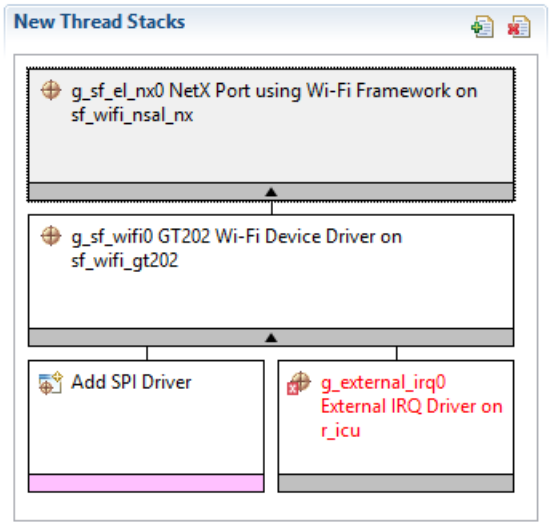


Figure 252: NetX Port using Wi-Fi Framework

Configuration Settings for the NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name (Must be a valid C Symbol)	g_sf_el_nx0	Module name

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Wi-Fi Device Driver

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.

ISDE Property	Setting	Description
On-Chip Stack Support	Enabled, Disabled Default: Disabled	On-chip stack support selection
Driver Heap Size in Bytes (Minimum 8192 bytes)	8192	Driver heap size selection
Name (Must be a valid C symbol)	g_sf_wifi0	Module name
Hardware Mode	802.11a, 802.11b, 802.11g, 802.11n Default: 802.11n	Hardware mode selection
Transmit (TX) Power (Valid Range 1-17)	10	Transmit power selection
Ready/Clear to Send (RTS/CTS) Flag	Enabled, Disabled Default: Enabled	Ready/Clear to send selection
Delivery Traffic Indication Message (DTIM) Interval (Valid Range: 1-255)	3	Delivery traffic indication message interval selection
Broadcast SSID (AP mode only)	Enabled, Disabled Default: Enabled	Broadcast SSID selection
Beacon Interval in Microseconds (AP mode only and must be greater than 1023)	1024	Beacon interval in microseconds selection
Station inactivity timeout in seconds (AP mode only and must be greater than 0)	100	Station inactivity timeout selection
Requested High Throughput	Enabled, Disabled Default: Disabled	Requested high throughput selection
Reset Pin (must be a valid C symbol)	IOPORT_PORT_06_PIN_00	Reset pin selection
Slave Select Pin (SSL)(Must be a valid C symbol)	IOPORT_PORT_01_PIN_03	Slave select pin selection

ISDE Property	Setting	Description
GT202 Driver Task Thread Priority (Modifying Task Thread Priority may cause Driver to malfunction)	5	GT202 driver task thread priority selection
Callback	NULL	Callback selection
Support NetX Packet Chaining	Enabled, Disabled Default: Enabled	Support NetX packet chaining selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SPI HAL Driver on r_rspi

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_spi0	Module name
Channel	0	SCI or SPI Channel number to which the device has been connected.
Operating Mode	Master	Configure as a Master or Slave device . NOTE: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on even edge, data variation on odd edge	Data sampling on odd or even clock edge.
Clock Polarity	High when idle	Clock level when idle.
Mode Fault Error	Disable	Indicates Mode fault error (master/slave conflict) flag.

ISDE Property	Setting	Description
Bit Order	MSB First	Select transmit order MSB/LSB first
Bitrate	500000	Transmission or reception rate. Bits per second.
Callback	NULL	Optional Callback function pointer.
SPI Mode	Clock synchronous operation	Select spi or clock syn mode operation.
SPI Communication Mode	Full Duplex	Select full-duplex or transmit-only communication.
Slave Select Polarity(SSL0)	Active Low	Select SSL0 signal polarity
Slave Select Polarity(SSL1)	Active Low	Select SSL1 signal polarity
Slave Select Polarity(SSL2)	Active Low	Select SSL2 signal polarity
Slave Select Polarity(SSL3)	Active Low	Select SSL3 signal polarity
Select Loopback1	Normal	Select loopback1
Select Loopback2	Normal	Select loopback2
Enable MOSI Idle State	Disable	Select MOSI idle fixed value and selection
MOSI Idle State	MOSI Low	Select mosi idle fixed value and selection
Enable Parity	Disable	Enable/disable parity
Parity Mode	Parity Even	Select parity
Select SSL(Slave Select)	SSL0	Select which slave to use; 0-SSL0; 1-SSL1; 2-SSL2; 3-SSL3
Select SSL Level After Transfer	SSL Level Do Not Keep	Select SSL level after transfer completion; 0-negate; 1-keep
Clock Delay Enable	Clock Delay Disable	Clock delay enable selection
Clock Delay Count	Clock Delay 1 RSPCK	Clock delay count selection
SSL Negation Delay Enable	Negation Delay Disable	SSL negation delay enable selection
Negation Delay Count	Negation Delay 1 RSPCK	Negation delay count selection

ISDE Property	Setting	Description
Next Access Delay Enable	Next Access Delay Disable	Next access delay enable selection
Next Access Delay Count	Next Access Delay 1 RSPCK	Next access delay count selection
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Receive interrupt priority selection
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Error interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SPI0 TXI

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build

ISDE Property	Setting	Description
Software Start	Enabled, Disabled Default: Disabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	4 Bytes	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SPI0 TXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SPI0 RXI

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	4 Bytes	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SPI0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection

ISDE Property	Setting	Description
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SPI HAL Module on r_sci_spi

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_spi0	Module name
Channel	0	SCI or SPI Channel number to which the device has been connected.
Operating Mode	Master	Configure as a Master or Slave device . NOTE: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on even edge, data variation on odd edge	Data sampling on odd or even clock edge.
Clock Polarity	High when idle	Clock level when idle.
Mode Fault Error	Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First	Select transmit order MSB/LSB first

ISDE Property	Setting	Description
Bitrate	100000	Transmission or reception rate. Bits per second.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bitrate Modulation Function enable or disable
Callback	NULL	Optional Call back function pointer.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Bitrate Modulation Function enable or disable. NOTE: This is applicable only for SCI SPI.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit end interrupt priority selection
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Error interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 TXI

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_spi0	Module name
Channel	0	SCI or SPI Channel number to which the device has been connected.
Operating Mode	Master	Configure as a Master or Slave device . NOTE: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on even edge, data variation on odd edge	Data sampling on odd or even clock edge.
Clock Polarity	High when idle	Clock level when idle.
Mode Fault Error	Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First	Select transmit order MSB/LSB first
Bitrate	100000	Transmission or reception rate. Bits per second.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bitrate Modulation Function enable or disable
Callback	NULL	Optional Call back function pointer.

ISDE Property	Setting	Description
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Bitrate Modulation Function enable or disable. NOTE: This is applicable only for SCI SPI.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit end interrupt priority selection
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Error interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Transfer Driver on r_dtc Event SCI0 RXI

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection

ISDE Property	Setting	Description
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Softare Event interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the External IRQ HAL Module on r_icu

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter checking setting enables or disables the addition of parameter checking code.
Name	g_external_irq0	Module name.
Channel	0	Specifies the hardware IRQ channel used.
Trigger	Falling	Selection for trigger event mode
Digital Filtering	Disabled	Digital filter enable/disable.
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCKL/64	Sets noise filter sampling period.
Interrupt enabled after initialization	TRUE	Determines if the interrupt is enabled immediately after initialization.

ISDE Property	Setting	Description
Callback	custom_hw_irq_isr	A user callback function can be registered in open. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the IRQn triggers. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

Wi-Fi Framework Module Clock Configuration

The Wi-Fi Framework module uses the clocks required for the specific selections of the low-level modules such as SPI.

Wi-Fi Framework Module Pin Configuration

The Wi-Fi Framework module uses input and output pins depending on the selections of the low-level modules such as SPI or IRQ.

4.1.29.6 Using the Wi-Fi Framework Module in an Application

The following description is a high-level overview of some typical Wi-Fi use cases. A more detailed description and a working application project (which is too lengthy to include in this document), is available on the Renesas web site. Just search for the associated application note document number, r11an0226eu, in the top page search bar on http://www.Renesas.com. It is highly recommended that you use the application note to augment the summary descriptions found in this document.

Each of the Wi-Fi framework implementations are treated differently in a target application. The typical control flow for initialization, packet transmission using NetX/NetX Duo, packet reception using NetX/NetX Duo, and using an on-chip networking stack are of particular interest. Example flow diagrams for some typical implementations of these functions are shown as follows:

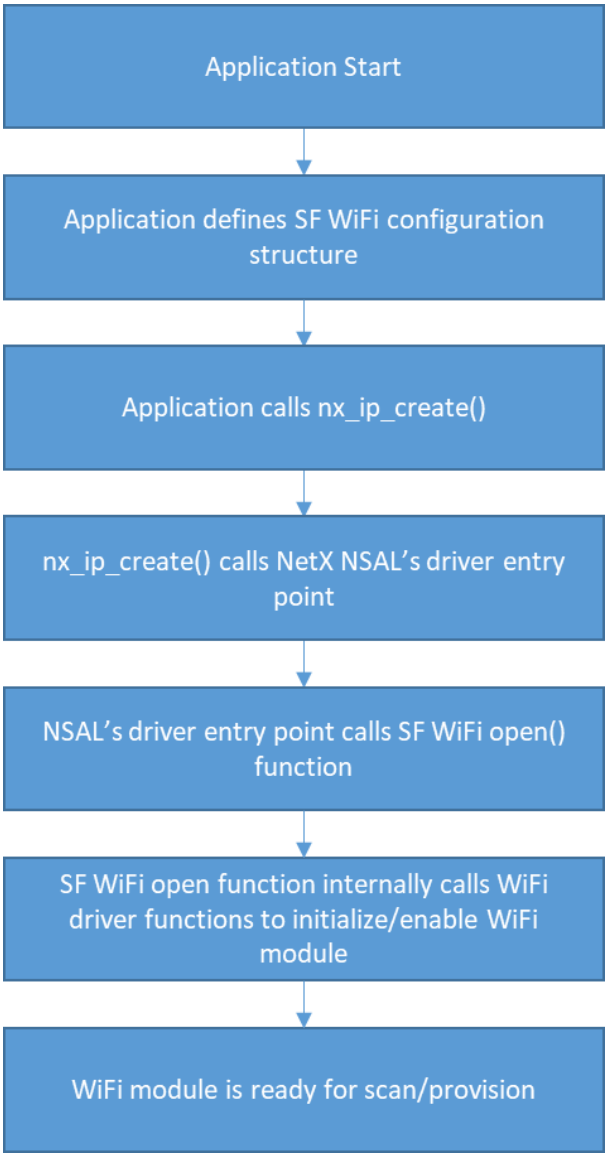


Figure 253: Application Control Flow using Wi-Fi module initialization

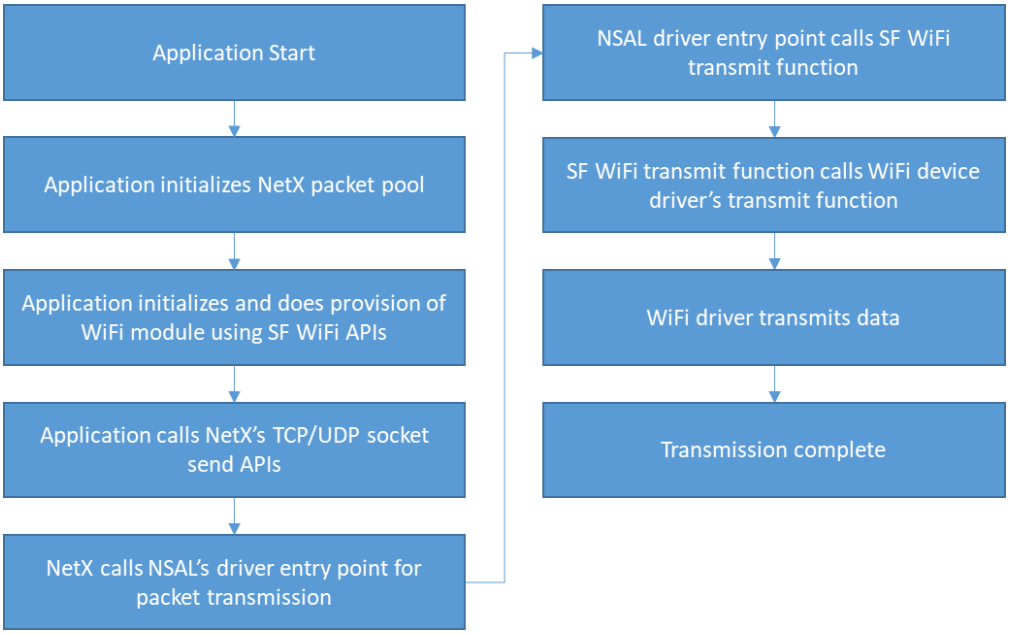


Figure 254: Application Control Flow doing packet transmission using NetX

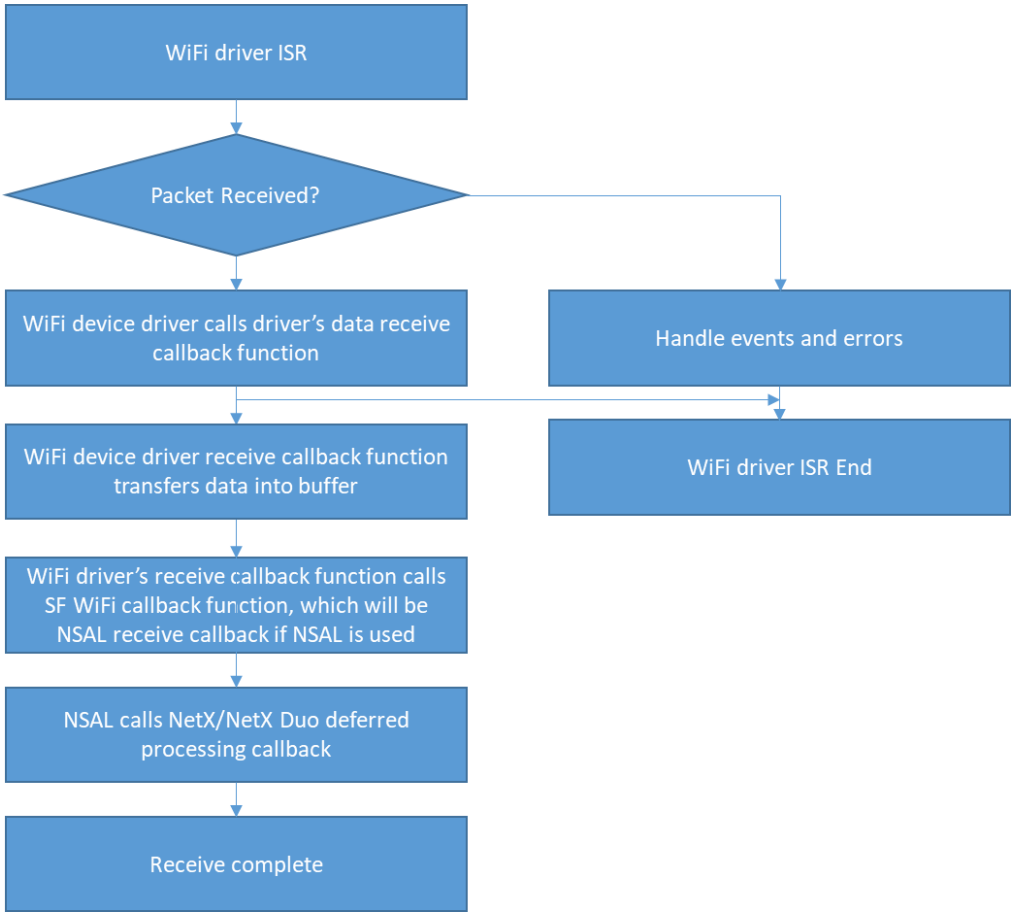


Figure 255: Application control flow receiving packet using NetX

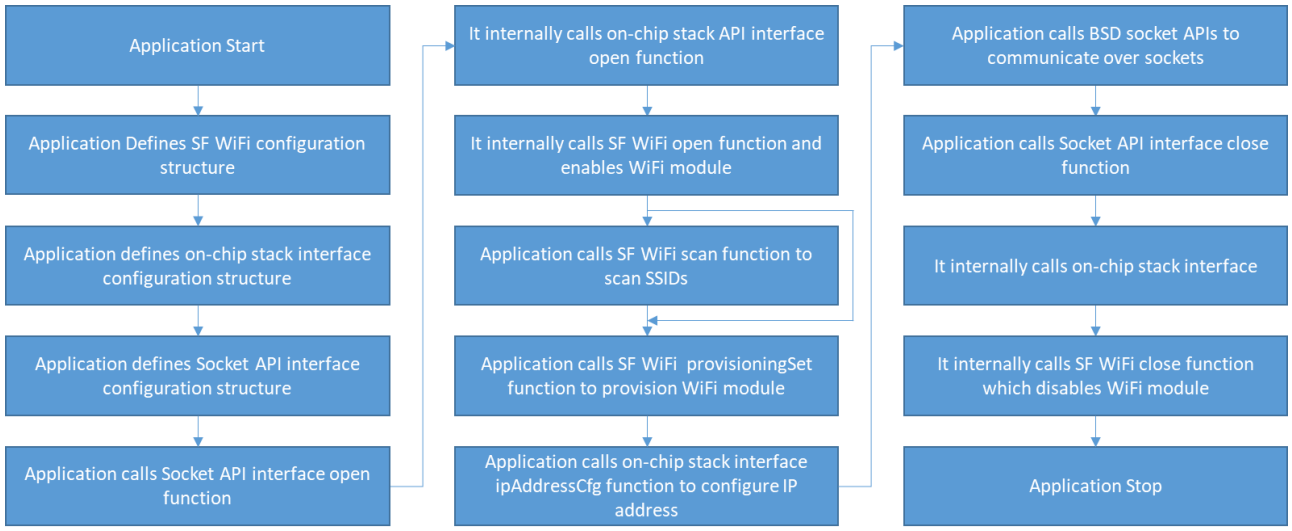


Figure 256: Application Flow Control using On-Chip Networking Stack

4.1.30 USBX Port DCD/HCD for USBFS/USBHS Framework

The Express Logic USBX Synergy Port framework module (sf_el_ux) is integrated into the SSP. This driver is meant to be used with Express Logic USBX. For more information about USBX, including API references, refer to the USBX User Guide.

4.1.30.1 Express Logic USBX Synergy Port Framework Module Features

The Express Logic USBX Synergy Port Framework module supports the following features:

- Implements Express Logic USBX in SSP- support USBX APIs
- Supports the Port Device Controller Driver (DCD) for the USBHS peripheral
- Supports the Port Device Controller Driver (DCD) for the USBFS peripheral
- Supports the Port Host Controller Driver (HCD) for the USBHS peripheral
- Supports the Port Host Controller Driver (HCD) for the USBFS peripheral
- Supports transfer module operation (optional)

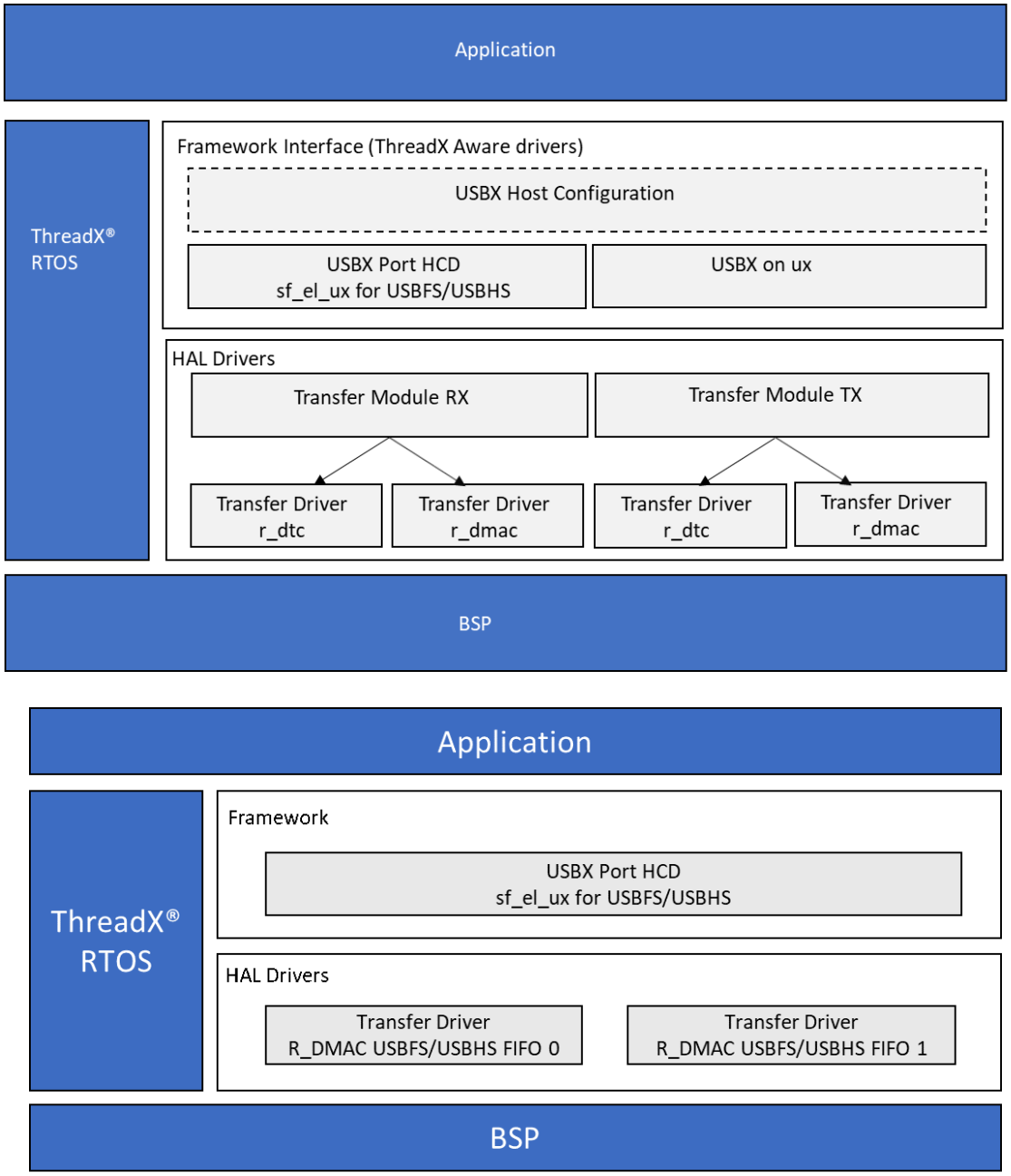


Figure 257: Express Logic USBX Synergy Port Framework Organization, Options and Stack Implementation for DCD and HCD

4.1.30.2 Express Logic USBX Synergy Port Framework Module APIs Overview

The Express Logic USBX Synergy Port Framework module doesn't have API calls of its own- it implements the API calls for the Express Logic USBX API calls. Documentation on these APIs is available in the Express Logic USBX User Manual.

4.1.30.3 Express Logic USBX Synergy Port Framework Module Operational Overview

The Express Logic USBX Synergy Port framework module provides the Synergy USB hardware port functions required to use the USBX stack on Synergy hardware. Application code using this module is expected to use USBX API calls.

Important Operational Notes and Limitations for the Express Logic USBX Synergy Port Framework Module

The Express Logic USBX Synergy Port framework module includes the support for the Express Logic USBX APIs in SSP. Refer to the Express Logic USBX User Manual for a complete description of the available APIs.

The Express Logic USBX Synergy Port framework module supports the Port Device Controller Driver (DCD) on USBHS and USBFS peripherals as well as the Port Host Controller Driver (HCD) for the USBHS on USBFS peripherals.

Users have the option of using the Transfer Module for the USBX Synergy Port framework module to get better USB data throughput by transferring data in the block transfer mode. To enable the Transfer module, just add two instances of transfer components to the USBX Class stack in the Synergy Configuration tool and enable the interrupts in the property. The Synergy Configuration tool auto-generates the driver setup code to enable DMAC or DTC transfer in common_data.c.

NOTE: The module uses the interrupt of a USB Controller. Set the appropriate interrupt priority level in the Synergy Configuration tool, otherwise it does not work.

The module uses the interrupt of a Transfer module (implemented as DMAC or DTC) if it is used. Set the appropriate priority level in the Synergy Configuration tool. The level must be higher than the level for the USB Controller, otherwise it does not work.

- Synergy USB controllers (USBHS and USBFS) have a limited number of PIPEs you can use for the isochronous transfer type (PIPE1 and PIPE2). This will limit the number of UVC devices (two devices) you can connect to.
- The device side driver (sf_el_ux DCD driver) does not support DTC as the transfer interface.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.1.30.4 Including the Express Logic USBX Synergy Port Framework Module in an Application

This section describes how to include the Express Logic USBX Synergy Port Framework module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few sections of the *SSP User's Manual* to learn how to manage each of these important steps in creating SSP-based applications.

The configurator has the capability to hide some modules from the user. This is done to simplify the selection of modules based on the context of the project. For example, some modules are not available for some MCUs, so if one of those MCUs is selected, these modules won't be made available to add to a thread. The Express Logic USBX Synergy Port framework module is usually not added as a standalone module- it is usually added as a lower level module to a higher-level USB module. For the advanced user, a mode can be set to show all modules, not just those the configurator 'thinks' are appropriate for the current project. This mode must be enabled in order to see the Synergy Port module, as illustrated in the following figures.

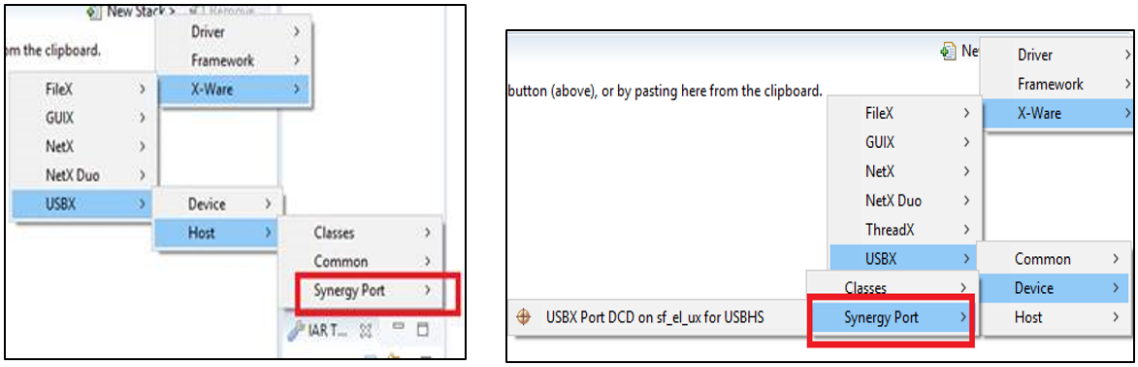


Figure 258: Selecting the USBX Synergy Port Module When Hidden

This mode can be enabled by turning on the Synergy module developer mode. Simply select the Window dropdown on the top line of the ISDE and then select Preferences>C/C++> Renesas>Synergy Configurator Editor. Then check the box for Synergy module developer mode. Then click Apply. This will allow the configurator to show all modules, not just those useable in the current project. Make sure to come back and disable this mode, after using it for a specific case, to continue to benefit from the context aware view.

Once you can view the Synergy Port selections, to use the USBX Synergy Port in an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the USBX Synergy Port is g_sf_el_ux_hcd/dcd_hs/fs_0. This name can be changed in the associated Properties window.)

USBX Synergy Port Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_el_ux_hcd_hs_0 USBX Port HCD on sf_el_ux for USBHS	Threads	New Stack> X-Ware> USBX> Host > Synergy Port> USBX Port HCD on sf_el_ux for USBHS
g_sf_el_ux_hcd_fs_0 USBX Port HCD on sf_el_ux for USBFS	Threads	New Stack> X-Ware> USBX> Host > Synergy Port> USBX Port HCD on sf_el_ux for USBFS
g_sf_el_ux_dcd_hs_0 USBX Port HCD on sf_el_ux for USBHS	Threads	New Stack> X-Ware> USBX> Device > Synergy Port> USBX Port HCD on sf_el_ux for USBHS
g_sf_el_ux_dcd_fs_0 USBX Port HCD on sf_el_ux for USBFS	Threads	New Stack> X-Ware> USBX> Device > Synergy Port> USBX Port HCD on sf_el_ux for USBFS

When the USBX Synergy Port Framework module on sf_el_ux is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information will have box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level drivers; these are either optional or recommended (this is indicated in the block with the inclusion of this text.) If the addition of lower-level drivers is

required, the module description will include "Add" in the text. Clicking on any Pink banded modules will bring up the "New" icon and then display the possible choices.

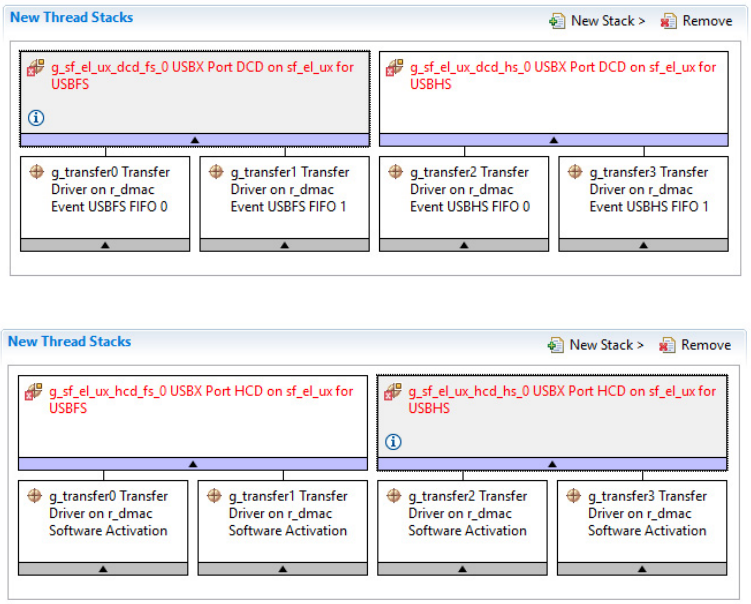


Figure 259: Express Logic USBX Synergy Port Framework Stacks

4.1.30.5 Configuring the Express Logic USBX Synergy Port Framework Module

The USBX Synergy Port Framework module must be configured by you for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority: this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for Express Logic USBX Port DCD on sf_el_ux for USBFS

ISDE Property	Value	Description
Full Speed Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Select the priority level of the full speed interrupt.
LDO Regulator (Only for S3 and S1 MCUs)	Enable, Disable Default: Disable	Select if the USB LDO regulator will be enabled.
Name	g_sf_el_ux_dcd_fs_0	Module name.
USB Controller Selection	USBFS	Select the USB controller.

Configuration Settings for Express Logic USBX Port DCD on sf_el_ux for USBHS

ISDE Property	Value	Description
Full Speed Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Select the interrupt priority for high speed USB.
Name	g_sf_el_ux_dcd_hs_0	Module name.
USB Controller Selection	USBHS	Select the USB controller.

Configuration Settings for Express Logic USBX Port HCD on sf_el_ux for USBFS

ISDE Property	Value	Description
High Speed Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Select the interrupt priority for full speed USB.
VBUSEN pin Signal Logic	Active High, Active Low (Default: Active High)	Select the VBUSEN pin signal logic.
LDO Regulator (Only for S3 and S1 MCUs)	Enable, Disable Default: Disable	Select if the LDO regulator will be enabled.
Name	g_sf_el_ux_hcd_fs_0	Module name.
USB Controller Selection	USBFS	Select the USB controller.

Configuration Settings for Express Logic USBX Port HCD on sf_el_ux for USBHS

ISDE Property	Value	Description
High Speed Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Select the interrupt priority for high speed USB.
FIFO size for Bulk/Isochronous Pipes	512, 1024, 1536, 2048 bytes Default: 512 bytes	Select the FIFO size for bulk and isochronous transfers.

ISDE Property	Value	Description
Number of Isochronous Pipes Reserved	0,1,2 Default: 0	Select the number of isochronous pipes to reserve.
VBUSEN pin Signal Logic	Active High, Active Low (Default: Active High)	Select the VBUSEN pin signal logic.
Enable High Speed	Enable, Disable Default: Enable	Select if high speed should be enabled.
Name	g_sf_el_ux_dcd_fs_0	Module name.
USB Controller Selection	USBFS	Select the USB controller.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the Express Logic USBX Synergy Port Framework Low Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

NOTE: One implementation of each of the lower level transfer drivers is provided. The other set of transfer drivers is very similar to the ones provided, with only minor changes. View the properties in the SSP configuration window to see the differences.

Configuration for the Transfer Driver on r_dmac Software Activation

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Name	g_transfer0	Module name
Channel	0	Channel selection

ISDE Property	Value	Description
Mode	Block	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source	Software Activation	Activation source selection
Auto Enable	False	Auto enable selection
Callback	NULL	Callback selection
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection

Configuration Settings for Transfer Driver on r_dmac Software Activation

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Name	g_transfer0	Module name

ISDE Property	Value	Description
Channel	1	Channel selection
Mode	Block	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source	Software Activation	Activation source selection
Auto Enable	False	Auto enable selection
Callback	NULL	Callback selection
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

Express Logic USBX Synergy Port Framework Clock Configuration

The USB peripheral module uses UCLK as its clock source. The clock frequency must be set to 48 MHz and this can be done in the Clocks tab of the configuration window.

Express Logic USBX Synergy Port Framework Pin Configuration

The USB peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window (depending on the USB function desired- USBFS or USBHS) and the subsequent tables illustrate example selections for the USB pins in each function.

NOTE: The operation mode determines what peripheral signals are available and thus what MCU pins are required. The operation mode selection must be consistent with the mode used by the USBX Device module.

Pin Selection Sequence for USBFS and USBHS

Resource	ISDE Tab	Pin
USBFS	Pins	Select Peripherals > Connectivity: USBFS> USBFS0
USBHS	Pins	Select Peripherals > Connectivity: USBHS> USBHS0

NOTE: The selection sequence assumes USBFS0 or USBHS0 are the desired hardware target for the driver.

Pin Configuration Settings for the USBX Device Module on USBFS0

Properties	Value	Description
Operation Mode	Device	Select Device as the Operation Mode
USBDP	USBDP	USPDP
USBDM	USBDM	USBDM
OVRCURB	None	OVRCURB
OVRCURA	None	OVRCURA
VBUSEN	None	VBUSEN
VBUS	None, P407 (Default: P407)	VBUS
EXICEN	None	EXICEN
ID	None	ID
VCCUSB	VCCUSB	VCCUSB

Properties	Value	Description
VSSUSB	VSSUSB	VSSUSB

NOTE: The example values are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

Pin Configuration Settings for the USBX Device USBHS0

Pin Configuration Property	Value	Description
Operation Mode	Device	Select Device as the Operation Mode
USBHSDP	USBHSDP	USBHSDP
USBHSDM	USBHSDM	USBHSDM
OVRCURB	None	OVRCURB
OVRCURA	None	OVRCURA
VBUSEN	None	VBUSEN
VBUS	None, PB01 (Default: PB01)	VBUS
EXICEN	None	EXICEN
ID	None	ID
USBHSRREF	USBHSRREF	USBHSRREF
AVCCUSBHS	AVCCUSBHS	AVCCUSBHS
AVSSUSBHS	AVSSUSBHS	AVSSUSBHS
PVSSUSBHS	PVSSUSBHS	PVSSUSBHS
VCCUSBHS	VCCUSBHS	VCCUSBHS
VSS1USBHS	VSS1USBHS	VSS1USBHS
VSS2USBHS	VSS2USBHS	VSS2USBHS

NOTE: The example values are for a project using the Synergy S7G2 Group and the SK-S7G2 Kit. Other Synergy Kits

and other Synergy MCUs may have different available pin configuration settings.

4.1.30.6 Using the Express Logic USBX Synergy Port Framework Module in an Application

Once the Express Logic Synergy Port USBX framework module is added to a thread the Express Logic APIs become available for use by higher level modules. There is usually no need to use the Express Logic Synergy Port USBX framework module directly in application code. Refer to the Express Logic USBX User manual if you need to use the module stand-alone and need to access the associated APIs directly.

4.2 HAL Layer

ACMPHS Driver
ACMPLP Driver
ADC Driver
AGT Driver
CAC Driver
CAN Driver
CGC Driver
CTSU Driver
CRC Driver
DAC Driver
DAC 8 Driver
GLCDC Display Driver
Data Operation Circuit Driver
DMAC Driver
DTC Driver
ELC Driver
External IRQ Driver
Flash Driver
FMI Driver
GPT Driver
I2C SCI Driver
I2C Master Driver
I2C Slave Driver
I2S Driver
Input Capture Driver

4.2.1 ACMPHS Driver

The ACMPHS HAL module implements the Comparator API for signal processing applications on r_acmphs. It supports the ACMPHS peripheral available on the Synergy microcontroller hardware. A callback is available to signal the user application on transition events.

4.2.1.1 ACMPHS HAL Module Features

- Callback on rising edge, falling edge, or both
- Configurable debounce filter
- Option to include comparator output on VCOUT pin

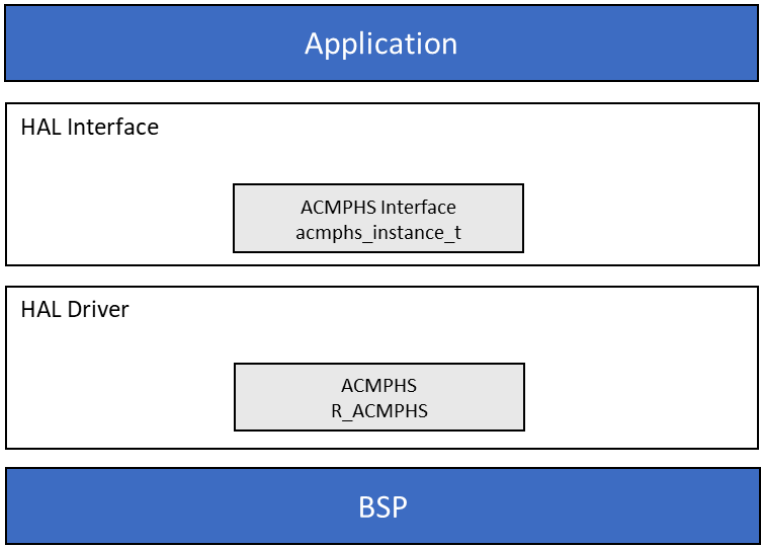


Figure 260: ACMPHS HAL Module Block Diagram

4.2.1.2 ACMPHS HAL Module APIs Overview

The ACMPHS HAL module defines APIs to open, enable, get status, and close the module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

ACMPHS HAL Module API Summary

Function Name	Example API Call and Description
.open	g_comparator0.p_api->open(g_comparator0.p_ctrl, g_comparator0.p_cfg); Configures the comparator and starts operation. Callbacks and pin output are not active until outputEnable() is called. outputEnable() should be called after the output has stabilized.
.outputEnable	g_comparator0.p_api->outputEnable(g_comparator0.p_ctrl); Enables the comparator output, which can be polled using statusGet(). Also enables pin output and interrupts as configured during open().

Function Name	Example API Call and Description
.infoGet	g_comparator0.p_api->infoGet(g_comparator0.p_ctrl, p_info); Provides the minimum stabilization wait time in microseconds..
.statusGet	g_comparator0.p_api->statusGet(g_comparator0.p_ctrl, p_status); Provides the operating status of the comparator.
.close	g_comparator0.p_api->close(g_comparator0.p_ctrl); Close the module.
.versionGet	g_comparator0.p_api->read(&version); Retrieves the version using the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_NOT_OPEN	Unit is not open
SSP_ERR_ASSERTION	An input pointer is NULL.
SSP_ERR_IN_USE	Peripheral is in use or hardware lock is taken.
SSP_ERR_TIMEOUT	The debounce filter is off and 2 consecutive matching values were not read within 1024 attempts.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.2.1.3 ACMPHS HAL Module Operational Overview

The ACMPHS HAL module controls the ACMPHS peripheral on a Synergy microcontroller. It directly controls the ACMPHS hardware without using any RTOS elements and provides convenient APIs to simplify development.

ACMPHS HAL Module Important Operational Notes and Limitations

Comparator Output on VCOUT Pin

The signal on the VCOUT pin is OR'd signal of all comparators with pin output enabled.

Interrupts and Callbacks

When a comparator event occurs, the ACMPHS HAL module calls the callback (comparator_api_t::p_callback) with the argument comparator_callback_args_t.

This module only works for selected Synergy MCUs. Consult the release notes for your current SSP release to see which MCUs are supported by this module. Additionally, the MCU Hardware Manual shows which peripherals are available.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.1.4 Including the ACMPHS HAL Module in an Application

This section describes how to include the ACMPHS HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the ACMPHS HAL module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the ACMPHS HAL module is g_comparator0. This name can be changed in the associated Properties window.)

ACMPHS HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_comparator0 Comparator Driver on r_acmphs	Threads	New Stack> Driver> Analog> Comparator Driver on r_acmphs

When the ACMPHS HAL module on r_acmphs is added to the thread stack as shown in the following figure, the configurator automatically adds any lower-level drivers that are required. Any drivers that need additional configuration information will be box-text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

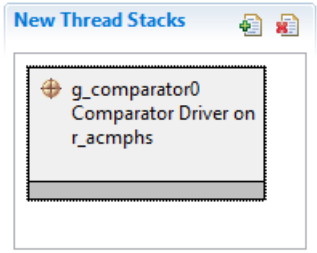


Figure 261: ACMPHS HAL Module Stack

4.2.1.5 Configuring the ACMPHS HAL Module

The ACMPHS HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the ACMPHS HAL Module on r_acmphs

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Controls whether to include code for API parameter checking.
Name	g_comparator	Module name.
Channel	0	Select the hardware channel.

ISDE Property	Value	Description
Trigger Edge	Rising, Falling, Both Edge Default: Both Edge	The trigger specifies when a comparator callback event should occur. Unused if the interrupt priority is disabled or the callback is NULL.
Debounce Filter	No Filter, 8, 16, 32 Default: No Filter	Select the PCLK divisor for the hardware digital debounce filter. Larger divisors provide a longer debounce and take longer for the output to update.
Invert	Not Inverted, Inverted Default: Not Inverted	Turns this on to invest comparator output. This affects the output read from StatusGet(), the pin output level, and the edge trigger.
Pin Output	Disabled, Enabled Default: Disabled	Turn this on to include the output from this comparator on VCOUT. The comparator output on VCOUT is ORed with output from all other ACMPHS and ACMPLP comparators.
Callback	NULL	Define this function in the application. It is called when the Trigger event occurs.
Comparator Interrupt Priority	Priority 0 (highest), Priority 1, Priority 2, Priority 3 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for the comparator interrupt.

NOTE: The example values and defaults are for a project using the Synergy S1JA. Other MCUs may have different default values and available configuration settings.

ACMPHS HAL Module Clock Configuration

The ACMPHS HAL module uses the PCLKB as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

ACMPHS HAL Module Pin Configuration

To use the ACMPHS HAL module, the port pins for the channels receiving the analog input must be set as input pins in the pin configurator in the ISDE. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection Sequence for the ACMPHS HAL Module

Resource	ISDE Tab	Pin selection Sequence
ACMPHS	Pins	Select Peripherals > Analog:ACMP

4.2.1.6 Using the ACMPHS HAL Module in an Application

Once the module has been configured and the files generated, the ACMPHS is ready to be used in an application.

The typical steps in using the ACMPHS HAL module in an application are:

1) Initialize the ACMPHS using the open API.
2) Before enabling the output, consult the hardware manual to configure the internal connections by setting the CMPSELn registers directly. If the internal reference voltage is used, set CPIOC.VREFEN.
3) After configuring the modules and internal connections, wait for the minimum stabilization wait time before enabling output. The minimum stabilization wait time can be queried using the infoGet API.
4) Enable the comparator output using the outputEnable API. This enables pin output, interrupts and the statusGet API as configured during the open API call.
5) [Optional] Use the statusGet API to poll comparator status.
6) [Optional] Use the close API to disable the comparator and power down the peripheral.

These common steps are illustrated in a typical operational flow in the following figure:

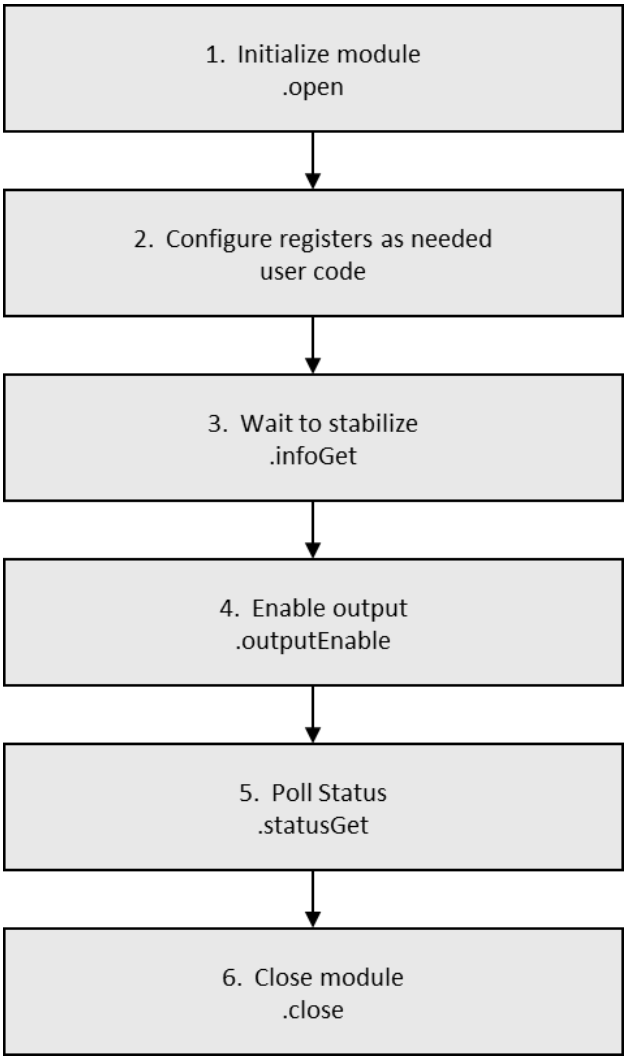


Figure 262: Flow Diagram of a Typical ACMPHS HAL Module Application

4.2.2 ACMPLP Driver

The ACMPLP HAL module implements the Comparator API for signal processing applications on r_acmplp. It supports the ACMPLP peripheral available on the Synergy microcontroller hardware. A callback is available to signal the user application on transition events.

4.2.2.1 ACMPLP HAL Module Features

- Normal mode or window mode
- Callback on rising edge, falling edge, or both
- Configurable debounce filter

- Option to include comparator output on VCOUT pin

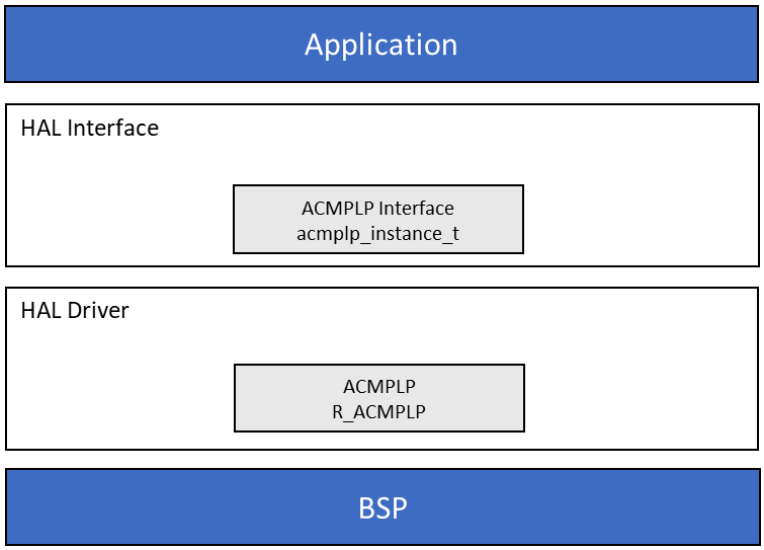


Figure 263: ACMPLP HAL Module Block Diagram

4.2.2.2 ACMPLP HAL Module APIs Overview

The ACMPHS HAL module defines APIs to open, enable, get status, and close the module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

ACMPLP HAL Module API Summary

Function Name	Example API Call and Description
.open	g_comparator0.p_api->open(g_comparator0.p_ctrl, g_comparator0.p_cfg); Configures the comparator and starts operation. Callbacks and pin output are not active until outputEnable() is called. outputEnable() should be called after the output has stabilized.
.outputEnable	g_comparator0.p_api->outputEnable(g_comparator0.p_ctrl); Enables the comparator output, which can be polled using statusGet(). Also enables pin output and interrupts as configured during open().

Function Name	Example API Call and Description
.infoGet	g_comparator0.p_api->infoGet(g_comparator0.p_ctrl, p_info); Provides the minimum stabilization wait time in microseconds..
.statusGet	g_comparator0.p_api->statusGet(g_comparator0.p_ctrl, p_status); Provides the operating status of the comparator.
.close	g_comparator0.p_api->close(g_comparator0.p_ctrl); Close the module.
.versionGet	g_comparator0.p_api->read(&version); Retrieves the version using the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_NOT_OPEN	Unit is not open
SSP_ERR_ASSERTION	An input pointer is NULL.
SSP_ERR_IN_USE	Peripheral is in use or hardware lock is taken.
SSP_ERR_TIMEOUT	The debounce filter is off and 2 consecutive matching values were not read within 1024 attempts.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.2.2.3 ACMPLP HAL Module Operational Overview

The ACMPLP HAL module controls the ACMPLP peripheral on a Synergy microcontroller. It directly controls the ACMPLP hardware without using any RTOS elements and provides convenient APIs to simplify development.

ACMPLP HAL Module Important Operational Notes and Limitations

Comparator Output on VCOUT Pin

The signal on the VCOUT pin is ORed signal of all comparators with pin output enabled.

Interrupts and Callbacks

When a comparator event occurs, the ACMPLP HAL module calls the callback (comparator_api_t::p_callback) with the argument comparator_callback_args_t.

This module only works for selected Synergy MCUs. Consult the release notes for your current SSP release to see which MCUs are supported by this module. Additionally, the MCU Hardware Manual shows which peripherals are available.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.2.4 Including the ACMPLP HAL Module in an Application

This section describes how to include the ACMPLP HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the ACMPLP HAL module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the ACMPLP HAL module is g_comparator0. This name can be changed in the associated Properties window.)

ACMPLP HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_comparator0 Comparator Driver on r_acmplp	Threads	New Stack> Driver> Analog> Comparator Driver on r_acmplp

When the ACMPLP HAL module on r_acmplp is added to the thread stack as shown in the following figure, the configurator automatically adds any lower-level drivers that are required. Any drivers that need additional configuration information will be box-text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

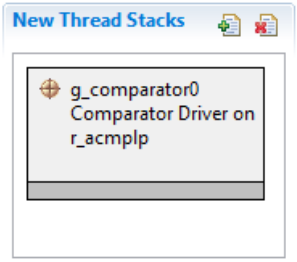


Figure 264: ACMPLP HAL Module Stack

4.2.2.5 Configuring the ACMPLP HAL Module

The ACMPLP HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the ACMPLP HAL Module on r_acmplp

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Controls whether to include code for API parameter checking.
Name	g_comparator	Module name.
Channel	0	Select the hardware channel.

ISDE Property	Value	Description
Mode	Mode Normal, Mode Window Default: Mode Normal	In normal mode, comparator output is high if VCMP > VREF. In window mode, comparator output is high if VCMP is outside the range of VREF0 to VREF1.
Trigger	Trigger Rising, Trigger Falling, Trigger Both Edge Default: Trigger Both Edge	The trigger specifies when a comparator callback event should occur. Unused if the interrupt priority is disabled or the callback is NULL.
Filter	Filter Off, Filter 1, Filter 8, Filter 16, Filter 32 Default: Filter Off	Select the PCLK divisor for the hardware digital debounce filter. Larger divisors provide a longer debounce and take longer for the output to update.
Invert	Off, On Default: Off	Turns this on to invest comparator output. This affects the output read from StatusGet(), the pin output level, and the edge trigger.
Pin Output	Off, On Default: Off	Turn this on to include the output from this comparator on VCOUT. The comparator output on VCOUT is ORed with output from all other ACMPHS and ACMPLP comparators.
Callback	NULL	Define this function in the application. It is called when the Trigger event occurs.
Comparator Interrupt Priority	Priority 0 (highest), Priority 1, Priority 2, Priority 3 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Select the interrupt priority for the comparator interrupt.

NOTE: The example values and defaults are for a project using the Synergy S1JA. Other MCUs may have different default values and available configuration settings.

ACMPLP HAL Module Clock Configuration

The ACMPHS HAL module uses the PCLKB as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

ACMPLP HAL Module Pin Configuration

To use the ACMPLP HAL module, the port pins for the channels receiving the analog input must be set as input pins in the pin configurator in the ISDE. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection Sequence for the ACMPLP HAL Module

Resource	ISDE Tab	Pin selection Sequence
ACMPLP	Pins	Select Peripherals > Analog:ACMP

4.2.2.6 Using the ACMPLP HAL Module in an Application

Once the module has been configured and the files generated, the ACMPLP is ready to be used in an application.

The typical steps in using the ACMPLP HAL module in an application are:

1) Initialize the ACMPLP using the open API.
2) Before enabling the output, consult the hardware manual to configure the internal connections by setting the COMPSELn registers directly. If the internal reference voltage is used, set COMPMDR.CiVRF.
3) After configuring the modules and internal connections, wait for the minimum stabilization wait time before enabling output. The minimum stabilization wait time can be queried using the infoGet API.
4) Enable the comparator output using the outputEnable API. This enables pin output, interrupts and the statusGet API as configured during the open API call.
5) [Optional] Use the statusGet API to poll comparator status.
6) [Optional] Use the close API to disable the comparator and power down the peripheral.

These common steps are illustrated in a typical operational flow in the following figure:

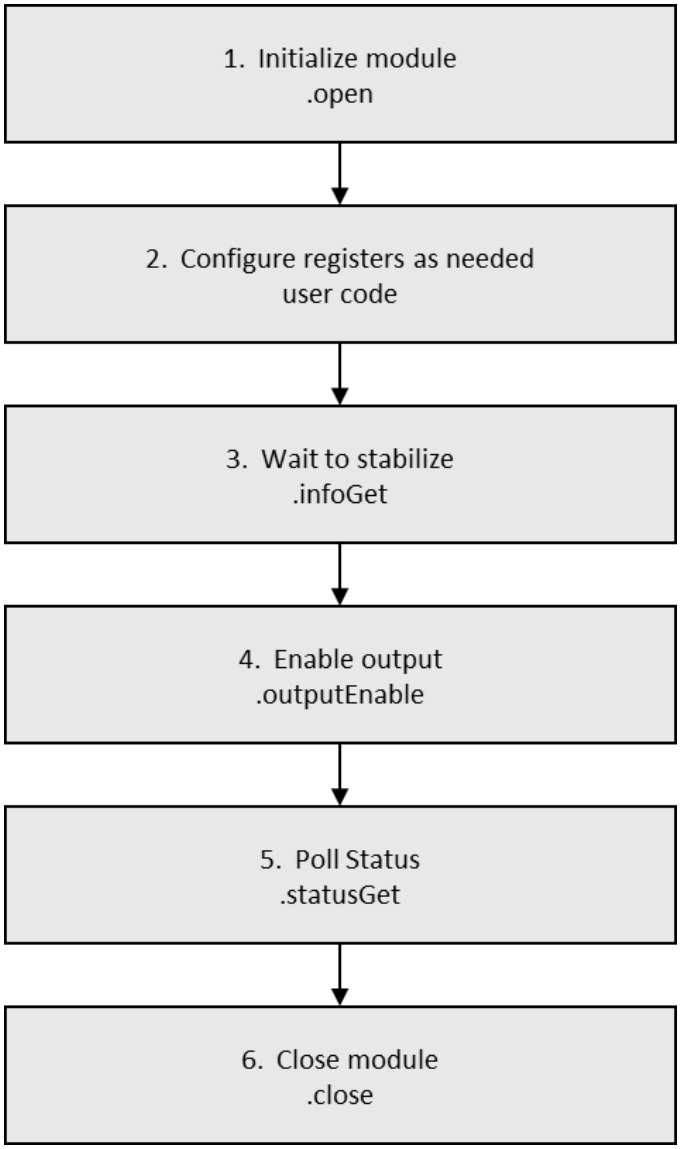


Figure 265: Flow Diagram of a Typical ACMPLP HAL Module Application

4.2.3 ADC Driver

The ADC HAL module implements APIs for analog-to-digital conversions. It supports the ADC12, ADC14 and ADC16 for the associated peripherals available on Synergy MCUs. A user-defined callback can be used to process the data each time a new sample is complete.

4.2.3.1 ADC HAL Module Features

- 16-Bit A/D Converter (S1JA), 14-Bit A/D Converter (S3A7,S3A3,S3A6,S3A1,S128,S124) and 12-Bit A/D Converter (S7G2,S5D9,S5D5)
- Multiple Operation Modes
  - Single Scan
  - Group Scan
  - Continuous Scan
- Multiple Channels
  - All analog channels on MCU
  - **13 channels (unit 0) or 12 channels (unit 1) for S7G2**
  - **17 channels for S1JA**
  - **18 channels for S124**
  - **28 channels for S3A7**
  - Temperature sensor channel
  - Voltage sensor channel

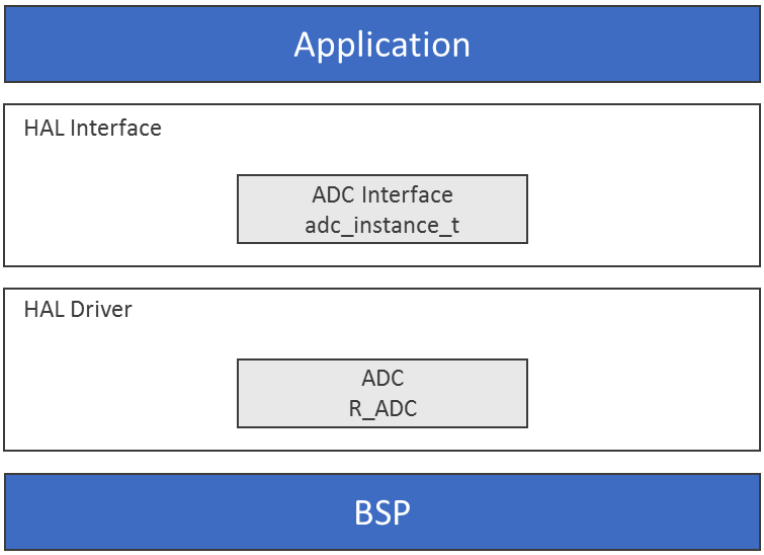


Figure 266: ADC HAL Module Block Diagram

4.2.3.2 ADC HAL Module APIs Overview

The ADC HAL module defines APIs to open, configure scans, start scans, stop scans, read the conversion results the ADC scans, and close the ADC unit. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

ADC HAL Module API Summary

Function Name	Example API Call and Description
.open	g_adc.p_api->open(g_adc.p_ctrl, g_adc.p_cfg); Initialize ADC Unit; apply power, set the operational mode, trigger sources, interrupt priority, and configurations common to all channels and sensors.
.scanCfg	g_adc.p_api->scanCfg(g_adc.p_ctrl, g_adc.*p*_channel_cfg); Configure the scan including the channels, groups and scan triggers to be used for the unit that was initialized in the open call.
.scanStart	g_adc.p_api->scanStart(g_adc.p_ctrl); Start the scan (in case of a software trigger), or enable the hardware trigger.
.scanStop	g_adc.p_api->scanStop(g_adc.*p_ctrl*); Stop the ADC scan (in case of a software trigger), or disable the hardware trigger.
.scanStatusGet	g_adc.p_api->scanStatusGet(g_adc.p_ctrl); Check scan status.
.read	g_adc.p_api->read(g_adc.p_ctrl, ADC_REG_CHANNEL_13, &adc_data); Read ADC conversion result(s).
.sampleStateCountSet	g_adc.p_api-> sampleStateCountSet(g_adc.p_ctrl,&adc_sample); Set the sample state count for the specified channel.

Function Name	Example API Call and Description
.close	g_adc.p_api->close(g_adc.p_ctrl); Close the specified ADC unit by ending any scan in progress, disabling interrupts, and removing power to the specified A/D unit.
.infoGet	g_adc.p_api->infoGet(g_adc.p_ctrl, &adc_info); Return the ADC data register address of the first (lowest number) channel and the total number of bytes to be read for the DTC/DMAC to read the conversion results of all configured channels.
.versionGet	g_adc.p_api->versionGet(&version); Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_NOT_OPEN	Unit is not open
SSP_ERR_ASSERTION	The parameter p_ctrl or p_sample is NULL.
SSP_ERR_IN_USE	Peripheral is still running in another mode; perform R_ADC_Close first.
SSP_ERR_INVALID_POINTER	The parameter p_data is NULL.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.2.3.3 ADC HAL Module Operational Overview

The ADC HAL module controls the ADC units on a Synergy microcontroller, as configured by the user. It directly controls the ADC hardware without using any RTOS elements and provides convenient APIs to simplify development. The driver supports three operation modes: single-scan, continuous-scan, and group-scan modes.

- Single-scan mode sequentially converts the analog inputs of selected channels in the ascending order of the channel number once per trigger.
- Continuous-scan mode sequentially converts the analog inputs of selected channels continuously in the ascending order of the channel numbers. A single trigger is required to start off the scan.
- Group-scan mode allows the user to add channels to one of two groups (A and B) and converts the analog inputs of the selected channels for each group in the ascending order of the channel number when the specified start trigger for that group is received.

In group-mode, operation channels can be assigned to one of two groups: group-A or group-B. A trigger is assigned for each group to start the scan. In group mode, only hardware triggers can be used, as opposed to normal mode, where software triggers or an external trigger can be used. With the priority configuration parameter, you can specify:

- Whether a trigger for one group can interrupt an ongoing scan for the other group.
- Whether an interrupted scan resumes or restarts or simply aborts the current scan and waits for the next trigger.

Interrupts and Callbacks

When a scan or calibration (on supported MCUs) is complete and a callback is provided to the user, (with interrupts enabled) the ADC HAL module invokes the callback (p_callback) with the argument adc_callback_args_t, indicating the unit and the event (adc_cb_event_t). If interrupts are not enabled, the scanStatusGet API supports checking the scan status to poll if the scan is complete and provides a function to read the converted ADC result. For MCUs that support Calibration, if the interrupts are not enabled, the user application must wait 24ms and then check the status of calibration using the infoGet API to see if the calibration is complete before using another API.

The ADC driver supports two interrupts:

- Normal/Group A Interrupt which fires when a scan is completed in single scan more or when a group A scan is completed in group mode or at the end of calibration for supported MCUs.
- Group B Interrupt which fires when a scan is completed a group B scan is completed in group mode.

Interrupts function differently in each mode:

- In single-scan mode, the Normal interrupt is triggered when the scan is completed.
- In continuous scan-mode, the hardware will constantly scan the selected channels. In this mode, the driver will return an error if interrupts are enabled and so must be disabled.
- In group mode, the ADC unit supports two interrupts: the Normal interrupt (also called group-A interrupt) and the group-B interrupt. The group-A interrupt is triggered when a group-A scan is completed. The group-B interrupt is triggered when a group-B scan is completed. In group mode, the normal interrupt is referred to as the group-A interrupt even though it is the same vector.

You must enable the ADC scan-complete interrupt for the selected unit in the BSP in the following situations:

- To get an interrupt when a normal scan is completed.
- To get an interrupt when a group scan is completed.
- To get an interrupt when a calibration is completed (for supported MCUs).

ADC HAL Module Important Operational Notes and Limitations

Sample-State Count Setting

The user can modify the setting of the sample-state count by calling the sampleStateCountSet API. You only need to modify the sample-state count settings from their default values if you want to increase the sampling time, either because the impedance of the input signal is too high to secure sufficient sampling time or the ADCLK is too slow. To modify the sample-state count for a given channel, set the channel number (adc_sample_state_reg_t) and the number of states (num_states). Valid sample state counts are 7-255. (Although the hardware supports minimum number of sample states as 5, it is 7 for some MCUs. It is fixed to 7 since at the lowest ADC conversion clock supported (1 MHz), this extra state will lead to at worst a "1 microsecond" increase in conversion time. At 60 MHz the extra sample state will add 16.7 ns to the conversion time). If you want to change the sample state count for multiple channels, call the sampleStateCountSet API repeatedly with modified arguments for each channel.

Triggering a Data Transfer with the ADC

To trigger a transfer of data when the ADC scan completes, configure the data transfer with activation_source set to ELC_EVENT_ADCn_SCAN_END or ELC_EVENT_ADCn_SCAN_END_B (where n is the ADC channel number). The ELC events are listed under elc_event_t. To retrieve the ADC unit-specific information to use with the transfer API, use the infoGet function call in the transfer API.

Triggering ELC Events with the ADC

The ADC unit can trigger the start of other peripherals listed in elc_peripheral_t. Refer to the "ELC Interface" in the SSP User's Manual for more information.

Using the Temperature Sensor with the ADC

The ADC HAL module supports reading the data from the on-chip temperature sensor. The value returned from the sensor can be converted into degrees Celsius or Fahrenheit by the user application using the following formula, T = (Vs – V1)/slope + T1, where

- T: Measured temperature (°C)
- Vs: Voltage output by the temperature sensor at the time of temperature measurement (Volts)
- T1: Temperature experimentally measured at one point (°C)
- V1: Voltage output by the temperature sensor at the time of measurement of T1 (Volts)
- T2: Temperature at the experimental measurement of another point (°C)
- V2: Voltage output by the temperature sensor at the time of measurement of T2 (Volts)
- Slope: Temperature gradient of the temperature sensor (V/°C); slope = (V2 – V1)/ (T2 – T1)

The slope value can be obtained from the hardware manual for each device under Electrical Characteristics, TSN Characteristics. The slope is positive for the S7/S5 devices and negative for the S3/S1 devices.

When configuring the ADC channels to be used with the ADC HAL module, the temperature and voltage sensors must not be selected if any of the other available channels are also selected. It is possible to only use the temperature sensor, voltage sensor, or any number of the regular ADC channels.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.3.4 Including the ADC HAL Module in an Application

This section describes how to include the ADC HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the ADC HAL module to an application, simply add it to a HAL /Common thread using the stacks selection

sequence given in the following table. (The default name for the ADC HAL module is g_adc0. This name can be changed in the associated Properties window.)

ADC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_adc0 ADC Driver on r_adc	Threads	New Stack> Driver> Analog> ADC Driver on r_adc

When the ADC HAL module on r_adc is added to the thread stack, as shown in the following figure, the configurator automatically adds any lower-level drivers that are required. Any drivers that need additional configuration information will be box-text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

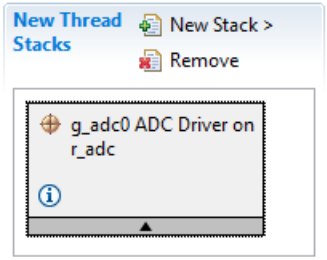


Figure 267: ADC HAL Module Stack

4.2.3.5 Configuring the ADC HAL Module

The ADC HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the ADC HAL Module on r_adc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: Enabled	If selected code for parameter checking is included in the build.
Name	g_adc0	Module name
Unit	0, 1 (S7G2 Only) Default: 0	Specify the ADC Unit to be used. The s7g2 has two units; 0 and 1.
Resolution (resolution varies by MCU)	14-Bit, 12-Bit, 10-Bit, 8_Bit Default: 8-Bit	Specify the conversion resolution for this unit.
Alignment	Right, Left Default: Right	Specify the conversion result alignment.
Clear after read	Off, On Default: On	Specify if the result register must be automatically cleared after the conversion result is read. NOTE: If this is enabled, then watching the result register using a debugger always results in a 0.
Mode	Single Scan, Continuous Scan, Group Scan Default: Single Scan	Specify the mode that this ADC unit is used in.
Internal Calibration During Open()	Enabled, Disabled Default: Enabled	Enable or Disable internal calibration during open. If enabled, open will only return after calibration is completed. The calibration times vary depending on PCLKB and ADCLK.

ISDE Property	Value	Description
Channels 0-13	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channel 14 (S3 Series Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channel 15 (S3A7/S3A3 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channels 16-20	Unused, Use in Normal/Group A, Use in Group B (Default: Unused)	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channel 21 (Unit 0 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.

ISDE Property	Value	Description
Channels 22-24	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channel 25 (S3 series only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channel 26 (S3A7/S3A3 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Channel 27 (S3A7/S3A3 Only)	Unused, Use in Normal/Group A, Use in Group B Default: Unused	In Normal mode of operation, this bitmask field is used to specify the channels that are enabled in that ADC unit. For example if it is set to 0x101, then channels 0 and 2 are enabled. In group mode, this field is used to specify which channels belong to group A.
Temperature Sensor	Unused, Use in Normal/Group A, Use in Group B Default: Unused	Temperature sensor use selection for Channel Scan Mask
Voltage Sensor	Unused, Use in Normal/Group A, Use in Group B Default: Unused	Voltage sensor use selection for Channel Scan Mask

ISDE Property	Value	Description
Normal/Group A Trigger	None, Asynchronous External Trigger 0, ELC Event, Software Default: Software	Specify the trigger type to be used for this unit. If group mode is used mode, then this field is used to set the Group A trigger. NOTE: The only valid option in group mode is the ELC trigger.
Group B Trigger (Valid Only in Group Scan Mode)	ELC Event (The only valid trigger for either group in Group Scan Mode)	Specify the group B trigger. This option is only valid if group mode is chosen in mode.
Group Priority (Valid only in Group Scan Mode)	Group A cannot interrupt Group B, Group A can interrupt Group B; Group B scan restarts at next trigger, Group A can interrupt Group B; Group B scan restarts immediately, Group A can interrupt Group B; Group B scan restarts immediately and scans continuously (Default: Group A cannot interrupt Group B)	Determines whether an ongoing group B scan can be interrupted by a group A trigger, whether it should abort on a group A trigger, or if it should pause to allow group A scan and restart immediately after group A scan is complete. NOTE: This field is valid only in group mode.
Add/Average Count	Disabled, Add two samples, Add three samples, Add four samples, Add sixteen samples, Average two samples, Average four samples Default: Disabled	Specify if addition or averaging needs to be done for any of the channels in this unit. The actual channels are specified by using a channel mask add_mask.
Channels 0-27	Disabled, Enabled Default: Disabled	This field is valid only if add_average_count is enabled. This field determines what channels results are to be averaged or summed.
Temperature Sensor	Disabled, Enabled Default: Disabled	Temperature sensor use selection for Addition/Averaging Mask

ISDE Property	Value	Description
Voltage Sensor	Disabled, Enabled Default: Disabled	Voltage sensor use selection for Addition/Averaging Mask
Channels 0-2	Disabled, Enabled Default: Disabled	Determines which of channels 0, 1 and 2 are using the updated sample-and-hold states value specified in sample_hold_states. This field must only be set if it is desired to modify the default sample and hold count value for channels 0, 1 and 2.
Sample Hold States (Applies only to the 3 channels selected above)	24	Specifies the updated sample-and-hold count for the channel dedicated sample-and-hold circuit. This field is valid only if sample_hold_mask is not 0. Only channels 0, 1 and 2 have dedicated sample and hold circuits. NOTE: Use this to modify the default number of states (24) for which the value is sampled. Each state is equal to 1/ADCLK time.
Callback	NULL	A user callback function can be registered in open. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the ADC scan completes. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

ISDE Property	Value	Description
Scan End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Scan End Interrupt Priority selection
Scan End Group B Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Scan End Group B Interrupt Priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

ADC HAL Module Clock Configuration

The ADC HAL module uses the PCLKC as its clock source (ADCLK.) The only restriction when configuring this clock is that it should be set to less than the max ADC clock; there is also a restriction on the ratio of the PCLKC and PCLKB clocks specified in the hardware manual.

The ADC-conversion time depends on the PCLKC setting.

To set the PCLKB and PCLKC frequencies, use the clock configurator in the ISDE.

To change the clock frequency at run-time, use the CGC Interface.

ADC HAL Module Pin Configuration

To use the ADC HAL module, the port pins for the channels receiving the analog input must be set as input pins in the pin configurator in the ISDE. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection Sequence for the ADC HAL Module

Resource	ISDE Tab	Pin selection Sequence
ADC	Pins	Select Peripherals > Analog Pins>ADC0/1>AN_XX

4.2.3.6 Using the ADC HAL Module in an Application

Once the module has been configured and the files generated, the ADC is ready to be used in an application.

The typical steps in using the ADC HAL module in an application are:

1) Initialize the ADC using the open API (If Calibration is enabled in the configuration, it will be performed as a part of the open call for the MCUs that support calibration).
2) Configure the channels using the scanCfg API. For MCUs that support calibration, if the calibration was not enabled in the configuration, then it must be performed before starting the first scan. Start the calibration (for supported MCUs) using the calibrate API.a. If interrupts are disabled wait for at least 24ms (for 32MHz PCLKB), check status using the infoGet API to insure the calibration is complete before using other ADC APIs.b. If interrupts are enabled, the callback will be invoked when the calibration is complete
3) Start a conversion using the desired trigger with the scanStart API. a If a hardware trigger is used, this call enables the ADC unit to b triggered by the hardware trigger. If a software trigger is used, then this call starts the ADC scan.
4) If interrupts are disabled, use the scanStatusGet API to determine if the scan is complete.
5) If interrupts are enabled, the callback will be invoked when the scan is complete
6) Read the results of the conversion using the read API
7) Stop the ADC scan by calling the scanStop APIa. This prevents the ADC from being triggered by an external trigger or a hardware trigger; it also forces a stop of a software-triggered scan if one is ongoing.
8) Operate on the received data as needed by the application.
9) Use the close API to power down the peripheral.

These common steps are illustrated in a typical operational flow in the following figure:

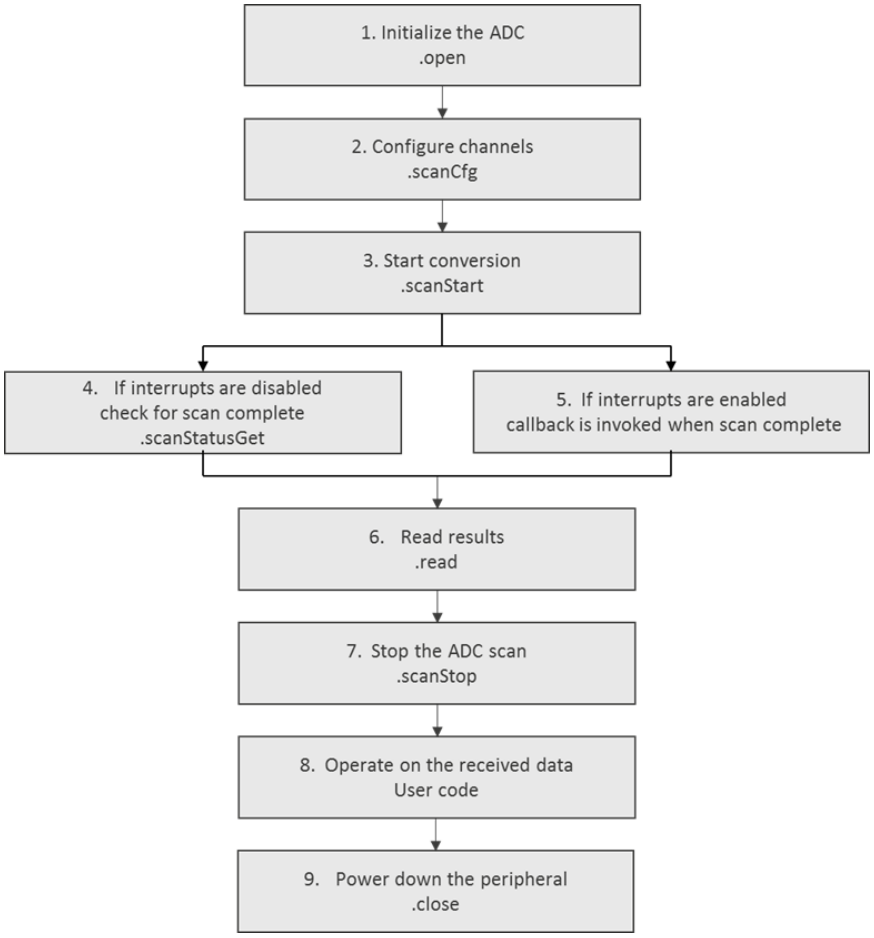


Figure 268: Flow Diagram of a Typical ADC HAL Module Application

4.2.4 AGT Driver

The AGT HAL module implements a high-level API for timing applications and is implemented on r_agt. The AGT HAL module uses the AGT peripheral on the Synergy MCU. A user-defined callback can be created to respond to a timer event.

The AGT HAL module configures a timer to a user-specified period. When the period elapses, any of the following events can occur:

- Interrupt the CPU, which calls a user-callback function (if provided)
- Toggle a port pin
- Transfer data using DMAC/DTC (if configured with transfer Interface)
- Start another peripheral (if configured with events and peripheral definitions)

4.2.4.1 AGT HAL Module Features

- Multiple Channels: 16-bit x 2 channels
  - Channel 1 can be clocked by the channel 0 underflow, creating a cascaded 32-bit timer
- Core Clock: Can be clocked using PCLKB, LOCO, or Fsub. When clocked by LOCO or Fsub, it can be used to wake up the MCU from sleep modes

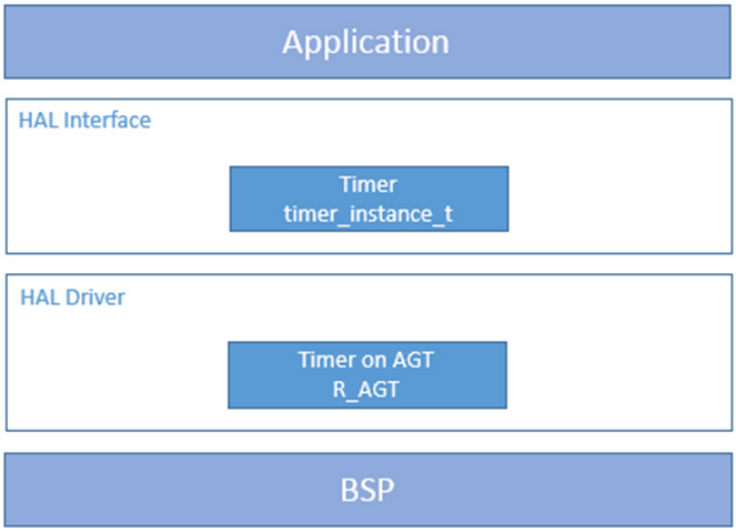


Figure 269: AGT HAL Module Block Diagram

4.2.4.2 AGT HAL Module APIs Overview

The AGT HAL module defines APIs for opening, closing, starting, and stopping timers. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows.

AGT HAL Module API Summary

Function Name	Example API Call and Description
open	g_timer0.p_api->open(g_timer0.p_ctrl, g_timer0.p_cfg) Initial configuration.
stop	g_timer0.p_api->stop(g_timer0.p_ctrl) Stop the counter.

Function Name	Example API Call and Description
start	g_timer0.p_api->start(g_timer0.p_ctrl) Start the counter.
reset	g_timer0.p_api->reset(g_timer0.p_ctrl) Reset the counter initial value.
counterGet	g_timer0.p_api->counterGet(g_timer0.p_ctrl, &value) Get current counter value and store it in the provided pointer, value.
periodSet	g_timer0.p_api->periodSet(g_timer0.p_ctrl, period, unit) Set the time until the timer expires.
dutyCycleSet	g_timer0.p_api->dutyCycleSet(g_timer0.p_ctrl, period, unit, pin) Sets the time until the duty cycle expires on the defined pin.
infoGet	g_timer0.p_api->infoGet(g_timer0.p_ctrl, &info) Get the time until the timer expires in clock counts and store it in provided pointer, info.
close	g_timer0.p_api->close(g_timer0.p_ctrl) Allows driver to be reconfigured and may reduce power consumption.
versionGet	g_timer0.p_api->versionGet(&version) Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Operation is successful
SSP_ERR_ASSERTION	Parameter is NULL or configuration setting is not allowed
SSP_ERR_IN_USE	The channel specified is already open
SSP_ERR_IRQ_BSP_DISABLED	A required interrupt is not enabled in the BSP
SSP_ERROR_NOT_OPEN	The channel is not open
SSP_ERR_INVALID_ARG	Invalid argument provided
SSP_ERR_INVALID_HW_CONDITION	Invalid hardware setting detected
SSP_ERR_INVALID_PTR	A pointer parameter was NULL, but needed a non-NULL value

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.4.3 AGT HAL Module Operational Overview

The AGT HAL module configures a timer to a user-specified period. When the period elapses, the CPU can be interrupted, a port pin can be toggled, a transfer of data using the DMAC or DTC can be initiated, or another peripheral can be triggered to begin operation.

The following figure shows a flowchart for toggling a port pin or generating a CPU interrupt after a specified period. This flowchart is appropriate for both AGT and GPT counters. (Replace the GPT references with AGT references for AGT operation and AGT is a down counter.)

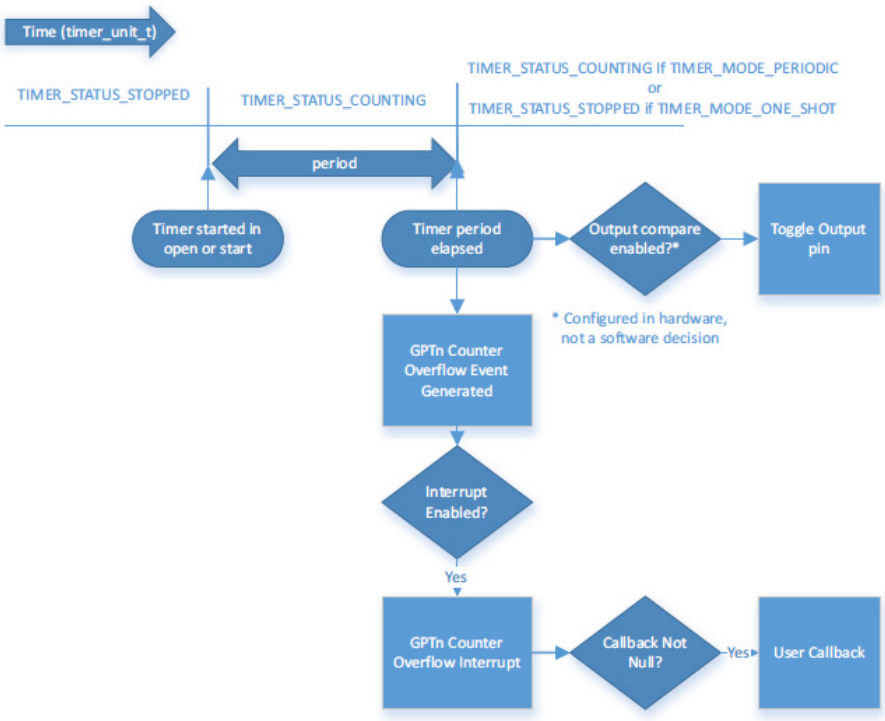


Figure 270: GPT Timer-Periodic or One-Shot Mode

Two different timer modules, the GPT and the AGT, are supported in the SSP. The following sections provide information on both modules so that the developer can compare and contrast the capabilities of each module for a particular application. For additional information on the GPT, refer to the GPT User's Guide.

The GPT module is recommended for most generic timer applications, but either module can be used for a basic timer functionality. The use cases in which one timer module would be preferred over the other are described as follows:

Selecting the GPT Timer Module

The GPT module uses a high-resolution 32-bit counter that can only be clocked by PCLKA. There are more GPT channels than AGT channels on Synergy devices, so using the GPT is less likely to cause a resource conflict.

Selecting the AGT Timer Module

The AGT module uses a 16-bit counter that can be clocked by PCLKB, LOCO, or Fsub. If clocked by LOCO or Fsub, the AGT interrupt can be used to wake the MCU from sleep modes. There are two channels, and channel 1 can be clocked by channel 0 underflow, effectively creating a 32-bit cascaded timer.

AGT HAL Module Important Operational Notes and Limitations

The maximum time period depends on the timer type and the input clock frequency.

- On a GPT with 32-bit resolution with PCLKA running at 120 MHz, the maximum period is approximately 36650 seconds, which is just over 10 hours.
- On a GPT with 16-bit resolution with PCLKA running at 32 MHz, the maximum period is approximately 2.09 seconds.
- On an AGT with 16-bit resolution with PCLKB running at 60 MHz, the maximum period is approximately 8.7 ms.

- On an AGT with 16-bit resolution with Fsub running at 32 kHz, the maximum period is approximately 2.0 seconds.

The AGT counter underflow interrupt for the selected channel used must be enabled in the BSP in the following situations:

1) To get a software interrupt when the timer period has elapsed.

2) To use one-shot mode.

When the AGTn AGTI interrupt is enabled in the BSP, the corresponding ISR is defined in the timer driver. The ISR calls a user-callback function if one was registered in open.

NOTE: Interrupts may be skipped when used with the DTC peripheral with the IRQ set to TRANSFER_IRQ_END.

AGT Output Timer Signal

If the timer output is configured, (AGTO Output Enabled set to true) the output pin starts at a high level if the output inverted is configured to True and a low level if it is configured to False. The output pin toggles every time the period elapses, beginning with the first time the period elapses after the timer is started.

In one-shot mode, the output is also configured to toggle when the timer starts counting. This generates a pulse - the timer toggles from the stop level when counting begins and toggles back to the stop level when counting ends.

Timer Period Calculation

The timer period is defined as the time until the timer expires. When output compare is used, the output pin toggles once per period, so the traditional period (from rising edge to rising edge) is twice the period specified in the software.

Runtime period calculation based on the current clock settings is available from open and periodSet.

If the specified timer period is different than the raw counts, the period is calculated using the current timer clock frequency (see Configuring the GPT Clocks or Configuring the AGT Clocks). The current timer clock frequency is determined using systemClockFreqGet. This frequency is used in the appropriate formula from the following table as clk_freq_hz.

Timer period calculation

Timer Units	Description
TIMER_UNIT_PERIOD_NSEC	Counts = (period * clk_freq_hz)/ 1000000000
TIMER_UNIT_PERIOD_USEC	Counts = (period * clk_freq_hz)/ 1000000
TIMER_UNIT_PERIOD_MSEC	Counts = (period * clk_freq_hz)/ 1000
TIMER_UNIT_PERIOD_SEC	Counts = (period * clk_freq_hz)
TIMER_UNIT_FREQUENCY_HZ	Counts = (clk_freq_hz)/ period
TIMER_UNIT_FREQUENCY_KHZ	Counts = (clk_freq_hz)/ (1000 * period)

If the requested period is larger than the counter size (32-bit or 16-bit), the driver selects the smallest divisor that allows the result to fit in the counter size. If the counter value is larger than the counter size with the largest divisor (1024), an error code (SSP_ERR_INVALID_ARGUMENT) is returned.

Triggering DMAC/DTC with GPT

To trigger a transfer of data using the DMAC or DTC peripheral when the timer period elapses, configure the DMAC/DTC transfer with activation_source set to ELC_EVENT_GPTn_COUNTER_OVERFLOW (where n is the GPT channel number). See the DMAC or DTC guides for further information.

NOTE: If you use the timer in one-shot mode with the DTC, the entire transfer completes before the interrupt stops the timer if the IRQ is set to TRANSFER_IRQ_END. To generate only one transfer after the timer period elapses, set the IRQ to TRANSFER_IRQ_EACH or use the DMAC for the transfer.

Triggering ELC Events with GPT

The GPT timer can trigger the start of other peripherals. The ELC guide provides a list of all available peripherals.

Triggering DMAC/DTC with AGT

To trigger a transfer of data using the DMAC or DTC peripheral when the timer period elapses, configure the DMAC/DTC transfer with activation_source set to ELC_EVENT_AGTn_AGTI (where n is the AGT channel number). See the Transfer Interface for further information.

NOTE: If you use the timer in one-shot mode with the DTC, the entire transfer completes before the interrupt stops the timer if irq is set to TRANSFER_IRQ_END. To generate only one transfer after the timer period elapses, set irq to TRANSFER_IRQ_EACH, or use the DMAC for the transfer.

Triggering ELC Events with AGT

The AGT timer can trigger the start of other peripherals. The ELC guide provides a list of all available peripherals listed in elc_peripheral_t. (See events and peripheral definitions for further information.)

Cascaded AGT Timers creating a 32-bit timer

AGT Channel 1 can be clocked by the AGT Channel 0 underflow, creating a cascaded 32-bit timer.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.4.4 Including the AGT HAL Module in an Application

This section describes how to include the AGT HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the AGT Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the AGT Driver is g_timer0. This name can be changed in the associated Properties window.)

AGT Driver Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
r_agt0 AGT Driver on r_agt	**Threads**	**New Stack> Driver> Timers> AGT Driver on r_agt**

When the AGT HAL module on r_agt is added to the thread stack as shown in the figure below, the configurator automatically adds any needed lower-level drivers. Any drivers that need additional configuration information will be box text highlighted in Red.

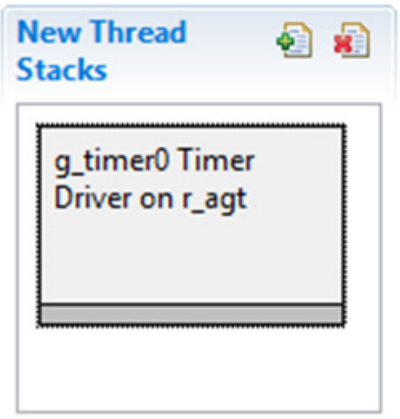


Figure 271: AGT HAL Module Stack

4.2.4.5 Configuring the AGT HAL Module

The AGT HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP Configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the AGT HAL Module on r_agt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables parameter checking.
Name	g_timer0	Module name.

ISDE Property	Value	Description
Channel	0	Physical hardware channel.
Mode	Periodic, One Shot Default: Periodic	Warning: One Shot functionality is not available in the GPT hardware, so it is implemented in software by stopping the timer in the ISR called when the period expires. For this reason, ISR's must be enabled for one-shot mode even if the callback is unused.
Period Value	10	See Timer Period Calculation
Period Unit	Raw Counts, Nanoseconds, Microseconds, Milliseconds, Seconds, Hertz, Kilohertz Default: Microseconds	See Timer Period Calculation
Auto Start	True, False Default: True	Set to true to start the timer after configuring or false to leave the timer stopped until start is called.
Count Source	PCLKB, PCLKB/8, PCLKB/2, LOCO, AGT0 Underflow, AGT0 fSub Default: PCLKB	The clock source for the AGT counter.
AGTO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTO pin). Set to false for no output of the timer signal.
AGTIO Output Enabled	True, False Default: False	Set to true to output the timer signal on a port pin configured for AGT (AGTIO pin). Set to false for no output of the timer signal.
Output Inverted	True, False Default: False	Set to false to start the output signal low. Set to true to start the output signal high.

ISDE Property	Value	Description
Callback	NULL	A user callback function can be registered in open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Timer interrupt priority. 0 is the highest priority.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults can be desirable. For example, it might be useful to select different period or duty cycle values. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

AGT HAL Module Clock Configuration

The AGT timer is clocked based on the PCLKB, LOCO, Fsub, or AGT Underflow frequency. The AGT clock is selectable in the Properties window in e^2 studio. You can set the clock frequencies using the clock configurator in e^2 studio Configuring Clocks or the CGC Interface at run-time.

AGT HAL Module Pin Configuration

The AGT peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the associated pins.

NOTE: The operation mode selection determines what peripheral signals are available and what MCU pins are required.

Pin Selection Sequence for AGT HAL Driver

Resource	ISDE Tab	Pin selection Sequence
AGT	Pins	Select **Peripherals > Timer: AGT>AGT0**

NOTE: The selection sequence assumes AGT0 is the desired hardware target for the driver.

Pin Configuration Settings for AGT HAL Driver

Pin Configuration Property	Value	Description
Operation Mode	Disabled, Custom, Timer Output, Compare Match, Count Measurement, Gated Count (Default: Disabled)	Select timer operation mode
AGTIO	None (Default: None)	AGTIO Pin
AGTO	None, P102 (Default: P102)	AGTO Pin
AGTOA	None (Default: None)	AGTOA Pin
AGTOB	None (Default: None)	AGTOB Pin
AGTEE	None, P101 (Default: P101)	AGTEE Pin

NOTE: The example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.4.6 Using the AGT HAL Module in an Application

The typical steps for using the AGT HAL module in an application are:

1) Initialize the AGT HAL module using the open API.
2) Start the AGT HAL module by calling the start API.
3) Read the counter value by calling the counterGet API.
4) Set the period value by using the periodSet API
5) Set the duty cycle (duty_cycle_counts) by using dutyCycleSet API
6) Get the timer information using infoGet API
7) Respond to the AGT HAL module callback as needed.
8) Resets the counter value using the reset API
9) Stop the AGT channel using stop API
10) Use the close call to power down the peripheral.

NOTE: The timer-period and duty-cycle parameters can be reconfigured based on the application's needs.

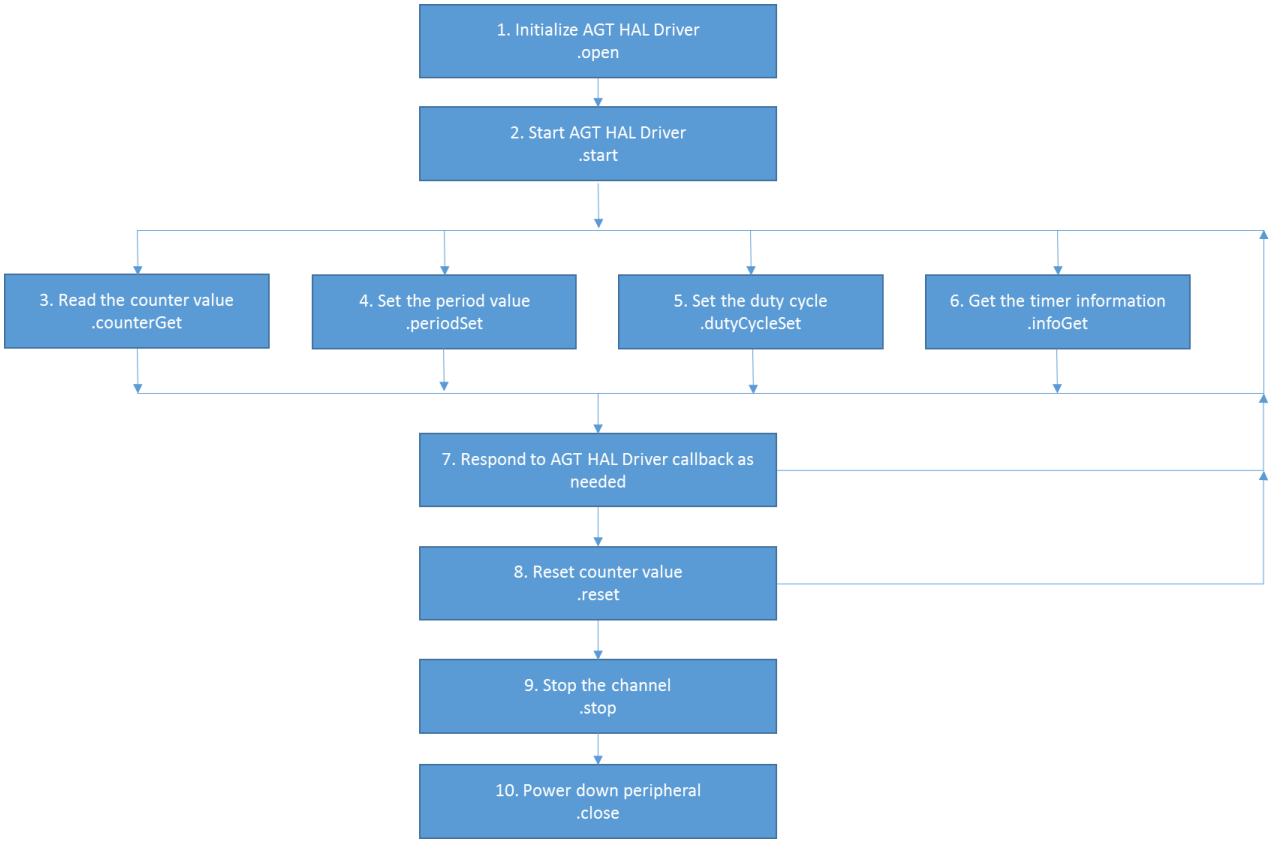


Figure 272: Typical AGT HAL Module Application Flow Chart

4.2.5 CAC Driver

The CAC HAL module is a high-level API for clock accuracy-control applications and is implemented on r_cac. The module uses the Clock Frequency Accuracy Measurement Circuit (CAC) peripheral on the Synergy MCU which is useful in implementing a fail-safe mechanism for reliability-oriented applications. A user-defined callback can be created to respond to various error indications.

4.2.5.1 CAC HAL Module Features

The CAC HAL module API interfaces with a clock frequency-measurement circuit capable of monitoring the clock frequency based on a reference-signal input. The reference signal may be an externally supplied clock source or one of several available internal clock sources. An interrupt request may optionally be generated by a completed measurement, a detected frequency error, or a counter overflow. A digital filter is available for an externally supplied reference clock, and dividers are available for both internally supplied measurement and reference clocks. Edge-detection options for the reference clock are configurable as rising, falling, or both.

The frequency of the following clocks can be measured:

- Clock output from main-clock oscillator (main clock)
- Clock output from sub-clock oscillator (sub clock)

- Clock output from high-speed on-chip oscillator (HOCO clock)
- Clock output from mid-speed on-chip oscillator (MOCO clock)
- Clock output from low-speed on-chip oscillator (LOCO clock)
- Clock output from IWDT-dedicated on-chip oscillator (IWDTCLK clock)
- Peripheral module clock (PCLKB)

The measurement clock is monitored using a reference clock. The reference clock may be an external clock (supplied on the CACREF input pin) or one of the following internal clocks:

- Clock output from main-clock oscillator (main clock)
- Clock output from sub-clock oscillator (sub clock)
- Clock output from high-speed on-chip oscillator (HOCO clock)
- Clock output from mid-speed on-chip oscillator (MOCO clock)
- Clock output from low-speed on-chip oscillator (LOCO clock)
- Clock output from IWDT-dedicated on-chip oscillator (IWDTCLK clock)
- Peripheral module clock (PCLKB)

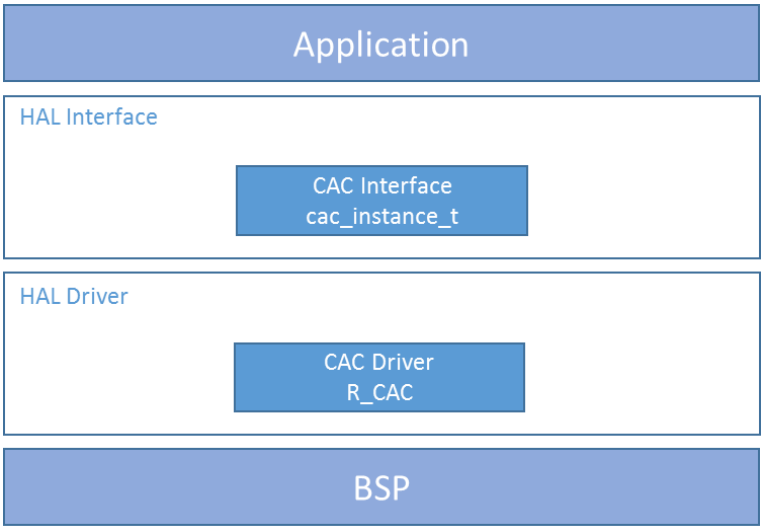


Figure 273: CAC HAL Module Block Diagram

4.2.5.2 CAC HAL Module APIs Overview

The CAC HAL module defines APIs for opening, closing, reading, starting, stopping, and resetting the CAC. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follow the API summary table.

CAC HAL Module API Summary

Function Name	Example API Call and Description
open	g_cac0.p_api->open(g_cac0.p_ctrl, g_cac0.p_cfg) Open function for CAC device.
read	g_cac0.p_api->read(g_cac0.p_ctrl, &cac0_status, &cac0_counter) Read function for CAC peripheral.
close	g_cac0.p_api->close(g_cac0.p_ctrl) Close function for CAC device.
stopMeasurement	g_cac0.p_api->stopMeasurement(g_cac0.p_ctrl) Ends a measurement for the CAC peripheral.
startMeasurement	g_cac0.p_api->startMeasurement(g_cac0.p_ctrl) Begin a measurement for the CAC peripheral.
reset	g_cac0.p_api->reset(g_cac0.p_ctrl) Reset function for CAC device.
versionGet	g_cac0.p_api->versionGet(&cac0_version) Get the CAC API and code version information.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful

Name	Description
SSP_ERR_INVALID_ARGUMENT	One or more configuration options are invalid
SSP_ERR_NOT_OPEN	Open has not been successfully called
SSP_ERR_ASSERTION	Null provided for p_ctrl, p_cfg and others
SSP_ERR_INVALID_POINTER	Interrupt specified with NULL callback
SSP_ERR_HW_LOCKED	Hardware lock for CAC peripheral is already taken
SSP_ERR_INVALID_CAC_REF_CLOCK	Measured clock rate smaller than reference clock rate

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.5.3 CAC HAL Module Operational Overview

The main function of the CAC HAL module is to measure the operation and accuracy of a selected clock. Once the measurement is requested, counting begins on the first valid edge detected for the reference clock and ends on the next valid edge. A valid edge can be configured to be rising, falling, or both. The count is incremented at each cycle of the measurement clock after it has passed through the divider circuit which is capable of dividing the clock by 1, 4, 8, or 32. An internally supplied reference clock also passes through a divider circuit which is capable of dividing the clock by 32, 128, 1024, or 8192. An externally supplied reference clock does not pass through a divider circuit, but may pass through a digital filter if it is configured to do so.

For example, if the sub-clock is specified as the measurement clock (32 kHz) and a divisor of 1 is specified, then the counter will increment at a 32 kHz rate. If a reference clock of 1 kHz is provided, then after one cycle of the reference clock you would expect the counter to be 32. This is where the CAC upper and lower-limit settings are examined. Part of the setup for a CAC measurement is the specification of an upper and lower limit for the measurement. When a measurement is complete, the CAC compares the counter contents to the limits configured for the measurement. If the counter is both ≤ upper limit and ≥ lower limit, then the measurement has completed without error and the measured frequency is operating within the defined limits. If the counter fails to meet these requirements, then a frequency error is indicated. A completed measurement may be identified by making API calls to poll the driver, or by establishing a callback function.

CAC HAL Module Important Operational Notes and Limitations

Continuous Mode

The CAC module may be operated in either a single or continuous measurement mode. In continuous mode, the measuring process is restarted after each completed measurement. In non-continuous, or single measurement mode, measuring stops after the first completed measurement. The continuous_mode configuration member controls this feature.

Interrupts and Callbacks

When a measurement is complete and a callback is provided by the user, (with one or more interrupts enabled) the CAC HAL module invokes the callback (p_callback) with the argument cac_callback_args_t, indicating the event cac_event_t. If interrupts are not enabled, the API supports checking the measurement status to poll if the measurement is complete (read), which will provide both the status of the measurement and the current value of the CAC counter register.

The CAC driver supports three interrupts:

- A Frequency error interrupt occurs when a measurement completes and the CAC counter register value is outside of the range that was specified as part of the CAC driver open. The configuration ferr_interrupt_enabled member provided in the open call must be enabled for this interrupt to be generated.
- An Overflow error interrupt occurs when the CAC counter register overflows its maximum (0xFFFF) value. The configuration ovf_interrupt_enabled member provided in the open call must be enabled for this interrupt to be generated.
- A measurement complete interrupt occurs when a measurement completes and the CAC counter register value is within the range that was specified as part of the CAC driver open. The configuration mei_interrupt_enabled member provided in the open call must be enabled for this interrupt to be generated.

Reset

The reset API can be used to reset the overflow, measurement end, and frequency Error interrupt flags after measurement has been stopped to eliminate any false triggers.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.5.4 Including the CAC HAL Module in an Application

This section describes how to include the CAC HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the CAC driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the CAC is g_cac0. This name can be changed in the associated Properties window.)

CAC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_cac0 CAC Driver on r_cac	Threads > HAL/Common	New Stack > Driver> Monitoring > Clock Accuracy Circuit Driver on r_cac

When the CAC HAL module on r_cac is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information will be box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

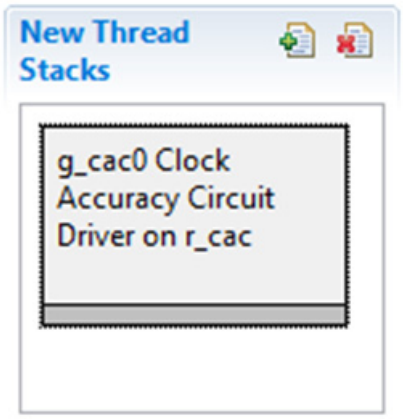


Figure 274: CAC HAL Module Stack

4.2.5.5 Configuring the CAC HAL Module

The CAC HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the configuration table settings given below. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the CAC HAL Module on r_cac

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Controls whether to include code for API parameter checking.
Name	g_cac0	Identifies this instance

ISDE Property	Value	Description
Continuous Measurement Operation	Enabled, Disabled Default: Enabled	If Enabled, measurement will continuously restart after completing
Measurement Complete Interrupt	Enabled, Disabled Default: Enabled	Enabling allows the CAC driver to generate an interrupt when a measurement is complete, providing the CAC MEASUREMENT END interrupt is enabled in the ICU.
Overflow Interrupt	Enabled, Disabled Default: Enabled	Enabling allows the CAC driver to generate an interrupt when a CAC overflow occurs, providing the CAC OVERFLOW interrupt is enabled in the ICU.
Frequency Error Interrupt	Enabled, Disabled Default: Enabled	Enabling allows the CAC driver to generate an interrupt when a frequency error occurs, providing the CAC FREQUENCY ERROR interrupt is enabled in the ICU.
Upper Limit Threshold	0 – 0xFFFF Default: 0	Top end of allowable range for measurement complete
Lower Limit Threshold	0 – 0xFFFF, must be < Upper Limit threshold Default: 0	Bottom end of allowable range for measurement complete
Reference Clock Source	Main Oscillator, Sub-clock, HOCO, MOCO, LOCO, PCLKB, IWDT Default: Main Oscillator	Reference clock source
Reference Clock Divider	32,128,1024,8192 Default: 32	Reference clock divider
Reference Clock Edge Detect	Rising, Falling, Both Default: Rising	Reference clock edge detection

ISDE Property	Value	Description
Reference Clock Digital Filter	Disabled, Sampling clock = Measuring freq, Sampling clock = Measuring freq/4, Sampling clock = Measuring freq/16 Default: Disabled	Reference clock Digital filter
Measurement Clock Source	Main Oscillator, Sub-clock, HOCO, MOCO, LOCO, PCLKB, IWDT Default: HOCO	Measurement clock source
Measurement Clock Divider	1,4,8,32 Default: 1	Measurement clock divider
Callback	Null	Function name for callback
Frequency Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid), Disabled Default: Disabled	CAC Frequency Error interrupt priority
Measurement End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid), Disabled Default: Disabled	CAC Measurement End interrupt priority

ISDE Property	Value	Description
Overflow Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid), Disabled Default: Disabled	CAC Overflow interrupt priority

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for the module can be desirable. For example, it might be useful to select different clock sources and dividers.

CAC HAL Module Clock Configuration

Clocks are selected from the Clock tab as needed by the application.

CAC HAL Module Pin Configuration

The pins for the CAC HAL module are selected as shown in the following table; the pin settings are shown in the subsequent table. If CACREF is used to the input pin of reference clock, you must configure this pin.

Pin Selection Sequence for the CAC HAL Module

Resource	ISDE Tab	Pin selection Sequence
CAC HAL Module	Pins	Peripherals > Monitoring: CAC > CAC0

NOTE: The selection sequence assumes CAC0 is the desired hardware target for the driver.

Pin Configuration Settings for the CAC HAL Module

Pin Configuration Property	Value	Description
Operation Mode	Disabled, External Reference (Default: Disabled)	Select Enable as the Operation Mode for CAC

Pin Configuration Property	Value	Description
CACREF:	None, P204 (Default: None)	CACREF Pin

NOTE: The example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.5.6 Using the CAC HAL Module in an Application

The typical steps in using the CAC HAL module in an application are:

1) Initialize the CAC HAL module using the open API
2) Start a measurement using the startMeasurement API
3) Poll using the CGC read function to look for the measurement result or get measurement status and result using callback function called in ISR
4) Stop the measurement using the CGC stopMeasurement API
5) Reset the overflow, measurement end, and frequency error interrupt flags using the CGC reset API
6) Close the CAC HAL module if no more measurements are needed using the CGC close API

These common steps for using the CAC HAL module are illustrated in a typical operational flow diagram in the figure below:

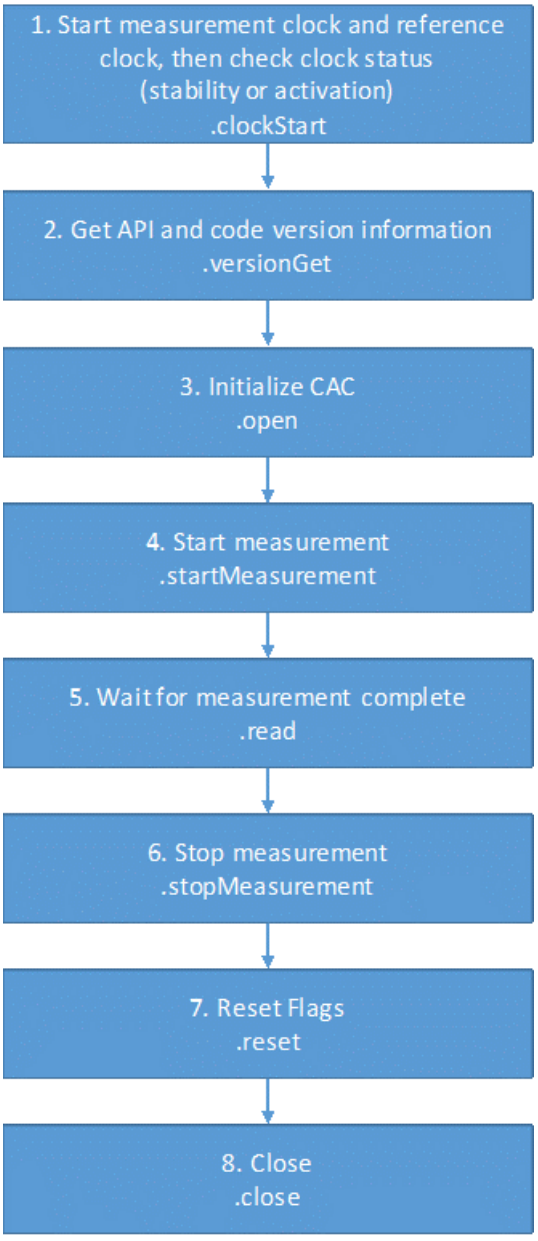


Figure 275: Flow Diagram of a Typical CAC HAL Module Application

4.2.6 CAN Driver

The CAN HAL module provides high-level APIs for a CAN network. The CAN HAL module is implemented on r_can and supports the CAN peripherals available on the Synergy microcontroller hardware. A user-callback function must be defined, which the driver will invoke when transmit, receive or error interrupts are received. The callback returns with a parameter which indicates the channel, mailbox and event.

4.2.6.1 CAN HAL Module Features

- Supports both standard (11-bit) and extended (29-bit) messaging format
- Support for bit-timing configuration as defined in the CAN specification
- Support for up to 32 transmit or receive mailboxes with standard or extended ID frames
- Receive mailboxes can be configured to capture either data or remote CAN frames
- Supports a user-callback function when transmit, receive or error interrupts are received

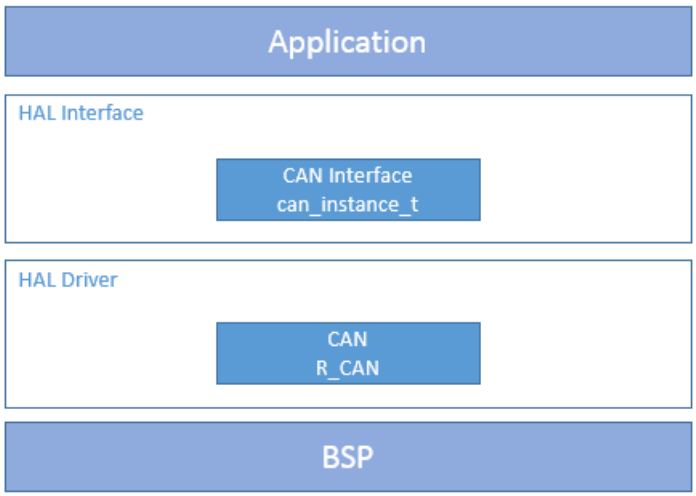


Figure 276: CAN HAL Module Organization, Options, and Stack Implementations

4.2.6.2 CAN HAL Module APIs Overview

The CAN HAL defines APIs for opening, closing, writing, (transmitting) and reading (receiving) CAN data; it also provides some additional functions, such as control, Infoget and VersionGet to assist in processing more complex commands. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

CAN HAL Module API Summary

Function Name	Example API Call and Description
open	g_can0.p_api->open(g_can0.p_ctrl, g_can0.p_cfg) The open API configures CAN Channel 0. This function must be called before any other CAN functions. Note: This call is made automatically during system initialization, prior to entering the users thread. Unless the user closes the module, open will not need to be called.
close	g_can0.p_api->close(g_can0.p_ctrl) The close API handles the clean-up of internal driver data.
read	g_can0.p_api->read (g_can0.p_ctrl, p_args->mailbox, &receiveFrame) The read API reads received CAN data.
write	g_can0.p_api->write (g_can0.p_ctrl, 0, &transmitFrame) The write API write data into the CAN transmit frame buffer and send it out.

control	g_can0.p_api->control(g_can0.p_ctrl, CAN_COMMAND_MODE_SWITCH, &mode);
	
	with can_mode_t mode = CAN_MODE_LOOPBACK_INTERNAL;
	
	The control API will be able to change the CAN mode of operation.

infoGet	g_can0.p_api->infoGet(g_can0.p_ctrl, p_info) The infoGet API retrieves the CAN mode of operation.

Function Name	Example API Call and Description
versionGet	g_can0.p_api->versionGet(version) The versionGet API retrieves the module version information.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_HW_LOCKED	Lock already owned by another user.
SSP_ERR_CAN_MODE_SWITCH_FAILED	Channel failed to switch modes.
SSP_ERR_CAN_INIT_FAILED	Channel failed to initialize.
SSP_ERR_ASSERTION	Null pointer presented.
SSP_ERR_NOT_OPEN	Port is not open.
SSP_ERR_CAN_DATA_UNAVAILABLE	No data available.
SSP_ERR_CAN_TRANSMIT_MAILBOX	Mailbox is not setup for receive.
SSP_ERR_CAN_TRANSMIT_NOT_READY	Transmit in progress, cannot write data at this time.
SSP_ERR_CAN_RECEIVE_MAILBOX	Mailbox is setup for receive and cannot send.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.6.3 CAN HAL Module Operational Overview

The CAN HAL module controls the CAN peripherals on Synergy microcontrollers according to the user configuration. The API provides open, close, read, write, control and information functions. The driver allows for bit-timing configuration as defined in the CAN specification; it can be configured for up to 32 transmit or receive mailboxes with

standard or extended ID frames. Receive mailboxes can be configured to capture either data or remote CAN frames. The user callback is invoked with the channel, mailbox and event information when a transmit, receive or error interrupt occurs.

Using the CAN IDs and Masks

Each CAN Mailbox configured to receive messages has an ID and Mask set. Incoming messages will be placed in the lowest mailbox where the following is true:

Incoming ID & Mailbox Mask == Mailbox ID & Mailbox Mask

Example 1: A mailbox with an ID of 0x25 and a mask of 0x1FFFFFFF can receive messages with IDs of 0x25.

Example 2: A mailbox with an ID of 0x25 and a mask of 0x1FFFFFF0 can receive messages with IDs of 0x20 through 0x2F.

Using the CAN HAL Module Test Modes

The CAN Module has three test modes: listen only, external loopback and internal loopback.

- In listen-only mode, valid data frames and remote frames can be received; however, only recessive bits can be sent on the CAN bus. The ACK bit, overload flag, and active error flag cannot be sent. Listen-only mode can be used for baud rate detection
- In external-loopback mode, the protocol module treats its own transmitted messages as those received by the CAN transceiver and stores them into the receive mailbox. To be independent from external stimulation, the protocol module generates the ACK bit. Connect the CTX and CRX pins to the transceiver.
- Internal-loopback mode is similar to external-loopback mode, excepting that the protocol controller performs internal loopback from the internal CTX pin to the internal CRX pin. The input value of the external CRX pin is ignored. The external CTX pin outputs only recessive bits. The CTX and CRX pins are not required to be connected to the CAN bus or any external device.

Changing the CAN HAL Module Operating Modes

The CAN Module can be switched between modes using the control API. Pass CAN_COMMAND_MODE_SWITCH and a pointer to a can_mode_t variable set to the desired mode into the control API.

Mode	can_mode_t value	Reason for use
Normal	CAN_MODE_NORMAL	Normal operation mode
Internal Loopback	CAN_MODE_LOOPBACK_INTERNAL	Internal loopback testing
External Loopback	CAN_MODE_LOOPBACK_EXTERNAL	External loopback testing
Listen Only	CAN_MODE_LISTEN	Baud rate detection
Halt	CAN_MODE_HALT	Mailbox configuration and test mode setting
Sleep	CAN_MODE_SLEEP	Stops the clock supply to the CAN module reducing current consumption

Mode	can_mode_t value	Reason for use
Exit Sleep	CAN_MODE_EXIT_SLEEP	Internal use only
Reset	CAN_MODE_RESET	Communication configuration

NOTE: To see an example use of the control API for setting the loopback mode, refer to the CAN HAL Module API Summary table presented earlier in this document.

CAN HAL Module Important Operational Notes and Limitations

- The user application must start the main-clock oscillator (CANMCLK or XTAL) at run-time using the CGC Interface if it has not already started (for example, if it is not used as the MCU clock source.)
- For S7, S5, S3 and S1 MCUs, the following clock restriction must be satisfied for the CAN HAL module when the clock source is the main-clock oscillator (CANMCLK): fPCLKB >= fCANCLK (XTAL / Baud Rate Prescaler)
- For S7, S5 and S3 MCUs, the source of the peripheral module clocks must be PLL for the CAN HAL module when the clock source is PCLKB.
- For S3 MCUs, the clock frequency ratio of PCLKA and PCLKB must be 2:1 when using the CAN HAL module. Operation is not guaranteed for other settings.
- For S1 MCUs, the clock frequency ratio of ICLK and PCLKB must be 2:1 when using the CAN HAL module. Operation is not guaranteed for other settings.
- SJW (Synchronization Jump Width) is often given by the bus administrator. Select 1 <= SJW <= 4.
  - Time segment and SJW settings must adhere to the following constraints: TS1 > TS2 >= SJW.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.6.4 Including the CAN HAL Module in an Application

This section describes how to include the CAN HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the CAN HAL module to an application, simply add it to a HAL/ Common Stack or any thread using the stacks selection sequence given in the following table. (The default name for the CAN HAL module is g_can0. This name can be changed in the associated Properties window.)

CAN HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_can0 CAN Driver on r_can	Threads	New Stack> Driver> Connectivity> CAN Driver on r_can

When the CAN HAL module on r_can is added to the thread stack as shown in the following figure, the configurator automatically adds the needed lower-level drivers. Any drivers that need additional configuration information will be highlighted in red.

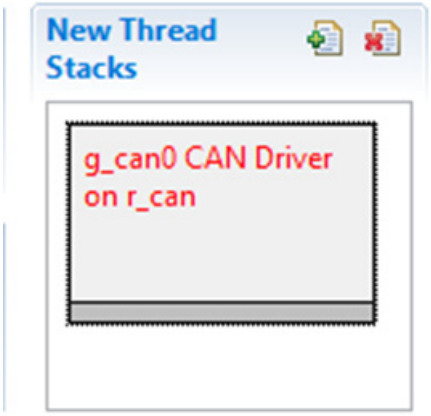


Figure 277: CAN HAL Module Stack

4.2.6.5 Configuring the CAN HAL Module

The CAN HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less-error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP Configurator, and are shown in the following tables for easy reference.

NOTE: You may want to open your ISDE, create the CAN HAL module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the CAN HAL Module on r_can

ISDE Property	Value	Description
Parameter Checking	Enabled, Disabled, BSP (Default: BSP)	Selects if code for parameter checking is to be included in the build.
Error Interrupt Priority	Disabled, Priority 0-15 (Default: Disabled)	Specify the error interrupt priority 0-15 (required).

ISDE Property	Value	Description
Receive Mailbox Interrupt Priority	Disabled, Priority 0-15 (Default: Disabled)	Specify the error interrupt priority 0-15 (required).
Transmit Mailbox Interrupt Priority	Disabled, Priority 0-15 (Default: Disabled)	Specify the error interrupt priority 0-15 (required).
Name	Default: g_can0	CAN driver module name.
Channel	0, 1	Specify if the CAN channel to use. 0 or 1(S7G2 only).
Baud Rate Prescaler	Default 5	Specify the baud rate prescaler(0-1023).
Time Segment 1	15 Time Quanta	Specify the time segment 1 value. (4-16).
Time Segment 2	8 Time Quanta	Specify the time segment 2 value (2-8).
Synchronization Jump Width	2 Time Quanta	Specify the Synchronization Jump Width value (1-4).
Clock Source	PCLKB (S7G2, S5D9, S5D5, S3A7, and S3A7 only), CANMCLK Default: CANMCLK	CAN clock source, CANMCLK or PCLKB (S7G2, S5D9, S5D5, S3A7, and S3A7 only).
Callback	NULL	A user callback function can be registered in can_api_t::open. If this callback function is provided, it is called from the interrupt service routine (ISR) each time any interrupt occurs.
Overwrite/Overrun mode	Overwrite Mode, Overrun mode Default: Overwrite mode	Select whether receive mailbox will be overwritten or overrun if data is not read in time.

ISDE Property	Value	Description
Standard or Extended ID Mode	Standard ID Mode, Extended ID Mode Default: Standard ID Mode	Select whether the driver will be using CAN standard or extended IDs.
Number of Mailboxes	32 Mailboxes	Select 4, 8, 16 or 32 mailboxes.
Mailbox [0] ID … Mailbox [31] ID	Default n	Select the receive ID for mailbox 0, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Mailbox [0] Type … Mailbox [31] Type	N = 0: Transmit Mailbox; N = 1 …31: Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Mailbox [0] Frame Type … Mailbox [31] Frame Type	N = 0: Remote Mailbox; N = 1 …31: Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (receive only).
Mailbox 0-3 Group Mask	Default: 0x1FFF FFFF	Select the Mask for mailboxes 0-3.
Mailbox 4 - 7 Group Mask	Default: 0x1FFF FFFF	Select the Mask for mailboxes 4-7.
Mailbox 8 - 11 Group Mask	Default: 0x1FFF FFFF	Select the Mask for mailboxes 8 -11.
Mailbox 12 - 15 Group Mask	Default: 0x1FFF FFFF	Select the Mask for mailboxes 12 -15.
Mailbox 16 - 19 Group Mask	Default: 0x1FFF FFFF	Select the Mask for mailboxes 16 -19.
Mailbox 20 - 23 Group Mask	Default: 0x1FFF FFFF	Select the Mask for mailboxes 20 -23.
Mailbox 24 - 27 Group Mask	Default: 0x1FFF FFFF	Select the Mask for mailboxes 24 -27.
Mailbox 28 - 31 Group Mask	Default: 0x1FFF FFFF	Select the Mask for mailboxes 28 -31.

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings. Most of the property settings for lower-level modules are fairly intuitive and usually can be determined by inspection of the associated Properties window from the SSP configurator.

CAN HAL Module Clock Configuration

The CAN peripheral uses the CANMCLK (main-clock oscillator) or PCLKB (S7G2, S5D9, S5D5, S3A7, and S3A7 only) as its clock source (fCAN, CAN System Clock.) Using the PCLKB with the default of 60 MHz and the Synergy default, (S7G2 DK) CAN configuration will provide a CAN bit rate of 500 Kbit.

To set the PCLKB frequency, use the clock configurator in e^2 studio (Clocks tab in the configurator.)

To change the clock frequency at run-time, use the CGC Interface. Refer to the CGC module guide for more information on configuring clocks.

CAN HAL Module Pin Configuration

The CAN peripheral module uses the pins on the MCU to communicate to external devices connected on the CAN bus. Under Peripherals, select CAN and then CAN0 for channel 0 or CAN1 (S7G2 and S3A7 only) for channel 1. The operation mode for the channel must be enabled and the CRXn and CTXn pins must be selected to match your PC board layout. The pin configurator sets appropriate CAN pin configuration in the pin_cfg field for the associated pin. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the CAN pins.

NOTE: The operation mode selection determines what peripheral signals are available and thus what MCU pins are required.

Pin Selection Sequence for CAN Driver on r_can

Resource	ISDE Tab	Pin selection Sequence
CAN	Pins	Select **Peripherals > Connectivity CAN>CAN0**

NOTE: The selection sequence assumes CAN0 is the desired hardware target for the driver.

Pin Configuration Settings for CAN0

Pin Configuration Property	Value	Description
Operation Mode	Disabled, Enabled	Enable the mode to use CAN0
CRX	None, P202, P402 (Default: P402)	CAN0_CRX0
CTX	None, P203 P401 (Default: P401)	CAN0_CTX0

NOTE: The example values are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

CAN Bit Rate Calculation

A time quanta (Tq) is one bit-time of the CAN communication clock, fCANCLK. This is not the CAN bit-time but the internal clock period of the CAN peripheral. The frequency is determined by the baud-rate prescaler value and the CAN source clock, fCAN (CANMCLK or PCLKB). One bit-time is divided into a number of time quanta, Tqtot. One time

quantum is equal to the period of the fCANCLK. Each bitrate register is then given a certain number of Tq of the total of Tq that make up one CAN-bit period. The default ISDE bitrate setting (S7G2 DK template) is 500 Kbps for a fCAN at 60 MHz (using PCLKB.)

The formulas to calculate the bitrate register settings are as follows:

fCAN = (fPCLKB or fCANMCLK)

The baud-rate prescaler scales the CAN peripheral clock down.

fCANCLK = fCAN/ Baud Rate Prescaler = 60 MHz (default)/ 5(default) = 12 MHz

One time quantum is one clock period of the CAN clock.

Tqtot =1/fCANCLK

Tqtot is the total number of CAN peripheral clock cycles during one CAN bit time and is by the peripheral built by the sum of the "time segments" and "SS" which is always 1.

Tqtot = TSEG1 + TSEG2 + SS (TSEG1 must be > TSEG2) = 15 + 8 + 1 = 24 (default)

The bitrate is then:

Bitrate = fCANCLK/Tqtot = 12 MHz / 24 = 500Kbps

Important notes:

- The user application must start the main-clock oscillator (CANMCLK or XTAL) at run-time using the CGC Interface if it is not already started (for example, if it is not used as the MCU clock source.)
- For S7G2, S3A7 and S124 MCUs, the following clock restriction must be satisfied for the CAN module when the clock source is the main-clock oscillator (CANMCLK): fPCLKB >= fCANCLK (fCAN / baud-rate prescaler)
- For S7G2 and S3A7 MCUs, the source of the peripheral module clocks must be PLL for the CAN HAL module when the clock source is PCLKB.
- For S3A7 MCUs, the clock frequency ratio, PCLKA and PCLKB must be 2:1 when using the CAN HAL module. Operation is not guaranteed for other settings.
- For S124 MCUs, the clock frequency ratio of ICLK and PCLKB must be 2:1 when using the CAN HAL module. Operation is not guaranteed for other settings.
- SJW (Synchronization Jump Width) is often given by the bus administrator. Select 1 <= SJW <= 4.

Configurator sample values for different CAN bit-rate

fCAN (either PCLKB or CAN MCLK)	Baud Rate Prescalar	fCANCLK = fCAN/Baud e Rate Prescalar	Time Segment 1 (TSEG1)	Time Segment 2 (TSEG2)	Synchroni zation Jump Width (SS)	Tqtot = TSEG1 + TSEG2 +SS	Bitrate = fCANCLK/ Tqtot
240	10	24	15	8	1	24	1 Mbps
60	5	12	15	8	1	24	500 kbps
240	48	5	16	2	2	20	250 kbps
240	96	2.5	16	2	2	20	125 kbps

Setting the Mailbox Group Masks

There are 8 mailbox group-masks, one for each group of 4 mailboxes. These masks allow the mailboxes to be configured to receive more than one ID. If the mask is all ones (0x7ff for standard IDs or 0x1FFFFFFF for extended ID) the mailboxes within the group do not mask any bits of the ID, requiring all bits of the mailbox ID to match the mailbox ID before a message is captured. If any bits of the mask are set to zero, those bits will not be necessary to match the same bits of the mailbox ID. For example, if Mailbox ID 1 is set to 0x7ff and Mailbox 0-3 Group Mask is set to 0x7ff, mailbox 1 will only capture messages with the ID of 0x7ff. If the mailbox 0-3 group mask is set to 0x7fe, mailbox 1 will still capture messages with IDs of 0x7f but will also capture messages with IDs of 0x7fe.

4.2.6.6 Using the CAN HAL Module in an Application

A CAN application requires a minimum of two nodes to demonstrate CAN communication. One node can be a transmitter, while the other can be a receiver (or both can behave as transmitter and receiver.)

The typical steps in using the CAN module in an application are:

1) Initialize the CAN HAL module using the open API.
2) (Optional) Enter internal loopback or external loopback test modes using control API.
3) (Optional) Information about the module status, including bit rate, can be retrieved using the infoGet API.
4) To transmit a message:a. Create and configure the CAN frame ensuring correct ID and frame type.b. Write the CAN frame to a mailbox configured in transmit mode using the write API.
5) To receive a message:a. Read from a mailbox that has received a frame using the read API.
6) Close the CAN HAL module using the close API (if needed.)

These common steps are illustrated in a typical operational flow diagram in the following figure:

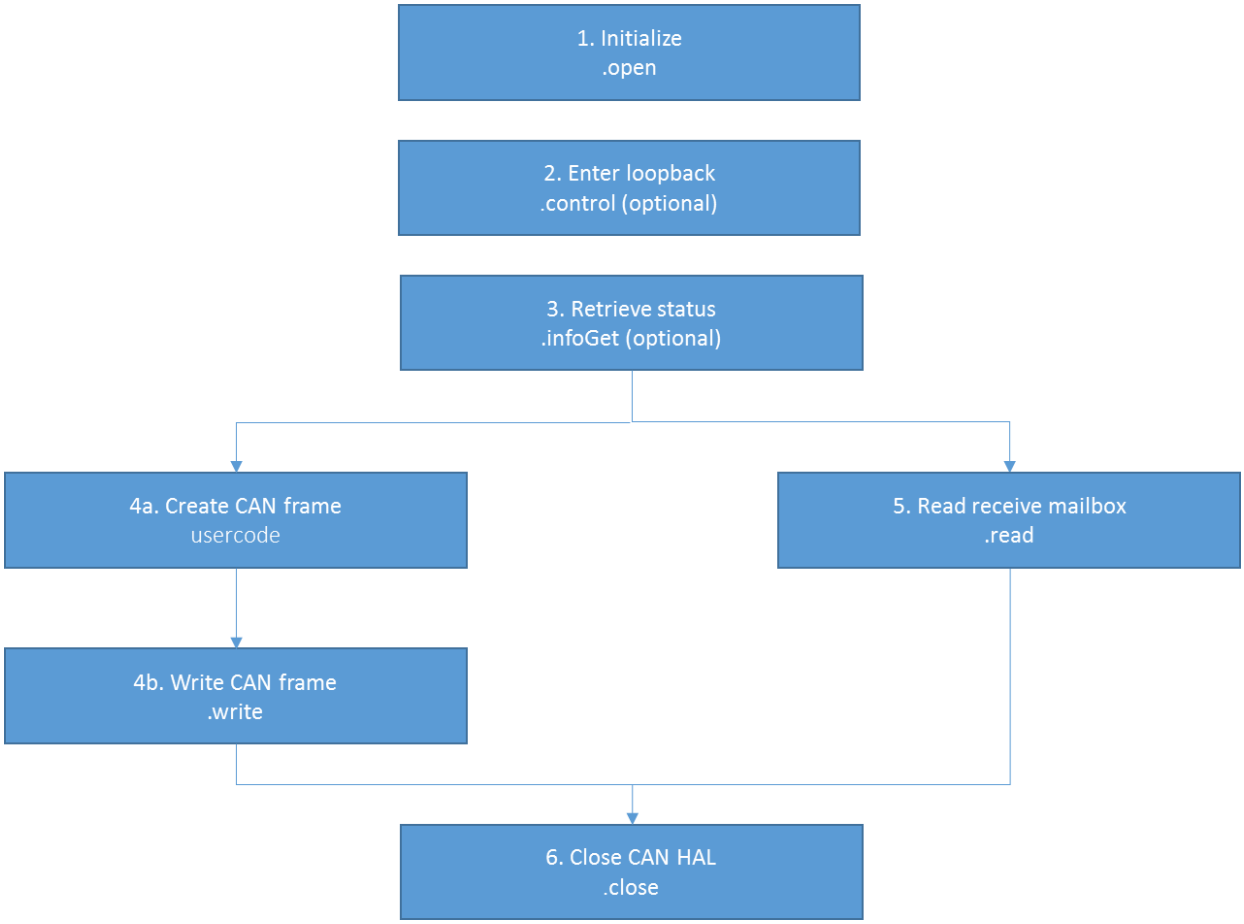


Figure 278: Flow Diagram of a Typical CAN HAL Module Application

4.2.7 CGC Driver

The CGC HAL module provides high-level APIs for clock-control applications and is implemented on r_cgc. The CGC HAL module configures and controls the clock-control functions of a Synergy MCU using the clock-control peripheral. Since every project requires a clock function, the CGC HAL module is added to a project by default. (The module is configured in the ISDE.) A user-defined callback can be created to signal when the main oscillator has stopped.

4.2.7.1 CGC HAL Module Features

The CGC HAL module supports the configuration and control of the various clocking function on the Synergy MCU. Key features include the following:

- Select the system clock source
  - HOCO (high-speed on-chip oscillator), MOCO (middle-speed on-chip oscillator), LOCO (low-speed on-chip oscillator), Main Clock, PLL, or Sub-Oscillator
- Configure internal clocks and turn them on or off

- Configure the output clocks
- Set up the Oscillation Stop Detection feature
- Set up clock divisors on each of the up to six clock domains
- Some Synergy MCUs also support controllable external clock outputs, which may have independent divisors

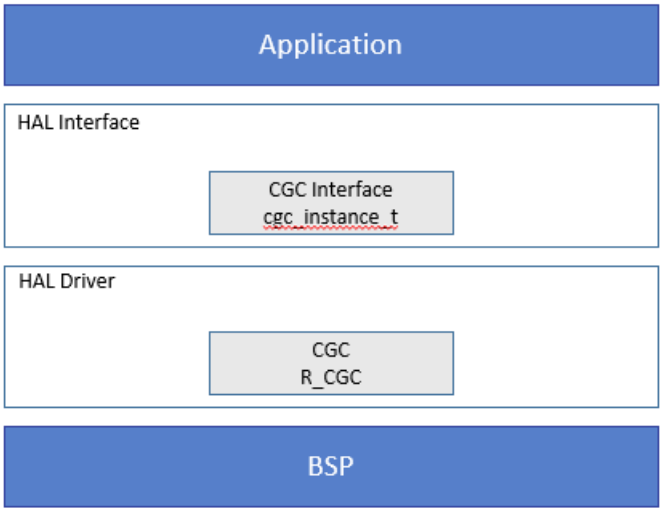


Figure 279: CGC HAL Module Block Diagram

4.2.7.2 CGC HAL Module APIs Overview

The CGC HAL module defines APIs for initializing, starting, controlling, and stopping the MCU clock. A complete list of the available APIs, an example API call, and a short description of each can be found in the following API Summary table. A table of status return values are listed after the API summary.

CGC HAL Module API Summary

Function Name	Example API Call and Description
.init	g_cgc.p_api->init(); Initial clock configuration called by BSP automatically.
.clocksCfg	g_cgc.p_api->clocksCfg(&p_clock_cfg); The BSP calls this function at startup, but it can also be called from the application to change clocks at runtime.

Function Name	Example API Call and Description
.clockStart	g_cgc.p_api->clockStart(clock_source, &p_clock_cfg); Start a clock.
.clockStop	g_cgc.p_api->clockStop(clock_source); Stop a clock.
.systemClockSet	g_cgc.p_api->systemClockSet(clock_source, &p_clock_cfg); Set the system clock.
.systemClockGet	g_cgc.p_api->systemClockGet(&clock_source, &clock_config); Get the system clock information.
.systemClockFreqGet	g_cgc.p_api->systemClockFreqGet(&clock_source, &frequency_hz); Return the frequency of the selected clock.
.clockCheck	g_cgc.p_api->clockCheck(clock_source); Check the stability of the selected clock.
.oscStopDetect	g_cgc.p_api->oscStopDetect(callback, enable); Configure the Main Oscillator stop detection.
.oscStopStatusClear	g_cgc.p_api->oscStopStatusClear(); Clear the oscillator stop detection flag.

Function Name	Example API Call and Description
.busClockOutCfg	g_cgc.p_api->busClockOutCfg (divider); Configure the bus clock output secondary divider. The primary divider is set using the BSP clock configuration and the systemClockSet function (S7G2 and S3A7 MCU only).
.busClockOutEnable	g_cgc.p_api->busClockOutEnable (); Enable the bus clock output (S7G2 and S3A7 MCU only).
.busClockOutDisable	g_cgc.p_api->busClockOutDisable (); Disable the bus clock output (S7G2 and S3A7 MCU only).
.clockOutCfg	g_cgc.p_api->clockOutCfg(clock_source, clock_dividers); Configure clockOut.
.clockOutEnable	g_cgc.p_api->clockOutEnable(); Enable clock output on the CLKOUT pin. The source of the clock is controlled by clockOutCfg.
.clockOutDisable	g_cgc.p_api->clockOutDisable(); Disable clock output on the CLKOUT pin. The source of the clock is controlled by clockOutCfg.
.lcdClockCfg	g_cgc.p_api->lcdClockCfg(clock); Configure the segment LCD Clock (S3A7 and S124 MCUs only).
.lcdClockEnable	g_cgc.p_api->lcdClockEnable(); Enable the LCD clock (S3A7 and S124 MCUs only).

Function Name	Example API Call and Description
.lcdClockDisable	g_cgc.p_api->lcdClockDisable(); Disables the LCD clock (S3A7 and S124 MCUs only).
.sdramClockOutEnable	g_cgc.p_api->sdramClockOutEnable(); Enables the SDRAM clock output (S7G2 MCU only).
.sdramClockOutDisable	g_cgc.p_api->sdramClockOutDisable(); Disables the SDRAM clock (S7G2 only).
.usbClockCfg	g_cgc.p_api->usbClockCfg(divider); Configures the USB clock (S7G2 only).
.systickUpdate	g_cgc.p_api->ssystickUpdate(period_count, units); Update the Systick timer.
.versionGet	g_cgc.p_api->versionGet(&version); Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_ABORTED	Attempt to update systick timer failed.
SSP_ERR_HARDWARE_TIMEOUT	Hardware timed out.
SSP_ERR_STABILIZED	Clock stabilized.
SSP_ERR_CLOCK_INACTIVE	Clock not turned on.

Name	Description
SSP_ERR_MAIN_OCO_INACTIVE	Main OCO off/unstable.
SSP_ERR_CLOCK_ACTIVE	Clock active.
SSP_ERR_NOT_STABILIZED	Clock source un-stabilized.
SSP_ERR_CLKOUT_EXCEEDED	Clock out exceeded.
SSP_ERR_NULL_PTR	Pointer null.
SSP_ERR_OSC_DET_ENABLED	Oscillation stop detection enabled.
SSP_ERR_OSC_STOP_DETECTED	The Oscillation stop detect status flag is set. Under this condition it is not possible to disable the Oscillation stop detection function.
SSP_ERR_OSC_STOP_CLOCK_ACTIVE	The Oscillation Detect Status flag cannot be cleared if the Main Osc or PLL is set as the system clock. Change the system clock before attempting to clear this bit.
SSP_ERR_INVALID_ARGUMENT	Invalid argument.
SSP_ERR_INVALID_MODE	Attempt to start a clock in a restricted operating power control mode.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.7.3 CGC HAL Module Operational Overview

The CGC HAL module interface provides the ability to configure and use all of the CGC HAL module's capabilities. Among those capabilities are the selection of several clock sources to use as the system clock source; additionally, the system clocks can be divided down to provide a wide range of frequencies for various system and peripheral needs.

Clock stability can be checked and clocks may also be stopped to save power when not needed. The API has a function to return the frequency of the system and system peripheral clocks at run time. There is also a feature to detect when the main oscillator has stopped, with the option of calling a user provided callback function.

The CGC HAL module to configure and control clock features:

- Configure any of the available clocks (HOCO, MOCO, LOCO, Main Clock, PLL, Sub-Oscillator) as the system clock source
- Configure the internal clocks (ICLK, PCLK, and so on)
- Switch the clocks on and off
- Configure the output clocks

- Set up the Oscillation Stop Detection feature

The Clock Generation Circuit peripheral features the following oscillators and clock generators:

- Main oscillator input of up to 24 MHz
- A 32.768 kHz sub-clock oscillator
- HOCO running at up to 64 MHz (depending on the device version)
- MOCO running at 8 MHz
- LOCO running at 32.768 kHz
- PLL circuit output running between 24 MHz and 240 MHz, depending on the device

The Synergy microcontrollers have six internal clock domains. Each of them has independent divisors but are dependent upon the clock input selected in the System Clock Control Register. These are:

- ICLK – The core clock, for CPU, DMAC, ROM and RAM (max 32/48/240 MHz)
- PCLKA – Peripheral clock for modules including EtherC, EDMAC, USB2.0 HS, QSPI and SCIF (max 32/48/120MHz)
- PCLKB – Peripheral clock for modules like IIC, CAN, DAC12, RTC, USBFS, I/O Ports, WDT and IWDT (max 32/60 MHz)
- PCLKC – Peripheral clock for ADC12 conversion clock (max 64/60 MHz)
- PCLKD – Peripheral clock for GPT count clock (max 64/120 MHz)
- FCLK –Clock source for the flash memory (max 32/60 MHz)

In addition, some of the Synergy microcontrollers also support controllable external clock outputs, some of which have independent divisors. These are:

- CLKOUT – CLOCKOUT/BUZZER clock (max 24 MHz) (independent clock selector and divisor)
- BCLK – External bus clock to external bus controller (max 16/120 MHz)
- SDCLK – SDRAM clock (max 120 MHz)
- UCLK – USB clock (max 120 MHz) (independent divisor/selector on Synergy version 2)
- LCD_CLK – LCD Clock (independent clock selector but no divisor)

Changing the System Clock Peripheral Clock Divisors at Runtime

The CGC HAL module also has the option to change the system clock and clock tree settings at runtime via the clocksCfg API function. Choose New Stack>System>CGC Configuration Instance to create a configuration structure for use with the clocksCfg API function.

The clocksCfg function allows changes to the system clock, the peripheral clock dividers, the PLL multiplier and divider, and the state of the system clocks (stop/start) (HOCO, Main Oscillator, Subclock oscillator, and so on). The options in the following figure show an example where the system clock is being changed from HOCO to MOCO, and the peripheral clock dividers are also being updated. The options for each clock are Start, Stop, and None (meaning no change.) Not all clocks are available on all MCUs. Not all peripheral clocks are available on all MCUs.

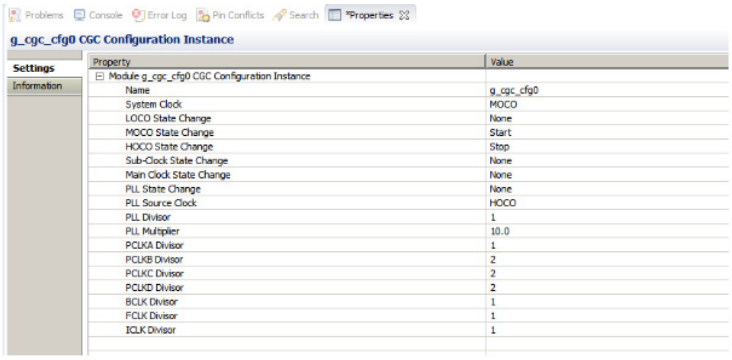


Figure 280: CGC Configuration Properties

The function call for the above example is:

```
g_cgc.p_api > clocksCfg(&g_cgc_cfg0);
```

Option Setting Memory

Synergy microcontrollers all include an Option-Setting Memory, this memory can be used to set the operating state of peripherals after a reset. The OFS can be used to set the state of the IWDT, WDT, LVD and CGC HOCO. The following table lists CGC HOCO parameters that can be configured by OFS registers.

OFS register setting possibilities

Control	Description
HOCO oscillation enable	Automatically starts the HOCO after a Reset, if enabled

Control	Description
HOCO Frequency	S7 and S5 Series - 16MHz - 18MHz - 20MHz S3 and S1 Series - 24 MHz - 32 MH - 48 MHz - 64 MHz

You can set the OFS register values through the properties dialog, the properties dialog is available on the Synergy Configuration editor when you select the BSP tab.

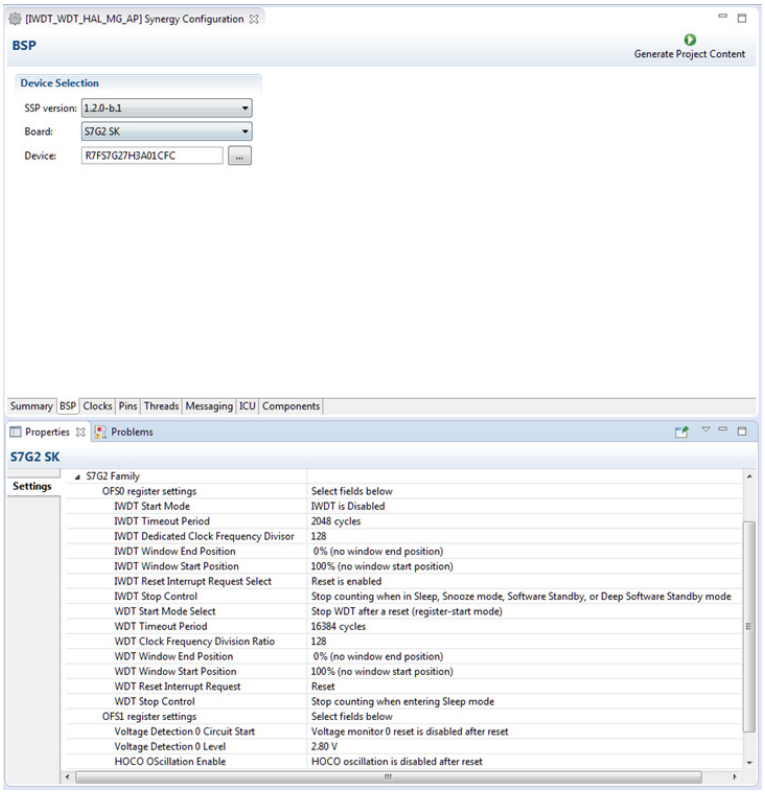


Figure 281: OFS Register Settings

The CGC HAL module also handles operating power control modes of the MCU since the Low Power Modes Version 2 HAL module will no longer handle operation-power control modes of the MCU.

CGC HAL Module Important Operational Notes and Limitations

- The CGC HAL module has no dependencies with respect to the ThreadX RTOS.
- The CGC HAL module is a core function of the MCU and is set after by the BSP initialization process. It is quite possible that the CGC can be left unchanged. However, the CGC HAL module provides functions that change the clock configuration that can balance the requirements of operating speed and power consumption, depending on application requirements.
- The CGC peripheral of the Synergy microcontrollers support Oscillator Stop Detection. If enabled in the application, the Oscillator Stop Detection function will automatically detect if the Main/PLL clock has stopped and switches operation to the MOCO. When enabling this functionality in the SSP a callback function has to be manually created by the user. The following steps detail this procedure.

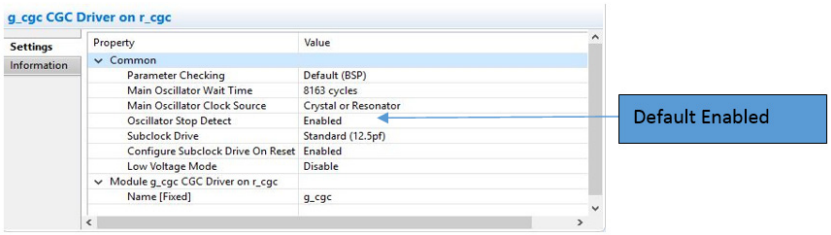


Figure 282: Oscillator Stop Detect Enable/Disable

NOTE: In SSP 1.4.0 and later the Oscillator Stop Detect property is set to default as Enabled. Previous versions of SSP set the default to Disabled.

- If the Configure Subclock Drive on Reset is set to Disabled, then the subclock will not be configured at startup. In order to configure the subclock user has to create a definition of weakly linked function R_BSP_WarmStart(). In this function if the argument is "BSP_WARM_START_POST_C", user can call the API R_CGC_ClockStart to start the subclock.

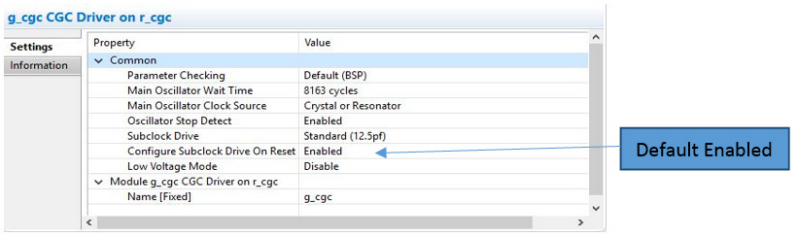


Figure 283: Configure Sub-clock drive on Reset

In the application code, create a callback function. In this example, it is called osc_stop_callback.

```
void osc_stop_callback(cgc_callback_args_t * p_args)

{

/* perform Oscillator Stop Detection processing */

}
```

Enable the oscillator stop detection by calling the API with the previously declared callback.

```
/* Enable the Osc Stop Detect functionality */

g_cgc.p_api-&gt;oscStopDetect( osc_stop_callback, true );
```

Enable the interrupt within the ICU

```
/* Osc Stop Detect is an NMI interrupt. Enable the NMI in ICU */

R_ICU->NMIER_b.OSTEN = 1;
```

Refer to the most recent SSP release notes for limitations on the use of this module.

4.2.7.4 Including the CGC HAL Module in an Application

This section describes how to include the CGC HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

The CGC Driver is automatically added to the HAL/Common thread, so it only needs to be added to a new thread if it has been removed. (The default name for the CGC is g_cgc. This name can be changed in the associated Properties window.)

CGC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_cgc CGC HAL on r_cgc	Threads	New Stack> Driver> System> CGC Driver on r_cgc

The CGC Driver on g_cgc is automatically added to the HAL/Common Stack, as shown in the following figure:

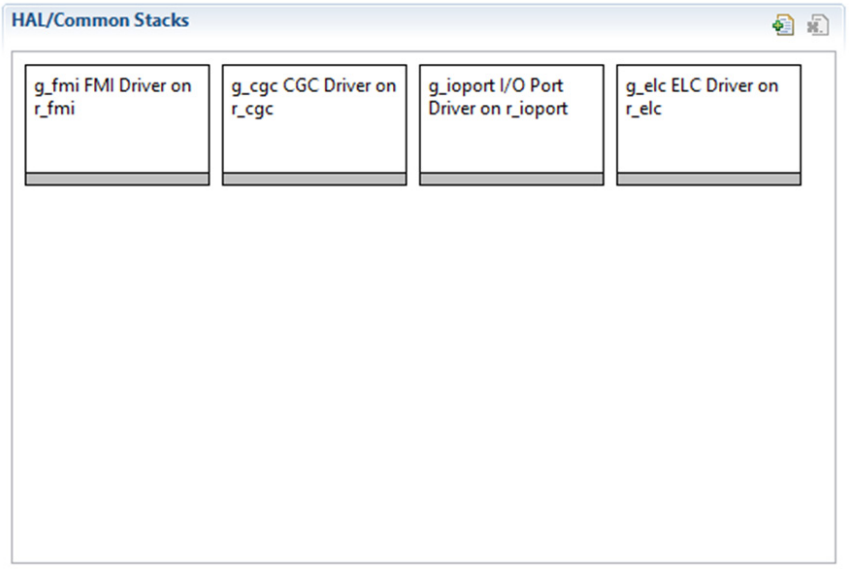


Figure 284: CGC HAL Module Stack

4.2.7.5 Configuring the CGC HAL Module

The CGC HAL module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the CGC HAL Module on r_cgc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enable or disable the parameter error checking.
Main Oscillator Wait Time	3,35,67,131,259,547,1059,2147,4291,8163 cycles (Default: 8163 cycles)	Set to one of these values. It should be at least as long as the main clock stabilization time. This delay will be configured only if #define CGC_CFG_MAIN_OSC_CLOCK_SOURCE is set to 0, indicating that a resonator/ crystal is used. Set the main clock oscillation stabilization time to longer than or equal to the stabilization time recommended by the oscillator manufacturer.
Main Oscillator Clock Source	External Oscillator, Crystal or Resonator (Default: Crystal or Resonator)	Set to 0 if a resonator, or crystal, is used. Set to 1 if an external oscillator input is used
Oscillator Stop Detect	Enabled, Disabled (Default: Enabled)	This allows the R_CGC_OscStopDetect function code to be generated if enabled. The user must call this function with a callback pointer to use this feature.
Subclock Drive	Standard (12.5pf), Middle (4.4pf) (Default: Standard)	This setting is for matching the subclock oscillator drive capacitance based on the crystal parameters #define CGC_CFG_SUBCLOCK_DRIVE.

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have

different default values and available configuration settings.

In some cases, settings other than the defaults for the CGC HAL module can be desirable. For example, it might be useful to selectively turn clocks on or off or change frequency to optimize power and performance characteristics.

NOTE: Most of the property settings for modules are fairly intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

CGC HAL Module Clock Configuration

The default CGC HAL module clock frequencies that will be set by the BSP initialization process are configurable in the ISDE by using the Clocks tab in the configurator. Invalid selections are indicated in red when selected.

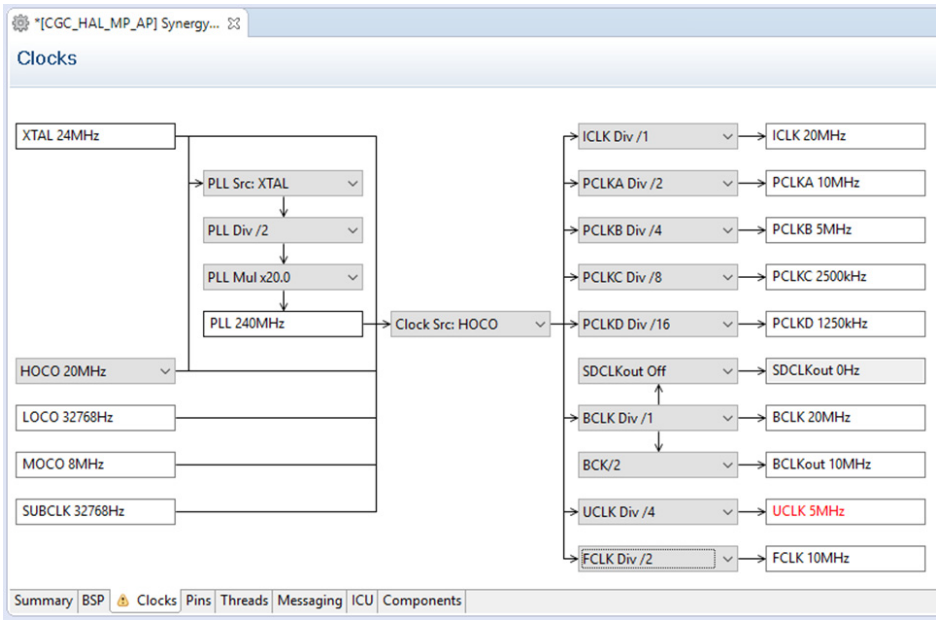


Figure 285: Default Clock Settings via the Clocks tab

In this example, the Clock Source is HOCO, and various clock dividers are chosen for the peripheral clocks. If a valid USB Clock (UCLK) cannot be achieved, it is highlighted in RED. It should be noted that this is only advisory, and the project will still build, as such a clock frequency may be required.

CGC HAL Module Pin Configuration

The CGC peripheral module controls the output of BCLK and SDCLK signals. Use the Clocks tab to enable/disable this functionality. The BCLK_SDCLK I/O pin must be selected and configured as required via the Pins tab.

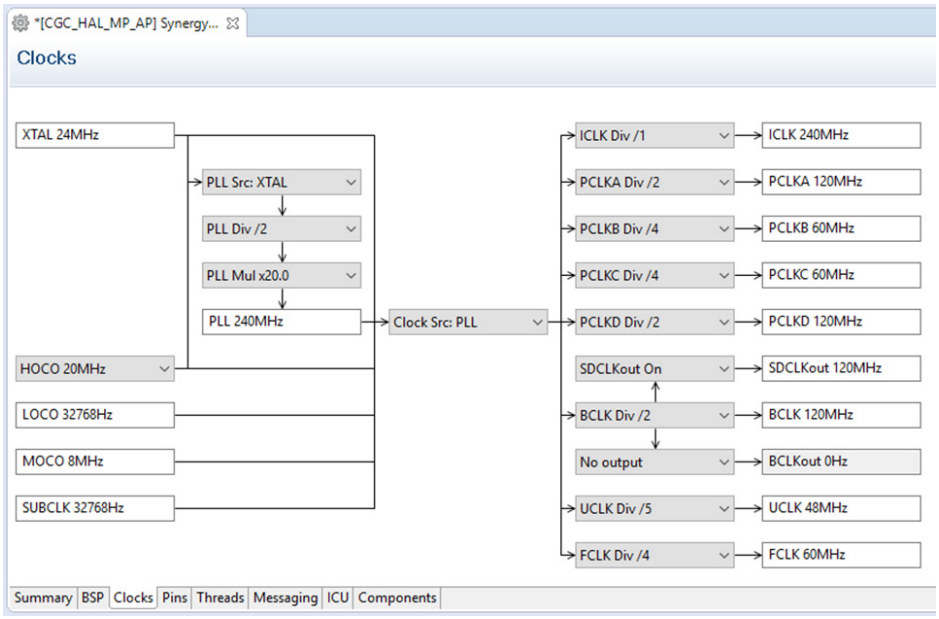


Figure 286: Enabling/Disabling SDCLK and BCLK via the Clock tab

In this example, SDRAM Clock is enabled, BUS Clock is disabled.

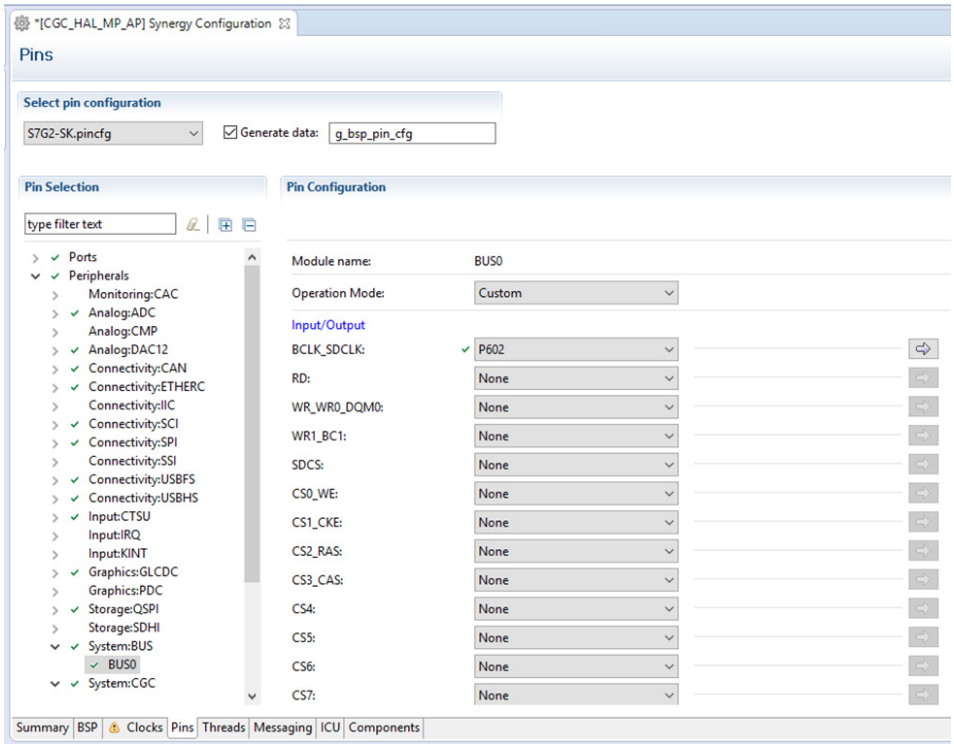


Figure 287: Enabling/Disabling SDCLK and BCLK via the Pins tab

In this example, SDRAM Clock/BUS Clock is enabled on P602.

Additional pin settings associated to the CGC allow for the enabling / disabling of the external oscillator pins and the setting of the system Clock Out pin.

The Synergy devices can run from its on chip oscillators, in which case there is no requirement for the main clock external oscillator pins XTAL and EXTAL. These could be used as input pins by the application. The functionality of the sub clock external oscillator pins XCIN and XCOUT is fixed.

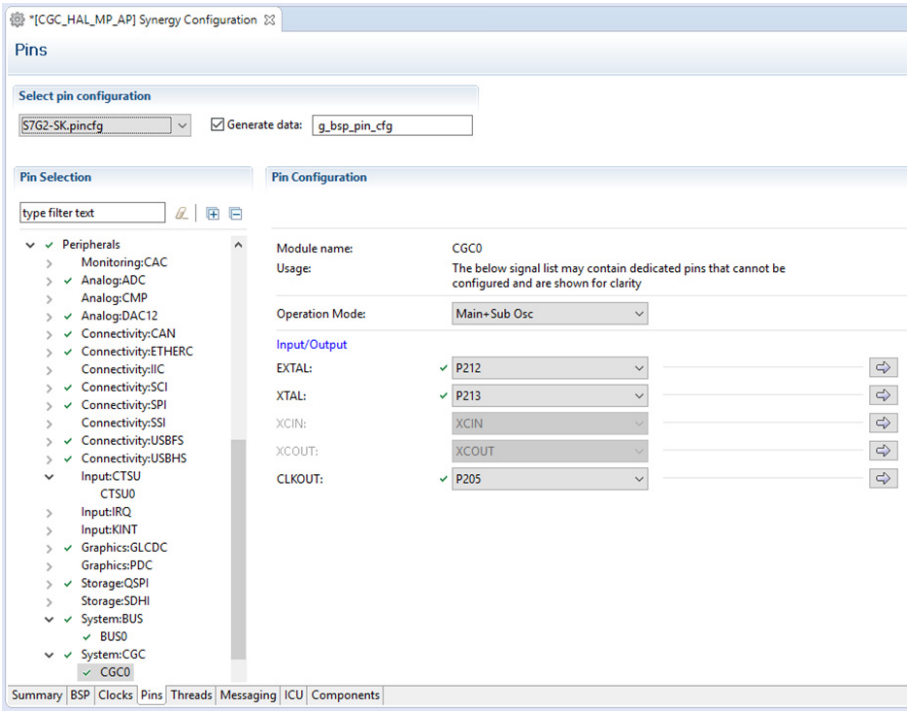


Figure 288: Enabling/Disabling EXTAL, XTAL and CLKOUT via the Pins tab

In this example, an external Main Oscillator is used via pins P212 and P213 and the CLKOUT is enabled on P205.

NOTE: The example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and Synergy MCUs may have different available pin configuration settings.

4.2.7.6 Using the CGC Module in an Application

The typical steps in using the CGC HAL module in an application are:

1) The CGC is automatically set after Reset.
2) Configure the clock as desired using the clocksCfg API.
3) Start clocks using the clockStart API.
4) Stop clocks using the clockStop API.

5) Other CGC functions as needed.

These common steps are illustrated in a typical operational flow diagram in the following figure:

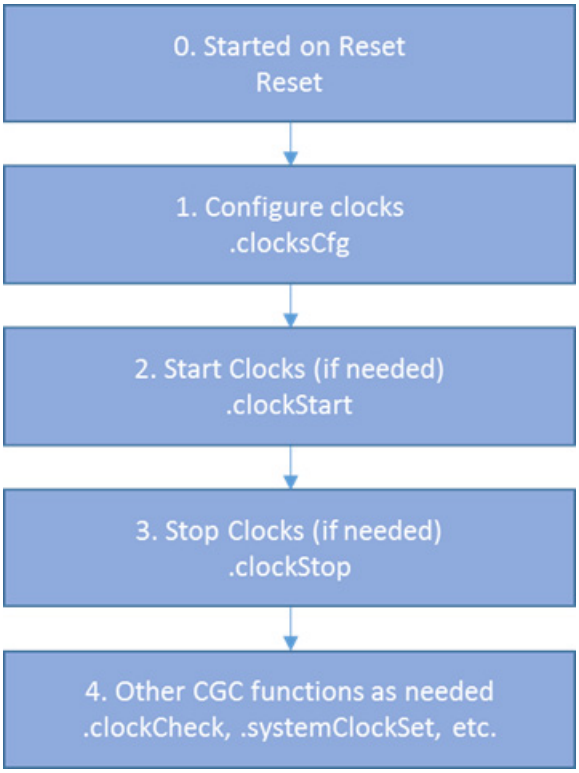


Figure 289: Flow Diagram of a Typical CGC HAL Module Application

4.2.8 CRC Driver

The CRC HAL module provides high-level APIs used to calculate 8, 16, and 32-bit CRC values on a block of data in memory or a stream of data over a Serial Communication Interface (SCI) channel using various types of industry-standard polynomials. The CRC HAL module is implemented on r_crc and uses the CRC peripheral on the Synergy MCU.

- CRC HAL module can calculate CRC on a block of data in memory.
- CRC HAL module can calculate CRC on a stream of data being transmitted or received over a serial communication Interface (SCI) channel (snoop mode).
- CRC HAL module supports the following 8 and 16 bit CRC polynomials which operates on 8-bit data in parallel
  - $X^8 + X^2 + X + 1$ (CRC-8)
  - $X^{16} + X^{15} + X^2 + 1$ (CRC-16)
  - $X^{16} + X^{12} + X^5 + 1$ (CRC-CCITT)
- CRC HAL module supports the following 32 bit CRC polynomials which operates on 32-bit data in parallel
  - $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$ (CRC-32)

– $X^{32} + X^{28} + X^{27} + X^{26} + X^{25} + X^{23} + X^{22} + X^{20} + X^{19} + X^{18} + X^{14} + X^{13} + X^{11} + X^{10} + X^{9} + X^{8} + X^{6} + 1$ (CRC-32C)

- CRC HAL module can calculate CRC with LSB first or MSB first bit order.

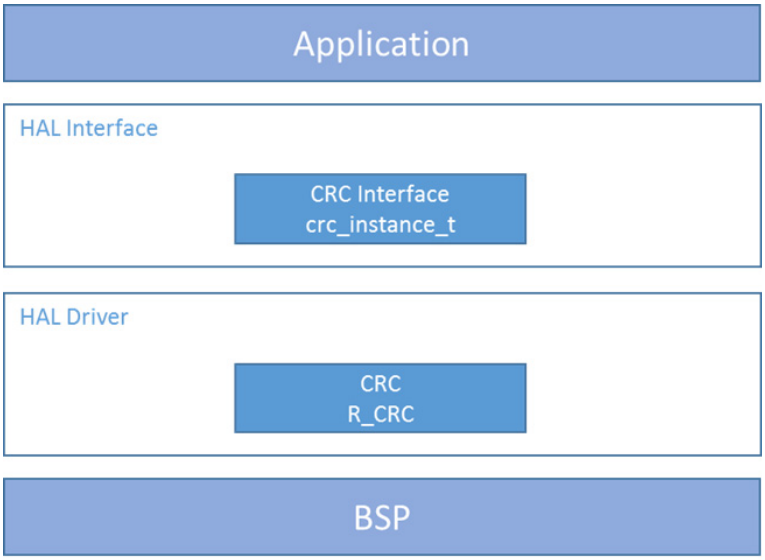


Figure 290: CRC HAL Module Block Diagram

4.2.8.1 CRC APIs Overview

The CRC HAL module defines APIs for **opening**, **closing**, **enabling**, and **calculating**. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows.

CRC HAL Module API Summary

Function Name	Example API Call and Description
open	g_crc.p_api->open(g_crc.p_ctrl, g_crc.p_cfg); Open the CRC driver module.
close	g_crc.p_api->close(g_crc.p_ctrl); Close the CRC module driver.
crcResultGet	g_crc.p_api->crcResultGet(g_crc.p_ctrl, &result); Return the current calculated value.

Function Name	Example API Call and Description
snoopEnable	g_crc.p_api->snoopEnable(g_crc.p_ctrl, seed); Enable snooping.
snoopDisable	g_crc.p_api->snoopDisable(g_crc.p_ctrl); Disable snooping.
snoopCfg	g_crc.p_api->snoopCfg(g_crc.p_ctrl, g_crc.p_cfg); Configure the snoop channel and direction.
calculate	g_crc.p_api->calculate(g_crc.p_ctrl, &input_buffer, num_bytes, crc_seed, &crc_result); Perform a CRC calculation on a block of data.
versionGet	g_crc.p_api->versionGet(&version); Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the *SSP User's Manual* API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Configuration was successful.
SSP_ERR_ASSERTION	Assertion error.
SSP_ERR_INVALID_ARGUMENT	Invalid argument error.
SSP_ERR_NOT_OPEN	The driver is not opened.
SSP_ERR_IN_USE	If driver is already open.

NOTE: Lower-level drivers may return common error codes. Refer to the *SSP User's Manual* API References for the

associated module for a definition of all relevant status return values.

4.2.8.2 CRC HAL Module Operational Overview

When the CRC HAL module is used to calculate the CRC value for a block of data in memory, the calculate API can be used which takes the input buffer pointer, length and the CRC seed value as input and outputs the calculated CRC value.

When CRC HAL module is used to calculate CRC on a stream of data being transmitted or received over a serial communication Interface (SCI) channel (snoop mode), then first the module should be configured to be in snoop mode by calling snoopCfg followed by snoopEnable APIs. After the requested number of data is transmitted or received on the SCI channel, the calculated CRC value can be polled from the module using crcResultGet API.

CRC HAL Module Important Operational Notes and Limitations

- The CRC block does not use any interrupts.
- There is no clock configuration for the CRC module.
- There are no callbacks for the CRC module.
- When using 32-bit CRC polynomials for calculating CRC values of data block in memory, the data block is interpreted using little-endian byte order.
- Refer to the most recent *SSP Release Notes* for any additional operational limitations for this module.

4.2.8.3 Including the CRC HAL Module in an Application

This section describes how to include the CRC HAL module in an application using the *SSP configurator*.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the *SSP User's Manual* to learn how to manage each of these important steps in creating SSP-based applications.

To add the CRC HAL module to an application, simply add it to a thread using the stacks selection sequence provided in the following table. (The default name for the CRC Driver is g_crc0. This name can be changed in the associated **Properties** window.)

CRC Driver Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
r_crc0 CRC Driver on r_crc	**Threads**	**New Stack> Driver> Monitoring> CRC Driver on r_crc**

When the CRC Driver on r_crc is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules.

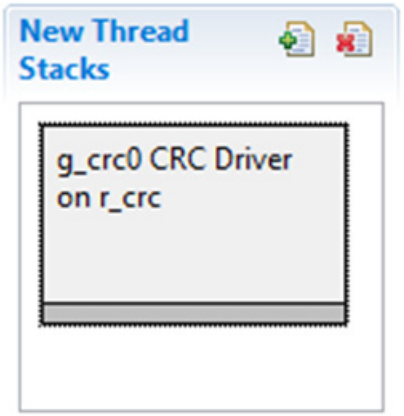


Figure 291: CRC HAL Module Stack

4.2.8.4 Configuring the CRC HAL Module

The CRC HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are locked and are not available for changes, and are identified with a lock icon for the locked property in the **Properties** window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the **Properties** tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority. This configuration setting is available within the **Properties** window of the associated module. Simply select the indicated module and then view the **Properties** window. The interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Note that the interrupt priorities listed in the **Properties** window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+.) This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the Property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the CRC HAL Module on r_crc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g-crc0	Module name.

ISDE Property	Value	Description
CRC Polynomial	CRC-8, CRC-16, CRC-CCITT, CRC-32, CRC-32C Default: CRC-32C	Specify the polynomial to use for calculation.
Bit Order	LSB, MSB Default: MSB	Specify the bit order of the calculation.

NOTE: The example settings and defaults are for a project using the S7G2 Synergy MSU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for a module can be desirable. For example, it might be useful to select a different clock source than the default. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

CRC HAL Module Clock Configuration

The CRC HAL module is clocked via the Peripheral Clock A (PCLKA.)

The CRC HAL module does not support any APIs for setting the frequency at which it operates.

CRC HAL Module Pin Configuration

The CRC HAL module does not have any configurable pins.

4.2.8.5 Using the CRC HAL Module in an Application

The typical steps in using the CRC HAL module for computing CRC of data block in memory are:

1) Initialize the CRC HAL module using the open API
2) Compute the CRC HAL module using the calculate API
3) Close the CRC HAL module using the close API

These common steps are illustrated in a typical operational flow diagram in the following figure:

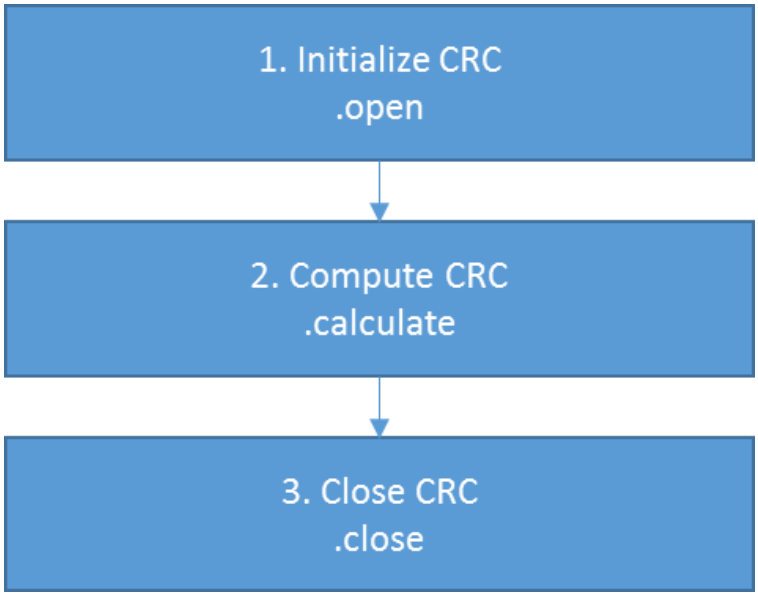


Figure 292: Flow Diagram of a Typical CRC HAL Module Application

The typical steps in using the CRC HAL module for computing CRC in snoop mode are:

1) Initialize the CRC HAL module using the open API.
2) Configure CRC module to snoop an SCI channel (and its direction) using snoopCfg API.
3) Enable snooping of the SCI channel using snoopEnable API.
4) Once the required number of bytes are transmitted or received on the SCI channel, get the calculated CRC value using crcResultGet API.
5) Disable the snooping operation using snoopDisable API.
6) Close the CRC HAL module using the close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

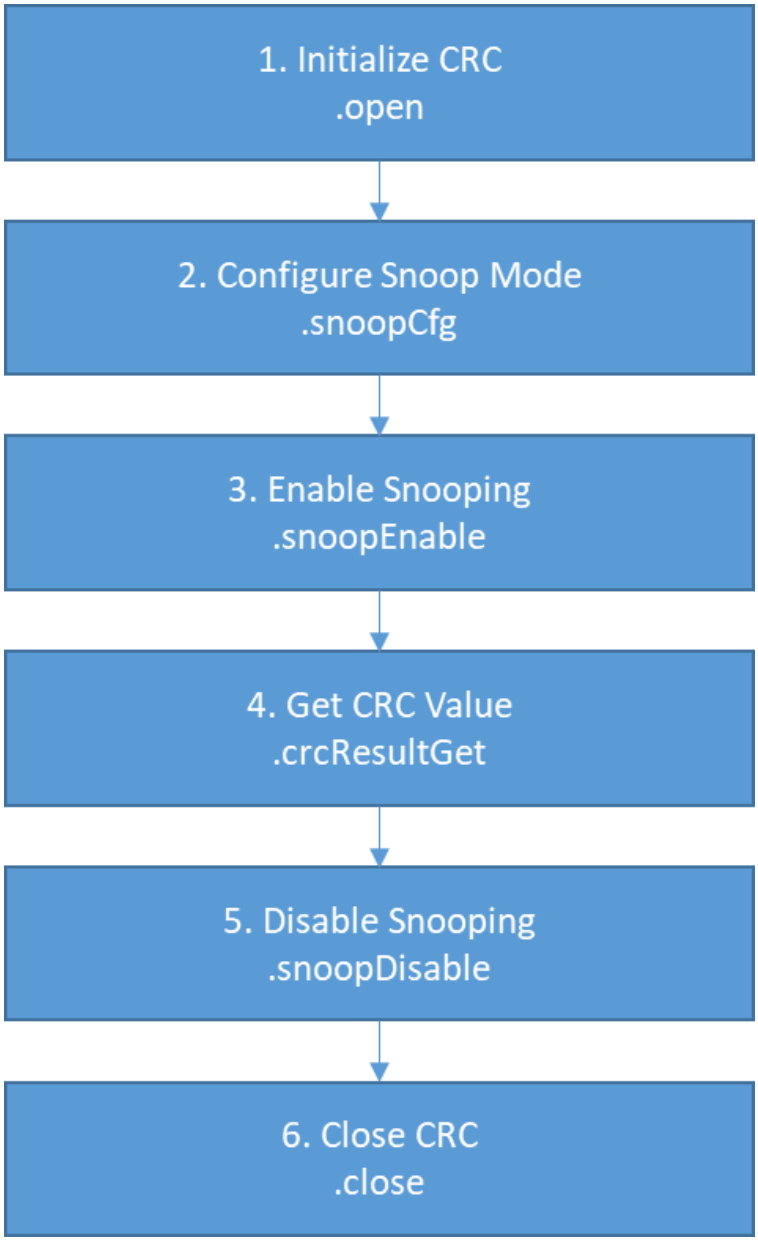


Figure 293: Flow Diagram of a Typical CRC HAL Module Snoop Mode Application

4.2.8.6 The CRC HAL Module Application Project

The application project associated with this module guide demonstrates the steps in a full design. The project can be found using the link provided in the References section at the end of this document. You may want to import and open the application project within the ISDE and view the configuration settings for the CRC HAL module. You can also read over the code (in crc_hal.c) which is used to illustrate the CRC HAL module APIs in a complete design.

The application project demonstrates the typical use of the CRC HAL module APIs. The application project main-thread entry initializes the CRC HAL module and periodically computes the CRC. The CRC is initially computed for a data buffer that is filled with generated bytes, then the data is copied to the so-called receive buffer, (the application simulates data transmission) and once again the CRC is computed. Both CRCs are compared, and based on the result, the **green** (CRCs are equal) or red (CRCs are different) LED is lit. The simulation of data transmission can introduce errors which result in lighting the red LED. The following table identifies the target versions for the associated software and hardware used by the application project.

Software and Hardware Resources Used by the Application Project

Resource	Revision	Description
e^2 studio	5.3.1	Integrated Solution Development Environment (ISDE)
SSP	1.2.0	Synergy Software Platform
IAR EW for Synergy	7.71.1	IAR Embedded Workbench® for Renesas Synergy™
SSC	5.2.1.016	Synergy Standalone Configurator
SK-S7G2	v3.0 to v3.1	Starter Kit

A simple flow diagram of the application project is given in the following figure:

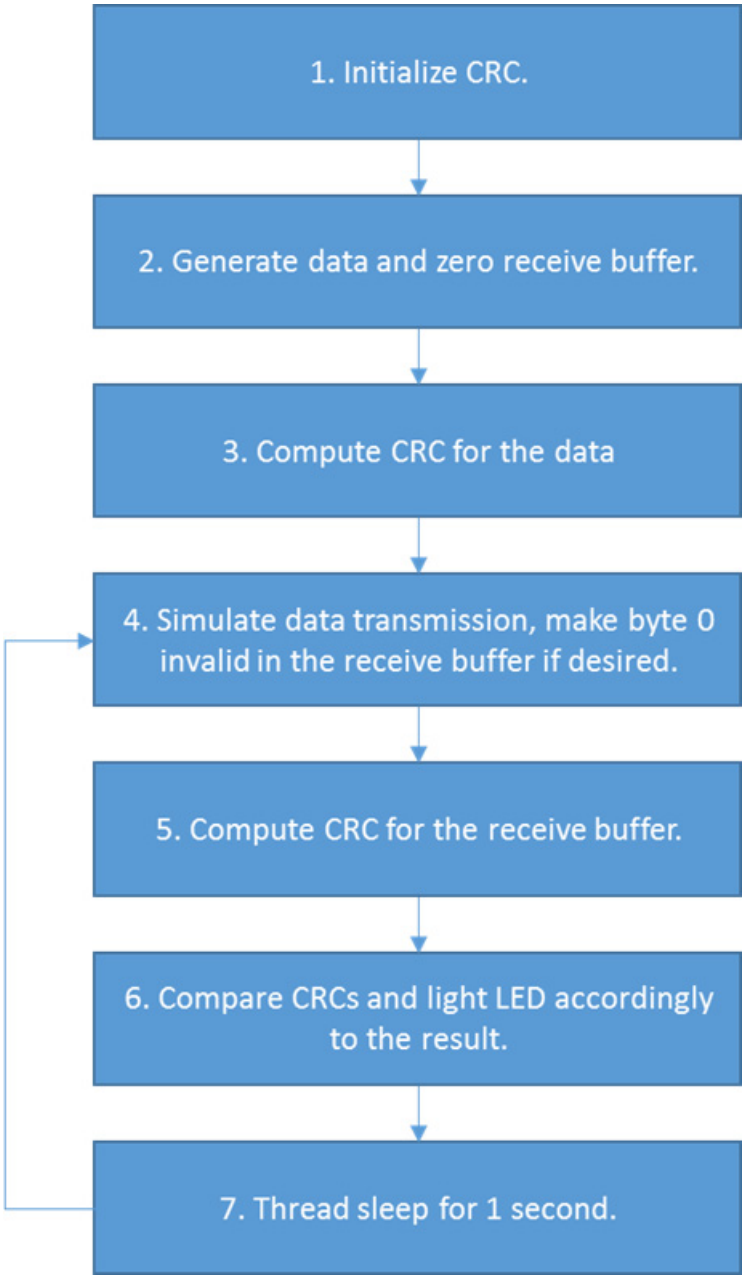


Figure 294: CRC Application Project Flow Diagram

The complete application project can be found using the link provided in the References section at the end of this document. The crc_hal.c file is located in the project once it has been imported into the ISDE. You can open this file, within the ISDE and follow along with the following description to help identify key uses of APIs.

In the application project, main-thread entry, the CRC HAL module is initialized using the open API right after the data array and receive buffer are set up. The data array contains growing numbers, as bytes, starting with 0, and the receive buffer is initialized with zeros. When the data is ready and the CRC HAL module is initialized, a CRC is computed for

the data. After that is code, the application flow enters a loop which simulates data transmission and fills the receive buffer, computes the CRC for the buffer, and lights the green (CRCs are equal) or red (CRCs are different) LED, according to the CRC comparison result. The application is initially configured not to introduce data-transmission errors. This can be changed by setting the makeError variable to true. Then, after copying bytes from the data array to the receive buffer, the byte at index 0 in the buffer is altered. This results in a different CRC, and lighting the red LED afterwards.

The CRC application project uses the CRC HAL module default property-values. No extra activities must be performed by the user.

4.2.9 CTSU Driver

This section describes how to configure and program the CTSU Driver using the e^2 studio ISDE.

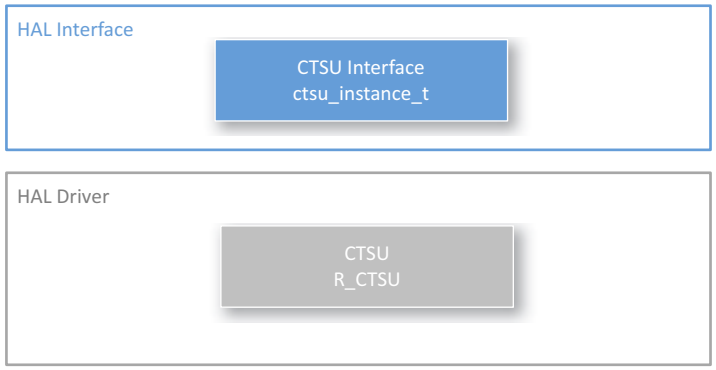


Figure 295: CTSU Driver – Block Diagram

The CTSU Driver is a generic driver implemented on r_ctsu for capacitive touch sensing applications and supports the CTSU peripheral on the Synergy microcontroller hardware. It is designed to be used in conjunction with the **Workbench 6** tool which generates the requisite structures that will be used by this module for initialization and operation.

In the Project Configurator in the e^2 studio ISDE, you can add and configure a CTSU Driver Module in the Modules pane of the Threads tab by selecting **New > Driver > Input > CTSU Driver on r_ctsu**. For details, see: Using e^2 studio to Write an Application with the CTSU Driver

For tuning the capacitive touch sensor, use the **Workbench 6** tool which can be downloaded separately from http://renesassynergy.com

You can find the API reference in the description of the CTSU Interface here: CTSU Interface

4.2.9.1 What Does the CTSU Driver Do?

The CTSU Driver is used to initialize the CTSU peripheral to detect a change in capacitance on any of the configured (and enabled) channels, perform requisite filtering and generate a variety of data that can be used by higher level widget layers like buttons wheel and sliders. To support the different types of data required by these layers, the implementation provides a Read() function that allows upper level layers to read different types of processed data based on their need. The driver also provides a callback when each scan is complete and when new processing data is available. These callbacks can be used by upper layers to read the data.

The CTSU Driver allows the user to configure the CTSU channels for all the supported operation modes including Mutual and Self Capacitance. The driver will scan the configured channels, move the data using the DTC, perform filtering, drift

compensation, auto-tuning and notify the user via a callback once each iteration is completed. The driver also allows the user to provide their own filtering and auto-tuning algorithms and integrate it into the process. The driver can only support one configuration at a time, but the user can reopen the driver with multiple channel configurations as required by the application.

4.2.9.2 Using e² studio to Write an Application with the CTSU Driver

The driver is integrated into the Synergy Software Package in e² studio: e² studio ISDE User Guide.

In e² studio, create and configure a project and add the drivers:

1) Create the project: Creating a Project.
2) Configure the project: Configuring a Project
3) Add the drivers: Adding Threads and Drivers

The following resources are required for any application using the CTSU Driver:

Resource	ISDE Tab	Selection
CTSU Driver	Threads	**Driver > Input > CTSU Driver on r_ctsu**
Interrupts	Threads	**CTSU WRITE, CTSU READ, CTSU END**
CTSU Clock	Clocks	See Configuring the CTSU Clocks
CTSU Pins	Pins	See Configuring the CTSU Pins

Configuring the CTSU Clocks

Set the CTSU clock speed to match the clock speed selected in the capacitive touch tuning tool **Workbench 6**.

Configuring the CTSU Pins

Set the pins as touch sensor pins TSxx and enable the TSCAP function pin.

Configuring the CTSU Interrupt

Enable the three CTSU interrupts in the ICU tab by setting them all, to the same priority. Any value will work.

Configuring the CTSU Parameters

Use the e² studio ISDE to configure the CTSU Driver Parameters.

For applications that do not use an RTOS: Adding and Configuring HAL Drivers

For ThreadX applications: Adding Drivers to a Thread and Configuring the Drivers.

Ensure that a **file/folder** with the **CTSU Configuration** generated by Workbench exists in the ${PROJECT_ROOT}\src folder and is included in the build

To use the SSP to configure the CTSU parameters, do the following:

1) See the file r_ctsu_api.h for details of the structure type ctsu_cfg_t.
2) Create a local variable of the type ctsu_cfg_t with the configuration set as below.

CTSU Common/Build-time configuration parameters

ISDE Property	Configuration	Setting	Description
Parameter Checking	-	Enabled (Default), Disabled	Enable or disable the parameter error checking.
Offset Adjustment	-	Enabled (Default), Disabled	Used to enable or disable offset adjustment. Leave as enabled unless you are tuning the board. (Rate is determined by *Runtime rate of tuning of sensor values*)
Drift Compensation	-	Enabled (Default), Disabled	Used to enable or disable drift compensation. Leave as enabled unless you are tuning the board. Drift compensation allows the baseline values used to determine the presence or absence of a touch to change over time as the board or surface ages or with changes in temperature and environment. Drift compensation rate is determined by *Steady state drift compensation rate*, *Startup drift compensation rate*, and *Channel release compensation rate*.

ISDE Property	Configuration	Setting	Description
Drift Compensation Methods	-	Alternate 1	Drift Compensation algorithms provided. Only Alternate 1 is currently supported. Other options may be supported in future releases. Drift compensation method Alternate 1 allows for different rates of drift compensation for the following events: steady state, startup, and channel release.
Dynamic Memory Usage	-	Enabled, Disabled (Default)	Used to save memory in applications where channels are dynamically enabled and disabled. Dynamic Memory Usage is currently not supported.
Perform auto-tune and drift compensation only when all channels are untouched	-	Enabled (Default), Disabled	Perform auto-tune and drift compensation only when all channels are untouched.
DTC Usage	-	Enabled (Default), Disabled	Allows the DTC to be used to move data in and out of the CTSU minimizing CPU overhead. This option cannot be currently disabled.
Maximum Active Channels	-	-	Specify the maximum number of channels that are to be used by the application. In Self Capacitance Mode, this is the number of channels used. In Mutual Capacitance Mode, this is the multiplication result of the Rx and Tx channels. For example: 4 Rx and 3 Tx channels = 12 Maximum Active Channels.

ISDE Property	Configuration	Setting	Description
Steady state drift compensation rate	-	Defaults to 500	Determines the rate of drift compensation (applied every n scans) when the channel is in steady state. (dependent on the scan rate)
Startup drift compensation rate	-	Defaults to 5	Determines the rate of drift compensation (applied every n scans) when the driver is performing its initialization. (Should be less than the *Steady state drift compensation rate*)
Channel release compensation rate	-	Defaults to 500	Determines the rate of drift compensation (applied every n scans) when a channel transitions from pressed to released. (Should be less than or equal to the *Steady state drift compensation rate*)
Default filter depth	-	Defaults to 14	Used in the software sensor count filter provided by driver. This default filter is a modification of the relatively common "leaky integrator" software filter. A depth of 4 equates to an approximate filter constant of 16. When the input is a constant value of 16 or greater, the filter output will reach 68-69% of the constant input value after 16 cycles/iterations of the filter.
Runtime rate of tuning of sensor values	-	Defaults to 800	Rate of offset adjustment.(If drift compensation is used this value is set to twice the rate of the steady state drift compensation)

ISDE Property	Configuration	Setting	Description
DTC Usage for CTSU	-	Defaults to *true*	DTC is required for use of r_ctsu in this release. Future releases may support r-ctsu data transfer without DTC.

CTSU Module configuration parameters

ISDE Property	Configuration	Setting	Property. Description
Transfer Instance for DTC Read	p_lower_lvl_transfer_read	-	Name of the Transfer Instance that will be used by the CTSU driver to read out data
Transfer Instance for DTC Write	p_lower_lvl_transfer_write	-	Name of the Transfer Instance that will be used by the CTSU driver to write configuration data
CTSU Hardware Configuration	p_ctsu_hw_cfg	-	Name of the configuration structure generated by Workbench
Processing Functions	p_ctsu_functions	-	Any processing functions the user would like to use to replace the default internal processing functions. Can be set to NULL if not used.
Callback	p_callback	-	The user callback function that will be invoked each time the scan is complete as well as when newly processed data is available. Can be set to NULL if not used.
Processing Option	ctsu_soft_option	-	Can be used to specify if scans should start after the data from the previous one is completed, if auto-calibration should be run between scans etc. Use the Default Setting for most cases.

ISDE Property	Configuration	Setting	Property. Description
Closing Option	ctsu_close_option	-	Can be used to specify if register states should be reset or maintained (higher power/memory consumption but quicker re-enable time) on closing.

4.2.9.3 Writing a CTSU Application

1) Generate a Synergy project.
2) Open the Synergy Configuration.
3) Go to the CTSU section in the Pins tab and enable the GPIO pins to be used as touch sensor pins TSxx.
4) Ensure that the TSCAP pin is enabled as well.
5) Ensure that the Clock speeds for the MCU match those selected in the external CapTouch Tuning Tool Workbench 6.
6) Add the DTC Transfer module from **Driver > Transfer > Transfer Driver on r_dtc**. No configuration is required.
7) Add the CTSU HAL driver to the HAL modules or to a thread from **Driver > Input > CTSU driver on r_ctsu**.
8) Ensure that a file/folder with the CTSU Configuration generated by Workbench exists in the ${PROJECT_-ROOT}\src folder and is included in the build.
9) Enable the three CTSU interrupts in the ICU tab by setting them all, to the same priority. Any value will work.
10) Using the Properties tab for the CTSU driver on r_ctsu, adjust the Maximum number of channels (or channel combinations if in Mutual Mode) used by any configuration.
11) Set the callback field to a user defined function.
12) Set the DTC Transfer name field to the name of the DTC module
13) Set the CTSU Configuration Used field to the name of the configuration generated by WorkBench.
14) Click Generate Project Content to generate the requisite structures.
15) Open hal_data.h (or thread_name.c) to see the structures that are generated to use this driver. For an overview of how to use SSP modules, refer to the SSP user manual.
16) Call Name.api->open ensuring properly initialized arguments are provided through a ctsu_cfg_t structure instance. This must be done before entering the main while loop.
17) From the main while loop keep calling Name.api->scan periodically to start a new scan. It is better to use a timer to run this task periodically which will determine the scan rate.
18) At the end of each scan, the user callback will be invoked. In the callback, check the event for the callback and call the Name.api->update to run processing on the data.
19) When the processing is complete the user callback will again be invoked with a different argument indicating that the data has been processed.

4.2.9.4 CTSU Driver Limitations

The driver only supports Mutual Capacitance Mode and Self Capacitance multi-scan mode. The driver does not support Dynamic memory allocation mode that would allow efficient memory usage in applications where the number of channels used changes dynamically. Also see the SSP release notes.

4.2.9.5 CTSU Driver Files

During the project configuration, the ISDE extracts the files shown in the following table to the /ssp directory.

Module	Directory
CTSU HAL Interface	synergy/ssp/inc/driver/spi/r_ctsu_api.h
CTSU HAL Instance	synergy/ssp/inc/driver/instances/r_ctsu.h
CTSU HAL Driver	synergy/ssp/src/driver/r_ctsu

4.2.9.6 Supported Devices for the CTSU Driver

The driver is designed to support the following families with no changes to the API:

- S7G2
- S3A7
- S128
- S124

The CTSU Framework is designed to support any Synergy device with a CTSU peripheral with no changes to the API.

4.2.10 SCE Crypto Driver

The Secure Cryptographic Engine (SCE) HAL module provides high-level APIs for random number generation, digest computation (hash), data encryption and decryption, digital signing and verification, key generation (using RSA, AES and ECC algorithms) and key installation (for RSA, AES and ECC keys). The SCE is a dedicated hardware block and the functionality provided by the SCE varies across the supported MCUs .

4.2.10.1 SCE HAL Module Features

The SCE HAL module configures the cryptographic module, which allows user to build cryptographic protocols for security with the following cryptographic primitives:

- Random-number generation
- Data encryption and decryption using AES or Triple DES (3DES) or ARC4 algorithms
- Signature generation and verification using the ECC, RSA or DSA algorithms
- Message-digest computation using HASH algorithms MD5, SHA1, SHA224, or SHA256

- Key generation - AES wrapped keys, RSA plain text and wrapped keys, ECC plain text and wrapped keys
- Installing the encrypted user key on to the Synergy platform.

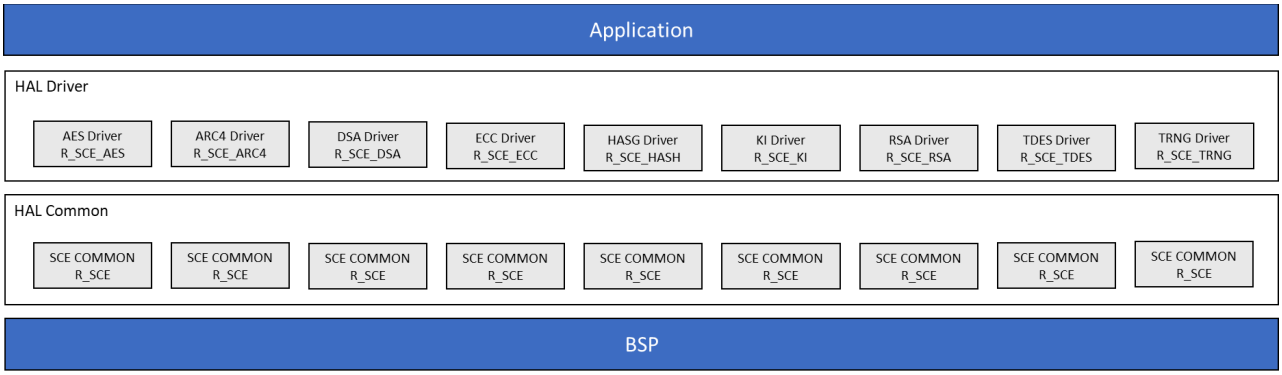


Figure 296: SCE HAL Module Block Diagram

NOTE: The above diagram includes all 9 available crypto modules. The SCE COMMON module is repeated for each one since it is included when the module is added to a thread stack. Common modules can be referenced by multiple other module instances across multiple Synergy stacks.

4.2.10.2 SCE APIs Overview

The SCE interface provides a common API for SCE HAL modules. The SCE interface supports multiple operations depending on the chosen module (AES, ARC4, RSA, DSA, HASH, TDES or TRNG.)

The AES interface defines APIs for opening, closing, generating wrapped keys, encrypting, and decrypting data using the AES algorithm. It uses a 128-bit, 192-bit, or 256-bit key and ECB, CBC, CTR, GCM, or XTS chaining-mode options. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. For return status values refer to the SCE API reference section of the SSP User's Manual.

SCE COMMON API Summary

Function Name	Example API Call and Description
.open	g_sce.p_api->open(g_sce.p_ctrl, g_sce.p_cfg); SCE Common module open function. Must be called before performing any other crypto operations.
.close	g_sce.p_api->close(g_sce.p_ctrl, num_words, p_key); Close the SCE Common module.

Function Name	Example API Call and Description
.interfaceGet	g_sce.p_api->interfaceGet(g_sce_aes.p_ctrl, p_interface_info, p_interface); Get the interface structure for the interface info provided.
.statusGet	g_sce.p_api->statusGet (g_sce.p_ctrl, p_status); Get status of SCE initialization.
.versionGet	g_sce.p_api->versionGet(p_version); Gets the module code and API version and stores it in provided pointer p_version

AES HAL Module API Summary

Function Name	Example API Call and Description
.open	g_sce_aes.p_api->open(g_sce_aes.p_ctrl, g_sce_aes.p_cfg); AES module open function. Must be called before performing any encrypt/decrypt operations.
.createKey	g_sce_aes.p_api->close(g_sce_aes.p_ctrl, num_words, p_key); Generate an AES key for encrypt / decrypt operations.
.encrypt	g_sce_aes.p_api->encrypt(g_sce_aes.p_ctrl, p_key, p_vi, num_words, p_source, p_dest); AES encryption using the chaining mode and padding mode specified in the open() function call.
.addAdditionalAuthenticationData	g_sce_aes.p_api->addAdditionalAuthenticationData (g_sce_aes.p_ctrl, p_key, p_vi, num_words, p_source); Add additional authentication data (called before starting an encryption or decryption operation).

Function Name	Example API Call and Description
.encryptFinal	g_sce_aes.p_api->encryptFinal(g_sce_aes.p_ctrl, p_key, p_iv, input_num_words, p_source, output_num_words, p_dest); AES final encryption using the chaining mode and padding mode specified in the open() function call.
.decrypt	g_sce_aes.p_api->decrypt(g_sce_aes.p_ctrl, p_key, p_iv, num_words, p_source, p_dest); AES decryption.
.setGcmTag	g_sce_aes.p_api-> setGcmTag(g_sce_aes.p_ctrl,num_words, p_source); Set authentication tag data.
.getGcmTag	g_sce_aes.p_api-> getGcmTag(g_sce_aes.p_ctrl,num_words, p_dest); Get authentication tag data.

Function Name	Example API Call and Description
.zeroPaddingEncrypt	g_sce_aes.p_api->zeroPaddingEncrypt(g_sce_aes.p_ctrl, p_key, p_iv, num_bytes, p_source, p_dest) AES zero padding encryption using the chaining mode and padding mode specified. Implementation for GCM mode only API usage - 1. To provide any Add Authentication Data (AAD): set p_dest = NULL 2. Encryption: set p_source to input data and p_dest will return encrypted data 3. Get/Compute Tag: set p_source = NULL

Function Name	Example API Call and Description
.zeroPaddingDecrypt	g_sce_aes.p_api->zeroPaddingDecrypt(g_sce_aes.p_ctrl, p_key, p_iv, num_words, p_source, p_dest); AES zero padding decryption using the chaining mode and padding mode specified. Implementation for GCM mode only API usage - 1. Set expected tag value using the setGcmTag() function 2. To provide any Add Authentication Data (AAD), invoke this API using p_dest = NULL 3. Decryption: set p_source to input encrypted data, decrypted data will be returned in p_dest 4. To verify the tag, invoke this API using p_source = NULL and p_dest = NULL, the return value indicates authentication tag verification status.
.versionGet	g_sce_aes.p_api->versionGet(p_version); Gets the module code and API version and stores it in provided pointer p_version
.close	g_sce_aes.p_api->close(g_sce_aes.p_ctrl); Close the AES module.

The ARC4 interface defines APIs for opening, closing, setting a key, and processing data. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table.

ARC4 HAL Module API Summary

Function Name	Example API Call and Description
.open	g_sce_arc4.p_api->open(g_sce_arc4.p_ctrl, g_sce_trng.p_cfg); Open the ARC4 module.
.keySet	g_sce_arc4.p_api->keySet(g_sce_arc4.p_ctrl, &rngbuf, nbytes); Set the key to be used by the ARC4 module.
.arc4Process	g_sce_arc4.p_api-> arc4Process(g_sce_arc4.p_ctrl, nbytes, &source, &destination); Encrypt or decrypt data using the ARC4 module.
.close	g_sce_arc4.p_api->close(g_sce_arc4.p_ctrl); Close the ARC4 module.
.versionGet	g_sce_arc4.p_api->versionGet (&version); Retrieve the version using the version pointer.

The DSA interface defines APIs for opening, closing, digital-signing, and verification. Available options include a 1024-bit public key and a 160-bit private key, a 2048-bit public key and a 224-bit private key, or a 2048-bit public key and a 256-bit private key. A complete list of the available APIs, an example API call and a short description of each can be found in the following table.

DSA HAL Module API Summary

Function Name	Example API Call and Description
.open	g_sce_dsa.p_api->open(g_sce_dsa.p_ctrl, g_sce_dsa.p_cfg); DSA module open function. Must be called before performing any sign/verify operations.

Function Name	Example API Call and Description
.verify	g_sce_dsa.p_api->verify(p_key, p_domain, num_words, p_signature, p_paddedHash); DSA signature verification using given DSA public key. This function is deprecated. The function hashVerify should be used instead.
.hashVerify	g_sce_dsa.p_api->hashVerify(g_sce_dsa.p_ctrl, p_key, p_domain, num_words, p_signature, p_paddedHash); DSA signature verification using given DSA public key.
.sign	g_sce_dsa.p_api->sign(p_key, p_domain, num_words, p_padded_hash, p_dest); DSA Signature generation using DSA private key. This function is deprecated. The function hashSign should be used instead.
.hashSign	g_sce_dsa.p_api->hashSign(g_sce_rsa.p_ctrl, p_key, p_domain, num_words, p_padded_hash, p_dest); DSA Signature generation using DSA private key.
.close	g_sce_dsa.p_api->close(g_sce_dsa.p_ctrl); Close the DSA module.
.versionGet	g_sce_dsa.p_api->versionGet(p_version); Gets version and stores it in provided pointer p_version.

The HASH interface defines APIs for calculating hash values for a given data-set. Available options include SHA1 and SHA256 algorithms. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table.

HASH HAL Module API Summary

Function Name	Example API Call and Description
.open	g_sce_hash.p_api->open(g_sce_hash.p_ctrl, g_sce_hash.p_cfg); HASH module open function. Must be called before performing any sign/verify operations.
.updateHash	g_sce_hash.p_api->updateHash(p_source, num_words, p_dest); Update hash for the num_words words from source buffer p_source. This function is deprecated. The function hashUpdate should be used instead.
.hashUpdate	g_sce_hash.p_api->hashUpdate(g_sce_hash.p_ctrl, p_source, num_words, p_dest); Update hash for the num_words words from source buffer p_source.
.close	g_sce_hash.p_api->close(g_sce_hash.p_ctrl); HASH module close function.
.versionGet	g_sce_hash.p_api->versionGet(p_version); Gets version and stores it in provided pointer p_version.

The RSA interface defines APIs for opening, closing, encrypting, and decrypting data using an RSA algorithm as well as digitally signing and verifying the algorithm. The RSA interface employs a 1024-bit or 2048-bit key. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table.

RSA HAL Module API Summary

Function Name	Example API Call and Description
.open	g_sce_rsa.p_api->open(g_sce_rsa.p_ctrl, g_sce_rsa.p_cfg); RSA module open function. Must be called before performing any encrypt/decrypt or sign/verify operations.

Function Name	Example API Call and Description
.encrypt	g_sce_rsa.p_api->encrypt(g_sce_rsa.p_ctrl, p_key, p_domain, num_words, p_source, p_dest); Encrypt source data from p_source using an RSA public key from p_key and write the results to destination buffer p_dest.
.decrypt	g_sce_rsa.p_api->decrypt (g_sce_rsa.p_ctrl, p_key, p_domain, num_words, p_source, p_dest); Decrypt source data from p_source using an RSA private key from p_key and write the results to destination buffer p_dest.
.decryptCrt	g_sce_rsa.p_api->decryptCrt(g_sce_rsa.p_ctrl, p_key, p_domain, num_words, p_source, p_dest); Decrypt source data from p_source using an RSA private key from p_key and write the results to destination buffer p_dest. RSA private key data is specified in CRT format.
.verify	g_sce_rsa.p_api->verify(g_sce_rsa.p_ctrl, p_key, p_domain, num_words, p_signature, p_padded_hash); Verify signature given in buffer p_signature using the RSA public key p_key for the given padded message hash from buffer p_padded_hash.
.sign	g_sce_rsa.p_api->sign(g_sce_rsa.p_ctrl, p_key, p_domain, num_words, p_padded_hash, p_dest); Generate signature for the given padded hash buffer p_padded_hash using the RSA private key p_key. Write the results to the buffer p_dest.
.signCrt	g_sce_rsa.p_api->signCrt(g_sce_rsa.p_ctrl, p_key, p_domain, num_words, p_padded_hash, p_dest); Generate signature for the given padded hash buffer p_padded_hash using the RSA private key p_key. RSA private key p_key is assumed to be in CRT format. Write the results to the buffer p_dest.

Function Name	Example API Call and Description
.close	g_sce_rsa.p_api->close(g_sce_rsa.p_ctrl); Close the RSA module.
.keyCreate	g_sce_rsa.p_api->keyCreate(g_sce_rsa.p_ctrl, p_private_key, p_public_key); Generates an RSA key pair.
.versionGet	g_sce_rsa.p_api->versionGet(p_version); Gets version and stores it in provided pointer p_version.

The TDES interface defines APIs for encrypting and decrypting data according to the TDES standard. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table.

TDES HAL Module API Summary

Function Name	Example API Call and Description
.open	g_sce_tdes.p_api->open(g_sce_tdes.p_ctrl, g_sce_tdes.p_cfg); Open the TDES module.
.encrypt	g_sce_tdes.p_api-> encrypt(g_sce_tdes.p_ctrl, &key, &iv, nwords, &source, &destination); Encrypt the data.
.decrypt	g_sce_trng.p_api-> decrypt(g_sce_tdes.p_ctrl, &key, &iv, nwords, &source, &destination); Decrypt the data.

Function Name	Example API Call and Description
.close	g_sce_tdes.p_api->close(g_sce_tdes.p_ctrl); Close the TDES module.
.versionGet	g_sce_tdes.p_api->versionGet(p_version); Gets version and stores it in provided pointer p_version.

The TRNG interface defines APIs for computing the random-number generator. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table.

TRNG HAL Module API Summary

Function Name	Example API Call and Description
.open	g_sce_trng.p_api->open(g_sce_trng.p_ctrl, g_sce_trng.p_cfg); Open the TRNG driver for reading random data from the hardware TRNG module.
.read	g_sce_trng.p_api->read(g_sce_trng.p_ctrl, p_rngbuf, nbytes); Generate nbytes of random number bytes and store them in p_rngbuf buffer.
.close	g_sce_trng.p_api->close(g_sce_trng.p_ctrl); Close the TRNG interface driver.
.versionGet	g_sce_trng.p_api->versionGet(p_version); Gets version and stores it in provided pointer p_version.

4.2.10.3 SCE HAL Module Operational Overview

Different cryptographic functions are available for different target MCUs; the following table shows the functionality that is available for each individual MCU-series:

Function	S7G2, S5D9, S5D5	S3A1, S3A3, S3A7, S3A6	S124, S128, S1JA	Notes
TRNG	Generate and read random number	Generate and read random number	Generate and read random number	Generate and read random number
AES	Encryption, decryption, Key Generation - wrapped keys	Encryption, decryption, Key Generation - wrapped keys	Encryption, decryption	Symmetric Key Encryption based on AES standard
AES Key Size	128-bit, 192-bit, 256-bit	128-bit, 256-bit	128-bit, 256-bit	
AES Key Type	Plain text / raw key, Wrapped key	Plain text / raw key, Wrapped key	Plain text / raw key	
AES Chaining Modes	ECB, CBC, CTR, GCM, XTS NOTE: XTS is supported for 128-bit and 256-bit keys only	ECB, CBC, CTR, GCM, XTS	ECB, CBC, CTR	
ARC4	Encryption, decryption	NA	NA	
TDES	Encryption, decryption	NA	NA	
TDES Key Size	192-bit	NA	NA	
TDES Chaining Modes	ECB, CBC, CTR	NA	NA	

Function	S7G2, S5D9, S5D5	S3A1, S3A3, S3A7, S3A6	S124, S128, S1JA	Notes
RSA	Signature Generation, Signature Verification, Public-key Encryption, Private-key Decryption, Key Generation - plain text and wrapped keys	NA	NA	Supports CRT keys and standard keys for private key operations
RSA Key Size	1024-bit, 2048-bit	NA	NA	
RSA Key Type	Plain text / raw key, Wrapped key	NA	NA	
DSA	Signature Generation, Signature Verification	NA	NA	
DSA Key Size	(1024, 128)-bit, (2048, 224)-bit, (2048, 256)-bit	NA	NA	
HASH	MD5, SHA1, SHA224, SHA256	NA	NA	Message digest algorithms
ECC	Key Generation – plain text and wrapped keys, Scalar Multiplication, ECDSA- Signature Generation, ECDSA -Signature Verification,	NA	NA	

Function	S7G2, S5D9, S5D5	S3A1, S3A3, S3A7, S3A6	S124, S128, S1JA	Notes
ECC Key Size	192-bit, 256-bit	NA	NA	
ECC Key Type	Plain Text / Raw Key, Wrapped Key			
Key Installation	AES, ECC, RSA keys (Plain text keys to be prepared as specified in the Appendix)	AES keys (Plain text keys to be prepared as specified in the Appendix)	NA	

Configuration Settings for the R_SCE Module

The endianness of the SCE is set to big endian by default. It can be set to little endian mode.

Please refer to the operational notes on endianness configuration parameter usage.

Configuration Settings for the TRNG Module

Random number-generation can be configured for the maximum number of attempts it makes to the underlying hardware to generate a unique 16-byte random number that differs from the previously-generated random number. On reaching the maximum number of attempts, the read API will return an error code to the caller, otherwise a success code is returned and the generated random number will be transferred to the caller-supplied data buffer.

Configuration Settings for the AES Module

The AES module can be configured for a user-specified key-length, key type (plain text or wrapped key) and chaining modes.

Configuration Settings for the RSA Module

The RSA module can be configured for a user-specified key length and key type: plain text or wrapped keys.

Configuration Settings for the DSA Module

The DSA module can be configured for a user-specified key length.

Configuration Settings for the HASH Module

The HASH module can be configured for a user specified HASH algorithm (depending on the target MCU).

Configuration Settings for the TDES Module

The TDES module can be configured for a user-specified chaining mode.

Configuration Settings for the Key Installation Module

The Key Installation module can be configured to install the user's encrypted key.

Configuration Settings for the ECC Module

The ECC module can be configured for a user-specified key length and key type.

SCE HAL Module Operational Notes and Limitations

- Synergy S7 and S5 devices have the SCE7 and therefore support AES, TRNG, RSA, HASH, DSA, ECC and Key Installation.
- Synergy S3 devices have the SCE5 and therefore support AES, TRNG, & GHASH. GHASH is supported as part of the AES GCM mode. Key Installation is supported as part of AES GCM, ECB, CTR, XTS, CBC chaining modes. Synergy S3 devices do not support MD5, SHA1 / SHA256 HASH functionality.
- Synergy S1 devices only support AES (ECB, CBC and CTR) and TRNG.
- If an unsupported module is added to the project, then a compiler warning will be generated indicating this fact.
- All modules support versionGet API which can be called even before a module is opened.
- R_SCE module interfaceGet API is provided for use by the Framework layer SF CRYPTO modules.InterfaceGet API is used to request a crypto HAL interface. The example below shows usage of this API:

```
crypto_instance_t          * p_crypto;  /* R_SCE  instance */
void * p_interface = NULL;    /* Declare a pointer to hold the output interface
structure object */
crypto_interface_get_param_t     param;
param.hash_type = CRYPTO_TYPE_HASH_256; /* Requesting SHA 256 interface*/
/* It is mandatory for the address of the p_interface pointer be passed to the A
PI */
p_crypto->p_api->interfaceGet(&param, &p_interface);
```

- All crypto APIs may return SSP_ERR_ASSERTION on null pointer input or invalid input parameters. All APIs return error codes documented in sf_crypto_err_t or ssp_err_t which are within the width of the type uint32_t.
- Crypto hardware engine does not support reentrancy. When the crypto hardware engine is busy performing a task, any new request will receive a status error code SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT.

Endianness configuration parameter usage:

- The default mode is big endian where the input and output parameters (example: keys, payload and IV) are required to be in uint32_t data type.
- The little endian mode allows the user to have uint8_t / byte array for input and output parameters (example: keys, payload and IV) and they should be cast to (uint32_t *).
- The endianness configuration is set at the initialization of the SCE module and remains in effect until the module is closed. Hence all data should be formatted accordingly.Example:
  - Select the Big endian mode when the data is in uint32_t and big endian format:

```
uint32\_t test\_data\[5\] = {0x84983E44, 0x1C3BD26E, 0xBAAE4AA1,
0xF95129E5, 0xE54670F1};
```

  - Select the Little endian mode when the same data is in byte array format

```
uint8\_t test\_data\_byte\_array\[20\] =
{0x84, 0x98, 0x3E, 0x44, 0x1C, 0x3B, 0xD2, 0x6E, 0xBA, 0xAE, 0x4A,
0xA1, 0xF9, 0x51, 0x29, 0xE5, 0xE5, 0x46, 0x70, 0xF1};
```

- The format of RSA keys generated by the keyCreate API is as follows:

NOTE: The endianness is the same as that set during SCE initialization.

RSA key format:**RSA Public key format**:WORD 0 : Public key exponentWORD 1: Start of RSA modulus(128 bytes for RSA 1024-bit and 256 bytes for RSA 2048-bit keys)**RSA Private key in plain text standard format** WORD 0 : Private key exponent (128 bytes for RSA 1024-bit and 256 bytes for RSA 2048-bit keys)Followed by RSA modulus. (128 bytes for RSA 1024-bit and 256 bytes for RSA 2048-bit keys)RSA Private key in plain text CRT formatThe components are ordered in the following order with exponent2 at byte 0:

```
exponent2 // the second factor's CRT exponent, a positive integer

prime2 // the second factor, a positive integer

exponent1 // the first factor's CRT exponent, a positive integer

prime1 // the first factor, a positive integer coefficient // the (first) CRT
coefficient, a positive integer
```

- The AES encrypt() and decrypt() functions do not support data padding. These functions operate on data lengths that are multiples of 16 bytes. (Data padding needs to be handled by the user application.) AES GCM mode may require support for authentication data that may not be a multiple of 16 bytes. To support this, zeroPaddingEncrypt() and zeroPaddingDecrypt() function APIs are provided only for the AES GCM mode.
  - For AES GCM, when using uint32_t arrays in big endian mode, if the number of bytes of data is not a multiple of 4 (WORD length) it should be zero padded.
- The TDES encrypt() and decrypt() functions do not support data padding. These functions operate on data lengths that are multiples of 8 bytes. (Data padding needs to be handled by the user application.)
- HASH Module - MD5
  - MD5 requires byte swapping of the final message digest output. Intermediate updates (partial updates) are not required to be byte swapped.
  - MD5 also requires length field within the formatted final block to be in big endian format before calling the hash hashUpdate API.
- ECC Module:
  - The data buffers for the keyCreate API need to be the exact length of the key that will be generated.
- Refer to the most recent SSP release notes for the most up-to-date limitations on this module.

4.2.10.4 Including the SCE HAL Module in an Application

This section describes how to include the SCE HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack.

To add the SCE Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the SCE HAL module is r_sce. This name can be changed in the associated Properties window.)

SCE Driver Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sce_aes_0 AES Driver on r_sce_aes	Threads	New Stack> Driver> Crypto> AES Driver on r_sce_aes
g_sce_arc4_0 ARC4 Driver on r_sce_arc4	Threads	New Stack> Driver> Crypto> ARC4 Driver on r_sce_arc4
g_sce_dsa_0 DSA Driver on r_sce_dsa	Threads	New Stack> Driver> Crypto> DSA Driver on r_sce_dsa
g_sce_ecc_0 ECC Driver on r_sce_ecc	Threads	New Stack> Driver> Crypto> ECC Driver on r_sce_ecc
g_sce_hash_0 HASH Driver on r_sce_hash	Threads	New Stack> Driver> Crypto> HASH Driver on r_sce_hash
g_sce_key_initialization_0 ECC Driver on r_sce_key_initialization	Threads	New Stack> Driver> Crypto> Key Initialization Driver on r_sce_key_initialization
g_sce_rsa_0 RSA Driver on r_sce_rsa	Threads	New Stack> Driver> Crypto> RSA Driver on r_sce_rsa
g_sce_tdes TDES Driver on r_sce_tdes	Threads	New Stack> Driver> Crypto> TDES Driver on r_sce_tdes
g_sce_trng TRNG Driver on r_sce_trng	Threads	New Stack> Driver> Crypto> TRNG Driver on r_sce_trng

When the Crypto Drivers on r_sce are added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information will be box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks.

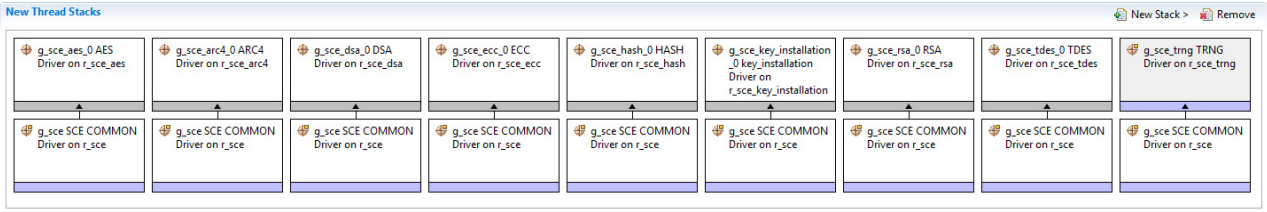


Figure 297: SCE HAL Module Stack (AES, ARC, DSA, ECC, HASH, Key Installation, RSA, TDES, and TRNG included)

4.2.10.5 Configuring the SCE HAL Module

The SCE HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for reference.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

AES HAL Module Parameters and Settings

ISDE Property	Value	Description
Name	g_sce_aes_0	Module name
Key Length	User defined, default is 128 bits. Allowed values for S7G2 and S5D9 devices: 128, 192, or 256 bits. Allowed values for S3A7 and S124 devices: 128 or 256 bits	Key length used for encryption/decryption operations by this instance of the driver
Channel	User defined, default is CBC. Allowed values for S7G2 and S5D9 devices: ECB, CBC, CTR, GCM, XTS. Allowed values for S3A7 device: ECB, CBC, CTR, GCM. Allowed values for S124 device: ECB, CBC, CTR	Block cipher chaining mode used for encryption/decryption operations by this instance of the driver

ISDE Property	Value	Description
Key Format	Plain Text Key, Wrapped Key (Not available for S1 MCU Series) Default: Plain Text Key	Key format selection

ARC4 HAL Module Parameters and Settings

ISDE Property	Value	Description
Name	g_sce_arc4_0	Module name
Key Length	Default 0	Key length in number of bytes
Key Name	g_arc4_0_key	Key name- must be defined as unit8_array type data in user code

DSA HAL Module Parameters and Settings

Parameter	Value	Description
Name	g_sce_dsa_0	Module name
Key Length	User defined, default is (2048, 256) bits. Allowed values for S7G2 and S5D9 devices: (1024, 160), (2048, 224) or (2048, 256) bits Allowed values for S3A7 and S124 devices: Not available.	Key length used for signing/verification operations by this instance of the driver.

ECC HAL Module Parameters and Settings

ISDE Property	Value	Description
Name	g_sce_ecc_0	Module name
Key Length	192, 256 Default: 256	Key length used for encryption/decryption operations by this instance of the driver

ISDE Property	Value	Description
Key Format	Plain Text Key, Wrapped Key (Not available for S1 MCU Series) Default: Plain Text Key	Key format selection

HASH HAL Module Parameters and Settings

Parameter	Value	Description
Name	g_sce_hash_0	Module Name
Algorithm	User defined, default is SHA256. Allowed values for S7G2 and S5D9 devices: SHA1, or SHA256. Allowed values for S3A7 and S124 devices: Not available.	Algorithm used for computing the message digest/hash on the message data.

Key Installation HAL Module Parameters and Settings (Not supported for S1 Series MCUs)

Parameter	Value	Description
Name	g_sce_key_installation_0	Module Name

RSA HAL Module Parameters and Settings

Parameter	Value	Description
Name	g_sce_rsa_0	Module Name
Key Length	User defined, default is 2048 bits. Allowed values for S7G2 and S5D9 devices: 1024 or 2048 bits. Allowed values for S3A7 and S124 devices: Not available.	Key length used for signing/verification/encryption/decryption operations by this instance of the driver
Key Format	Plain Text Key, Wrapped Key (Not available for S1 MCU Series) Default: Plain Text Key	Key format selection

TDES HAL Module Parameters and Settings

Parameter	Value	Description
Name	g_sce_tdes_0	Module Name
Chaining Mode	ECB, CBC, CTR Default: CBC	Chaining mode selection

TRNG HAL Module Parameters and Settings

Parameter	Value	Description
Name	g_sce_trng_0	Module Name
Max. Attempts	User defined, default is 2	Sets the maximum number of attempts when a newly generated random number differs from the previously generated random number

4.2.10.6 Using the SCE HAL Module in an Application

The SCE COMMON module must be initialized prior to calling any other crypto operation. This is done by using the open API. (For example, g_sce.p_api->open(g_sce.p_ctrl, g_sce.p_cfg);

Other SCE COMMON APIs can be used to determine SCE operation. For example, the initialization status of the SCE module can be obtained by using the statusGet API. The interfaceGet API can be used to obtain interface information. See the SCE COMMON API table for additional information.

The typical steps in using the SCE HAL module in an example AES application are:

1) Use the SCE open API to initialize the SCE COMMON module. This must be called prior to any other crypto operation.
2) Specify parameters using the associate API (AES addAdditionalAuthenticationData, for example).
3) Use the AES open API to start the AES module.
4) Encrypt data using the encrypt API.
5) Decrypt data using the decrypt API.
6) Close the SCE instance using the SCE close API.

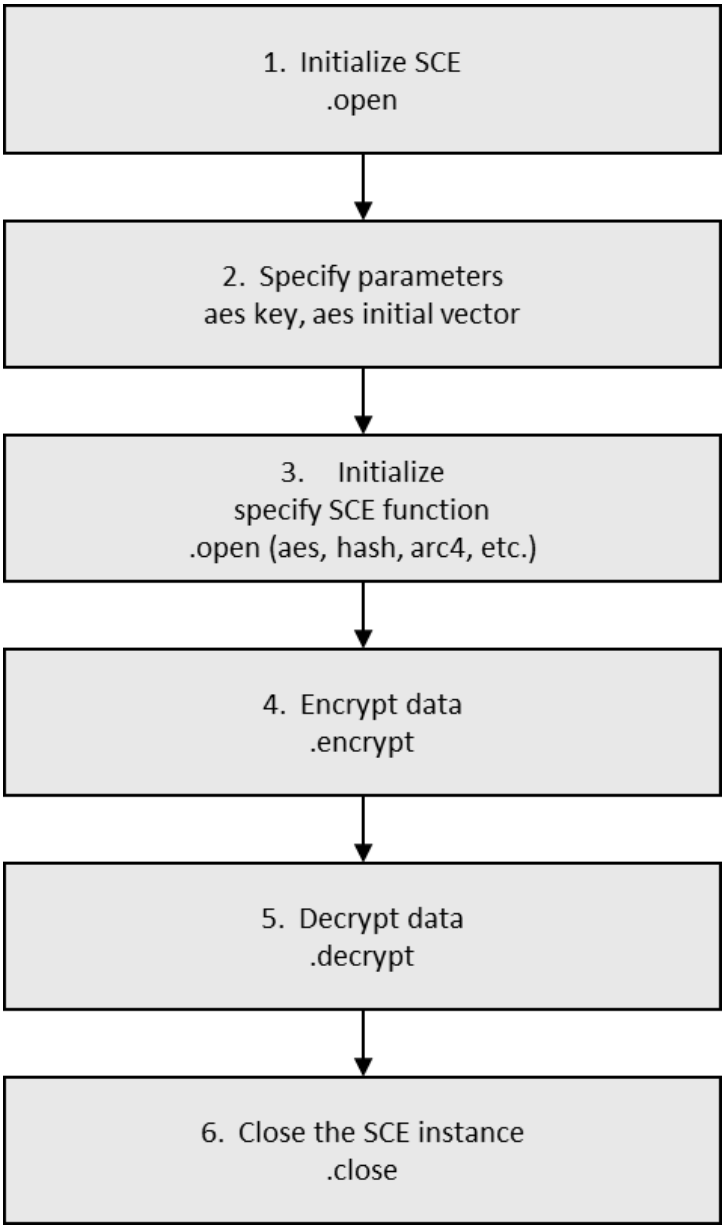


Figure 298: Flow Diagram of a Typical Cryptography Application

Specific cryptographic flows and associated use notes are given below:.

1) To use the SCE module:

- Use the open API to Initialize the SCE and the SCE HAL module (R_SCE) through the SCE common driver.

- Configure the endianness (little-endian or big-endian) for the input /output data for all the HAL APIs. The big endian mode is configured by default. See HAL Module Operational notes for details on endianness configuration.
- The open function cannot be called again until the module is closed.

2) To use the AES functions:

- Initialize an AES interface instance with open API.
- Specify the configuration parameters for the instance to use the associated APIs. AES key sizes available are 128-bit, 192-bit or 256-bit. Chaining modes supported are ECB, CBC, CTR, GCM and XTS.
- To encrypt data, use the encrypt API.
- To decrypt data, use the decrypt API.
- To generate keys, use the createKey API.

NOTE: The createKey API in the AES module creates AES wrapped keys.

AES plain text keys can be generated with the services of the TRNG module.

- Close the interface instance with close API.
- IV provided for AES GCM operations can be either 96-bits or a 96-bit IV formatted to 128-bits.

– **Example:**

– **96-bit IV: 94c1935afc061cbf254b936f**

– **96-bit IV formatted to 128-bits: 94c1935afc061cbf254b936f00000001**

Note that subtle differences exist in GCM operations as follows.

AES GCM encryption:

1) AAD (Additional Authenticated Data) is optional. If it is to be used, it has to be provided prior to encrypting / decrypting any data by setting p_dest = NULL
2) Encryption: set p_source to input data and p_dest will return encrypted data
3) Get/Compute Tag: set p_source = NULL.

AES GCM decryption:

1) Set expected tag value using the setGcmTag() function
2) Provide any Add Authentication Data (AAD), invoke this API using p_dest = NULL
3) Decryption: set p_source to input encrypted data, decrypted data will be returned in p_dest
4) To verify the tag, invoke this API using p_source = NULL and p_dest = NULL. The return value indicates authentication tag verification status.The decrypted data is to be used only if the tag is verified successfully.zeroPaddingEncrypt/zeroPaddingDecrypt APIs can be used for GCM operations when the data is not a multiple of the block size.
5) To use the TDES functions:

- Initialize the TDES interface instance with open API.
- Specify the configuration parameters for the instance to use the associated APIs. Select the TDES chaining mode ECB, CBC or CTR.
- Encrypt data using the encrypt API.
- Decrypt data using the decrypt API
- Close the interface instance with close API.

6) To use the ARC4 functions:

- Initialize an ARC4 interface instance with open API.
- Specify the configuration parameters for the instance to use the associated APIs. ARC4 key can be specified by length (anywhere from 64-bits, 2048-bits) and location.
- The key to be used can be set through the keySet API.
- To encrypt or decrypt data use the arc4Process API.
- Close the interface instance with close API.

7) To use the RSA functions:

- Supported key-lengths are 1024-bits and 2048-bits.

  NOTE: For encryptAPI, decrypt API, decryptCrt API, sign and signCrt API: The size of the data buffer is indicated in num_words. It must be 32 words /128 bytes/1024-bits for 1024-bit keys and 64 words /256 bytes/2048-bits for the 2048-bit keys.

- Select the RSA interface instance based on the desired key length and initialize the selected RSA interface instance with open API.
- To encrypt data using RSA public Key use the encrypt API.
- To Decrypt data using RSA private Key use the decrypt API.
- To Decrypt data using RSA private Key which is in the CRT format use the decryptCrt API.
- To generate signature for a given padded hash using RSA private Key which is in the standard format use the sign API.
- To generate signature for a given padded hash using RSA private Key which is in the CRT format use the signCrt API.
- To verify signature for a given padded hash using RSA public Key which is in the standard format use the verify API.

– **To generate keys use the keyCreateKey API.**

  NOTE: keyCreate API in RSA module creates RSA plain-text keys or wrapped keys based on the input parameters to the API.

- Close the interface instance with close API.

8) To use the DSA functions:
   - Supported key-lengths are (1024,160)-bits, (2048,224)-bits and (2048,256)-bits
   - Select the DSA interface instance based on the desired key length and initialize the selected DSA interface instance with open API.
   - To generate signature using the DSA private key use hashSign API.
   - To verify signature using the DSA public key use hashVerify API.
   - Close the interface instance with close API.
9) To use the HASH algorithms:
   - MD5, SHA1 and SHA256 hash methods are supported. Select the desired
   - To compute the message digest use hashUpdate API.
   - Close the interface instance with close API.
10) To use the True Random Number Generator functions:
   - Initialize an TRNG interface instance with open API.
   - Generate random number with read API.
   - Close the interface instance with close API.
11) To use the Key Installation API:
   - Initialize Key Installation interface instance with open API.
   - Specify Key installation key structures for the user's encrypted key and Renesas provided key index (key size, key format, pointer to buffer and buffer length).
   - Specify Output key structure with pointer to buffer and buffer length.
   - To install the key use keyInstall API.
   - Close the interface instance with close API.
12) To use the ECC functions:
   - To generate the domain parameters for NIST curves, use OpenSSL command to generate curves as shown below.
   - **ECC P-192: openssl ecparam -name secp192r1 -param_enc explicit -text | more**
   - **ECC P-256: openssl ecparam -name secp256r1 -param_enc explicit -text | more**
   - Supported key sizes are 192-bits and 256-bits.

NOTE: For scalarMultiplication API keyCreate, sign and verify API: The size of the data buffer is indicated in data_length field of r_crypto_data_handle_t. The actual buffer must be pointed by p_data field of r_cryp-

to_data_handle_t. Select the ECC interface instance based on the desired key length and key type and initialize the selected ECC interface instance with open API.

- To perform ECC Scalar Multiplication use the scalarMultiplication API.
- To generate signature for a given padded hash using ECC private Key which is in the standard format use the sign API.
- To verify signature for a given padded hash using ECC public Key which is in the standard format use the verify API.
- To generate ECC keys use the keyCreate API.
- Close the interface instance with close API.

Close the SCE and the SCE HAL module using the close API.

4.2.11 DAC Driver

The DAC HAL module provides high-level APIs for digital-to-analog conversion applications implemented on r_dac. The DAC HAL module supports a dual-channel 12-bit D/A converter (DAC12) peripheral on Synergy MCUs.

4.2.11.1 DAC HAL Module Features

This module configures the dual-channel 12-bit D/A Converter (DAC12) to output one of 4096 voltage levels between the positive and negative reference voltages. The module include configuration settings to:

- Set either a left-justified or right-justified 12-bit value format for the 16-bit input data registers
- Enable or disable output amplifiers
- Operate in synchronous anti-interference mode with the Analog-to-Digital Converter (ADC) module.

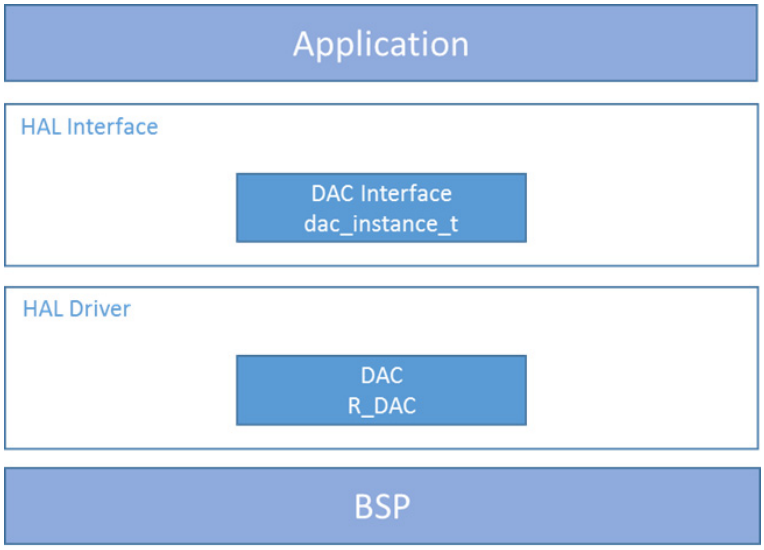


Figure 299: DAC HAL Module Block Diagram

4.2.11.2 DAC HAL Module APIs Overview

The DAC HAL module defines APIs to open, close, start, stop and write to the DAC. The following table lists each of the APIs available with an example call and a short description. A listing of status return values follows the API summary table.

DAC HAL Module API Summary

Function Name	Example API Call and Description
open	g_dac.p_api->open(g_dac.p_ctrl, g_dac.p_cfg) Initial Configuration.
close	g_dac.p_api->close(g_dac.p_ctrl) Close the D/A Converter.
write	g_dac.p_api->write(g_dac.p_ctrl, val) Write Sample value to the D/A Converter.

Function Name	Example API Call and Description
start	g_dac.p_api->start(g_dac.p_ctrl) Start the D/A Converter if it has not been started yet.
stop	g_dac.p_api->stop(g_dac.p_ctrl) Stop the D/A Converter if the converter is running.
versionGet	g_dac.p_api->versionGet(&version) Retrieve the API version with the version pointer.

NOTE: Review the *SSP User's Manual* API References for the associated module. The SSP defines operations, function data structures, typedefs, defines, API data, API structures, and function variables.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	Wrong parameter.
SSP_ERR_IN_USE	DAC resource is locked.

NOTE: Lower-level drivers may return common error codes. Refer to API References in the *SSP User's Manual* for a definition of relevant status return values in the associated module.

4.2.11.3 DAC HAL Module Operational Overview

The DAC HAL module configures the dual-channel 12-bit D/A converter (DAC12) to output one of 4096 voltage levels between positive and negative reference voltages. The module can be used to configure the 12-bit output to left-or-right-justified format for 16-bit input data registers. The DAC HAL module can also enable or disable output amplifiers or operate in synchronous anti-interference mode with the ADC HAL module.

DAC HAL Module Important Operational Notes and Limitations

The DAC HAL module requires the following initialization steps:

- DAC module stop-bit cleared to zero.

- DAC channel output-enable set to one.

The DAC module stop-bit is cleared to zero at the time of an open call when the driver's instance counter is zero. The driver's instance counter is initialized to zero, incremented when a channel open returns successfully, and decremented when a channel close is called. The DAC module stop-bit is set to one when the driver's instance counter is decremented to zero on a close call.

The DAC channel output-enable is set to one when a channel write is called the first time after open was called successfully. The open call writes a zero to the dac_ctrl_t structure element channel_started. When write is called with channel_started cleared to zero, the DAC output-enable bit for that channel is set to one. The DAC output-enable for the channel is cleared to zero on close and stop calls.

- Pin configuration is not implemented for the DAC HAL module. Currently, DA0 and DA1 outputs are enabled by the reset values in the pin configuration control register's ASEL field.
- Voltage reference selection for the DAC HAL module is not implemented. Currently, no reference is selected by the reset values in the D/A VREF control register (DAVREFCR) which is a valid condition.
- Configuration of DAC input-event triggering for conversion is not currently implemented. The default reset value (zero) of the control register DAE bit allows individual triggering for each channel.
- Event signal-input for synchronization of the DAC HAL module conversions is not currently implemented.
- Refer to the latest SSP Release Notes for operational limitations related to this module.

4.2.11.4 Including the DAC HAL Module in an Application

This section describes how to include the DAC HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications

To add the DAC Driver to an application, use the stacks selection sequence in the following table. (The default name for the DAC is g_dac0. This name can be changed in the associated Properties window.)

Tale DAC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_dac0 DAC Driver on r_dac	Threads	New Stack > Driver > Analog > DAC Driver on r_dac

When the DAC HAL module on DAC is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level drivers. Any drivers that need additional configuration information will be box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

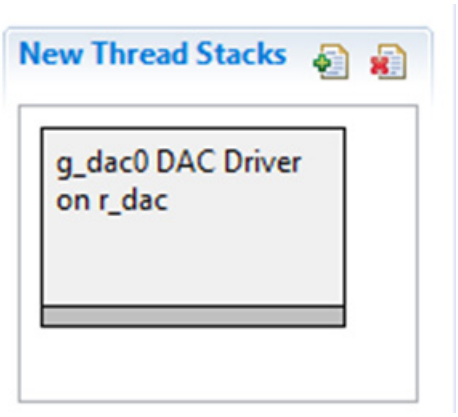


Figure 300: DAC HAL Module Stack

4.2.11.5 Configuring the DAC HAL Module

The DAC HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window. The interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+.) This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the DAC HAL Module on r_dac

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enables or disables the parameter checking.
Name	g_dac0	Module Name.

ISDE Property	Value	Description
Channel	0,1	Set to 0 for output DA0 or 1 for output DA1.
Synchronize with ADC	Enabled, Disabled (Default: Disabled)	Set to true for anti-interference synchronization with the Analog-to-Digital Converter (ADC) Module. Set to false if the power supply interference between the analog modules is not a problem, or if the asynchronous conversion by the DAC Module is desired.
Data Format	Right Justified, Left Justified (Default: Right Justified)	Set to zero if 12-bit data values are loaded in bits 11 through 0, or right justified. Set to one, if 12-bit data values are loaded in bits 15 through 4, or left justified.
Output Amplifier	Enable, Disable (Default: Disable)	Set to true, if output amplifier hardware function is desired. Set to false to bypass output amplifier hardware function.

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults can be desirable. For example, it might be useful to select different events, depending on the operation desired.

DAC HAL Module Clock Configuration

The DAC HAL module does not require any specific clock configuration.

DAC HAL Module Pin Configuration

To use the DAC HAL module, the port pins for the channels receiving the analog input must be set as analog pins in the pin configurator. The following table lists a method to select pins within the SSP configuration window. The subsequent table illustrates configuration settings for DAC pins.

Pin Selection Sequence for the DAC Driver on r_dac

Resource	ISDE Tab	Pin Selection Sequence
DAC	Pins	Select Peripherals > Analog: DAC12 > DAC120

Pin Configuration Settings for the DAC Driver on r_dac

Property	Value	Description
Module Name	DAC120	DAC Peripheral Module.
Operation Mode	Enabled, Disabled (Default: Enabled)	DAC Peripheral operation mode.
DA0	None, DA0 (Default: None)	DAC Output Pin.
DA1	None, DA1 (Default: None)	DAC Output Pin.

NOTE: The example settings illustrate a project using the Synergy S7G2 and SK-S7G2 Kits. Other Synergy Kits and Synergy MCUs may have pin configuration settings that differ in availability.

4.2.11.6 Using the DAC HAL Module in an Application

The typical steps in using the DAC HAL module in an application are:

1) Initialize the DAC HAL module using the open API.
2) Write a data value using the write API.
3) Start writing data using the start API.
4) Continue writing data values as needed using the write API.
5) Stop writing data using the stop API.
6) Close the DAC HAL module using the close API.

The following figure shows common steps in a typical operational flow:

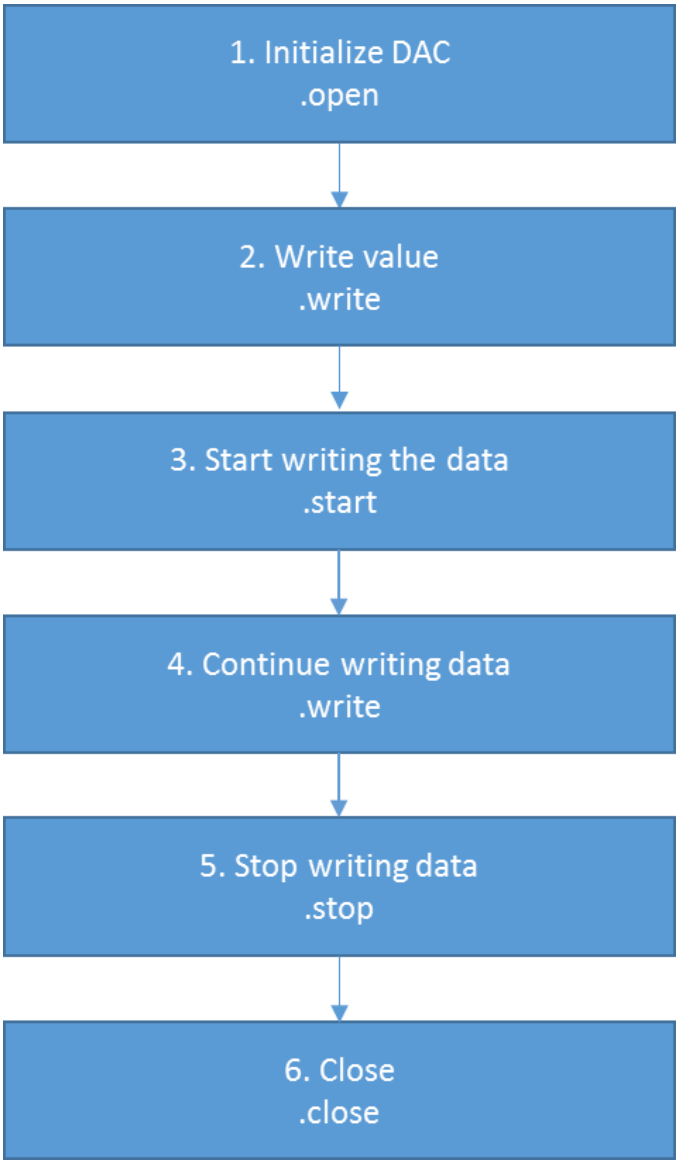


Figure 301: Typical DAC HAL Module application flow diagram

4.2.12 DAC 8 Driver

4.2.12.1 DAC 8 Driver

The DAC 8 HAL module provides high-level APIs for digital-to-analog conversion applications implemented on r_dac8. The DAC 8 HAL module supports an 8-bit D/A converter (DAC 8) peripheral on Synergy S128 MCUs.

4.2.12.2 DAC 8 HAL Module Features

- Available on the Synergy S128 MCU
- 8-Bit D/A Converter- three channels
- Left-Justified or Right-Justified Input Data Format
- Synchronization with the Analog-to-Digital Converter (ADC) module
- Multiple Operation Modes
  - Normal
  - Real-Time (Event Link)
- Charge Pump Control

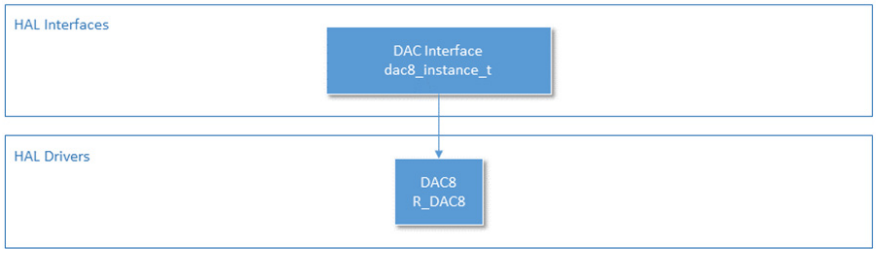


Figure 302: DAC 8 HAL Module Organization, Options and Stack Implementations

4.2.12.3 DAC 8 HAL Module APIs Overview

The DAC 8 HAL module defines APIs for opening, closing, starting, stopping, and writing to the DAC. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows.

DAC 8 API HAL Module Summary

Function Name	Example API Call and Description
open	g_dac8.p_api->open(g_dac8.p_ctrl, g_dac8.p_cfg) Initial Configuration.
close	g_dac8.p_api->close(g_dac8.p_ctrl) Close the D/A Converter.

Function Name	Example API Call and Description
write	g_dac8.p_api->write(g_dac8.p_ctrl, val) Write Sample value to the D/A Converter.
start	g_dac8.p_api->start(g_dac8.p_ctrl) Start the D/A Converter if it has not been started yet.
stop	g_dac8.p_api->stop(g_dac8.p_ctrl) Stop the D/A Converter if the converter is running.
versionGet	g_dac8.p_api->versionGet(&version) Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the *SSP User's Manual* API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_HW_LOCKED	DAC resource is locked.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ASSERTION	Wrong parameter.
SSP_ERR_IP_CHANNEL_NOT_PRESENT	Wrong channel selected.

NOTE: Lower level drivers may return Common Error Codes. Refer to SSP User's Manual* API References for the associated module for a definition of all relevant status return values.

4.2.12.4 DAC 8 HAL Module Operational Overview

The DAC 8 HAL module configures the 8-bit D/A converter (DAC 8) to output one of 256 voltage levels between positive and negative reference voltages. The driver can be configured to accept the 8-bit output data in left-or-right-justified format in a 16-bit input data. The driver supports two modes for the DAC.

- Normal mode – The D/A output is updated on writes to the data register.
- Real-Time (Event Link) – The D/A output is updated on an Event Link event. While in this mode the data register can be written at any time. An Event Link event triggers the start of conversion. Refer to the "ELC Interface" in the *SSP User's Manual* for more information.

To reduce the noise present in ADC readings the driver can configure synchronous anti-interference mode with the ADC module. This reduces conversion noise by disabling the DAC charge while the ADC is sampling. Check the hardware manual to determine if this feature is supported.

For operation at low AVCC voltage the driver can enable or disable the hardware charge pump.

4.2.12.5 DAC 8 HAL Module Important Operational Notes and Limitations

4.2.12.6 DAC 8 HAL Module Operational Notes

The DAC 8 channel output is enabled during dac8_api_t::start() and dac8_api_t::write() and disabled during dac8_api_t::stop() and dac8_api_t::close().

4.2.12.7 DAC 8 HAL Module Limitations

- The DAC 8 uses the DAC 8 peripheral available on the Synergy S128 MCU.
- The DAC 8 driver does not configure the output pins for analog output.
- The DAC 8 driver does not configure the ELC peripheral for real-time mode. The user needs to configure the Event Link Controller in addition to enabling real-time mode in the DAC 8 module.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.12.8 Including the DAC 8 HAL Module in an Application

This section describes how to include the DAC 8 HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the *SSP User's Manual* to learn how to manage each of these important steps in creating SSP based applications.

To add the DAC 8 Driver to an application, simply add it to a thread using the Stacks Selection Sequence provided in the following table. (The default name for the DAC is g_dac8_0. This name can be changed in the associated Properties window.)

DAC Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_dac8_0 DAC Driver on r_dac8	Threads	New Stack > Driver > Analog > DAC Driver on r_dac8

When the DAC HAL module on DAC is added to the Thread Stack as shown in the following figure, the configurator automatically adds any needed lower level drivers. Any drivers that need additional configuration information is box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

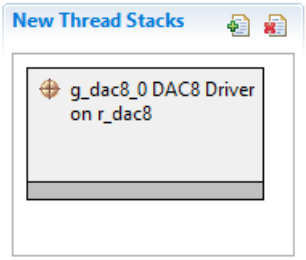


Figure 303: DAC 8 HAL Module Stack

4.2.12.9 Configuring the DAC 8 HAL Module

The DAC 8 HAL module must be configured by you for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections (such as interrupts or operating modes) which must be configured for lower level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP Configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the properties window of the associated module. Simply select the indicated module and then view the properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the properties window in the ISDE includes an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+.) This level of detail is not included in the configuration properties tables below, but is easily visible with the ISDE when configuring interrupt priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the Configuration Table Settings given below. This helps to orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for DAC 8 HAL Module

Parameter	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enable or disable the parameter error checking.

Parameter	Value	Description
Name	g_dac8_0	Module name
Channel	0	Channel selection- three channels are supported.
Synchronize with ADC	Enabled, Disabled (Default: Disabled)	Choose whether to sync with the ADC module
Data Format	Right Justified, Left Justified (Default: Right Justified)	Data format selection
DAC Mode	Normal Mode, Real-time (Event Link) Mode (Default: Normal Mode)	DAC mode selection
Charge Pump Enabled (Requires MOCO active)	Enabled, Disabled (Default: Enabled)	Enable or disable the charge pump

NOTE: The above setting examples and defaults are for a project using the Synergy S1 MCU Family.

In some cases, settings other than the defaults for module properties can be desirable. For example, it might be useful to enable the output amplifier in some applications.

4.2.12.10 DAC 8 HAL Module Clock Configuration

The DAC 8 HAL module requires no specific clock configuration.

4.2.12.11 DAC 8 HAL Module Pin Configuration

To use the DAC 8 HAL module, the port pins for the channels receiving the analog input must be set as analog pins in the pin configurator. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table has an example selection for the DAC pins.

Pin Selection Sequence for the DAC 8 Driver on r_dac8

Resource	ISDE Tab	Pin Selection Sequence
DAC 8	Pins	Select Peripherals > Analog:DAC8 > DAC80

Pin Configuration Settings for the DAC 8 Driver on r_dac8

Property	Value	Description
Module Name	DAC80	DAC Peripheral Module.
Operation Mode	Enabled, Disabled (Default: Enabled)	DAC Peripheral operation mode.
DA0	None, DA0 (Default: None)	DAC Output Pin.
DA1	None, DA1 (Default: None)	DAC Output Pin.

NOTE: The above example settings are for a project using the Synergy S128 MCU and the DK-S128 Kit.

4.2.12.12 Using the DAC 8 HAL Module in an Application

The typical steps in using the DAC 8 HAL module in an application are:

1) Initialize the DAC 8 HAL module using the dac8_api_t::open API.
2) Write a value using the dac8_api_t::write API.
3) Start a conversion using the dac8_api_t::start API.
4) Continue writing values as needed using the dac8_api_t::write API.
5) Stop conversion using the dac8_api_t::stop API.
6) Use the dac8_api_t::close call to power down the peripheral.

The preceding steps are shown in a typical operational flow diagram in the following figure.

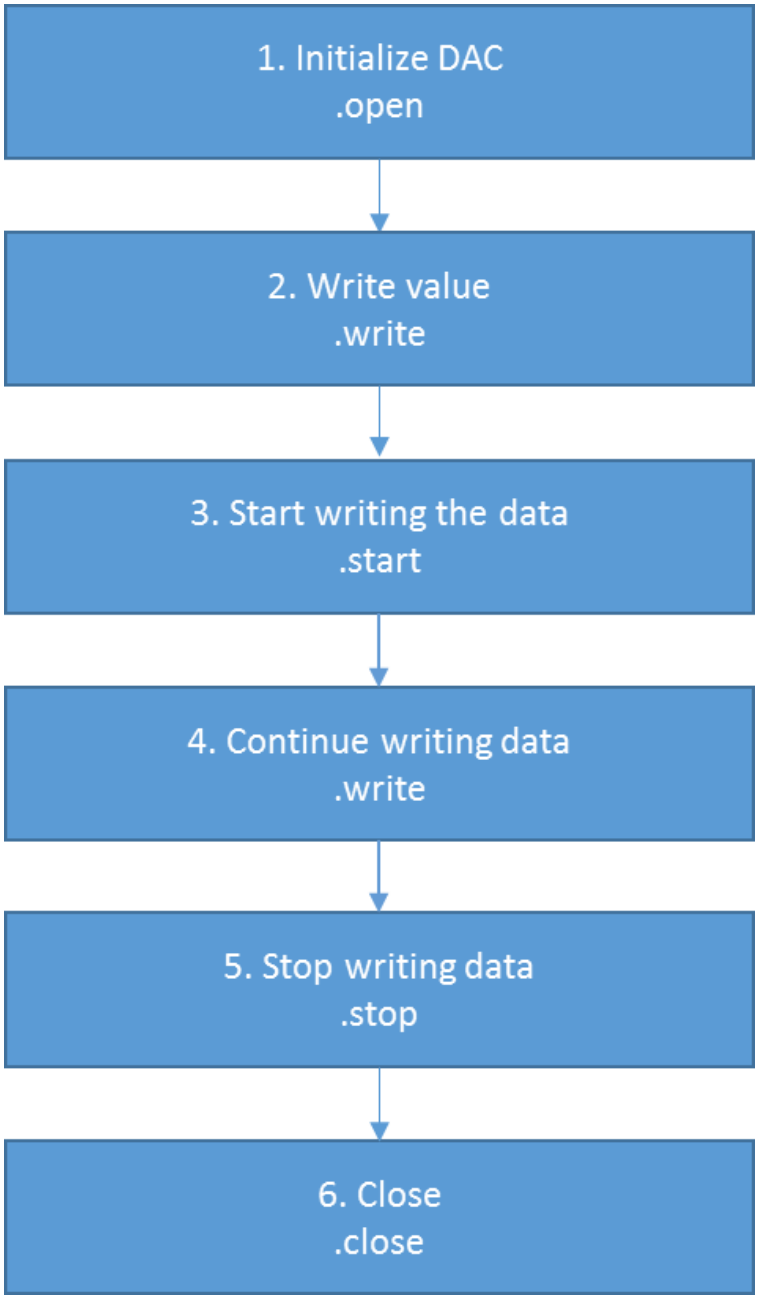


Figure 304: Flow Diagram of a Typical DAC 8 HAL Module Application

4.2.13 GLCDC Display Driver

The Graphics LCD Controller (GLCDC) HAL module is a high-level API for GLCDC applications and is implemented on r_glcd. The GLCDC HAL module uses the Graphics LCD Driver peripheral on the Synergy MCU. A user-defined callback can be created to handle frame buffer switching and underflow detection.

4.2.13.1 GLCDC HAL Module Features

- Supports LCD panels with RGB interface (up to 24 bits) and sync signals (HSYNC, VSYNC, and Data Enable (optional)
- Supports various color formats for input graphics planes (RGB888, ARGB888, RGB565, ARGB1555, ARGB4444, CLUT8, CLUT4, CLUT1)
- Supports the Color Look-Up Table (CLUT) usage for input graphics planes with 512 words (32 bits/word)
- Supports various color formats for output (RGB888, RGB666, RGB565, Serial RGB)
- Can input two graphics planes on top of the background plane and blend them on the screen
- Generates a dot clock to the panel. The clock source is selectable from internal or external (LCD_EXTCLK)
- Supports brightness adjustment, contrast adjustment and gamma correction
- Supports GLCDC interrupts to handle frame-buffer switching or underflow detection

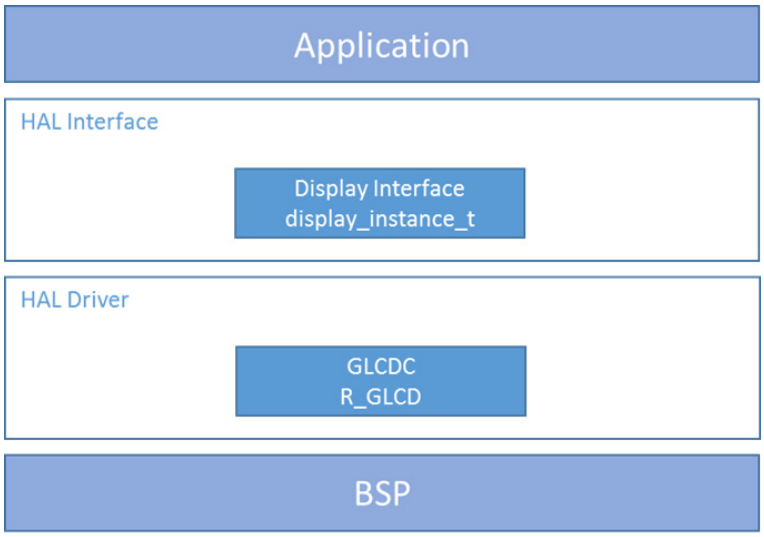


Figure 305: GLCDC HAL Module Block Diagram

4.2.13.2 GLCDC HAL Module APIs Overview

The GLCDC HAL module defines APIs for opening, closing, starting, stopping and controlling the display of information on an LCD panel. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

GLCDC HAL Module API Summary

Function Name	Example API Call and Description
open	g_display.p_api->open (g_display.p_ctrl, g_display.p_cfg); Open display device.
close	g_display.p_api->close (g_display.p_ctrl); Close display device.
start	g_display.p_api->start(g_display.p_ctrl); Display start.
stop	g_display.p_api->stop(g_display.p_ctrl); Display stop.
layerChange	g_display.p_api->layerChange(g_display.p_ctrl, &layercng, frame) Change layer parameters at runtime.
correction	g_display.p_api->correction(g_display.p_ctrl, &display_correction) Color correction.
clut	g_display.p_api->clut(g_display.p_ctrl, &clut, frame) Set CLUT for display device.
statusGet	g_display.p_api->statusGet(g_display.p_ctrl, &status) Get status for display device.
versionGet	g_display.p_api->versionGet(&version) Retrieve the API version using the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API call successful.
SSP_ERR_ASSERTION	Parameter has invalid value.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter in the argument.
SSP_ERR_HW_LOCKED	GLCDCC resource is locked.
SSP_ERR_CLOCK_GENERATION	Dot clock cannot be generated from clock source.
SSP_ERR_INVALID_TIMING_SETTING	Invalid panel timing parameter.
SSP_ERR_INVALID_LAYER_SETTING	Invalid layer setting found.
SSP_ERR_INVALID_LAYER_FORMAT	Invalid format is specified.
SSP_ERR_INVALID_GAMMA_SETTING	Invalid gamma correction setting found.
SSP_ERR_NOT_OPEN	The function call is performed when the driver state is equal to DISPLAY_STATE_CLOSED.
SSP_ERR_INVALID_UPDATE_TIMING	A function call is performed when the GLCDC is updating register values internally.
SSP_ERR_INVALID_MODE	Function call is performed when the driver state is not DISPLAY_STATE_OPENED.
SSP_ERR_INVALID_CLUT_ACCESS	Illegal CLUT entry or size is specified.
p_version	The version number.

NOTE: Lower level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.13.3 GLCDC HAL Module Operational Overview

The GLCDC HAL module controls an LCD panel. The following figure shows an overview of the graphics data flow using the GLCDC HAL module. The module supports reading graphics frame image data from memory (up to two frames)

and blending those images on top of the monochrome background screen. The driver supports CLUT memory and specifies the graphic frame format for the CLUT.

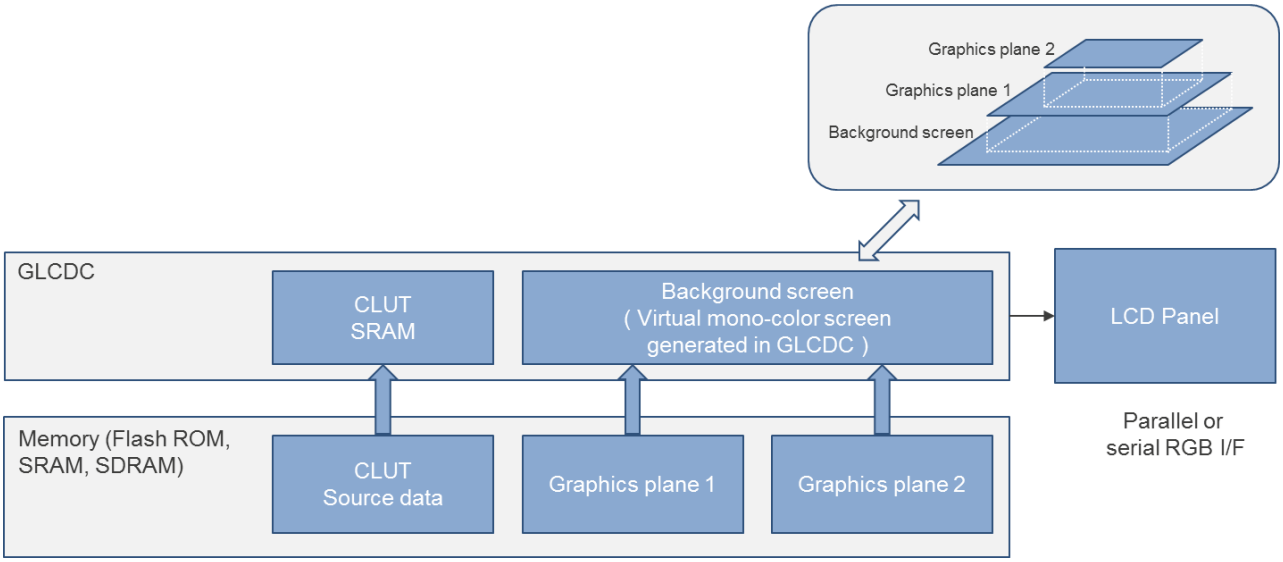


Figure 306: GLCDC Data Flow

The following figure shows a display system with a ping-pong frame buffer. It is recommended to have more than two frame buffers in a display system to avoid image tearing, which happens in a single frame buffer display system. In such designs, the GLCDC hardware can read a graphics frame image from one of the frame buffers while the image drawing engines (e.g. DRW and JPEG), CPU or DMAC/DTC transfer a graphics frame image to another frame buffer simultaneously. The module supports frame buffer toggling by the layerChange API at run-time.

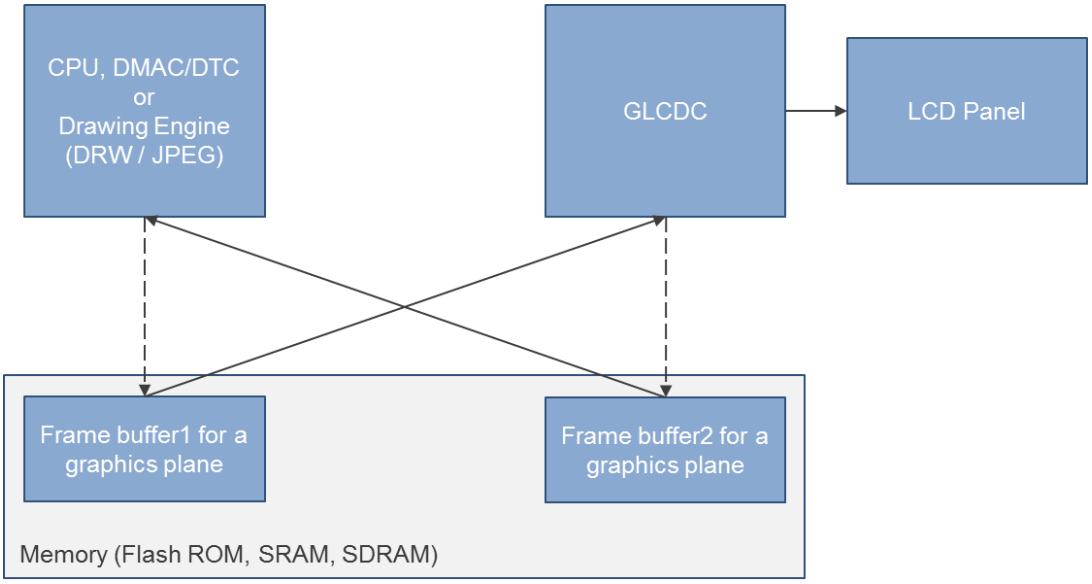


Figure 307: GLCDC Display- Typical ping-pong buffer system using GLCDC

Screen Format

The following figure shows the relationship between the LCD screen format and LCD timing parameters of the GLCDC module. The module has generic timing parameters for the LCD panel setting that support a variety of LCD panels.

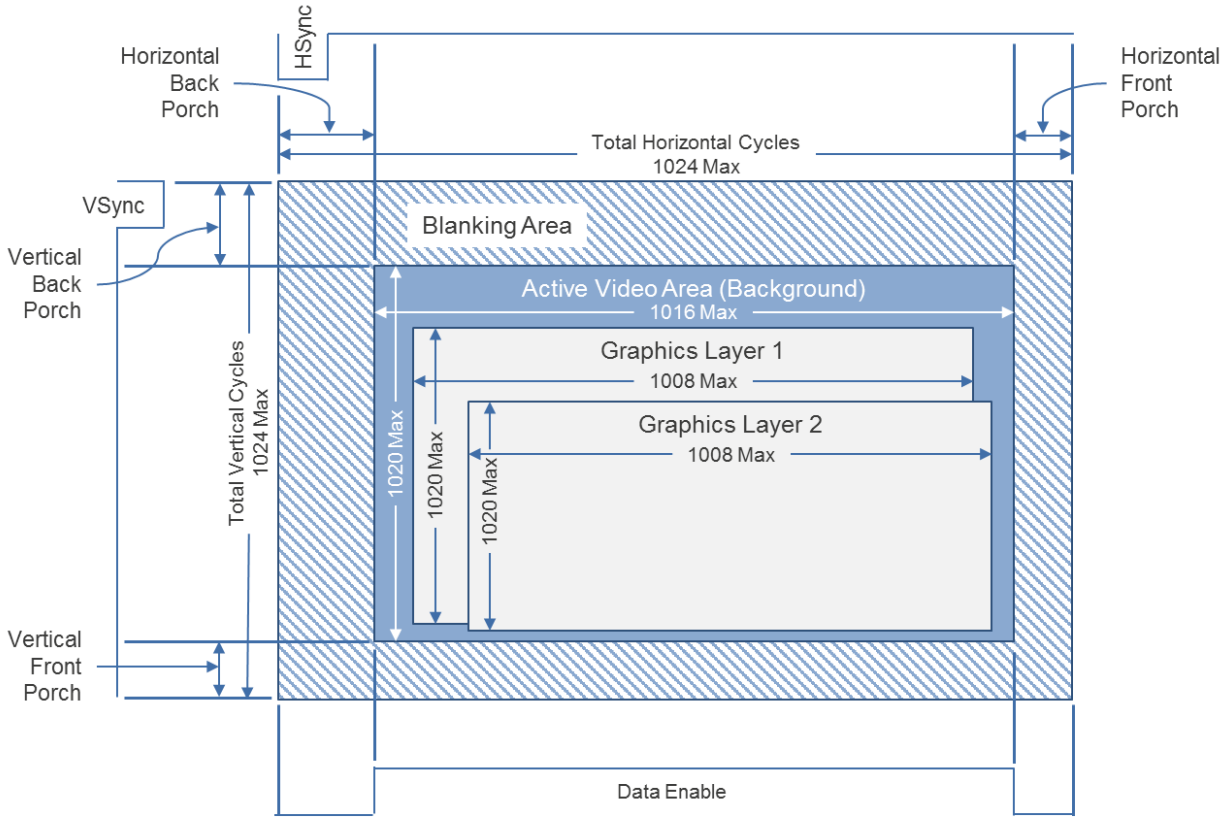


Figure 308: GLCDC Screen Format

Front Porch Period

The GLCDC module does not have a setting for horizontal/vertical front porch cycles/lines. Those cycles/lines must be included in the total horizontal cycles/vertical lines setting.

NOTE: The module requires setting the back porch cycles/lines based on the GLCDC hardware specification. Since typical LCD panels have a greater number of back porch cycles/lines than described, this is not a true limitation of the module.

- Number of the horizontal back porch cycles >= 3
- Number of the vertical back porch lines >= 2

Example Parameter Settings

PE-HMI1 v2.0 board (LXD Research & Display, LLC, M7504A):

The following example adjusts the horizontal total cycles, vertical total lines, and panel clock division ratio to generate an LCD panel refresh rate of 60 Hz. Regarding symbols for the LCD panel, see the M7504A data sheet.

LCD Panel Parameter Settings — PE-HMI1 v2.0 Board

ISDE Property	Setting
Panel clock source select	Internal clock
Graphics screen1/2 input horizontal size	800(thd)
Graphics screen1/2 input vertical size	480(tvd)
Graphics screen1/2 input horizontal stride	800(thd)
Horizontal total cycles	976(th)
Horizontal active video cycles	800(thd)
Horizontal back porch cycles	46(thp + thb)
Horizontal sync signal cycles	20(thp)
Horizontal sync signal polarity	Low active
Vertical total lines	512(tv)
Vertical active video lines	480(tvd)
Vertical back porch cycles	23(tvp + tvb)
Vertical sync signal lines	10(tvp)
Vertical sync signal polarity	Low active
Output Format	24bits RGB888
Data Enable Signal Polarity	High active
Sync edge	Rising edge
TCON - Hsync pin select	LCD_TCON0
TCON - Vsync pin select	LCD_TCON1
TCON - DataEnable pin select	LCD_TCON2
TCON - Panel clock division ratio	1/8

DK-S7G2 v3.0 board (LXD Research & Display, LLC, M7190A):

The following example adjusts the horizontal total cycles, vertical total lines, and panel clock division ration to generate an LCD panel refresh rate of 60 Hz. Regarding symbols for the LCD panel, see the M7190A data sheet.

LCD Panel Parameter Settings — DK-S7G2 v3.0 Board

ISDE Property	Setting
Panel clock source select	Internal clock
Graphics screen N input horizontal size	480(thd)
Graphics screen N input vertical size	272(tvd)
Graphics screen N input horizontal stride	480(thd)
Horizontal total cycles	582(th)
Horizontal active video cycles	480(thd)
Horizontal back porch cycles	43(thp + thb)
Horizontal sync signal cycles	41(thp)
Horizontal sync signal polarity	Low active
Vertical total lines	286(tv)
Vertical active video lines	272(tvd)
Vertical back porch cycles	12(tvp + tvb)
Vertical sync signal lines	10(tvp)
Vertical sync signal polarity	Low active
Output Format	16bits RGB565
Data Enable Signal Polarity	High active
Sync edge	Rising edge
TCON - Hsync pin select	LCD_TCON1
TCON - Vsync pin select	LCD_TCON2
TCON - DataEnable pin select	LCD_TCON0
TCON - Panel clock division ratio	1/24

SK-S7G2 v2.0 board (ILI Technology Corp., IL9341C):

The following example sets the horizontal total cycles and vertical total lines as large as allowed for the panel for an LCD panel refresh rate of about 76.8 Hz. Regarding symbols for the LCD panel, see the LIL9314V data sheet.

LCD Panel Parameter Settings — SK-S7G2 v2.0 Board

ISDE Property	Setting
Panel clock source select	Internal clock
Graphics screen N input horizontal size	256(See note)
Graphics screen N input vertical size	320(VAdr)
Graphics screen N input horizontal stride	256(See note)
Horizontal total cycles	290(HAdr + Hsync(16) + HBP(24) + HFP(16))
Horizontal active video cycles	240(HAdr)
Horizontal back porch cycles	40(Hsync(16) + HBP(24))
Horizontal sync signal cycles	16(Hsync)
Horizontal sync signal polarity	Low active
Vertical total lines	330(VAdr + Vsync(4) + VBP(2) + VFP(4))
Vertical active video lines	320(VAdr)
Vertical back porch cycles	6(Vsync + VBP)
Vertical sync signal lines	4(Vsync)
Vertical sync signal polarity	Low active
Output Format	16bits RGB565
Data Enable Signal Polarity	High active
Sync edge	Rising edge
TCON - Hsync pin select	LCD_TCON2
TCON - Vsync pin select	LCD_TCON1
TCON - DataEnable pin select	LCD_TCON0
TCON - Panel clock division ratio	1/32

NOTE: The input horizontal size and stride are intentionally set to 256 pixels, even though the parameter should be 240 pixels for the panel. This is because a horizontal line has to be 64-byte aligned for GLCDC hardware. Only 240 pixels from the beginning in a line are valid and the rest of the pixels in the line (16 pixels) don't care.

CLUT

The GLCDC module supports a Color Look-Up table that is used if the color format is ARGB1555, CLUT8, CLUT4 or CLUT1. The CLUT API can update CLUT0/CLUT1 SRAM (implemented inside the GLCDC hardware) for each of the graphics foreground or background screens.

NOTE: Make sure to call the CLUT API if you select a color format that uses the CLUT, before using the start API; otherwise, CLUT0 and CLUT1 become an unknown condition and the graphics do not display properly.

You can also call the CLUT API at run-time to update CLUT SRAM.

NOTE: The API copies the source of CLUT data to the CLUT SRAM, which is not currently used (each CLUT SRAM consists of a ping-pong buffer). After completing the CLUT data update, the API automatically switches the CLUT SRAM to be read by the GLCDC hardware from the next frame to avoid tearing of the image.

Line Repeating Mode

Line repeating is an important mode, especially for a system that does not have enough memory. In this mode, the GLCDC module reads a raster image, which has fewer pixels than the LCD panel screen size, and displays the raster repeatedly on the screen. The following figure shows an example of a screen image constructed by reading a small raster image repeatedly in the background graphics plane.

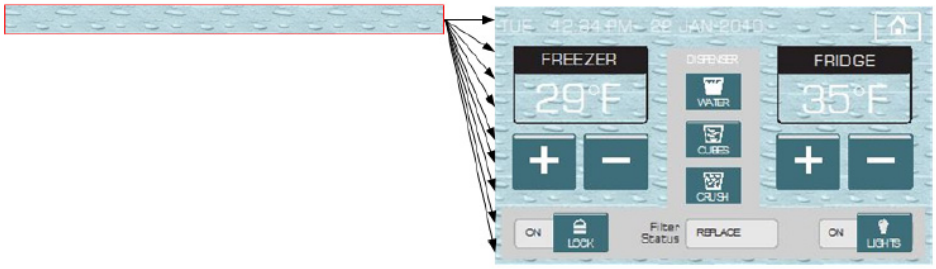


Figure 309: GLCDC Line Repeating Mode

NOTE: To enable this mode, set the GLCDC module property "Input - Graphics screen N input lines repeat" (where N = 1 or 2) to ON with the Synergy configurator. Also specify the repeat times to read a raster image to: "Input - Graphics screen N input lines repeat times". Specify the horizontal pixel size of the raster image in "Input - Graphics screen N input horizontal size" and "Input - Graphics screen N input horizontal stride," and then specify the vertical pixel size of the raster image in "Input - Graphics screen N input vertical size."

Gamma Correction

Gamma Correction is used to change the color characteristic of LCD panels to a flat characteristic. The following figure shows the gamma correction curve which can be configured by the GLCDC module. The module supports 16 threshold values for the input color level for each (R, G, B) color and defines the gain level for each of 16 areas divided by thresholds.

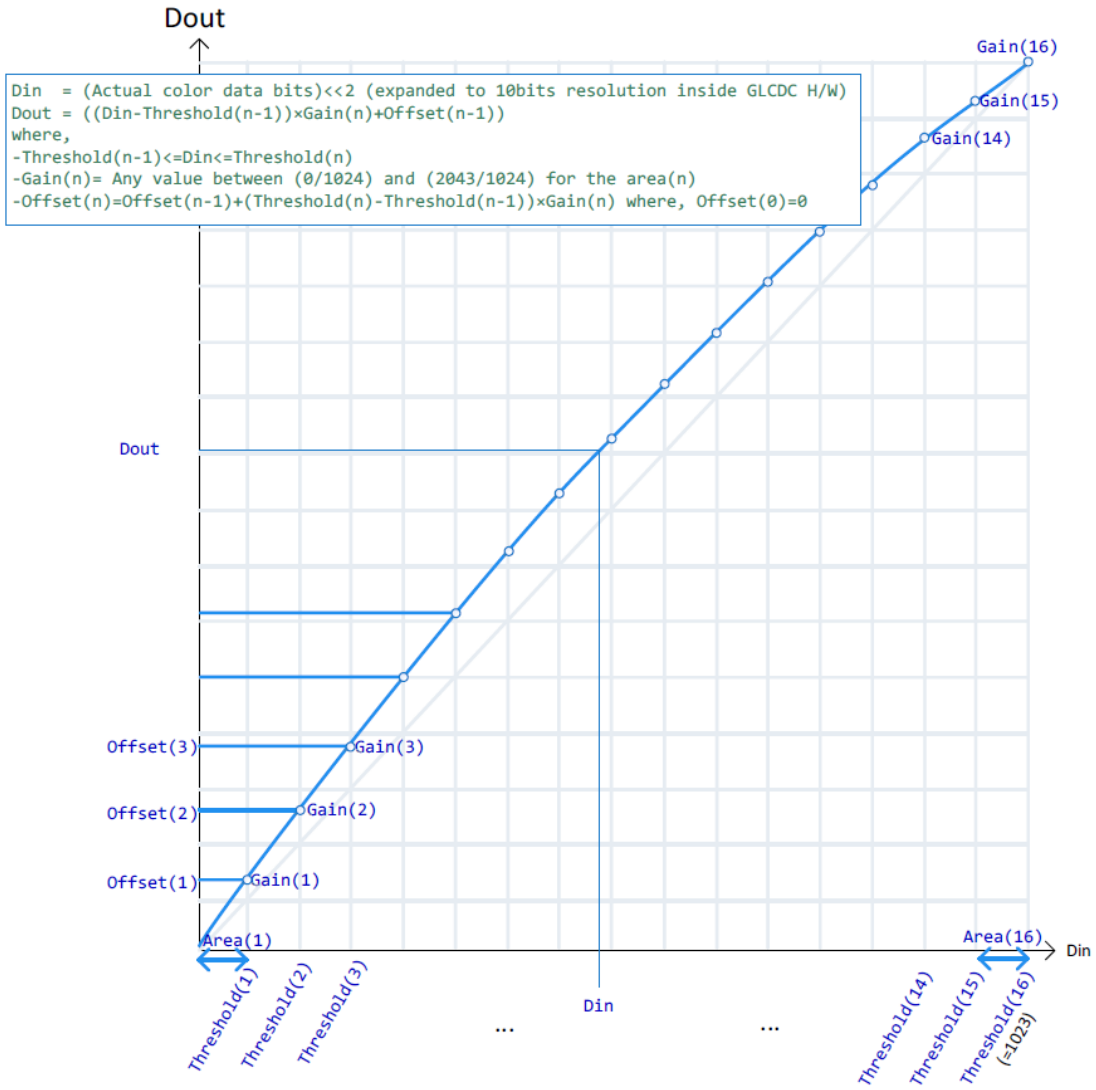


Figure 310: GLCDC Gamma Correction Curve

NOTE: To enable the gamma correction for each channel (R, G, B), set the GLCDC module property "Color correction – Gamma correction (R, G, B)" to ON using the Synergy configurator. Thresholds (total 16) are set to "Color correction – Gamma correction threshold (R, G, B) \[n\]" where, n=[0..15]. The gain value for each of areas are set to "Color correction – Gamma correction gain (R, G, B) \[n\]," where, n= [0..15].

GLCDC HAL Module Important Operational Notes and Limitations

You have the option to configure multiple GLCDC interrupts covered in the following sections.

Line Detection Interrupt

The line-detection interrupt is used to indicate when the GLCDC finishes outputting all lines to the LCD panel and goes into the blanking period. Use this interrupt to handle frame buffer switching in a graphics system and uses frame buffers with more than two frames.

Layer1 or Layer2 Line Buffer Underflow Interrupt

You can use the GLCDC layer1 or layer2 buffer underflow interrupt to detect a lack of memory bandwidth in your system. The buffer underflow occurs when the graphics data transfer from memory (for example, SDRAM or SRAM) to the GLCDC internal line buffer is blocked by the other data transfer, and not enough against the data transfer from GLCDC line buffer to the LCD panel interface. You have to design the graphics system to prevent this interrupt from occurring.

GLCDC Callbacks

A user-callback function can be registered in open. If a user-callback function is provided, the callback function is called from the interrupt service routine (ISR) each time an interrupt happens. The argument of the callback function event can take the following enumerated value listed in the table, so that a user can identify which event occurred in the graphics system. The DISPLAY_EVENT_LINE_DETECTION event is used for switching frame buffers to update the screen, and the DISPLAY_EVENT_GRn_UNDERFLOW event is used for error handling if an underflow occurs.

Event and Interrupt Summary

Name of Event	Name of Interrupt	Condition for the Event
DISPLAY_EVENT_LINE_DETECTION	Line detection	When GLCDC is done outputting the last line in the active video region
DISPLAY_EVENT_GR1_UNDERFLOW	Graphics 1 underflow	When GLCDC underflows during reading the data for graphics1 plane
DISPLAY_EVENT_GR2_UNDERFLOW	Graphics 2 underflow	When GLCDC underflows during reading the data for graphics2 plane

NOTE: Since the callback is called from an ISR, be careful not to use blocking calls or lengthy processing. Spending an excessive time in an ISR can affect the responsiveness of the system.

- The Display driver on r_GLCDC does not support RGB-index chroma key.
- The Display driver on r_GLCDC does not support the event-link function.
- Refer to the latest SSP Release Notes for any additional operational limitations applicable to this module.

4.2.13.4 Including the GLCDC HAL Module in an Application

This section describes how to include the GLCDC HAL module in an application using the SSP configurator.
NOTE: It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the GLCDC HAL module to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Display Driver is g_display0. This name can be changed in the associated Properties window.)

GLCDC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_display0 Display Driver on r_glcdc	Threads	New Stack> Driver> Graphics> Display Driver on r_glcdc

When the GLCDC HAL module on r_glcdc is added to the thread stack as shown in the following figure, the configurator automatically adds any required lower-level modules. Any drivers that need additional configuration information are box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

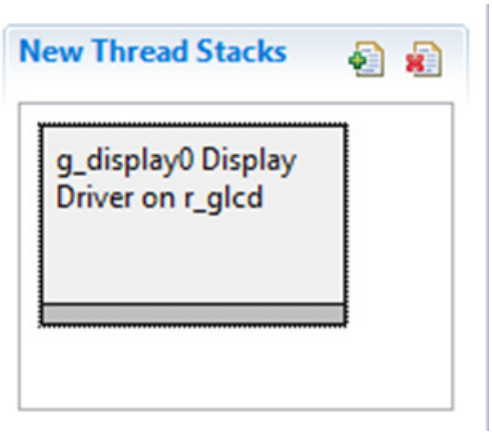


Figure 311: GLCDC HAL Module Stack

4.2.13.5 Configuring the GLCDC HAL Module

The GLCDC HAL module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this helps to orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the GLCDC HAL Module on r_glcdc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enable or disable the parameter checking.
Name	g_display0	The name to be used for a GLCDC module control block instance. This name is also used as the prefix of the other variable instances.
Name of display callback function to be defined by user	NULL	Name must be a valid C symbol.
Input – Panel clock source select	Internal clock(GLCDCLK), External clock(LCD_EXTCLK) (Default: Internal clock)	Choose the panel clock source depends on your system.
Input – Graphics screen1	Used, Not used (Default: Used)	Specify "Used" if the graphics screen N is used. Then the frame buffer named "display_fb_background" for graphics screen1 and "display_fb_foreground" for graphics screen2 is auto-generated by ISDE. If not using either of the graphics screens, specify "Not used." The frame buffer is then not created. Note that there is no memory read access to the frame buffer when you specify "Not used," which reduces the consumption of bus bandwidth.
Input – Graphics screen1 frame buffer name	fb_background	Custom name for frame buffer.
Input – Number of Graphics screen1 frame buffer	2	Graphics screen1 frame buffer number
Input – section where Graphics screen1 frame buffer allocated	sdram	Specify the section name to allocate the frame buffer. This is valid if "Input – Graphics screen1" is set as "Used."
Input – Graphics screen1 input horizontal size	800	Specify the number of horizontal pixels. Default value is the size for an image with 800x480 pixels.
Input – Graphics screen1 vertical size	480	Specify the number of vertical pixels. Default value is the size for an image with 800x480 pixels.

ISDE Property	Value	Description
Input – Graphics screen1 input horizontal stride (not bytes but pixels)	800	Specify the memory stride for a horizontal line. This value must be specified with the number of pixels, not actual bytes. Typically, this parameter is set to same number as parameter 'input horizontal size'. Default value is the size for an image with 800x480 pixels.
Input – Graphics screen1 input format	32 bits ARGB888, 32 bits RGB888, 16 bits RGB565, 16 bits ARGB1555, 16 bits ARGB4444, CLUT 8, CLUT 4, CLUT 1 (Default: 16 bits RGB565)	Specify the graphics screen Input format. If selecting CLUT formats, you must write CLUT data using clut before performing start. Default setting supports a RGB565 formatted image.
Input – Graphics screen1 input line descending	Used, Not used (Default: Not used)	Specify "On" if image data descends from the bottom line to the top line in the frame buffer. Usually "Off."
Input – Graphics screen1 input line repeat	On, Off (Default: Off)	Specify "On" if expecting to repeatedly read a raster image which is smaller than the LCD panel size. Usually "Off". For details, see the description of Line Repeating function.
Input – Graphics screen1 input line repeat times	0	Specify the number of repeating times for a raster image which is read repeatedly in a frame.
Input – Graphics screen1 layer coordinate X	0	Specify the horizontal offset in pixels of the graphics screen from the background screen.
Input – Graphics screen1 layer coordinate Y	0	Specify the vertical offset in pixels of the graphics screen from the background screen.
Input – Graphics screen1 layer background color alpha	255	Based on the alpha value, either the graphics screen2 (foreground graphics screen) is blended into the graphics screen1 (background graphics screen) or the graphics screen1 is blended into the monochrome background screen.
Input – Graphics screen1 layer background color Red	255	Specify the background color in the graphics screen N.

ISDE Property	Value	Description
Input – Graphics screen1 layer background color Green	255	Specify the background color in the graphics screen N.
Input – Graphics screen1 layer background color Blue	255	Specify the background color in the graphics screen N.
Input – Graphics screen1 layer fading control	None, Fade-in, Fade-out (Default: None)	Specify "On" when performing a fade-in for the graphics screen. The transparent screen changes gradually to opaque. Specify "Off" when performing the fade-out for the graphics screen. The opaque screen changes gradually to transparent. Note that this processing is accelerated by the GLCDC hardware and cannot stop once started. The transition status can be monitored by statusGet.
Input – Graphics screen1 layer fade speed	0	Specify the number of frames for the fading transition to complete.
Input – Graphics screen2	Used, Not used (Default: Not used)	Specify "Used" if the graphics screen N is used. Then the frame buffer named "display_fb_background" for graphics screen1 and "display_fb_foreground" for graphics screen2 is auto-generated by ISDE. If not using either of the graphics screens, specify "Not used." Then, the frame buffer is not created. Note that there is no memory read access to the frame buffer when you specify "Not used", which reduces the consumption of bus bandwidth.
Input – Graphics screen2 frame buffer name	fb_foreground	Custom name for frame buffer.
Input – Number of Graphics screen2 frame buffer	2	Graphics screen2 frame buffer n umber
Input – section where Graphics screen2 frame buffer allocated	sdram	Specify the section name to allocate the frame buffer. This is valid if "Input – Graphics screen1" is set as "Used."
Input – Graphics screen2 input horizontal size	800	Specify the number of horizontal pixels. Default value is the size for an image with 800x480 pixels.

ISDE Property	Value	Description
Input – Graphics screen2 vertical size	480	Specify the number of vertical pixels. Default value is the size for an image with 800x480 pixels.
Input – Graphics screen2 input horizontal stride (not bytes but pixels)	800	Specify the memory stride for a horizontal line. This value must be specified with the number of pixels, not actual bytes. Typically, this parameter is set to same number as parameter 'input horizontal size'. Default value is the size for an image with 800x480 pixels.
Input – Graphics screen2 input format	32 bits ARGB888, 32 bits RGB888, 16 bits RGB565, 16 bits ARGB1555, 16 bits ARGB4444, CLUT 8, CLUT 4, CLUT 1 (Default: 16 bits RGB565)	Specify the graphics screen Input format. If selecting CLUT formats, you must write CLUT data using clut before performing start. Default setting supports a RGB565 formatted image.
Input – Graphics screen2 input line descending	On, Off (Default: Off)	Specify "On" if image data descends from the bottom line to the top line in the frame buffer. Usually "Off."
Input – Graphics screen2 input line repeat	On, Off (Default: Off)	Specify "On" if expecting to repeatedly read a raster image, which is smaller than the LCD panel size. Usually "Off." For details, see the description of Line Repeating function.
Input – Graphics screen2 input line repeat times	0	Specify the number of repeating times for a raster image which is read repeatedly in a frame.
Input – Graphics screen2 layer coordinate X	0	Specify the horizontal offset in pixels of the graphics screen from the background screen.
Input – Graphics screen2 layer coordinate Y	0	Specify the vertical offset in pixels of the graphics screen from the background screen.

ISDE Property	Value	Description
Input – Graphics screen2 layer background color alpha	255	Based on the alpha value, either the graphics screen2 (foreground graphics screen) is blended into the graphics screen1 (background graphics screen) or the graphics screen1 is blended into the monochrome background screen.
Input – Graphics screen2 layer background color Red	255	Specify the background color in the graphics screen N.
Input – Graphics screen2 layer background color Green	255	Specify the background color in the graphics screen N.
Input – Graphics screen2 layer background color Blue	255	Specify the background color in the graphics screen N.
Input – Graphics screen2 layer fading control	None, Fade-in, Fade-out (Default: None)	Specify "On" when performing a fade-in for the graphics screen. The transparent screen changes gradually to opaque. Specify "Off" when performing the fade-out for the graphics screen. The opaque screen changes gradually to transparent. Note that this processing is accelerated by the GLCDC hardware and cannot stop once started. The transition status can be monitored by statusGet.
Input – Graphics screen2 layer fade speed	0	Specify the number of frames for the fading transition to complete.
Output – Horizontal total cycles	1024	Specify the total cycles in a horizontal line. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output – Horizontal active video cycles	800	Specify the number of active video cycles in a horizontal line. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.

ISDE Property	Value	Description
Output – Horizontal back porch cycles	46	Specify the number of back porch cycles in a horizontal line. Back porch starts from the beginning of Hsync cycles, which means back porch cycles contain Hsync cycles. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output – Horizontal sync signal cycles	20	Specify the number of Hsync signal assertion cycles. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches LCD panel on S7G2 PE-HMI1 board.
Output – Horizontal sync signal polarity	Low active, High active (Default: Low active)	Select the polarity of Hsync signal to match your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output – Vertical total lines	525	Specify number of total lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output – Vertical active video lines	480	Specify the number of active video lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output – Vertical back porch lines	23	Specify the number of back porch lines in a frame. Back porch starts from the beginning of Vsync lines, which means back porch lines contain Vsync lines. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.

ISDE Property	Value	Description
Output – Vertical sync signal lines	10	Specify the Vsync signal assertion lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output – Vertical sync signal polarity	Low active, High active (Default: Low active)	Select the polarity of Vsync signal to match to your system. Default setting matches LCD panel on S7G2 PE-HMI1 board.
Output – Format	24bits RGB888, 18bits RGB666, 16 bits RGB565, 8bits serial (Default: 24bits RGB888)	Specify the graphics screen output format to match to your LCD panel. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output – Endian	Little endian, Big endian (Default: Little endian)	Select data endian for output signal to LCD panel. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output – Color order	RGB, BGR (Default: RGB)	Select data order for output signal to LCD panel. The order of blue and red can be swapped if needed. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output – Data Enable Signal Polarity	Low active, High active (Default: High active)	Select the polarity of Data Enable signal to match to your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output – Sync edge	Rising Edge, Falling Edge (Default: Rising Edge)	Select the polarity of Sync signals to match to your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output – Background color alpha channel	255	Specify the background color of the background screens.
Output – Background color R channel	0	Specify the background color of the background screens.
Output – Background color G channel	0	Specify the background color of the background screens.
Output – Background color B channel	0	Specify the background color of the background screens.

ISDE Property	Value	Description
CLUT	Used, Not used (Default: Not used)	Specify "Used" if selecting CLUT formats for a graphics screen input format. Then, a buffer named "CLUT_buffer" for the CLUT source data is generated in the ISDE auto-generated source file.
CLUT - CLUT buffer size	256	Specify the number of entries for the CLUT source data buffer. Each entries consumes 4 bytes (1 word). Words of CLUT source data specified by this parameter are generated in the ISDE auto-generated source file.
TCON – Hsync pin select	Not used, LCD_TCON0, LCD_TCON1, LCD_TCON2, LCD_TCON3 (Default: LCD_TCON0)	Select the TCON pin used for the Hsync signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON – Vsync pin select	Not used, LCD_TCON0, LCD_TCON1, LCD_TCON2, LCD_TCON3 (Default: LCD_TCON1)	Select TCON pin used for Vsync signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON – DataEnable pin select	Not used, LCD_TCON0, LCD_TCON1, LCD_TCON2, LCD_TCON3 (Default: LCD_TCON2)	Select TCON pin used for DataEnable signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON – Panel clock division ratio	8-Jan	Select the clock source divider value. See the note at bottom of this table about the source clock for the pixel clock.
Color correction – Brightness	Off, On (Default: Off)	Specify "On" when performing brightness control. If specifying "Off", the setting below does not affect the output color.
Color correction – Brightness R channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each R, G, and B channel.
Color correction – Brightness G channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each R, G, and B channel.

ISDE Property	Value	Description
Color correction – Brightness B channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each R, G, and B channel.
Color correction – Contrast	Off	Specify "On" when performing contrast control. If specifying "Off", the setting below does not affect the output color.
Color correction – Contrast(gain) R channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set the value for each R, G, and B channel.
Color correction – Contrast(gain) G channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set value for each R, G, and B channel.
Color correction – Contrast(gain) B channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set the value for each R, G, and B channel.
Color correction – Gamma correction(Red)	Off, On (Default: Off)	Control for each channel R/G/B. Specify "On" when performing gamma correction for the red channel. If specifying "Off," the settings for gain and threshold do not affect the output color.
Color correction – Gamma gain R[0-15]	0	Set the gain value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data, with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as: Output color level = Input color level / 1024 (/128).

ISDE Property	Value	Description
Color correction – Gamma threshold R[0-15]	0	Set the threshold value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as: Output color level = Input color level / 1024 (/128).
Color correction – Gamma correction(Green)	Off	Control for each channel R/G/B. Specify "On" when performing gamma correction for the red channel. If specifying "Off," the settings for gain and threshold do not affect the output color.
Color correction – Gamma gain G[0-15]	0	Set the gain value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data, with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as: Output color level = Input color level / 1024 (/128).
Color correction – Gamma threshold G[0-15]	0	Set the threshold value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as: Output color level = Input color level / 1024 (/128).
Color correction – Gamma correction(Blue)	Off, On (Default: Off)	Control for each channel R/G/B. Specify "On" when performing gamma correction for the red channel. If specifying "Off", the settings for gain and threshold do not affect the output color.

ISDE Property	Value	Description
Color correction – Gamma gain B[0-15]	0	Set the gain value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction – Gamma threshold B[0-15]	0	Set the threshold value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as: Output color level = Input color level / 1024 (/128).
Dithering	Off, On (Default: Off)	Dithering enable. Specify "On" when applying the dither effect to reduce the banding in case of selecting RGB666 or RGB565 output formats. Dithering can be applied when converting. If specified "Off", the settings for dithering below do not affect the output. For details on the dither effect, see Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering – Mode	Truncate, Round off, 2x2 Pattern (Default: Truncate)	Specify the dither mode. For detail, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering – Pattern A	Pattern 00, Pattern 01, Pattern 10, Pattern 11 (Default: Pattern 11)	Specify the dither pattern for 2X2 pattern mode. For details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.

ISDE Property	Value	Description
Dithering – Pattern B	Pattern 00, Pattern 01, Pattern 10, Pattern 11 (Default: Pattern 11)	Specify the dither pattern for 2X2 pattern mode. For details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering – Pattern C	Pattern 00, Pattern 01, Pattern 10, Pattern 11 (Default: Pattern 11)	Specify the dither pattern for 2X2 pattern mode. For details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering – Pattern D	Pattern 00, Pattern 01, Pattern 10, Pattern 11 (Default: Pattern 11)	Specify the dither pattern for 2X2 pattern mode. For details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Misc – Correction Process Order	Brightness and Contrast then Gamma, Gamma then Brightness and Contrast (Default: Brightness and Contrast then Gamma)	Specify the color correction processing order if needed.
Line Detect Interrupt Priority	Priority 0(highest)-15(lowest), Disabled (Default: Disabled)	Line detect interrupt priority selection.
Underflow 1 Interrupt Priority	Priority 0(highest)-15(lowest), Disabled (Default: Disabled)	Underflow 1 interrupt priority selection.

| Underflow 2 Interrupt Priority | Priority 0(highest)-15(lowest), Disabled (Default: Disabled) | Underflow 2 interrupt priority selection. | |

NOTE: The example values and defaults are from a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and different configuration settings available.

GLCDC HAL Module Clock Configuration

The GLCDC module can generate the pixel clock from either of the following clock sources. The source clock selection is available through Synergy Configuration in e^2 studio.

- Internal clock source (PLLOUT; 240 MHz)
- External clock source from the LCD_EXTCLK pin

NOTE: The internal clock is different in S7G2 WS1 (Working Sample1) chip and the WS2 (Working Sample2) chip or later. WS1 chip uses PCLKB (max. 60 MHz), but WS2 or later chips use PLLOUT (max. 240 MHz).

GLCDC HAL Module Pin Configuration

The GLCDC module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The pin selection table lists methods to select pins within the SSP configuration window and the configuration settings table lists an example depicting selection of GLCDC pins.

Pin Selection Sequence for the GLCDC HAL Module

Resource	ISDE Tab	Pin selection Sequence
GLCDC	Pins	Select Peripherals > Graphics: GLCDC> GLCDC0

NOTE: The selection sequence assumes GLCDC0 is the desired hardware target for the driver.

Pin Configuration Settings for the GLCDC HAL Module

Property	Value	Description
Pin Group Selection	Mixed, _A Only, _B Only (Default: Mixed)	Pin group selection
Operation Mode	Disabled, Custom, RGB888, RGB666, RGB565 (Default: Disabled)	Select desired operation mode
LCD_CLK	None, P900, P101 (Default: None)	LCD_CLK Pin
LCD_DATA00:15	None, Pn, Pm (Default: None)	LCD_DATA Pins
LCD_TCON0:3	None, Pn, Pm (Default: None)	LCD_TCON Pins
LCD_EXTCLK	None, Pn, Pm (Default: None)	LCD_EXTCLK Pin

NOTE: The example values in the table are from a project using the Synergy S7G2 and SK-S7G2 Kits. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

NOTE: To use the GLCDC module on the S7G2 PE-HMI1 board, be sure to set up PORT10 pin3 (PA03) and pin5 (PA05) as IOPORT pins with the output level HIGH. Pin PA03 controls the DISP signal (Display on/off) and Pin PA05 controls the backlight of LCD panel. For details, see the schematics of S7G2 PE-HMI1 board.

4.2.13.6 Using the GLCD HAL Module in an Application

The typical steps in using the GLCDC HAL module in an application are:

1) Initialize the GLCDC HAL module with the open API.
2) Draw a primary image in the frame buffer with application code.
3) Start the image displaying using the start API.
4) Draw a new image in the frame buffer to update the display with application code. Typical user systems consist of the ping-pong frame buffer system, so draw the image to another frame buffer which is not used for displaying at the point.
5) Request the frame buffer toggling to GLCDC hardware with the layerChange API.
6) The GLCDC hardware toggles frame buffer and display a new image from the next frame.

To synchronize application code to the completion of drawing current frame buffer, use the line-detection interrupt and notify the timing to application code through the GLCDC callback.

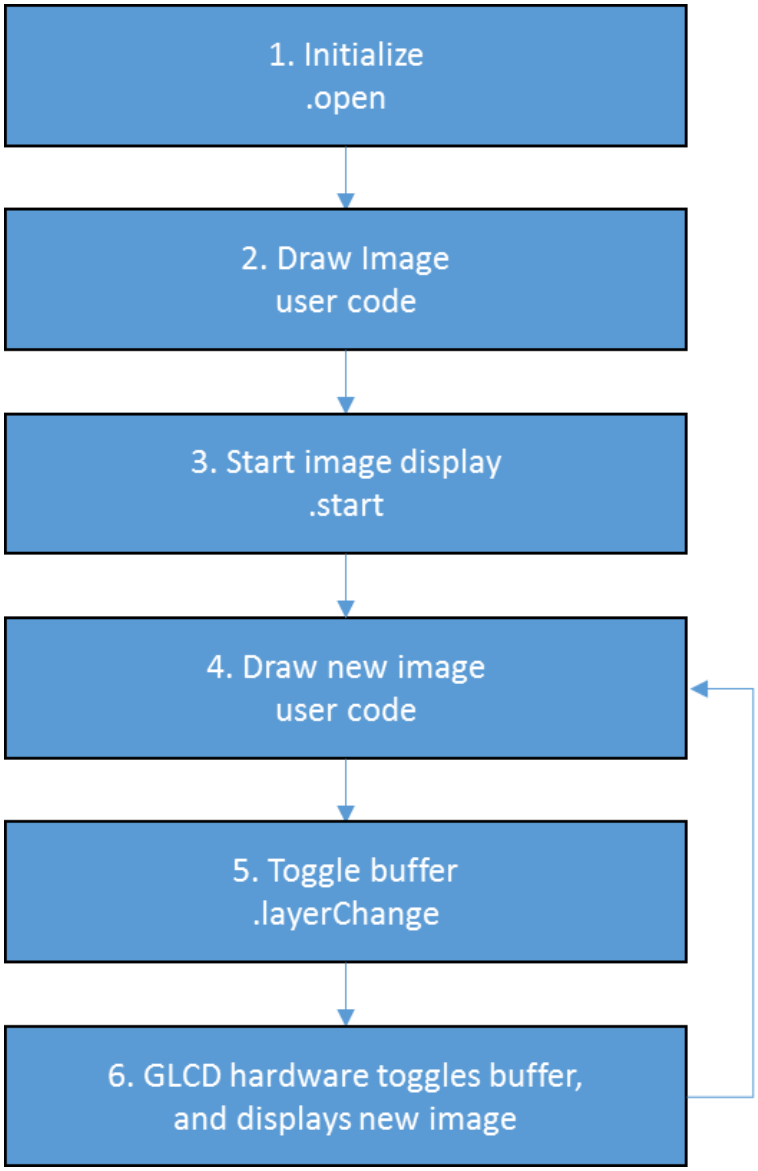


Figure 312: Flow Diagram of a Typical GLCDC HAL Module Application

4.2.14 Data Operation Circuit Driver

The Data Operation Circuit (DOC) HAL module provides high-level APIs for DOC applications and is implemented on r_doc. The DOC HAL module uses the DOC peripherals on the Renesas Synergy™ MCU device. A user-defined callback can be created to inform the CPU when an event occurs.

4.2.14.1 DOC HAL Module Features

The DOC HAL module peripheral is used to compare 16-bit data and can detect the following events:

- A mismatch or match between data values
- Overflow of an addition operation
- Underflow of a subtraction operation

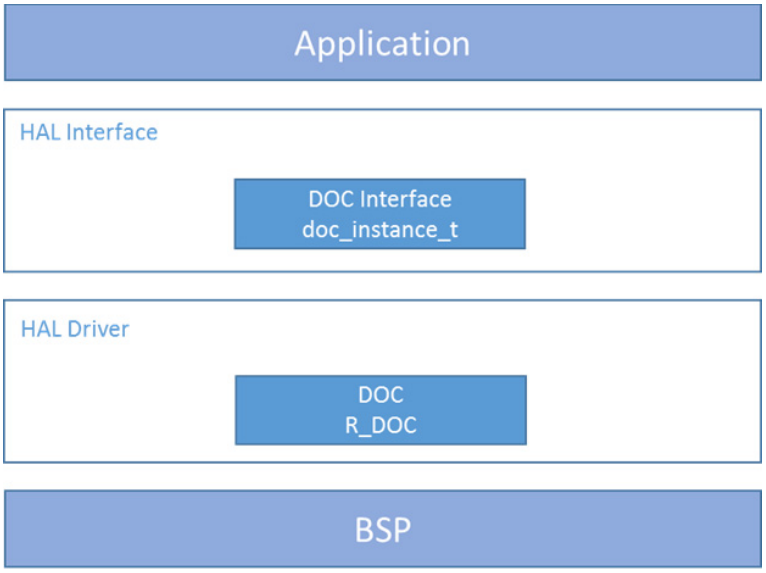


Figure 313: DOC HAL Module Block Diagram

4.2.14.2 DOC HAL Module APIs Overview

The DOC HAL module defines APIs for opening, closing, checking the status of, and writing data to the data operation circuit. The DOC HAL module uses the DOC peripheral on the Synergy MCU. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

DOC HAL Module API Summary

Function Name	Example API Call and Description
open	g_doc.p_api->open(g_doc.p_ctrl, g_doc.p_cfg) Initial configuration.
close	g_doc.p_api->close(g_doc.p_ctrl) Allow the driver to be reconfigured. Will reduce power consumption.

Function Name	Example API Call and Description
statusGet	g_doc.p_api->statusGet(g_doc.p_ctrl, &my_Status) Get the DOC status and stores it in the provided pointer p_status.
statusClear	g_doc.p_api->statusClear(g_doc.p_ctrl) Clear DOPCF status flag.
write	g_doc.p_api->write(g_doc.p_ctrl, &value) Write to the DODIR and DODSR registers.
inputRegisterWrite	g_doc.p_api->inputRegisterWrite(g_doc.p_ctrl, doc_values) Write to the DODIR register.
versionGet	g_doc.p_api->versionGet(g_doc.p_ctrl, &version) Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	DOC successfully configured.
SSP_ERR_IN_USE	Module already open.
SSP_ERR_ASSERTION	One or more pointers point to NULL.
SSP_ERR_INVALID_ARGUMENT	ISR is not enabled. Enable the ISR in bsp_irq_cfg.h.
SSP_ERR_HW_LOCKED	DOC resource is locked.
SSP_ERR_NOT_OPEN	Driver not open.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.14.3 DOC HAL Module Operational Overview

The DOC HAL module controls the DOC peripheral on a Synergy MCU. It is used to compare 16-bit data and can detect a mismatch between data values, an overflow of an addition value, or an underflow of a subtraction operation. If a callback is available and the associated interrupt is enabled, the callback function will be called in response to a DOC event.

The DOC uses two data registers to perform operations: the DOC Data Input Register (DOCDIR) holds the data to be operated on and the DOC Data Setting Register (DOCDSR) holds the value that is used to operate on the input data. In addition and subtraction modes, this register stores the results of data operations. (Both these registers are 16-bits wide.)

DOC HAL Module Important Operational Notes and Limitations

The initial setting of comparison data is written to the DOC by calling the write API. The write API writes to the DOC DODSR and DODIR registers. The write API uses a variable of type doc_data_t.

```
doc_data_t g_doc_values;

   g_doc_values.dodir = 0x1000;

   g_doc_values.dodsr = 0x1000;

   g_doc.p_api->write(g_doc.p_ctrl, &g_doc_values)
```

If the data to be compared does not change, there is no need to re-write it each time a comparison is required. The input data value can be written to the DOC by using the inputRegisterWrite API. The inputRegisterWrite API writes only to the DOC data-input register.

Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.14.4 Including the DOC HAL Module in an Application

This section describes how to include the DOC HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the DOC Driver to an application, simply add it to a thread using the stacks selection sequence provided in the following table. (The default name for the DOC Driver is g_doc0. This name can be changed in the associated Properties window.)

DOC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_doc0 Data Operation Circuit Driver on r_doc	Threads	New Stack > Driver > Monitoring > Data Operation Circuit Driver on r_doc

When the DOC HAL module on r_doc is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information will be box text highlighted in Red.

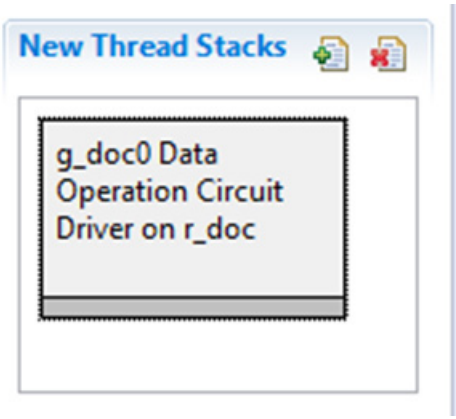


Figure 314: DOC HAL Module Stack

4.2.14.5 Configuring the DOC HAL Module

The DOC HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE indicates the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the following configuration table settings. This helps to orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the DOC HAL Module on r_doc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enable or disable the parameter error checking.
Name	g_doc0	Module name.

ISDE Property	Value	Description
Event	Comparison mismatch, Comparison match, Addition overflow, Subtraction underflow (Default: Comparison mismatch)	Specify the event which will trigger the DOC interrupt.
Callback	NULL	A user callback function can be defined here. If this callback function is provided, it is called from the interrupt service routine (ISR) when the configured DOC event occurs. NOTE: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
DOC Interrupt Priority	Priority 0 (highest), 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 (lowest, not valid if using Thread X), Disabled (Default: Disabled)	Use the pull down to set the DOC interrupt priority.

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults can be desirable. For example, it might be useful to select different events depending on the operation desired.

DOC HAL Module Clock Configuration

The DOC HAL module does not require any specific clock configuration.

DOC HAL Module Pin Configuration

The DOC HAL module does not require and specific pin configurations.

4.2.14.6 Using the DOC HAL Module in an Application

The typical steps in using the DOC HAL module in an application are:

1) Initialize the DOC using the open API.
2) Set register values in DODIR and DODSR using the write API.
3) Steam data to the DOC using the inputRegisterWrite API.

4) Read the status of the comparison using the statusGet API or in the callback if enabled.
5) Clear status flags using the statusClear API.
6) Close the module using the close API.

The following figure illustrates these common steps in a typical operational flow diagram.

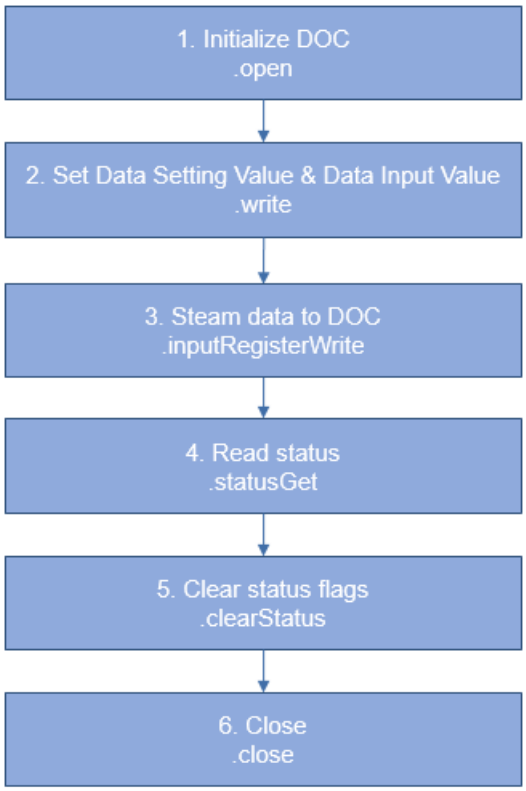


Figure 315: Flow Diagram of a Typical DOC HAL Module Application

4.2.15 DMAC Driver

4.2.15.1 DMAC Driver

The Direct Memory Access Controller or DMAC HAL module provcides high-level APIs for data-transfer applications and is implemented on r_dmac. The DMAC HAL module uses the DMAC peripheral on the Synergy MCU. A user-defined callback can be created to inform the CPU when transfer events occur.

4.2.15.2 DMAC HAL Module Features

The DMAC HAL module moves data from a user-specified source to a user-specified destination when an interrupt or event occurs. The DMAC HAL module supports the following:

- DMAC module on a Synergy MCU

- Interrupts, if desired
- Multiple transfer modes
  - Single Transfer
  - Repeat Transfer
  - Block Transfer
  - Address increment or fixed modes
- Multiple channels, with the number depending on the MCU used

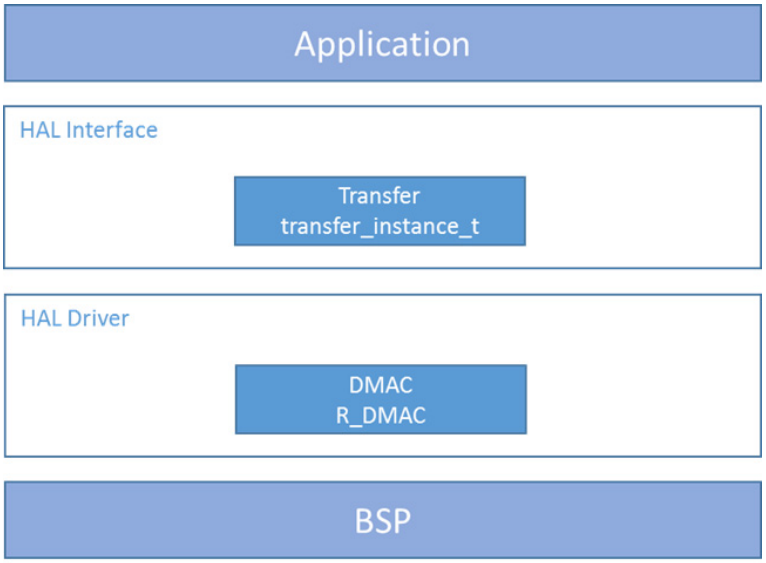


Figure 316: DMAC HAL Module Block Diagram

4.2.15.3 DMAC HAL Module APIs Overview

The DMAC HAL module defines APIs for opening, closing, starting, and stopping timers. Note that the Data Transfer Controller (DTC) and the DMAC use the same transfer interface; sharing an interface makes it easier to change between DTC and DMA implementations. The API calls are the same independent of the lower level implementations. A complete list of the available APIs, an example API call, and a short description of each function can be found in the following API summary table. A table of status return values follows the API summary.

DMAC HAL Module API Summary

Function Name	Example API Call and Description
open	g_transfer0.api->open(g_transfer0.p_ctrl, g_transfer0.p_cfg) Open device channel. Initialize driver and hardware on first call.
close	g_transfer0.api->close(g_transfer0.p_ctrl) Close device channel. Turns off hardware if last channel open.
reset	g_transfer0.api->reset(g_transfer0.p_ctrl, &source, &destination, number_of_transfers) Reset channel settings.
start	g_transfer0.api->start(g_transfer0.p_ctrl, mode) Start data transfer.
stop	g_transfer0.api->stop(g_transfer0.p_ctrl) Stop data transfer.
enable	g_transfer0.api->enable(g_transfer0.p_ctrl) Enable channel.
disable	g_transfer0.api->disable(g_transfer0.p_ctrl) Disable channel.
versionGet	g_transfer0.api->versionGet(&version) Retrieve the API version with the version pointer.
infoGet	g_transfer0.api->infoGet(g_transfer0.p_ctrl, &info) Get transfer channel info.

Function Name	Example API Call and Description
blockReset	g_transfer0.api->blockReset(g_transfer0.p_ctrl, &source, &destination, length, size, number_of_transfers) Reset Block Transfer parameters.

NOTE: Review the *SSP User's Manual* API References for the associated module, where there are detail descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_ASSERTION	Parameter has invalid value.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_UNSUPPORTED	Operation not configured correctly.
SSP_ERR_IN_USE	The channel specified has already been opened. No configurations were changed. Call the associated Close function or use associated Control commands to reconfigure the channel.
SSP_ERR_IRQ_BSP_DISABLED	IRQ not enabled in BSP.
SSP_ERR_NOT_ENABLED	Operation failed.
SSP_ERR_NOT_OPEN	The channel is not opened.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual* API References for the associated module to view definitions of all relevant status return values.

4.2.15.4 DMAC HAL Module Operational Overview

The DMAC HAL module moves data from a user-specified source to a user-specified destination when an interrupt or event occurs. The DMAC HAL module uses DMAC peripheral registers, so the number of transfers in the system is limited to the number of DMAC channels on the device. The activation source does not have to be enabled to use the DMAC. When the DMAC transfer completes, a DMAC interrupt is called. If the activation source interrupt is enabled, it fires at the same time the transfer is triggered. If the DMAC interrupt is enabled, it fires after all transfers are complete.

For example, if a normal mode transfer, with a length of 16 is triggered by a timer, the timer interrupt fires at the same time each transfer occurs and the DMAC interrupt fires after the 16th transfer completes. The DMAC does not support chained transfers.

DMAC HAL Module Important Operational Notes and Limitations

DMAC HAL Module Operational Notes

Normal Mode

In normal mode, a single transfer triggers each time an activation source event occurs. A single transfer is 1 byte, 2 bytes, or 4 bytes, depending on the setting selected in the size parameter. Each time a transfer occurs, the transfer length decrements by 1. When the transfer length reaches 0, the transfer is complete.

Repeat Mode

In repeat mode, a single transfer triggers each time an activation source event occurs. A single transfer is 1 byte, 2 bytes, or 4 bytes, depending on the setting selected in the size parameter. Each time a transfer occurs, the transfer length decrements by 1. When transfer length reaches 0, the transfer length reloads with its initial value and the transfer restarts. If the repeat area is set to source, the source register also reloads with its initial value when the transfer restarts. Alternatively, if the repeat area is set to destination, the destination register reloads with its initial value when the transfer restarts.

Block Mode

In block mode, the entire transfer length transfers each time an activation source event occurs. For example, if a transfer is configured in the block mode with a timer as the activation source, a 2-byte size and 12-byte length, 24 bytes transfer each time the activation source event occurs. Each time a transfer occurs, the transfer length decrements by 1. When the length reaches 0, the transfer length is reloaded with its initial value and the transfer restarts. If the repeat area is set to source, the source register is also reloaded with its initial value when the transfer restarts. Alternatively, if the repeat area is set to destination, the destination register reloads with its initial value when the transfer restarts.

Address Mode

After each transfer of size (1 byte, 2 bytes, or 4 bytes), the source pointer and destination pointer adjust by src_addr_mode and dest_addr_mode, respectively.

For example, if src_addr_mode is set to:

TRANSFER_ADDR_MODE_INCREMENTED, and size is set to TRANSFER_SIZE_4_BYTES, the p_dest pointer is incremented by 4 (the transfer size) after each transfer.

The pointer does not change if set to TRANSFER_ADDR_MODE_FIXED.

DMAC HAL Module Limitations

Refer to the latest *SSP Release Notes* for any additional operational limitations for this module.

4.2.15.5 Including the DMAC HAL Module in an Application

This section describes how to include the DMAC Driver in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the Starting Development section in the *SSP User's Manual* to learn how to manage each of these important steps in creating SSP based applications.

To add the DMAC Driver to an application, simply add it to a thread using the **Stacks Selection Sequence** provided in the table below. (The default name for the DMAC Driver is g_transfer0. This name can be changed in the associated **Properties** window.)

Table 3 GPT Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_ transfer0 Transfer Driver on r_dmac	Threads	New Stack > Driver > Transfer> Transfer Driver on r_dmac

When the DMAC Driver on r_dmac is added to the **New Thread Stack** as shown in the figure below, the configurator automatically adds any needed lower level drivers. Any drivers that need additional configuration information are box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

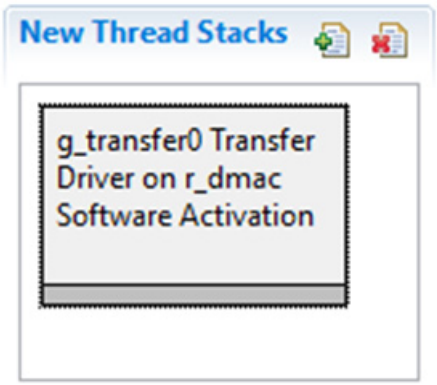


Figure 317: DMAC HAL Module Stack

4.2.15.6 Configuring the DMAC HAL Module

The DMAC HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections (such as interrupts or operating modes) which must be configured for lower level modules for successful operation. Only those properties that can be changed without causing conflicts are available for modification. Other properties are **locked** and are not available for changes, and are identified with a lock icon for the **locked property** in the property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the **Properties tab** within the SSP Configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority. This configuration setting is available within the properties window of the associated module. Simply select the indicated module and then view the properties window. The interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Note that the interrupt priorities listed in the properties window in the ISDE indicate the validity of the setting based on the targeted MCU (CM4 or CM0+.) This level of detail is not included in the configuration properties following tables, but is easily visible within the ISDE when configuring interrupt priority levels.

NOTE: **Property Settings** in parallel with looking over the following configuration settings table. The configuration settings in the table can help orient you and provide a useful hands-on approach to learning the ins and outs of developing with the SSP.

Table 4 Configuration Settings for the DMAC HAL Module on r_dmac

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Selects if code for parameter checking is to be included in the build
Name	g_transfer0	Module name.
Channel	0	
Mode	Block	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Software Activation	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection

| Interrupt Priority | Priority 0 (highest), 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 (lowest, not valid if using Thread X), Disabled (Default: Disabled) | ||

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

DMAC HAL Module Clock Configuration

The DMAC peripheral module use ICLK as the clock source. The ICLK frequency is set by using the SSP configurator clock tab, prior to a build, or by using the CGC Interface at run-time.

DMAC HAL Module Pin Configuration

The DMAC HAL Module is not associated with any pins.

4.2.15.7 Using the DMAC HAL Module in an Application

The typical steps in using the DMAC HAL module in an application are:

1) Initialize the DMAC HAL module using the open API.
2) Enable the DMAC HAL module using the enable API (if not auto enabled).
3) Manage transfers using other APIs as needed.
4) Close the DMAC HAL module when needed.

The following operational flow diagram shows common steps in using the DMAC driver.

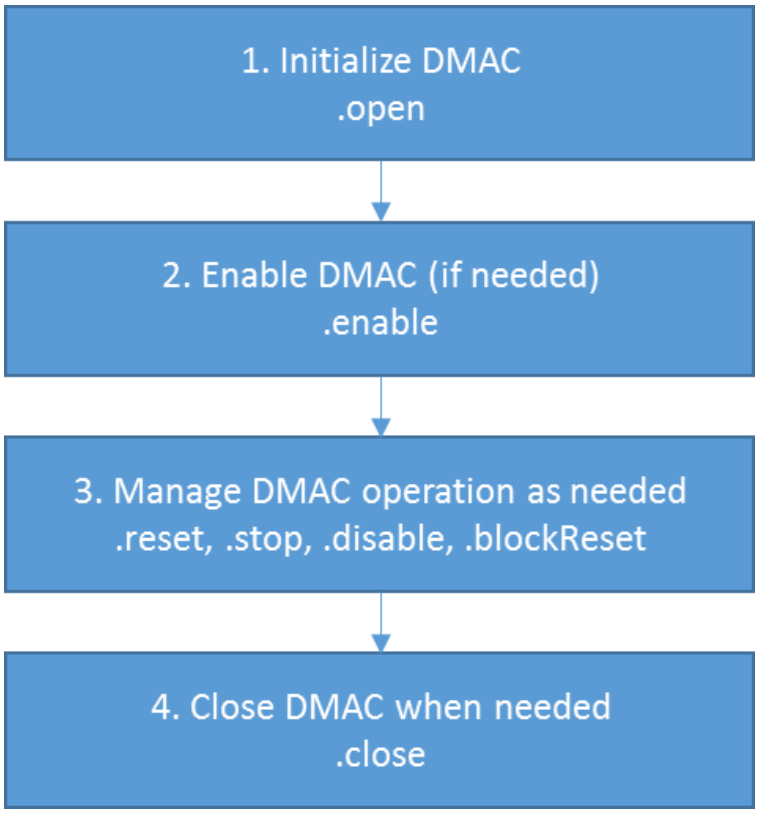


Figure 318: Flow Diagram of a Typical DMAC HAL Module Application

4.2.16 DTC Driver

The Data Transfer Controller (DTC) HAL module provides high-level APIs for data-transfer applications and is implemented on r_dtc. The DTC HAL module uses the DTC peripheral on the Synergy MCU. A user-defined callback can be created to inform the CPU when transfer events occur.

4.2.16.1 DTC HAL Module Features

The Data Transfer Controller (DTC) HAL module moves data from a user-specified source to a user-specified destination when an interrupt or event occurs.

- Supports the DTC module on a Synergy MCU
- Supports interrupts if needed
- Supports multiple transfer modes
  - Single transfer
  - Repeat transfer
  - Block transfer
  - Address increment or fixed modes
  - Chain transfers
- Supports multiple channels (depending on selected implementation)
  - Number of channels is limited only by the size of the RAM-based vector table

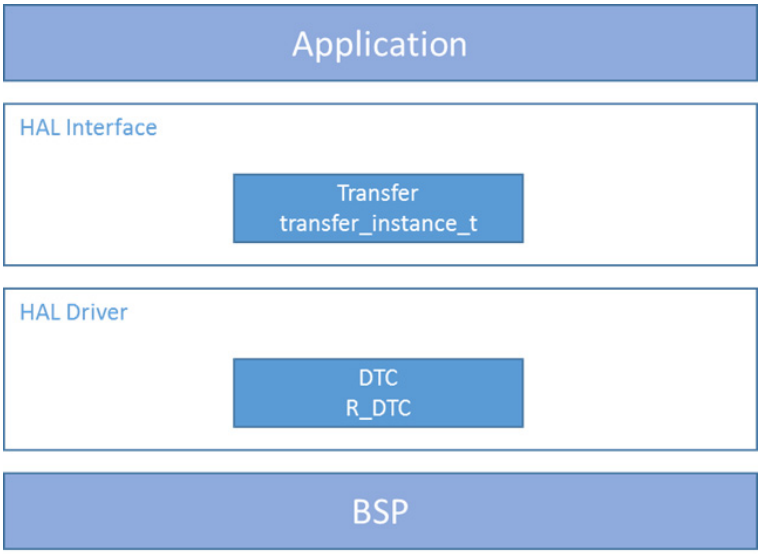


Figure 319: DTC HAL Module Block Diagram

4.2.16.2 DTC HAL Module APIs Overview

The DTC HAL module defines APIs for opening, closing, reset, enabling, disabling, starting, and stopping. Note that the DTC and the DMAC use the same transfer interface to make it easier to change between DTC and DMA implementations. The API calls are the same independent of the lower-level implementations. A complete list of the available APIs, an example API call, and a short description of each can be found in the table below. A table of status return values follows.

DTC HAL Module API Summary

Function Name	Example API Call and Description
open	g_transfer0.api->open(g_transfer0.p_ctrl, g_transfer0.p_cfg) Initial configuration. Enables the transfer if auto_enable is true and p_src, p_dest, and length are valid. Transfers can also be enabled using enable or reset.
close	g_transfer0.api->close(g_transfer0.p_ctrl) Close device channel. Turns off hardware if last channel open.
reset	g_transfer0.api->reset(g_transfer0.p_ctrl, &source, &destination, number_of_transfers) Reset source address pointer, destination address pointer, and/or length, keeping all other settings the same. Enable the transfer if p_src, p_dest, and length are valid.
.start	g_transfer0.api->start(g_transfer0.p_ctrl, mode) Start transfer in software.
stop	g_transfer0.api->stop(g_transfer0.p_ctrl) Stop transfer in software. The transfer will stop after completion of the current transfer.
enable	g_transfer0.api->enable(g_transfer0.p_ctrl) Enable transfer. Transfers occur after the activation source event (or when start is called if ELC_EVENT_ELC_SOFTWARE_EVENT_0 or ELC_EVENT_ELC_SOFTWARE_EVENT_0 is chosen as activation source).

Function Name	Example API Call and Description
disable	g_transfer0.api->disable(g_transfer0.p_ctrl) Disable transfer. Transfers do not occur after the transfer_info_t::activation source event (or when start is called if ELC_EVENT_ELC_SOFTWARE_EVENT_0 or ELC_EVENT_ELC_SOFTWARE_EVENT_0 is chosen as transfer_info_t::activation_source).
versionGet	g_transfer0.api->versionGet(&version) Gets version and stores it in provided pointer version.
infoGet	g_transfer0.api->infoGet(g_transfer0.p_ctrl, &info) Provides information about this transfer.
blockReset	g_transfer0.api->blockReset(g_transfer0.p_ctrl, &source, &destination, length, size, number_of_transfers) Reset source address pointer, destination address pointer, and/or length, for block transfer keeping all other settings the same. Enable the transfer if p_src, p_dest, and length are valid.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_ASSERTION	Parameter has invalid value.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_UNSUPPORTED	Operation not configured correctly.

Name	Description
SSP_ERR_IN_USE	The channel specified has already been opened. No configurations were changed. Call the associated Close function or use associated Control commands to reconfigure the channel.
SSP_ERR_HW_LOCKED	The DTC hardware resource is locked.
SSP_ERR_IRQ_BSP_DISABLED	IRQ not enabled in BSP.
SSP_ERR_NOT_ENABLED	Operation failed.
SSP_ERR_NOT_OPEN	The channel is not opened.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API references for the associated module for a definition of all relevant status return values.

4.2.16.3 DTC HAL Module Operational Overview

The Direct Memory Access Controller (DMAC) and the Data Transfer Controller (DTC) can be used to move data within the Synergy MCU. There are some considerations when selecting between these implementations; the following operational overview includes information on each to help you determine which implementation is best for your application. The DTC module is recommended for most generic transfer applications, but either module can be used for basic transfer functionality. The use-cases for each transfer module are given below:

Selecting the DTC HAL Module

The DTC HAL module uses a RAM-based vector table with slots for every interrupt in the system. When the DTC transfer completes, the activation source interrupt is called. The activation source interrupt must be enabled to use the DTC. The activation source interrupt is generally muted by the DTC until the transfer completes, unless TRANSFER_IRQ_EACH is specified in the configuration. For example, if a normal-mode transfer with a length of 16 is triggered by a timer, the timer interrupt does not fire the first 15 times while the transfer is in effect. After the 16th transfer, the timer interrupt fires. The DTC also allows chained transfers, meaning that more than one transfer can occur after a single activation-source interrupt. This feature is supported by the driver but must be configured outside the ISDE.

Selecting the DMAC HAL Module

The DMAC HAL module moves data from a user-specified source to a user-specified destination when an interrupt or event occurs. The DMAC HAL module uses DMAC peripheral registers, so the number of transfers in the system is limited to the number of DMAC channels on the device. The activation source does not have to be enabled to use the DMAC. When the DMAC transfer completes, a DMAC interrupt is called. If the activation source interrupt is enabled, it fires at the same time the transfer is triggered. If the DMAC interrupt is enabled, it fires after all transfers are complete. For example, if a normal-mode transfer with a length of 16 is triggered by a timer, the timer interrupt fires at the same time each transfer occurs and the DMAC interrupt fires after the 16th transfer completes. The DMAC HAL module does not support chained transfers.

DTC HAL Module Important Operational Notes and Limitations

Normal Mode

In normal mode, a single transfer is triggered each time an activation-source event occurs. A single transfer is 1 byte, 2 bytes, or 4 bytes, depending on the setting selected in the size parameter. Each time a transfer occurs, the transfer length is decremented by 1. When the transfer length reaches 0, the transfer is complete.

Repeat Mode

In repeat mode, a single transfer is triggered each time an activation-source event occurs. A single transfer is 1 byte, 2 bytes, or 4 bytes, depending on the setting selected in the size parameter. Each time a transfer occurs, the transfer length is decremented by 1. When the transfer length reaches 0, the transfer length is reloaded with its initial value and the transfer restarts. If the repeat area is set to source, the source register is also reloaded with its initial value when the transfer restarts. Alternatively, if the repeat area is set to destination, the destination register is reloaded with its initial value when the transfer restarts.

Block Mode

In the block mode, the entire transfer length is transferred each time an activation-source event occurs. For example, if a transfer is configured in block mode with the timer as the activation source, a 2-byte size, and a 12-byte length, 24 bytes are transferred each time the activation source event occurs. Each time a transfer occurs, the transfer length is decremented by 1. When the transfer length reaches 0, the transfer length is reloaded with its initial value and the transfer restarts. If the repeat area is set to source, the source register is also reloaded with its initial value when the transfer restarts. Alternatively, if the repeat area is set to destination, the destination register is reloaded with its initial value when the transfer restarts.

Address Mode

After each transfer of size (1 byte, 2 bytes, or 4 bytes), the source pointer and destination pointer are adjusted by src_addr_mode and dest_addr_mode, respectively. For example, if src_addr_mode is set to TRANSFER_ADDR_MODE_INCREMENTED and size is set to TRANSFER_SIZE_4_BYTES, the p_dest pointer is incremented by 4 (the transfer size) after each transfer. The pointer does not change if set to TRANSFER_ADDR_MODE_FIXED.

Chained Transfers

Chained transfers are only supported by the DTC. To use a chained transfer, create an array of transfer_info_t structures. Set chain_mode to TRANSFER_CHAIN_MODE_ENABLED for all transfers except the last transfer. Set p_info to the base of the first structure in the array for transfer_info_t structures.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.16.4 Including the DTC HAL Module in an Application

This section describes how to include the DTC HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the DTC HAL driver to an application, simply add it to a thread using the Stacks Selection Sequence provided in the table below. (The default name for the Transfer Driver is g_transfer0. This name can be changed in the associated **Properties** window.)

DTC HAL Driver Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_ transfer0 DTC Driver on r_dtc	**Threads > HAL/Common**	**New Stack > Driver > Transfer > Transfer Driver on r_dtc**

When the DTC HAL module on r_dtc is added to the thread stack as shown in the figure below, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information will be box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

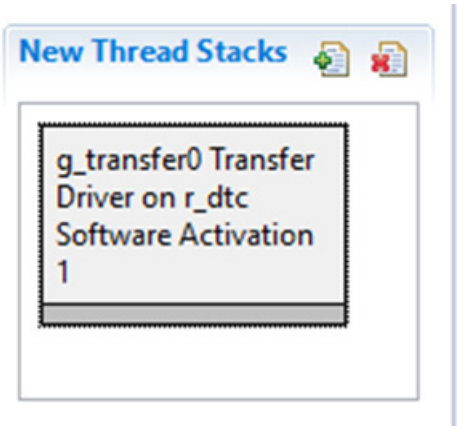


Figure 320: DTC HAL Module Stack

4.2.16.5 Configuring the DTC HAL Module

The DTC HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the **Properties** window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the **Properties** tab in the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the **Properties** window of the associated module. Simply select the indicated module and then view the **Properties** window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the **Properties** window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+.) This level of detail is not included in the configuration properties tables below, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the configuration table settings given below. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the DTC HAL Module on r_dtc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Selects if code for parameter checking is to be included in the build.

ISDE Property	Value	Description
Software Start	Disabled, Enabled (Default: Disabled)	Selects if Software Start to be enabled
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Section to place dtc vector table in
Name	g_transfer0	Module name.
Mode	Normal	Mode selection.
Transfer Size	1 Byte, 2 Bytes, 4 Bytes (Default: 2 Bytes)	Transfer size selection.
Destination Address Mode	Fixed, Incremented, Decremented (Default: Fixed)	Destination address mode selection.
Source Address Mode	Fixed, Incremented, Decremented (Default: Fixed)	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source, Destination (Default: Source)	Repeat area selection.
Interrupt Frequency	After all transfers have completed, After each transfer (Default: After all transfers have completed)	Defines when an interrupt occurs.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.

ISDE Property	Value	Description
Activation Source (Must enable IRQ)	The full list of interrupt activation sources can be found in Table 14.4 of the MCU user manual. (Default: Software Activation 1)	Activation source selection.
Auto Enable	True, False (Default: True)	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 (lowest, not valid if using Thread X), Disabled (Default: Disabled)	Interrupt priority.

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

DTC HAL Module Clock Configuration

The DTC peripheral module use ICLK as the clock source. The ICLK frequency is set by using the SSP configurator **Clocks** tab prior to a build, or by using the CGC Interface at run-time.

DTC HAL Module Pin Configuration

The DTC is not associated with any pins.

4.2.16.6 Using the DTC HAL Module in an Application

The typical steps in using the DTC HAL module in an application are:

1) Initialize the DTC using the open API.
2) Enable the DTC using the enable API (if not auto enabled).
3) Manage transfers using other APIs as needed.
4) Close the DTC when needed using the close API.

The common steps for using the DTC HAL module are illustrated in a typical operational flow diagram in the following figure:

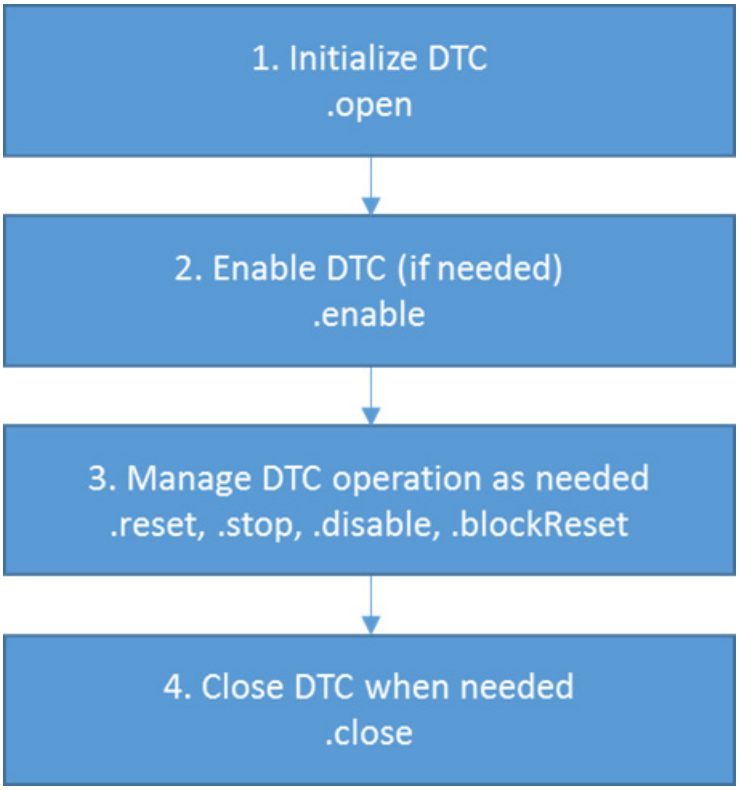


Figure 321: Flow Diagram of a Typical DTC HAL Module Application

4.2.17 ELC Driver

The Event Link Controller (ELC) HAL module provides high-level APIs for ELC HAL applications and is implemented on r_elc. The ELC HAL module uses the ELC peripheral on the Synergy MCU. There are no callbacks associated with the ELC HAL module. The project configurator in the e^2 studio Integrated Solution Development Environment (ISDE), includes the ELC HAL module in every project by default. To configure the ELC HAL module, select it in the HAL/Common module in the **Threads** tab, and then click on it in the HAL/Common Stacks window.

4.2.17.1 ELC HAL Module Features

The ELC HAL module supports the following functions:

- Create an event link between two blocks.
- Break that event link between two blocks.
- Generate one of two software events that interrupt the CPU.

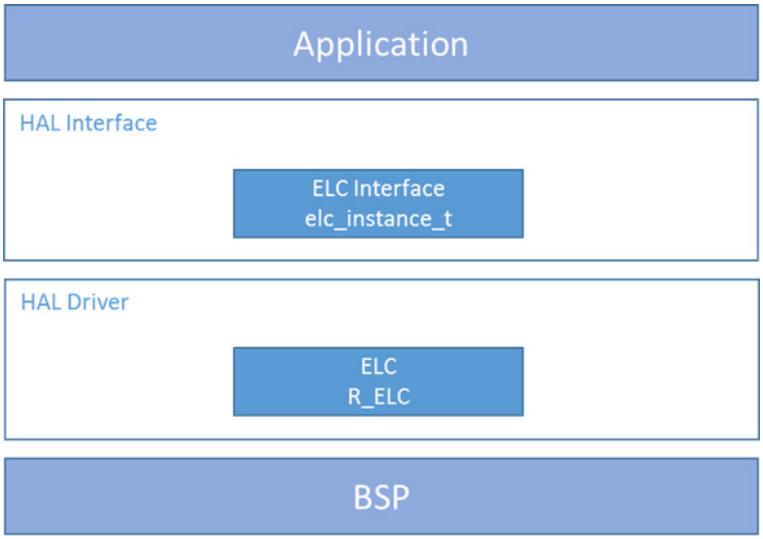


Figure 322: ELC HAL Module Block Diagram

4.2.17.2 ELC HAL Module APIs Overview

The ELC HAL module defines APIs for initializing, enabling, disabling, and creating or breaking event links between modules. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

ELC HAL Module API Summary

Function Name	Example API Call and Description
init	g\elc.p_api->init(g_elc.p_cfg) Initialize all links in the Event Link Controller.
softwareEventGenerate	g_elc.p_api->softwareEventGenerate(event_num) Generate a software event in the Event Link Controller.
linkSet	g_elc.p_api->linkSet(peripheral, signal) Create a single event link.
linkBreak	g_elc.p_api->linkBreak(peripheral) Break an event link.

Function Name	Example API Call and Description
enable	g_elc.p_api->enable() Enable the operation of the Event Link Controller.
disable	g_elc.p_api->disable() Disable the operation of the Event Link Controller.
versionGet	g_elc.p_api->versionGet(&version) Retrieve the API version with the version pointer.

NOTE: For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successfully completed.
SSP_ERR_ASSERTION	p_version is NULL.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.17.3 ELC HAL Module Operational Overview

The ELC HAL module allows the developer to link various peripheral operations by using events generated by one peripheral to trigger the start of operation of another peripheral within the Synergy MCU. The ELC HAL module APIs make it easy to create a link between two blocks (for example, from a timer to an ADC to control a periodic scan interval). By connecting various peripherals in this way, intelligent functions can be constructed that require little, if any, CPU intervention.

The following figure shows a simplified block diagram of the ELC, showing the input event sources and the peripherals that can be triggered by these events. (The number of input and output triggers is specific to the S7G2 MCU.) Other Synergy devices support a different number of events.

Figure 323: ELC Hardware Block Diagram

The application project associated with this module guide shows an example of the ELC in use.

It is possible to find the mapping of ELC peripherals in the file, bsp_elc.h, in the Synergy-generated code. The event list and peripherals for the S7G2 microcontroller unit (MCU) follows. Additional information on ELC operation is also available in the associated MCU user's manual.

```
/** Possible peripherals to be linked to event signals */
typedef enum e_elc_peripheral
{
    ELC_PERIPHERAL_GPT_A                                    =    (0),
    ELC_PERIPHERAL_GPT_B                                    =    (1),
    ELC_PERIPHERAL_GPT_C                                    =    (2),
    ELC_PERIPHERAL_GPT_D                                    =    (3),
    ELC_PERIPHERAL_GPT_E                                    =    (4),
    ELC_PERIPHERAL_GPT_F                                    =    (5),
    ELC_PERIPHERAL_GPT_G                                    =    (6),
    ELC_PERIPHERAL_GPT_H                                    =    (7),
    ELC_PERIPHERAL_ADC0                                     =    (8),
    ELC_PERIPHERAL_ADC0_B                                   =    (9),
    ELC_PERIPHERAL_ADC1                                     =   (10),
    ELC_PERIPHERAL_ADC1_B                                   =   (11),
    ELC_PERIPHERAL_DAC0                                     =   (12),
    ELC_PERIPHERAL_DAC1                                     =   (13),
    ELC_PERIPHERAL_IOPORT1                                  =   (14),
    ELC_PERIPHERAL_IOPORT2                                  =   (15),
    ELC_PERIPHERAL_IOPORT3                                  =   (16),
    ELC_PERIPHERAL_IOPORT4                                  =   (17),
    ELC_PERIPHERAL_CTSU                                     =   (18),
} elc_peripheral_t;


/** Sources of event signals to be linked to other peripherals or the CPU1
 * @note This list may change based on device. This list is for S7G2.
 * */
typedef enum e_elc_event
{
    ELC_EVENT_ICU_IRQ0                                      =    (1),
    ELC_EVENT_ICU_IRQ1                                      =    (2),
    ELC_EVENT_ICU_IRQ2                                      =    (3),
    ELC_EVENT_ICU_IRQ3                                      =    (4),
.
.
.
    ELC_EVENT_ICU_IRQ12                                     =   (13),
    ELC_EVENT_ICU_IRQ13                                     =   (14),
    ELC_EVENT_ICU_IRQ14                                     =   (15),
    ELC_EVENT_ICU_IRQ15                                     =   (16),
.
.
.
    ELC_EVENT_GLCDC_LINE_DETECT                             =  (506),
    ELC_EVENT_GLCDC_UNDERFLOW_1                             =  (507),
    ELC_EVENT_GLCDC_UNDERFLOW_2                             =  (508),
    ELC_EVENT_DRW_INT                                       =  (509),
    ELC_EVENT_JPEG_JEDI                                     =  (510),
    ELC_EVENT_JPEG_JDTI                                     =  (511),
} elc_event_t;
```

Figure 324: Mapping of ELC Peripherals in bsp_elc.h example

ELC HAL Module Important Operational Notes and Limitations

The ELC HAL module needs no pin, clocking, or interrupt configuration. It is just a 'connect' mechanism between peripherals. However, if linking I/O Ports via the ELC, the I/O pins need to be configured as inputs or outputs.

Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.17.4 Including the ELC HAL Module in an Application

The ISDE automatically adds the necessary ELC HAL module to the HAL/Common thread in the project by default; as such, the steps described later in this section are not necessary or even forbidden (if the ELC HAL module is already configured in terms of the HAL/Common thread — only one driver instance is allowed). The following descriptions are provided for completeness and in case the ELC HAL module is mistakenly removed.

This section describes how to include the ELC HAL module in an application using the SSP configurator.

NOTE: It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these tasks, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications*. *

To add the ELC Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the ELC HAL module is g_elc. This name can be changed in the associated Properties window.)

ELC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_elc ELC Driver on r_elc	Threads	New Stack> Driver> System> ELC Driver on r_elc

When the ELC HAL Driver on r_elc is added to the thread stack as shown in the following figure, the configurator will automatically display the options dialog. The only available change to the ELC HAL module is to enable or disable parameter checking.

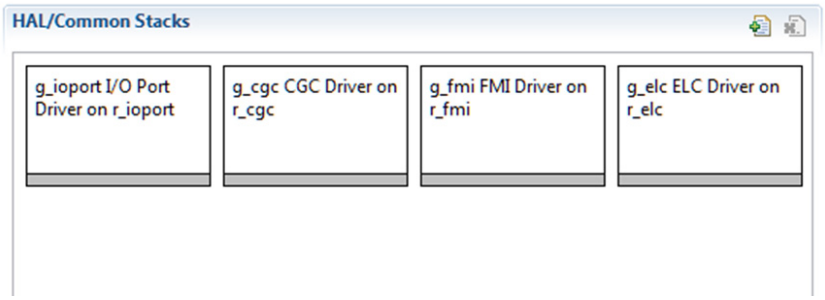


Figure 325: ELC HAL Module Stack

4.2.17.5 Configuring the ELC HAL Module

The ELC HAL module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or

operating modes, which must be configured for lower-level modules for successful operation. Only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP Configurator, and are shown in the following tables for easy reference.

NOTE: You may want to open your ISDE and create the ELC HAL module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the ELC HAL Module on r_elc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enable or disable the parameter error checking.
Name	g_elc	Module name.

NOTE: The example settings and defaults are for a project using the Synergy S7G2 Group. Other MCUs may have different default values and available configuration settings.

ELC HAL Module Clock Configuration

There is no clock configuration for the ELC block.

ELC HAL Module Pin Configuration

There are no pins associated directly with the ELC HAL Module that require configuration.

4.2.17.6 Using the HAL Module in an Application

The typical steps in using the ELC HAL module in an application are:

1) Initialize the ELC using the init and enable APIs (automatically done by ISDE).
2) Link a peripheral with an event using the linkSet API.
3) Enable the linkage with the enable API.

The following figures illustrates the common steps in a typical operational flow diagram.

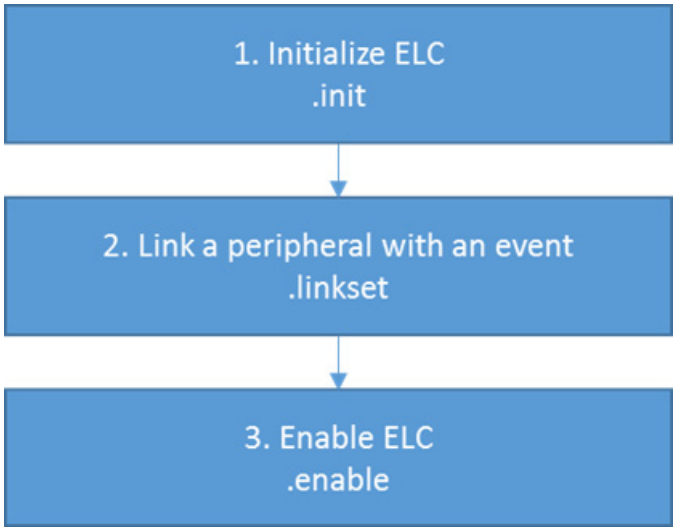


Figure 326: Flow Diagram of a Typical ELC HAL Module Application

4.2.18 External IRQ Driver

The External IRQ HAL module provides high-level APIs for configuring and using external IRQ pins on Synergy MCUs. The External IRQ HAL module is implemented on r_icu and uses the Interrupt Controller Unit (ICU) of the Synergy MCU.

4.2.18.1 External IRQ HAL Module Features

- Supports the external interrupt pins available on the target Synergy MCU
- Supports multiple function options:
  - Enabling and disabling generation of an interrupt
  - Enabling and disabling the IRQ noise filter
  - Setting external pin IRQ trigger (Rising edge, falling edge or low level on the IRQ pin)
- Supports configuring a user callback function, which will be invoked by the HAL module when an external pin interrupt is generated.

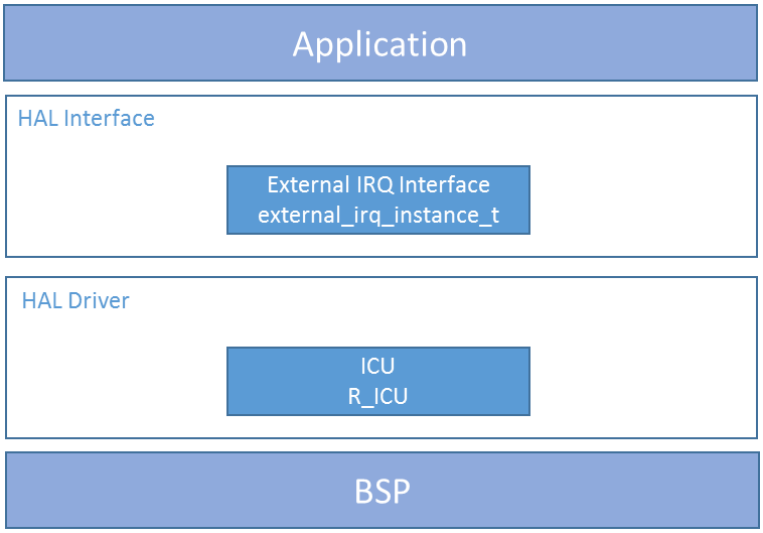


Figure 327: External IRQ HAL Module Block Diagram

4.2.18.2 External IRQ HAL Module APIs Overview

The External IRQ HAL module defines APIs for opening, closing, and waiting for interrupt events from external pins. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows.

External IRQ HAL Module API Summary

Function Name	Example API Call and Description
open	g_external_irq.p_api->open(g_external_irq.p_ctrl, g_external_irq.p_cfg) Open instance and initialize.
enable	g_external_irq.p_api->enable(g_external_irq.p_ctrl) Enable callback when IRQ occurs.
disable	g_external_irq.p_api->disable(g_external_irq.p_ctrl) Disable callback when IRQ occurs.

Function Name	Example API Call and Description
triggerSet	g_external_irq.p_api->triggerSet(g_external_irq.p_ctrl, trigger) Set trigger.
filterEnable	g_external_irq.p_api->filterEnable(g_external_irq.p_ctrl) Enable noise filter.
filterDisable	g_external_irq.p_api->filterDisable(g_external_irq.p_ctrl) Disable noise filter.
close	g_external_irq.p_api->close(g_external_irq.p_ctrl); Close instance.
versionGet	g_external_irq.p_api->wait(&version); Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_ASSERTION	Assertion error.
SSP_ERR_INVALID_ARGUMENT	Callback is not NULL but ISR is not enabled.
SSP_ERR_IN_USE	Device in use.
SSP_ERR_NOT_OPEN	Device unopened.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API references for the associated module for a definition of all relevant status return values.

4.2.18.3 External IRQ HAL Module Operational Overview

The External IRQ HAL module provides a set of APIs for controlling external interrupts. Interrupts can be triggered on rising edge, falling edge, both edges or low level of the input signal on the external IRQ pin. A digital-filtering function can be enabled to eliminate some noise on the input signal. A user-callback function is supported and is triggered each time an IRQ event occurs.

To trigger a transfer of data using the DMAC or DTC peripheral when the configured external IRQ event occurs, configure the DMAC or DTC transfer with the activation source set to ELC_EVENT_PORTn_IRQ (where n is the IRQ channel number.)

Other peripherals can be triggered to start from an external interrupt using the Event Link Controller (ELC.) Refer to the SSP User Manual User Guide for the ELC HAL module for more information.

External IRQ HAL Module Important Operational Notes and Limitations

- Refer to the datasheet for the Synergy device to be programmed to find the port pins which support the external interrupt functions and to obtain the external IRQ number for a given port pin.
- The external IRQ number corresponds to the channel setting in the ISDE Properties window for the External IRQ HAL module.
- The PORTn (where n is the IRQ number) interrupt must be enabled in the BSP to notify the module that the anticipated hardware event has occurred.
- A user-callback function can be registered in the open API. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the IRQn triggers. NOTE: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can adversely affect the responsiveness of the system.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.18.4 Including the External IRQ HAL Module in an Application

This section describes how to include the External IRQ HAL module in an application using the SSP configurator.

Note: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the External IRQ HAL driver to an application, simply add it to a thread using the Stacks Selection Sequence given in the table below. (The default name for the module is g_external_irq0. This name can be changed in the associated Properties window.)

External IRQ HAL Driver Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
r_icu0 External IRQ Driver on r_icu	Threads	New Stack> Driver> Input> External IRQ Driver on r_icu

When the External IRQ HAL module on r_icu is added to the thread stack as shown in the figure below, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information will be box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

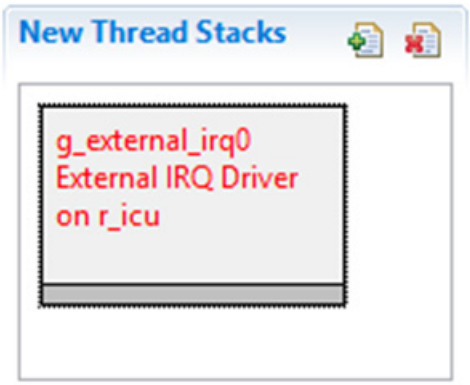


Figure 328: External IRQ HAL Module Stack

4.2.18.5 Configuring the External IRQ HAL Module

The External IRQ HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+.) This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt priority levels.

Note: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the configuration table settings given below. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the External IRQ HAL Module on r_icu

ISDE Property	Value	Description
Parameter Checking	Default, Enabled, Disabled ; (Default: Default)	Parameter Checking
Name	g_external_irq0	Driver name.

ISDE Property	Value	Description
Channel	0	Specifies the hardware IRQ channel used.
Trigger condition	Falling, Rising, Both Edges, Low Level; (Default: Rising)	Trigger selection.
Digital Filtering	Enabled, Disabled ; (Default: Disabled)	Digital filter enable/disable.
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCLK/1, PLCK/8, PLCK/32, PCLK/64;(Default: PCKL/64)	Sets noise filter sampling period.
Interrupt enabled after initialization	True, False ; (Default: True)	Interrupt enable selection.
Callback	NULL	A user callback function can be registered using the open API. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the IRQn triggers. Note: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Interrupt Priority	Enabled, Disabled ; (Default: Disabled)	Interrupt priority setting

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

External IRQ HAL Module Clock Configuration

The IRQ peripheral module doesn't require any specific clock settings.

External IRQ HAL Module Pin Configuration

The External IRQ peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the IRQ pins.

Selection Sequence for the External IRQ HAL Driver on r_icu

Resource	ISDE Tab	Pin selection Sequence
IRQ	Pins	Select Peripherals > Input: IRQ > IRQ0

NOTE: The selection sequence assumes IRQ0 is the desired hardware target for the driver.

Pin Configuration Settings for External IRQ Driver on r_icu

Pin Configuration Property	Value	Description
Operation Mode	Disabled, Enabled; (Default: Disabled)	Select Enabled to enable interrupts
NMI	None, P200; (Default: None)	Non-maskable interrupt Pin
IRQ00:14	None, Pnn, Pmm; Default: None	Interrupt request Pin

NOTE: The example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.18.6 Using the External IRQ HAL Module in an Application

The typical steps in using the External IRQ HAL module in an application are:

The typical steps in using the External IRQ HAL module in an application are:

1) Initialize the External IRQ HAL module using the open API
2) Enable the IRQ (if needed) with the enable API
3) Enable the noise filter (if needed) with the filterEnable API
4) Change the trigger condition (only if the module is closed previously to avoid any false events) with the triggerSet API
5) Disable the noise filter (if enabled) with filterDisable API
6) Close the module (if needed) with the close API

These common steps are illustrated in a typical operational flow diagram in the figure below:

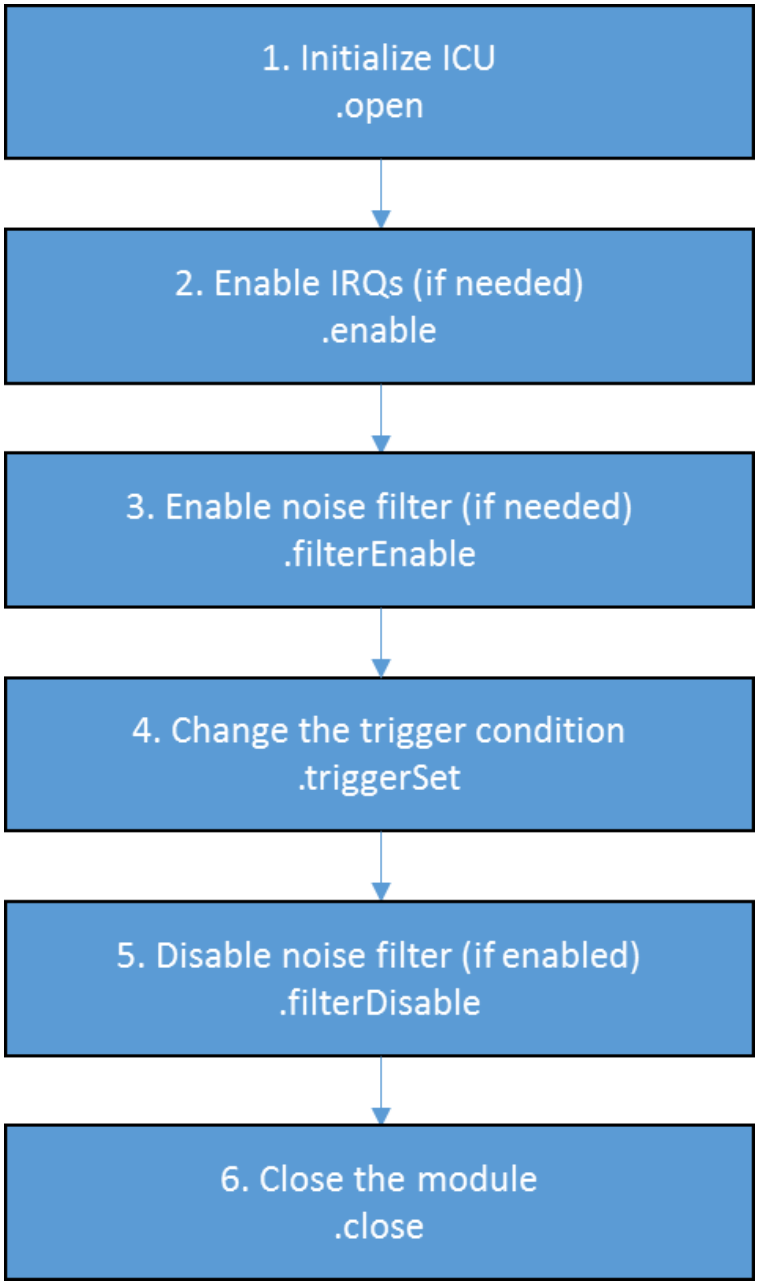


Figure 329: Typical External IRQ HAL Module Application Flow Chart

4.2.19 Flash Driver

There are two separate Flash modules: r_flash_lp and r_flash_hp. The High-Performance Flash module (Flash_HP) is used for programming the S7 and S5 family of MCUs. The Low-Power Flash module (Flash_LP) is used for programming

the S3 and S1 family of MCUs. The two are not interchangeable, although the APIs and other features of the modules are very similar. This guide covers the operation of both of these HAL modules.

4.2.19.1 Flash HAL Module Features

The Flash HAL modules APIs allow an application to read, write, and erase both the data and ROM flash areas that reside within the MCU. The amount of flash memory available varies across MCU parts, but the API functions apply to all devices. Key features of the Flash HAL modules are listed below:

- Support for block erasing, reading, writing, and blank checking of code flash (ROM).
- Support for both blocking and non-blocking erasing, reading, writing, and blank checking of data flash.
- Support for blocking erasing, reading, writing, and blank checking of code flash.
- Support for callback functions for completion of non-blocking data-flash operations.
- Support for access window (write protection) for ROM Flash, allowing only specified areas of code flash to be erased or written.
- Support for boot block-swapping which allows safe rewriting of the startup program without first erasing it.

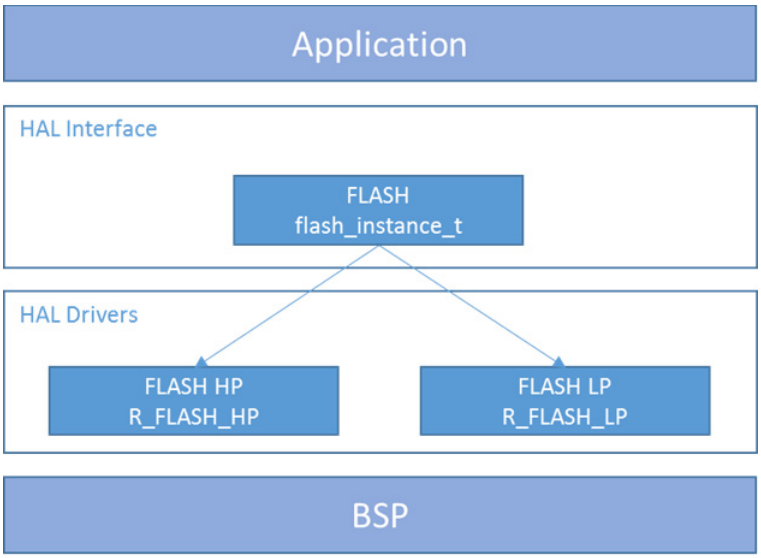


Figure 330: Flash HAL Module Block Diagram

4.2.19.2 Flash HAL Module APIs Overview

The Flash HAL module defines APIs for several operations including opening, reading, erasing, and closing the flash memory. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows.

Flash HAL Module API Summary

Function Name	Example API Call and Description
open	g_flash0.p_api->open(g_flash0.p_ctrl, g_flash0.p_cfg); Open FLASH device.
write	g_flash0.p_api->write(g_flash0.p_ctrl,(uint32_t) write_buffer, FLASH_CF_32KB_BLOCK55, CODE_BLOCK_SIZE_32KB); Write FLASH device.
read	g_flash0.p_api->read(g_flash0.p_ctrl, read_buffer, DATA_FLASH_ADDR, num_bytes); Read FLASH device.
read	g_flash0.p_api->erase(g_flash0.p_ctrl, FLASH_CF_32KB_BLOCK55,num_sectors); Erase FLASH device.
blankCheck	g_flash0.p_api->blankCheck(g_flash0.p_ctrl, FLASH_CF_32KB_BLOCK55, FLASH_DATA_BLOCK_SIZE, &blankCheck); Blank check FLASH device.
close	g_flash0.p_api->close(g_flash0.p_ctrl); Close FLASH device.
statusGet	g_flash0.p_api->statusGet(g_flash0.p_ctrl); Get Status for FLASH device.
accessWindowSet	g_flash0.p_api->accessWindowSet(g_flash0.p_ctrl, FLASH_CF_32KB_BLOCK1, FLASH_CF_32KB_BLOCK3); Set Access Window for FLASH device.

Function Name	Example API Call and Description
accessWindowClear	g_flash0.p_api->accessWindowClear(g_flash0.p_ctrl); Clear any existing code-flash access window for FLASH device.
reset	g_flash0.p_api->reset(g_flash0.p_ctrl); Reset function for FLASH device.
updateFlashClockFreq	g_flash0.p_api-> updateFlashClockFreq (g_flash0.p_ctrl); Update Flash clock frequency (FCLK) and recalculate timeout values.
startupAreaSelect	g_flash0.p_api->startupAreaSelect(g_flash0.p_ctrl, FLASH_STARTUP_AREA_BLOCK1, true); Select which block - Default (Block 0) or Alternate (Block 1) is used as the start-up area block. Refer to the table below for all the possible values for parameter2.
versionGet	g_flash0.p_api->versionGet(&version); Retrieve the API version using the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables review the *SSP User's Manual* available as described in the References section at the end of this document.

.setupAreaSelect parameter2 options

Swap Type	Is_temporary	Operation
FLASH_STARTUP_AREA_BLOCK0	False	On next reset, Startup area will be Block 0.
FLASH_STARTUP_AREA_BLOCK0	False	On next reset, Startup area will be Block 0.

Swap Type	Is_temporary	Operation
FLASH_STARTUP_AREA_BLOCK1	False	On next reset, Startup area will be Block 1.
FLASH_STARTUP_AREA_BLOCK1	True	Startup area is immediately, but temporarily switched to Block 1.
FLASH_STARTUP_AREA_BTFLG	True	Startup area is immediately, but temporarily switched to the Block determined by the Configuration BTFLG.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_IN_USE	Device in use error.
SSP_FLASH_ERR_FAILURE	Flash failure error.
SSP_ERR_FCLK	FCLK must be a minimum of 4 MHz for Flash operations.
SSP_ERR_TIMEOUT	Timeout error.
SSP_ERR_INVALID_SIZE	Invalid size error.
SSP_ERR_INVALID_ADDRESS	Invalid address error.
SSP_ERR_ASSERTION	Assertion error.
SSP_ERR_INVALID_BLOCKS	Invalid number of blocks specified.
SSP_ERR_INVALID_ARGUMENT	Invalid argument error.
SSP_ERR_HW_LOCKED	Peripheral already in use.
SSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of attempting to Erase an area that is protected by an Access Window.
SSP_ERR_NOT_OPEN	Flash has not yet been opened.
SSP_ERR_IRQ_BSP_DISABLED	Caller is requesting BGO (background mode operation) but the Flash interrupt is not enabled.

Name	Description
SSP_ERR_WRITE_FAILED	Write operation failed. This may be returned if the requested Flash area is not blank.
SSP_ERR_PE_FAILURE	Failed to enter P/E mode
false	Supplied address is valid flash address on this MCU.
true	Supplied address is valid and p_block info contains the details on this address's block.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API Reference section for the associated module for a definition of all relevant status return values.

4.2.19.3 Flash HAL Module Operational Overview

The Flash API makes the process of programming and erasing on-chip flash areas easy. Both code (User ROM) and data-flash areas are supported. The API, in its simplest form, can be used to perform blocking erase and program operations. The term "blocking" means that when a program or erase function is called, the function does not return until the operation has finished. This API supports blocking for both code and data-flash, with BGO (background-mode operation) available for data-flash operations only. When a code-flash operation is on-going, you cannot access that code-flash area. If you attempt to access the code-flash area while a code-flash operation is in progress, the flash-control unit will transition into an error state.

It is important to keep in mind that even though a code-flash operation is blocking, there are several situations where the code-flash could still end up being accessed while the operation is blocking and these must be prevented. This includes:

- Vector table access if the Vector table is located in ROM.
- ROM access by an interrupt vectoring to a ROM address, even if the vector table itself is not in ROM.

A multithreaded application where multiple threads are allowed to continue to run while a code-flash operation is blocking.

Flash HAL Module Important Operational Notes and Limitations

startupAreaSelect() swaps Data in to Block 0. Be sure that the swapped-in data is valid if you use startupAreaSelect().

Data-Flash BGO Precautions

When using the data-flash BGO, the User ROM, RAM and external memory can still be accessed. You must ensure that the data-flash is not accessed during a data-flash operation. This includes interrupts that may access the data-flash.

Code-Flash Precautions

BGO mode is not supported for code-flash, so a code-flash operation will not return before the operation has completed. By default, the vector table resides in the User ROM (code-flash.) If an interrupt occurs during the ROM operation, then ROM will be accessed to fetch the interrupt's starting address and an error will occur.

The simplest work-around is to disable interrupts during code-flash operations. Another option is to copy the vector table to RAM, update the VTOR (Vector Table Offset Register) accordingly and ensure that any interrupt service routines execute out of RAM. Similarly, you must insure that if in a multithreaded environment, threads running from ROM cannot become active while a code-flash operation is in progress.

Blank Checking

The blankCheck API function checks whether code or data-flash contents are blank. Note that it is not possible to write to flash (code or data) without first erasing it. The blankCheck function determines whether a specified area is blank and therefore writable. In almost all cases, it is not sufficient to compare flash contents to 0xFF to determine whether the area is blank. The one exception is Flash HP code-flash. A 0xFF in Flash_HP code-flash does indicate blank. Renesas recommends strongly using the blankCheck API function in all cases.

Flash Status

The statusGet API function allows the application to query the 'Ready' status of the flash. This is useful in data-flash BGO operations when you choose not to use a callback function, so there is no asynchronous notification of a completed data-flash operation. In this case, the data-flash is configured to operate in BGO mode, so once the operation is started (an erase, for example), the call returns immediately with the operation executing in the background. By calling the statusGet API function, you can determine when the operation has safely completed or generated an error, and it is now safe to proceed with another flash operation.

Swap Blocks

The startupAreaSelect API function allows the user to select which block - default (Block 0) or alternate (Block 1) - is used as the startup-area block. The provided parameters determine which block will become the active startup block and whether that action will be immediate (but temporary) or permanent subsequent to the next reset.

Doing a temporary switch might appear to have limited usefulness; however, if there is an access window in place such that Block 0 is write-protected, then you could do a temporary switch, update the block, and switch them back without having to touch the access window.

Flash Clock (FCLK)

The FCLK is the clock used by the Flash peripheral in performing all Flash operations. It must be >= 4 MHz for successful flash operations. As part of the open function the Flash clock is checked and if < 4 MHz open will return SSP_ERR_FCLK. Once the Flash API has been opened, if the FCLK frequency is changed, the updateFlashClockFreq API function must be called to inform the API of the change. Failure to do so could result in flash operation failures and possibly damage the part.

Interrupts

Enable the flash ready interrupt only if you plan to use the data-flash BGO. In this mode, the application can initiate a data-flash operation and then be asynchronously notified of its completion, or an error, using a user supplied-callback function. The callback function is passed a structure containing event information that indicates the source of the callback event (that is,FLASH_EVENT_ERASE_COMPLETE)

When the FLASH FRDYI interrupt is enabled, the corresponding ISR will be defined in the flash driver. The ISR will call a user-callback function if one was registered with the open API.

NOTE: The Flash HP supports an additional flash-error interrupt and if the BGO mode is enabled for the FLASH HP then both FRDYI and FIFERR interrupts must be given a priority.

AccessWindow

An access window defines a contiguous area in Code Flash for which programming/erase is enabled. This area is on block boundaries with a starting and ending address being provided to accessWindowSet. The block containing the start address is the first block. The block containing the end address is the last block. The access window then becomes the first block – last block inclusive. Anything outside this range is write protected. Invalid address information provided to accessWindowSet will return SSP_ERR_INVALID_ADDRESS. An access window may be removed by calling the accessWindowClear API function.

- The High-Performance Flash module (Flash_HP) is the API used for programming the S7 and S5 family of MCUs.
- The Low-Power Flash module (Flash_LP) is the API used for programming the S3 and S1 family of MCUs.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.19.4 Including the Flash HAL Module in an Application

This section describes how to include the Flash HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Flash Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Flash Driver is g_flash0. This name can be changed in the associated Properties window.)

Flash Driver Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_flash0 Flash Driver on r_rflash_hp	Threads	New Stack > Driver > Storage > Flash Driver on r_flash_hp
g_flash0 Flash Driver on r_rflash_lp	Threads	New Stack > Driver > Storage > Flash Driver on r_flash_lp

When the Flash HAL modules on r_flash_hp or r_flash_lp are added to the thread stack as shown in the figure below, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information will be box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

NOTE: Both Flash HAL modules are shown in the figure below, but only one should be used, depending on the selected MCU; they are shown together only for completeness.

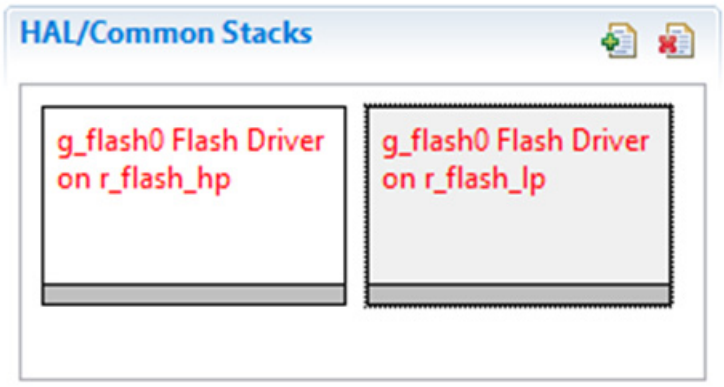


Figure 331: Flash HAL Module Stack

4.2.19.5 Configuring the Flash HAL Module

The Flash HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+.) This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the configuration table settings in the following tables. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

The Flash HAL Driver is implemented on one of two different modules, the r_flash_hp and the r_flash_lp, and the configuration settings for these implementations are given in the following tables:

Table 5 Configuration Settings for the Flash HAL Module on r_flash_hp

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Controls whether to include code for API parameter checking.
code-flash Programming Enable	Enable, Disabled Default: Disabled	Controls whether or not code-flash programming is enabled. Disabling reduces the amount of ROM used by the API.
Name	g_flash0	Module name.
data-flash Background Operation	Enabled, Disabled Default: Enabled	Enabling allows Flash API calls that reference data-flash to return immediately, with the operation continuing in the background.

ISDE Property	Value	Description
Callback	NULL	Callback function called when a data-flash BGO operation completes or errors. A user callback function can be registered in open. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Flash Ready Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Flash ready interrupt priority selection.
Flash Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Flash error interrupt priority selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration for the Flash HAL Module on r_flash_lp

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Controls whether to include code for API parameter checking.

ISDE Property	Value	Description
code-flash Programming Enable	Enable, Disabled Default: Disabled	Controls whether or not code-flash programming is enabled. Disabling reduces the amount of ROM used by the API.
Name	g_flash0	Module name.
data-flash Background Operation	Enabled, Disabled Default: Enabled	Enabling allows Flash API calls that reference data-flash to return immediately, with the operation continuing in the background.
Callback	NULL	Callback function called when a data-flash BGO operation completes or errors. A user callback function can be registered in open. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Flash Ready Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Flash ready interrupt priority selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration for the Flash HAL Module on r_flash_lp

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Controls whether to include code for API parameter checking.

ISDE Property	Value	Description
code-flash Programming Enable	Enable, Disabled Default: Disabled	Controls whether or not code-flash programming is enabled. Disabling reduces the amount of ROM used by the API.
Name	g_flash0	Module name.
data-flash Background Operation	Enabled, Disabled Default: Enabled	Enabling allows Flash API calls that reference data-flash to return immediately, with the operation continuing in the background.
Callback	NULL	Callback function called when a data-flash BGO operation completes or errors. A user callback function can be registered in open. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Flash Ready Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Flash ready interrupt priority selection.

NOTE: The example settings and defaults are for a project using the Synergy S3A7 MCU. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults can be desirable. For example, it might be useful to disable code-flash programming to reduce the code size of the driver.

Flash HAL Module Clock Configuration

Enable the flash-ready interrupt only if you plan to use the data-flash BGO (background mode operation.) In this mode, the application can initiate a data-flash operation and then be asynchronously notified of its completion (or an error) using a user-supplied callback function. The callback function is passed a structure containing event information that indicates the source of the callback event (for example, FLASH_EVENT_ERASE_COMPLETE.)

To enable interrupts, set the priority of the FCU > FRDYI interrupt on the ICU tab of the Project Configurator in e^2 studio. This sets BSP_IRQ_CFG_FCU_FRDYI in synergy_cfg/ssp_cfg/bsp/bsp_irq_cfg.h to the priority level selected.

When the FLASH FRDYI interrupt is enabled in the BSP, the corresponding ISR will be defined in the flash driver. The ISR will call a user-callback function if one was registered in open.

NOTE: Flash HP supports an additional flash-error interrupt and if BGO mode is enabled for FLASH HP, then both FRDYI and FIFERR interrupts must be given a priority.

Flash HAL Module Clock Configuration

The flash circuit uses FCLK as its clock. FCLK must be <= 4 MHz. If this clock rate changes after the flash Open() function is called, then you must call updateFlashClockFreq() to inform the flash API of the change.

Flash HAL Module Pin Configuration

The flash circuit does not use any MCU pins.

4.2.19.6 Using the Flash HAL Module in an Application

Some typical steps in using the Flash HAL module in an application are:

1) Initialize the Flash HAL using the open API.
2) Disable Interrupts.
3) Blank check a code flash area with the blankCheck API.
4) Erase one or more code-flash blocks with the erase API.
5) Write to code-flash with the write API.
6) Enable Interrupts.
7) Blank check a data flash area with the blankCheck API.
8) Erase one or more data-flash blocks using the erase API.
9) Write to data-flash using the write API.
10) Close using the close API if finished with all Flash operations.

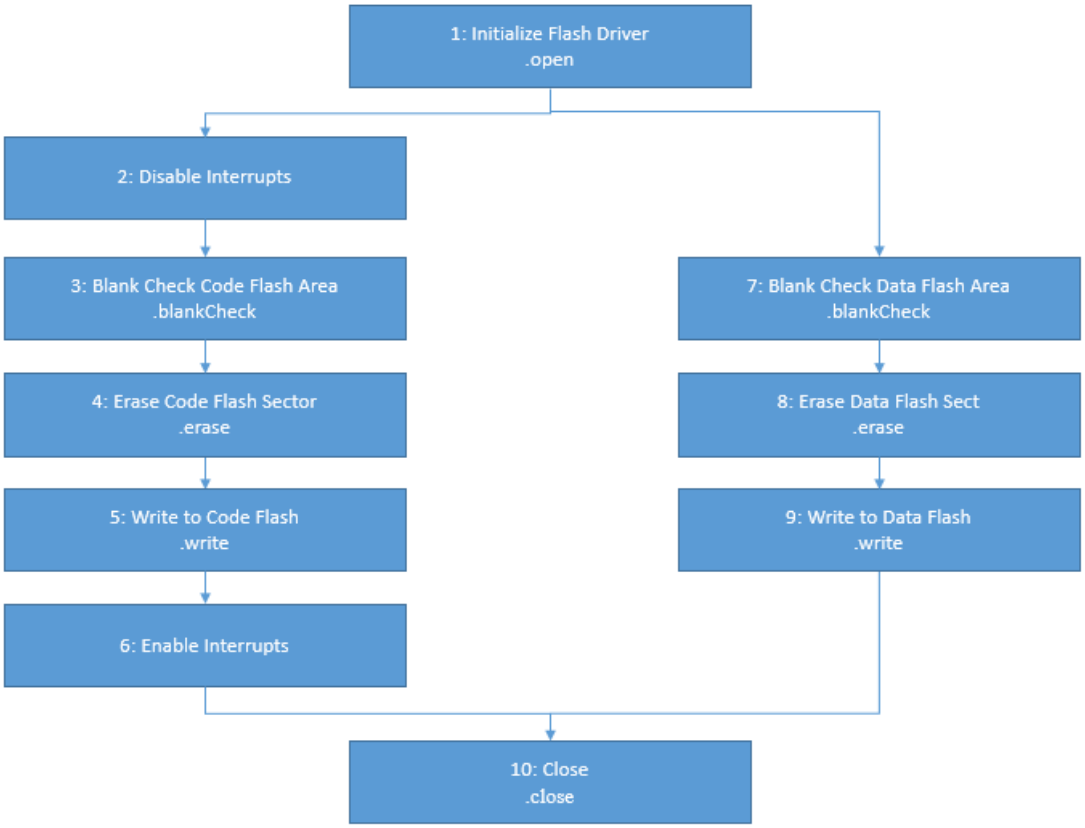


Figure 332: Flow Diagram of a Typical Flash HP HAL Module Application

4.2.20 FMI Driver

The FMI HAL module is a high-level API for applications that read records from the Factory MCU Information Flash Table. The FMI HAL module is implemented on r_fmi. The FMI HAL module uses the Flash Interface on the Synergy MCU.

4.2.20.1 FMI HAL Module Features

The FMI HAL module reads the FMIFRT (Factory MCU Information Flash Root Table) on a Synergy microcontroller, looking up the address of the start of the table in flash. The module sets the caller's pointer to the Product Information record from the table. This information may be used to determine the capabilities of features specific to this MCU package. Information available from the FMI HAL module includes:

- Product Information: Product Name, Package, Pin count, and Temperature range
- Product Features: Version major, Version minor, and Variant data
- Event Information such as Interrupts and Events

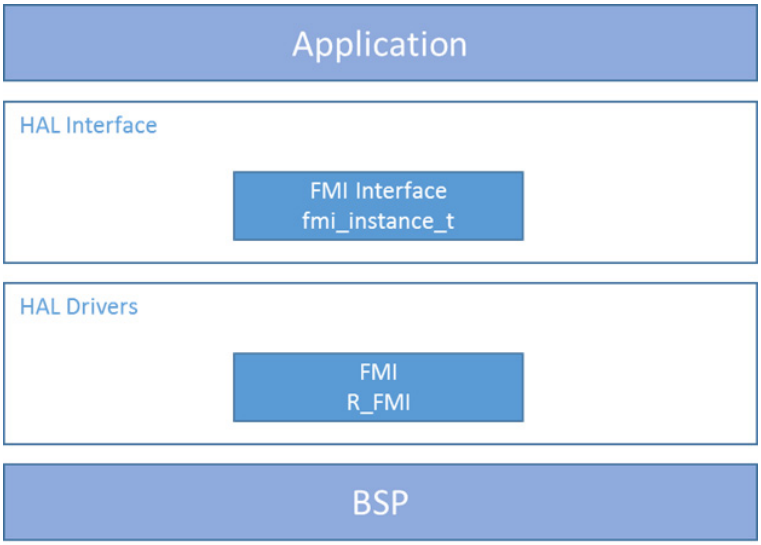


Figure 333: FMI HAL Module Block Diagram

4.2.20.2 FMI HAL Module APIs Overview

The FMI HAL module defines an API for accessing the FMIFRT. The following table has a complete list of the available APIs, an API call example, and a brief description of each API. Following the API summary table is a table listing status return values.

FMI HAL Module API Summary

Function Name	Example API Call and Description
init	g_fmi.p_api->init(); Initialize the FMI base pointer.
productInfoGet	g_fmi.p_api->productInfoGet(&g_pp_product_info); Get product information record address into g_pp_product_info pointer.
uniqueIdGet	g_fmi.p_api->uniqueIdGet(&g_p_unique_id); Copy the unique ID into the g_p_unique_id pointer.

Function Name	Example API Call and Description
productFeatureGet	g_fmi.p_api->productFeatureGet(&g_ssp_feature, &g_feature_info); Get feature information and store it in g_feature_info pointer.
eventInfoGet	g_fmi.p_api->peventInfoGet(&g_ssp_feature,SSP_SIGNAL_GPT_COUNTER_OVERFLOW, &g_event_info); Get event information and store it in g_event_info pointer.
versionGet	g_fmi.p_api->versionGet(&g_p_version); Get the driver version based on compile time macros.

NOTE: Review the SSP User's Manual API References for the associated module to learn more about operations and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_INVALID_FMI_DATA	The FMI data table provided is not valid
SSP_ERR_IP_CHANNEL_NOT_PRESENT	Requested channel does not exist on this MCU
SSP_ERR_IP_UNIT_NOT_PRESENT	Requested unit does not exist on this MCU
SSP_ERR_INTERNAL	Requested feature is in a format not supported at this time
SSP_ERR_IRQ_BSP_DISABLED	Event information could not be found
SSP_ERR_ASSERTION	Caller's pointer is null
SSP_ERR_INVALID_FACTORY_FLASH	Factory flash is not valid

NOTE: Lower level drivers may return Common Error Codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.20.3 FMI HAL Module Operational Overview

The FMI HAL module retrieves the product information record address and populates the fmi_product_info_t structure using the productInfoGet API.

The FMI HAL module copies unique ID and populates the fmi_unique_id_t structure using the uniqueIdGet API.

The FMI HAL module gets feature information and populates the fmi_feature_info_t structure using the productFeatureGet API.

The FMI HAL module fetches event information and populates the fmi_event_info_t structure using the eventInfoGet API.

The FMI HAL module gets code version and API version in ssp_version_t structure using the versionGet API.

FMI HAL Module Important Operational Notes and Limitations

The fmi_product_info_t::unique_id is deprecated. It does not contain a unique ID if the factory MCU information is linked in by the application code. Use uniqueIdGet for the unique ID.

- For limitations of the FMI HAL Interface and its implementation, see the latest SSP release notes.
- The FMI Driver has been tested on the S7G2 (WS2) Synergy microcontroller family using the FMIFRT peripheral register. It is the only Synergy MCU that is currently programmed with data in the Factory MCU Information Table.
- For the S5D9 MCU, the uniqueIdGet API of the r_fmi module returns an error. There is no workaround.

4.2.20.4 Including the FMI HAL Module in an Application

This section describes how to include the FMI HAL module in an application using the SSP configurator.

NOTE: This process assumes that you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack.

To add the FMI to an application, simply add it to a thread using the Stacks Selection Sequence given in the following table. (The default name for the FMI is r_fmi0. This name can be changed in the associated **Properties** window.)

Flash Driver Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
r_fmi FMI Driver on r_fmi	**Threads > HAL/Common**	**New Stack> Driver> System> FMI Driver on r_fmi**

The following figure shows that the FMI Driver on r_fmi is automatically added to the HAL/Common Stack when a new project is created, since every design requires the message to be supplied by the FMI HAL module.

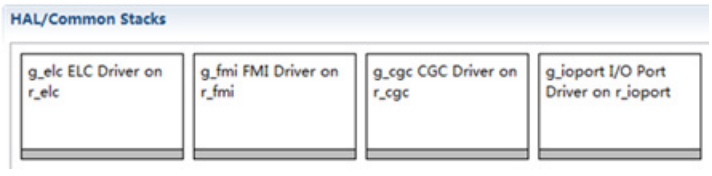


Figure 334: FMI HAL Module Stack

4.2.20.5 Configuring the FMI HAL Module

The FMI HAL module must be configured by the user for the desired operation. The available configuration settings and defaults for all the user accessible properties are given in the **Properties** tab within the SSP configurator and are shown in the following tables for easy reference.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the Configuration Table Settings given below. This can help orient you and provide a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the FMI HAL Module on r_fmi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Controls whether to include code for API parameter checking.
SSP MCU Information Symbol Name	g_fmi_data (Default)	This symbol maps to the base address where the factory flash table information is found. It should not be modified.
Part Number Mask	0xFE00 (Default)	Each bit represents one character in the Synergy part number, where the MSB is the first character in the part number ('R'). Set bits to ensure the part number in the MCU factory flash matches the part number in the SSP MCU Information. The default mask checks everything except operating temperature, software ID, and quality ID.
Name	g_fmi (Default)	Module instance name.

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

FMI HAL Module Clock Configuration

No specific clock configurations are required for the FMI HAL Driver.

FMI HAL Module Pin Configuration

No specific pin configurations are required for the FMI HAL Driver.

4.2.20.6 Using the FMI HAL Module in an Application

The typical steps in using the FMI in an application are:

1) Initialize the FMI using the init API. It is automatically initialized after Reset.
2) Use the productInfoGet API to get product information.
3) Use the uniqueIdGet API to get unique ID.
4) Use the productFeatureGet API to get feature information.
5) Use the eventInfoGet API to get event information.
6) Use the versionGet API to get driver version information.

The following figure illustrates these common steps in a typical operational flow diagram.

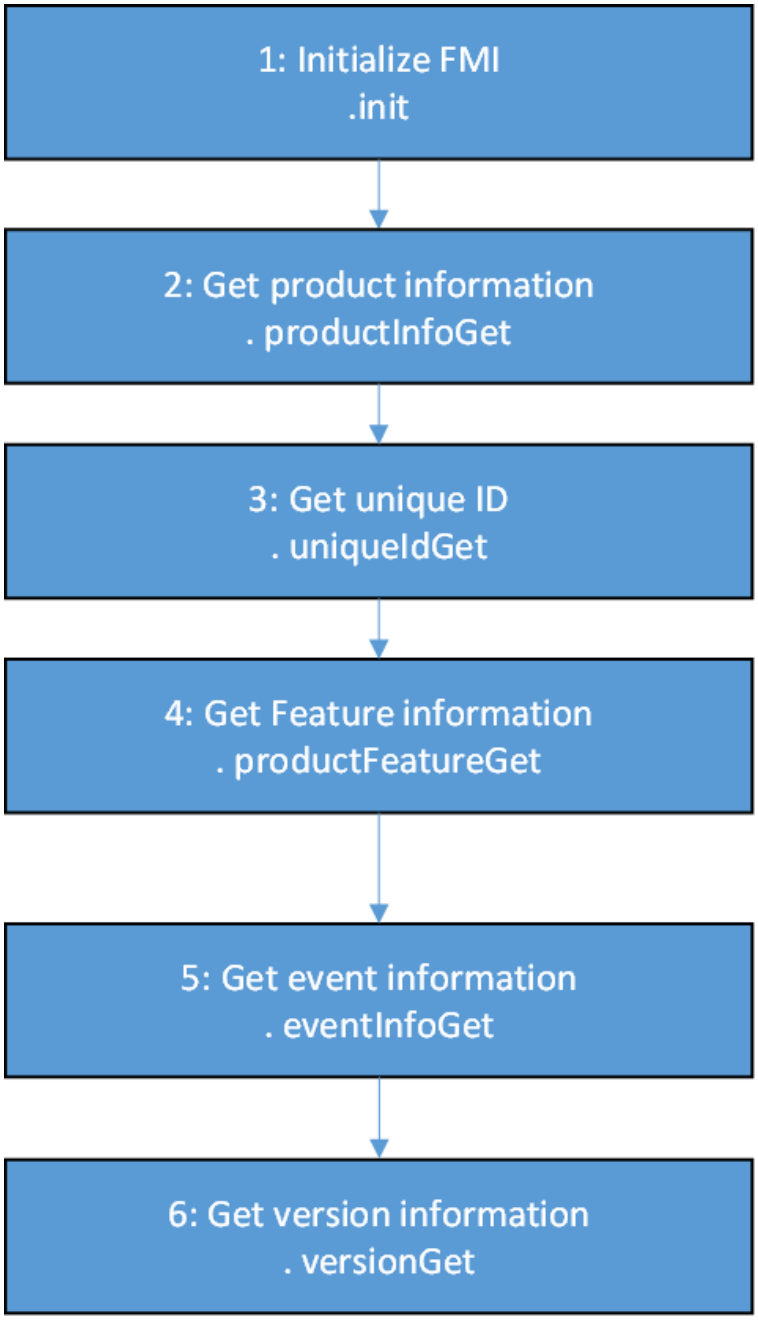


Figure 335: Flow Diagram of a Typical FMI HAL Module Application

4.2.21 GPT Driver

The General PWM Timer (GPT) HAL module provides high-level APIs for timer applications and is implemented on r_gpt. The GPT HAL module uses the GPT peripheral on the Synergy MCU. A user-defined callback can be created to respond to a timer event.

4.2.21.1 GPT HAL Module Features

The GPT HAL module configures a timer to a user-specified period. When the period elapses, any of the following events can occur:

- CPU interrupt that calls a user callback function, if provided
- Toggle a port pin
- Data transfer using DMAC/DTC if configured with Transfer Interface
- Starting of another peripheral if configured with events and peripheral definitions

General PWM Timer (GPT)

- Multiple Channels
  - S7G2: 32-bit x 14 channels
  - S5D9: 32-bit x 14 channels
  - S3A7: 32-bit x 10 channels
  - S124: 32-bit x 1 channel and 16-bit x 6 channels
- PCLKD as core clock
- Two I/O pins per channel
- Up/down-counting saw/triangle waves
- Two output compare and input capture registers
- Can be configured as a response to 8 Event Link Controllers (ELCs), and as an event in ELC

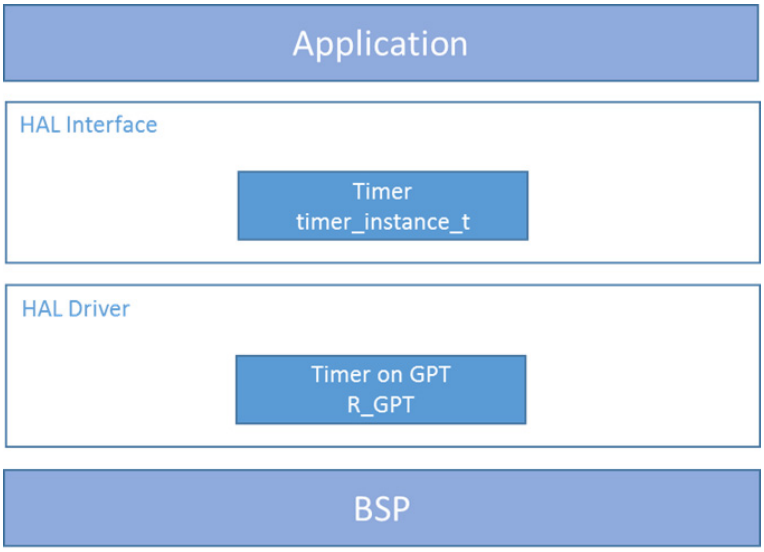


Figure 336: GPT HAL Module Block Diagram

4.2.21.2 GPT HAL Module APIs Overview

The GPT HAL module defines APIs for opening, closing, starting, and stopping timers. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

GPT HAL Module API Summary

Function Name	Example API Call and Description
open	g_timer0.p_api->open(g_timer0.p_ctrl, g_timer0.p_cfg) Initial configuration.
stop	g_timer0.p_api->stop(g_timer0.p_ctrl) Stop the counter.
start	g_timer0.p_api->start(g_timer0.p_ctrl) Start the counter.

Function Name	Example API Call and Description
reset	g_timer0.p_api->reset(g_timer0.p_ctrl) Reset the counter to the initial value.
counterGet	g_timer0.p_api->counterGet(g_timer0.p_ctrl, &value) Get current counter value and store it in provided pointer value.
periodSet	g_timer0.p_api->periodSet(g_timer0.p_ctrl, period, unit) Set the time until the timer expires.
dutyCycleSet	g_timer0.p_api->dutyCycleSet(g_timer0.p_ctrl, duty_cycle, duty_cycle_unit, pin) Sets the time until the duty cycle expires.
infoGet	g_timer0.p_api->infoGet(g_timer0.p_ctrl, &info) Get the time until the timer expires in clock counts and store it in provided pointer info.
close	g_timer0.p_api->close(g_timer0.p_ctrl) Allows driver to be reconfigured and may reduce power consumption.
versionGet	g_timer0.p_api->versionGet(&version) Get version and store it in provided pointer version.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Table Status Return Values

Name	Description
SSP_SUCCESS	Operation is successful.
SSP_ERR_ASSERTION	Parameter is NULL or configuration setting is not allowed.
SSP_ERR_IN_USE	The channel specified has already been opened.
SSP_ERROR_NOT_OPEN	The channel is not open.
SSP_ERR_INVALID_ARGUMENT	Invalid argument provided.

NOTE: Lower level drivers may return Common Error Codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.21.3 GPT HAL Module Operational Overview

The GPT HAL module configures a timer to a user-specified period. When the period elapses, the CPU can be interrupted, a port pin can be toggled, a transfer of data using the DMAC or DTC can be initiated, or another peripheral can be triggered to begin operation.

The following figure shows a flowchart for toggling a port pin or generating a CPU interrupt after a specified period.

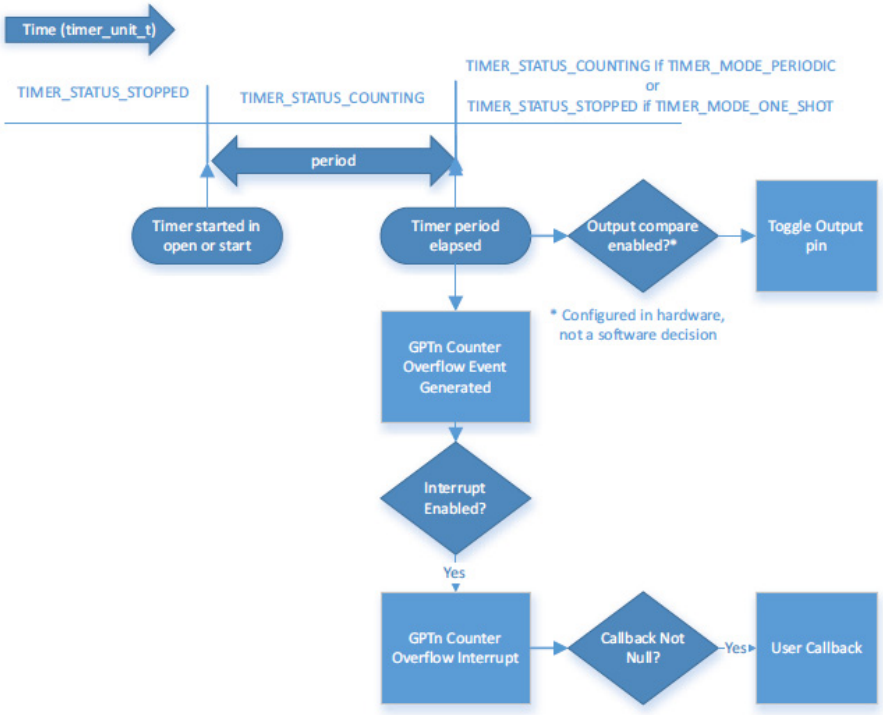


Figure 337: GPT Timer-Periodic or One-Shot Mode

Two different timer modules, the GPT and the AGT, are supported in the SSP. The following sections provide information on both modules so that the developer can compare and contrast the capabilities of each module for a particular application. For additional information on the AGT, refer to the AGT User's Guide.

The GPT module is recommended for most generic timer applications, but either module can be used for a basic timer functionality. The following use cases describe why one timer module would be preferred over the other.

Selecting the GPT Timer Module

The GPT module uses a high-resolution 32-bit counter that can only be clocked by PCLKD. There are more GPT channels than AGT channels on Synergy devices, so using GPT is less likely to cause a resource conflict.

Selecting the AGT Timer Module

The AGT module uses a 16-bit counter that can be clocked by PCLKB, LOCO, or Fsub. If clocked by LOCO or Fsub, the AGT interrupt can be used to wake the MCU from sleep modes. There are two channels, and channel 1 can be clocked by channel 0 underflow, effectively creating a 32-bit cascaded timer.

GPT HAL Module Important Operational Notes and Limitations

The maximum time period depends on the timer type and the input clock frequency.

- On a GPT with 32-bit resolution with PCLKD running at 120 MHz, the maximum period is approximately 36650 seconds, which is just over 10 hours (AGT Count Clock is PCLKD/1024).
- On a GPT with 16-bit resolution with PCLKD running at 32 MHz, the maximum period is approximately 2.09 seconds (AGT Count Clock is PCLKD/1024).

- On an AGT with 16-bit resolution with PCLKB running at 60 MHz, the maximum period is approximately 8.7 ms (AGT Count Clock is PCLKB/8).
- On an AGT with 16-bit resolution with Fsub running at 32 kHz, the maximum period is approximately 2.0 seconds (AGT Count Clock is AGTSCLK/128).

The AGT counter underflow interrupt for the selected channel used must be enabled in the BSP in the following situations:

1) To get a software interrupt when the timer period has elapsed.
2) To use one-shot mode

When the AGTn AGTI interrupt is enabled in the BSP, the corresponding ISR will be defined in the timer driver. The ISR will call a user callback function if one was registered in open.

NOTE: Interrupts may be skipped when used with the DTC peripheral with irq set to TRANSFER_IRQ_END.

GPT Output Timer Signal

If the timer output is configured (GTIOCA/B Output Enabled set to True), the output pin will start at the GTIOCA/B Stop Level and toggle every time the period elapses, beginning with the first time the period elapses after the timer is started.

In one-shot mode, the output is also configured to toggle when the timer starts counting. This generates a pulse - the timer toggles from the stop level when counting begins and toggles back to the stop level when counting ends.

Timer Period Calculation

The timer period is defined as the time until the timer expires. When output compare is used, the output pin will toggle once per period, so the traditional period (from rising edge to rising edge) is twice the period specified in the software.

Runtime period calculation based on the current clock settings is available from open and periodSet.

If the specified timer period is different than the raw counts, the period is calculated using the current timer clock frequency (see Configuring the GPT Clocks or Configuring the AGT Clocks). The current timer clock frequency is determined using systemClockFreqGet. This frequency will be used in the appropriate formula from the following table as clk_freq_hz.

Timer period calculation

Timer Units	Formula
TIMER_UNIT_PERIOD_NSEC	Counts = (period * clk_freq_hz) / 1000000000
TIMER_UNIT_PERIOD_USEC	Counts = (period * clk_freq_hz) / 1000000
TIMER_UNIT_PERIOD_MSEC	Counts = (period * clk_freq_hz) / 1000
TIMER_UNIT_PERIOD_SEC	Counts = (period * clk_freq_hz)
TIMER_UNIT_FREQUENCY_HZ	Counts = (clk_freq_hz) / period
TIMER_UNIT_FREQUENCY_KHZ	Counts = (clk_freq_hz) / 1000 * period

If the requested period is larger than the counter size (32-bit or 16-bit), the driver selects the smallest divisor that allows the result to fit in the counter size. If the counter value is larger than the counter size with the largest divisor (1024), an error code (SSP_ERR_INVALID_ARGUMENT) is returned.

Triggering DMAC/DTC with GPT

To trigger a transfer of data using the DMAC or DTC peripheral when the timer period elapses, configure the DMAC/DTC transfer with activation_source set to ELC_EVENT_GPTn_COUNTER_OVERFLOW (where n is the GPT channel number). See the DMAC or DTC guides for further information.

NOTE: If you use the timer in one-shot mode with the DTC, the entire transfer will complete before the interrupt stops the timer if irq is set to TRANSFER_IRQ_END. To generate only one transfer after the timer period elapses, set irq to TRANSFER_IRQ_EACH, or use the DMAC for the transfer.

Triggering ELC Events with GPT

The GPT timer can trigger the start of other peripherals. The ELC guide provides a list of all available peripherals.

Triggering DMAC/DTC with AGT

To trigger a transfer of data using the DMAC or DTC peripheral when the timer period elapses, configure the DMAC/DTC transfer with activation_source set to ELC_EVENT_AGTn_AGTI (where n is the AGT channel number). See Transfer Interface for further information.

NOTE: If you use the timer in one-shot mode with the DTC, the entire transfer completes before the interrupt stops the timer if irq is set to TRANSFER_IRQ_END. To generate only one transfer after the timer period elapses, set irq to TRANSFER_IRQ_EACH, or use the DMAC for the transfer.

Triggering ELC Events with AGT

The AGT timer can trigger the start of other peripherals. The ELC guide provides a list of all available peripherals listed in elc_peripheral_t. See events and peripheral definitions for further information.

Cascaded AGT Timers creating a 32-bit timer

AGT Channel 1 can be clocked by the AGT Channel 0 underflow, creating a cascaded 32-bit timer.

For GPT Power Down, the GPT module does not set the Module Stop bit (MSTP) for GPT in the close API. This is intentional because the GPT module stop bits control the power to multiple GPT channels, and the GPT module cannot know if other GPT modules are used in the application. Use the following procedure to set the module stop bit for GPT to reduce power consumption when no GPT channels are in use:

1) In the hardware manual for your MCU, go to the Low Power Modes chapter, and look in the Register Descriptions chapter for Module Stop Control Register D.
2) Look at MSTPD5 and MSTPD6. Identify which of these bits contains the channel used by your application.
3) Add the LPM driver to your project on the Threads tab of the Synergy Configuration tool:
4) New> Driver > Power > Low Power Modes Driver on r_lpm.
5) In the application code, confirm that no other GPT channels in the MSTP bit identified in step 2 are currently used by the application. If no channels are being used, use the LPM API to power down the set of GPT channels:

    a) To power down GPT channels controlled by MSTPD5 with an LPM module named g_lpm0, call: g_lpm0.p_api->moduleStop(LPM_MSTP_GPT_CH7_0); // Ignore the channel numbers in the enum value – this is for MSTPD5

    a) To power down GPT channels controlled by MSTPD6, call: g_lpm0.p_api->moduleStop(LPM_MST-P_GPT_CH13_8); // Ignore the channel numbers in the enum value – this is for MSTPD6

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.21.4 Including the GPT HAL Module in an Application

This section describes how to include the GPT HAL module in an application using the SSP configurator.

NOTE: This section assumes that you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack.

To add the GPT HAL module to an application, simply add it to a thread using the Stacks Selection Sequence given in the following table. (The default name for the Timer Driver is g_timer0 is indicated. This name can be changed in the associated Properties window.)

GPT Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_timer0 Timer Driver on r_gpt	Threads -> HAL/Common	New Stack> Driver> Timers> Timer Driver on r_gpt

When the GPT HAL module is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower level modules. Any drivers that need additional configuration information will be box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

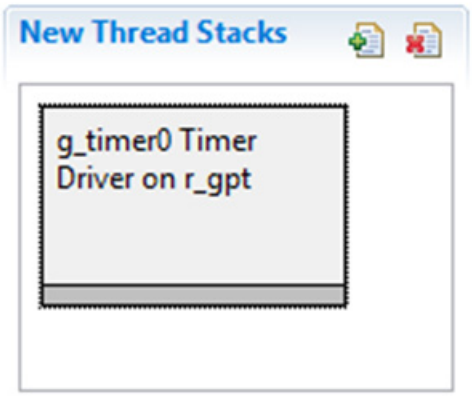


Figure 338: GPT HAL Module Stack

4.2.21.5 Configuring the GPT HAL Module

The GPT HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections (such as interrupts or operating modes) which must be configured for lower level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the properties window of the associated module. Simply select the indicated module and then view the

properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+.) This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the following Configuration Table Settings given. These property settings will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Table Configuration Settings for the GPT HAL Module on r_gpt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Parameter selection.
Name	g_timer0	Module name.
Channel	0	Physical hardware channel. 0-13 for S7G2, 0-9 for S3A7, 0-6 for S124
Mode	Periodic, One Shot, PWM (Default: Periodic)	Mode selection. NOTE: One Shot functionality is not available in the GPT hardware, so it is implemented in software by stopping the timer in the ISR called when the period expires. For this reason, ISR's must be enabled for one-shot mode even if the callback is unused.
Period Value	10	Period value selection. See the Timer Period Calculation section earlier in this document.

ISDE Property	Value	Description
Period Unit	Raw Counts, Nanoseconds, Microseconds, Milliseconds, Seconds, Hertz, Kilohertz (Default: Milliseconds)	Period unit selection. See the Timer Period Calculation section earlier in this document.
Duty Cycle Value	50	Duty cycle value selection.
Duty Cycle Unit	Unit Raw Counts, Unit Percent, Unit Percent x 1000 (Default: Unit Raw Counts)	Duty cycle unit selection.
Auto Start	True, FALSE (Default: True)	Auto start selection. Set to true to start the timer after configuring or false to leave the timer stopped until start is called.
GTIOCA Output Enabled	True, False (Default: False)	GTIOCA output enabled selection. Set to true to output the timer signal on a port pin configured for GPT. Set to false for no output of the timer signal.
GTIOCA Stop Level	Pin Level Low, Pin Level High, Pin Level Retained (Default: Pin Level Low)	Controls output pin level when the timer is stopped.
GTIOCB Output Enabled	True, False (Default: False)	GTIOCB output enabled selection.
GTIOCB Stop Level	Pin Level Low, Pin Level High, Pin Level Retained (Default: Pin Level Low)	GTIOCB stop level selection.
Callback	NULL	Callback selection. A user callback function can be registered in open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses.

ISDE Property	Value	Description
Interrupt Priority	Priority 0(highest)-15(lowest), Disabled (Default: Disabled)	Interrupt priority selection.

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for a module can be desirable. For example, it might be useful to select a different clock source than the default. The configurable properties for the lower level stack modules are given in the following sections for completeness and as a reference.

GPT HAL Module Clock Configuration

The GPT timer is clocked based on the PCLKD frequency. You can set the PCLKD frequency using the clock configurator in the ISDE Configuring Clocks tab, or the CGC Interface at run-time.

GPT HAL Module Pin Configuration

The GPT peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The first table lists the method used to select pins within the SSP configuration window and the following table lists an example selection for the associated pins.

NOTE: The Operation Mode selection mode determines what peripheral signals are available and what MCU pins are required.

Pin Selection Sequence for GPT HAL Driver

Resource	ISDE Tab	Pin selection Sequence
GPT	Pins	Select Peripherals > Timer: GPT>GPT0

NOTE: The above selection sequence assumes GPT0 is the desired hardware target for the driver.

Pin Configuration Settings for GPT HAL Driver

Property	Value	Description
Pin Group Selection	Mixed, _A Only, _B Only (Default: Mixed)	Select pin group mapping.

Property	Value	Description
Operation Mode	Disabled, GTIOCA or GTIOCB, GTIOCA and GTIOCB (Default: Disabled)	Select timer operation mode.
GTIOCA:	None, P300, P512 (Default: P512)	GTIOCA Pin.
GTIOCB:	None, P108, P511 (Default: P511)	GTIOCB Pin.

NOTE: The above example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.21.6 Using the GPT Module in an Application

The typical steps in using the GPT HAL module in an application are:

1) Initialize the GPT HAL module using the open API.
2) Start the GPT HAL module by calling the start API.
3) Respond to the timer callback as needed (user code).

NOTE: The GPT period and duty cycle parameters can be reconfigured based on the application needs.

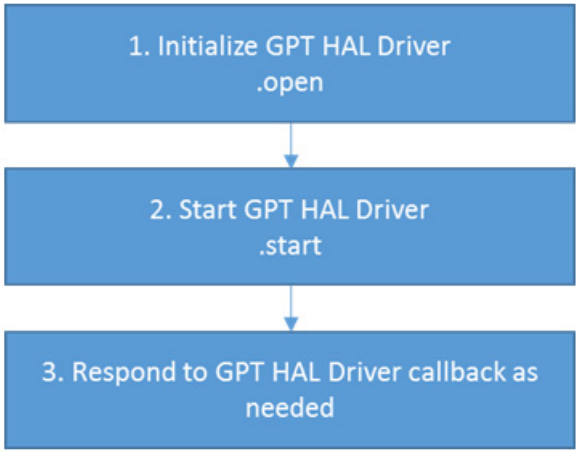


Figure 339: Typical GPT HAL Module Application Flow Chart

4.2.22 I2C SCI Driver

The I2C SCI HAL module implements provides high-level APIs for I2C Master applications with the module implemented on r_sci_i2c. The I2C SCI HAL module uses the SCI peripheral on the Synergy MCU device. Callbacks are provided for transmit complete and receive complete.

4.2.22.1 I2C SCI HAL Module Features

- Support for I2C SCI operations
- Supports the following operations with a slave I2C SCI device
  - Read
  - Write
  - Reset
- Callback support
  - Transfer aborted
  - Transmit complete (number of bytes transmitted provided)
  - Receive complete (number of bytes received provided)

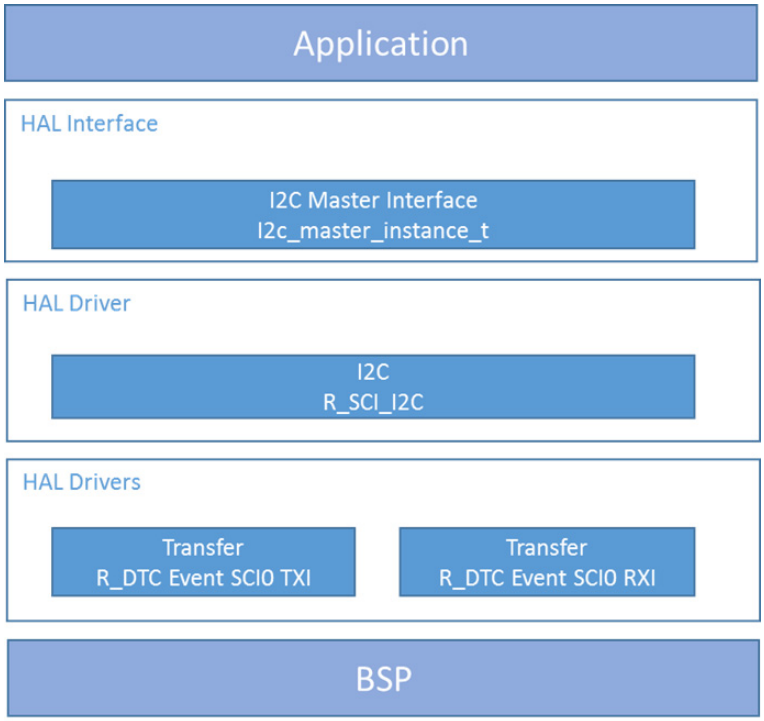


Figure 340:I2C SCI HAL Module Block Diagram

4.2.22.2 I2C SCI HAL Module APIs Overview

The I2C SCI HAL module defines APIs for reading and writing using a master I2C device. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

I2C SCI HAL Module API Summary

Function Name	Example API Call and Description
.open	g_i2c.p_api->open(g_i2c.p_ctrl, g_i2c.p_cfg); Open the instance and initialize the hardware.
.close	g_i2c.p_api->close(g_i2c.p_ctrl); Closes the driver and releases the I2C device.

Function Name	Example API Call and Description
.read	g_i2c.p_api->read(g_i2c.p_ctrl, &destination, bytes, restart); Performs a read operation on an I2C device.
.write	g_i2c.p_api->write(g_i2c.p_ctrl, &destination, bytes, restart); Performs a write operation on an I2C device.
.reset	g_i2c.p_api->reset(g_i2c.p_ctrl); Reset the peripheral.
.versionGet	g_i2c.p_api->versionGet(&version); Retrieve the API version with the version pointer.

NOTE: For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the *SSP User's Manual* API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_IN_USE	Attempted to open an already open device instance.
SSP_ERR_ABORTED	Device was closed while a transfer was in progress.
SSP_ERR_INVALID_RATE	The requested rate cannot be set
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Device was not even opened.
SSP_ERR_IRQ_BSP_DISABLED	Event information could not be found.

NOTE: Lower-level drivers may return common error codes. Refer to the *SSP User's Manual* API References for the associated module for a definition of all relevant status return values.

4.2.22.3 I2C SCI HAL Module Operational Overview

The I2C SCI HAL module supports transactions with an I2C slave device. Callbacks are provided to interrupt the CPU when a transmission has been completed or aborted, or receive completed. The I2C SCI HAL module invokes the callback with the argument i2c_callback_args_t, indicating the number of received or transmitted bytes in buffer, pointer to user provided context, and the event 2c_event_t.

I2C SCI HAL Module Important Operational Notes and Limitations

I2C SCI HAL Module Operational Notes

Interrupts

- The I2C interrupts (SCI Error (EEI), Receive Buffer Full (RXI), Transmit Buffer Empty (TXI), and Transmit End (TEI)) for the selected channel must be enabled in the board support package (BSP), without consideration of whether the user wants to use callbacks.
- Setting the interrupts to different priority levels could result in improper operation.

IIC Rate Calculation

- The I2C SCI HAL module calculates the internal baud-rate setting based on the configured transfer rate and passes this to open. The closest possible baud rate that can be achieved (less than or equal to the requested rate) at the current PCLKB settings is calculated and used.
- If a valid clock rate could not be calculated, an error is returned.

Triggering DMAC/DTC with the IIC

- DTC transfer support is added by default in the configurator, this can be removed for CPU transfer cases. The DTC is configured in the module. No user configuration is required for this.
- DMA transfer is not supported.

Triggering ELC Events with the IIC

- The I2C SCI HAL module can trigger the start of other peripherals. See the ELC User Guide for further information.

Multiple Devices on the Bus

- An API slaveAddressSet() can be used to switch the slave devices without reconfiguring (no need to close and open) the bus in an application for using multiple slave devices on the same bus.
- Control instance and bus configuration remains the same, but the slave address and addressing mode changes.
- Other ways of communication with different slave devices can be used by closing the first device and opening a new device. This is recommended if slaves are having different configurations.
- Applications using multiple devices connected on the same channel need to define the following macro in the pre-processor settings of your project (or the project may not build correctly):SSP_SUPPRESS_ISR_<device_name>Where <device_name> is the name of the additional device connected to the same channel.

The I2C SCI HAL module in IRQ mode may not work with certain slave devices; you need to enable DTC transfer mode to work with such devices.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.22.4 Including the I2C SCI HAL Module in an Application

This section describes how to include the I2C SCI HAL module in an application using the SSP configurator.

NOTE: It is assumed that you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these steps, refer to the first few chapters of the SSP User's Manual to learn how to manage these important steps in creating SSP-based applications.

To add the I2C SCI Driver to an application, simply add it to a thread using the stacks selection sequence in the following table. (The default name for the I2C SCI HAL Driver is g_i2c0. This name can be changed in the associated Properties window.)

I2C SCI HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_i2c0 I2C Master Driver on r_sci_i2c	Threads	New Stack> Driver> Communications> I2C Master Driver on r_sci_i2c

When the I2C SCI HAL Module on r_sci_i2c is added to the thread stack as shown in the following figure; the configurator automatically adds any needed lower-level drivers. Any drivers that need additional configuration information are box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

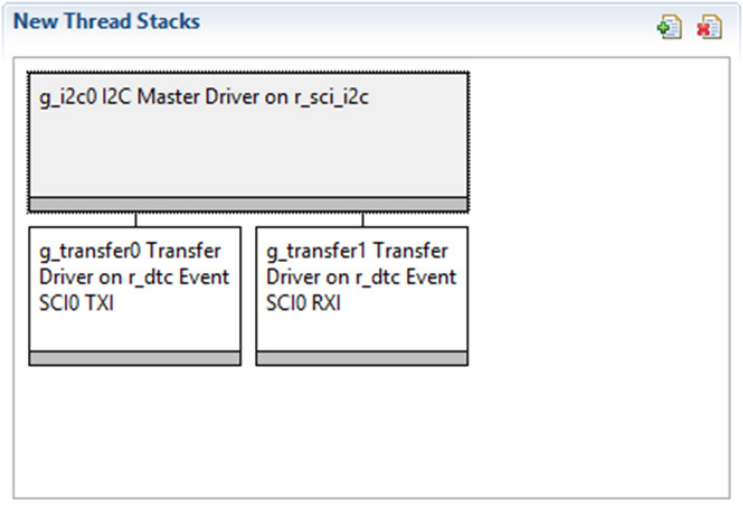


Figure 341: I2C SCI HAL Module Stack

4.2.22.5 Configuring the I2C SCI HAL Module

The I2C SCI HAL module must be configured by you for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or

operating modes, which must be configured for lower-level modules for successful operation. Only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP Configurator and are shown in the table later in this document for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE includes an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE and create the module and explore the property settings in parallel with looking over the following configuration table settings. This step helps to orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the I2C SCI HAL Module on r_sci_i2c

ISDE Property	Value	Description
Parameter Checking	Enabled, Disabled, BSP (Default: BSP)	Selects if code for parameter checking is to be included in the build
Name	Default: g_i2c0	Module name
Channel	0	Channel Number
Rate	Standard, Fast mode (Default: Standard)	Rate selection
Slave Address	Default: 0x00	Slave address
Address Mode	7-bit, 10-bit (Default: 7-bit)	Address mode
SDA Output Delay (nanoseconds)	300	SDA output delay value
Bit Rate Modulation Enable	Enabled, Disable (Default: Enable)	Bit rate modulation setting
Callback	Default: NULL	User defined callback

ISDE Property	Value	Description
Receive Interrupt Priority	Priority 0 (highest)-15 (lowest), Disabled (Default: Priority 2)	Receive Interrupt priority
Transmit Interrupt Priority	Priority 0 (highest)-15 (lowest), Disabled (Default: Priority 2)	Transmit Interrupt priority
Transmit End Interrupt Priority	Priority 0 (highest)-15 (lowest), Disabled (Default: Priority 2)	Transmit end interrupt priority

NOTE: The example settings and defaults are for a project using the Synergy S7G2 Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for stack modules can be desirable. For example, it might be useful to select different slave addresses or address modes. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

NOTE: Most property settings for modules are fairly intuitive and usually can be determined by inspection of the associated Properties window from the SSP configurator.

Transfer Driver on r_dtc Event SCI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled (Default: Disabled)	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section selection
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection

ISDE Property	Value	Description
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 TXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0(highest)-15(lowest), Disabled (Default: Disabled)	Interrupt priority for ELC Event

NOTE: The example settings and defaults are for a project using the Synergy S7 MCU Group. Other MCUs may have different default values and available configuration settings.

Transfer Driver on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled (Default: Disabled)	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section selection
Name	g_transfer1	Module name

ISDE Property	Value	Description
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest) -15 (lowest), Disabled (Default: Disabled)	Interrupt priority for ELC Event

I2C SCI HAL Module Clock Configuration

The SCI peripheral module uses PCLKB as its clock source. The actual I2C transfer rate is calculated and set internally by the driver (depending on the selected transfer rate). If the PCLKB is configured in such a manner that the selected internal rate cannot be achieved, an error is returned when initializing the driver.

I2C SCI HAL Module Pin Configuration

The SCI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the pins.

NOTE: For some peripherals, the operation mode selection determines what peripheral signals are available and thus what MCU pins are required for the I2C SCI HAL Module.

Pin Selection Sequence for the I2C SCI HAL Module

Resource	ISDE Tab	Pin selection Sequence
SCI	Pins	Select Peripherals > Connectivity: SCI > SCI0

NOTE: The selection sequence assumes SCI0 is the desired hardware target for the driver.

Pin Configuration Settings for the I2C SCI HAL Module

Property	Value	Description
Pin Group Selection	_A only, _B only, Mixed (Default: _A only)	Pin group selection
Operation Mode	Enabled, Disabled (Default: Disabled)	Enable or disable peripheral module
SDA	None, P401, P407 (Default: None)	SDA Pin
SCL	None, P400, P204 (Default: None)	SCL Pin

NOTE: The example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.22.6 Using the I2C SCI HAL Module in an Application

The typical steps in using the I2C SCI HAL module in an application are:

1) Initialize and open the I2C SCI HAL module using the open API.
2) Transfer data to the slave using the write API.
3) Receive data from the slave using the read API.
4) Operate on the received data as needed by the application.
5) Reset the instance with the reset API (if needed).
6) Perform transactions with slave device (if needed).
7) Close the channel using the close API.

NOTE: Optionally change the slave address using the slaveAddressSet API and then perform read/write transactions with the new slave device.

The following figure illustrates these common steps within a typical operational flow diagram.

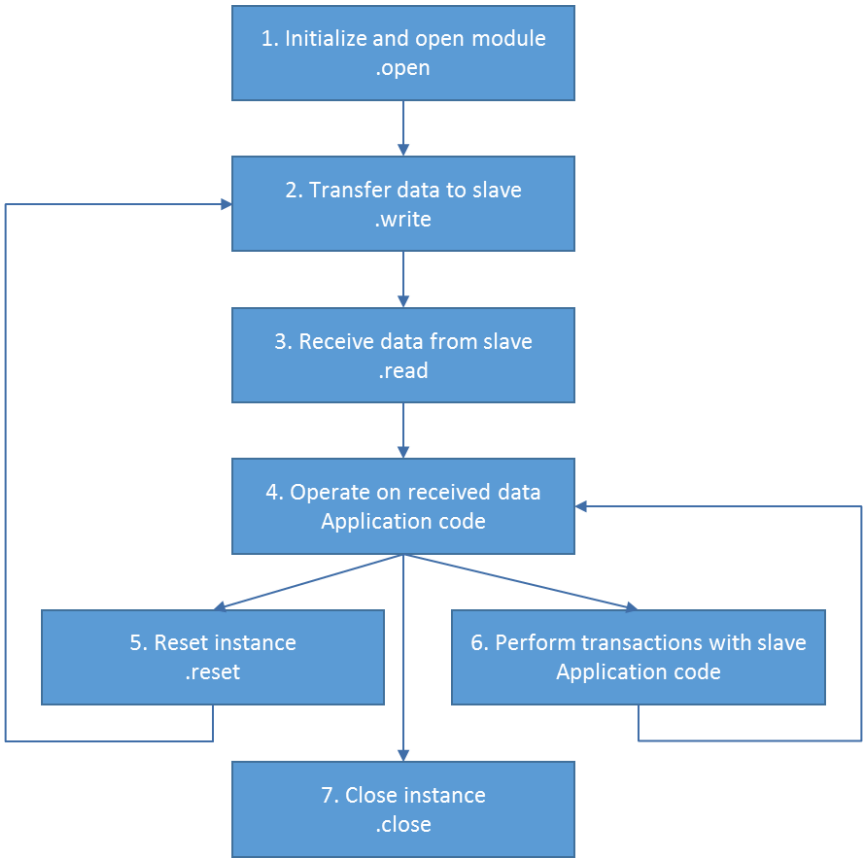


Figure 342: Flow Diagram of a Typical I2C SCI HAL Module Application

4.2.23 I2C Master Driver

The I2C Master HAL module provides high-level APIs for I2C Master applications and is implemented on r_riic. The I2C Master RIIC module uses the IIC peripheral on the Synergy MCU. Callbacks are provided for transmit complete and receive complete event notification.

4.2.23.1 I2C Master HAL Module Features

- Support for I2C RIIC operations
  - Standard (up to 100 kHz)
  - I2C fast-mode (up to 400 kHz)
  - I2C fast-mode plus (up to 1 MHz on channel 0 (SCL0-A, SDA0-A) of S7G2 and S5D9 MCU families)

- Initialization of the RIIC module
- Read from a slave device
- Write to a slave device
- Reset the I2C peripheral
- Set the address of the slave device
- Callback support
  - Transfer aborted
  - Transmit complete (number of bytes transmitted provided)
  - Receive complete (number of bytes received provided)

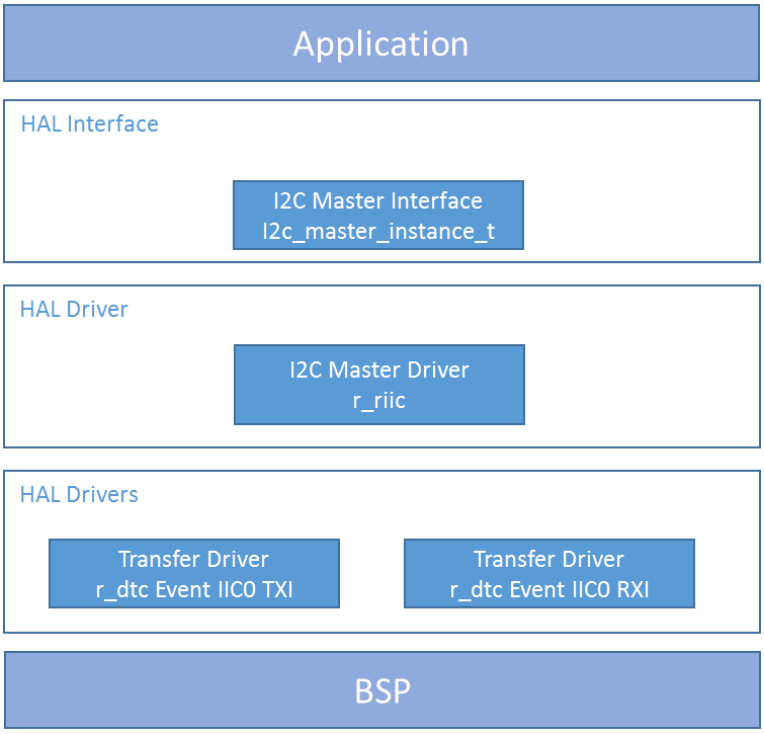


Figure 343:I2C Master HAL Module Block Diagram

4.2.23.2 I2C Master HAL Module APIs Overview

The I2C Master HAL module defines APIs including reading and writing using a master I2C device. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows.

I2C Master HAL Module API Summary

Function Name	Example API Call and Description
.open	g_i2c.p_api->open(g_i2c.p_ctrl, g_i2c.p_cfg); Open the instance and initialize the hardware.
.close	g_i2c.p_api->close(g_i2c.p_ctrl); Closes the driver and releases the I2C device.
.read	g_i2c.p_api->read(g_i2c.p_ctrl, &destination, bytes, restart); Performs a read operation on an I2C device.
.write	g_i2c.p_api->write(g_i2c.p_ctrl, &destination, bytes, restart); Performs a write operation on an I2C device.
.reset	g_i2c.p_api->reset(g_i2c.p_ctrl); Reset the peripheral.
.versionGet	g_i2c.p_api->versionGet(&version); Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_INVALID_POINTER	Pointer is NULL
SSP_ERR_IN_USE	Attempted to open an already open device instance.

Name	Description
SSP_ERR_ABORTED	Device was closed while a transfer was in progress.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value
SSP_ERR_INVALID_RATE	The requested rate cannot be set

NOTE: Lower-level drivers may return common error Codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.23.3 I2C Master HAL Module Operational Overview

The I2C master HAL module supports transactions with an I2C Slave device. Callbacks are provided to interrupt the CPU when a transmission or receive has been completed. The RIIC HAL module invokes the callback with the argument i2c_callback_args_t, indicating the number of received or transmitted bytes in buffer, pointer to user provided context, and the event i2c_event_t.

I2C Master HAL Module Important Operational Notes and Limitations

Interrupts

- The RIIC error (EEI), receive buffer full (RXI), transmit buffer empty (TXI), and transmit end (TEI) interrupts for the selected channel used must be enabled in the properties of the selected device irrespective of whether the user wants to use callbacks.
- Set equal priority levels for all the interrupts mentioned above. Setting the interrupts to different priority levels could result in improper operation.

IIC Rate Calculation

- The I2C Master module calculates the internal baud-rate setting based on the configured transfer rate and passed to open. The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current PCLKB settings is calculated and used.
- If a valid clock rate could not be calculated, an error is returned.

Triggering DMAC/DTC with the IIC

- DTC transfer support added by default in the configurator. This can be removed for CPU transfer cases. The DTC is configured in the module. No user configuration is required for this.
- DMA transfer is not supported.

Triggering ELC Events with the IIC

- The I2C Master module can trigger the start of other peripherals. See events and peripheral definitions in the ELC User Guide for further information.

Multiple Devices on the Bus

- If multiple devices are connected on the same bus, only one device can be opened at a time.

Usage of Restart Condition

- Passing value 'true' to the restart parameter of write/read API will generate restart condition after specified number (length) of bytes. Master will continue to hold the bus busy (low) without timeout so that current master can trigger next write/read API.

Multi-Master Support

- If multiple masters are connected on the same bus, the I2C Master is capable of detecting bus busy state before initiating the communication.
- Any of the supported IIC channel can be configured for either Master or Slave mode operation but not for both.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.23.4 Including the I2C Master HAL Module in an Application

This section describes how to include the I2C RIIC HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the I2C Master Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the I2C RIIC HAL Module is g_i2c0. This name can be changed in the associated Properties window).

I2C Master HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_i2c0 I2C Master Driver on r_riic	Threads	New Stack> Driver> Communications> I2C Master Driver on r_riic

NOTE: When the I2C RIIC HAL module on r_riic is added to the thread stack as shown in the figure below, the configurator automatically adds any needed lower-level drivers.

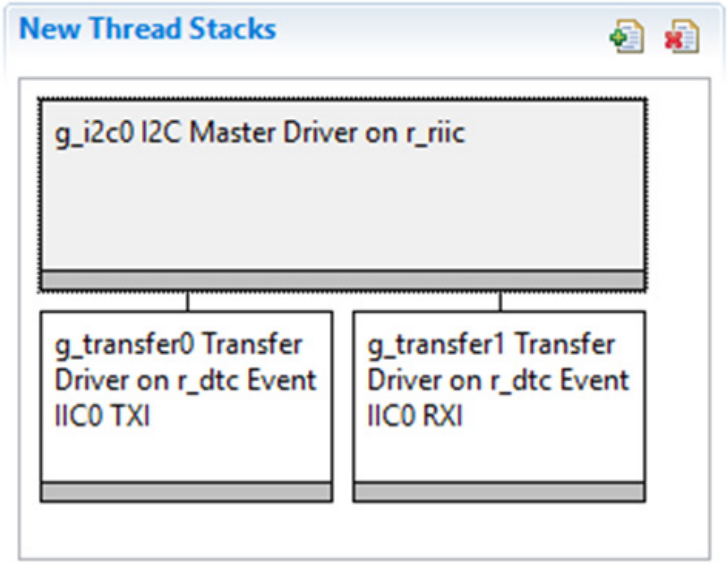


Figure 344: I2C Master HAL Module Stack

4.2.23.5 Configuring the I2C Master HAL Module

The I2C RIIC HAL module must be configured by you for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operations. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the table later in this document for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority. This configuration setting is available with the Properties window of the associated module. Simply select the indicated module and then view the Properties window. The interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+.) This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE and create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the I2C Master HAL Module on r_riic

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.
Name	g_i2c0	Module name.
Channel	0, 1, or 2	Specify the IIC channel to be used with this configuration.
Rate	Standard, Fast-mode, Fast-mode Plus Default: Standard	Standard, Fast, and Fast-plus. (See IIC Rate Calculation.)
Slave Address	0x00	Set the address of the slave device the I2C master will be communicating with.
Address Mode	7-Bit, 10-Bit Default: 7-Bit	Only 7-bit addresses are currently supported.
Callback	NULL	A user callback function can be registered in open. If this callback function is provided, it will be called from the interrupt service routine (ISR) for each of the conditions defined in i2c_event_t. Warning: Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Receive interrupt priority selection.

ISDE Property	Value	Description
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Error interrupt priority selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the DTC HAL Module on r_dtc Event IIC0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Enabled	Software start selection.

ISDE Property	Value	Description
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event IICI0 TXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the RTC HAL Module on r_dtc Event IIC0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection.
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table.
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event IIC0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection

ISDE Property	Value	Description
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for the module can be desirable. For example, it might be useful to select different slave addresses or address modes. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

I2C Master HAL Module Clock Configuration

The IIC peripheral module uses PCLKB as its clock source. The actual I2C transfer rate will be calculated and set internally by the driver depending on the selected transfer rate. If the PCLKB is configured in such a manner that the selected internal rate cannot be achieved, an error will be returned when initializing the driver.

I2C Master HAL Module Pin Configuration

The IIC peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the pins.

NOTE: For some peripherals, the operation mode selection determines what peripheral signals are available and thus what MCU pins are required.

Pin Selection Sequence for I2C Master HAL Module

Resource	ISDE Tab	Pin selection Sequence
IIC	Pins	Select Peripherals > Connectivity: IIC > IIC0

NOTE: The selection sequence assumes IIC0 is the desired hardware target for the driver.

Pin Configuration Settings for I2C Master HAL Module

Pin Configuration Property	Value	Description
Pin Group Selection	_A only, _B only, Mixed (Default: _A only)	Pin group selection
Operation Mode	Enabled, Disabled (Default: Disabled)	Enable or disable peripheral module
SDA	None, P401, P407 (Default: None)	SDA Pin
SCL	None, P400, P204 (Default: None)	SCL Pin

NOTE: The example settings are for a project using the Synergy S7G2 MCU. Other Synergy MCUs may have different available pin configuration settings.

4.2.23.6 Using the I2C Master HAL Module in an Application

The typical steps in using the I2C RIIC HAL Module in an application are:

1) Initialize and open the I2C RIIC HAL Module using the open API
2) Transfer data to the slave using the write API
3) Receive data from the slave using the read API
4) Reset the instance with the reset API (if needed)
5) Close the channel using the close API

NOTE: If the application wants to switch the device without opening and closing the bus, use the slaveAddressSet API where g_i2c.p_ctrl is the same control instance that was used in the last opened device. The module will use the same bus configuration to communicate with the new device. In this case, you can use the same control instance to communicate with different slave devices by setting the new slave address and calling the read or write APIs.

These common steps to communicate with a slave device are illustrated in a typical operational flow in the figure below:

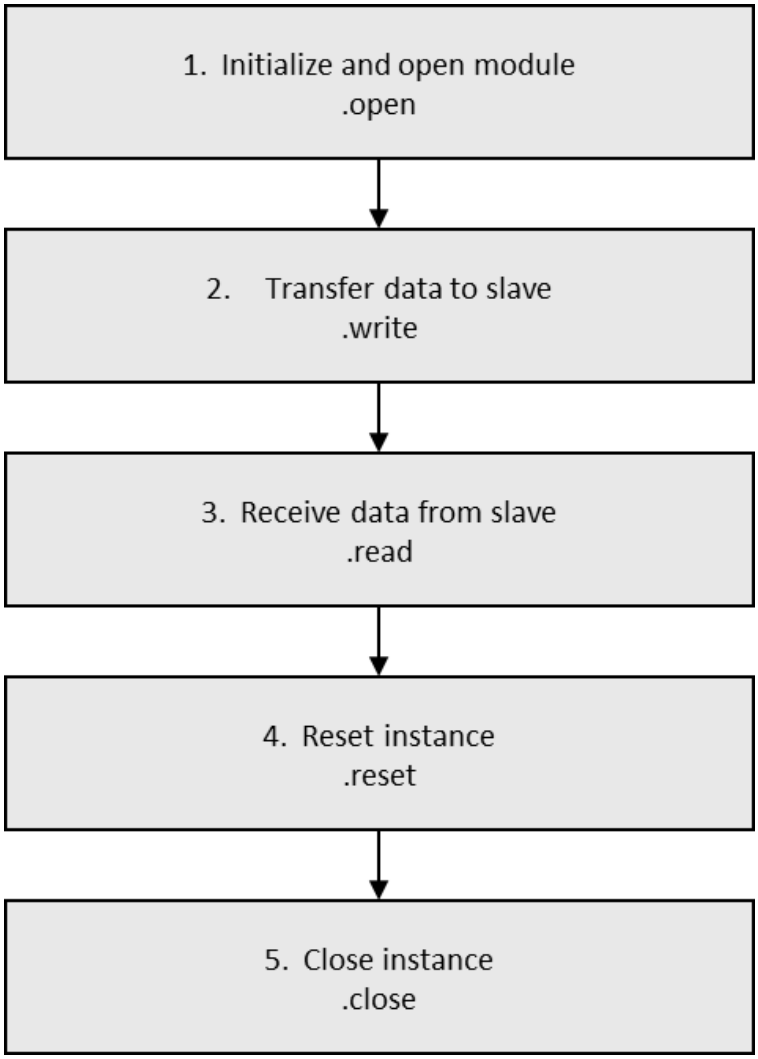


Figure 345: Flow Diagram of a Typical I2C Master HAL Module Application

4.2.24 I2C Slave Driver

The I2C Slave HAL Module provides high-level API for I2C slave applications and is implemented on r_riic_slave. The I2C Slave Module uses the RIIC peripheral on the Synergy MCU. Callbacks are provided for transmit complete and receive complete notification.

4.2.24.1 I2C RIIC Slave HAL Module Features

- Support for I2C Slave operations
- Support transactions with a I2C master device
  - Read

  - Write
- Callback support
  - Transmit complete (number of bytes transmitted provided)
  - Receive complete (number of bytes received provided)

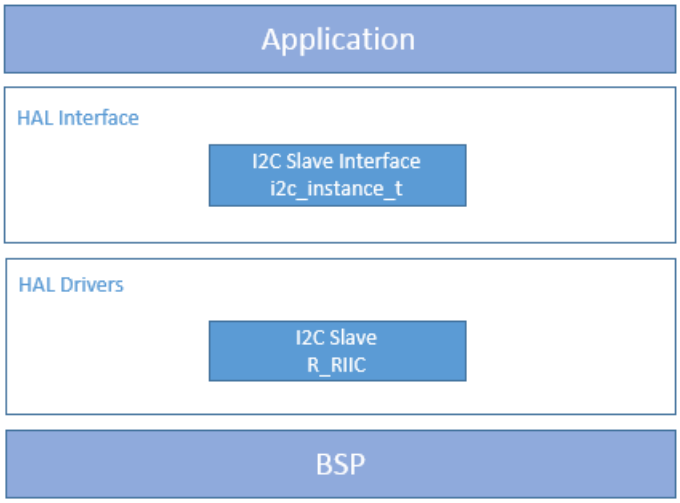


Figure 346:I2C RIIC Slave HAL Module Organization, Options and Stack Implementations

4.2.24.2 I2C RIIC Slave HAL Module APIs Overview

The I2C RIIC Slave HAL Module defines APIs for reading and writing to a master I2C device. A complete list of the available APIs, an example API call, and a short description of each can be found in the table below. A table of status return values follows.

I2C RIIC Slave HAL Module API Summary

Function Name	Example API Call and Description
.open	g_i2c.p_api->open(g_i2c.p_ctrl, g_i2c.p_cfg); Open the instance and initialize the hardware..
.close	g_i2c.p_api->close(g_i2c.p_ctrl); Closes the driver and releases the I2C device.

Function Name	Example API Call and Description
.masterWriteSlaveRead	g_i2c.p_api->masterWriteSlaveRead(g_i2c.p_ctrl, &source, bytes); Performs a read operation on an I2C device.
.masterReadSlaveWrite	g_i2c.p_api->masterReadSlaveWrite(g_i2c.p_ctrl, &source, bytes); Performs a write operation on an I2C device.
.versionGet	g_i2c.p_api->versionGet(&version); Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_INVALID_POINTER	Pointer is NULL
SSP_ERR_IN_USE	Attempted to open an already open device instance.
SSP_ERR_ABORTED	Device was closed while a transfer was in progress.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value

NOTE: Lower level drivers may return Common Error Codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.24.3 I2C RIIC Slave HAL Module Operational Overview

The I2C RIIC Slave HAL Module supports transfers to an I2C Master device. Callbacks are provided to interrupt the CPU when a transmission has been completed, or when a receive has been completed.

I2C RIIC Slave HAL Module Important Operational Notes and Limitations

- The RIIC Error (EEI), Receive Buffer Full (RXI), Transmit Buffer Empty (TXI) and Transmit End (TEI) interrupts for the selected channel used must be enabled in the BSP irrespective of whether the user wants to use callbacks.
- Set equal priority levels for all the interrupts mentioned above. Setting the interrupts to different priority levels could result in improper operation.
- If you are using RIIC and RIIC_Slave modules on the same board, we suggest setting equal interrupt priority for TXI, TEI, and EI and set RXI interrupt priority higher than these.

This is the initial version of I2C RIIC Slave Driver with only basic functionality implemented. The following limitations are known:

1) The Driver will lock up the I2C bus when any of the following operations occur:
   - When Master is Reading 'M' bytes and Slave has 'N' Bytes available to write. In this case (M < N) or (M > N).
   - When Master and Slave are both writing to the bus at same time.
   - When Master and Slave are both reading from the bus at same time.

   NOTE: To come out of bus lock-up situation, it is suggested to use callback during configuration. Application can decide time-out for callback wait and after timeout application can close the slave module instance and re-open for further operations.

2) Any of the supported IIC channel can be configured for either Master or Slave mode operation but not for both.
3) Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.24.4 Including the I2C RIIC Slave HAL Module in an Application

This section describes how to include the I2C RIIC Slave HAL Module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP based applications.

To add the I2C RIIC Slave HAL Module to an application, simply add it to a thread using the Stacks Selection Sequence given in the table below. (The default name for the I2C RIIC Slave HAL Module is g_i2c0 and this is shown in the below table. This name can be changed in the associated Properties window).

I2C RIIC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_i2c0 I2C Slave Driver on r_riic_slave	Threads	New Stack> Driver> Communications> I2C Slave Driver on r_riic_slave

When the I2C Slave HAL Module on r_riic_slave is added to the Thread Stack, as shown in the figure below, the configurator automatically adds any needed lower level drivers.

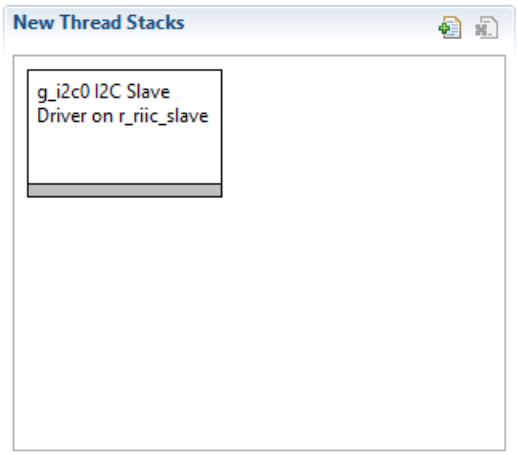


Figure 347:I2C RIIC Slave HAL Module Stack

4.2.24.5 Configuring the I2C RIIC Slave HAL Module

The I2C RIIC Slave HAL Module must be configured by you for the desired operation. The SSP configuration window will automatically identify, by highlighting the block in red, any required configuration selections, such as Interrupts or operating modes, which must be configured for lower level modules, for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP Configurator and are shown in the table later in this document for easy reference.

One of the properties most often identified as requiring a change is the Interrupt Priority. This configuration setting is available with the Properties window of the associated module. Simply select the indicated module and then view the properties window. The Interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the Interrupt Priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the below configuration properties tables, but is easily visible with the ISDE when configuring Interrupt Priority levels.

NOTE: You may want to open your ISDE and create the module and explore the property settings in parallel with looking over the Configuration Table Settings given below. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for I2C Slave HAL Module on r_i2c

ISDE Property	Value	Description
Parameter Checking	Enabled, Disabled, BSP (Default: BSP)	Selects if code for parameter checking is to be included in the build

ISDE Property	Value	Description
Name	Default: g_i2c0	Module name
Channel	0	Channel number
Rate	Standard, Fast mode, Fast mode plus (Default: Standard)	Rate selection
Slave Address	0x00	Slave address
Address Mode	7-bit, 10-bit (Default: 7-bit)	Address mode
Callback	NULL	Callback function
Receive Interrupt Priority	Priority 0(highest)-15(lowest), Disabled (Default: 2)	Receive Interrupt priority
Transmit Interrupt Priority	Priority 0(highest)-15(lowest), Disabled (Default: 2)	Transmit Interrupt priority
Error Interrupt Priority	Priority 0(highest)-15(lowest), Disabled (Default: 2)	Error Interrupt priority

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for stack modules can be desirable. For example, it might be useful to select different slave addresses or address mode.

I2C RIIC Slave HAL Module Clock Configuration

The RIIC peripheral module uses PCLKB as its clock source.

I2C RIIC Slave HAL Module Pin Configuration

The RIIC peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The first table below illustrates the method for selecting the pins within the SSP configuration window and the following table illustrates an example selection for the pins.

NOTE: For some peripherals, the Operation Mode selection mode determines what peripheral signals are available and thus what MCU pins are required.

Pin Selection Sequence for I2C RIIC Slave HAL Module

Resource	ISDE Tab	Pin selection Sequence
I2C	Pins	Select Peripherals > Connectivity: IIC > IIC0

NOTE: The above selection sequence assumes IIC0 is the desired hardware target for the driver.

Pin Configuration Settings for I2C RIIC Slave HAL Module on r_riic_slave

Pin Configuration Property	Value	Description
Pin Group Selection	_A only, _B only, Mixed (Default: _A only)	Select Simple I2C as the Operation Mode for I2C on SCI
Operation Mode	Enabled, Disabled (Default: Disabled)	Enable or disable peripheral module
SDA	None, P401, P407 (Default: None)	SDA Pin
SCL	None, P400, P204 (Default: None)	SCL Pin

NOTE: The above example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.24.6 Using the I2C RIIC Slave HAL Module in an Application

The typical steps in using the I2C RIIC Slave HAL Module in an application are:

1) Initialize and open the I2C Slave HAL Module using the open API
2) Transfer data to the master using the write API
3) Receive data from the master using the read API
4) Close the channel using the close API

The above common steps are illustrated in a typical operational flow diagram in the figure below.

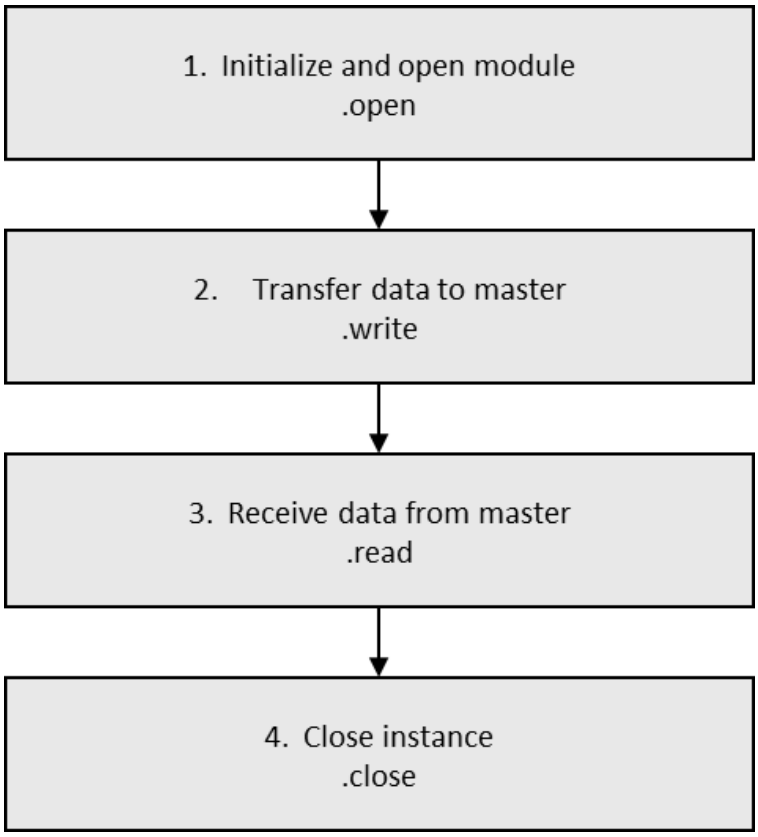


Figure 348: Flow Diagram of a Typical I2C RIIC Slave HAL Module Application

4.2.25 I2S Driver

The I2S HAL Driver module provides high-level APIs for the generic I2S audio serial communication protocol; it is used to send or receive uncompressed audio data in master mode.

4.2.25.1 I2S HAL Module Features

The I2S Driver used with the SSI peripheral in I2S master mode supports the following features in addition to the standard I2S protocol:

- Full-duplex I2S communication (SSI channel 0 only)
- Interrupt driven data transmission and reception
- Integration with the DTC transfer module.
- A user-defined callback can be created to respond to the need for additional data.

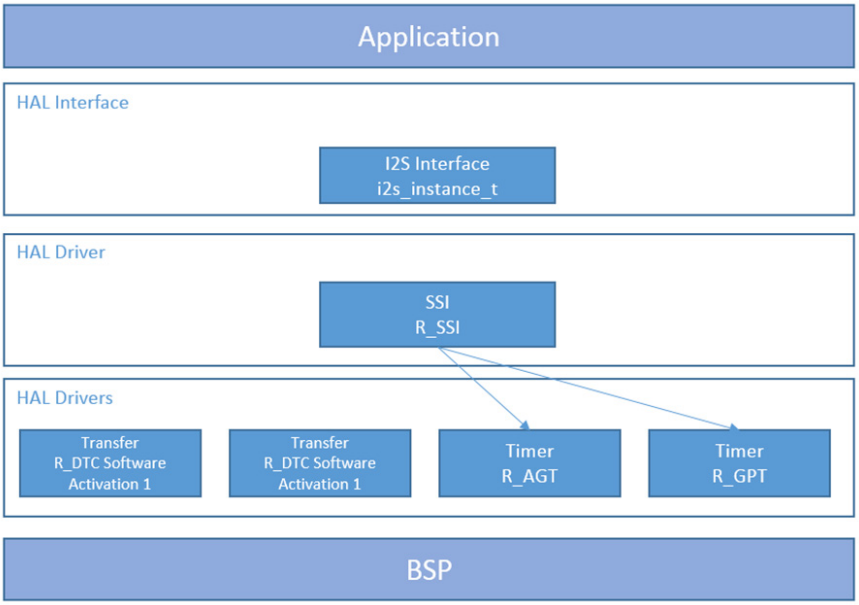


Figure 349:I2S HAL Module Block Diagram

4.2.25.2 I2S HAL Module APIs Overview

The I2S HAL module defines APIs for operations such as opening, muting, writing, and reading. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows.

I2S HAL Module API Summary

Function Name	Example API Call and Description
open	g_i2s0.p_api->open(g_i2s0.p_ctrl, g_i2s0.p_cfg); Initial configuration.

Function Name	Example API Call and Description
stop	g_i2s0.p_api->stop(g_i2s0.p_ctrl, direction_to_stop); Stop communication. Transmission is stopped when callback is called with I2S_EVENT_IDLE. Reception is stopped when callback is called with I2S_EVENT_RX_EMPTY.
mute	g_i2s0.p_api->mute(g_i2s0.p_ctrl, mute_enable); Enable or disable mute.
write	g_i2s0.p_api->write(g_i2s0.p_ctrl, &data, bytes); Write I2S data. All transmit data is queued when callback is called with I2S_EVENT_TX_EMPTY. Transmission is complete when callback is called with I2S_EVENT_IDLE.
read	g_i2s0.p_api->read(g_i2s0.p_ctrl, &data, bytes); Read I2S data. Reception is complete when callback is called with I2S_EVENT_RX_EMPTY.
writeRead	g_i2s0.p_api->writeRead(g_i2s0.p_ctrl, &source, &destination, bytes); Simultaneously write and read I2S data. Transmission and reception are complete when callback is called with I2S_EVENT_IDLE.
infoGet	g_i2s0.p_api->infoGet(g_i2s0.p_ctrl, &info); Get instance specific information and store it in provided pointer info.
close	g_i2s0.p_api->close(g_i2s0.p_ctrl); Allows driver to be reconfigured and may reduce power consumption.

Function Name	Example API Call and Description
versionGet	g_i2s0.p_api->versionGet(&version); Get version and store it in provided pointer version.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful.
SSP_ERR_OUT_OF_MEMORY	The number of streams open at once is limited to SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS. If this number is exceeded, an out of memory error occurs.
SSP_ERR_TIMEOUT	Timeout occurred before playback finished.
SSP_ERR_ASSERTION	A critical assertion has failed
SSP_ERR_IN_USE	Channel is running/busy
SSP_NOT_OPEN	Requested channel is not configured or API not open

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.25.3 I2S HAL Module Operational Overview

The I2S HAL module supports audio communications using the I2S protocol. The driver supports the SSI Peripheral on a Synergy MCU in I2S master mode. It can send and receive uncompressed audio data. It provides full-duplex I2S communication (SSI channel 0 only), interrupt driven data transmission and reception and integration with the DTC transfer module.

I2S HAL Module Important Operational Notes and Limitations

To enable audio data reception on channel 0, enable the SSI0 RXI interrupt. To enable audio data transmission on channel 0, enable the SSI0 TXI interrupt. To enable both transmission and reception on channel 0, enable both the SSI0 TXI and SSI0 RXI interrupts. To enable transmission or reception on channel 1, enable the SSIn TXI RXI interrupt. In all cases, enable the SSIn INT interrupt.

When the interrupts are enabled in the BSP, the corresponding ISRs will be defined in the I2S driver. The ISR will call the user callback function registered in open.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.25.4 Including the I2S HAL Module in an Application

This section describes how to include the I2S HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack.

To add the I2S HAL driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the I2S HAL Driver is g_i2s0. This name can be changed in the associated Properties window.)

I2S HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_i2s0 I2S Driver on r_ssi	Threads-> HAL/Common Stacks	New Stack> Driver> Connectivity> I2S Driver on r_ssi

When the I2S HAL module on r_ssi is added to the HAL/Common Stack or a thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information will be box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level drivers; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level drivers is required, the module description will include "Add" in the text. Clicking on any Pink banded modules will bring up the "New" icon and then display the possible choices.

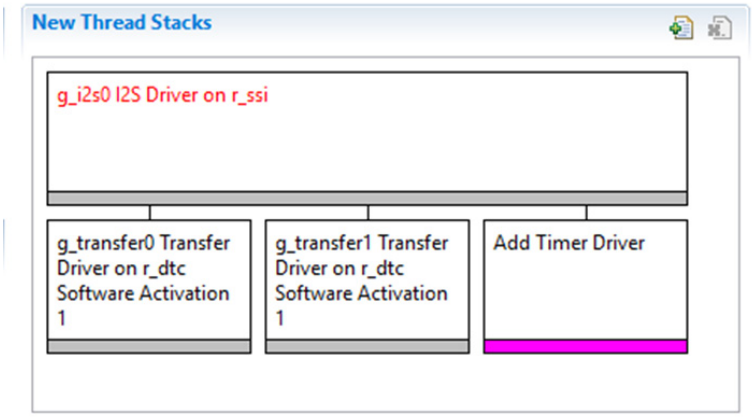


Figure 350:I2S HAL Module Stack

4.2.25.5 Configuring the I2S HAL Module

The I2S HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or

operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the properties window of the associated module. Simply select the indicated module and then view the properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the configuration properties tables below, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the I2S HAL Module on r_ssi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enables or disables the parameter checking.
Name	g_i2s0	Module name.
Channel	0	Physical hardware channel.
Audio Clock Frequency (Hertz)	2822400	Input audio clock frequency, used to generate the I2S clock. Must be a multiple between 1 and 128 of: (sampling_freq_hz * word_length_in_bits)
Sampling Frequency (Hertz)	44100	Sampling frequency of audio data.
Data Bits	8 bits, 16, 18, 20, 22, 24 Default: 16 bits	Bit depth of audio data, which is the size in bits of one sample of audio data.
Word Length	8 bits, 16, 24, 32 Default: 16 bits	Word length of audio data, must be at least the same size as the bit depth (Data Bits field).

ISDE Property	Value	Description
WS Continue Mode	Enabled, Disabled Default: Disabled	Enable WS continue mode to continue to output the word select line when the peripheral is idle. Disable to stop outputting the word select line when the peripheral is idle.
Name of I2S callback function to be defined by user	NULL	A user callback function must be registered in open. The callback will be called from the interrupt service routine (ISR) when the transmission FIFO reaches the high watermark point after all data for transmission is transmitted or when reception is complete (the requested number of bytes have been received). Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Transmit interrupt priority selection
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Receive interrupt priority selection

ISDE Property	Value	Description
Idle/Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Idle/error interrupt priority selection

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower level modules can be desirable. For example, it might be useful to select the DAC or I2S Channel based on the target hardware implementation. The configurable properties for the lower level stack modules are given in the below sections for completeness and as a reference.

NOTE: Most of the property settings for modules are fairly intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the I2S HAL Module Low-Level Modules

There are two implementation choices for the Timer Driver used with the I2S HAL module and each of these options requires different configuration settings for the associated low-level drivers. The configuration options for these two options are provided in the following tables.

Typically, only a small number of settings must be modified from the default for lower-level drivers and these are indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

Table 5 Configuration Settings for the DTC HAL Module on r_dtc Software Activation

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Name	g_transfer0	Driver name.
Mode	Normal	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Fixed	Destination address mode selection.

ISDE Property	Value	Description
Source Address Mode	Incremented	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	FALSE	Auto enable selection.
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection
Callback (Only valid with Software start)	NULL	Callback selection

NOTE: The example settings and defaults are for a project using the Synergy S7 MCU Family. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the DTC HAL Module on r_dtc Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Name	g_transfer1	Driver name.
Mode	Normal	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection.
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	0	Number of transfers selection.
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection.
Activation Source (Must enable IRQ)	Software Activation 1, Software Activation 2, Peripheral Events Default: Software Activation 1	Activation source selection.
Auto Enable	FALSE	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.

ISDE Property	Value	Description
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC software event interrupt priority selection

NOTE: The example settings and defaults are for a project using the Synergy S7 MCU Family. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the AGT HAL Module on r_agt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection
Name	g_timer0	Module name
Channel	0	Channel selection
Mode	Periodic	Mode selection
Period Value	2822400 *2	Period value selection
Period Unit	Hertz	Period unit selection
Auto Start	False	Auto start selection
Count Source	PCLKB, PCLKB/8, PCLKB/2, LOCO, AGT0 Underflow, AGT0 fSUB Default: PCLKB	Count source selection
AGTO Output Enabled	True, False Default: False	AGTO output selection

ISDE Property	Value	Description
AGTIO Output Enabled	True, False Default: False	AGTIO output selection
Output Inverted	True, False Default: False	Output inverted selection
Callback	NULL	Callback selection
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection

NOTE: The example settings and defaults are for a project using the Synergy S7 MCU Family. Other MCUs may have different default values and available configuration settings.

Configuration Settings for Timer Driver on r_gpt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection
Name	g_timer0	Module name
Channel	0	Channel selection
Mode	Periodic	Mode selection
Period Value	2822400 *2	Period value selection
Period Unit	Hertz	Period unit selection
Duty Cycle Value	50	Duty cycle value selection

ISDE Property	Value	Description
Duty Cycle Unit	Unit Raw Counts, Unit Percent, Unit Percent x 1000 Default: Unit Raw Counts	Duty cycle unit selection
Auto Start	False	Auto start selection
GTIOCA Output Enabled	True, False Default: False	GTIOCA output enabled selection
GTIOCA Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	GTIOCA stop level selection
GTIOCB Output Enabled	True, False Default: False	GTIOCB output enabled selection
GTIOCB Stop Level	Pin Level Low, Pin Level High, Pin Level Retained Default: Pin Level Low	GTIOCB stop level selection
Callback	NULL	Callback selection
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection

NOTE: The example settings and defaults are for a project using the Synergy S7 MCU Family. Other MCUs may have different default values and available configuration settings.

I2S HAL Module Clock Configuration

The SSI module uses the peripheral clock (PCLKB) available from the Clock configuration window. It also uses an external clock input to the AUDIO_CLK pin.

I2S HAL Module Pin Configuration

The SSI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device.

Configure the SSI RX and/or TX pins for the selected SSI channel (Pins Tab > Peripherals > SSI > SSIn > SSITXD0/SSIRXD0). For channel 0, enable one or both of these pins. For channel 1, enable the SSIDATA1 pin.

Configure the word select and clock pins (Pins Tab > Peripherals > SSI > SSIn > SSITWSn and SSISCKn). Both of these pins are required in most cases. Consult the datasheet of the I2S device used for the required pins.

Configure the audio clock pin (Pins Tab > Peripherals > SSI > SSI0_SSI1_AUDIO_CLK) for SSI. Connect an external audio clock to this clock input pin. If a GPT timer is used to generate the audio clock, configure the GPT timer output pin (Pins tab > Peripherals > GPT1 > GPTn > GTIOCx) and connect the GPT output pin used to the audio clock input pin.

The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the associated pins.

Pin Selection Sequence for the I2S HAL Module

Resource	ISDE Tab	Pin selection Sequence
I2S	Pins	Select Peripherals > Connectivity:SSI > SSI0/SSI1
I2S	Pins	Select Peripherals > Connectivity:SSI > SSI0_SSI1_AUDIO_CLK

NOTE: The selection sequence assumes SCI0 is the desired hardware target for the driver.

Pin Configuration Settings for I2S Driver on SSI

Pin Configuration Property	Settings	Description
Pin Group Selection	_A only, Mixed	Pin group for I2S port.
Operation Mode	Custom, Disabled	Operation selection.
SSISCK	None, P400 (Default: None)	AUDIO_CLK pin(P400), used in this application

NOTE: The above example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

Pin Configuration Settings for I2S Driver on SSI0

Pin Configuration Property	Settings	Description
Pin Group Selection	_A only, _B only, Mixed	Pin group for I2S port.
Operation Mode	Enabled, Custom, Disabled	Operation selection.
SSISCK	None, P112 (Default: None)	SSISCK pin(P112), used in this application
SSIWS	None, P113 (Default: None)	SSIWS pin(P113), used in this application
SSITXD	None, P115 (Default: None)	SSITXD pin(P115), used in this application
SSIRXD	None,P114 (Default:None)	SSIRXD pin(P114), used in this application

NOTE: The above example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

Pin Configuration Settings for I2S Driver on SSI1

Pin Configuration Property	Settings	Description
Pin Group Selection	_A only, _B only, Mixed	Pin group for I2S port.
Operation Mode	Enabled, Custom, Disabled	Operation selection.
SSISCK	None, P204 (Default: None)	SSI Serial Clock, not used in this application.

Pin Configuration Property	Settings	Description
SSIWS	None, P205 (Default: None)	SSI Stereo pin selection, not used in this application.
SSIDATA	None, P206 (Default: None)	SSI Data pin selection, not used in this application.

NOTE: The above example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.25.6 Using the I2S HAL Module in an Application

The typical steps in using the I2S HAL module in an application are:

1) Open the I2S HAL module using the open API.
2) Use the write API to write audio data to the I2S bus.
3) Wait for callback with I2S_EVENT_TX_EMPTY and free the source buffer.
4) Use the read API to read data from the I2S bus.
5) Wait for callback with I2S_EVENT_RX_FULL before accessing the destination buffer or reading the next buffer.
6) Use other APIs as needed by the application.
7) Close the module with the close API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

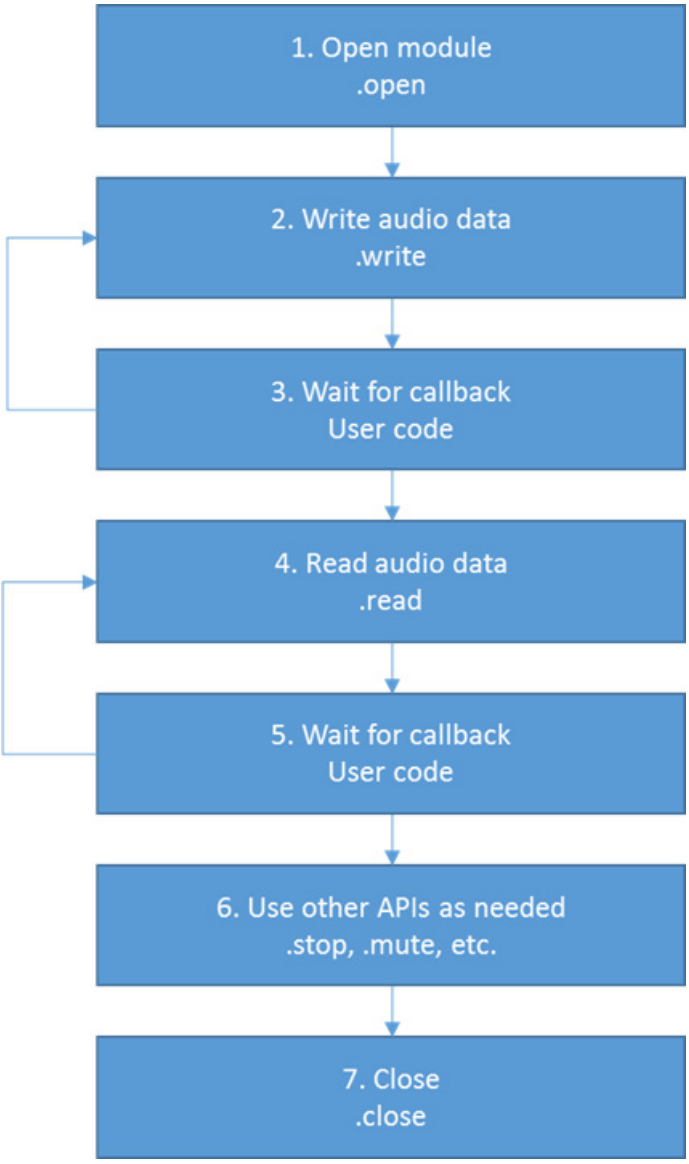


Figure 351: Flow Diagram of a Typical I2S HAL Module Application

4.2.26 Input Capture Driver

The Input Capture HAL module provides high-level APIs used for measuring input pulse-width and pulse-period measurement. The Input Capture HAL module configures the input capture parameters to use with the GPT peripheral on the Synergy MCU. A user-defined callback can be created to acquire the value each time a new measurement is complete.

4.2.26.1 Input Capture HAL Module Features

The Input Capture HAL module configures the GPT for an input capture function.

- The Input Capture HAL allows the user to perform the following tasks:
  - Initialize the module
  - Enable input capture measurement
  - Disable input capture measurement
  - Get the status (running or not) of the measurement counter
  - Get the last captured timer/overflows counter value
  - Close the input capture operation
- The Input Capture HAL module supports:
  - Pulse-width measurement and pulse-period measurement
  - Rising-edge or falling-edge measurement start
  - One-shot or periodic mode
  - Hardware-enable signals to enable captures (low enable/high enable)
  - Callback function with the following events:
  - **Counter overflow**
  - **Input capture occur**
  - Callback structure (input_capture_callback_args_t) that provides data on the interrupting event, including which interrupt occurs and the associated counter values.

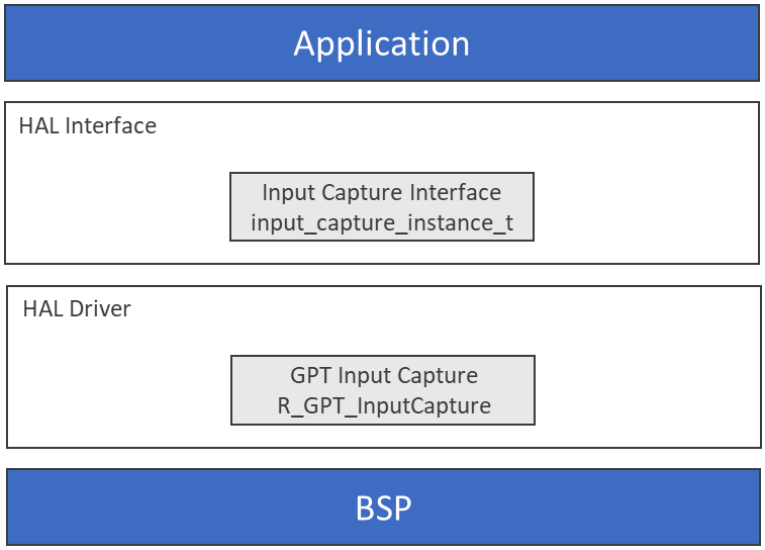


Figure 352: Input Capture HAL Module Block Diagram

4.2.26.2 Input Capture HAL Module APIs Overview

The Input Capture HAL module interface defines APIs for opening, closing, enabling, disabling, accessing status information and last-capture value accessing using the General PWM Timer (GPT) with Input Capture. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the HAL Module API Summary.

Input Capture HAL Module API Summary

Function Name	Example API Call and Description
.open	g_input_capture.p_api->open(g_input_capture.p_ctrl, g_input_capture.p_cfg); Opens the Input Capture HAL and initializes configuration.
.close	g_input_capture.p_api->close(g_input_capture.p_ctrl); Closes the input capture operation. Allow drive to be reconfigured, and may reduce power consumption.
.enable	g_input_capture.p_api->enable(g_input_capture.p_ctrl); Enables input capture measurement.
.disable	g_input_capture.p_api-> disable(g_input_capture.p_ctrl); Disables input capture measurement.
.infoGet	g_input_capture.p_api->infoGet(g_input_capture.p_ctrl, &input_capture_info); Gets the status (running or not) of the measurement counter.

Function Name	Example API Call and Description
.lastCaptureGet	g_input_capture.p_api->lastCaptureGet(g_input_capture.p_ctrl, &input_capture_counter); Gets the last captured timer/overflows counter value.
.versionGet	g_input_capture.p_api->versionGet(&input_capture_version); Retrieve the API version with the input_capture_version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the Synergy Software Platform (SSP) User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_ASSERTION	One of the parameters is NULL. Or the channel requested in the p_cfg parameter may not be available on the device selected in r_bsp_cfg.h. Or, p_cfg->mode is invalid.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value. Or ISR is not enabled.
SSP_ERR_IN_USE	Attempted to open an already open device instance.
SSP_ERR_NOT_OPEN	The channel is not opened.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.26.3 Input Capture HAL Module Operational Overview

The Input Capture HAL module controls the GPT HAL module units on a Synergy microcontroller (as configured by the user). It directly accesses the GPT hardware without using any RTOS elements and provides convenient APIs to simplify development.

When a normal measurement is complete and a callback is available (with interrupts enabled,) the Input Capture HAL module invokes the callback with the argument input_capture_callback_args_t.

The argument (input_capture_callback_args_t) indicates the channel, the event (input_capture_event_t) , the value of the timer captured when the interrupt occurred, and the number of counter overflows that occurred during this measurement.

If the interrupts are not enabled, the values read by the APIs would be the last captured timer/overflows counter value.

Input Capture HAL Module Important Operational Notes and Limitations

GPT Input Capture Measurement Mode

The input capture interface provides support for one-shot measurement and periodic measurement. The GPT hardware does not natively support one-shot functionality. Code is automatically included in the interrupt service routine (ISR) to stop and clear the timer. For this reason, ISRs must be enabled for one-shot mode, even when the callback is unused.

GPT Input Capture Signal

The input capture measurement starts when the input capture signal edge (rising or falling) is detected on the input capture signal pin (GTIOCA/GTIOCB) and the enable condition is met. The enable condition is defined by the enable level and can be disabled (none), or a specified low or high level on the input capture enable pin (GTIOCA/GTIOCB). The input capture enable pin is the pin not used as the input capture signal pin.

Converting Measurement Counts to Time

When a measurement completes, the raw-count data and the number of overflows is returned to the user in the callback function.

If desired, the raw measurement data can be converted to logical time units in the callback or user application. To convert the raw data, the current PCLKD clock frequency and its pre-scaler value, number of overflows, maximum counter value, and measurement counts should be considered. The measurement counts and the number of overflows are provided in the callback arguments input_capture_callback_args_t.

The recommended method to obtain the current PCLKD frequency is to use the systemClockFreqGet API. The input clock frequency is the PCLKD frequency divided by the pre-scalar value and is represented as clk_freq_hz in the following Input Capture Time Calculation table.

The maximum counter value on the S7G2 (all channels), S3A7 (all channels), and S124 (channel 0) is 0xFFFFFFFF. The maximum counter value for S124 (channels 1 - 6) is 0xFFFF. This maximum counter value plus one (since counter starts from zero) is represented as max_counts in the following table:

Input Capture Time Calculation

Desired Time Units	Formula
Nanoseconds (ns)	time_ns = ((overflows * max_counts) + counter) * 1000000000/clk_freq_hz
Microseconds (us)	time_ns = ((overflows * max_counts) + counter) * 1000000/clk_freq_hz
Milliseconds (ms)	time_ns = ((overflows * max_counts) + counter) * 1000/clk_freq_hz

Desired Time Units	Formula
Seconds (s)	time_ns = ((overflows * max_counts) + counter)/clk_freq_hz

- Currently, the Input Capture HAL module supports only pulse-width measurement and pulse-period measurement.
- Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.26.4 Including the Input Capture HAL Module in an Application

This section describes how to include the Input Capture HAL module in an application using the SSP configurator.

NOTE: It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the Input Capture Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Input Capture Driver is g_input_capture. This name can be changed in the associated Properties window.)

Input Capture HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_input_capture Input Capture Driver on r_gpt_input_capture	Threads -> HAL/Common Stacks	Highlight Threads > HAL/Common Stacks and select New Stack > Driver > Timers > Input Capture Driver on r_gpt_input_capture

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_input_capture	Module name.
Channel	0	Physical hardware channel.
Mode	Pulse Width	Measures inputs from the Signal edge until the opposite edge.

ISDE Property	Value	Description
Signal Edge	Rising, Falling Default: Rising	Start measurement on rising or falling edge. Measurement stops on the opposite edge.
Repetition	Periodic, One Shot Default: Periodic	Capture a single measurement, then disable captures (one shot) until enable is called, or capture measurements continuously (periodic).
Auto Start	True, False Default: True	Set to true to enable measurements after configuring or false to leave the measurements disabled until enable is called.
Callback	NULL	A user callback function must be registered in open. The callback will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Input Capture Signal Pin	GTIOCA, GTIOCB Default: GTIOCA	Select the input pin used to trigger the start of a measurement.
GTIOCx Signal Filter	None, PCLK/1, PCLK/4, PCLK/16, PCLK/64 Default: None	The noise filter samples the external signal at intervals of the PCLK divided by one of the values. When 3 consecutive samples are at the same level (high or low), that level is passed on as the observed state of the signal.
Clock Divider	PCLK/1, PCLK/4, PCLK/16, PCLK/64, PCKL/256, PCLK/1024 Default: PCLK/1	Clock divider used to scale the measurement counter.

ISDE Property	Value	Description
Input Capture Enable Level	None, Low, High Default: None	Each GPT channel has 2 I/O pins (GTIOCA and GTIOCB). One must be selected as the Input Capture Signal Pin. The other GPT I/O pin can be used as a hardware enable signal to enable captures. Select None and captures are always enabled, select low and captures are enabled only while the enable input pin is low, select high and captures are enabled only while the enable input pin is high.
Input Capture Enable Filter	None (No filtering), PCLK/1 (Fast sampling), PCLK/4, PCLK/16, PCLK/64 (Slow sampling) Default: None (No filtering)	The enable filter samples the enable signal at intervals of the PCLK divided by one of the values. When 3 consecutive samples are at the same level (high or low), that level is passed on as the observed state of the signal.
Capture Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Default: Priority 12	Capture interrupt priority selection.
Overflow Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Default: Priority 12	Overflow interrupt priority selection.

The following figure shows when the Input Capture HAL module on r_gpt_input_capture is added to the thread stack, the configurator automatically adds any lower-level drivers needed. Any drivers requiring configuration information are box text highlighted in Red. Modules with a Gray band are standalone modules. Items with a Blue band are shared or common modules; they only need to be added once and can be used by multiple stacks.

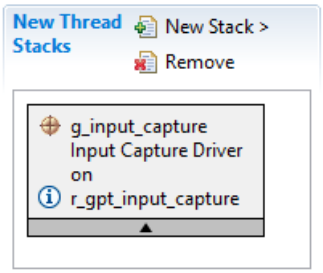


Figure 353: Input Capture HAL Module Stack

4.2.26.5 Configuring the Input Capture HAL Module

The user configures the Input Capture HAL module for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in Red) any configuration selections required, such as interrupts or operating modes, for lower-level modules to achieve successful operation. Only properties that can be changed without causing a conflict are available for modification. Properties that are 'locked' are identified with a lock icon in the **Properties** window and cannot be changed in the ISDE. This approach simplifies the configuration process, making it much less error prone than previous 'manual' approaches to configuration. The **Properties** tab within the SSP Configurator show all available user-accessible properties. The configuration settings and defaults for these properties are given in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority. This configuration setting is available in the **Properties** window of the associated module. Simply select the indicated module, then view the Properties window. Note that the **Properties** window the interrupt priorities also indicate the validity of the setting based on the MCU targeted (CM4 or CM0+).

NOTE: You may want to open your ISDE. and create the module and explore the property settings in parallel with looking over the following configuration table settings. This can help orient you and can also be a useful 'hands-on' approach as you learn the ins and outs of developing with SSP.

Configuration Settings for the Input Capture HAL Module on r_gpt_input_capture

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_input_capture	Module name.
Channel	0	Physical hardware channel.
Mode	Pulse Width	Measures inputs from the Signal edge until the opposite edge.

ISDE Property	Value	Description
Signal Edge	Rising, Falling Default: Rising	Start measurement on rising or falling edge. Measurement stops on the opposite edge.
Repetition	Periodic, One Shot Default: Periodic	Capture a single measurement, then disable captures (one shot) until enable is called, or capture measurements continuously (periodic).
Auto Start	True, False Default: True	Set to true to enable measurements after configuring or false to leave the measurements disabled until enable is called.
Callback	NULL	A user callback function must be registered in open. The callback will be called from the interrupt service routine (ISR) each time the timer period elapses. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Input Capture Signal Pin	GTIOCA, GTIOCB Default: GTIOCA	Select the input pin used to trigger the start of a measurement.
GTIOCx Signal Filter	None, PCLK/1, PCLK/4, PCLK/16, PCLK/64 Default: None	The noise filter samples the external signal at intervals of the PCLK divided by one of the values. When 3 consecutive samples are at the same level (high or low), that level is passed on as the observed state of the signal.
Clock Divider	PCLK/1, PCLK/4, PCLK/16, PCLK/64, PCKL/256, PCLK/1024 Default: PCLK/1	Clock divider used to scale the measurement counter.

ISDE Property	Value	Description
Input Capture Enable Level	None, Low, High Default: None	Each GPT channel has 2 I/O pins (GTIOCA and GTIOCB). One must be selected as the Input Capture Signal Pin. The other GPT I/O pin can be used as a hardware enable signal to enable captures. Select None and captures are always enabled, select low and captures are enabled only while the enable input pin is low, select high and captures are enabled only while the enable input pin is high.
Input Capture Enable Filter	None (No filtering), PCLK/1 (Fast sampling), PCLK/4, PCLK/16, PCLK/64 (Slow sampling) Default: None (No filtering)	The enable filter samples the enable signal at intervals of the PCLK divided by one of the values. When 3 consecutive samples are at the same level (high or low), that level is passed on as the observed state of the signal.
Capture Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Default: Priority 12	Capture interrupt priority selection.
Overflow Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Default: Priority 12	Overflow interrupt priority selection.

NOTE: The example values and defaults listed in the table are for a project using the Synergy S7G2 MCU family. Other MCUs may have different default values and available configuration settings.

Input Capture HAL Module Clock Configuration

The GPT HAL module uses the PCLKD as its clock source. The PCLKD frequency is set using the SSP configurator clocks tab prior to a build, or using the CGC Interface at run-time.

Input Capture HAL Module Pin Configuration

To access a particular channel and pin, the GTIOCx pins must be set in the Pins tab of the ISDE. The following table has the method for selecting the pins within the SSP configuration window, with the subsequent table listing an example selection for GTIOCx pins.

Pin Selection Sequence for Input Capture HAL Module

Resource	ISDE Tab	Pin selection Sequence
GPT Input Capture	Pins	Select **Peripherals > Timer: GPT > GPT0**

NOTE: The selection sequence assumes GPT0 is the desired hardware target for the driver.

Pin Configuration Settings for Input Capture HAL Module

Property	Value	Description
Pin Group Selection	Mixed, _A Only, _B Only (Default: Mixed)	Pin grouping selection
Operation Mode	Disabled, GTIOCA or GTIOCB, GTIOCA and GTIOCB (Default: Disable)	Select GTIOCA or GTIOCB as the Operation Mode for Input Capture on GPT
GTIOCA	None, P300, P512 (Default: None)	GTIOCA Pin
GTIOCB	None, P108, P511 (Default: None)	GTIOCB Pin

NOTE: The example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.26.6 Using the Input Capture HAL Module in an Application

Once the module has been configured and the files generated, the Input Capture HAL module is ready to be used in an application. The typical steps to using the Input Capture HAL module in an application are:

1) Initialize the module using the open API
2) The desired value can be found either in the main loop routine using the lastCaptureGet API or in the callback function

3) The capture and overflow interrupt can be disabled and the timer stopped using the disable API.
4) The capture and overflow interrupt can be enabled and the timer started using the enable API.
5) The status of the captured counter (running or stopped) can be queried using infoGet API
6) The module can be closed using the close API once done.

The following figure illustrates these common steps in a typical operational flow:

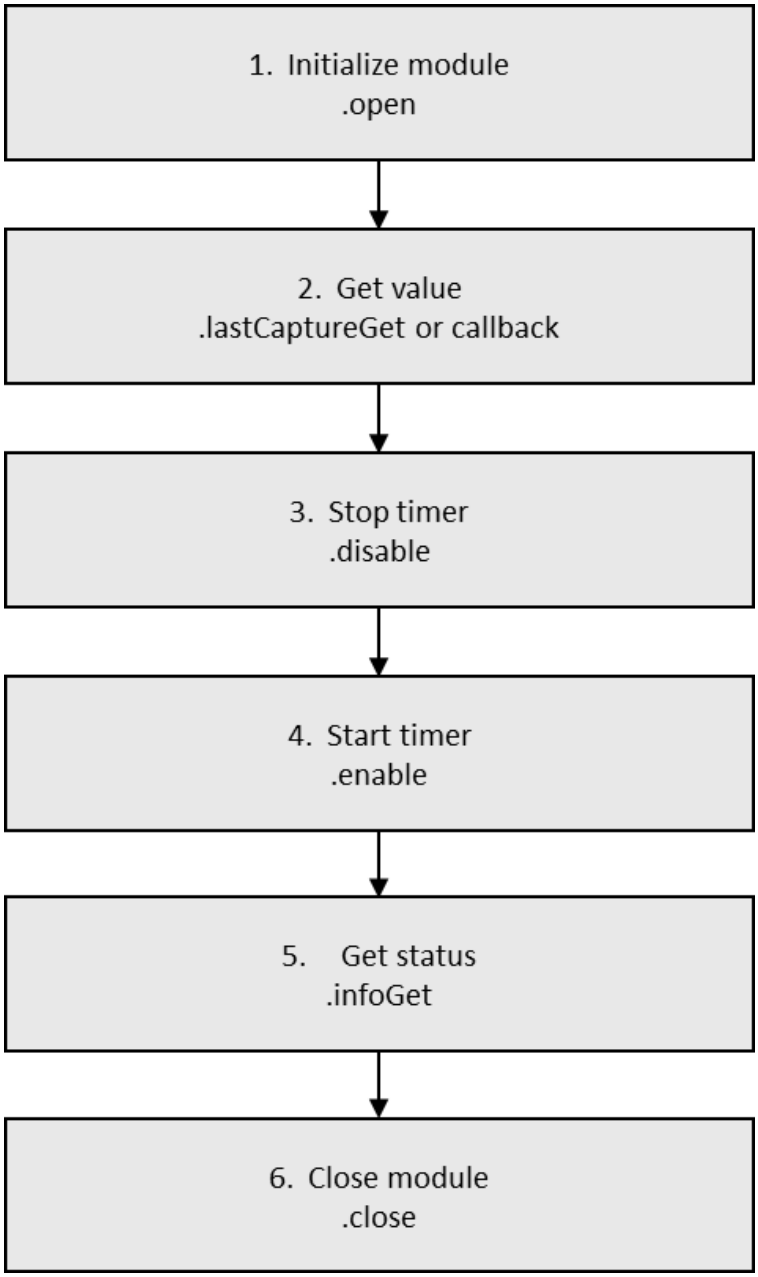


Figure 354: Flow Diagram of a Typical Input Capture HAL Module Application

4.2.27 I/O Port Driver

The I/O Port HAL module provides high-level APIs for controlling I/O pins and is implemented on r_ioport. The I/O Port HAL module configures the board's pins and provides functions for manipulating them. The operating state of the I/O

pins can be set via the Synergy configurator. When the Synergy project is built, a pin configuration file is created, and when the application runs, the BSP will configure the IO port accordingly, using the same APIs detailed in this document.

4.2.27.1 I/O Port HAL Module Features

This module configures one or more I/O pins. The direction of the pin or pins can be configured along with a number of other options provided as follows:

- Pull-up
- NMOS/PMOS
- Drive strength
- Event edge trigger (falling, rising or both)
- Whether the pin is to be used as an IRQ pin
- Whether the pin is to be used as an analog pin
- Whether the pin is to be used as a peripheral pin and which peripheral

The module also provides the following functionality:

- Changes the direction of one or more pins on a port
- Writes to one or more pins on a port
- Reads from one or more pins on a port
- Sets event output data
- Reads event input data

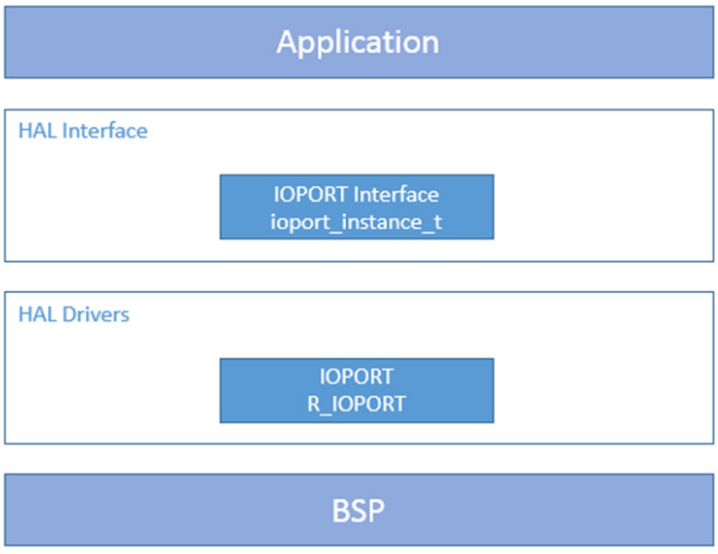


Figure 355:I/O Port HAL Module Block Diagram

4.2.27.2 I/O Port HAL Module APIs Overview

The I/O Port HAL module defines APIs for reading and writing from particular pins and ports. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of return status values follows the API summary table.

I/O Port HAL Module API Summary

Function Name	Example API Call and Description
init	g_ioport.p_api->init(g_ioport.p_cfg); Initialize configuration of multiple pins.
pinCfg	g_ioport.p_api->pinCfg(IOPORT_PORT_00_PIN_00, IOPORT_CFG_IRQ_ENABLE \ IOPORT_CFG_PORT_DIRECTION_INPUT); Configure settings for an individual pin.
pinsCfg	g_ioport.p_api->pinsCfg(&pin_config); Configure settings for a set of pins.
pinDirectionSet	g_ioport.p_api->pinDirectionSet(IOPORT_PORT_00_PIN_00, IOPORT_DIRECTION_INPUT); Set the pin direction of a pin.
pinEventInputRead	g_ioport.p_api->pinEventInputRead(IOPORT_PORT_00_PIN_00, &pin_level); Read the event (ELC) input data of the specified pin and return the level.
pinEventOutputWrite	g_ioport.p_api->pinEventOutputWrite(IOPORT_PORT_00_PIN_00, IOPORT_PIN_LEVEL_HIGH); Write pin event (ELC) data.
pinEthernetModeCfg	g_ioport.p_api->pinEthernetModeCfg(IOPORT_ETHERNET_CHANNEL_0, IOPORT_ETHERNET_MODE_MII); Configure the PHY mode of the Ethernet channels.

Function Name	Example API Call and Description
pinRead	g_ioport.p_api->pinRead(IOPORT_PORT_00_PIN_00, &pin_level); Read level of a pin.
pinWrite	g_ioport.p_api->pinWrite(IOPORT_PORT_00_PIN_00, IOPORT_PIN_LEVEL_HIGH); Write specified level to a pin.
portDirectionSet	g_ioport.p_api->portDirectionSet(IOPORT_PORT_00, direction_values, mask); Set the direction of one or more pins on a port.
portEventInputRead	g_ioport.p_api->portEventInputRead(IOPORT_PORT_00, &pin_levels); Read captured event (ELC) data for a port.
portEventOutputWrite	g_ioport.p_api->portEventOutputWrite(IOPORT_PORT_00, pin_levels, mask); Write event (ELC) output data for a port.
portRead	g_ioport.p_api->portRead(IOPORT_PORT_00, &pin_levels); Read states of pins on the specified port.
portWrite	g_ioport.p_api->portWrite(IOPORT_PORT_00, pin_levels, mask); Write to multiple pins on a port.
versionGet	g_ioport.p_api->versionGet(&version); Retrieve version information using the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines,

API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	The port/pin/mask/direction/level (etc.) not valid.
SSP_ERR_ASSERTION	Unexpected null pointer.
SSP_ERR_UNSUPPORTED	Feature not supported – for instance Ethernet configuration not supported on the device.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.27.3 I/O Port HAL Module Operational Overview

The I/O Port HAL module provides the ability to access the I/O ports of a device at both bit and port level; both port and pin direction can be changed. In addition, a number of configuration APIs are provided to change the functionality of individual pins.

The I/O Port HAL module provides the following operations for configuring pins:

- Initializes the driver – performed by calling init:
  - Performs parameter checking and processes error conditions.
  - Handles VBATT domain pin configuration.
  - Writes PFS registers for pins.
- Configures pin – performed by calling pinCfg API:
  - Performs parameter checking and processes error conditions (checks pin number pin, VBATT support).
  - Writes PFS register for the pin.
- Reads pin level – performed by calling pinRead API:
  - Performs parameter checking and processes error conditions (checks pin number pin).
  - Reads PFS register for the pin.
- Reads all pin levels on a port – performed by calling portRead API:
  - Performs parameter checking and processes error conditions (checks port number port).
  - Reads current value of PCNTR register value for the specified port.

- Writes pin level – performed by calling pinWrite API :
  - Performs parameter checking and processes error conditions (check pin number pin and written level level).
  - Write to PFS register for the pin.
- Write multiple pin levels on a port – performed by calling portWrite v:
  - Performs parameter checking and processes error conditions (checks port number port and pin mask mask).
  - Reads current configuration from the PCNTR register for the specified port.
  - Writes the pin levels to the PCNTR register for the specified port accordingly to the mask, preserving pin levels out of the scope.
- Sets the direction of multiple pins on a port – performed by calling portDirectionSet API :
  - Performs parameter checking and processes error conditions (checks port number port and pin mask mask).
  - Reads current configuration from the PCNTR register for the specified port.
  - Writes the pin levels to the PCNTR register for the specified port accordingly to the mask, preserving pin directions out of the scope.
- Writes pin direction – performed by calling pinDirectionSet API:
  - Performs parameter checking and processes error conditions (checks pin number pin and written direction direction).
  - Writes to the PFS register for the pin.
- Reads event (ELC) port input – performed by calling portEventInputRead API :
  - Performs parameter checking and processes error conditions (checks port number port).
  - Reads current value of the PCNTR register value for the specified port.
- Reads event (ELC) pin input – performed by calling pinEventInputRead API :
  - Performs parameter checking and processes error conditions (checks pin number pin).
  - Reads current value of the PCNTR register value for the specified pin's port.
  - Gets the pin level by applying a pin mask to the PCNTR register value.
- Writes event (ELC) port output – performed by calling portEventOutputWrite API :
  - Performs parameter checking and processes error conditions (checks port number port and pin mask mask_value).
  - Reads current configuration from the PCNTR register for the specified port.
  - Writes the pin levels to the PCNTR register for the specified port accordingly to the mask preserving pin levels out of the event scope.
- Writes event (ELC) pin output – performed by calling pinEventOutputWrite API:
  - Performs parameter checking and processes error conditions (checks pin number pin and written level pin_value).

  - Writes the pin level to the PCNTR register for the specified pin's port accordingly to the mask that is preserving pin levels out of the event scope.
- Configures Ethernet channel PHY mode – performed by calling ethernetModeCfg API:
  - Performs parameter checking and processes error conditions (checks Ethernet channel channel and mode mode).
  - Updates the Ethernet Control Register (PFENET).

I/O Port HAL Module Important Operational Notes and Limitations

- A bit mask of 16 bits needs to be applied in order to read and write to a specific pin on a port; ports are numbered from 0 (LSB) to 15 (MSB).
- Ethernet configuration may not be supported on some devices.

Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.27.4 Including the I/O Port HAL Module in an Application

The e^2 studio Integrated Solution Developer Environment (ISDE) automatically adds the necessary I/O Port stack to the HAL/Common thread in the project and the stack is ready by default to operate. If the I/O Port Driver was inadvertently removed, the following process is available to add the driver to the HAL/Common Thread.

NOTE: It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the *SSP User's Manual* to learn how to manage each of these important steps in creating SSP-based applications.

To add the I/O Port HAL Driver to your application, simply add it to a project Thread using the stacks selection sequence given in the following table. (The default name for the I/O Port is g_ioport.)

I/O Port HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_ioport I/O Port driver on r_ioport	Threads	Highlight HAL/Common and select New > Driver > System > I/O Port Driver on r_ioport

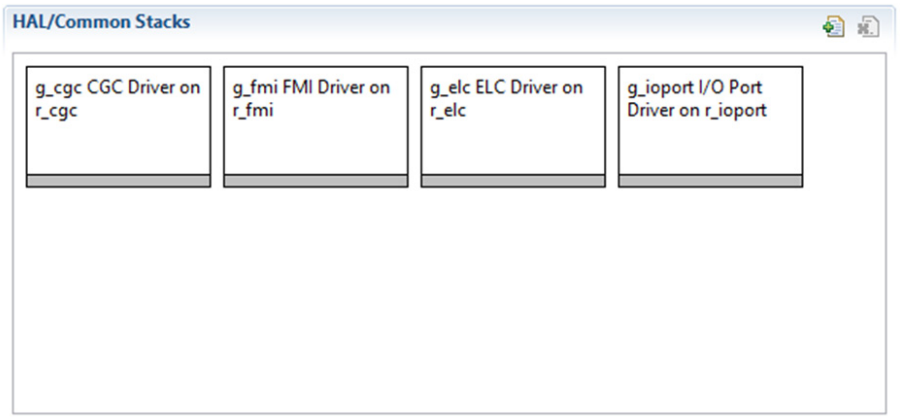


Figure 356: I/O Port HAL Module Stack

No additional lower-level modules need to be added or configured.

4.2.27.5 Configuring the I/O Port HAL Module

The module does not need any additional configuration. All the pin configuration is generated in the src/synergy_gen/pin_data.c file according to the BSP.

4.2.27.6 Using the I/O Port HAL Module in an Application

The typical steps in using the I/O Port HAL module in an application are:

1) Initialize the driver using the init API.
2) Configure the pins using the pinCfg API.
3) Read from specified pins and ports using the pinRead, or portRead API.
4) Write to specified pins and ports using the pinWrite, or portWrite API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

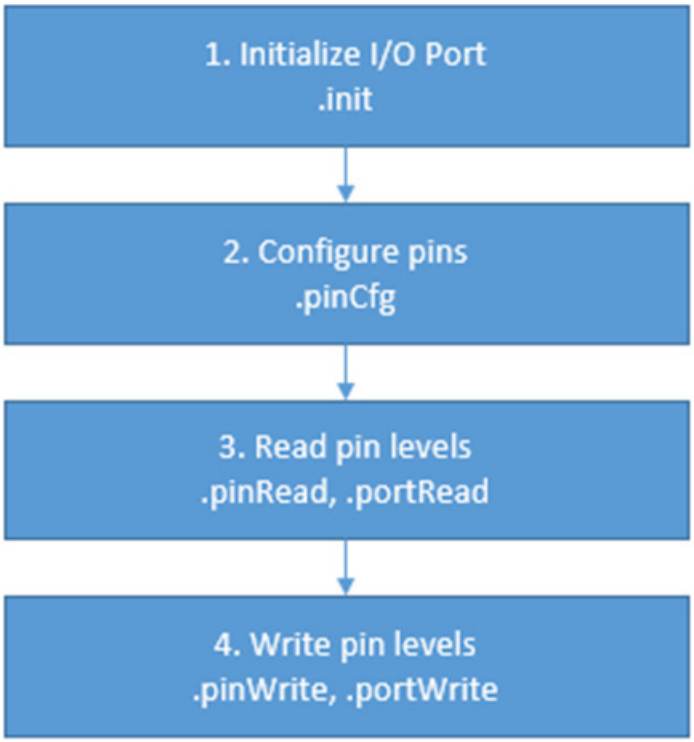


Figure 357: Flow Diagram of a Typical I/O Port HAL Module Application

4.2.28 Independent Watchdog Driver

4.2.28.1 Independent Watchdog Driver

The Independent Watchdog Timer (IWDT) HAL module provides high-level APIs for watchdog timer applications and is implemented on r_iwdt. The Watchdog Timer uses the IWDT peripheral on the Synergy MCU. A user-defined callback can be created to respond to event notifications.

4.2.28.2 IWDT HAL Module Features

The IWDT HAL module has the following key features:

- When the WDT underflows or is refreshed outside of the permitted refresh window, one of the following events can occur:
  - Device reset
  - NMI generation
- Supports the internal Watchdog timer peripheral (IWDT), which has its own clock source which improves safety.
- Supports automatic hardware configuration after reset.

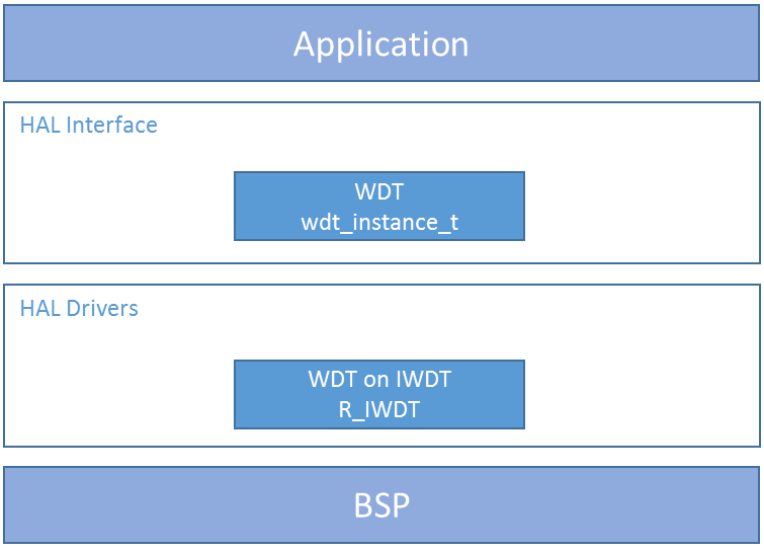


Figure 358: IWDT HAL Module Organization and Stack Implementations

4.2.28.3 IWDT HAL Module APIs Overview

The IWDT HAL module defines APIs for open, refresh, read, and get status. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows.

IWDT HAL Module Timer API Summary

Function Name	Example API Call and Description
cfgGet	g_wdt0.p_api->cfgGet(g_wdt0.p_ctrl, g_wdt0.p_cfg); Initialize the iWDT in register start mode. In auto-start mode with NMI output it registers the NMI callback.
open	g_wdt0.p_api->open(g_wdt0.p_ctrl, g_wdt0.p_cfg); Initialize the iWDT in register start mode. In auto-start mode with NMI output it registers the NMI callback.
refresh	g_wdt0.p_api->refresh(g_wdt0.p_ctrl); Refresh the watchdog timer.

Function Name	Example API Call and Description
statusGet	g_wdt0.p_api->statusGet(g_wdt0.p_ctrl, &status); Read the status of the iWDT.
statusClear	g_wdt0.p_api->statusClear(g_wdt0.p_ctrl, clear); Clear the status flags of the iWDT.
counterGet	g_wdt0.p_api->counterGet(g_wdt0.p_ctrl, &counter); Read the current iWDT counter value.
timeoutGet	g_wdt0.p_api->timeoutGet(g_wdt0.p_ctrl, &timeout); Read the watchdog timeout values.
versionGet	g_wdt0.p_api->versionGet(&version); Retrieve the API version using the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successfully executed.
SSP_ERR_ASSERTION	Null Pointer(s).
SSP_ERR_INVALID_ARGUMENT	One or more configuration options is invalid.
SSP_ERR_INVALID_MODE	An attempt to open the WDT in register-start mode when the OFS0 register is configured for auto-start mode. Or to open the WDT in auto-start mode when the OSF0 is configured for register start mode.
SSP_ERR_ABORTED	Invalid clock divider for this watchdog

NOTE: Lower level drivers may return Common Error Codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.28.4 IWDT HAL Module Operational Overview

The IWDT HAL module configures the IWDT Interface. When the IWDT underflows or is refreshed outside of the permitted refresh window, one of the following events can occur:

- Resetting of the device
- Generation of an NMI

The following figure shows an example of the operation of the IWDT. When refreshed in the valid refresh period of the counter the timer count value is reset. If the count is allowed to underflow or refresh occurs outside of the valid refresh period, the IWDT resets the device or generates an NMI.

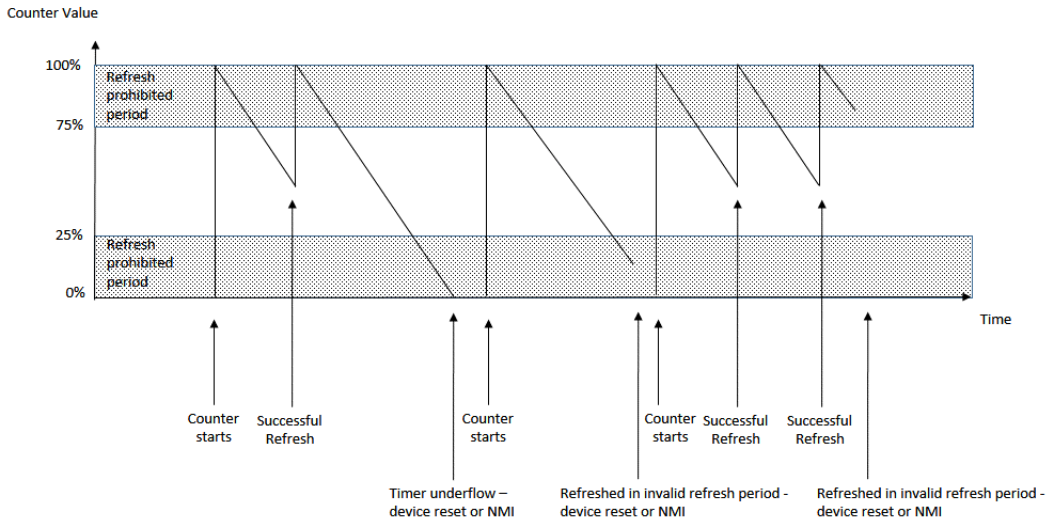


Figure 359: Independent Watchdog Timer Operational Diagram

All series of Synergy microcontrollers have an option-setting Memory which can be used to set the operating state of peripherals after a reset. The OFS can be used to set the state of the IWDT, WDT, LVD and CGC HOCO.

The following table details which parameters of the IWDT can be configured by the OFS registers.

NOTE: The IWDT can only be configured via the OFS registers. The IWDT does not support Register Start mode.

Control	Description
IWDT Start Mode Select	Automatically starts the IWDT after a Reset, if enabled.
IWDT Timeout Period	Specifies the IWDT timeout (number of clock cycles) 128 cycles 512 cycles 1024 cycles 2048 cycles
IWDT-Dedicated Clock Frequency Division Ratio	1 1/16 1/32 1/64 1/128 1/256
IWDT Window End Position	25% 50% 75% 100% (no window end position set)
IWDT Window Start Position	25% 50% 75% 100% (no window start position set)
IWDT Reset Interrupt Request	The IWDT can either generate an Interrupt Signal or a Reset signal.
IWDT Stop Control	The IWDT can continue to count or Stop counting in Low Power Mode.

NOTE: For further information on the contents of the OFS0 register see the Synergy MCU hardware manual.

The OFS register values are set in the Synergy Configuration editor via properties on the BSP tab.

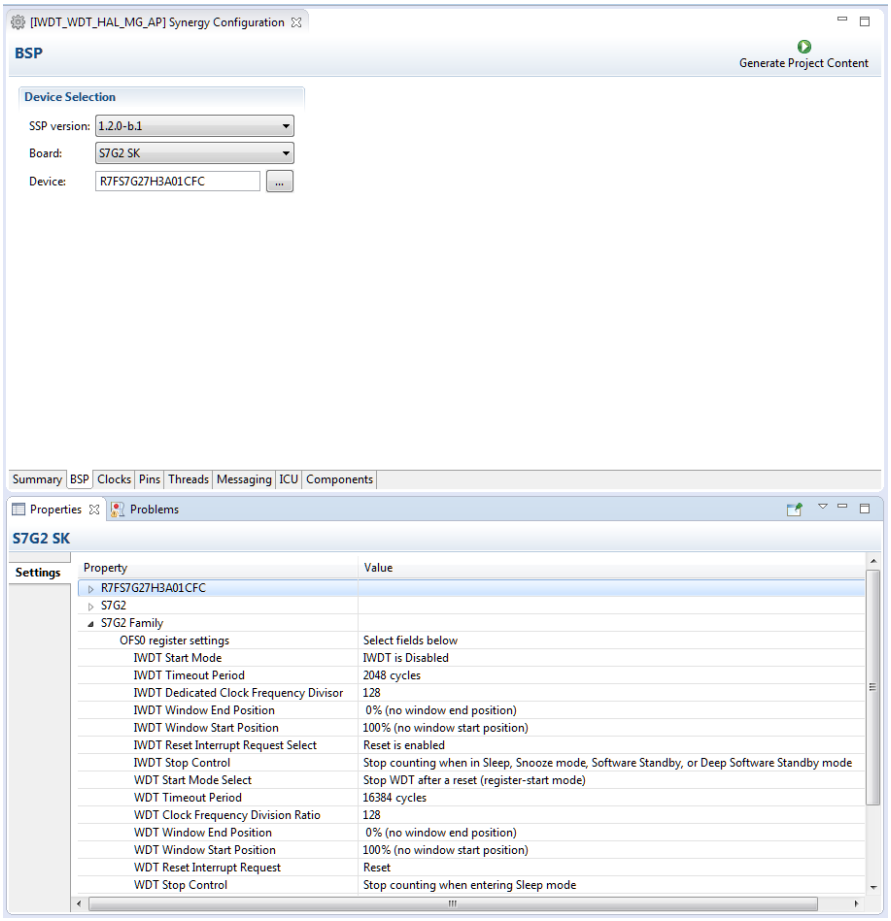


Figure 360: OFS Register Value Settings

4.2.28.5 IWDT HAL Module Important Operational Notes and Limitations

4.2.28.6 IWDT HAL Module Period Calculation

The IWDT operates from IWDTCLK. Assuming the largest parameters for the WDT and an IWDTCLK frequency of 15 kHz, the time from the last refresh to device reset or NMI generation is under 35 seconds. The clock specifications for the IWDT module period are as follows:

IWDTLCK = 15 kHz

Clock division ratio = IWTCLK/256

Timeout period = 2048 cycles

IWDT clock frequency = 15 kHz / 256 = 58.59 Hz

Cycle time = 1 / 58.59 Hz = 17.07 ms

Timeout = 17.07 ms x 2048 cycles = 34.95 seconds

4.2.28.7 Triggering DMAC/DTC with the IWDT HAL Module

To trigger a transfer of data using the DMAC or DTC peripheral when the watchdog counter underflows or when a refresh is attempted outside of the valid refresh period, configure the DMAC/DTC transfer with activation_source set to ELC_EVENT_IWDT_UNDERFLOW. See the appropriate module User Guide for additional information.

4.2.28.8 Triggering ELC Events with the IWDT HAL Module

The IWDT can trigger the start of other peripherals as available with the Event Link Controller (ELC). For details, see the ELC HAL module guide.

4.2.28.9 IWDT HAL Module Limitations

- When using a JLink debugger the IWDT counter does not count and therefore will not reset the device or generate an NMI.
- When there is no active task ready to run, ThreadX puts the MCU into sleep mode. If the IWDT is configured to stop the counter in low-power mode, then your application must restart the timer when used with the ThreadX RTOS.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.28.10 Including the IWDT HAL Module in an Application

This section describes how to include the IWDT HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack.

To add the IWDT HAL module to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the IWDT HAL module is r_iwdt as shown in the table below. This name can be changed in the associated properties window.)

Watchdog Timer Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_wdt0 IWDT HAL on r_iwdt	Threads	New Stack> Driver> Monitoring> IWDT HAL on r_iwdt

When the IWDT HAL module on r_iwdt is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower level modules. Any drivers that need additional configuration information will be box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

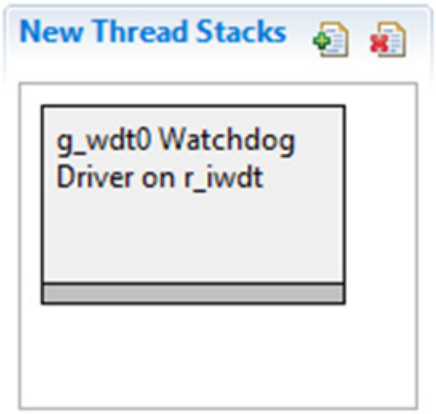


Figure 361:IWDT HAL Module Stack

4.2.28.11 Configuring the IWDT HAL Module

The IWDT HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections (such as interrupts or operating modes) which must be configured for lower level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the properties window of the associated module. Simply select the indicated module and then view the properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+.) This level of detail is not included in the configuration properties tables below, but is easily visible with the ISDE when configuring interrupt priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the Configuration Table Settings given below. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration settings for IWDT implementation are given in the below tables.

Configuration Settings for the IWDT HAL Module on r_iwdt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Include parameter checking code
Name	g_wdt0	Module Name

ISDE Property	Value	Description
NMI Callback	NULL	Callback. A user callback function can be registered in the open API. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the IRQn triggers.

Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system. |

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

4.2.28.12 Configure Option Function Select Register 0 (OFS0)

All series of Synergy microcontrollers have an Option-Setting Memory which can be used to set the operating state of peripherals after a reset. The OFS can be used to set the state of the IWDT, WDT, LVD and CGC HOCO.

4.2.28.13 Configure the Interrupts for the IWDT HAL Module

Use the ISDE to configure the IWDT interrupts in the same way as configuring the other options for the IWDT module. If the IWDT is configured to generate an NMI interrupt on underflow or invalid refresh the interrupt must be enabled in the BSP.

NOTE: To enable interrupts, set the priority of the IWDT > IWDT NMIUNDF N. This sets BSP_IRQ_CFG_IWDT_UNDERFLOW in ssp_cfg/bsp/bsp_irq_cfg.h to the priority level selected.

When the IWDT NMIUNDF N interrupt is enabled in the BSP, the corresponding ISR will be defined. The ISR will call a user callback function if one was registered in the open API.

4.2.28.14 IWDT HAL Module Clock Configuration

The IWDT has its own dedicated clock operating at a set frequency which cannot be modified.

4.2.28.15 IWDT HAL Module Pin Configuration

4.2.28.16 IWDT does not require pins for its operation.

4.2.28.17 Using the IWDT HAL Module in an Application

The typical steps in using the IWDT HAL module in an application are:

1) Register the IWDT NMI callback using the open API.
2) Read the configuration of the IWDT HAL module using the cfgGet API.
3) Refresh the independent watchdog timer using the refresh API.
4) Read the IWDT status flags using the statusGet API.
5) Clear the IWDT Status and error flags using the statusClear API.
6) Read the current count value of the IWDT using the counterGet API.
7) Read the timeout values of the IWDT HAL module using the timeoutGet API.

The above steps are illustrated in a typical operational flow in the following figure:

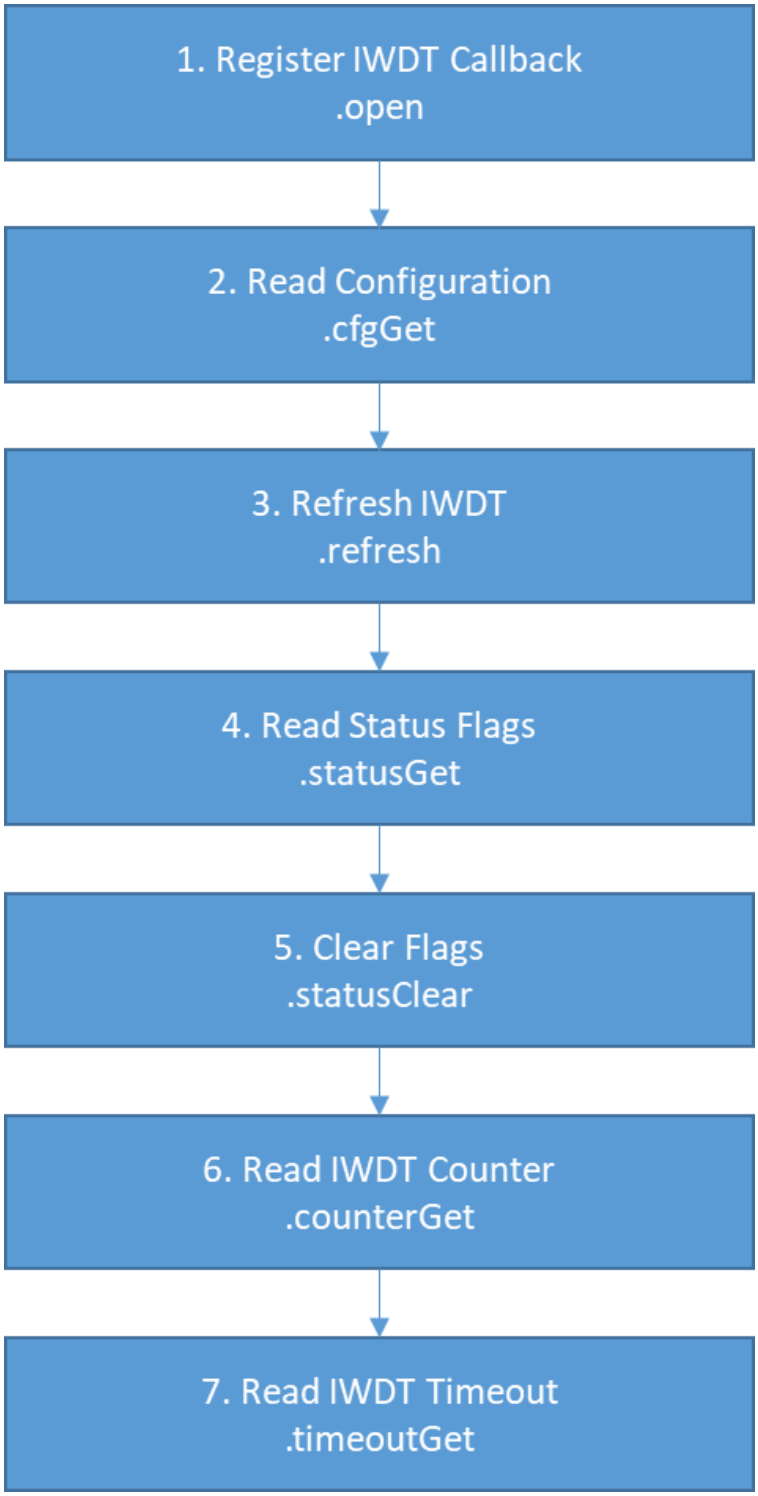


Figure 362: Flow Diagram of a Typical IWDT HAL Module Application

4.2.29 JPEG Decode Driver

The JPEG Decode HAL module provides high-level APIs for JPEG Decode image processing. The JPEG Decode HAL module uses the Synergy MCU JPEG Codec peripheral. A user callback function is available to inform the application program of key processing events.

4.2.29.1 JPEG Decode HAL Module Features

- Supports JPEG decompression.
- Supports polling mode that allows an application to wait for JPEG Decoder to complete.
- Supports interrupt mode with user-supplied callback functions.
- Configures parameters such as horizontal and vertical subsample values, horizontal stride, decoded pixel format, input and output data format, and color space.
- Obtains the size of the image prior to decoding it.
- Supports putting coded data in an input buffer and an output buffer to store the decoded image frame.
- Supports streaming coded data into JPEG Decoder module. This feature allows an application to read coded JPEG image from a file or from network without buffering the entire image.
- Configures the number of image lines to decode. This feature enables the application to process the decoded image on the fly without buffering the entire frame.
- Supports the input decoded format YCbCr444, YCbCr422, YCbCr420, YCbCr411.
- Supports the output format ARGB8888, RGB565.
- Returns error when the JPEG image's size, height and width don't meet the requirements.

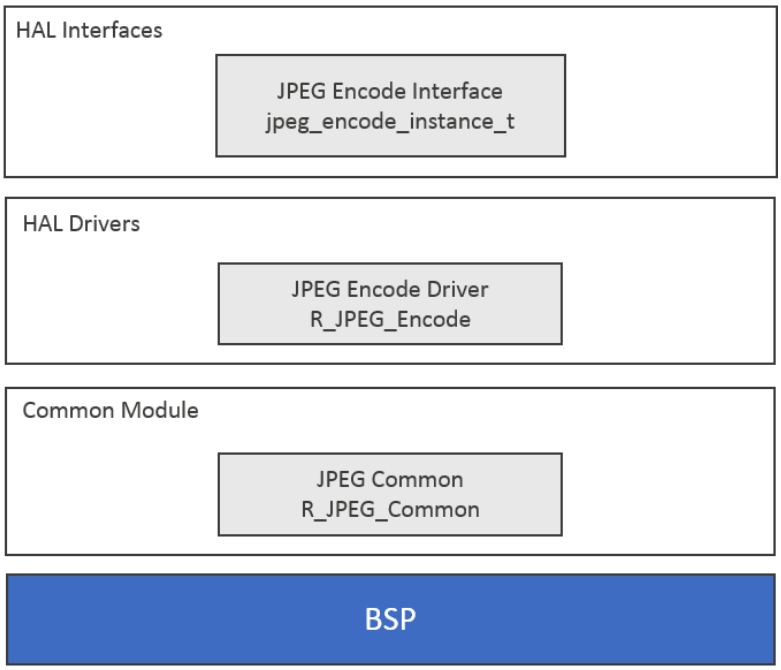


Figure 363: JPEG Decode HAL Module Block Diagram

4.2.29.2 JPEG Decode HAL Module APIs Overview

The JPEG Decode HAL Module implements APIs to open, set processing parameters, start processing, get processing statusand close the module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

JPEG Decode HAL Module API Summary

Function Name	Example API Call and Description
.open	g_jpeg_decode0.p_api->open(g_jpeg_decode0.p_ctrl, g_jpeg_decode0.p_cfg); Initial configuration.
.outputBufferSet	g_jpeg_decode0.p_api->outputBufferSet(g_jpeg_decode0.p_ctrl, p_buffer, buffer_size); Assign output buffer to JPEG *codec* for storing output data.

Function Name	Example API Call and Description
.horizontalStrideSet	g_jpeg_decode0.p_api->horizontalStrideSet(g_jpeg_decode0.p_ctrl, stride); Configure the horizontal stride value.
.imageSubsampleSet	g_jpeg_decode0.p_api->imageSubsampleSet(g_jpeg_decode0.p_ctrl, horizontal, vertical); Configure the horizontal and vertical *subsample* settings.
.inputBufferSet	g_jpeg_decode0.p_api->inputBufferSet(g_jpeg_decode0.p_ctrl, p_buffer, size); Assign input data buffer to JPEG *codec*.
.linesDecodedGet	g_jpeg_decode0.p_api->linesDecodedGet g_jpeg_decode0.p_ctrl, p_lines); Return the number of lines decoded into the output buffer.
.imageSizeGet	g_jpeg_decode0.p_api->imageSizeGet(g_jpeg_decode0.p_ctrl, p_horizontal, p_vertical); Retrieve image size during decoding operation.
.statusGet	g_jpeg_decode0.p_api->statusGet(g_jpeg_decode0.p_ctrl, p_status); Retrieve current status of the JPEG *codec* module.
.close	g_jpeg_decode0.p_api->close(g_jpeg_decode0.p_ctrl); Cancel an outstanding operation.
.versionGet	g_jpeg_decode0.p_api->versionGet(&version); Get version and store it in provided pointer p_version.

Function Name	Example API Call and Description
.pixelFormatGet	g_jpeg_decode0.p_api->pixelFormatGet(g_jpeg_decode0.p_ctrl, p_color_space); Get the input pixel format.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_INVALID_ALIGNMENT	Horizantal stride is not 8-byte aligned.
SSP_ERR_NOT_OPEN	Unit is not open
SSP_ERR_ASSERTION	An input pointer is NULL.
SSP_ERR_IN_USE	Peripheral is in use or hardware lock is taken.
SSP_ERR_HW_LOCKED	JPEG Codec resource is locked.
SSP_ERR_INVALID_CALL	An invalid call has been made, Codec output buffer address is attempted to change during codec operation. Or set output buffer first.
SSP_ERR_JPEG_IMAGE_SIZE_ERROR	Image size is not supported by JPEG codec.
SSP_ERR_JPEG_BUFFERSIZE_NOT_ENOUGH	Invalid buffer size
SSP_ERR_INVALID_MODE	JPEG Codec module is not decoding.
SSP_ERR_IMAGE_SIZE_UNKNOWN	The image size is unknown. More input data may be needed.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.2.29.3 JPEG Decode HAL Module Operational Overview

The JPEG Decoder HAL module can be used in the Input Buffer Streaming mode or JPEG Output Buffer Streaming mode.

Input Buffer Streaming Mode Operational Description

In this scenario the JPEG image data resides on a file, or is received from network. The HAL-layer driver is able to handle this scenario without requiring the input data to be stored in memory first.

Output Buffer Streaming Mode Operational Description

In this scenario the application needs to write the decoded image data to a file or to a network. The HAL-layer driver does not require the application to allocate memory for the entire frame. Instead the application may choose to decode one or more lines at a time. With this feature the amount of memory needed for the output data is greatly reduced.

MJPEG Decode operational Description

In this scenario, the application needs to display a continuous stream of JPEG images on the native display. The JPEG images can reside in a file or can be received from the network. The HAL driver can handle this by using Input Buffer Streaming Mode.

The basic flow to achieve this would be:

1) Open the JPEG driver.
2) Set the image parameters to: horizontal stride, image sub-samples.
3) Set the input buffer which holds the jpeg image frame.
4) Set the output buffer to hold the raw image.
5) Display the decoded image.
6) Close the JPEG driver
7) Start the process from step 1.

JPEG Decode HAL Module Important Operational Notes and Limitations

JPEG Decode Callbacks

A user callback function can be registered in the open API. If a user callback function is provided, the callback function will be called from the interrupt service routine (ISR) each time an interrupt happens. The argument of the callback function status can take the enumerated values listed below so that user can identify which event occurred in the decoding procedure.

Event Name Event Condition

JPEG_DECODE_STATUS_ERROR JPEG: Decode module encountered an error.

JPEG_DECODE_STATUS_IMAGE_SIZE_READY: JPEG Decode obtained the image size of data to be Decoded, and paused.

JPEG_DECODE_STATUS_INPUT_PAUSE: JPEG Decode paused waiting for more input data.

JPEG_DECODE_STATUS_OUTPUT_PAUSE: JPEG Decode paused after decoded the number of lines specified by user.

JPEG_DECODE_STATUS_DONE: JPEG Decode operation has successfully completed.

NOTE: Since a user callback function is called from an ISR, be careful not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

- The JPEG Decode HAL module only support JPEG decoding processing, for Encoding please JPEG Encode driver.
- In application if both driver is used, JPEG Encode driver needs to be close to use JPEG Decode driver or vice versa, as both driver shares the same IP.
- Refer to the most recent SSP Release notes for the most up to date limitations for this module.

4.2.29.4 Including the JPEG Decode HAL Module in an Application

This section describes how to include the JPEG Decode HAL Module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the JPEG Decode HAL Module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the JPEG Decode HAL Module is g_jpeg_decode0. This name can be changed in the associated Properties window.)

JPEG Decode HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_jpeg_decode0 JPEG Decode Driver on r_jpeg_decode	Threads	New Stack> Driver> Graphics> JPEG Dencode Driver

When the JPEG Decode HAL Module on r_jpeg_decode is added to the thread stack as shown in the following figure, the configurator automatically adds any lower-level drivers that are required. Any drivers that need additional configuration information will be box-text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

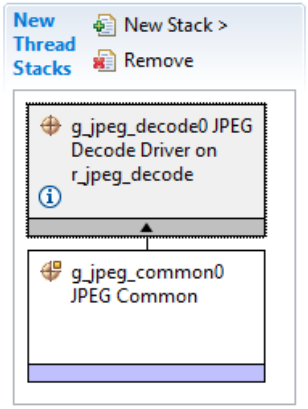


Figure 364: JPEG Decode HAL Module Stack

4.2.29.5 Configuring the JPEG Decode HAL Module

The JPEG Decode HAL Module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the JPEG Decode HAL Module on r_jpeg_decode

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_jpeg_decode0	The name to be used for a JPEG Decode module instance.

ISDE Property	Value	Description
Byte Order for Input Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2) Default: Normal Byte order	Specify the byte order for input data. The order is swapped as specified in every 8-byte.

ISDE Property	Value	Description
Byte Order for Output Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2) Default: Normal Byte order	Specify the byte order for output data. The order is swapped as specified in every 8-byte.
Output Data Color Format	Pixel Data RGB565 format, Pixel Data ARGBB888 format Default: Pixel Data RGB565 format	Specify the output data format.
Alpha value to be applied to decoded pixel data (only valid for ARGB8888 format)	255	Specify the alpha value for the output data format (only valid for ARGB8888 format).
Name of user callback function	NULL	Specify the name of user callback function.
Decompression Interrupt Priority	Priority 0 (highest), Priority 2:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Decompression interrupt priority selection.

ISDE Property	Value	Description
Data Transfer Interrupt Priority	Priority 0 (highest), Priority 2:14, Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Data transfer interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S1JA. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the JPEG Common Module

ISDE Property	Value	Description
Name	g_jpeg_common0	Module name

NOTE: The example values and defaults are for a project using the Synergy S1JA. Other MCUs may have different default values and available configuration settings.

JPEG Decode HAL Module Clock Configuration

The JPEG Decode HAL Module uses the PCLKA as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

JPEG Decode HAL Module Pin Configuration

The JPEG Decode HAL Module has no configurable input or output pins.

4.2.29.6 Using the JPEG Decode HAL Module in an Application

Once the module has been configured and the files generated, the JPEG Decode is ready to be used in an application. The typical steps in using the JPEG Decoder HAL module in an application are initializing the JPEG Decode using the open API, configure the horizontal stride, image sub-sample, input buffer and output buffer. Once the input and output buffers are set, the JPEG codec triggers the decode operation and stores the decoded image to the output buffer. The statusGet API can be used to poll the status of the JPEG operation.

The typical steps in using the JPEG Decode module in an application are:

1) Initialize the JPEG Decode using the open API.
2) Set the horizontal stride using the horizontalStrideSet API.
3) Set vertical and horizontal image sub-sample using the imageSubsampleSet API.
4) Set the input buffer address (which contains the JPEG image) using the inputBufferSet API.

5) Set the output buffer (should be large enough to hold the raw image data) using the outputBufferSet API.
6) The statusGet API can be used to get the JPEG operation. It returns an enumerated value (described above) to notify the user.
   a) The status JPEG_DECODE_STATUS_DONE returned from the statusGet API shows that the Decode operation is complete.
   b) The status JPEG_DECODE_STATUS_INPUT_PAUSE or JPEG_DECODE_STATUS_OUTPUT_PAUSE returned from the statusGet API shows that the Decode operation is not complete.
7) Operate on the received raw image data as needed by the application.
8) Close the module using the close API.

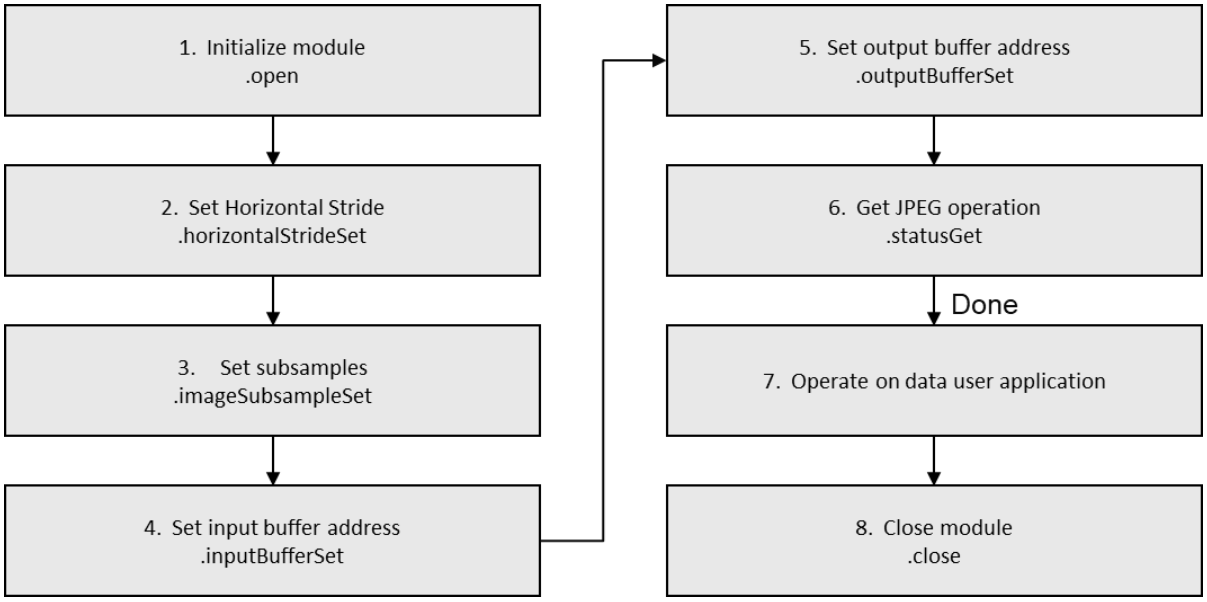


Figure 365: Flow Diagram of a Typical JPEG Decode HAL Module Application

4.2.30 JPEG Encode Driver

The JPEG Encode HAL module provides high-level APIs for JPEG Encode image. The JPEG Encode HAL module uses the Synergy MCU JPEG Codec peripheral. A user callback function is available to inform the application program of key processing events.

4.2.30.1 JPEG Encode HAL Module Features

- Supports JPEG Compression.
- Supports polling mode that allows an application to wait for JPEG Encoder to complete.
- Supports interrupt mode with user-supplied callback functions.
- Configures parameters such as horizontal and vertical resolution, horizontal stride, and Quality factor.

- Supports putting raw image data in an input buffer and an output buffer to store the encoded/compressed jpeg image.
- Supports streaming raw image data into JPEG Encoder module. This feature allows an application to read coded raw image from a capture device or camera or from network without buffering the entire image.
- Only supports the YCbCr422 color space to input.
- Support DRI Maker for RTP streaming application.
- Support quality factor configuration: The quality factor value determines the quality of output image.

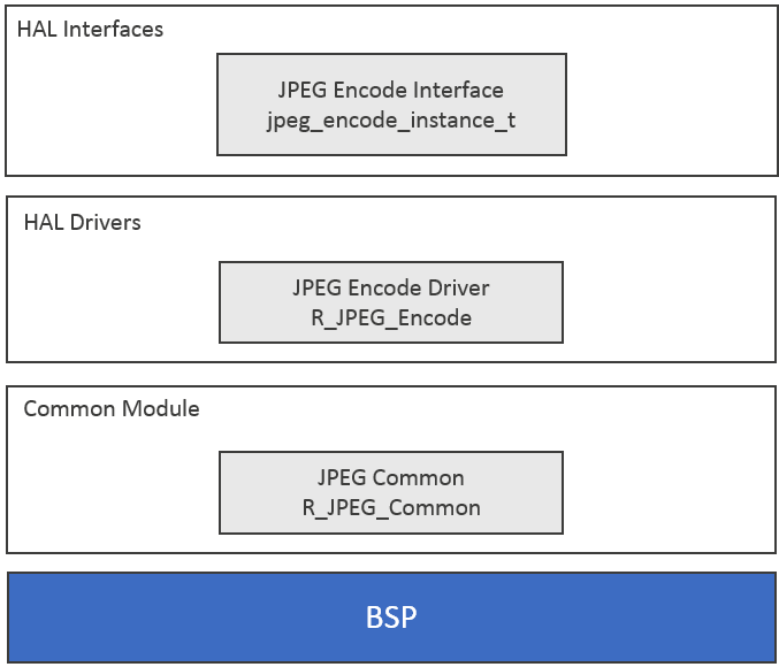


Figure 366: JPEG Encode HAL Module Block Diagram

4.2.30.2 JPEG Encode HAL Module APIs Overview

The JPEG Encode HAL Module defines APIs to open, set up processing parameters, process, get statusand close the module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

JPEG Encode HAL Module API Summary

Function Name	Example API Call and Description
.open	g_jpeg_encode0.p_api->open(g_jpeg_encode0.p_ctrl, g_jpeg_encode0.p_cfg); Initial configuration.

Function Name	Example API Call and Description
.imageParameterSet	g_jpeg_encode0.p_api->imageParameterSet(g_jpeg_encode0.p_ctrl, p_image_parameters); Set image parameters to JPEG Codec.
.outputBufferSet	g_jpeg_encode0.p_api->outputBufferSet(g_jpeg_encode0.p_ctrl, p_buffer); Assign output buffer to JPEG Codec for storing output data..
.inputBufferSet	g_jpeg_encode0.p_api->statusGet(g_jpeg_encode0.p_ctrl, p_buffer, buffer_size); Assign input data buffer to JPEG Codec.
.statusGet	g_jpeg_encode0.p_api->close(g_jpeg_encode0.p_ctrl); Retrieve current status of the JPEG Codec module.
.close	g_jpeg_encode0.p_api->read(&version); Cancel an outstanding operation.
.versionGet	g_jpeg_encode0.p_api->outputEnable(&version); Get version and store it in provided pointer p_version.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_INVALID_ALIGNMENT	Horizantal stride is not 8-byte aligned.

Name	Description
SSP_ERR_NOT_OPEN	Unit is not open
SSP_ERR_ASSERTION	An input pointer is NULL.
SSP_ERR_IN_USE	Peripheral is in use or hardware lock is taken.
SSP_ERR_HW_LOCKED	JPEG Codec resource is locked.
SSP_ERR_INVALID_CALL	An invalid call has been made, Codec output buffer address is attempted to change during codec operation. Or set output buffer first.
SSP_ERR_JPEG_IMAGE_SIZE_ERROR	Image size is not supported by JPEG codec.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.2.30.3 JPEG Encode HAL Module Operational Overview

The JPEG Encoder HAL module can be used in the Input Buffer Streaming mode or normal mode.

Input Buffer Streaming Mode Operational Description

In this scenario the raw image data coming from network, file or capturing device in chunks. The HAL-layer driver is able to handle this scenario without requiring the input data to be stored in memory first.

Normal Operational Description

In this scenario raw image coming from network, file or capturing device is a complete frame. The HAL-layer driver is able to handle this scenario and can compressed entire raw image frame.

MJPEG Encoding Operational Description

The JPEG Encoder can be used for Motion JPEG (MJPEG) format by configuring and using the driver in the following fashion:

1) Open the JPEG Encoder driver with the desired quality factor value (default = 50)
2) Set up the image resolution (optional if already configured in Thread stack window)
3) Initialize the Capturing device for capturing YCbCr422 image (like PDC camera Ov7670)
4) Set the output buffer to hold the jpeg image using the outputBufferSet API.
5) Capture the image.
6) Set the input buffer which holds the RAW YCbCr 422 image captured from the capturing device using the input-BufferSet API.
7) Check the status of the encode operation. If DONE send the image to rendering device
8) Continue the process from step 5.

JPEG Encode HAL Module Important Operational Notes and Limitations

JPEG Encode Callbacks

A user callback function can be registered in the open API. If a user callback function is provided, the callback function will be called from the interrupt service routine (ISR) each time an interrupt happens. The argument of the callback function status can take the enumerated values listed below so that user can identify which event occurred in the encoding procedure.

Event Name	Event Condition	
JPEG_ENCODE_STATUS_INPUT_PAUSE	JPEG Encode paused waiting for more input data.	
JPEG_ENCODE_STATUS_DONE		JPEG Encode operation has successfully completed.

NOTE: Since a user callback function is called from an ISR, be careful not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

- The JPEG Encode HAL module only support JPEG Encode processing. For decoding please use the JPEG Decode HAL module.
- In an Application where both the encode and decode modules are used, JPEG Decode module needs to be closed for the application to use JPEG Encode driver (or vice versa) as both modules share the same MCU peripheral.
- The JPEG Encode HAL module only supports the "Normal byte order" for output data format in the thread stack window. Other option may result in an invalid image.

4.2.30.4 Including the JPEG Encode HAL Module in an Application

This section describes how to include the JPEG Encode HAL Module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the JPEG Encode HAL Module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the JPEG Encode HAL Module is g_jpeg_encode0. This name can be changed in the associated Properties window.)

JPEG Encode HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_jpeg_encode0 JPEG Encode Driver on r_jpeg_encode	Threads	New Stack> Driver> Graphics> JPEG Encode Driver

When the JPEG Encode HAL Module on r_jpeg_encode is added to the thread stack as shown in the following figure, the configurator automatically adds any lower-level drivers that are required. Any drivers that need additional configuration information will be box-text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

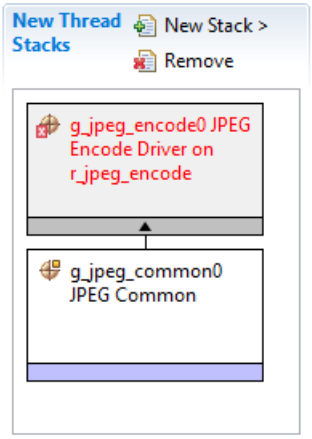


Figure 367: JPEG Encode HAL Module Stack

4.2.30.5 Configuring the JPEG Encode HAL Module

The JPEG Encode HAL Module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the JPEG Encode HAL Module on r_jpeg_encode

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_jpeg_encode0	Module name.
RAW Image Vertical Resolution	800	RAW image vertical resolution selection
RAW Image Horizontal Resolution	480	RAW image horizontal resolution selection
Byte Order for Input Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2) Default: Normal Byte order	Byte order for input data format selection

ISDE Property	Value	Description
Byte Order for Output Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2) Default: Normal Byte order	Byte order for output data format selection
Define Restart Marker	512	Define restart marker selection
Quality Factor	50	Quality factor selection
Name of user callback function	NULL	Name of user callback function selection
Decompression Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Decompression interrupt priority selection.

ISDE Property	Value	Description
Data Transfer Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Data transfer interrupt priority selection.

NOTE: The example values and defaults are for a project using the Synergy S1JA. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the JPEG Common Module

ISDE Property	Value	Description
Name	g_jpeg_common0	Module name.

NOTE: The example values and defaults are for a project using the Synergy S1JA. Other MCUs may have different default values and available configuration settings.

JPEG Encode HAL Module Clock Configuration

The JPEG Encode HAL Module uses the PCLKA as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

JPEG Encode HAL Module Pin Configuration

The JPEG Encode HAL module has no configurable input or output pins on the device.

4.2.30.6 Using the JPEG Encode HAL Module in an Application

Once the module has been configured and the files generated, the JPEG Encode is ready to be used in an application. The typical steps in using the JPEG Encode HAL module in an application are to initialize the JPEG Encode module using the open API, and configure the output buffer and the input buffer. Once the input and output buffers are set, the JPEG codec triggers the compression operation and stores the encoded image to the output buffer. The statusGet API can be used to poll the status of JPEG operation.

The typical steps in using the JPEG Encode module in an application are:

1) Initialize the JPEG Encode using open API.
   a) Configures the quality factor, horizontal stride, horizontal and vertical resolution.
2) Set the output buffer address (should be large enough to hold compressed jpeg image) using the outputBufferSet API.

3) Set the input buffer to start the encoding operation (address of raw image data and size) using the inputBufferSet API.
4) The statusGet API can be used to get the JPEG operation, the statusGet API return an enumerated value (described above) to notify user.
   a) Status JPEG_ENCODE_STATUS_DONE from the statusGet API shows that the encode operation is complete.
   b) Status JPEG_ENCODE_STATUS_INPUT_PAUSE from the statusGet API shows that driver is waiting for more input – Go to step 3 and set the input buffer with remaining data.
5) Operate on the received JPEG image data as needed by the application.
6) Close the module using the close API.

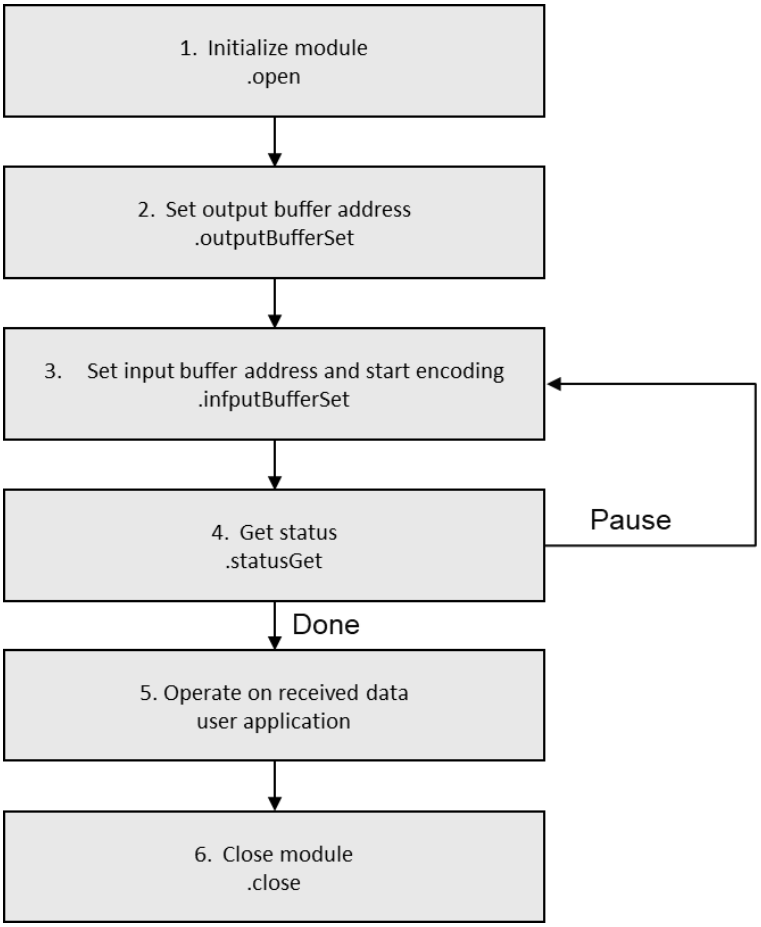


Figure 368: Flow Diagram of a Typical JPEG Encode HAL Module Application

4.2.31 Key Matrix Driver

The Key Matrix HAL module provides high-level APIs for Key Matrix HAL applications and is implemented on r_kint. The Key Matrix HAL module uses the key-interrupt function peripheral on the Synergy MCU. A user-defined callback can be created to inform the CPU of a key press event.

4.2.31.1 Key Matrix HAL Module Features

This Key Matrix HAL module configures and controls the Key Interrupt (KINT) peripheral. It implements the following key functions:

- Supports both rising and falling edges on KINT channels
- Supports interrupt-based event notification
- Supports a bit-masking function to capture multiple events efficiently
- Supports a matrix keypad with edges on any two channels

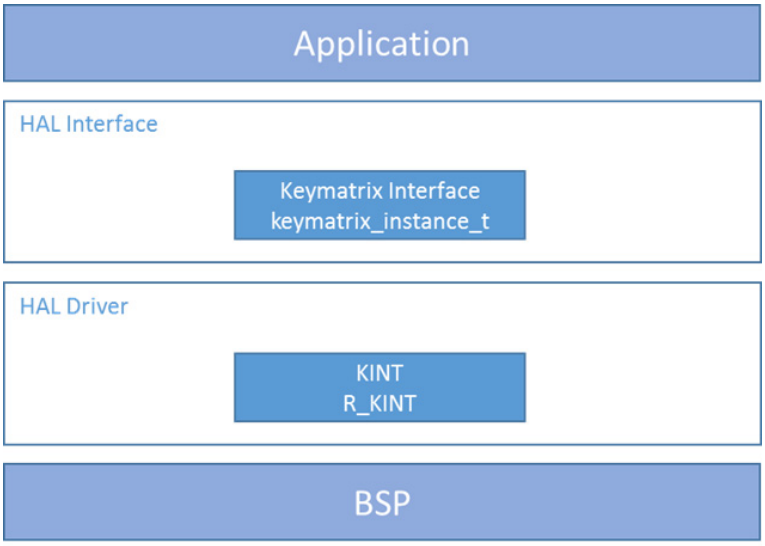


Figure 369: Key Matrix HAL Module Block Diagram

4.2.31.2 Key Matrix HAL Module APIs Overview

The Key Matrix HAL module defines APIs for opening, closing, enabling, and disabling key-interrupt functions. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Key Matrix HAL Module API Summary

Function Name	Example API Call and Description
.open	g_keymatrix_on_kint.p_api->open(&g_kint.p_ctrl, g_kint.p_cfg_cfg) Initial configuration.
.enable	g_keymatrix_on_kint.p_api->enable(&g_kint.p_ctrl) Enable Key interrupt.
.disable	g_keymatrix_on_kint.p_api->disable(g_kint.p_ctrl) Disable Key interrupt.
.triggerSet	g_keymatrix_on_kint.p_api->triggerSet() Set trigger for Key interrupt.
.close	g_keymatrix_on_kint.p_api->close(&g_ keymatrix) Allow driver to be reconfigured. May reduce power consumption.
.versionGet	g_keymatrix_on_kint.p_api->versionGet(&version) Get version and store it in provided pointer version.

NOTE: For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the *SSP User's Manual* API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successfully completed.
SSP_ERR_ASSERTION	Parameter has invalid value.
SSP_ERR_INVALID_ARGUMENT	Argument is invalid.

Name	Description
SSP_ERR_HW_LOCKED	The API has already been opened. It must be closed before it can be opened again.
SSP_ERR_NOT_OPEN	The peripheral is not opened.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual* API References for the associated module for a definition of all relevant status return values.

4.2.31.3 HAL Module Operational Overview

The Key Matrix HAL module configures the Key Interrupt (KINT) peripheral to detect rising or falling edges on any of the KINT channels. When such an event is detected on any of the configured pins, the module generates an interrupt; the interrupt then calls the user callback (p_callback) with the callback argument keymatrix_callback_args_t that specifies the channel(s) on which the edge was detected using a bitmask.

Even though detection of an edge on any one channel generates the interrupt, the callback returns a bitmask keymatrix_channels_t of all the pins that were triggered at that time if any other pins also detected an edge. Thus, an interrupt is not necessarily generated for edge detection on each pin if an edge was also detected on another pin before the callback was called. If a new edge is detected after the callback was called, then the interrupt is triggered again, resulting in a new callback. The bit mask in the user callback should be checked to identify the interrupt source channels.

This module can be used to implement a matrix keypad with edges on any two channels indicating the actual key that was pressed; alternatively, the module can be used as a single input to detect an edge on an input pin.

Key Matrix HAL Module Important Operational Notes and Limitations

- To trigger a transfer of data using the DMAC or DTC peripheral when a trigger edge is detected, configure the DMAC/DTC transfer with activation_source set to ELC_EVENT_KEY_INT.
- The KINT module can trigger the start of other peripherals available to the ELC. For details, see the ELC User's Guide in the *SSP User's Manual*.
- You must enable the KINT (INTKR) interrupt in the BSP for this module to operate, regardless of whether a callback is used in the open call.
- This module does not support polling-mode operation.
- Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.31.4 Including the Key Matrix HAL Module in an Application

This section describes how to include the Key Matrix HAL module in an application using the SSP configurator.

NOTE: It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the *SSP User's Manual* to learn how to manage each of these important steps in creating SSP-based applications.

To add the Key Matrix Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Key Matrix Driver is g_kint0. This name can be changed in the associated Properties window.)

Key Matrix HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_kint0 Key Matrix Driver on r_kint	Threads	New Stack> Driver> Input> Key Matrix Driver on r_kint

When the Key Matrix Driver on r_kint is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information is box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

Figure 370: Key Matrix HAL Module Stack

4.2.31.5 Configuring the Key Matrix HAL Module

The Key Matrix HAL module on r_kint must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the Integrated Solution Developer Environment (ISDE). This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the Properties window in the ISDE includes an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the following configuration table settings. This helps to orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Key Matrix HAL Module on r_kint

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enable or disable the parameter error checking.
Name	g_kint0	A user callback function can be registered using the open API. If this callback function is provided, it is called from the interrupt service routine (ISR) each time a configured edge is detected on any of the channels. NOTE: Without callback, the application cannot determine whether an event has occurred. Since the callback is called from an ISR, do not use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Keymatrix Channel Mask	Select Channels Below	This is a bit-mask with each bit specifying if that channel is to be enabled or not. Select the channels to be used.
Channel 0:7	Unused, Used (Default: Unused)	Channel 0:7 selection.
Trigger Type	Rising Edge, Falling Edge (Default: Rising Edge)	Specify if the enabled channels detect a rising edge or a falling edge. NOTE: Either all channels detecting a rising edge or all channels detecting a falling edge.

ISDE Property	Value	Description
Interrupt enabled after initialization	True, False (Default: False)	Specify if the module interrupts must be enabled as part of the open call.
Callback	NULL	Callback selection.
Interrupt Priority	Priority 0 (highest), 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 (lowest, not valid if using Thread X), Disabled (Default: Disabled)	Interrupt priority selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

Key Matrix HAL Module Clock Configuration

The Key Matrix HAL module does not require a specific clock configuration.

Key Matrix HAL Module Pin Configuration

The KINT peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the KINT pins.

Pin Selection for the Key Matrix HAL Module on r_kint

Resource	ISDE Tab	Pin selection Sequence
KINT	Pins	Select Peripherals > Input:KINT> KINT0

NOTE: The selection sequence assumes KINT0 is the desired hardware target for the driver.

Pin Configuration Settings for the Key Matrix HAL Module on r_kint

Property	Value	Description
Operation Mode	Disabled, Custom (Default: Disabled) Custom	Select Custom as the Operation Mode
KRM0:7	None, Pnn (Default: None)	Key Interrupt Pin selection

NOTE: The example settings are for a project using the Synergy S7G2 MCU Family and the SK-S7G2 Kit. Other Synergy MCUs and Synergy Kits may have different available pin configuration settings.

4.2.31.6 Using the Key Matrix HAL Module in an Application

The typical steps in using the Key Matrix HAL module in an application are:

1) Initialize the Key Matrix HAL module using the open API
2) If the autostart configuration setting is true, the module starts operation immediately
3) If the autostart is not set, use enable API to enable operation
4) Respond to key inputs
5) Disable operation using the disable API
6) To modify trigger edge after initialization, use the triggerSet API
7) Close the module by using the close API

The following figure illustrates these common steps are illustrated in a typical operational flow diagram.

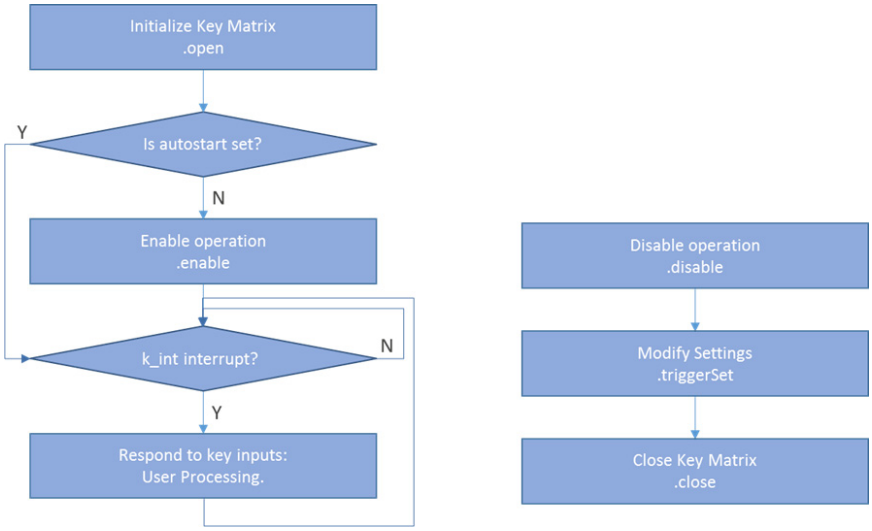


Figure 371: Flow Diagram of a Typical Key Matrix HAL Module Application

4.2.32 Low Power Modes Driver

The Low Power Modes HAL module provides high-level APIs for low-power mode applications and is implemented on r_lpm. The Low Power Modes HAL module uses the low-power mode hardware peripheral on the Synergy MCU.

4.2.32.1 Low Power Modes HAL Module Features

The LPM HAL module supports configuration of MCU operating power control modes and MCU low power modes using the Low Power Mode hardware peripheral.

The LPM driver supports the following operating power control modes:

- Low-voltage mode
- Low-speed mode
- Middle-speed mode
- High-speed mode
- Subosc-speed mode

The LPM driver supports the following low power modes:

- Deep Software Standby mode
- Software Standby mode
- Sleep mode
- Snooze mode

NOTE: Not all operating modes are available on all MCUs.

NOTE: Not all low power modes are available on all MCUs.

NOTE: This is a deprecated module and must not be used for new projects; use the lpm_v2 module instead.

This document is divided into the following sections which can be read independently (by a more experienced Synergy developer) or in series (by developers new to the Synergy Platform).

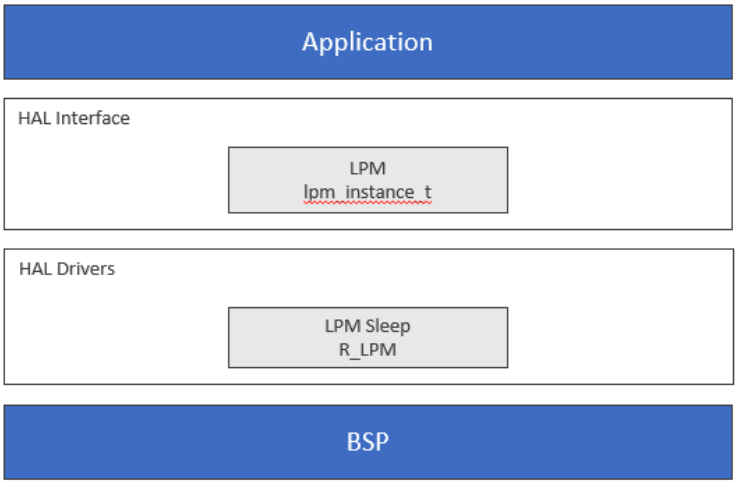


Figure 372: Low Power Modes Driver Organization, Options, and Stack Implementations

4.2.32.2 Low Power Modes HAL Module APIs Overview

The Low Power Modes Driver defines API's for initializing, setting and getting values and stopping modules. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of return status values is also provided.

Low Power Modes HAL Module API Summary

Function Name	Example API Call and Description
init	g_lpm0.p_api->init(g_lpm0.p_cfg); Open the LPM driver module Initialized the LPM block according to the passed in config structure.
mstpcrSet	g_lpm0.p_api->mstpcrSet(value1, value2, value3, value4); Set the value of all the Module Stop Control Registers.

Function Name	Example API Call and Description
mstpcrGet	g_lpm0.p_api->mstpcrSet(&value1, &value2, &value3, &value4); Get the values of all the Module Stop Control Registers.
moduleStop	g_lpm0.p_api->moduleStop(module); Stop a module.
moduleStart	g_lpm0.p_api->moduleStart(module); Run the specified module.
operatingPowerModeSet	g_lpm0.p_api->operatingPowerModeSet(power, osc); Set the operating power mode and oscillator.
snoozeEnable	g_lpm0.p_api->snoozeEnable(rdx0_mode, dtc_mode, requests, triggers); Configure and enable snooze mode.
snoozeDisable	g_lpm0.p_api->snoozeDisable(); Disable snooze mode.
lowPowerCfg	g_lpm0.p_api->lowPowerCfg(power_mode, output_port_enable, power_supply, io_port_state); Configure a low power mode.
wupenSet	g_lpm0.p_api->wupenSet(wupen_value); Set the value of the Wake Up Interrupt Enable Register WUPEN.
wupenGet	g_lpm0.p_api->wupenGet(&wupen_value); Get the value of the Wake Up Interrupt Enable Register WUPEN.

Function Name	Example API Call and Description
deepStandbyCancelRequestEnable	g_lpm0.p_api-> deepStandbyCancelRequestEnable (pin_signal, rising_falling); Enable a Deep Standby Cancel Request.
deepStandbyCancelRequestDisable	g_lpm0.p_api-> deepStandbyCancelRequestDisable (pin_signal); Disable a Deep Standby Cancel Request.
lowPowerModeEnter	g_lpm0.p_api-> lowPowerModeEnter (); Enter low power mode (sleep/standby/deep standby) using WFI macro. Function will return after waking from low power mode.
versionGet	g_lpm0.p_api->versionGet(&version); Retrieve the API version using the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables review the SSP Reference Manuals found as described in the Reference Section at the end of this document.

Error Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_ASSERTION	Parameter has invalid value.
SSP_ERR_INVALID_PTR	p_version is NULL.

NOTE: Lower level drivers may return Common Error Codes. Refer to the SSP Reference Manual available as described in the Reference Section at the end of this document for a definition of all relevant Error codes.

4.2.32.3 Low Power Modes HAL Module Operational Overview

The LPM HAL module APIs supports the configuration of MCU operating power control modes and MCU low power modes using the Low Power Mode hardware peripheral.

LPM Initialization

The LPM API function init should be called before calling any other LPM function. The init function handles initialization of internal variables and locks.

Sleep low-power mode

By default, at power on, sleep mode is enabled as the low-power mode. Sleep mode is the most convenient low-power mode available, as it does not require any special configuration (other than configuring and enabling a suitable interrupt or event to wake the MCU from sleep) to return to normal program-execution mode. The states of the SRAM, the processor registers, and the hardware peripherals are all maintained in sleep mode, and the time needed to enter and wake from sleep is minimal. Any interrupt causes the MCU device to wake from sleep mode, including the Systick interrupt used by the ThreadX® thread scheduler. The LPM API function init should be called before any other function. The LPM API function, lowPowerCfg, can be used to configure the MCU to use sleep as its low-power mode. The LPM API function lowPowerModeEnter should be used to directly enter sleep mode.

```
/* HAL-only entry function */

#include "hal_data.h"

void hal_entry(void)

{

/* Not shown: Enable any interrupt to wake from sleep mode */

/* Configure LPM peripheral for sleep mode */

g_lpm_on_hal.p_api->lowPowerCfg(LPM_LOW_POWER_MODE_SLEEP,

LPM_OUTPUT_PORT_ENABLE_RETAIN,

LPM_POWER_SUPPLY_DEEPCUT0,

LPM_IO_PORT_NO_CHANGE);

/* Enter sleep mode */

g_lpm_on_hal.p_api->lowPowerModeEnter();

}
```

Software Standby Mode

In software standby mode, the CPU, most of the on-chip peripheral functions, and all of the internal oscillators are stopped. However, the contents of the CPU internal registers and SRAM data, the states of on-chip peripheral functions and the I/O Ports are retained. Software Standby Mode allows significant reduction in power consumption because most of the oscillators are stopped in this mode. Like sleep mode, standby mode also requires that an interrupt or event be configured and enabled in order to wake from standby mode. See the table Interrupt Sources To Transition To Normal Mode from Snooze and Software Standby Mode in the MCU Synergy Hardware User's Manual for the complete list of events and

interrupts that can wake the MCU from sleep mode. The Wake Up Interrupt Enable Register (WUPEN) must be modified before entering software standby mode. See Wake Up Interrupt Enable Register (WUPEN) and section Software Standby Mode of MCU Synergy User's Manual: Microcontrollers for more information.

NOTE: The ThreadX idle thread and the ThreadX function tx_thread_sleep() expect the MCU to be configured for sleep as the low power mode. See the following usage notes for more information.

```
/* HAL-only entry function */

#include "hal_data.h"

#define WUPEN_AGT1_UNDERFLOW_MASK (1 << 28)

void hal_entry(void)

{

uint32_t wupen_value = 0;

/* Not shown: Configuration of the interrupt or event to wake from standby */

/* Set bit in WUPEN register to allow mcu to wake from standby through a specifi
c interrupt or event. AGT1 underflow is used in this example. */

/* Get current value of WUPEN register */

g_lpm.p_api->wupenGet(&wupen_value);

/* Set wake by AGT1 underflow bit in WUPEN register */

g_lpm.p_api->wupenSet(wupen_value | WUPEN_AGT1_UNDERFLOW_MASK);

/* Configure LPM peripheral for standby mode */

g_lpm.p_api->lowPowerCfg(LPM_LOW_POWER_MODE_STANDBY,

LPM_OUTPUT_PORT_ENABLE_RETAIN,

LPM_POWER_SUPPLY_DEEPCUT0,

LPM_IO_PORT_NO_CHANGE);

/* Enter standby mode */

g_lpm.p_api->lowPowerModeEnter();

}
```

Snooze Mode with Software Standby Mode

Snooze mode can be used with some MCU peripherals to execute basic tasks while keeping the MCU in a low power state. ADC, DTC, and other peripherals can be enabled in Snooze mode. The following code shows the process of enabling Snooze mode and running DTC while in Snooze. The process of setting up the timer AGT1 is not shown. Once the MCU is in Software Standby mode, Snooze is entered when the Compare Match A of AGT1 is detected. The MCU will leave Snooze mode when the DTC transfer is complete. For more details about the DTC, please refer to the DTC usage notes. The ISDE configures this instance structure when you include the g_transfer_on_dtc driver and generate the project in the ISDE generated source file.

NOTE: The ThreadX idle thread and the ThreadX function tx_thread_sleep() expect the MCU to be configured for sleep as the low power mode. See the following usage notes for more information.

```
#define WUPEN_AGT1_UNDERFLOW_MASK (1 << 28)

/* DTC settings */

g_transfer.p_cfg->p_info->p_src = &my_tx_data[0];

g_transfer.p_cfg->p_info->p_dest = &my_rx_data[0];

g_transfer.p_cfg->p_info->length = TRANSFER_SIZE;

g_transfer.p_api->open(g_transfer.p_ctrl, g_transfer.p_cfg);

/* snooze settings */

lpm_snooze_rxd0_t rdx0_mode = LPM_SNOOZE_RXD0_FALLING_EDGE_IGNORE;

lpm_snooze_dtc_t dtc_mode = LPM_SNOOZE_DTC_ENABLE;

lpm_snooze_request_t requests = LPM_SNOOZE_REQUEST_AGT1_COMPARE_A;

lpm_snooze_end_t triggers = LPM_SNOOZE_END_DTC_TRANS_COMPLETE;

g_lpm.p_api->snoozeEnable(rdx0_mode, dtc_mode, requests, triggers);

/* standby settings */

lpm_low_power_mode_t power_mode = LPM_LOW_POWER_MODE_STANDBY;

lpm_output_port_enable_t output_port_enable = LPM_OUTPUT_PORT_ENABLE_RETAIN;

lpm_power_supply_t power_supply = LPM_POWER_SUPPLY_DEEPCUT0;

lpm_io_port_t io_port_state = LPM_IO_PORT_RESET;

g_lpm.p_api->lowPowerCfg(power_mode, output_port_enable, power_supply, io_port_s
tate);
```

```
/* Clear WUPEN */

g_lpm.p_api->wupenSet(0);

/* Wake from AGT1 interrupt underflow */

g_lpm.p_api->wupenSet(WUPEN_AGT1_UNDERFLOW_MASK);
```

Deep Software Standby Mode

Deep Software Standby Mode is only available on the S7G2 MCU, and is a low power mode than Software Standby. The MCU always wakes from Deep Software Standby Mode by going through reset, either by the negation of the reset pin or by one of a set of wake up events as described by the API typedef lpm_deep_standby_t. The following code demonstrate how to enable and enter Deep Software Standby Mode using AGT1 underflow to wake the MCU from Deep Standby Mode. The MCU will reset once AGT1 expires and cause the MCU to reset.

NOTE: The ThreadX idle thread and the ThreadX function tx_thread_sleep() expect the MCU to be configured for sleep as the low power mode. See the following usage notes for more information.

```
#define WUPEN_AGT1_UNDERFLOW_MASK (1 << 28)

/* Deep standby wake signal (wake will cause reset) */

g_lpm.p_api->deepStandbyCancelRequestEnable(LPM_DEEP_STANDBY_AGT1,

LPM_CANCEL_REQUEST_EDGE_RISING);

/* Deep standby settings */

lpm_low_power_mode_t power_mode = LPM_LOW_POWER_MODE_DEEP;

lpm_output_port_enable_t output_port_enable = LPM_OUTPUT_PORT_ENABLE_RETAIN;

lpm_power_supply_t power_supply = LPM_POWER_SUPPLY_DEEPCUT0;

lpm_io_port_t io_port_state = LPM_IO_PORT_RESET;

g_lpm.p_api->lowPowerCfg(power_mode, output_port_enable, power_supply, io_port_s
tate);

/* Clear WUPEN */

g_lpm.p_api->wupenSet(0);

/* Wake from AGT1 interrupt underflow */

g_lpm.p_api->wupenSet(WUPEN_AGT1_UNDERFLOW_MASK);
```

```
g_lpm.p_api->lowPowerModeEnter();
```

Operational Notes Low Power Modes HAL Module

Using this module to change operating modes requires use of the CGC module. Please review the usage notes for the CGC driver as well. Please review the section "Function for Lower Operating Power Consumption" of the MCU Synergy Hardware User's Manual for detailed information about the sequence of events required to properly change the MCU's operating mode.

Using this driver to configure the LPM peripheral based on an interrupt requires the use of the BSP to configure and enable the interrupt.

Calling the LPM API function init only sets the operating mode of the MCU. The init function does not configure the low power modes.

Use the LPM API function lowPowerCfg to configure the low power mode. Once configured, the LPM API function lowPowerModeEnter can be used at any time to enter the low power mode.

Using deep standby or snooze requires additional configuration of the LPM peripheral via the LPM API functions deepStandbyCancelRequestEnable and snoozeEnable.

You must modify the Wake Up Interrupt Enable Register (WUPEN) before entering Software Standby mode. See Wake Up Interrupt Enable Register (WUPEN) and section "Software Standby Mode of the MCU Synergy User's Manual: Microcontrollers" for more information.

To enter the sub-oscillator Power control mode, you must stop the MOCO and HOCO clocks using the CGC module prior to calling the API function operatingPowerModeSet.

If the main oscillator or PLL with main oscillator source is used for the system clock, the wake time from standby mode can be affected by the Main Oscillator Wait Time Setting in the MOSCWTCR register. This register setting is available to be changed through the Main Oscillator Wait Time setting in the CGC driver properties. See the "Wakeup Timing and Duration" table in Electrical Characteristics for more information.

Important: When a project uses ThreadX, the application should only switch to LPM_LOW_POWER_MODE_STANDBY or LPM_LOW_POWER_MODE_DEEP immediately before calling the API function lowPowerModeEnter. If LPM_LOW_POWER_MODE_STANDBY or LPM_LOW_POWER_MODE_DEEP are used with ThreadX then the user must make sure to revert the low power mode to LPM_LOW_POWER_MODE_SLEEP immediately after the MCU wakes from LPM_LOW_POWER_MODE_STANDBY.

Limitations Low Power Modes HAL Module

- Flash stop (code flash disable) is not supported. See the section, "Flash Operation Control Register (FLSTOP)" in the S124, S128, S3A3, or S3A7 Synergy Hardware User's Manual.
- Reduced SRAM retention area in software standby mode is not supported. See the section, "Power Save Memory Control Register (PSMCR)" of the S3A7 or S3A3 Synergy Hardware User's Manual.
- The MCU may not enter or stay in Software Standby and Deep Software Standby modes with the debugger attached. Instead, the MCU may be woken from Software Standby and Deep Software Standby modes by the debugger. To properly test and verify Software Standby and Deep Software Standby modes, the debugger must not be attached.
- If the main oscillator or PLL with main oscillator source is used for the system clock, the wake time from standby mode can be affected by the Main Oscillator Wait Time Setting in the MOSCWTCR register. This register setting is available to be changed through the Main Oscillator Wait Time setting in the CGC HAL module properties. See the "Wakeup Timing and Duration" table in Electrical Characteristics for more information.
- Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.32.4 Including the Low Power Modes Driver in an Application

This section describes how to include the Low Power Modes Driver in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual available as described in the Reference Section at the end of this document to learn how to manage each of these important steps in creating SSP based applications.

To add the Low Power Modes Driver to an application, simply add it to a thread using the Stacks Selection Sequence given in the table below. (The default name for the Low Power Modes Driver is <name> and this is shown in the following table. This name can be changed in the associated **Properties** window). When the driver is added it shows up in the thread as illustrated in the following figure.

Low Power Modes HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_lpm0 Low Power Modes Driver on r_lpm	**Threads**	**New Stack> Driver> Power> Low Power Modes on r_lpm**

When the Low Power Modes HAL module on r_lpm is added to the Thread Stack, as shown in the figure below, the configurator automatically adds the needed lower level drivers. Any drivers that need additional configuration information will be box text highlighted in red. Modules with a gray band are individual modules that stand alone. Modules with a blue band are shared or common and need only be added once, since they can be used by multiple stacks.

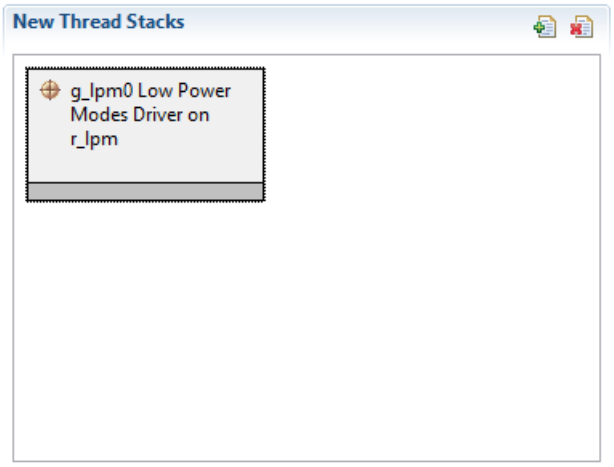


Figure 373: Low Power Modes HAL Module Stack

4.2.32.5 Configuring the Low Power Modes HAL Module

The Low Power Modes HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify, by highlighting the block in red, any required configuration selections, such as Interrupts or operating modes, which must be configured for lower level modules, for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are

'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the **Properties** tab within the SSP Configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the Interrupt Priority. This configuration setting is available with the **Properties** window of the associated module. Simply select the indicated module and then view the **Properties** window. The Interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the Interrupt Priorities listed in the **properties** window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring Interrupt Priority levels.

NOTE: You may want to open your ISDE and create the Low Power Modes Driver and explore the property settings in parallel with looking over the Configuration Table Settings given below. This will help orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for Low Power Mode Deep Standby Driver on r_lpmv2

ISDE Property	Setting	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enables or disables the parameter checking
Name	g_lpm0	Module name
Operating power mode	High speed operating mode, Middle speed operating mode, Low speed operating mode, Low voltage operating mode (Default: High speed operating mode)	Select operating power mode
Sub oscillator mode	Sub oscillator mode not enabled, Sub oscillator mode enabled (Default: Sub oscillator mode not enabled)	Select sub oscillator mode

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

LPM Clock Configuration

The Low Power Mode module does not have a configurable clock input.

LPM Pin Configuration

The Low Power Mode module does not have specific input and output pins that need configuration.

4.2.32.6 Using the Low Power Modes Driver in an Application

The typical steps in using the Low Power Modes HAL module in an application are:

1) Initialize the Low Power Modes HAL module using the init API.
2) Configure a low-power mode with the lowPowerCfg API.
3) Enter a low-power mode with the lowPowerModeEnter API.

These common steps are illustrated in a typical operational flow diagram in the following figure.

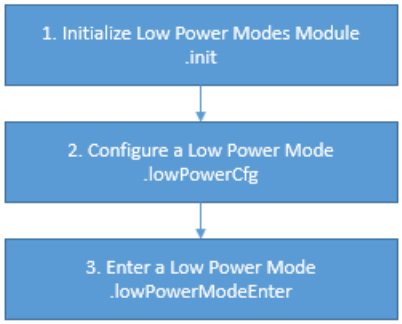


Figure 374: Flow Diagram of a Typical Low Power Modes Driver Application

4.2.33 Low Power Modes V2 Driver

The Low Power Modes V2 HAL module provides high-level APIs for low-power mode applications and is implemented on r_lpm2. The Low Power Modes HAL module uses the low-power mode hardware peripheral on the Synergy MCU.

4.2.33.1 Low Power Modes V2 HAL Module Features

The LPM V2 HAL module supports configuration of MCU operating power-control modes and MCU low-power modes using the low-power mode hardware peripheral.

The LPM V2 HAL module supports the following low power modes:

- Deep Software Standby mode
- Software Standby mode
- Sleep mode
- Snooze mode

The LPM V2 HAL module supports reducing power consumption when in deep stand-by mode through internal power-supply control and by resetting the states of I/O ports. The LPM V2 HAL module supports disabling and enabling of the MCU's other hardware peripherals.

NOTE: Not all low-power V2 modes are available on all MCUs Groups.

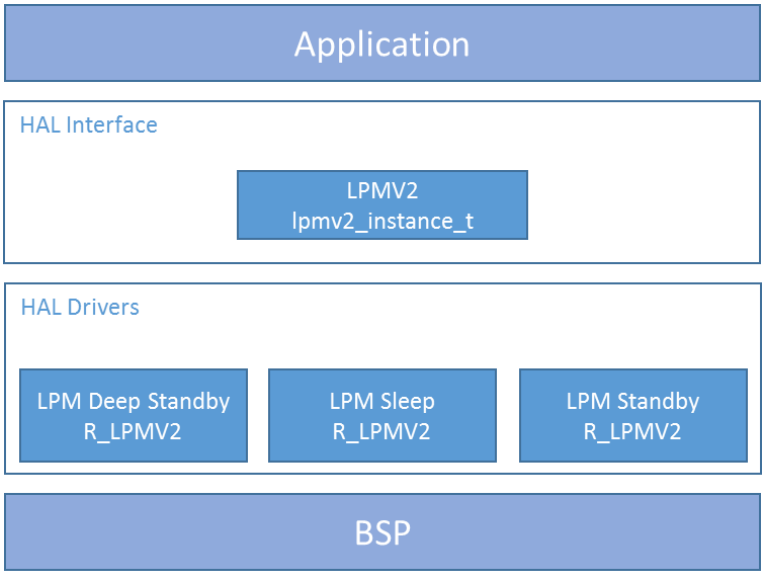


Figure 375: Low Power Modes V2 Module Block Diagram

4.2.33.2 Low Power Modes V2 HAL Module APIs Overview

The Low Power Modes V2 HAL module defines APIs for configuring operations and enabling and disabling low-power operations. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of return status values follows the API summary table.

NOTE: The Low Power Modes V2 HAL module will no longer handle operating power-control modes of the MCU; these are now handled by the CGC HAL module.

The following API examples illustrate sleep-mode use; "deep_standby" and "standby" can be substituted for "sleep" in the API examples to create examples for those modes.

Low Power Modes V2 Module API Summary

Function Name	Example API Call and Description
init	g_lpmv2_sleep0.p_api->init(g_lpmv2_sleep0.p_cfg); Open the LPM driver module Initialized the LPM block according to the passed in config structure.

Function Name	Example API Call and Description
lowPowerCfg	g_lpmv2_sleep0.p_api->lowPowerCfg(g_lpmv2_sleep0.p_cfg); Configure a low power mode.
lowPowerModeEnter	g_lpmv2_sleep0.p_api->lowPowerModeEnter(void); Enter low power mode (sleep/standby/deep standby) using WFI macro. Function will return after waking from low power mode.
versionGet	g_lpmv2_sleep0.p_api->versionGet(&version); Get the driver version and place it at the pointer version.

NOTE: For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_INVALID_POINTER	Pointer is NULL
SSP_ERR_INVALID_MODE	Invalid settings for specified mode
SSP_ERR_INVALID_HW_CONDITION	OPCMTSF and SOPCMTSF flags are not cleared within internally set timeout.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.33.3 Low Power Modes V2 HAL Module Operational Overview

LPM V2 Initialization

The LPM V2 API function lpm_v2_api_t::init should be called before calling any other LPM V2 function. The init function handles initialization of internal variables and locks.

Sleep low-power mode

By default, at power on, sleep mode is enabled as the low-power mode. Sleep mode is the most convenient low-power mode available, as it does not require any special configuration (other than configuring and enabling a suitable interrupt or event to wake the MCU from sleep) to return to normal program-execution mode. Any interrupt wakes the MCU device from sleep low-power mode. The states of the SRAM, the processor registers, and the hardware peripherals are all maintained in sleep mode, and the time needed to enter and wake from sleep is minimal. Any interrupt causes the MCU device to wake from sleep mode, including the Systick interrupt used by the ThreadX® thread scheduler. The LPM API function lpm_v2_api_t::init should be called before any other function. The LPM API function, lpm_v2_api_t::lowPowerCfg, can be used to configure the MCU to use sleep as its low-power mode. The LPM API function lpm_v2_api_t::lowPowerModeEnter should be used to directly enter sleep mode.

The following code example configures sleep as a low-power mode and enters low-power mode sleep. In this example, the LPM V2 sleep module uses the name g_lpmv2_sleep0.

```
/* HAL-only entry function */
#include "hal_data.h"
void hal_entry(void)
{
ssp_err_t error = SSP_SUCCESS;
/* Initialize the LPM V2 Driver */
error = g_lpmv2_sleep0.p_api->init();
/* Handle error if any */
/* Configure LPM peripheral for sleep mode */
error = g_lpmv2_sleep0.p_api->lowPowerCfg(g_lpmv2_sleep0.p_cfg);
/* Handle error if any */
/* Entry sleep mode */
error = g_lpmv2_sleep0.p_api->lowPowerModeEnter();
/* Handle error if any */
}
```

Software Standby Mode for LPM V2

In software-standby mode, the CPU, as well as most of the on-chip peripheral functions and all of the internal oscillators, are stopped. Retained are the contents of the CPU internal registers and SRAM data, the states of on-chip peripheral functions, and I/O Ports. Software-standby mode allows significant reduction in power consumption, because most of the oscillators are stopped in this mode. Like sleep mode, standby mode requires an interrupt or event be configured and enabled to wake from standby mode.

The possible triggers for waking from standby mode are enumerated in the **Properties** window for convenience; multiple triggers can be enabled.

The following code example is for configuring standby as the low-power mode and entering low-power mode standby. In this example, the LPM V2 standby module with name g_lpmv2_standby0 is used. The code for using the standby module with snooze enabled is identical.

```
/* HAL-only entry function */
#include "hal_data.h"
void hal_entry(void)
{
```

```
ssp_err_t error = SSP_SUCCESS;
/* Initialize the LPM V2 Driver */
error = g_lpmv2_standby0.p_api->init();
/* Handle error if any */
/* Configure LPM peripheral for standby mode */
error = g_lpmv2_standby0.p_api->lowPowerCfg(g_lpmv2_standby0.p_cfg);
/* Handle error if any */
/* Entry standby mode */
error = g_lpmv2_standby0.p_api->lowPowerModeEnter();
/* Handle error if any */
}
```

Snooze Mode with Software Standby Mode for LPM V2

Snooze mode is available through the standby mode LPM V2 instance. Choose "Standby with Snooze Enabled" for "Choose the low power mode" in the **Properties** window. Snooze mode can be used with some MCU peripherals to execute basic tasks, while keeping the MCU in a low-power state. The snooze settings are below the standby settings in the **Properties** window. The ADC, DTC, and other peripherals can be enabled in snooze mode. All the settings for snooze are available through configuration properties for the standby instance, with the exception of the Event Link Controller settings for registers SELSR0 and IELSRn. Snooze is considered an advanced feature.

The Snooze Mode Settings are only used if the low-power mode choice is **Standby with Snooze Enabled** as shown in the following screen capture:

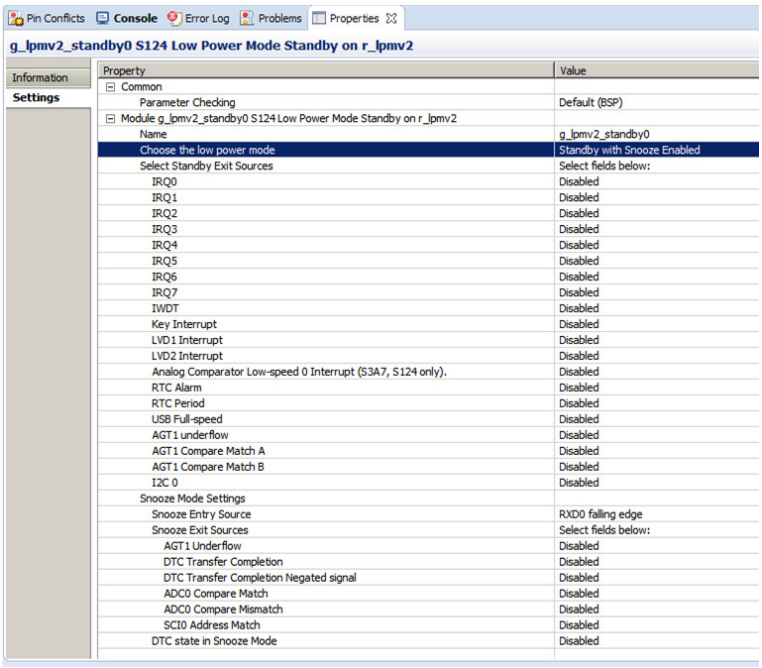
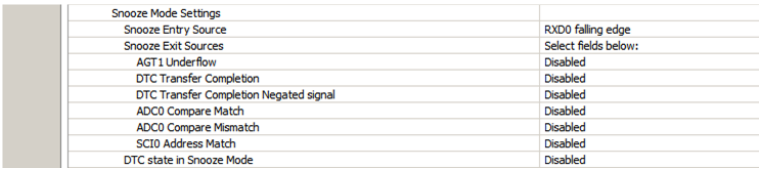


Figure 376: Standby mode with Snooze setting enabled

Snooze is a feature of standby mode that allows some peripherals to run even though the MCU core is not executing instructions. The low-power mode peripheral options related to snooze mode are shown in the following image. Only one snooze-entry source can be enabled; multiple snooze-exit sources can be enabled. The DTC peripheral can be enabled in snooze mode as well.



Snooze Mode Settings	
Snooze Entry Source	RXD0 falling edge
Snooze Exit Sources	Select fields below:
AGT1 Underflow	Disabled
DTC Transfer Completion	Disabled
DTC Transfer Completion Negated signal	Disabled
ADC0 Compare Match	Disabled
ADC0 Compare Mismatch	Disabled
SCI0 Address Match	Disabled
DTC state in Snooze Mode	Disabled

Figure 377: Snooze Mode Settings

Deep Software Standby Mode for LPM V2

Deep Software Standby Mode is only available on some MCU devices. The MCU device always wakes from Deep Software Standby Mode by going through reset, either by the negation of the reset pin or by one of a set of wake up events displayed in the configuration **Properties** window for the LPM deep standby instance.

The possible triggers for waking from deep standby mode are enumerated in the **Properties** window for convenience. Multiple triggers can be enabled. Some triggers have an associated edge type, falling or rising. These options are enumerated also as shown above and below.

The following code example is for configuring deep standby as the low power mode and entering low power mode deep standby. In this example, the LPM V2 deep standby module with name g_lpmv2_deep_standby0 is used.

```
/* HAL-only entry function */
#include "hal_data.h"
void hal_entry(void)
{
ssp_err_t error = SSP_SUCCESS;
/* Initialize the LPM V2 Driver */
error = g_lpmv2_deep_standby0.p_api->init();
/* Handle error if any */
/* Configure LPM peripheral for deep sleep mode */
error = g_lpmv2_deep_standby0.p_api->lowPowerCfg(g_lpmv2_deep_standby0.p_cfg);
/* Handle error if any */
/* Entry deep sleep mode */
error = g_lpmv2_deep_standby0.p_api->lowPowerModeEnter();
/* Handle error if any */
```

Low Power Modes V2 HAL Module Important Operational Notes and Limitations

Using this driver to configure the LPM peripheral to wake the MCU from standby mode through interrupts requires the interrupt to be configured and enable by the peripheral driver or framework that uses that interrupt. For example, to wake from standby through AGT1 underflow, that interrupt must be enabled through the configuration of the AGT timer module.

If the main oscillator or PLL with main-oscillator source is used for the system clock, the wake time from standby mode can be affected by the Main Oscillator Wait Time Setting in the MOSCWTCR register. This register setting is available to be changed through the Main Oscillator Wait Time setting in the CGC HAL module properties. See the Wakeup Timing and Duration table in Electrical Characteristics for more information.

NOTE: When a project uses ThreadX and the low-power mode standby, deep standby, or standby with snooze enabled, the call to the lpm_v2_api_t::lowPowerCfg API function should occur immediately before the call to the lpm_v2_api_t::lowPowerModeEnter API function. This is necessary since ThreadX also uses low-power modes in its idle loop and tx_thread_sleep function; ThreadX expects the MCU device to be configured for the low-power mode sleep. When a project uses ThreadX and the low-power mode standby or standby with snooze enabled, the low-power mode should be reverted to sleep after the MCU device wakes from standby after returning from the

lpm_v2_api_t::lowPowerModeEnter function. This is necessary since ThreadX also uses low-power modes in its idle loop and tx_thread_sleep function; ThreadX expects the MCU device to be configured for the low-power mode sleep. The API function lpm_v2_api_t::lowPowerCfg needs to be called again before lpm_v2_api_t::lowPowerModeEnter to re-configure the low-power mode to sleep, if the tx_thread_sleep function is used in the project, or if there may not always be a thread ready to run.

Detailed information about the expected power consumption of the MCU device in operating states and in Low Power Modes V2 can be found in the Operating and Standby Current section within the Electrical Characteristics section of the MCU Synergy Hardware User's Manual.

- Flash stop (code flash disable) is not supported. See the section *Flash Operation Control Register (FLSTOP)* of the S1/S3 Synergy Hardware User's Manual.
- Reduced SRAM retention area in software standby mode is not supported. See the section *Power Save Memory Control Register (PSMCR)* of the S3 Synergy Hardware User's Manual.
- The MCU may not enter or stay in Software Standby and Deep Software Standby modes with the debugger attached. Instead, the MCU may be woken from Software Standby and Deep Software Standby modes by the debugger. To properly test and verify Software Standby and Deep Software Standby modes, the debugger must not be attached.
- If the main oscillator or PLL with main oscillator source is used for the system clock, the wake time from standby mode can be affected by the Main Oscillator Wait Time Setting in the MOSCWTCR register. This register setting is available to be changed through the Main Oscillator Wait Time setting in the CGC HAL module properties. See the Wakeup Timing and Duration table in Electrical Characteristics for more information.

Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.33.4 Including the Low Power Modes V2 HAL Module in an Application

This section describes how to include the Low Power Modes V2 HAL module in an application using the SSP configurator.

NOTE: It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few sections of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add a Low Power Modes V2 Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Low Power Modes Driver is g_lpmv2_<mode>0. This name can be changed in the associated **Properties** window.)

Low Power Modes V2 Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_lpmv2_deep_standby0 S7G2 Low Power Mode Sleep on r_lpmv2	Threads	**New Stack> Driver> Power> Low Power Mode Deep Standby on r_lpmv2**
g_lpmv2_sleep0 S7G2 Low Power Mode Sleep on r_lpmv2	Threads	**New Stack> Driver> Power> Low Power Mode Sleep on r_lpmv2**

Resource	ISDE Tab	Stacks Selection Sequence
g_lpmv2_standby0 S7G2 Low Power Mode Sleep on r_lpmv2	Threads	**New Stack> Driver> Power> Low Power Mode Standby on r_lpmv2**

NOTE: The selection sequences lists modes available for the S7G2 Group. Other MCUs will have a different selection sequence available in the ISDE. If you add the LPM V2 HAL module to more than one thread, or to a thread and to HAL/Common, give each instance of the LPM V2 Driver a unique name by highlighting the instance of the driver in the Modules window and changing the entry for name in the **Properties** view. The default settings for the LPM V2 HAL module configuration structure can also be changed using this view.

When the Low Power Modes V2 HAL module on r_lpmv2 is added to the thread stack as shown in the following figure. (The figure shows the three available low power modes for the S7G2 MCU. These modules can be added individually or in groups. This figure shows all three for completeness; the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information will be box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks.

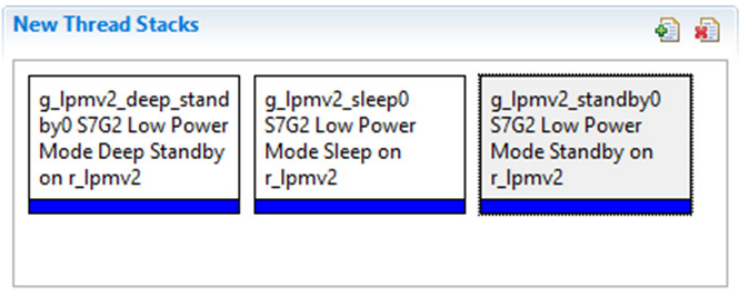


Figure 378: Low Power Modes V2 Module Stack

4.2.33.5 Configuring the Low Power Modes V2 HAL Module

The Low Power Modes V2 Driver module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the **Properties** window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the **Properties** window of the associated module. Simply select the indicated module and then view the **Properties** window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the **Properties** window in the ISDE includes an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the following configuration table settings. This helps orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the Low Power Mode Deep Standby Module on r_lpmv2

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enables or disables the parameter checking
Name	g_lpmv2_deep_standby0	Module name
Output port state in standby and deep standby, applies to address output, data output, and other bus control output pins	High impedance state, No change (Default: No change)	Output port state setting in Standby and Deep Standby
Maintain or reset the IO port states on exit from deep standby mode	Maintain the IO port states, Reset the IO port states (Default: Maintain the IO port states)	Output port state setting exit
Internal power supply control in deep standby mode	Maintain the internal power supply, Cut the power supply to standby RAM, low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit, Cut the power supply to LVDn, standby RAM, low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit (Default: Maintain the internal power supply)	Internal power supply control in deep standby mode setting
Deep Standby Cancel Sources/Edges: Select Fields Below		
IRQ0	Enabled, Disabled (Default: Disabled)	IRQ0 selection
IRQ0 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	IRQ0 Edge selection

ISDE Property	Value	Description
IRQ1	Enabled, Disabled (Default: Disabled)	IRQ1 selection
IRQ1 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	IRQ1 Edge selection
IRQ2	Enabled, Disabled (Default: Disabled)	IRQ2 selection
IRQ2 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	IRQ2 Edge selection
IRQ3	Enabled, Disabled (Default: Disabled)	IRQ3 selection
IRQ3 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	IRQ3 Edge selection
IRQ4	Enabled, Disabled (Default: Disabled)	IRQ4 selection
IRQ4 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	IRQ4 Edge selection
IRQ5	Enabled, Disabled (Default: Disabled)	IRQ5 selection
IRQ5 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	IRQ5 Edge selection

ISDE Property	Value	Description
IRQ6	Enabled, Disabled (Default: Disabled)	IRQ6 selection
IRQ6 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	IRQ6 Edge selection
IRQ7	Enabled, Disabled (Default: Disabled)	IRQ7 selection
IRQ7 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	IRQ7 Edge selection
IRQ8	Enabled, Disabled (Default: Disabled)	IRQ8 selection
IRQ8 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	IRQ8 Edge selection
IRQ9	Enabled, Disabled (Default: Disabled)	IRQ9 selection
IRQ9 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	IRQ9 Edge selection
IRQ10	Enabled, Disabled (Default: Disabled)	IRQ10 selection
IRQ10 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	IRQ10 Edge selection

ISDE Property	Value	Description
IRQ11	Enabled, Disabled (Default: Disabled)	IRQ11 selection
IRQ11 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	IRQ11 Edge selection
IRQ12	Enabled, Disabled (Default: Disabled)	IRQ12 selection
IRQ12 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	IRQ12 Edge selection
IRQ13	Enabled, Disabled (Default: Disabled)	IRQ13 selection
IRQ13 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	IRQ13 Edge selection
IRQ14	Enabled, Disabled (Default: Disabled)	IRQ14 selection
IRQ14 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	IRQ14 Edge selection
IRQ15	Enabled, Disabled (Default: Disabled)	IRQ15 selection
IRQ15 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	IRQ15 Edge selection

ISDE Property	Value	Description
LVD1	Enabled, Disabled (Default: Disabled)	LVD1 selection
LVD1 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	LVD1 Edge selection
LVD2	Enabled, Disabled (Default: Disabled)	LVD2 selection
LVD2 Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	LVD2 Edge selection
RTC Interval	Enabled, Disabled (Default: Disabled)	RTC Interval selection
RTC Alarm	Enabled, Disabled (Default: Disabled)	RTC Alarm selection
NMI	Enabled, Disabled (Default: Disabled)	NMI selection
NMI Edge	Disabled, Rising Edge, Falling Edge (Default: Disabled)	NMI Edge selection
USBFS	Enabled, Disabled (Default: Disabled)	USBFS selection
UBSHS	Enabled, Disabled (Default: Disabled)	UBSHS selection

ISDE Property	Value	Description
AGT1	Enabled, Disabled (Default: Disabled)	AGT1 selection

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

Configuration Settings for Low Power Mode Sleep Driver on r_lpmv2

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enables or disables the parameter checking
Name	g_lpmv2_sleep0	Module name

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for Low Power Mode Standby Driver on r_lpmv2

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enables or disables the parameter checking
Name	g_lpmv2_standby0	Module name
Choose the low power mode	Standby, Standby with Snooze enabled (Default: Standby)	Low power mode selection
Output port state in standby and deep standby, applies to address output, data output, and other bus control pins	No Change, High Impedance state (Default: No Change)	Output port state selection

ISDE Property	Value	Description
Select Standby Exit Sources: Select Fields below		
IRQ0:15	Enabled, Disabled (Default: Disabled)	IRQ0:16 selection
IWDT	Enabled, Disabled (Default: Disabled)	IWDT selection
Key Interrupt	Enabled, Disabled (Default: Disabled)	Key Interrupt selection
LVD1 Interrupt	Enabled, Disabled (Default: Disabled)	LVD1 selection
LVD2 Interrupt	Enabled, Disabled (Default: Disabled)	LVD2 selection
Analog Comparator High-speed 0 Interrupt	Enabled, Disabled (Default: Disabled)	Analog Comparator selection
RTC Period	Enabled, Disabled (Default: Disabled)	RTC Period selection
RTC Alarm	Enabled, Disabled (Default: Disabled)	RTC Alarm selection
USBFS	Enabled, Disabled (Default: Disabled)	USBFS selection

ISDE Property	Value	Description
UBSHS	Enabled, Disabled (Default: Disabled)	UBSHS selection
AGT1 Underflow	Enabled, Disabled (Default: Disabled)	AGT1 Underflow selection
AGT1 Compare Match A	Enabled, Disabled (Default: Disabled)	AGT1 CMA selection
AGT1 Compare Match B	Enabled, Disabled (Default: Disabled)	AGT1 CMB selection
I2C 0	Enabled, Disabled (Default: Disabled)	I2C 0 selection
Snooze Mode Settings		
Snooze Entry Source	RXD0 falling edge, IRQ0:15, KINT (Key Interrupt), ACMPHS0 (High-speed Analog Comparator 0), RTC Alarm, RTC Period, AGT1 Underflow, AGT1 Compare Match A, AGT1 Compare Match B (Default: RXD0 falling edge)	Source of Snooze mode entry
Snooze Exit Sources: Select fields below		
AGT1 Underflow	Enabled, Disabled (Default: Disabled)	AGT1 Underflow selection
DTC Transfer Completion	Enabled, Disabled (Default: Disabled)	DTC Transfer Completion selection

ISDE Property	Value	Description
DTC Transfer Completion Negated signal	Enabled, Disabled (Default: Disabled)	DTC Transfer Completion Negated signal selection
ADC0 Compare Match	Enabled, Disabled (Default: Disabled)	ADC0 Compare Match selection
ADC0 Compare Mismatch	Enabled, Disabled (Default: Disabled)	ADC0 Compare Mismatch selection
ADC1 Compare Match	Enabled, Disabled (Default: Disabled)	ADC1 Compare Match selection
ADC1 Compare Mismatch	Enabled, Disabled (Default: Disabled)	ADC1 Compare Mismatch selection
SCI0 Address Match	Enabled, Disabled (Default: Disabled)	SCI0 Address Match selection
DTC State Selection		
DTC state in Snooze Mode	Enabled, Disabled (Default: Disabled)	DTC state in Snooze Mode selection

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults can be desirable. For example, it might be useful to select different states for entering or exiting low-power states.

Low Power Modes V2 HAL Module Clock Configuration

The Low Power Modes V2 peripheral module does not have any selectable clock sources.

Low Power Modes V2 HAL Module Pin Configuration

The Low Power Modes V2 peripheral module needs no pin assignments. Pin function selections are done in the properties configuration window.

4.2.33.6 Using the Low Power Modes V2 HAL Module in an Application

The typical steps in using the Low Power Modes V2 HAL module in an application are:

1) Initialize the Low Power Modes V2 HAL module using the lpm_v2_api_t::init API
2) Configure a low-power mode with the lpm_v2_api_t::lowPowerCfg API
3) Enter a low-power mode with the lpm_v2_api_t::lowPowerModeEnter API

These common steps are illustrated in a typical operational flow diagram in the figure below:

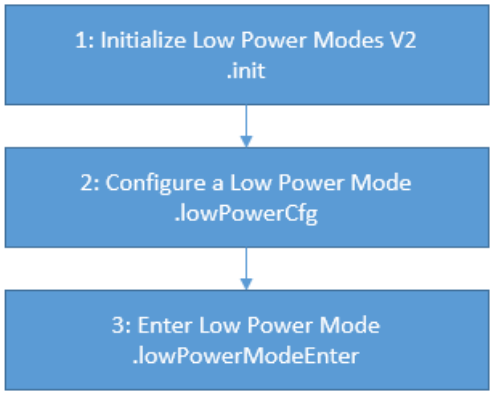


Figure 379: Flow Diagram of a Typical Low Power Modes V2 Module Application

4.2.34 Low Voltage Detection Driver

The Low Voltage Detection (LVD) HAL module is a high-level API for voltage-detection applications and is implemented on r_lvd. The LVD HAL module uses the LVD peripheral on the Synergy MCU. A user-defined callback can be created to notify the CPU when a voltage-detection event is triggered. The VCC is the source for all voltage-detection functions.

4.2.34.1 LVD HAL Module Features

The LVD HAL module supports the following functions:

- VCC as the voltage-detection input
- One build-time configurable low-voltage detector (via OFS1 register)
- Two run-time configurable low-voltage detectors
- Two result flags; one for a threshold check and one for the current state
- Support for both interrupt or polling-event checking

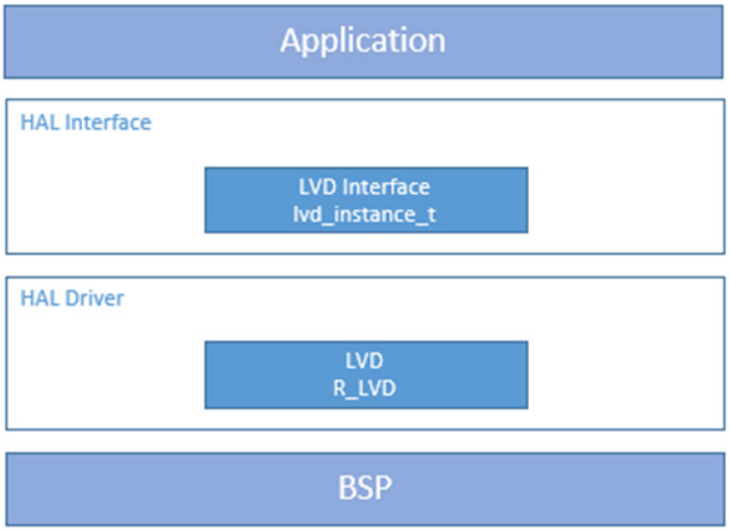


Figure 380: LVD HAL Module Block Diagram

4.2.34.2 LVD HAL Module APIs Overview

The LVD HAL module defines APIs for opening, closing, statusGet, and statusClear. The following table includes a complete list of the available APIs, an example API call, and a short description of each API. A table of status return values follows the API summary table.

LVD HAL Module API Summary

Function Name	Example API Call and Description
open	g_lvd.p_api->open(g_lvd.p_ctrl, g_lvd.p_cfg; Initializes a low voltage detection driver according to the passed in configuration structure. Enables an LVD peripheral based on configuration structure.
statusGet	g_lvd.p_api->statusGet(g_lvd.p_ctrl, &monitor_status); Get the current state of the monitor, (threshold crossing detected, voltage currently within range) Can be used to poll the state of the LVD monitor at any time. Must be used if the peripheral was initialized with the lvd_response_t set to LVD_RESPONSE_NONE.

Function Name	Example API Call and Description
statusClear	g_lvd.p_api->statusClear(g_lvd.p_ctrl); Clears the latched status of the monitor. Must be used if the peripheral was initialized with lvd_response_t set to LVD_RESPONSE_NONE.
close	g_lvd.p_api->close(g_lvd.p_ctrl, g_lvd.p_cfg); Disables the LVD peripheral. Closes the driver instance.
versionGet	g_lvd.p_api->versionGet(&version); Retrieve the API version with the version pointer.

NOTE: For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the *SSP User's Manual* API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_IN_USE	Driver already open or unable to acquire hardware lock
SSP_ERR_NOT_OPEN	Unit is not open
SSP_ERR_ASSERTION	Invalid configuration value
SSP_ERR_INVALID_MODE	If the attempted mode is invalid for this configuration

NOTE: Lower-level drivers may return common error codes. Refer to the *SSP User's Manual* API References for the associated module for a definition of all relevant status return values.

4.2.34.3 LVD HAL Module Operational Overview

The LVD HAL module supports the configuration and operation of LVD monitors in the Synergy MCUs. The LVD HAL module provide configuration structures with all the information needed to fully configure a single LVD monitor. There is one instance of the LVD HAL module per LVD monitor instance, with the exception of the LVD0 monitor. The LVD0 monitor is not configurable at runtime; it must be configured at compile time via the OFS1 register.

Both the LVD1 and LVD2 monitors are configurable at runtime by this module. The open function allows the user to configure and enable an LVD monitor with a single function call; the close function disables the LVD monitor. The statusGet function returns the current status of the LVD monitor. The statusGet function should be used if the module is in polling mode, i.e. without the LVD monitor interrupt enabled. The monitor status consists of two flags. The first flag is a latched flag called crossing_detected which indicates whether or not the voltage being monitored has crossed the voltage threshold. In polling mode, this flag must be cleared via a call to statusClear. The flag does not need to be cleared explicitly if the LVD interrupt is in use; it clears in the LVD interrupt by the driver code after the user-callback function is called. The second flag, current_state, is the instantaneous status of the monitored voltage with respect to the voltage threshold; this flag is not latched and its status changes with the monitored voltage changes.

The LVD HAL module can be configured to enable one or several of the LVD peripheral interrupts. If using an interrupt, the user should provide a callback function for that monitor. Separate callback routines should be provided for each LVD monitor.

The LVD HAL module requires functionality provided by the BSP; this driver makes use of the BSP hardware locks for locking registers, as well as enabling and clearing interrupts.

LVD HAL Module Important Operational Notes and Limitations

- Once the appropriate values are chosen for these LVD settings, you should add the code to call the LVD HAL module open API function for your project. This function should be called once early in the application.
- The module can be closed and opened whenever the configuration of an LVD monitor needs to be changed. Calling the LVD open API function configures and enables the LVD hardware peripheral the specified LVD monitor.
- The close function disables the LVD monitor and closes the driver.
- Using this module to configure the LVD peripheral to generate an interrupt requires the corresponding interrupt to be enabled in the module **Properties** tab.
- When using the LVD interrupts a callback function is not required, but is recommended.
- A unique callback function for each LVD interrupt is not required, but is recommended.
- Clock system initialization, configuration, and runtime modification are handled outside this module. This driver makes changes to the digital filter sample clock based on the user's choice of a sample clock divisor. The digital filter sample clock is derived from the low-speed on-chip oscillator (LOCO) system clock.
- Not all voltage thresholds are available on all MCU's.
- Digital filtering of the VCC input to the LVD monitor is not available on all Synergy MCU devices.
- The LVD driver requires functionality provided by the BSP; it makes use of hardware locks provided by the BSP for register locks, as well as enabling and clearing interrupts.
- Configuring and enabling a LVD monitor requires specific timing constraints and register write ordering. Due to the constraints, the entire process of configuring and enabling a voltage monitor is most effective when performed by a single function. The open API function performs configuration and enables the monitor and properly enforces the timing and register write ordering constraints.
- All the Synergy MCU Series include **Option-Setting Memory** that can be used to set the operating state of peripherals after a reset. The (option function select) OFS can be used to set the state of the IWDT, WDT, LVD, and CGC high-speed on-chip oscillator (HOCO).

See the latest SSP Release Notes for any additional operational limitations for this module.

NOTE: The LVD0 can only be configured via the OFS registers. The LVD0 does not support Register Start mode.

OFS Settings for Low Voltage Detection

Control	Description	
LVD0 detection level	S7 & S5 Series 2.94 V 2.87 V 2.80 V	S3 & S1 Series 3.84 V 2.82 V 2.51 V 1.90 1.70 V
LVD0 detection start	Automatically starts the LVD0 after a Reset, if enabled	

Set the OFS register values in the Synergy Configuration editor via the BSP tab **Properties** dialog box.

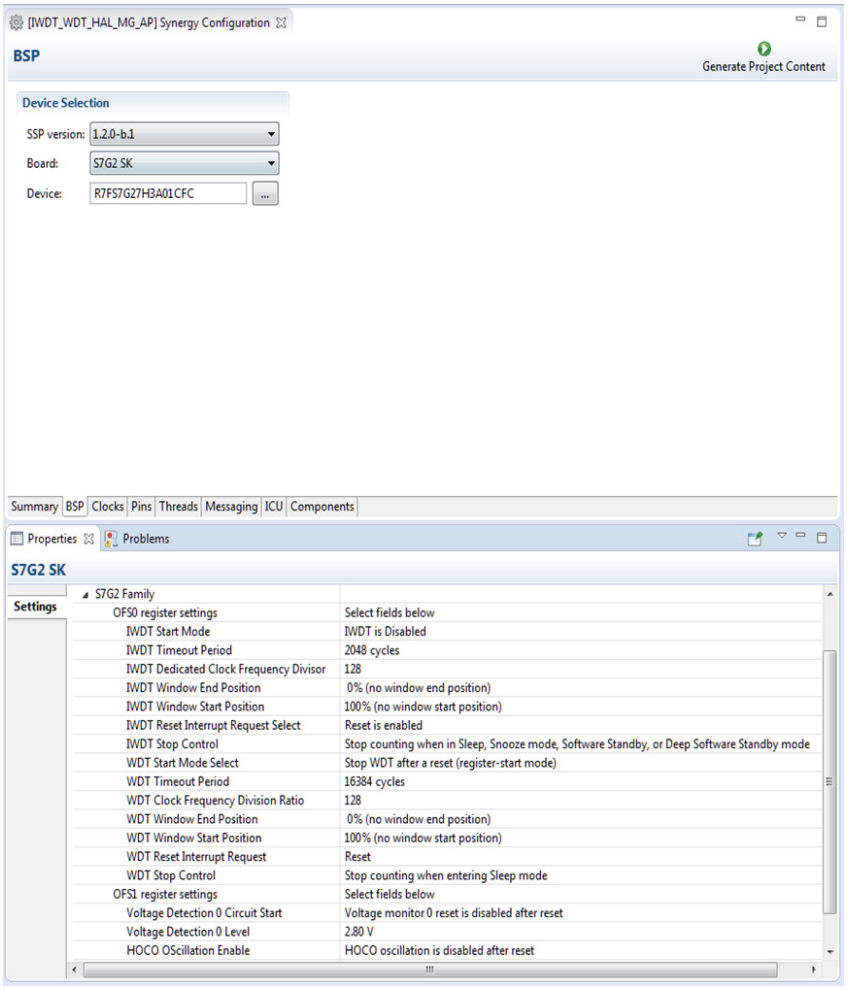


Figure 381: OFS Register Settings

4.2.34.4 Including the LVD HAL Module in an Application

This section describes how to include the LVD HAL Module in an application using the SSP configurator.

NOTE: It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the *SSP User's Manual* to learn how to manage each of these important steps in creating SSP-based applications.

To add the LVD Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the LVD HAL module is g_lvd. This name can be changed in the associated **Properties** window.)

Table 4 LVD Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_lvd Low Voltage Detection Driver on r_lvd	Threads	New Stack> Driver> Power> Low Voltage Detection Driver on r_lvd

When the LVD Driver on r_lvd is added to the thread stack, as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information are box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

Figure 382: LVD HAL Module Stack

4.2.34.5 Configuring the LVD HAL Module

The LVD HAL module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the Properties window in the ISDE includes an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the following configuration table settings. This helps to orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Table 5 Configuration Settings for the LVD HAL Module on r_lvd

ISDE Property	Value	Description
Parameter Checking	Enabled, Disabled, BSP (Default: BSP)	Selects if code for parameter checking is to be included in the build
Name	Default: g_lvd	LVD Driver module name
Monitor Number	1	Monitor number selection
Digital Filter, enable by selecting a valid sample clock rate (S7G2 only).	Enabled with clock LOCO/2/4/8/16, Disabled (Default: Disabled)	Digital filter selection
Voltage Threshold	2.85V (Vdet1_13) (S7G2 only).	Voltage threshold level selection
Detection Response, either reset, interrupt, non-maskable interrupt, or no response (polling mode).	Maskable interrupt is triggered when the voltage crosses the detection threshold. The non-maskable interrupt is triggered when voltage crosses the detection threshold. The MCU resets when the voltage falls below the detection threshold. No response: the driver is in polled mode (using statusGet and statusClear functions.) (Default: Maskable interrupt)	Detection response selection
Voltage Slope	Threshold crossing detected with decreasing voltage. Threshold crossing detected with increasing voltage. Threshold crossing detected with increasing or decreasing voltage. (Default: Threshold crossing detected with decreasing voltage.)	Direction of voltage slope for threshold detection
Negation of Monitor Signal	Negation of reset signal is based on delay from the reset. Negation of reset signal is based on delay from voltage returning to normal range. (Default: Negation of reset signal is based on delay from reset.)	Negation option selection
Monitor Interrupt Callback	NULL	Interrupt callback function name
LVD Monitor Interrupt Priority	Priority 0 (highest), 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 (lowest, not valid if using Thread X), Disabled (Default: Disabled)	Interrupt priority setting

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings (other than the defaults) for a module can be desirable. For example, it might be useful to select a voltage level appropriate for the target system.

LVD HAL Module Clock Configuration

Clock system clock initialization, configuration, and runtime modification are handled outside this module. This module only makes changes to the digital filter sample clock based on the user's choice of sample clock divisor. The digital filter sample clock is derived from the LOCO system clock.

LVD HAL Module Pin Configuration

The LVD HAL module measures the voltage on the VCC pin only and doesn't need to be configured.

4.2.34.6 Using the LVD HAL Module in an Application

The typical steps in using the LVD HAL module in an application are:

1) Initialize the LVD HAL module using the open API.
2) If using software polling, monitor the LVD status flags with the statusGet API and process accordingly. If using the interrupt mode, process accordingly within the callback function which will return both the monitor number as well as the status.
3) Process as needed
4) Close the LVD Instance with the close API

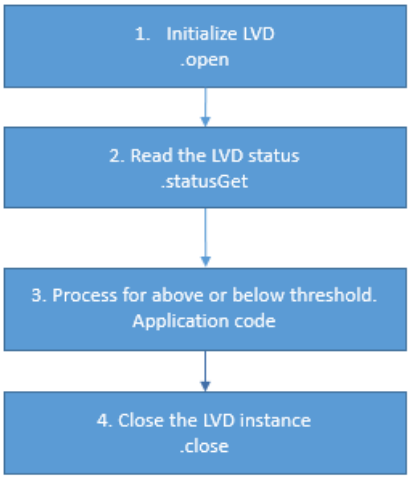


Figure 383: Flow Diagram of a Typical LVD HAL Module in software polling mode

4.2.35 OPAMP Driver

The OPAMP HAL module implements the OPAMP API for signal amplification applications on r_opamp. It supports the OPAMP peripheral available on Synergy MCUs.

4.2.35.1 OPAMP HAL Module Features

- Low power or high-speed mode
- Start by software or AGT compare match
- Stop by software or ADC conversion end (stop by ADC conversion end only supported on op-amp channels configured to start by AGT compare match)
- Trimming available on some MCUs (see hardware manual)

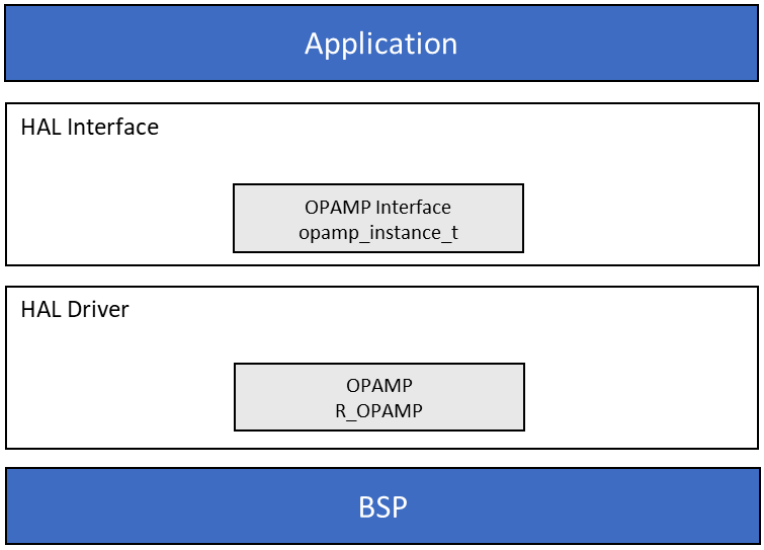


Figure 384: OPAMP HAL Module Block Diagram

4.2.35.2 OPAMP HAL Module APIs Overview

The OPAMP HAL module defines APIs to open, start, stop, read status, trim, and close the ADC module. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

OPAMP HAL Module API Summary

Function Name	Example API Call and Description
.open	g_opamp0.p_api->open(g_opamp0.p_ctrl, g_opamp0.p_cfg); Applies power to the OPAMP and initializes the hardware based on the user configuration.

Function Name	Example API Call and Description
.start	g_opamp0.p_api->start(g_opamp0.p_ctrl, channel_mask); If the OPAMP is configured for hardware triggers, enables hardware triggers. Otherwise, starts the op-amp.
.stop	g_opamp0.p_api->stop(g_opamp0.p_ctrl, channel_mask); Stops the op-amp. If the OPAMP is configured for hardware triggers, disables hardware triggers.
.trim	g_opamp0.p_api->trim(g_opamp0.p_ctrl, cmd, p_args); On MCUs that support trimming, the op-amp trim register is set to the factory default after open(). This function allows the application to trim the operational amplifier to a user setting, which overwrites the factory default factory trim values. See Operational Notes for important details.
.infoGet	g_opamp0.p_api->infoGet(g_opamp0.p_ctrl, p_info); Provides the minimum stabilization wait time in microseconds.
.statusGet	g_opamp0.p_api->statusGet(g_opamp0.p_ctrl, p_status); Provides the operating status for each op-amp in a bitmask. This bit is set when operation begins, before the stabilization wait time has elapsed.
.close	g_opamp0.p_api-> close(g_opamp0.p_ctrl); Stops the op-amps.
.versionGet	g_opamp0.p_api->versionGet(&version); Retrieve the module version using the version pointer

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_NOT_OPEN	Unit is not open
SSP_ERR_ASSERTION	An input pointer is NULL.
SSP_ERR_IN_USE	Peripheral is in use or hardware lock is taken.
SSP_ERR_INVALID_POINTER	The parameter p_data is NULL.
SSP_ERR_INVALID_STATE	The command is not valid for the current state of the trim function.
SSP_ERR_INVALID_MODE	Trim is not allowed in low power mode.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.2.35.3 OPAMP HAL Module Operational Overview

The OPAMP HAL module controls the OPAMP peripheral on a Synergy microcontroller. It directly controls the OPAMP hardware without using any RTOS elements and provides convenient APIs to simplify development.

OPAMP HAL Module Important Operational Notes and Limitations

Trimming the OPAMP

On MCUs that support trimming, the op-amp trim register is set to the factory default after open(). This function allows the application to trim the operational amplifier to a user setting, which overwrites the factory default factory trim values.

Trimming is not supported on all MCUs. See hardware manual for details. It is not supported if configured for low power mode (OPAMP_MODE_LOW_POWER).

This function is not reentrant. Only one side of one op-amp can be trimmed at a time. Complete the procedure for one side of one channel before calling trim() with command OPAMP_TRIM_CMD_START again.

The trim procedure works as follows:

- Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_START.
- Connect a fixed voltage to the Pch (+) input.

- Connect the Nch (-) input to the op-amp output to create a voltage follower.
- Ensure the op-amp is operating and stabilized.
- Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_START.
- Measure the fixed voltage connected to the Pch (+) input using the SAR ADC and save the value (referred to as A later in this procedure).
- Iterate over the following loop 5 times:
  - Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_NEXT_STEP.
  - Measure the op-amp output using the SAR ADC (referred to as B in the next step).
  - If A <= B, call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_CLEAR_BIT.
- Call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_START.
- Measure the fixed voltage connected to the Pch (+) input using the SAR ADC and save the value (referred to as A later in this procedure).
- Iterate over the following loop 5 times:
  - Call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_NEXT_STEP.
  - Measure the op-amp output using the SAR ADC (referred to as B in the next step).
  - If A <= B, call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_CLEAR_BIT.

The following Status Return values are associated with the trim API.

SSP_SUCCESS: Conversion result in p_data.

SSP_ERR_UNSUPPORTED: Trimming is not supported on this MCU.

SSP_ERR_INVALID_STATE: The command is not valid in the current state of the trim state machine.

SSP_ERR_INVALID_ARGUMENT:The requested channel is not operating or the trim procedure is not in progress for this channel/input combination.

SSP_ERR_INVALID_MODE: Trim is not allowed in low power mode.

SSP_ERR_ASSERTION: An input pointer was NULL.

SSP_ERR_NOT_OPEN: Instance control block is not open. Trimming is not supported on all MCUs.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.35.4 Including the OPAMP HAL Module in an Application

This section describes how to include the OPAMP HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the OPAMP HAL module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the OPAMP HAL module is g_opamp0. This name can be changed in the associated Properties window.)

OPAMP HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_opamp0 OPAMP Driver on r_opamp	Threads	New Stack> Driver> Analog> OPAMP Driver on r_opamp

When the OPAMP HAL module on r_opamp is added to the thread stack as shown in the following figure, the configurator automatically adds any lower-level drivers that are required. Any drivers that need additional configuration information will be box-text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

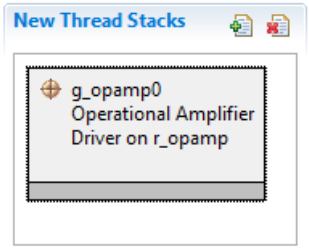


Figure 385: OPAMP HAL Module Stack

4.2.35.5 Configuring the OPAMP HAL Module

The OPAMP HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the OPAMP HAL Module on r_opamp

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_opamp	Module name
AGT Start Trigger Configuration (N/A unless AGT Start Trigger is Selected for the Channel)	AGT1 Compare Match Starts OPAMPs 0 and 2 if configured for AGT Start/AGT0 Compare Match Starts OMPAMPs 1 and 3 if configured for AGT Start AGT1 Compare Match Starts OPAMPs 0 and 1 if configured for AGT Start/AGT0 Compare Match Starts OPAMPs 2 and 3 if configured for AGT Start AGT1 Compare Match Starts all OPAMPs configured for AGT Start Default: AGT1 Compare Match Starts all OPAMPs configured for AGT Start	Configure which AGT channel event triggers op-amp channel. The AGT compare match event only starts the op-amp channel if the AGT Start trigger is selected in the Trigger configuration for the channel.
Power Mode	Low Power, Middle Speed, High Speed Default: High Speed	Configure the op-amp based on power or speed requirements. This setting affects the minimum required stabilization and trim time. Middle speed is not available for all MCUs.
Trigger Channel 0	Software Start Software Stop, AGT Start Software Stop, AGT Start ADC Stop Default: Software Start Software Stop	Select the event triggers to start or stop op-amp channel 0. If the event trigger is selected for start, the start() API enables the event trigger for this channel. If the event trigger is selected for stop, the stop() API disables the event trigger for this channel.

ISDE Property	Value	Description
Trigger Channel 1	Software Start Software Stop, AGT Start Software Stop, AGT Start ADC Stop Default: Software Start Software Stop	Select the event triggers to start or stop op-amp channel 1. If the event trigger is selected for start, the start() API enables the event trigger for this channel. If the event trigger is selected for stop, the stop() API disables the event trigger for this channel.
Trigger Channel 2	Software Start Software Stop, AGT Start Software Stop, AGT Start ADC Stop Default: Software Start Software Stop	Select the event triggers to start or stop op-amp channel 2. If the event trigger is selected for start, the start() API enables the event trigger for this channel. If the event trigger is selected for stop, the stop() API disables the event trigger for this channel.
Trigger Channel 3	Software Start Software Stop, AGT Start Software Stop, AGT Start ADC Stop Default: Software Start Software Stop	Select the event triggers to start or stop op-amp channel 3. If the event trigger is selected for start, the start() API enables the event trigger for this channel. If the event trigger is selected for stop, the stop() API disables the event trigger for this channel.

NOTE: The example values and defaults are for a project using the Synergy S1JA. Other MCUs may have different default values and available configuration settings.

OPAMP HAL Module Clock Configuration

The OPAMP HAL module does not require a clock for module operation.

OPAMP HAL Module Pin Configuration

To use the OPAMP HAL module, the port pins for the channels receiving the analog input must be set as input pins in the pin configurator in the ISDE. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection Sequence for the OPAMP HAL Module

Resource	ISDE Tab	Pin selection Sequence
OPAMP	Pins	Select Peripherals > Analog: OPAMP0/1/2

4.2.35.6 Using the OPAMP HAL Module in an Application

Once the module has been configured and the files generated, the OPAMP is ready to be used in an application.

The typical steps in using the OPAMP HAL module in an application are:

1) Initialize the OPAMP using the open API.
2) Before starting any op-amp, consult the hardware manual to determine if the MCU used has internal connections switches in the OPAMP peripheral. If the OPAMP peripheral does have internal connection switches, configure the internal connections by setting the AMPnMS, AMPnPS, and AMP0OS registers directly.
3) Start the OPAMP channel(s) using the desired trigger with the start API.
    a) If AGT compare match start is used, this call enables the OPAMP to be triggered by the AGT compare match. If a software trigger is used, then this call starts OPAMP channel(s).
4) [Optional] Stop the OPAMP channel(s) by calling the stop API.
    a) This stops the OPAMP regardless of if ADC conversion end triggers are enabled to stop the OPAMP.
5) [Optional] Use the close API to power down the peripheral.

These common steps are illustrated in a typical operational flow in the following figure:

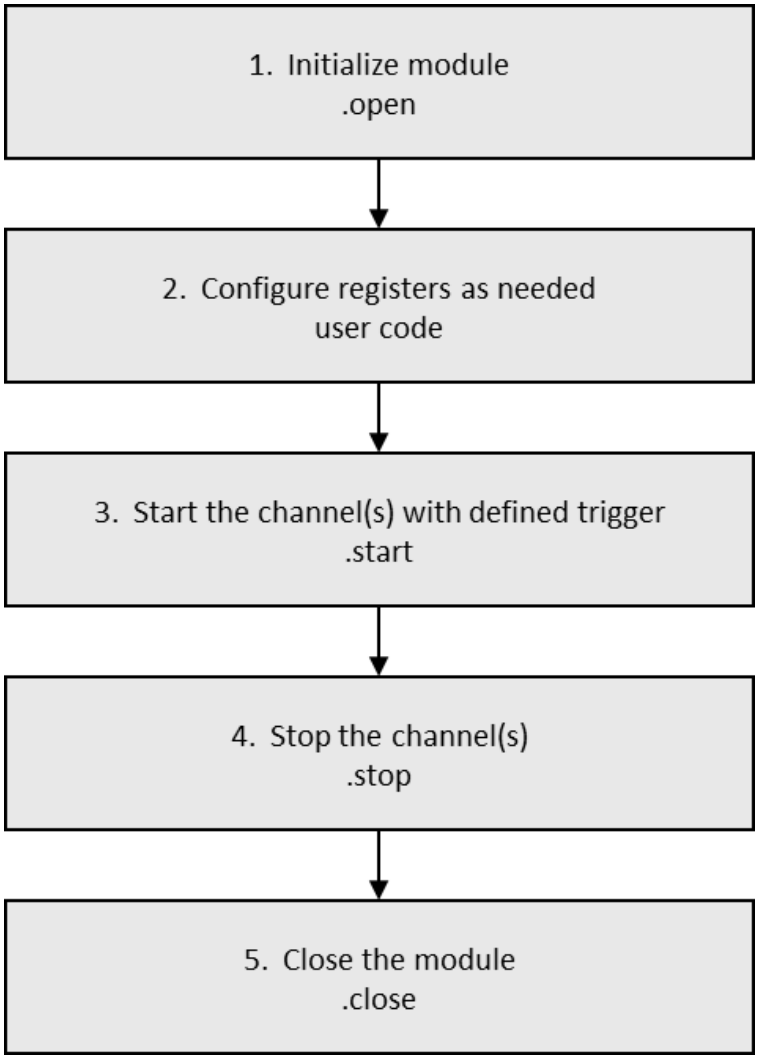


Figure 386: Flow Diagram of a Typical OPAMP HAL Module Application

4.2.36 PDC Driver

The Parallel Data Capture Unit (PDC) HAL module provides high-level APIs to capture images from a camera application and is implemented on r_pdc. The PDC HAL module uses the PDC peripherals on the Synergy MCU. A user-defined callback can be created to inform the CPU when a capture has been completed.

4.2.36.1 PDC HAL Module Features

This PDC HAL module controls the PDC on a Synergy microcontroller and has the following key features:

- Supports capture from a connected and configured camera.
- Supports a callback that informs the CPU when a capture is complete

- Provides a pointer to the capture buffer
- Provides an indication of the event triggering the callback

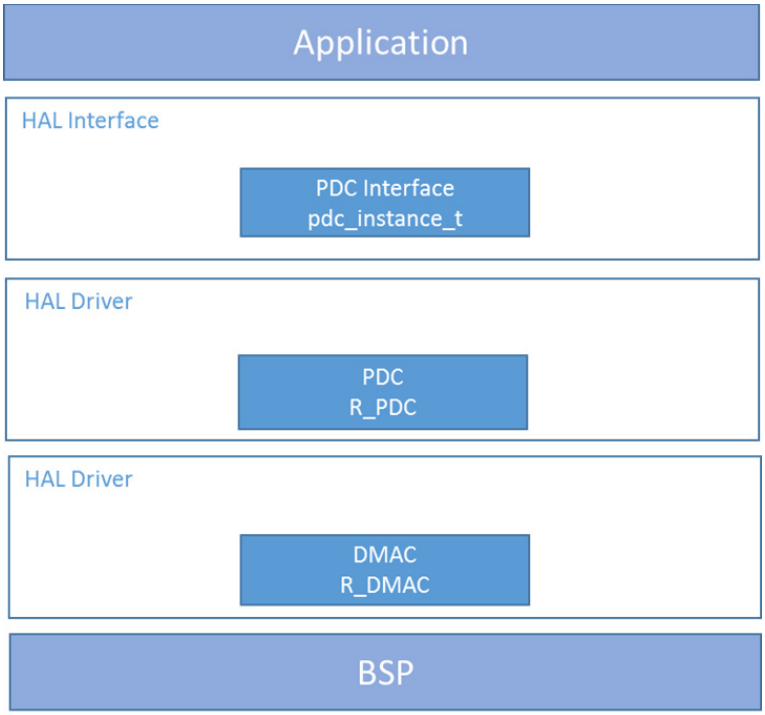


Figure 387: PDC HAL Module Block Diagram

4.2.36.2 PDC HAL Module APIs Overview

The PDC HAL module defines APIs for opening, closing, and starting data capture. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API Summary table.

PDC HAL Module API Summary

Function Name	Example API Call and Description
open	g_pdc.p_api->open(g_pdc.p_ctrl, g_pdc.p_cfg) Initial configuration.
close	g_pdc.p_api->close(g_pdc.p_ctrl)) Closes the driver and allows reconfiguration. May reduce power consumption.

Function Name	Example API Call and Description
captureStart	g_pdc.p_api->captureStart(g_pdc.p_ctrl, &buffer) Start a capture.
stateGet	g_pdc.p_api->stateGet(g_pdc.p_ctrl, &state_data) Get the state of VSYNC and HSYNC pins.
versionGet	g_pdc.p_api->versionGet(&version) Return the API version using the version pointer.

NOTE: Review the SSP User's Manual API References for the associated module to learn more about operations and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_ASSERTION	The parameter p_ctrl or p_sample is NULL.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_ALREADY_OPEN	Unit is already open.
SSP_ERR_HW_LOCKED	Unable to reserve BSP hardware lock.
SSP_ERR_TIMEOUT	Reset Operation timed out.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.36.3 PDC HAL Module Operational Overview

The capture operation requires a configured external camera connected to the Synergy microcontroller. Before performing a capture, it is important the camera is configured and generating a PIXCLK-clock input into the microcontroller. In some instances, a camera requires a running-clock input before it can be configured.

Use the call open, which configures and starts the PCKO-clock output from the PDC into the camera, before initializing the camera. Once the camera is configured, captureStart can be called to capture an image. Configuration of a camera module may require the use of an I^2C or SPI interface.

PDC HAL Module Important Operational Notes and Limitations

In most instances, the data rate from a camera and from the PDC peripheral is too fast to be serviced by the CPU in an interrupt service routine (ISR). Therefore, this module requires an implementation of the transfer driver on the DMAC to perform a high-speed transfer from the PDC peripheral and memory.

You must enable both the PDC frame-end and PDC error interrupts to generate interrupts in the following situations:

- An interrupt when an image is captured (frame end)
- An interrupt when an error occurs

Data Buffer Setting

If p_buffer is set to anything other than NULL, one or more data buffers are created to store image data. The size of each buffer is calculated using the following formula:

Buffer size (bytes) = x_capture_pixels x y_capture_pixels x bytes_per_pixel

For large resolution cameras, the captured image could result in a large amount of data. It may be necessary to locate the buffer(s) in external memory (such as, SDRAM). Consideration should be given to bus bandwidth when using external memory.

For example, when using a high frame-rate camera to do an image capture via the PDC into SDRAM, and using SDRAM to hold the display buffer for an LCD display with a high refresh rate, may cause a data bottleneck from the PDC to memory that results in an overrun-error condition.

NOTE: If p_buffer is set to NULL, no memory is allocated to store the captured image data. The user's responsibility is to ensure that there is suitable memory of sufficient size available to the PDC. The PDC could capture directly into the display buffer of a connected LCD panel.

For PDC HAL module operational limitations, refer to the latest SSP Release Notes.

4.2.36.4 Including the PDC HAL Module in an Application

This section describes how to include the PDC HAL module in an application using the SSP configurator.

NOTE: This process assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the PDC Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the PDC is r_pdc0. This name can be changed in the associated Properties window.)

CRC Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
r_pdc0 PDCDriver on r_pdc	Threads	New Stack> Driver> Graphics> PDC Driver on r_pdc

As shown in the following figure, when the PDC HAL module on r_pdc is added to the thread stack, the configurator automatically adds the needed lower-level drivers. Any drivers needing additional configuration information are highlighted in Red. Modules with a Gray band are individual standalone modules.

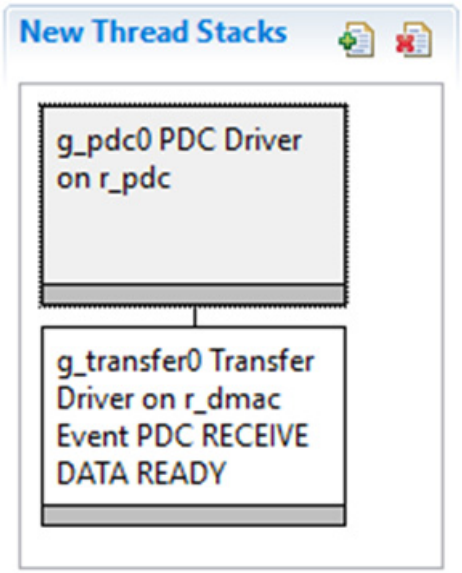


Figure 388: PDC HAL Module Stack

4.2.36.5 Configuring the PDC HAL Module

The PDC HAL module is configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, configured for lower-level modules to enable successful operation. Only properties that can be changed without causing conflicts are available for modification. Properties that cannot be changed are 'locked' and identified with a lock icon to indicate the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process, making it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the Properties window in the ISDE includes an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+.) This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking

over the following configuration table settings. This helps orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the PDC HAL Module on r_pdc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_pdc0	The name of the PDC module instance. Specify arbitrary C symbol.
Name of the data buffer to store image data	g_user_buffer	Specify the name of the data buffer to create or set to NULL, if it is to be created by the user external to the PDC driver.
Section where data buffer is allocated	sdram	Specify the RAM section for the image data buffer. Typically BSS (internal RAM) or SDRAM.
Number of bytes per pixel	2	Specify the number of bytes per pixel of the captured image data.
Number of image data buffers	1	Specify the number of buffers to create.
Clock Divider	CLK/2, CLK/4, CLK/6, CLK/8, CLK10, CLK12, CLK14, CLK16 Default: CLK/2	Specify the clock divider of the clock input to the PDC peripheral.
Endian of image data	Little, Big Default: Little	Specify the endian of the captured image data.
HYSNC signal polarity	High, Low Default: High	Specify the active polarity of the HSYNC signal.
VSYNC signal polarity	High, Low Default: High	Specify the active polarity of the VSYNC signal.

ISDE Property	Value	Description
Number of pixels to capture horizontally	640	Number of horizontal pixels to capture.
Number of lines to capture vertically	480	Number of vertical lines to capture.
Horizontal pixel to start capture from	0	Horizontal pixel to start capturing image data from. Allows an image smaller than the native resolution of a camera to be captured.
Line to start capture from	0	Vertical line to start capturing image data from. Allows an image smaller than the native resolution of a camera to be captured.
Callback	g_pdc_user_callback	A user callback function can be registered in open. If this callback function is provided, it is called from the interrupt service routine (ISR) each time a frame is captured and ready to be processed. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.
Frame End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	The driver needs a valid interrupt priority setting. It won't function if disabled.
PDC Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	The driver needs a valid interrupt priority setting. It won't function if disabled.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for a module can be desirable. For example, it might be useful to select a different clock source than the default. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

When configuring settings for the PDC HAL Module Low-Level Driver, typically, only a small number of settings must be modified from the default for lower-level drivers and these are indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

Configuration Settings for the DMAC HAL Module

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Parameter selection.
Name	g_transfer0	Driver name.
Mode	Block	Mode selection.
Transfer Size	4 Bytes	Transfer size selection.
Destination Address Mode	Incremented	Destination address mode selection.
Source Address Mode	Fixed	Source address mode selection.
Repeat Area (Unused in Normal Mode	Source	Repeat area selection.
Destination Pointer	NULL	Destination pointer selection.
Source Pointer	NULL	Source pointer selection.
Number of Transfers	8	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	1	Number of blocks selection.
Activation Source (Must enable IRQ)	Event PDC RECEIVE DATA READY	Activation source selection.
Auto Enable	FALSE	Auto enable selection.
Callback (Only valid with Software start)	NULL	Callback selection.

ISDE Property	Value	Description
Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Interrupt priority selection

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

PDC HAL Module Clock Configuration

The PDC uses PCLKB as its clock source. The only restriction when configuring this clock is that the PIXCLCK should be less than 0.6 x PCLKB so the PCLKB frequency should be set accordingly.

PDC HAL Module Pin Configuration

The PDC peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table depicts the method to select pins within the SSP configuration window. The subsequent table gives an example selection for PDC pins.

NOTE: The operation mode selection determines the peripheral signals available and the MCU pins required.

Pin Selection Sequence for the PDC HAL Module

Resource	ISDE Tab	Pin selection Sequence
PDC	Pins	Select **Peripherals > Graphics: PDC > PDC0**

NOTE: The selection sequence assumes PDC0 is the desired hardware target for the driver.

Pin Configuration Settings for the PDC HAL Module

Property	Value	Description
Pin Group Selection	Mixed, _A Only (Default: Mixed)	Pin group selection

Property	Value	Description
Operation Mode	Disabled, Custom, Enabled (Default: Disabled)	Select Enabled as the Operation Mode for PDC
HSYNC	None, P704 (Default: None)	HSYNC Pin
PCKO	None, P511 (Default: P511)	PCKO Pin
PIXCLK	None, P705 (Default: None)	PIXCLK Pin
VSYNC	None, P512 (Default: P512)	VSNC Pin
PIXD0:7	None, Pnnn (Default: None)	PIX Data0:7 Pins

NOTE: The example settings are for a project using the Synergy S7G2 and the DK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.36.6 Using the PDC HAL Module in an Application

The typical steps in using the PDC HAL module in an application are:

1) Initialize the PDC HAL module using the open API
2) Configure the camera as needed
3) Start image capture using the captureStart API
4) Callback is called when image is captured
5) Read state of HSYNC and VSTNC using stateGet API
6) Process data as needed

7) Close using the close API

These common steps are illustrated in a typical operational flow diagram in the figure below:

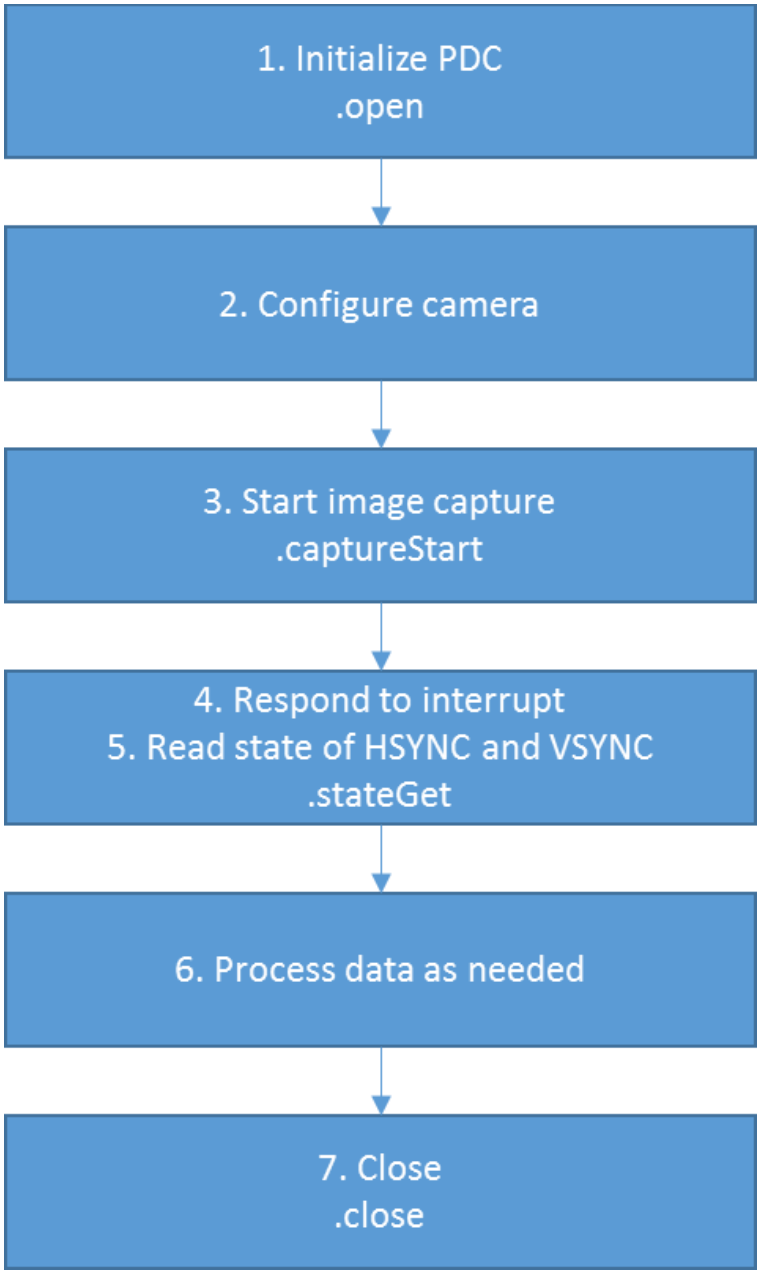


Figure 389: Flow Diagram of a Typical PDC HAL Module Application

4.2.37 QSPI Driver

The Quad SPI (QSPI) HAL module is a high-level API for erasing and programming the contents of a QSPI flash device connected to the microcontroller and is implemented on the r_qspi peripheral on the Synergy MCU. Unlike many other modules, there is no callback function for the QSPI.

4.2.37.1 QSPI HAL Module Features

The QSPI HAL Module is used to initialize the QSPI peripheral that allows erasing and programming the contents of a QSPI flash device connected to the microcontroller over the Quad SPI interface. Key features include:

- Access Quad SPI flash devices using Direct Communication Mode
- Read data from a QSPI flash device
- Program the page of a QSPI flash device
- Erase sectors of a QSPI flash device
- Select a bank to control access to a QSPI flash device

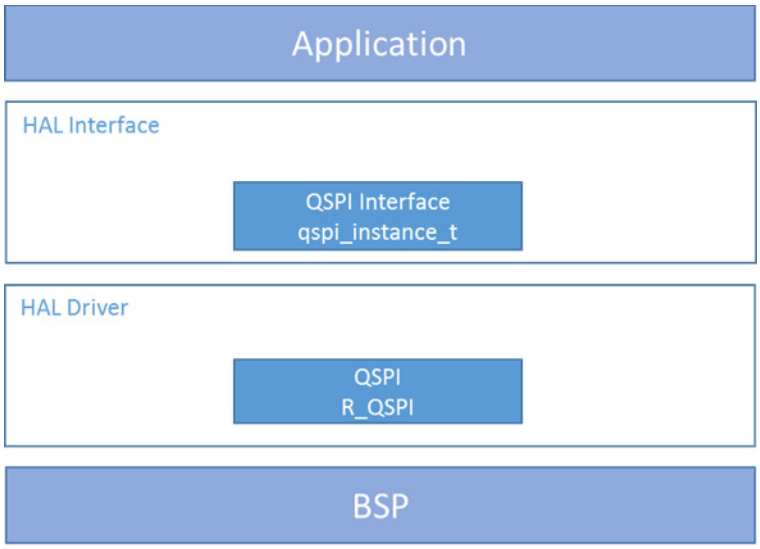


Figure 390: QSPI HAL Module Block Diagram

4.2.37.2 QSPI HAL Module APIs Overview

The QSPI interface defines APIs for opening, closing, reading, writing, erasing, and device bank selection using the QSPI HAL module. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

QSPI HAL Module API Summary

Function Name	Example API Call and Description
open	g_qspi0.p_api->open(g_qspi0.p_ctrl, g_qspi0.p_cfg); Open the QSPI HAL module.
close	g_qspi0.p_api->close(g_qspi0.p_ctrl); Close the QSPI HAL module.
read	g_qspi0.p_api->read(g_qspi0.p_ctrl, (uint8_t *) QSPI_DEVICE_ADDRESS, readBuffer, BUFFER_LENGTH); Read data from the flash.
pageProgram	g_qspi0.p_api->pageProgram(g_qspi0.p_ctrl, (uint8_t *) QSPI_DEVICE_ADDRESS, writeBuffer, BUFFER_LENGTH); Program a page of data to the flash.
sectorErase	g_qspi0.p_api->sectorErase(g_qspi0.p_ctrl, (uint8_t *) QSPI_DEVICE_ADDRESS); Erase a sector on the flash.
erase	g_qspi0.p_api->erase(g_qspi0.p_ctrl, (uint8_t *) QSPI_DEVICE_ADDRESS, *BYTE_COUNT*); Erase a block of memory depending on the input argument "byte_count"
statusGet	g_qspi0.p_api->statusGet(g_qspi0.p_ctrl, &in_progress); Get the write or erase status of the flash.
bankSelect	g_qspi0.p_api->bankSelect(0); Select the bank to access.

Function Name	Example API Call and Description
infoGet	g_qspi0.p_api->infoGet(g_qspi0.p_ctrl, &qspi_info); Provides information about the underlying QSPI flash, as specified in bsp_qspi.c
versionGet	g_qspi0.p_api->versionGet(& ssp_version); Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter is passed.
SSP_ERR_ASSERTION	p_cfg was NULL.
SSP_ERR_NOT_OPEN	Driver is not opened.
SSP_ERR_UNSUPPORTED	Driver not able to query the following information from the flash manufacturer id, memory capacity and memory type.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.37.3 QSPI HAL Module Operational Overview

The QSPI HAL module is used to initialize the QSPI peripheral so that the Synergy device can communicate (read, write, erase data) with a QSPI serial flash device.

The driver supports three operation modes: page program (write), read and erase.

- *Page program* programs a single page of data to the flash device. Page size is specific to flash memory and may vary from vendor to vendor. Typical page sizes of flash are 128,256 and 512 bytes. Use the infoGet API to get the supported page size for the underlying flash device.

- *Read* operation will read the data from the flash and store it to the user-provided buffer.
- *Erase* operation will erase the block of data from the flash. Use infoGet API to get the supported erase size for the underlying flash device.

NOTE: After any erase/write operation and before starting the next operation, it is advisable to use the statusGet API to poll the status of operation; not doing this may corrupt user data.

QSPI HAL Module Important Operational Notes and Limitations

In the case of using a board supported by the SSP and a BSP based project (SK-S7G2 and DK-S7G2), and the board having a QSPI memory device pre-installed, the BSP initializes and places the QSPI peripheral in ROM access mode with XIP (execute in place) enabled. This process enables the memory to be read like standard memory, meaning the QSPI HAL Module is only be needed when erasing and programming the QSPI flash device.

The typical QSPI application programs or erases data on the QSPI flash device. When this driver is not open, the QSPI flash device contents get mapped to 0x60000000 and can be read as if ordinary memory.

This driver has been tested on the S7G2 and S3A7 Synergy microcontroller groups using the QSPI peripheral block and the Micron N25Q256A QSPI flash device.

Refer to the latest SSP Release Notes for any additional operational limitations for this module.

4.2.37.4 Including the QSPI HAL Module in an Application

This section describes how to include the QSPI HAL module in an application using the SSP configurator.

NOTE: The process assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the QSPI Driver to an application, add it to a thread using the stacks selection sequence given in the following table. (The default name for the QSPI Driver g_qspi0. This name can be changed in the associated Properties window).

QSPI HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_qspi0	Threads	Select HAL/Common and select New > Driver > Storage > QSPI HAL on QSPI

Drivers that need additional configuration information are highlighted in Red. The QSPI does not require additional configuration.

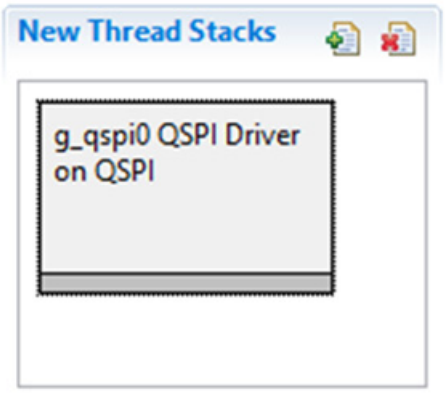


Figure 391: QSPI HAL Module Stack

4.2.37.5 Configuring the QSPI HAL Module

No component options of the QSPI HAL module need to be changed. The driver does not use the QSPI interrupt, so it is not enabled. Consider changing only the name of the driver instance for easier source code development.

The SSP configuration window automatically highlights any selections, such as interrupts or operating modes, requiring configuration for lower-level modules to achieve successful operation. In most applications, the default values for modules in the lower layers of the stack can be accepted. The following tables detail the available options that can be specified in the Properties window.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the configuration settings in the following table. This helps to orient you and provides a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the QSPI HAL Module on r_qspi

ISDE Property	Value	Description
Parameter Checking	Default (BSP)	Parameter checking level.
Name	g_qspi0	Name of the QSPI HAL instance. This may be edited to an application specific name if required.

NOTE: The example values and defaults given for settings come from a project using the Synergy S7G2 MCU. Other MCUs may have different default values and available configuration settings.

QSPI HAL Module Pin Configuration

The QSPI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table indicates the method for selecting pins within the SSP configuration window. The subsequent table lists an example selection sequence for the QSPI pins.

Pin Selection Sequence for the QSPI HAL Module

Resource	ISDE Tab	Pin selection Sequence
QSPI	Pins	Select Peripherals > Storage:QSPI > QSPI0

NOTE: The selection sequence assumes the QSPI0 is the desired hardware target for the driver.

Pin Configuration Settings for the QSPI HAL Module

Property	Value	Description
Pin Group Selection	- Mixed - _A only	Pin group selection.
Operation Mode	- Disabled - Custom - Single or Dual - Quad	Operation mode.
QSPCLK	None, P500	QSPI clock output pin.
QSSL	None, P501	QSPI slave select pin.
QIO0	None, P502	Data 0 input/output.
QIO1	None, P503	Data 1 input/output.
QIO2	None, P504	Data 2 input/output.
QIO3	None, P505	Data 3 input/output.

NOTE: The example settings come from a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.37.6 Using the QSPI HAL Module in an Application

Once the module has been configured and the files generated, the QSPI is ready to be used in an application.

The typical steps in using the QSPI HAL module in an application are:

1) Initialize the QSPI HAL module using the open API call.
2) Read a block of data using the read API call.
3) Erase a sector of data using the sectorErase API call.
4) Erase n bytes of data using the erase API call.
    a) infoGet can be used to get the supported erase sizes by the underlying flash.
    b) statusGet API can be used to poll the status of ease operation applicable for sectorErase API also.
5) Program a page of data using the pageProgram API call.
    a) Use infoGet API to get the page size supported by the underlying flash.
    b) statusGet API can be used to poll the status of write operation.
6) Close the QSPI HAL module using the close API call.

NOTE: It is advisable to use erase API instead of sectorErase API as the Sector is not a standard size for flash device and it varies with different vendors.

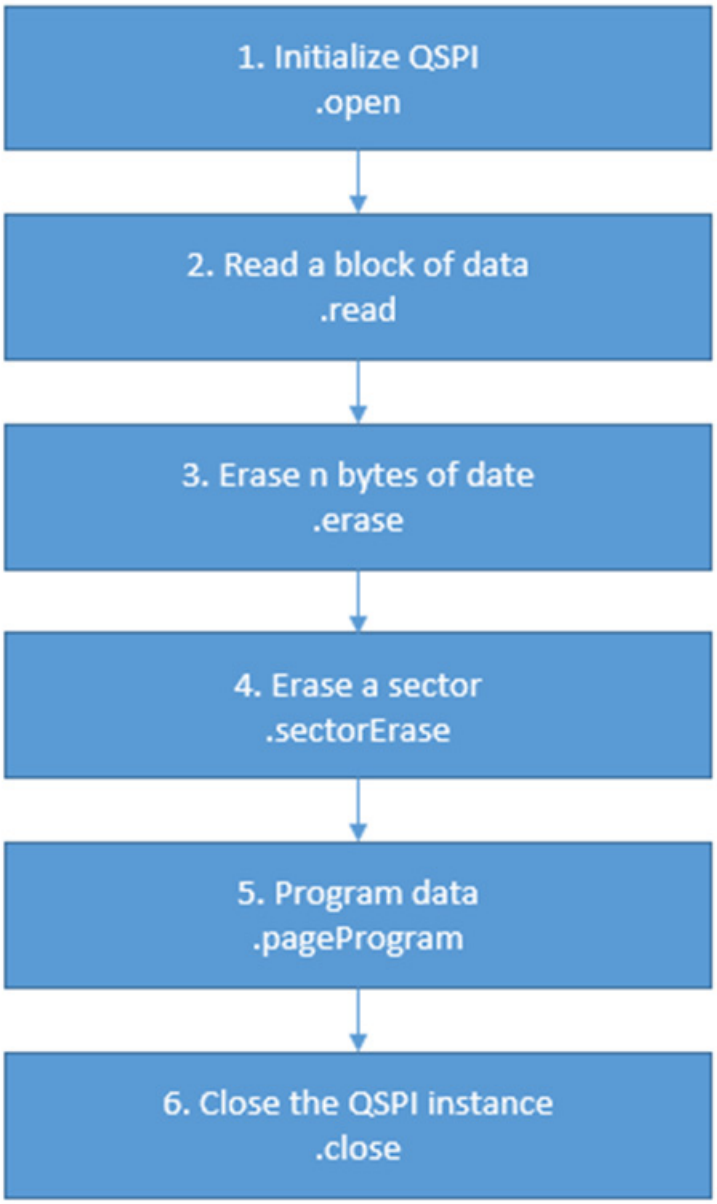


Figure 392: Flow Diagram of a Typical QSPI HAL Module Application

4.2.38 RTC Driver

The Real-Time Clock (RTC) HAL module provides high-level APIs for timing applications. The RTC HAL module configures the RTC peripheral and controls clock, calendar and alarm functions. The RTC uses the real-time clock peripheral on the Synergy MCU. A callback can be used to respond to any of the three supported interrupt types: alarm, periodic or carry.

4.2.38.1 RTC HAL Module Features

The RTC HAL module supports the following functions of the real-time clock:

- RTC peripheral configuration
- RTC time and date get and set
- RTC time and date alarm get and set
- RTC time counter start and stop
- RTC alarm, periodic, and carry event notification
- RTC event type enable and disable
- RTC event rate configuration
- RTC clock source get and set
- RTC sub-clock error adjustment
- RTC status get

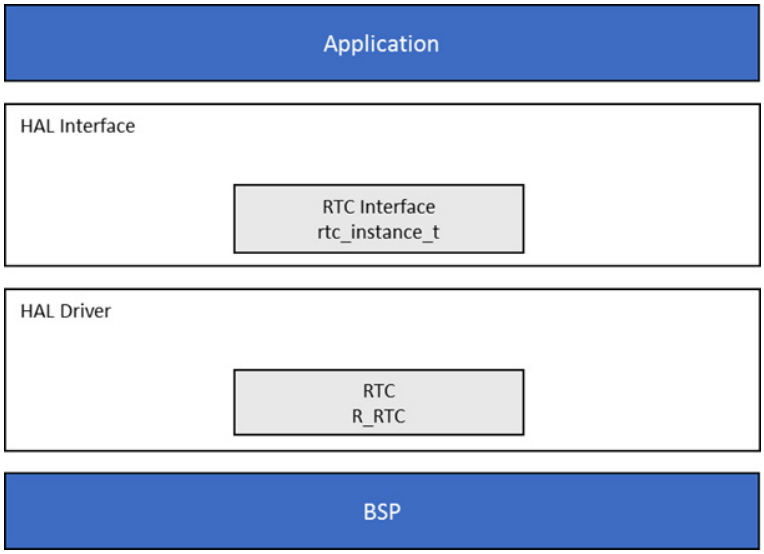


Figure 393: RTC HAL Module Block Diagram

4.2.38.2 RTC APIs Overview

The RTC HAL module defines APIs for opening, closing, setting alarms, starting, and stopping RTC operations. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of all applicable status return values follows.

RTC HAL Module API Summary

Function Name	Example API Call and Description
.open	g_rtc0.p_api->open(g_rtc0.p_ctrl, g_rtc0.p_cfg); Open the RTC HAL.
.close	g_rtc0.p_api->close(g_rtc0.p_ctrl); Close the RTC HAL.
.configure	g_rtc0.p_api->configure(g_rtc0.p_ctrl, p_extend); Configure the RTC HAL.
.calendarTimeSet	g_rtc0.p_api->calendarTimeSet(g_rtc0.p_ctrl, &start_time_struct_in, true); Set the calendar time.
.calendarTimeGet	g_rtc0.p_api->calendarTimeGet(g_rtc0.p_ctrl, &current_time_struct_out); Get the calendar time.
.calendarAlarmSet	g_rtc0.p_api->calendarAlarmSet(g_rtc0.p_ctrl, &in_alarm_time_struct_in, true); Set the calendar alarm time.
.calendarAlarmGet	g_rtc0.p_api->calendarAlarmGet(g_rtc0.p_ctrl, &get_alarm_time_struct_out); Get the calendar alarm time.
.calendarCounterStart	g_rtc0.p_api->calendarCounterStart(g_rtc0.p_ctrl); Start the calendar counter.
.calendarCounterStop	g_rtc0.p_api->calendarCounterStop(g_rtc0.p_ctrl); Stop the calendar counter.

Function Name	Example API Call and Description
.irqEnable	g_rtc0.p_api->irqEnable(g_rtc0.p_ctrl, CALLBACK); Enable the alarm irq.
.irqDisable	g_rtc0.p_api->irqDisable(g_rtc0.p_ctrl, CALLBACK); Disable the alarm irq.
.periodicIrqRateSet	g_rtc0.p_api->periodicIrqRateSet(g_rtc0.p_ctrl, Rate); Set the periodic irq rate.
.infoGet	g_rtc0.p_api->infoGet(g_rtc0.p_ctrl, clk_src); Return the currently configure clock source for the RTC.
.errorAdjustmentModeSet	g_rtc0.p_api->errorAdjustmentModeSet(g_rtc0.p_ctrl, p_mode); Set time error adjustment mode.
.errorAdjustmentSet	g_rtc0.p_api->errorAdjustmentSet(g_rtc0.p_ctrl, p_config); Set time error adjustment.
.versionGet	g_rtc0.p_api->versionGet(&version); Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function executed successfully

Name	Description
SSP_ERR_ASSERTION	API dependent error
SSP_ERR_INVALID MODE	Invalid mode.
SSP_ERR_INVALID_PTR	Invalid parameter.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.38.3 RTC HAL Module Operational Overview

The RTC HAL module controls the operation of the real-time clock module on a Synergy MCU. The typical RTC application configures the real-time clock controller periodically based on a system configuration driven by the user. Common operations include setting the time, setting an alarm, configuring a periodic interrupt, and starting or stopping operation. An RTC application usually consists of calls to the RTC HAL module and an optional callback from the ISR handler.

- The RTC HAL module can use two main clock sources
  - A Low Speed On-Chip Oscillator (LOCO) with lower power, but with less accuracy
  - A sub-clock oscillator with higher power, increased accuracy, and more cost (external crystal required)
- The RTC HAL module supports three different interrupt types
  - An alarm interrupt generated on a match of any combination of year, month, day, day of the week, hour, minute or second
  - A periodic interrupt generated every 2, 1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/128, or 1/256 second(s)
  - A carry interrupt when either a carry to the second counter occurs or when a carry to the R64CNT counter occurs during a read access to the 64 Hz counter

A user-defined callback function can be registered (in the open API call) and will be called from the interrupt service routine (ISR) for any supported interrupt type. When called, it is passed a pointer to a structure (rtc_callback_args_t) that holds a user-defined context pointer and an indication of which type of interrupt was fired.

NOTE: Carry interrupt priority must be set to avoid incorrect time returned from calendarTimeGet API during roll-over.

RTC HAL Module Important Operational Notes and Limitations

The RTC HAL module must be opened before any of the other RTC module APIs can be called. A configuration structure is passed to the open call which specifies the clock source, the name of the user callback from the ISR handler, and a user-specified context for the callback. Configuration structures can be either manually defined or generated by the ISDE based on user input during the configuration process.

Functions in the driver can be accessed by either making direct calls to the HAL layer or by using the RTC interface structure. The name of this interface structure is based on the name setting entered in the module's configuration. For example, if name = g_rtc, then the interface structure is called g_rtc_api.

The user may miss a carry event callback if a carry interrupt occurs when the calendarTimeGet API is in progress.

This module has no support for the following functions:

- Binary-count mode
- Binary alarm set and get
- Binary time get and set
- LOCLO clock-error correction
- 1 Hz/64 Hz Clock output

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.38.4 Including the RTC HAL Module in an Application

This section describes how to include the RTC HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the RTC Driver to an application, simply add it to a thread using the stacks selection sequence given in the table below. (The default name for the RTC is r_rtc0. This name can be changed in the associated Properties window.)

RTC Driver Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
r_rtc0 RTC HAL on r_rtc	Threads	New Stack> Driver> Timers> RTC HAL on r_rtc

When the RTC HAL module on r_rtc is added to the thread stack as shown in following figure, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information will be box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

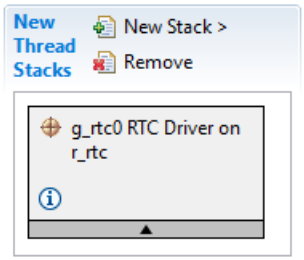


Figure 394: RTC HAL Module Stack

4.2.38.5 Configuring the RTC HAL Module

The RTC HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+.) This level of detail is not included in the configuration properties tables below, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the RTC HAL Module on r_rtc

Parameter	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.
Name	g_rtc0	The name to be used for the RTC module control block instance. This name is also used as the prefix of the other variable instances. See the example code below
Clock Source	LOCO, Sub-clock Default: LOCO	Clock source for the RTC block.
Configure RTC hardware in open() call	Yes, No Default: Yes	If enabled, the RTC registers and clock source will be initialized in the open() call. If disabled, the user call must call the configure() api to initialize the hardware.
Error Adjustment Value	0	Warning: Deprecated configuration field. Must be 0.

Parameter	Value	Description
Error Adjustment Type	None	Warning: Deprecated configuration field. Must be 0.
Callback	NULL	The name of the ISR that is called when one of the three interrupts fire. The argument passed into this ISR has an indication of which interrupt caused it to be called. See the example code below.
Alarm Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Alarm interrupt priority selection
Period Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	Period interrupt priority selection
Carry Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 (lowest - not valid if using ThreadX), Default: Priority 12	Carry interrupt priority selection

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for a module can be desirable. For example, it might be useful to select a different clock source than the default. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

RTC HAL Module Clock Configuration

The RTC HAL module can use the following clock sources:

- LOCO (Low Speed On-Chip Oscillator)
  - Lower-power consumption
  - Less accurate
- Sub-clock oscillator
  - Higher-power consumption

– More accurate

– More cost (requires a crystal)

The LOCO is the default selection during configuration.

RTC HAL Module Pin Configuration

The RTC doesn't currently support outputs, so no output pin selections are available.

4.2.38.6 Using the RTC HAL Module in an Application

General usage

The typical RTC application configures the real-time clock controller periodically based on a system configuration driven by the user. Examples include setting the time, setting an alarm, configuring a periodic interrupt, etc. An RTC application consists of calls to the RTC module and an optional callback.

The RTC module must be opened before any of the other APIs can be called. A configuration structure is passed to the open call which specifies the clock source, the name of the callbacks, and user-specified context for the handler. Configuration structures can be either manually defined or generated by the ISDE based on user input during the configuration process. Functions in the module can be accessed by using the RTC interface structure. The name of this interface structure is based on the name setting entered in the module's configuration.

Avoid drift issue after reset

To avoid drift in RTC time across MCU reset, the application needs to follow the steps given below.

Make these changes to module configuration settings:

1) Disable the "Configure Subclock Drive On Reset" option in the CGC stack element of the ISDE configurator.

2) Disable the "Configure RTC hardware in open() call" option in the RTC stack element of the ISDE configurator.

Make these calls in the application code:

1) Call the RTC open API as usual.

2) Call the RTC configure API only on a cold start. This will initialize the RTC only in a cold start condition.

The aforementioned steps can be performed by using the following initialization sequence in the application:

```
g_rtc.p_api->open(g_rtc.p_ctrl,g_rtc.p_cfg);
g_rtc.p_api->infoGet(g_rtc.p_ctrl,&info1);
/* initialize RTC if its status is stopped state i.e. on cold start */
if(RTC_STATUS_STOPPED == info1.status)
{
/* if the RTC clock source is sub-clock, stop it so that the sub-clock drive cap
acity is set correctly in the configure API */
       g_cgc.p_api->clockStop(CGC_CLOCK_SUBCLOCK);

       g_rtc.p_api->configure(g_rtc.p_ctrl);
       g_rtc.p_api->calendarTimeSet(g_rtc.p_ctrl,&rt_time,true);
}
```

An alternative way to determine cold start condition (instead of status) is to obtain the information from the reset status registers (RSTSRx).

Date and Time validation

The "Parameter Checking" needs to be enabled in the ISDE configurator if date and time validation is required for calendarTimeSet and calendarAlarmSet APIs. If "Parameter Checking" is enabled, the 'day of the week' field is automatically calculated and updated by the driver for the provided date. When using the calendarAlarmSet API, only the fields which have their corresponding match flag set are written to the registers. Other register fields are reset to default value.

Sub-Clock error adjustment

The errorAdjustmentModeSet and errorAdjustmentSet APIs can be used to correct the error in the RTC sub-clock source. These APIs need to be used only after the RTC is configured and time is set.

The error adjustment is reset every time RTC is reconfigured or time is set.

Usage Note: The configuration property 'Configure RTC hardware in open() call' in the RTC stack of ISDE configurator controls the behavior of open API. If enabled, the RTC peripheral is configured in the open API. If disabled, it is the responsibility of the application to make sure that RTC is configured before usage, by using the configure API.

The typical steps in using the RTC Alarm IRQ in an application are:

1) Initialize the RTC using the open API
2) Set calendar time using the calendarTime Set API
3) Set alarm time using the calendarAlarmSet API
4) Start calendar counter using the calendarCounterStart API

These common steps are illustrated in a typical operational flow diagram in the following figure:

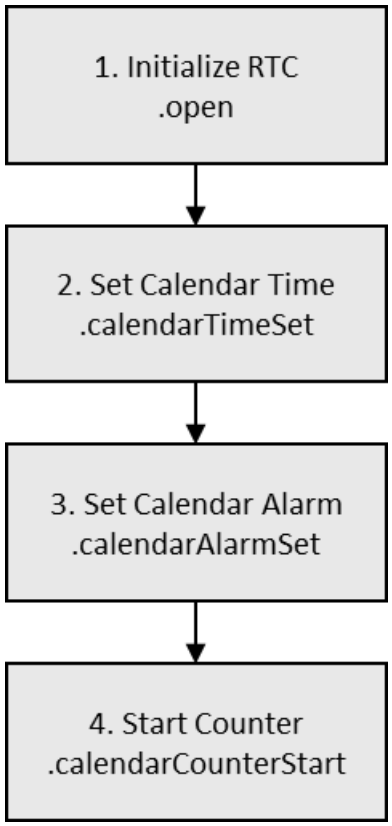


Figure 395: Flow Diagram of a Typical RTC Alarm Interrupt Application

The typical steps in using the RTC periodic IRQ in an application are:

1) Initialize the RTC using the open API
2) Set periodic IRQ rate using the periodicIrqRateSet API
3) Start calendar counter using the calendarCounterStart API
4) Enable interrupt using the irqEnable API.

These common steps are illustrated in a typical operational flow diagram in the following figure:

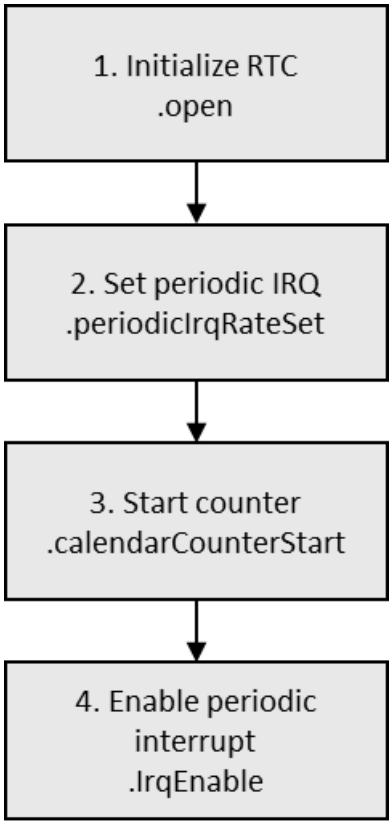


Figure 396: Flow Diagram of a Typical RTC Periodic Interrupt Application

4.2.39 SDADC Driver

The SDADC HAL module implements the ADC API for analog-to-digital conversions on r_sdadc. It supports the SDADC24 peripheral available on the Synergy microcontroller hardware. A user-defined callback can be created to process the data each time a new sample is available.

4.2.39.1 SDADC HAL Module Features

- 24-bit sigma delta A/D Converter
- Single scan or continuous scan operation mode

- Single-ended or differential input
- Gain of up to 32 on differential inputs
- Oversampling ratio configurable on differential inputs

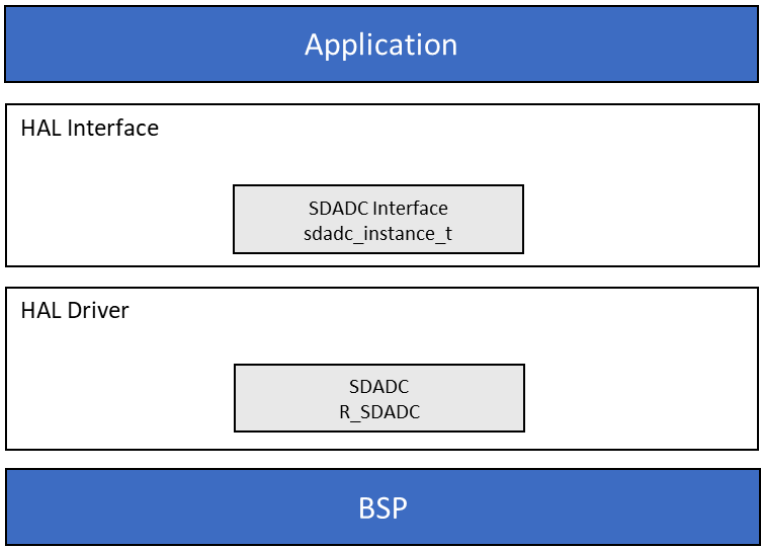


Figure 397: SDADC HAL Module Block Diagram

4.2.39.2 SDADC HAL Module APIs Overview

The SDADC HAL module defines APIs to open, configure scans, start scans, stop scans, read the conversion results the ADC scans, and close the ADC unit. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

SDADC HAL Module API Summary

Function Name	Example API Call and Description
.open	g_adc.p_api->open(g_adc.p_ctrl, g_adc.p_cfg); Initialize ADC Unit; apply power, set the operational mode, trigger sources, interrupt priority, and configurations common to all channels and sensors.
.scanCfg	g_adc.p_api->scanCfg(g_adc.p_ctrl, g_adc.*p*_channel_cfg); Configure the scan including the channels, groups and scan triggers to be used for the unit that was initialized in the open call.

Function Name	Example API Call and Description
.scanStart	g_adc.p_api->scanStart(g_adc.p_ctrl); Start the scan (in case of a software trigger), or enable the hardware trigger.
.scanStop	g_adc.p_api->scanStop(g_adc.*p_ctrl*); Stop the ADC scan (in case of a software trigger), or disable the hardware trigger.
.scanStatusGet	g_adc.p_api->scanStatusGet(g_adc.p_ctrl); Check scan status.
.read	g_adc.p_api->read(g_adc.p_ctrl, ADC_REG_CHANNEL_13, &adc_data); Read ADC conversion result.
.read32	g_adc.p_api->read32(g_adc.p_ctrl, ADC_REG_CHANNEL_13, &adc_data); Read ADC conversion result into a 32-bit word.
.sampleStateCountSet	g_adc.p_api-> sampleStateCountSet(g_adc.p_ctrl,&adc_sample); Set the sample state count for the specified channel.
.calibrate	g_adc.p_api-> calibrate(g_adc.p_ctrl, reg_id, offset); Calibrate ADC or associated PGA (programmable gain amplifier). The driver may require implementation specific arguments to the p_extend input.
.offsetSet	g_adc.p_api-> offsetSet(g_adc.p_ctrl, p_extend); Set offset for input PGA configured for differential input.

Function Name	Example API Call and Description
.close	g_adc.p_api->close(g_adc.p_ctrl); Close the specified ADC unit by ending any scan in progress, disabling interrupts, and removing power to the specified A/D unit.
.infoGet	g_adc.p_api->infoGet(g_adc.p_ctrl, &adc_info); Return the ADC data register address of the first (lowest number) channel and the total number of bytes to be read for the DTC/DMAC to read the conversion results of all configured channels.
.versionGet	g_adc.p_api->versionGet(&version); Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.
SSP_ERR_NOT_OPEN	Unit is not open
SSP_ERR_ASSERTION	The parameter p_ctrl or p_sample is NULL.
SSP_ERR_IN_USE	Peripheral is still running in another mode; perform R_ADC_Close first.
SSP_ERR_INVALID_POINTER	The parameter p_data is NULL.
SSP_ERR_CALIBRATION_FAILED	Calibration failed.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status-return values.

4.2.39.3 SDADC HAL Module Operational Overview

The SDADC HAL module controls the SDADC peripheral on a Synergy microcontroller. It directly controls the SDADC hardware without using any RTOS elements and provides convenient APIs to simplify development.

In this document, the term 'scan' refers to the AUTOSCAN feature of the SDADC, which works as follows:

1) Conversions are performed on enabled channels in ascending order of channel number. All conversions required for a single channel are completed before the sequencer moves to the next channel.
2) Conversions are performed at the rate (in Hz) of the SDADC oversampling clock frequency / oversampling ratio (configured per channel). The SSP uses the normal mode SDADC oversampling clock frequency.
3) If averaging is enabled for the channel, the number of conversions to average are performed before each conversion end interrupt occurs.
4) If the number of conversions for the channel is more than 1, performs the number of conversions requested. If averaging is enabled for the channel, each averaged result counts as a single conversion.
5) Continues to the next enabled channel only after completing all conversions requested.
6) After all enabled channels are scanned, a scan end interrupt occurs.

The driver supports single-scan and continuous scan operation modes.

- Single-scan mode performs one scan per trigger (hardware trigger or software start using scanStart).
- In continuous scan mode, the scan is restarted after each scan completes. A single trigger is required to start continuous operation of the SDADC.

Interrupts and Callbacks

When a conversion is complete and a callback is provided by the user, the SDADC HAL module calls the callback (adc_api_t::p_callback) with the argument adc_callback_args_t, indicating the unit and the event adc_cb_event_t.

The SDADC driver supports the following callback events:

- ADC_EVENT_CONVERSION_COMPLETE to notify the application that new conversion data is available.
- ADC_EVENT_SCAN_COMPLETE to notify the application when a scan is complete.
- ADC_EVENT_CALIBRATION_COMPLETE to notify the application that the calibration process is complete.

SDADC HAL Module Important Operational Notes and Limitations

Triggering a Data Transfer with the SDADC

To trigger a transfer of data when the SDADC scan completes, configure the data transfer with activation_source set to ELC_EVENT_SDADCn_SCAN_END. The ELC events are listed under elc_event_t. To retrieve the SDADC specific information to use with the transfer API, use the infoGet function call in the transfer API.

Triggering ELC Events with the SDADC

The SDADC unit can trigger the start of other peripherals listed in elc_peripheral_t. Refer to the "ELC Interface" in the SSP User's Manual for more information.

This module only works for selected Synergy MCUs. Consult the release notes for your current SSP release to see which MCUs are supported by this module. Additionally, the MCU Hardware Manual shows which peripherals are available.

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.39.4 Including the SDADC HAL Module in an Application

This section describes how to include the SDADC HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the SDADC HAL module to an application, simply add it to a HAL /Common thread using the stacks selection sequence given in the following table. (The default name for the SDADC HAL module is g_sdadc0. This name can be changed in the associated Properties window.)

SDADC HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sdadc0 SDADC Driver on r_adc	Threads	New Stack> Driver> Analog> SDADC Driver on r_sdadc

When the SDADC HAL module on r_sdadc is added to the thread stack as shown in the following figure, the configurator automatically adds any lower-level drivers that are required. Any drivers that need additional configuration information will be box-text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

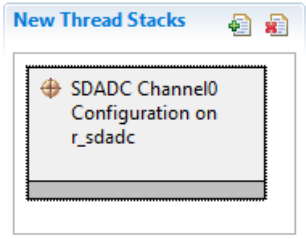


Figure 398: SDADC HAL Module Stack

4.2.39.5 Configuring the SDADC HAL Module

The SDADC HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included

in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the SDADC HAL Module on r_sdadc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.
Name	g_adc0	Module name.
Mode	Single Scan, Continuous Scan Default: Continuous Scan	In single scan mode, all channels are converted once per start trigger, and conversion stops after all enabled channels are scanned. In continuous scan mode, conversion starts after a start trigger, then continues until stopped in software.
Resolution	16 Bit, 24 Bit Default: 24 Bit	Select 24-bit or 16-bit resolution.
Alignment	Right, Left Default: Right	Select left or right alignment.
Trigger	ELC Hardware Event, Software Default: Software	Select conversion start trigger. Conversion can be started in software, or conversion can be started when a hardware event occurs if the hardware event is linked to the SDADC peripheral using the ELC API.
Vref Source	Internal, External Default: Internal	Vref can be sourced internally and output on the SBIAS pin, or Vref can be input from VREFI.

ISDE Property	Value	Description
Vref Voltage	0.8V, 1.0V, 1.2V, 1.4V, 1.6V, 1.8V, 2.0V, 2.2V, 2.4V Default: 1.0V	Select Vref voltage. If Vref is input externally, the voltage on VREFI must match the voltage selected within 3%.
Internal Calibration During Open()	Enabled, Disabled Default: Enabled	Calibration is required for all channels configured for differential input. Internal calibration is performed automatically during open for these channels unless it is disabled here.
Callback	NULL	Enter the name of the callback function to be called when conversion completes or a scan ends.
Conversion End Interrupt Priority	Priority 0 (highest), Priority 1, Priority 2, Priority 3 (lowest - not valid if using ThreadX) Default: Priority 2	Select the interrupt priority for the conversion end interrupt. [Required]
Scan End Interrupt Priority	Priority 0 (highest), Priority 1, Priority 2, Priority 3 (lowest - not valid if using ThreadX), Disabled Default: Priority 2	Select the interrupt priority for the scan end interrupt. [Required]
Calibration End Interrupt Priority	Priority 0 (highest), Priority 1, Priority 2, Priority 3 (lowest - not valid if using ThreadX), Disabled Default: Priority 2	Select the interrupt priority for the calibration end interrupt. [Required]

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SDADC Channel Configuration on r_sdadc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable parameter error checking.
Input	Differential, Single-Ended Default: Differential	Select differential or single-ended input.
Stage 1 Gain	1,2,3,4,8, Default: 1	Select the gain for stage 1 of the PGA. Must be 1 for single-ended input.
Stage 2 Gain	1,2,3,4,8, Default: 1	Select the gain for stage 2 of the PGA. Must be 1 for single-ended input.
Oversampling	64, 128, 256, 512, 1024, 2048 Default: 256	Select the oversampling ratio for the PGA. Must be 256 for single-ended input.
Polarity (Valid for Single-Ended Input Only)	Positive, Negative Default: Positive	Select positive or negative polarity for single-ended input. VBIAS (1.0 v typical) is connected on the opposite input.
Conversions to Average per Result	Do Not Average (Interrupt after Each Conversion), Average 8, Average 16, Average 32, Average 64	Select the number of conversions to average for each result. The ADC_EVENT_CONVERSION_END event occurs after each average, or after each individual conversion if averaging is disabled.
Invert (Valid for Negative Single-Ended Input Only)	Result Not Inverted, Result Inverted Default: Result Not Inverted	Select whether to invert negative single-ended input. When the result is inverted, the lowest measurable voltage gives a result of 0, and the highest measurable voltage gives a result of 2 carrot resolution -1.

ISDE Property	Value	Description
Number of Conversions Per Scan	Default: 1	Number of conversions on this channel before AUTOSCAN moves to the next channel. When all conversions of all channels are complete, the ADC_EVENT_SCAN_END event occurs.

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

SDADC HAL Module Clock Configuration

The SDADC HAL module uses the SDADCCLK as its clock source and it must be set to 4MHz.

SDADC HAL Module Pin Configuration

To use the SDADC HAL module, the port pins for the channels receiving the analog input must be set as input pins in the pin configurator in the ISDE. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection Sequence for the SDADC HAL Module

Resource	ISDE Tab	Pin selection Sequence
SDADC	Pins	Select Peripherals > Analog:SDADC>SDADCn where n is the channel number

4.2.39.6 Using the SDADC HAL Module in an Application

Once the module has been configured and the files generated, the SDADC is ready to be used in an application.

The typical steps in using the SDADC HAL module in an application are:

1) Initialize the SDADC using open. Calibration is performed on all channels configured for differential input during this function unless calibration during open is disabled in the configuration.
2) If calibration during open is disabled in the configuration, calibrate each channel configured for differential input using calibrate. Wait for a calibration complete interrupt after calibrating each channel. See R_SDADC_Calibrate for details regarding the calibration procedure.
3) [Optional] Configure active channels using scanCfg.

4) Start a conversion using the desired trigger with scanStart.
   a) If a hardware trigger is used, this call enables the ADC unit to be triggered by the hardware trigger. If a software trigger is used, then this call starts the ADC scan.
5) The callback will be called after each conversion, and when the entire scan is complete.
6) Read the results of the conversion using read.
7) [Optional] Stop the ADC scan by calling scanStop.
   a) This prevents the ADC from being triggered by an external trigger or a hardware trigger; it also forces a stop of a software-triggered scan if one is ongoing.
8) Operate on the received data as needed by the application.
9) [Optional] Use close to power down the peripheral.

These common steps are illustrated in a typical operational flow in the following figure:

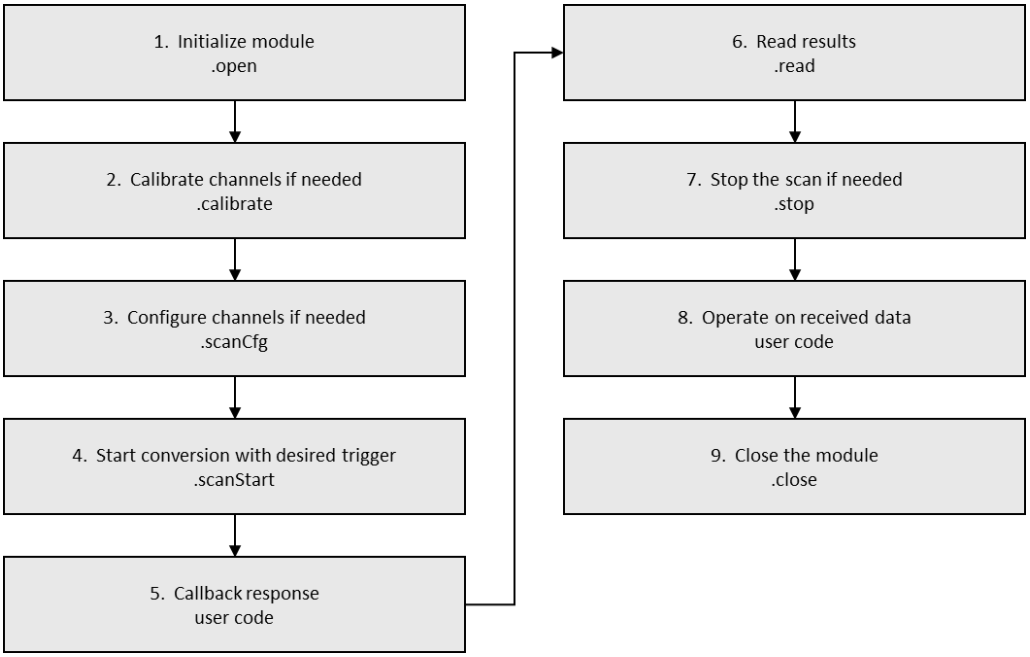


Figure 399: Flow Diagram of a Typical SDADC HAL Module Application

4.2.40 SD/MMC Driver and SDIO Driver

The SDMMC (SD/MMC and SDIO) module is implemented on r_sdmmc and is used to read/write and control SD/MMC media devices as well as SDIO cards. The SDMMC module can be used as a standalone SD card, or eMMC, media driver or it can be used with FileX™, or any other compatible file system. The r_sdmmc module uses the SD/MMC peripheral on the Synergy MCU.

4.2.40.1 SDMMC HAL Module Features

- Supports the following memory devices: SDSC (SD Standard Capacity), SDHC (SD High Capacity), SDXC (SD Extended Capacity), and eMMC (embedded Multi Media Card)
  - Supports reading, writing, and erasing SD and eMMC memory devices
  - Supports 1, 4, or 8 bit data bus (8-bit bus is supported for eMMC only)
  - Supports detection of hardware write protection (SD cards only)
  - Automatically selects between backwards compatible mode and high-speed SRD mode (eMMC)
- Supports SDIO
  - Supports SDIO single register access (CMD52)
  - Supports SDIO multiple register access (CMD53)
  - Supports SDIO interrupts
- Automatically configures the clock to the maximum clock rate supported by both host (MCU) and device

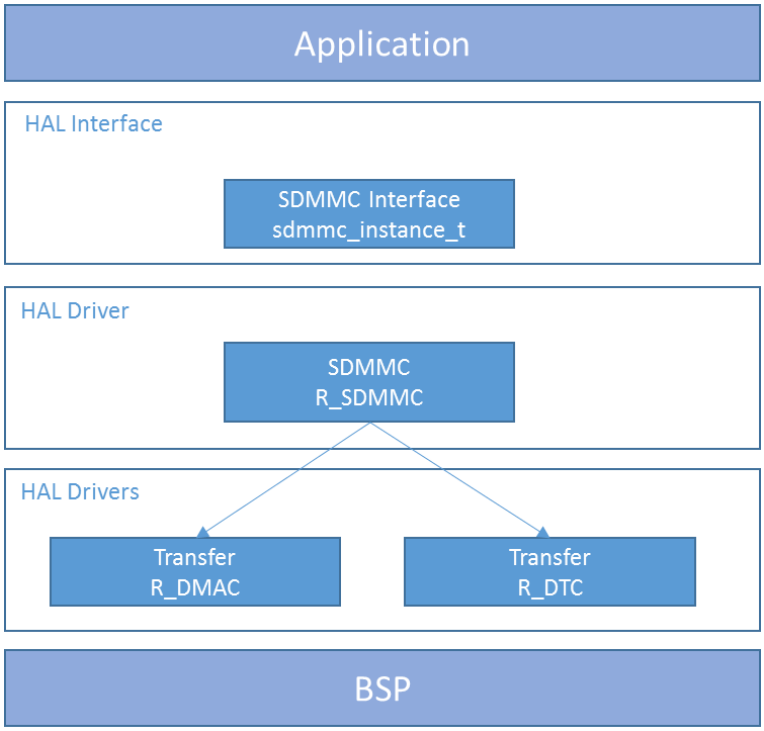


Figure 400: SDMMC HAL Organization, Options and Stack Implementations

4.2.40.2 SDMMC HAL Module APIs Overview

This document is divided into the following sections which can be read independently (by a more experienced Synergy developer) or in series (by developers new to the Synergy Platform). SD/MMC Driver and SDIO Driver APIs Overview

SD/MMC Driver API Summary

Function Name	Example API Call and Description
open	g_sdmmc.p_api->open(g_sdmmc.p_ctrl, g_sdmmc.p_cfg); Open device channel for read/write and control. Initialize driver and hardware on first call.
read	g_sdmmc.p_api->read(g_sdmmc.p_ctrl, &destination, start, count); Read data from SD/MMC channel.
write	g_sdmmc.p_api->write(g_sdmmc.p_ctrl, &source, start, count); Write data to SDMMC channel.
control	g_sdmmc.p_api->control(g_sdmmc.p_ctrl, command, &data); Send control commands to the SD/MMC port and receive the status of the SD/MMC port.
close	g_sdmmc.p_api->close(g_sdmmc.p_ctrl); Close open SDMMC channel. Turns off hardware if last channel open.
versionGet	g_sdmmc.p_api->versionGet(&version); Get version of Block Media SD/MMC driver.
readIo	g_sdmmc.p_api->readIo(g_sdmmc.p_ctrl, &data, function, address); Read I/O data from an SDMMC channel.

Function Name	Example API Call and Description
writeIo	g_sdmmc.p_api->writeIo(g_sdmmc.p_ctrl, &data, function, address, read_after_write); Write I/O data to SDMMC channel.
readIoExt	g_sdmmc.p_api->readIoExt(g_sdmmc.p_ctrl, &destination, function, address, count, transfer_mode, address_mode); Read I/O data, extended, from an SDMMC channel.
writeIoExt	g_sdmmc.p_api->writeIoExt(g_sdmmc.p_ctrl, &source, function, address, count, transfer_mode, address_mode); Write I/O data, extended, to SDMMC channel.
ioIntEnable	g_sdmmc.p_api->ioIntEnable(g_sdmmc.p_ctrl, enable); Enables SDIO interrupt for SDMMC channel.
infoGet	g_sdmmc.p_api->infoGet(g_sdmmc.p_ctrl, &info); Get SDMMC channel info.
erase	g_sdmmc.p_api->erase(g_sdmmc.p_ctrl, start, count); Erase SDMMC sectors.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables review the SSP Reference Manual listed in the Reference Section at the end of this document. For FileX API documentation refer to X-Ware and NetX Component Documents for Renesas Synergy listed in the Reference Section as well.

Error Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.

Name	Description
SSP_ERR_IN_USE	The channel specified has already been opened. No configurations were changed. Call the associated Close function or use associated Control commands to reconfigure the channel.
SSP_ERR_ASSERTION	The pointer is NULL.
SSP_ERR_WRITE_PROTECTED	SD or MMC card is Write Protected.
SF_INFO_NOT_AVAILABLE	Information not available possibly because card has been removed or is defective.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_HW_LOCKED	The hardware lock has already been acquired.
SSP_ERR_TRANSFER_BUSY	The transfer is in progress.
SSP_ERR_CARD_NOT_READY	The card is not ready yet.
SSP_ERR_NOT_ENABLED	SDIO interrupt enable failed.
SSP_ERR_READ_FAILED	Read operation failed.
SSP_ERR_WRITE_FAILED	Write operation failed.

NOTE: Lower level drivers may return Common Error Codes. Refer to the SSP Reference Manual listed in the Reference Section at the end of this document for a definition of all relevant Error codes.

4.2.40.3 SDMMC HAL Module Operational Overview

The following descriptions cover the operational functions and requirements when using the SDMMC HAL module.

4.2.40.4 SDMMC HAL Module Interrupt Configurations

The interrupts required are:

- Using SD/MMC with DTC:
  - Access Interrupt
  - DTC Interrupt
- Using SD/MMC with DMAC:
  - Access Interrupt

  - DMAC Interrupt (in DMAC instance)
- Using SDIO with DTC:
  - Access Interrupt
  - SDIO Interrupt
  - DTC Interrupt
- Using SDIO with DMAC:
  - Access Interrupt
  - SDIO Interrupt
  - DMAC Interrupt (in DMAC instance)

The Card interrupt is optional, and only available on MCU packages that have the SDHIn CD pin (n = channel number) available on the Pins tab of the Synergy Configuration Tool.

4.2.40.5 SDMMC HAL Module Block Size Configuration

Block size configuration is provided for use with SDIO cards only. For SDIO cards, block size may be configured to 1-512 bytes. Block size must remain at the default 512 bytes for SD cards and e/MMC memory devices.

4.2.40.6 SDMMC HAL Module Card Detection Configuration

See Card Detection Limitations before using card detection in the r_sdmmc driver. If card detection is not available or not desired for the application, Card Detection must be set to Not Used in the Properties for the r_sdmmc instance in the Synergy Configuration Tool. To enable card detection, set Card Detection to CD Pin and Media Type to Card in the Properties for the r_sdmmc instance in the Synergy Configuration Tool.

4.2.40.7 SDMMC HAL Module Access Interrupt Priority

When data transfers are not 4-byte aligned or not a multiple of 4 bytes, a software copy of the block size (up to 512 bytes) is done in the access interrupt. This blocks other interrupts that are a lower or equal priority to the access interrupt until the software copy is complete.

4.2.40.8 SDMMC HAL Module General Timing Notes

Several functions in this driver block. open and erase block until the entire operation is complete. read,write,readIo,writeIo,readIoExt, and writeIoExt block for a single command response cycle. In a multithreaded system, care should be taken to use this driver in a lower priority thread if other threads require a response time faster than the time this driver could block for during one of these function calls.

4.2.40.9 SDMMC HAL Module Timing Notes for Open

The open API completes the entire device identification and configuration process. This involves several command-response cycles at a bus width of 1-bit and a bus speed of 400 kHz or less.

4.2.40.10 SDMMC HAL Module Timing Notes for Read, Write, ReadIoExt, and WriteIoExt

The read and write media (read and write) and extended read and write SDIO APIs (readIoExt and writeIoExt) block until the response is received from the device. The data transfer operation is non-blocking and requires interrupts and a transfer instance, either DMAC or DTC. These APIs return SSP_SUCCESS to indicate that the initial operations have started successfully. The application must wait for a callback with the event SDMMC_EVENT_TRANSFER_COMPLETE or SDMMC_EVENT_TRANSFER_ERROR to indicate completion of the read or write.

4.2.40.11 SDMMC HAL Module Timing Notes for ReadIo and WriteIo

The SDIO readIo and writeIo APIs block until the response is received from the device. The read or write operation is complete when these APIs return.

4.2.40.12 SDMMC HAL Module Timing Notes for Erase

The erase API blocks until the erase operation is complete. This may be several seconds or more depending on how many sectors are being erased.

4.2.40.13 SDMMC HAL Module SDIO Interrupts

Many SDIO cards use level interrupts, meaning the interrupt is not deasserted until the interrupt is cleared on the device. In order to ensure SDIO interrupts are cleared appropriately, one of the following methods are recommended:

- Ensure the SDIO interrupt is cleared on the device before exiting the callback function with the event SDMMC_EVENT_SDIO. If this method is chosen and any SDIO API must be used in the interrupt, the access interrupt must be a higher priority than the SDIO interrupt.
- Disable the SDIO interrupt in the callback function with the event SDMMC_EVENT_SDIO using IoIntEnable. Clear the SDIO interrupt elsewhere in the application, then re-enable SDIO interrupts if desired using IoIntEnable.

4.2.40.14 Important Operational Notes and Limitations

SDMMC HAL Module Operational Notes

SD HALA Compliance

When developing host devices that are compliant with the SD Specifications, the host must comply with the SD Host/Ancillary Product License Agreement (SD HALA).

SDMMC HAL Module Limitations

Data Alignment and Size

Data transfers should be 4-byte aligned and a multiple of 4 bytes in size whenever possible. This recommendation applies to the read(), write(), readIoExt(), and writeIoExt() APIs. When data transfers are 4-byte aligned and a multiple of 4-bytes, the r_sdmmc driver is zero copy and takes full advantage of hardware acceleration by the DMAC or DTC. When data transfers are not 4-byte aligned or not a multiple of 4 bytes an extra CPU interrupt is required for each block transferred (see Access Interrupt Priority).

Card Detection Limitations

Card detection support in the r_sdmmc driver is limited. Card detection is only available after open is complete, and open() cannot be completed unless a card is inserted. Card detection in the r_sdmmc driver is therefore only useful to detect card removal and reinsertion. After reinsertion is detected, the driver must be closed and reopened, and card detection will not be available until the reopen is complete. Card detection is available only on MCU packages that have the SDHIn CD pin (n = channel number) available on the Pins tab of the Synergy Configuration Tool. If the MCU does not have the SDHIn

CD pin, or card detection is not desired for the application, card detection must be disabled in the Properties for the r_sdmmc instance in the Synergy Configuration Tool. An alternative to using the card detection feature of the r_sdmmc driver is to use an External IRQ instance and handle card detection at the application layer. If card detection is handled at the application layer, the open should be called after card insertion is detected, and the close should be called after card removal is detected.

Refer to the most recent SSP Release notes for the most up to date limitations for this module.

4.2.40.15 Including the SDMMC HAL Module in an Application

This section describes how to include the SDMMC HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the "SSP User's Manual", available as described in the Reference Section at the end of this document to learn how to manage each of these important steps in creating SSP based applications.

To add the module to an application, simply add it to a thread using the Stacks Selection Sequence given in the following table. (The default name for the modules is shown in the following table. This name can be changed in the associated Properties window). When the driver is added it shows up in the thread as illustrated in the following figure.

Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sdmmc0 SD/MMC driver on r_sdmmc	Threads	New Stack> Driver> Storage> SD/MMC Driver on r_sdmmc

When the SD/MMC driver on r_sdmmc is added to the Thread Stack, as shown in the following figure, the configurator automatically adds the needed lower level drivers. Any drivers that need additional configuration information will be box text highlighted in red. Modules with a gray band are individual modules that stand alone. Modules with a blue band are shared or common and need only be added once, since they can be used by multiple stacks. Modules with a pink band can require the selection of lower level drivers. Sometimes these are optional or recommended and this is indicated in the block with the inclusion of this text. If the addition of lower level drivers is required, the module description will include "Add" in the text. Clicking on any pink banded modules will bring up the "New" icon and then will show the possible choices.

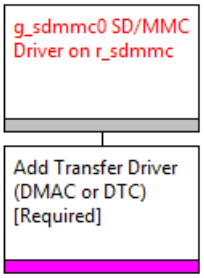


Figure 401: SD/MMC Driver on r_sdmmc Stack

4.2.40.16 Configuring the SDMMC HAL Module

The module must be configured by the user for the desired operation. The SSP configuration window will automatically identify, by highlighting the block in red, any required configuration selections, such as Interrupts or operating modes, which must be configured for lower level modules, for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP Configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the Interrupt Priority. This configuration setting is available with the Properties window of the associated module. Simply select the indicated module and then view the properties window. The Interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the Interrupt Priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the below configuration properties tables, but is easily visible with the ISDE when configuring Interrupt Priority levels.

NOTE: You may want to open your ISDE and create the SDMMC HAL module and explore the property settings in parallel with reviewing the following table. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration for the SD/MMC and SDIO Driver

ISDE Property	Setting	Description
Parameter Checking	Enabled, Disabled, BSP (Default: BSP)	Selects if code for parameter checking is to be included in the build
Name	Default: g_sdmmc0	Module name
Channel	0,1 (Default: 1)	Channel of SDMMC peripheral, channel 0 or 1
Media type	Embedded, Card (Default: Embedded)	Media is a card or an embedded device. This allows to firmware to know whether to look for card insertion/removal and write protect pins.
Bus Width	1 bit, 4 bits, 8 bits	Data bus with as defined by hardware interface. (8 Bits for eMMC only)
Block Size	512	Size of block

ISDE Property	Setting	Description
Callback	NULL (default) Name of user callback function.	(Not required if using FileX) Set to name of user callback function. Provides event that caused interrupt: SDMMC_EVENT_CARD_REMOVED, SDMMC_EVENT_CARD_INSERTED, SDMMC_EVENT_ACCESS, SDMMC_EVENT_SDIO, SDMMC_EVENT_TRANSFER_COMPLETE, SDMMC_EVENT_TRANSFER_ERROR
Access Interrupt Priority	Priority level 0-15 (Default: Disabled)	Interrupt level priority required for reading, writing, and errors for SD, eMMC and SDIO. Calls the interrupt callback if enabled.
Card Interrupt Priority	Priority level 0-15 (Default: Disabled)	(Optional) Interrupt for card removal. Automatically closes device when card is removed. Calls the interrupt callback if enabled.
DMA Interrupt Priority	Priority level 0-15 (Default: Disabled)	Interrupt priority level required for SDIO reading, writing, and errors. Calls the interrupt callback if enabled. Not used for memory cards.

NOTE: The examples and defaults listed apply to projects using the Synergy S7 MCU. Other MCUs may have different default values and available configuration settings.

4.2.40.17 SDMMC HAL Module Clock Configuration

The SDMMC MCU peripheral (SDHI) uses PCLKA for its clock source. There is no need to configure the clock specifically for the SDMMC peripheral, unless you need to optimize the data rate. The SDMMC driver selects the appropriate built-in divider based on the PCLKA frequency and the maximum clock rate allowed by the SD, SDIO, or eMMC device, obtained at media device initialization.

4.2.40.18 SDMMC HAL Module Pin Configuration

Use the ISDE pin configurator to configure the I/O pins for SDMMC peripheral (SDHI). The drive capacity for each pin should be set to "Medium" or "High" for most boards and high-speed memory and SDIO devices. The following tables illustrate the method for selecting the pins and configuring the pins within the associated SSP configuration window.

NOTE: The Operation Mode selection mode determines what peripheral signals are available and thus what MCU pins are required.

Pin Selection Sequence for SDHI Peripheral

Resource	ISDE Tab	Pin selection Sequence
SDHI	Pins	Select **Peripherals > Storage:SHDI > SDHI0**

NOTE: The selection sequence assumes the SCI1 is the desired hardware target for the driver.

Pin Configuration Settings for SDHI Peripheral

Pin Configuration Property	Settings	Description
Operation Mode	Disabled,	Select mode as per application
	Custom,	
	SD_MMC 1 bit	
	SD_MMC 4 bit	
	MMC 8 bit (Default: Custom)	
CLK	None, P413 (Default: P413)	Clock Pin
CMD	None, P412 (Default: P412)	Command Pin
DAT0-7	None, PXXX (Default: PXXX)	Data Pin
CD	None, P903 (Default: P903)	Card detection Pin
WP	None, P414 (Default: P414)	Card write protection pin

NOTE: The settings listed apply to a project using the Synergy S7G2 MCU and the DK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.40.19 Using the SDMMC HAL Module in an Application

The typical steps for using the r_sdmmc driver with an SD or eMMC memory device are:

1) Open the driver using the open API.
2) Read data from the card using the read API or write data to the card using write API.

3) Wait for a callback with the event SDMMC_EVENT_TRANSFER_COMPLETE (meaning the operation completed successfully) or SDMMC_EVENT_TRANSFER_ERROR (meaning the operation did not complete successfully) after each time the read API or the write API is called.

These common steps are shown in a typical operational flow in the following figure.

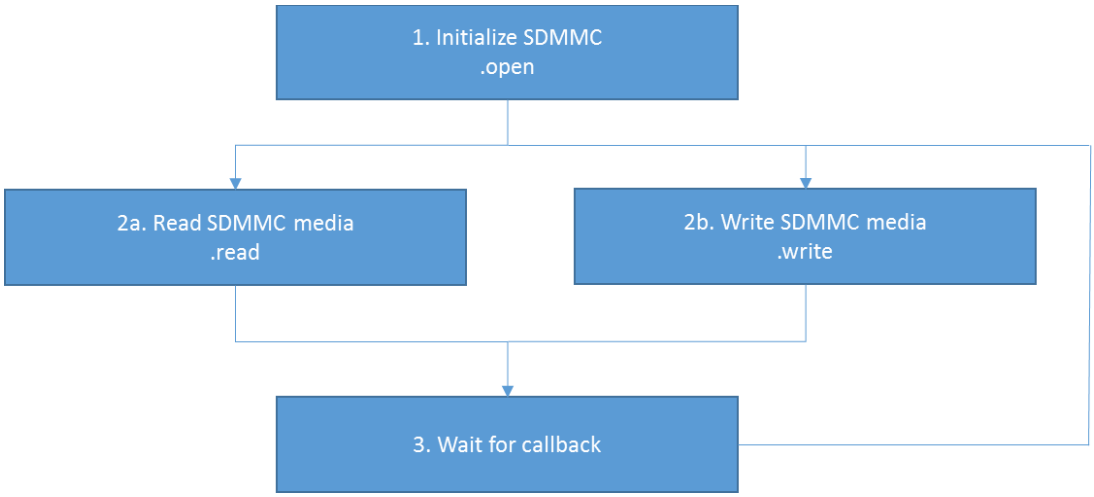


Figure 402: Flow for Typical SDMMC HAL Module Application for SD or

eMMC memory devices"

The typical steps for using the r_sdmmc driver with an SDIO card are:

1) Open the driver using the open API.
2) Set or read back single registers using the readIo API or the writeIo API. The operation is complete immediately after these calls and there is no need to wait for a callback.
3) Set or read back multiple registers using the readIoExt API or the writeIoExt API. The block size can be changed using the control API between calls if necessary.
4) Wait for a callback with the event SDMMC_EVENT_TRANSFER_COMPLETE (meaning the operation completed successfully) or SDMMC_EVENT_TRANSFER_ERROR (meaning the operation did not complete successfully) after each time the readIoExt API or the writeIoExt API is called.
5) If the card requests an interrupt, the callback is called with the event SDMMC_EVENT_SDIO. Handle the SDIO interrupt as described in the documentation for the card (see the SDIO Interrupts section of this documentation). SDIO interrupts from the card can be enabled or disabled at any time using the IoIntEnable API.

These common steps are illustrated in a typical operational flow diagram in the following figure.

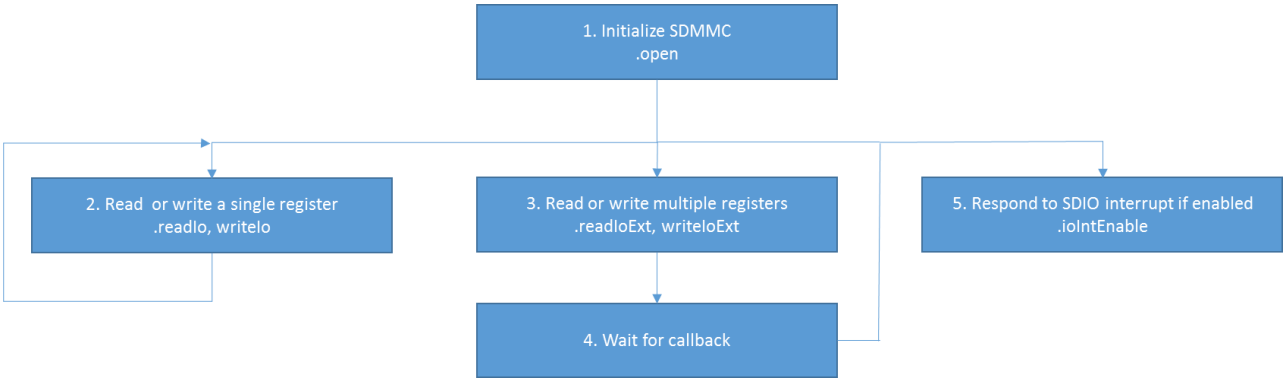


Figure 403: Flow for Typical SDMMC HAL Module Application for SDIO Card

4.2.40.20 FileX Port Block Media Framework Application Project

The Application Project associated with this Module Guide demonstrates the above steps in a full design. The project can be found using the link provided in the Reference Section at the end of this document. You may want to import and open the Application Project within ISDE and view the configuration settings for the FileX Port Block Media Framework Module. You can also read over the code, in sdmmc_thread_entry.c, which is used to illustrate the FileX on Block Media API's in a complete design.

The Application Project demonstrates the typical use of the FileX on Block Media Framework APIs. The Application Project auto-generated code initializes FileX framework using the FileX Port Block Media Framework. It is a driver which aims to implement FileX-specific accesses using the Block Media Framework. The underlying implementation of the Block Media Framework uses SD/MMC Driver. Specifically, the Application Project is designed to work with onboard eMMC memory, however it is easy to change configuration settings to access SD card. The following table identifies the target versions for the associated software and hardware used by the Application Project.

Software and Hardware Resources Used by the Application Project

Resource	Revision	Description
e^2 studio	5.3.1.002	Integrated Solution Development Environment
IAR Workbench	7.71.1.11989	IAR Workbench IDE for Synergy platform
SSP	1.2.0	Synergy Software Platform
SSC	5.3.1.002	Synergy Standalone Configurator
DK-S7G2	3.0	Development Kit

A simple flow diagram of the Application project is given in the following figure, below.

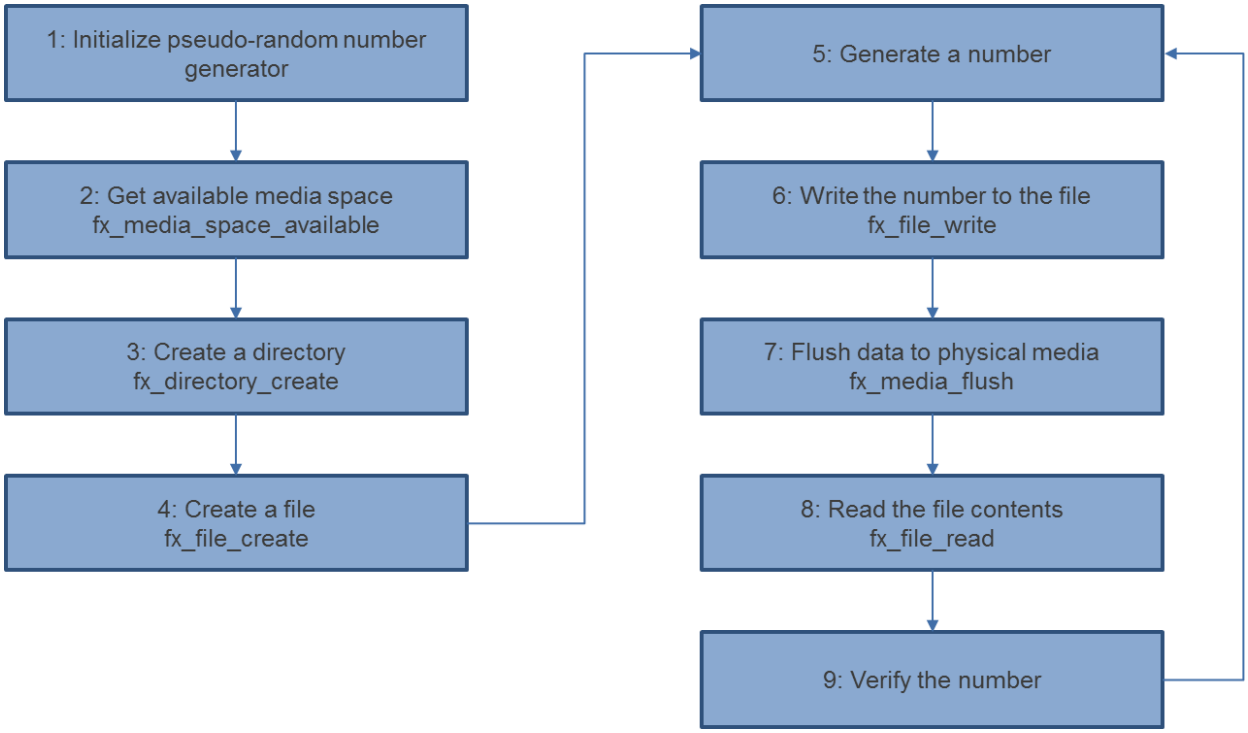


Figure 404: FileX Port Block Media Application Project Flow Diagram

NOTE: FileX initialization is done automatically in the generated code.

The complete Application Project can be found using the link provided in the Reference Section at the end of this document. The sdmmc_thread_entry.c file is located in the project once it has been imported into the ISDE. You can open this file, within the ISDE, and follow along with the below description, to help identify key uses of APIs.

The first section of sdmmc_thread_entry.c has the header files which references the FileX Media structure and a code section which allows semi-hosting to display results using printf(). The next section contains macro constants and variable definitions. Afterwards there are function prototypes. The first function generates a pseudo-random number. Then two functions for two-way conversion between integers and strings are defined. There is also simple code section for error handling. If the semi-hosting is enabled, it displays the error code. The next function is used for writing a number to a file. At first, the number is converted into a string. Then it accesses FileX APIs in order to open, clear, write, close the file and flush data to physical media. The last step is necessary because FileX operations are buffered and after reset, not all recent changes might be reflected. Alternatively, FileX flushes data when fx_media_close is called. The following function is very similar to the previous one, except it is used for reading a number from given file instead of performing a write.

The last section is the thread entry function section. At the beginning, pseudo-random number generator is initialized using predefined number. If the semi-hosting is enabled, available space is displayed. Then the application needs to create a file, but beforehand a directory is created. Then the software enters the 'forever' while loop, it starts with generating a number. This number is then written to the previously created file. Afterwards the number is read and verified with the generated one. In case of error the message is provided. Then a thread sleep function pauses execution for several ThreadX timer ticks and then the while loop functions are repeated.

NOTE: The above description assumes you are familiar with using printf() the Debug Console in the Synergy Software Package. If you are unfamiliar with this, refer to the "How do I Use Printf() with the Debug Console in the Synergy Software Package" given in the Reference Section at the end of this document. Alternatively, the user can see results via

the watch variables in the debug mode.

A few key properties are configured in this Application Project to support the required operations and the physical properties of the target board and MCU. Below are the properties with the values set for this specific project. You can also open the Application Project and view these settings in the property window as a hands-on exercise.

FileX Port Block Media Framework Configuration Settings for the Application Project

ISDE Property	Value Set
Name	g_fx_media
Format System is on SDMMC	True
Volume Name	Volume 1
Number of FATs	1
Directory Entries	256
Hidden Sectors	0
Total Sectors	500000
Bytes per Sector	512
Sectors per Cluster	1
Working media memory size	512

SD/MMC Driver on r_sdmmc Configuration Settings for the Application Project

ISDE Property	Value Set
Parameter Checking Enable	Default (BSP)
Name	g_sdmmc
Channel	0
Media Type	Embedded
Bus Width	8 Bits
Block Size	512
Callback	NULL

ISDE Property	Value Set
Access Interrupt Priority	Priority 8 (CM4: valid, CM0+: invalid)
Card Interrupt Priority	Disabled
DMA Request Interrupt Priority	Disabled

Transfer Driver on r_dmac Configuration Settings for the Application Project

ISDE Property	Value Set
Parameter Checking Enable	Default (BSP)
Name	g_sdmmc
Channel	0
Mode	Normal
Transfer Size	1 Byte
Destination Address Mode	Fixed
Source Address Mode	Incremented
Repeat Area (Unused in Normal Mode)	Source
Destination Pointer	NULL
Source Pointer	NULL
Number of Transfers	0
Number of Blocks (Valid only in Block Mode)	0
Activation Source	Software Activation
Auto Enable	False
Callback	NULL
Interrupt Priority	Priority 8 (CM4: valid, CM0+: invalid)

4.2.41 Segment LCD Driver

The Segment LCD Controller HAL module provides high-level APIs for Segment LCD applications and is implemented on r_slcdc. The Segment LCD Controller HAL module displays data on a Segment LCD and modifies the displayed data. The Segment LCD Controller HAL module uses the Segment LCD Controller module on a Synergy MCU.

4.2.41.1 SLCDC HAL Module Features

The SLCDC HAL module uses the Segment LCD Controller (SLCDC) to display data on a Segment LCD. The driver initializes the LCD for displaying data and configures the drive-voltage generator, the display waveform, the number of time slices, and the bias methods to drive the LCD. This module provides functions to display data to a specified set of segments, to update existing segment data, to enable and disable display, to set the display area, and to adjust the contrast.

This module supports the following features:

- Internal voltage-boosting for the LCD driver voltage generator: select the capacitor split method or the external resistance division.
- Display bias: select the 1/2 bias method, 1/3 bias method, or 1/4 bias method.
- Time slice of the display: select static, 2-time slice, 3-time slice, 4-time slice, or 8-time slice.
- Display waveform: select waveform A or waveform B.
- Display data area: select A-pattern, B-pattern, or blinking. You can switch the display data area.
- Use the RTC periodic interrupt (PRD) to generate a blinking display with A-pattern and B-pattern.
- Adjust the reference voltage (which is generated when operating the voltage boost circuit) in 16 steps (contrast adjustment).

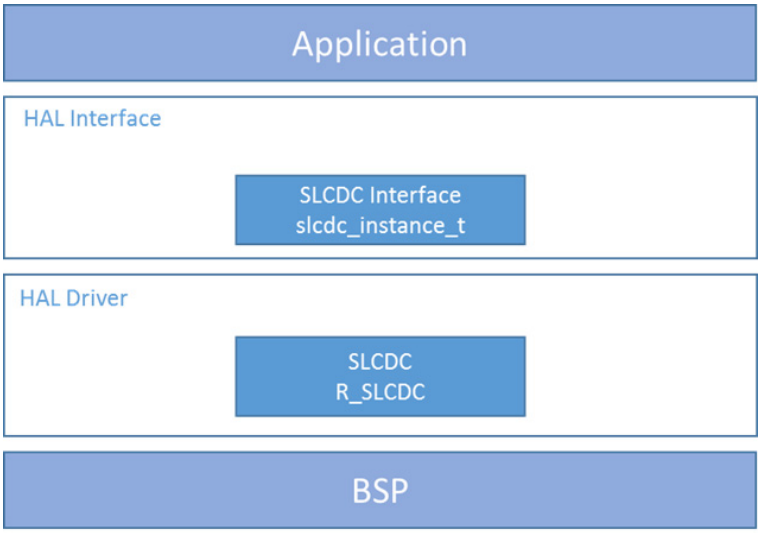


Figure 405: SLCD HAL Module Block Diagram

4.2.41.2 SLCDC HAL Module APIs Overview

The Segment LCD Controller HAL module defines APIs for functions such as opening, writing, starting, modifying, and closing. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows.

SLCDC HAL Module API Summary

Function Name	Example API Call and Description
open	g_slcdc.p_api->open(g_slcdc.p_ctrl, g_slcdc.p_cfg); Open SLCD device.
write	g_slcdc.p_api->write(g_slcdc.p_ctrl, start_segment, &data, segment_count); Write data to SLCD segments. Specifies the initial display data. The data parameter is a pointer to an array of bytes consisting at least segment_count items, in which each byte is associated with one segment data register. When the number of time slices is static, 2, 3 or 4, the lower 4 bits of the data become an A-pattern area and the upper 4 bits become a B-pattern area. See .setdisplayArea for setting a display area.
modify	g_slcdc.p_api->modify(g_slcdc.p_ctrl, segment, data_mask, data); Rewrite data in the SLCD segment. Rewrites the LCD display data in 1-bit units. If a bit is not specified for rewriting, the value stored in the bit is held as it is. Specifies the data to rewrite.
start	g_slcdc.p_api->start(g_slcdc.p_ctrl); Enable display on the SLCD. Displays the specified data on the LCD. Before that data should be written to the segments.

Function Name	Example API Call and Description
stop	g_slcdc.p_api->stop(g_slcdc.p_ctrl); Disable display on the SLCD. Stops displaying data on the SLCD.
contrastIncrease	g_slcdc.p_api->contrastIncrease(g_slcdc.p_ctrl); Increase the display contrast. Increase by 1 unit. This function can be selected when the internal voltage boosting method is used for the drive voltage generator.
contrastDecrease	g_slcdc.p_api->contrastDecrease(g_slcdc.p_ctrl); Decrease the display contrast. Decrease by 1 unit. This function can be selected when the internal voltage boosting method is used for the drive voltage generator.
setdisplayArea	g_slcdc.p_api->setdisplayArea(g_slcdc.p_ctrl, display_area); Set LCD display area. This function sets a specified display area, A-pattern or B-pattern. This function can be used to set blink on where A-pattern and B-pattern area data will be alternately displayed. When using blinking, the RTC is required to operate before this function is executed. To configure the RTC, follow the steps below. 1) Open RTC 2) Set Periodic interrupt request, ½ second 3) Start RTC counter 4) Enable IRQ, RTC_EVENT_PERIODIC_IRQ Refer to the User's Manual: Microcontrollers for the detailed procedure.
close	g_slcdc.p_api->close(g_slcdc.p_ctrl); Close display device.

Function Name	Example API Call and Description
versionGet	g_slcdc.p_api->versionGet(&version); Retrieve the API version using the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successful
SSP_ERR_ASSERTION	Assertion error
SSP_ERR_INVALID_ARGUMENT	Invalid Argument
SSP_ERR_HW_LOCKED	SLCDC resource is locked
SSP_ERR_NOT_OPEN	Device is not opened or initialized
SSP_ERR_UNSUPPORTED	Unsupported operation
SSP_ERR_NOT_ENABLED	RTC not enabled for blink operation

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.41.3 SLCDC HAL Module Operational Overview

This module uses the Segment LCD controller (SLCDC) to display data on a Segment LCD. The driver initializes the LCD for displaying data and configures the drive-voltage generator, the display waveform, the number of time slices, and the bias methods to drive the LCD. This module provides functions to display data to a specified set of segments, to modify existing segment data, to enable and disable display, to set the display area, and to adjust the contrast. The information displayed on the LCD can be modified by changing the contents of the LCD display data registers.

SLCDC HAL Module Important Operational Notes and Limitations

- This driver is a HAL driver and has no dependencies with the ThreadX RTOS. You can add the Segment LCD HAL module to a thread in the ThreadX RTOS if it is desirable.

- To write to a sequence of segments, give start segment number and number of segments to be written in the write API.
- There are no known limitations in the SLCDC HAL module.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.41.4 Including the SLCDC HAL Module in an Application

This section describes how to include the Segment LCD Controller HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the *Getting Started Guide for SSP* listed in the References section at the end of this document to learn how to manage each of these important steps in creating SSP-based applications.

To add the Segment LCD Controller HAL Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Segment LCD HAL module is g_slcdc0. This name can be changed in the associated **Properties** window.)

SLCD HAL Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_slcdc0 Segment LCD Driver on r_slcdc	**Threads**	**New Stack> Driver> Graphics> Segment LCD Driver on r_slcdc**

When the Segment LCD Controller HAL Module on r_slcdc is added to the Thread Stack as shown in the figure below, the configurator automatically adds the needed lower-level modules. Any drivers that need additional configuration information will be highlighted in Red. Modules with a Gray band are individual modules that stand alone.

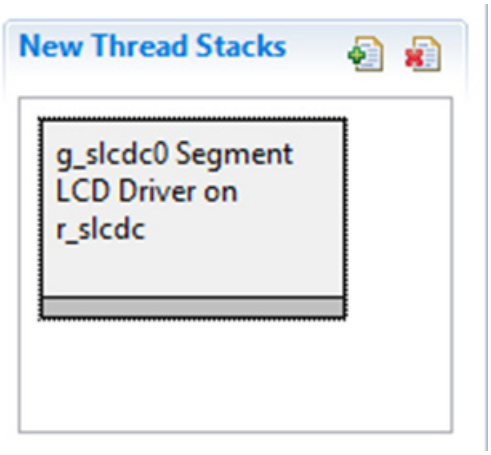


Figure 406: SLCD HAL Module Stack

4.2.41.5 Configuring the SLCDC HAL Module

The SLCD Controller HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify *by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the **Properties** window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the **Properties** tab within the SSP Configurator, and are shown in the following tables for easy reference.

NOTE: You may want to open your ISDE and create the Segment LCD Controller HAL module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for the SLCDC HAL Module on r_slcdc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Select if extra code will be added to check parameter values
Name	g_slcdc0	Module Name
Slcdc Clock	Clock LOCO, Clock SOSC, Clock MOSC, Clock HOCO (Default: Clock HOCO)	SLCD clock source (LCDSCKSEL)
Slcdc Clock Divisor	Clk Divisor LOCO 4/8/16/32/64/128/256/512/1,024 Clk Divisor HOCO 256/1,024/2,048/4,096/8,192/16,384/32,768/65,536/13,1072/262,144/524,288 (Default: Clk Divisor Hoco 16,384)	LCD clock setting (LCDC0), clock divisor
Bias Method	Bias 2, Bias 3, Bias 4 (Default: Bias 2)	LCD display bias method select (LBAS bit)

ISDE Property	Value	Description
Time Slice	Static, Slice 2, Slice 3, Slice 4 (Default: Static)	Time slice of LCD display select (LDTY bit)
Wave Form	Wave A, Wave B (Default: Wave A)	LCD display waveform select (LWAVE bit)
Slcdc Drive Voltage Generator	External resistance division, Internal voltage boosting, Capacitor Split (Default: External resistance division)	LCD Drive Voltage Generator Select (MDSTET bit)

NOTE: The example settings and defaults are for a project using the Synergy S3A7 MCU and the DK-S3A7 Kit. Other MCUs may have different default values and available configuration settings.

SLCDC HAL Module Clock Configuration

The SLCDC clock cannot be configured from the **Clocks** Tab. Configure the clock in the **Properties** window of the g_slcdc driver. The operating clock of the Segment LCD HAL Module is specified by the SLCDC Clock and SLCDC Clock Divisor settings in the **Properties** window. The Segment LCD HAL module-source clock can be configured as the main (MOSC), HOCO (high-speed clock oscillator), LOCO (Low-speed clock oscillator), or sub-clock (SOSC) using the ISDE configurator. For HOCO and LOCO settings, several clock divisors are available.

SLCDC HAL Module Pin Configuration

The SLCDC peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the pins.

NOTE: For some peripherals, the operation mode selection determines what peripheral signals are available and thus what MCU pins are required.

Pin Selection Sequence for the SLCDC HAL Module

Resource	ISDE Tab	Pin selection Sequence
SLCDC	Pins	Select **Peripherals > Graphics: SLCDC > SLCDC0**

NOTE: The selection sequence assumes SLCDC0 is the desired hardware target for the driver.

Pin Configuration Settings for the SLCDC HAL Module

Pin Configuration Property	Value	Description
Operation Mode	Disabled, Custom, Static, 2x Slice, 3x Slice, 4x Slice, 8x Slice (Default: Custom)	Select operation mode enable or disable
CAPH	None, P111 (Default: None)	Capacitor connection pin
CAPL	None, P112 (Default: P112)	Capacitor connection pin
COM0:3	None, Pn (Default: P104:107)	Common pins
COM4:7	None, Pn (Default: None)	Common pins
VL1:4	None, Pn (Default: P100:103)	Power supply pins
SEG00:02, SEG06:07, SEG16:17, SEG21:25, SEG46:51	None, Pn (Default: None)	Segment pins

Pin Configuration Property	Value	Description
SEG03	None, P303 (Default: P303)	Segment pin
SEG04:05	None, Pn (Default: P314:315)	Segment pins
SEG08	None, P902 (Default: P902)	Segment pin
SEG09:15	None, Pn (Default: P312:306)	Segment pins
SEG18:19	None, Pn (Default: P808:809)	Segment pins
SEG20	None, P313 (Default: P313)	Segment pin
SEG26:27	None, Pn (Default: P806:807)	Segment pins
SEG28:34	None, Pn (Default: P608:614)	Segment pins
SEG35:41	None, Pn (Default: P606:600)	Segment pins
SEG42:43	None, Pn (Default: P805:804)	Segment pins

Pin Configuration Property	Value	Description
SEG44:45	None, Pn (Default: P800:801)	Segment pins

NOTE: The example settings are for a project using the Synergy S3A7 MCU and the DK-S3A7 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.41.6 Using the SLCDC HAL Module in an Application

The typical steps in using the Segment LCD Controller HAL module in an application are:

1) Initialize the SLCDC HAL module using the open API
2) Write a sequence of segments using the write API
3) Change the display area or blinking display using the setdisplayArea API
4) Enable the display by using the start API
5) Adjust contrast using the contrastIncrease or contrastDecrease APIs
6) Disable the display by using the stop API
7) Close the driver using the close API

These common steps are illustrated in a typical operational flow diagram in the figure below:

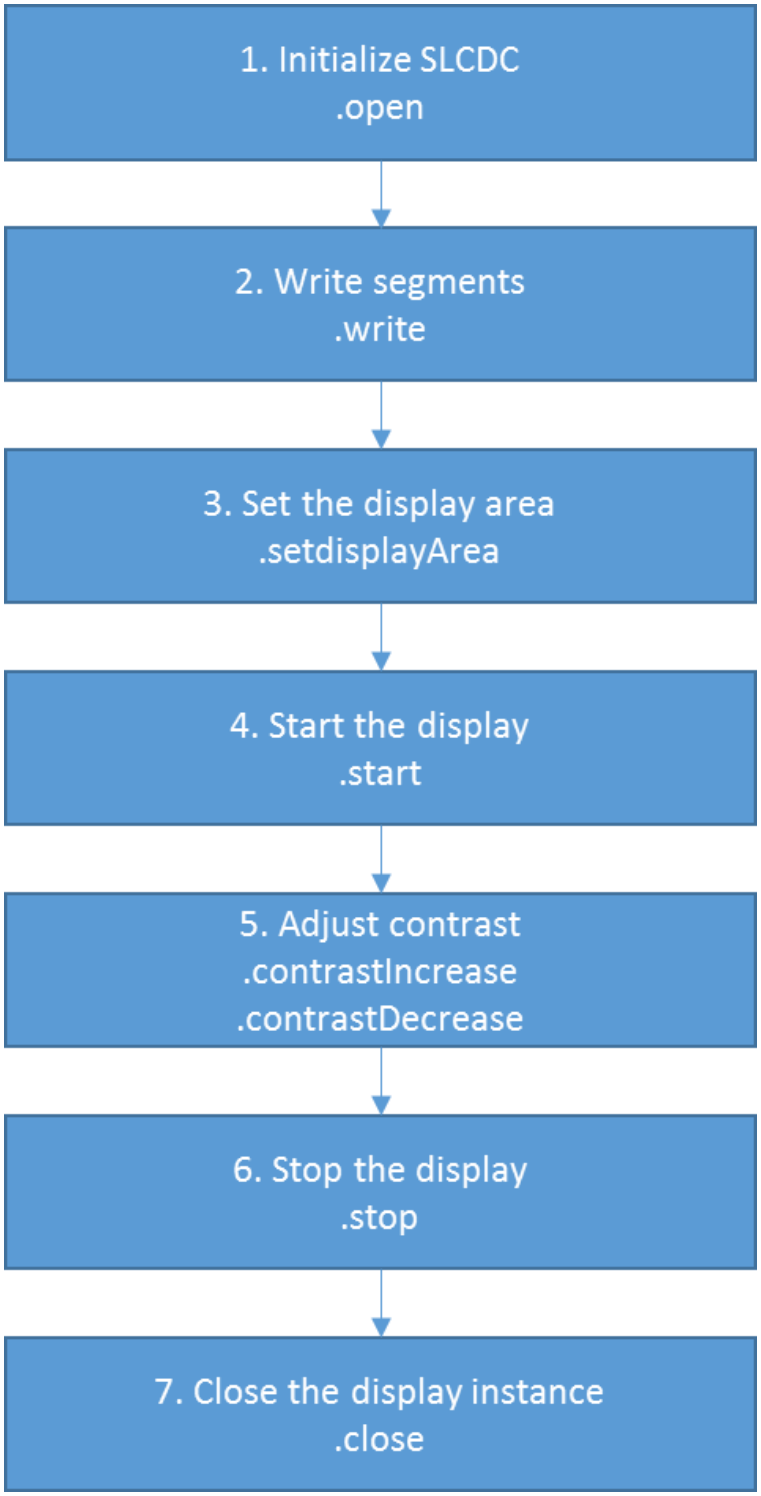


Figure 407: Flow Diagram of a Typical Segment LCDC HAL Module Application

4.2.42 SCI SPI Driver

The SCI SPI HAL module provides high-level APIs for master-based SPI serial communications and is implemented on the r_sci_spi (simple SPI) module. The SCI SPI HAL module configures and controls the SPI functionality of a Synergy MCU using the SCI (Serial Communications Interface) peripheral. The module is configured in the ISDE. A user-defined callback can be created to signal when the SPI has transmitted data, aborted a data transfer, or detected an error condition.

The SCI SPI HAL modules are enabled with a data transfer support by incorporating the DTC HAL module of the MCU. This performs SCI SPI transfer through DTC without intervention of the CPU.

4.2.42.1 SCI SPI HAL Module Features

The SCI SPI HAL module supports the configuration and control of the various SPI functionality on the Synergy MCU. Key features include the following:

- Driver initialization
- Serial communication through SPI operation using 8-bit data transfers
- Configurable among four clock phase and clock polarity settings
- Support for callbacks. The callback functions are called with the following events:
  - Transfer aborted
  - Transfer complete
  - Over run error
- SPI communication in master mode.

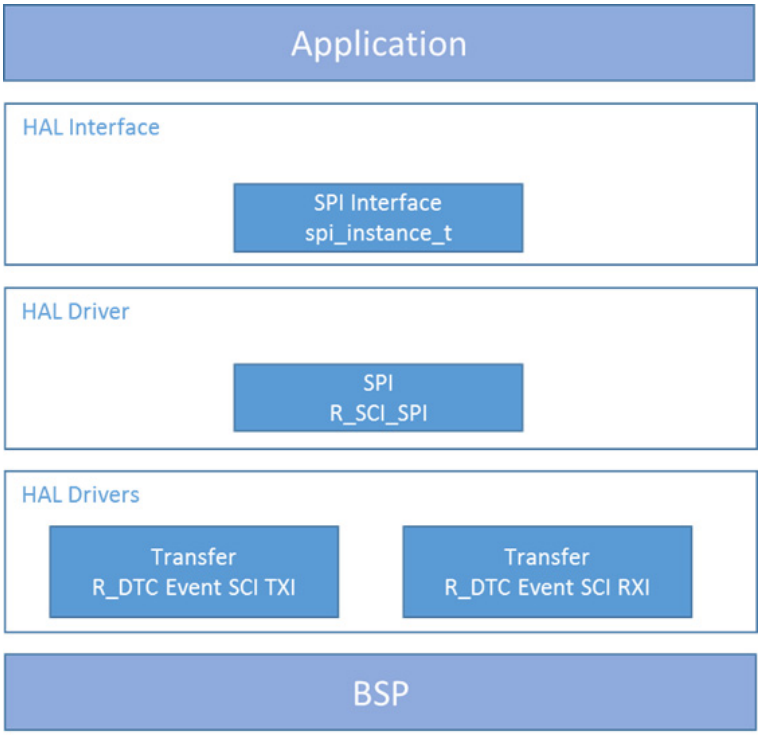


Figure 408: SCI SPI HAL Module Block Diagram

4.2.42.2 SCI SPI Module APIs Overview

The SPI defines APIs for opening and closing the SCI peripheral and transmitting and receiving data. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows.

SCI SPI HAL Module API Summary

Function Name	Example API Call and Description
open	g_spi.p_api->open(g_spi.p_ctrl, g_spi.p_cfg); Opens the SPI driver and initializes the hardware.
close	g_spi.p_api->close(g_spi.p_ctrl); Closes the driver and releases the SPI device.
read	g_spi.p_api->read(g_spi.p_ctrl, &read_buffer, number_of_bytes_to_read, bit_width); Performs a read operation on an SPI device.

Function Name	Example API Call and Description
write	g_spi.p_api->write(g_spi.p_ctrl, &write_buffer, number_of_bytes_to_write, bit_width); Performs a write operation on an SPI device.
writeRead	g_spi.p_api->writeRead(g_spi.p_ctrl, &read_buffer, &write_buffer, number_of_bytes_to_read_and_write, bit_width); Performs a read and write (full duplex) operations on an SPI device.
versionGet	g_spi.p_api->versionGet(&version); Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful.
SSP_ERR_IN_USE	Attempted to open an already open device instance OR Another transfer was in progress.
SSP_ERR_INVALID_POINTER	p_version is NULL.
SSP_INVALID_ARGUMENT	Channel number invalid.
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.
SSP_ERR_CH_NOT_OPEN	The channel has not been opened. Open channel first.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.42.3 SCI SPI HAL Module Operational Overview

The SCI SPI interface provides the ability to configure and use the SPI functionality of the Synergy MCU. The HAL module enables communication with a peripheral device using the SPI communications protocol. After opening the SCI SPI HAL module instance, the SCI SPI module handle is used to perform various transfer operations. The device control handle will be used within the API calls to indicate the specific SCI SPI device to communicate with.

The Driver allows the user to:

- Initialize the driver.
- Serial Communication through SPI operation. Read from and write to (and simultaneous read/write – full duplex) a SPI device – performed by calling the read, write, and writeRead APIs.

The Driver also provides support for callbacks. The callback functions are called with the following events:

- Transfer aborted
- Transfer complete
- Overrun error

The SCI SPI Modules support only 8-bit data transfer operations. SCI SPI Modules uses GPIO pins configured as chip selects.

Clock settings:

The SCI SPI uses PCLKA as its clock source. You can set the PCLKA frequency using the clock configurator in e^2 studio or the CGC Interface at run-time.

IO Port settings:

To use with the SPI, the I/O port pin(s) used as output pins must be configured as SCI SPI peripheral pins in the pin configurator. For external chip select, configure Chip select pin as GPIO output.

SCI SPI Interrupts:

To enable interrupts of SCI SPI, highlight the driver module and set the priority of the SCI RXI, TXI, TEI and ERI interrupts on the Threads tab of the Project Configurator in e^2 studio: Configuring Interrupts.

This sets the corresponding interrupts in ssp_cfg/bsp/bsp_irq_cfg.h to the priority level selected.

ATTENTION: Setting the interrupts to different priority levels could result in improper operation.

SCI SPI HAL Module Important Operational Notes and Limitations

- Chip select outputs are supported using GPIOs
- The SPI on SCI module uses only 8-bit data transfers.
- Setting the interrupts to different priority levels could result in improper operation.
- The SCI SPI HAL module is enabled with a data transfer support by incorporating the Data Transfer Controller module of the MCU. This performs SPI transfers through the DTC without intervention of the CPU. The DTC transfer is enabled by default, the user has to remove it from the configurator for an IRQ mode transfer.
- When implementing SPI on SCI, only the master mode is supported.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.42.4 Including the SCI SPI HAL Module in an Application

This section describes how to include the SPI HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack.

To add the SPI Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for a SPI is g_spi0. This name can be changed in the associated Properties window.)

SPI Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_spi0 SPI Driver on g_sci_spi	Threads	New Stack> Driver> Connectivity> SPI Driver on g_sci_spi

When the SPI HAL module is added to the thread stack as shown in the following figure, the configurator automatically adds the needed lower-level drivers. Any drivers that need additional configuration information will be highlighted in Red. Modules with a Gray band are individual modules that stand alone.

The following figure shows the SPI on SCI drivers added to the HAL/Common Thread. The resource name is g_spi0. The SPI HAL module can support a transfer driver, whereas the data-transfer controller can be used to transmit and receive data between the hardware peripherals and data buffers in RAM (rather than the CPU performing the transfers.) By default, a transfer driver is added for both transmit and receive.

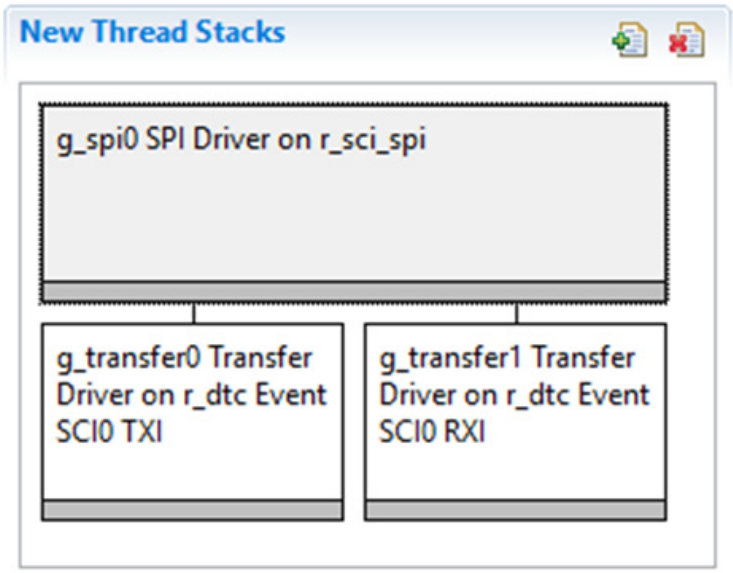


Figure 409: SCI SPI HAL Module Stack

4.2.42.5 Configuring the SCI SPI HAL Module

The SPI HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in Red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator, and are shown in the following tables for easy reference.

NOTE: You may want to open your ISDE and create the SPI and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the SCI SPI HAL Module on r_sci_spi

ISDE Property	Value	Description
Parameter Checking	Default (BSP), Enabled, Disabled	Enable or disable the parameter error checking.
Name	g_spi0	Instance name of driver. Can be renamed by user.
Channel	0	SCI channel number to be used for SPI communication.
Operating Mode	Master	How the SPI driver operates – master or slave (not supported).
Clock Phase	Data sampling on odd edge, data variation on even edge Data sampling on even edge, data variation on odd edge (Default: Data sampling on odd edge, data variation on even edge)	How the clock signal is working in terms of clock signal edges and data operations.
Clock Polarity	Low when idle, High when idle (Default: Low when idle)	High output when no operations are performed. Alternative: low output when no operations are performed.
Mode Fault Error	Disable, Enable (Default: Disable)	How errors are handled.
Bit Order	MSB First, LSB First (Default: MSB First)	Bit order within transmitted/received bytes.
Bitrate	100000	Transmission speed in bits per second.
Bit Rate Modulation Enable	Enable, Disable (Default: Disable)	The bit rate can be evenly corrected using the MDDR register. This will help to reduce error on baud rates in data communications. Beneficial for systems with none standard crystal frequencies and for lower baud rate communications.

ISDE Property	Value	Description
Callback	NULL	A user callback function can be registered in open. If this callback function is provided, it will be called from the interrupt service routine (ISR) for each of the conditions defined in spi_event_t.
Receive Interrupt Priority	Priority 0 > 15 (Default: Priority 2)	The SPI driver is interrupt driven. MCU interrupt priority level.
Transmit Interrupt Priority	Priority 0 > 15 (Default: Priority 2)	The SPI driver is interrupt driven. MCU interrupt priority level.
Transmit End Interrupt Priority	Priority 0 > 15 (Default: Priority 2)	The SPI driver is interrupt driven. MCU interrupt priority level.
Error Interrupt Priority	Priority 0 > 15 (Default: Priority 2)	The SPI driver is interrupt driven. MCU interrupt priority level.

NOTE: The example settings and defaults are for a project using the Synergy S7 MCU Family. Other MCUs may have different default values and available configuration settings.

In many cases, settings other than the defaults for the SPI Driver will be required to communicate via the SPI. For example, changes to the bit rate, clock polarity, and bit order may differ between slave devices.

SCI SPI HAL Module Clock Configuration

The SCI peripheral is clocked via the Peripheral Clock A (PCLKA.) The clock frequencies are configurable in the ISDE by using the Clocks tab in the configurator. Invalid selections are indicated in red when selected. Ensure that desired SPI bitrate can be achieved with the stated value of PCLKA. The ISDE will not be indicated if the specified bitrate is not achievable. At run time, the SPI HAL module will attempt to configure the SCI peripherals to the correct bitrate and will return an error if the desired bitrate cannot be set. The bitrate is calculated via the equations in the following table. If the result of the equation (N) is in the range of 0 to 255, then the bit rate can be achieved.

Baud Rate Calculation Equations

SPI HAL	Bitrate calculation	Description
SPI on SCI	$N = \frac{PCLKA\ (MHz)}{8 * 2^{(2n-1)} * \left(\frac{256}{M}\right) * B} - 1$ **Figure 410: N = Peripheral register value. This must be in the range of 0 to 255 PLCKA = value of PLCKA in MHz n = 0, 1, 2 or 3 M = Bit Rate Modulation Index 128 < M < 256 If the Bit Rate Modulation is disabled, then M=256 B = Desired Bit Rate**	

SCI SPI HAL Module Pin Configuration

The SCI peripheral use pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the SPI pins.

Pin Selection Sequence for SCI SPI HAL Driver

Resource	ISDE Tab	Pin selection Sequence
SPI on SCI	Pins	Select Peripherals > Connectivity:SCI > SCIx Where x is the required SCI peripheral channel.

Pin Configuration Settings for SPI on SCI – Example Case of using SCI0

Pin Configuration Property	Value	Description
Pin Group Selection	Mixed, _A only, _B only	Synergy devices support peripheral functionality via multiple pins location, identified by _A, _B. Selecting Mixed allows the user to select any combination of locations (_A and _B). Selecting _A allows the user to select only _A locations. Selecting _B allows the user to select only _B locations.
Operation Mode	Disabled Custom Asynchronous UART Simple SPI Simple I2C Synchronous UART Smart Card	Set the operating mode to: Simple SPI.
TXD_MOSI	None, P411, P101	Specify the port pin to be used as MOSI.
RXD_MISO	None, P410, P100	Specify the port pin to be used as MISO.
SCK	None, P412, P102	Specify the port pin to be used as CLK.
CTS_RTS_SS	None, P413, P103	Specify the port pin to be used as SS.

NOTE: The example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.42.6 Using the SCI SPI HAL Module in an Application

A sample procedure for using the SCI SPI in an application is as follows.

The g_spi.p_api->open() function must be called first. The rest of the calls may be used in any order depending on the application requirements:

1) Open a SPI instance with the SPI implemented by SCI SPI. The SCI SPI driver is called through the SPI Interface

```
g_spi.p_api->open (g_spi.p_ctrl, g_spi.p_cfg)
```

where p_ctrl and p_cfg are the instances of control and configuration structures autogenerated after the configuration step.

2) Initiate a write to a slave device by calling

```
g_spi.p_api->write (g_spi.p_ctrl, source, length,
SPI_BIT_WIDTH_8_BITS);
```

where g_spi.p_ctrl is the same control instance that was used in the open call.

3) Initiate a read from a slave device by calling

```
g_spi.p_api->read(g_spi.p_ctrl, dst, length,
SPI_BIT_WIDTH_8_BITS);
```

where g_spi.p_ctrl is the same control instance that was used in the open call.

4) Initiate a simultaneous transfer from and to a slave device by calling

```
g_spi.p_api->writeRead(g_spi.p_ctrl, source,
   dst, length, SPI_BIT_WIDTH_8_BITS);
```

where g_spi.p_ctrl is the same control instance that was used in the open call.

5) To close the SPI channel, do so by calling

```
g_spi.p_api->close(g_spi,p_ctrl)
```

where g_spi.p_ctrl is the same control structure that was used in the open call.

The following flow diagram illustrates the previously described common steps:

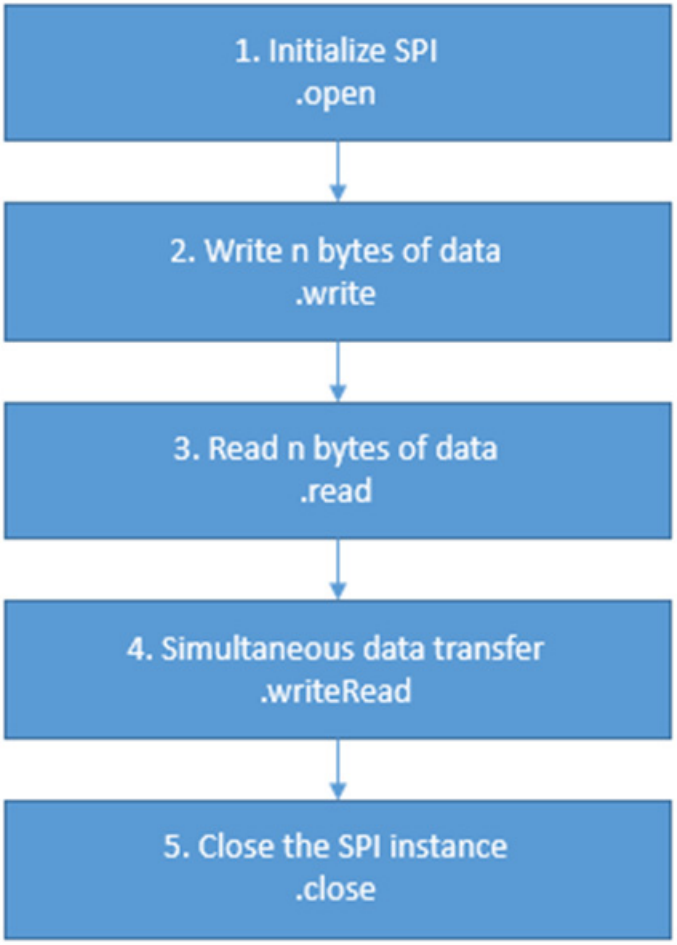


Figure 411: Flow Diagram of a Typical SPI Application

4.2.43 SPI Driver

The SPI HAL module provides high-level APIs for communication using the SPI protocol. The module supports both the SPI and SCI peripherals available on the Synergy microcontroller hardware, and is implemented on r_rspi and r_sci_spi. This section refers to the r_rspi HAL module, which is referred as the SPI module (formerly known as RSPI). The SPI HAL module supports standard SPI master and slave mode communications functions. Callbacks are provided for transfer events.

The SPI HAL modules are enabled with data-transfer support by incorporating the Data Transfer Controller (DTC) module of the MCU. This performs SPI transfers through the DTC without requiring interrupt processing, by the CPU, for each frame.

4.2.43.1 RSPI HAL Module Features

The SPI HAL module support following key features:

- Initialization of the driver
- SPI transfer functions:
  - Allows serial communication through the SPI operation using the four-wire method
  - Capable of serial communication in master and slave modes
  - Switching the polarity of the serial transfer clock
  - Switching the phase of the serial transfer clock
- Data Format
  - MSB-first/LSB-first selectable
  - Transfer bit length is selectable as 8, 16 and 32 bits
- Error Detection
  - Mode fault detection
  - Overrun error detection
  - Parity error detection
- SSL control functions
  - Internally select up to four SSL signals (SSLn0 to SSLn3) for each channel in master mode
  - External hardware slave select can be used in master mode
- Interrupts
  - RSPI receive interrupt (receive buffer full)
  - RSPI transmit interrupt (transmit buffer empty)
  - RSPI error interrupt (mode fault, overrun and parity error)
- Delays
  - Add SPI clock delay
  - Add slave select negation delay
  - Add next-access delay

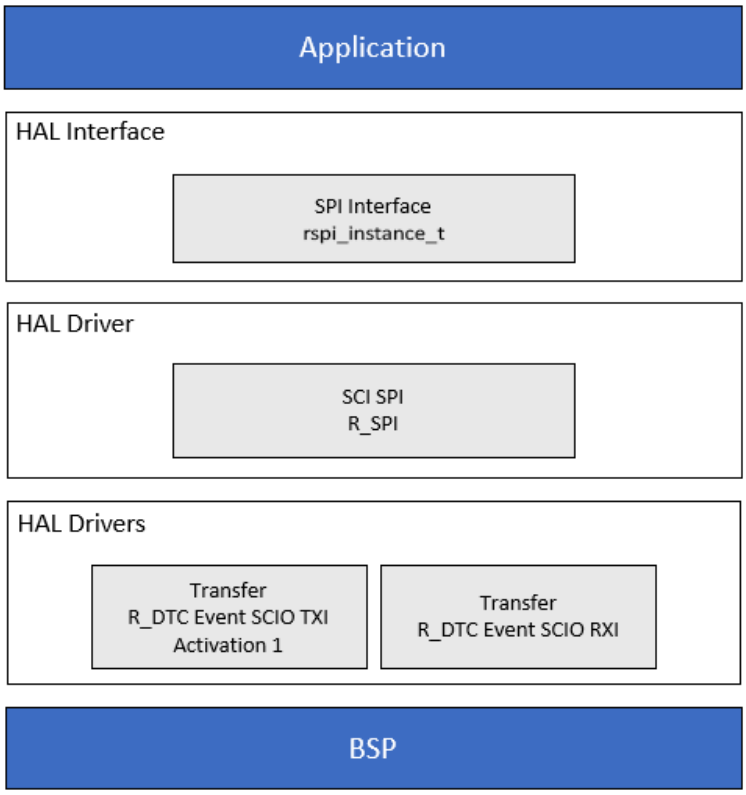


Figure 412: RSPI HAL Module Block Diagram

4.2.43.2 RSPI HAL Module APIs Overview

The RSPI HAL module defines APIs for opening, closing, reading, writing and other useful functions. A complete list of the available APIs, an example API call, and a short description of each can be found in the follow table. A subsequent table of status return values is listed afterwards.

RSPI HAL Module API Summary

Function Name	Example API Call and Description
open	g_spi.p_api ->open(g_spi.p_cntl, g_spi.p_cfg); Open a designated SPI device.
read	g_spi.p_api->read(g_spi.p_ctrl, dst16, length, SPI_BIT_WIDTH_16_BITS); Receive data from SPI device.

Function Name	Example API Call and Description
write	g_spi.p_api->write (g_spi.p_ctrl, source, length, SPI_BIT_WIDTH_8_BITS); Transmit data to SPI device
writeRead	g_spi.p_api ->writeRead (g_spi.p_cntl, &source, &destination, length, SPI_BIT_WIDTH_8_BITS, TX_WAIT_FOREVER); Simultaneously transmits data to an SPI device, while receiving data from an SPI device (full duplex). The writeRead API fetches the mutex object, handles SPI data transmission at SPI HAL layer, and receives data from the SPI HAL layer. The API uses the event flag wait to synchronize to complete the data transfer.
close	g_spi.p_api->close(g_spi,p_ctrl) Disable the SPI device designated by the control handle and close the RTOS services used by the bus if no devices are connected to the bus. This function removes power to the SPI channel designated by the handle and disables the associated interrupts.
versionGet	g_spi.p_api ->versionGet (&version); Get the version information of the underlying driver.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function completed successfully
SSP_ERR_INVALID_MODE	Invalid mode
SSP_ERR_INVALID_CHANNEL	Invalid channel
SSP_ERR_IN_USE	In-use error

Name	Description
SSP_ERR_INVALID_ARGUMENT	Invalid argument
SSP_ERR_QUEUE_UNAVAILABLE	Queue unavailable
SSP_ERR_INVALID_POINTER	Invalid pointer
SSP_ERR_INTERNAL	Internal error
SSP_ERR_TRANSFER_ABORTED	Transfer aborted
SSP_ERR_MODE_FAULT	Mode fault
SSP_ERR_READ_OVF	Read overflow
SSP_ERR_PARITY	Parity error
SSP_ERR_OVERRUN	Overrun error
SSP_ERR_UNDEF	Unknown error
SSP_ERR_TIMEOUT	Timeout error
SSP_ERR_NOT_OPEN	Device not opened

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.43.3 RSPI HAL Module Operational Overview

The SPI HAL module enables communication with a peripheral device using the SPI communications protocol. After opening the SPI HAL module instance, the SPI module handle is used to perform various transfer operations. The device control handle will be used within the API calls to indicate the specific SPI device to communicate with.

The Driver allows the user to:

- Initialize the driver.
- Serial Communication through SPI operation.

The Driver also provides support for callbacks. The callback functions are called with the following events- spi_event_t:

- Transfer aborted
- Transfer complete
- Mode fault
- Error events

The SPI module supports 8, 16, and 32 bit data transfers. The SPI Module supports GPIO pins configured as chip selects. In addition, the SPI peripheral supports dedicated chip select signals SSLn0 to SSLn3. When SSL pins are enabled in the SPI peripheral, all chip select handling is performed by the hardware.

Clock settings:

The SPI uses PCLKA as its clock source. You can set the PCLKA frequency using the clock configurator in e^2 studio or the CGC_API at run-time.

NOTE: For S1 devices the SPI clock source is PCLKB.

IO Port settings:

To use with the SPI, the I/O port pin(s) used as output pins must be configured as SPI peripheral pins in the pin configurator. If you are using an external chip select, configure Chip select pin as GPIO output.

Extended configuration:

A number of extended hardware specific configurations are present for SPI Driver.

NOTE: All parameters are set in the SPI extended driver configuration structure spi_on_rspi_cfg_t.

RSPI HAL Module Important Operational Notes and Limitations

While configuring SPI HAL drivers, setting the interrupts to different priority levels could result in improper operation.

The SPI HAL driver module is enabled with a data transfer support by incorporating the Data Transfer Controller module of the MCU. This performs SPI transfer through DTC without intervention of the CPU.

In the application, data transfer over DTC is used in the same way as for normal SPI transfer. To enable DTC transfer, add the DTC module under the SPI HAL module.

SPI module supports 8, 16 and 32-bit data transfer in both CPU and DTC based transfer modes. Renesas recommends using the DTC with SPI to reduce CPU interrupt processing time, and configuring the SPI peripheral for 8-bit transfers to ensure data is transferred with the expected endianness.

Important Note: 16 and 32 bit transfers will be endian swapped.

SPI HAL Module Performance Notes

The R_RSPI module can be configured for several different modes each with different performance characteristics. DTC transfers take slightly longer to setup than CPU based transfers due to resetting the DTC, but DTC transfers offer greater performance for transfers larger than 1 frame because no intervention is required from the CPU.

Write operations will configure the module for transmit only mode, disabling the receive interrupt and ignoring incoming data. CPU based write operations at high bitrates can result in the transmit ISR being constantly called, blocking other code from running. WriteRead and Read operations will configure the module for full duplex mode.

There is a lower limit of 3 SPI clock cycles between transfers resulting in an effective bitrate slower than configured. At high bitrates, especially for CPU based transfers, the time between transfers can be longer.

The module will wait for the hardware to enter an idle state when in master mode, or all data to be transmitted or received when in slave mode, before invoking the callback.

- When using R_RSPI in slave mode either the data must be sampled on the even clock edge or the master must de-assert the slave select line between frames. This is a hardware limitation.
- S124, S128 and S3A6 do not support SSL Level Keep.
- Once the driver has been opened using DTC, all transfers must use the transfer width the DTC is configured for.
- 16- and 32-bit transfers will be endian swapped.

- The R_RSPI bit rate value must be positive integer less than 30 MHz or PCLK/2, whichever is minimum.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.43.4 Including the RSPI HAL Module in an Application

This section describes how to include the RSPI HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the "Getting Started Guide for SSP" listed in the Reference Section at the end of this document to learn how to manage each of these important steps in creating SSP based applications.

To add the RSPI HAL Driver to an application, simply add it to a thread using the Stacks Selection Sequence given in the following table. (The default name for the SPI Framework is g_spi0. This name can be changed in the associated Properties window.)

RSPI HAL Driver Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_spi0 SPI Driver on r_rspi	Threads	New Stack> Driver> Connectivity> SPI Driver on r_rspi

The following figure shows when the RSPI HAL module on r_rspi is added to the Thread Stack, the configurator automatically adds the needed lower level drivers. Any drivers that need additional configuration information are box text highlighted in red. Modules with a gray band are individual modules that stand alone. Modules with a blue band are shared or common and need only be added once, since they can be used by multiple stacks. Modules with a pink band can require the selection of lower level drivers. Sometimes these are optional or recommended and this is indicated in the block with the inclusion of this text. If the addition of lower level drivers is required, the module description includes "Add" in the text. Clicking on any pink banded modules brings up the "New" icon and shows possible choices.

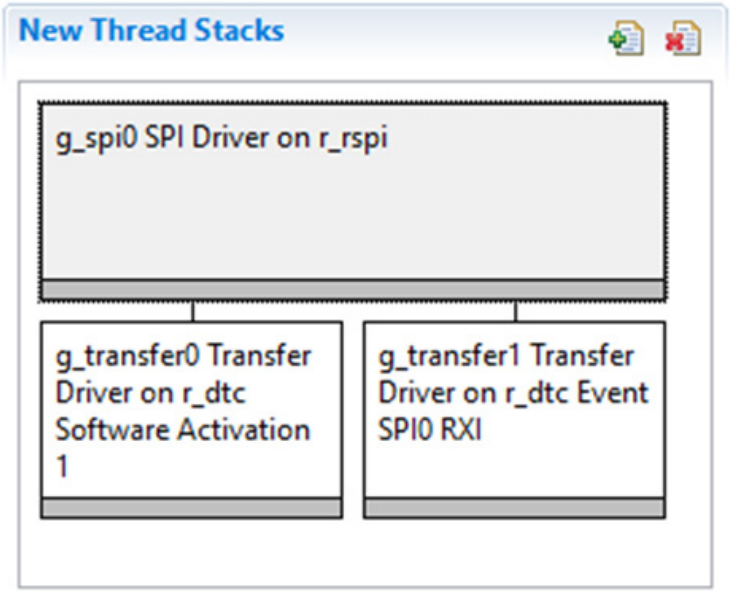


Figure 413: RSPI HAL Module Stack

4.2.43.5 Configuring the RSPI HAL Module

Configure RSPI HAL module on r_rspi for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any configurations (such as interrupts or operating modes) for lower level modules required for successful operation. Only properties that can be changed without causing conflicts are available for modification. Properties that cannot be modified are 'locked' with a lock icon displayed for the 'locked' property in the ISDE property window. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the properties window of the associated module. Simply select the indicated module and then view the properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+.) This level of detail is not included in the configuration properties tables below, but is easily visible with the ISDE when configuring interrupt priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the Configuration Table Settings given below. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the RSPI HAL Module on r_rspi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_spi0	Module name
Channel	0	SCI or SPI Channel number to which the device has been connected.
Operating Mode	Master, Slave Default: Master	Configure as a Master or Slave device.
Clock Phase	Data sampling on odd edge, data variation on even edge/Data sampling on even edge, data variation on odd edge Default: Data sampling on odd edge, data variation on even edge	Data sampling on odd or even clock edge.
Clock Polarity	Low when idle, High when idle Default: Low when idle	Clock level when idle.
Mode Fault Error	Enable, Disable Default: Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First, LSB First Default: MSB First	Select transmit order MSB/LSB first
Bitrate	500000	Transmission or reception rate. Bits per second.
Callback	NULL	Optional Callback function pointer.

ISDE Property	Value	Description
SPI Mode	SPI Operation, Clock synchronous operation Default: SPI Operation	Select SPI or clock sync mode operation.
SPI Communication Mode	Full Duplex, Transmit Only Default: Full Duplex	Select full-duplex or transmit-only communication.
Slave Select Polarity(SSL0)	Active Low, Active High Default: Active Low	Select SSL0 signal polarity
Slave Select Polarity(SSL1)	Active Low, Active High Default: Active Low	Select SSL1 signal polarity
Slave Select Polarity(SSL2)	Active Low, Active High Default: Active Low	Select SSL2 signal polarity
Slave Select Polarity(SSL3)	Active Low, Active High Default: Active Low	Select SSL3 signal polarity
Select Loopback1	Normal, Inverted Default: Normal	Select loopback1
Select Loopback2	Normal, Inverted Default: Normal	Select loopback2
Enable MOSI Idle State	Enable, Disable Default: Disable	Select MOSI idle fixed value and selection

ISDE Property	Value	Description
MOSI Idle State	MOSI Low, MOSI High Default: MOSI Low	Select MOSI idle fixed value and selection
Enable Parity	Enable, Disable Default: Disable	Enable/disable parity
Parity Mode	Parity Odd, Parity Even Default: Parity Odd	Select parity
Select SSL(Slave Select)	SSL0, SSL1, SSL2, SSL3 Default: SSL0	Select which slave to use; 0-SSL0; 1-SSL1; 2-SSL2; 3-SSL3
Select SSL Level After Transfer	SSL Level Keep, SSL Level Do Not Keep Default: SSL Level Do Not Keep	Select SSL level after transfer completion; 0-negate; 1-keep
Clock Delay Enable	Clock Delay Enable, Clock Delay Disable Default: Clock Delay Disable	Clock delay enable selection
Clock Delay Count	Clock Delay 1 thru 8 RSPCK Default: Clock Delay 1 RSPCK	Clock delay count selection
SSL Negation Delay Enable	Negation Delay Enable, Negation Delay Disable Default: Negation Delay Disable	SSL negation delay enable selection
Negation Delay Count	Negation Delay 1 thru 8 RSPCK Default: Negation Delay 1 RSPCK	Negation delay count selection

ISDE Property	Value	Description
Next Access Delay Enable	Next Access Delay Enable, Next Access Delay Disable Default: Next Access Delay Disable	Next access delay enable selection
Next Access Delay Count	Next Access Delay 1 thru 8 RSPCK Default: Next Access Delay 1 RSPCK	Next access delay count selection
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Receive interrupt priority selection
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Transmit interrupt priority selection

ISDE Property	Value	Description
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Priority 2	Error interrupt priority selection

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the DTC HAL Module on r_dtc Software Activation 1

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	4 Bytes	Transfer size selection

ISDE Property	Value	Description
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Software Activation 1	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

Configuration Settings for the DTC HAL Module on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Name	g_transfer1	Module name

ISDE Property	Value	Description
Mode	Normal	Mode selection
Transfer Size	4 Bytes	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SPI0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection.

RSPI HAL Module Clock Configuration

The SPI peripheral is clocked via Peripheral Clock A (PCLKA). The clock frequencies are configurable in the ISDE by using the Clocks Tab in the configurator. Invalid selections are indicated in red when selected. Ensure that desired SPI bitrate can be achieved with the stated value of PCLKA. The ISDE will not be indicated if the specified bitrate is not achievable. At run time, the SPI driver will attempt to configure the SPI peripheral to the correct bitrate and will return an error if the desired bitrate cannot be set. The bitrate is calculated via the equations in the table below. If the result of the equation (n) is in the range of 0 to 255, then the bit rate can be achieved.

Baud Rate Calculation Equations

SPI HAL	Bitrate calculation	Description
SPI on SPI	$n = \frac{PCLKA\ (MHz)}{2 * 2^N * B} - 1$	n = Peripheral register value. This has to be in the range of 0 to 255 PLCKA = value of PLCKA in MHz N = 0, 1, 2 or 3 B = Desired Bit Rate

RSPI HAL Module Pin Configuration

The SPI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example listing a selection for SPI pins.

Pin Selection Sequence for the RSPI HAL Module

Resource	ISDE Tab	Pin selection Sequence
RSPI	Pins	Select Peripherals > RSPI > SPI0_Pin_Option_A/B

NOTE: The top selection sequence assumes SPIO is the desired hardware target for the driver and the bottom selection sequence assumes SPI_0 is the desired target.

Pin Configuration Settings for the RSPI HAL Module

Property	Value	Description
Operation Mode	Disabled, Custom, Enabled (Default: Disabled)	Select Enabled for SPI Operation
MISO	None, P100, P410 (Default: None)	MISO Pin selection
MOSI	None, P101, P411 (Default: None)	MOSI Pin selection

Property	Value	Description
RSPCLK	None, P102, P412 (Default: None)	RSPCLK Pin selection
SSL0:3	None, P103:106, P413:415 (Default: None)	SSL0:3 Pin selections

NOTE: The example lists settings for a project using the Synergy S7G2 Series 7 Group MCU and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.43.6 Using the RSPI Module in an Application

To write a SPI application using the SPI, follow these steps. The g_spi.p_api->open() function must be called first. The rest of the calls may be used in any order depending on the application requirements:

1) Open an SPI instance with the SPI implemented by SPI. The SPI driver is called through the SPI Interface

```
g\_spi.p\_api-&gt;open
   (g\_spi.p\_ctrl, g\_spi.p\_cfg)
```

where p_ctrl and p_cfg are the instances of control and configuration structures autogenerated after the SPI configuration step.

2) Initiate a write to a slave device by calling

```
g\_spi.p\_api-&gt;write (g\_spi.p\_ctrl, source, length,
SPI\_BIT\_WIDTH\_8\_BITS);
```

where g_spi.p_ctrl is the same control instance that was used in the open call.

3) Initiate a read from a slave device by calling

```
g\_spi.p\_api-&gt;read(g\_spi.p\_ctrl, dst16, length,
SPI\_BIT\_WIDTH\_16\_BITS);
```

where g_spi.p_ctrl is the same control instance that was used in the open call.

4) Initiate a simultaneous transfer from and to a slave device by calling

```
g\_spi.p\_api-&gt;writeRead(g\_spi.p\_ctrl, source,
   dst16, length, SPI\_BIT\_WIDTH\_8\_BITS, TX\_WAIT\_FOREVER);
```

where g_spi.p_ctrl is the same control instance that was used in the open call.

5) To close the SPI channel, do so by calling

```
g\_spi.p\_api-&gt;close(g\_spi,p\_ctrl);
```

where g_spi.p_ctrl is the same control structure that was used in the open call.

The following figure illustrates these steps in a typical operational flow.

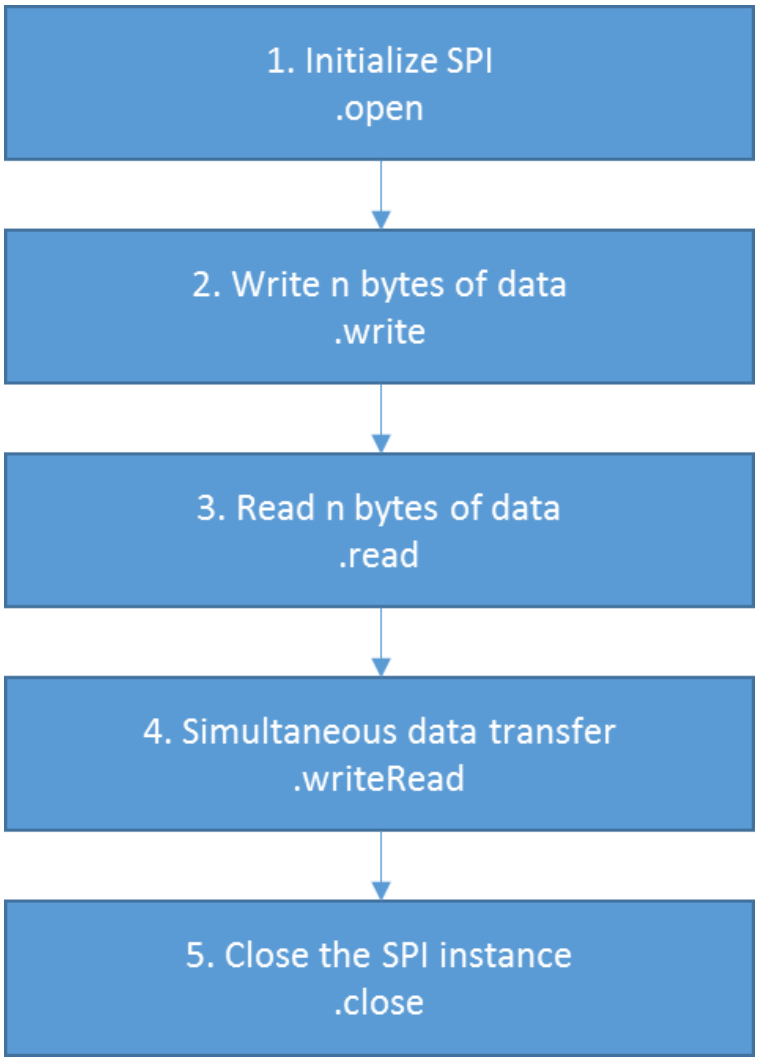


Figure 414: Flow Diagram of a Typical RSPI HAL Module Application

4.2.44 Transfer Driver

The Transfer Driver is a generic Driver for Transfer applications and supports the two Transfer peripherals available on the Synergy microcontroller hardware: DMAC and DTC. The Driver can be therefore implemented on r_dmac and r_dtc. This section describes how to configure the Transfer Driver using the e^2 studio ISDE and how to include the API functions in your application.

In the project configurator in the e^2 studio ISDE, you can add and configure a Transfer Driver Module in the Modules pane of the Threads tab by selecting **New > Driver > Transfer > Transfer Driver on r_dmac** or **New > Driver > Transfer > Transfer Driver on r_dtc**. For details, see: Using e^2 studio to write an application with the Transfer Driver

You can find the API reference in the description of the Transfer Interface here: Transfer Interface.

To understand how to program with Interfaces in the SSP, see: SSP Architecture

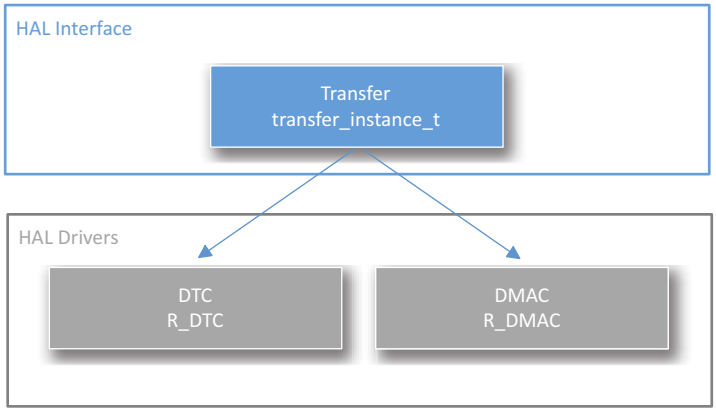


Figure 415: TransferDriver - Block diagram

4.2.44.1 What Does the Transfer Driver Do?

The Transfer Driver moves data from a user specified source to a user specified destination when an interrupt or event occurs.

4.2.44.2 Selecting a Module for the Transfer Driver

The DTC module is recommended for most generic Transfer applications, but either module can be used for basic Transfer functionality. The use cases in which one Transfer module would be preferred over the other are described below.

Selecting the DTC Transfer Module

The DTC module uses a RAM based vector table, with slots for every interrupt in the system. When the DTC transfer completes, the activation source interrupt is called. The activation source interrupt must be enabled to use the DTC. The activation source interrupt is generally muted by the DTC until the transfer completes, unless TRANSFER_IRQ_EACH is specified in the configuration. For example, if a normal mode transfer with a length of 16 is triggered by a timer, the timer interrupt does not fire the first 15 times while the transfer is in effect. After the 16th transfer, the timer interrupt fires. The DTC also allows chained transfers, meaning that more than one transfer can occur after a single activation source interrupt. This feature is supported by the driver, but must be configured outside the e^2 studio IDE.

Selecting the DMAC Transfer Module

The DMAC module uses DMAC peripheral registers, so the number of transfers in the system is limited to the number of DMAC channels on the device. The activation source does not have to be enabled to use the DMAC. When the DMAC transfer completes, a DMAC interrupt is called. If the activation source interrupt is enabled, it fires at the same time the transfer is triggered. If the DMAC interrupt is enabled, it fires after all transfers are complete. For example, if a normal

mode transfer with a length of 16 is triggered by a timer, the timer interrupt fires at the same time each transfer occurs and the DMAC interrupt fires after the 16th transfer completes. The DMAC does not support chained transfers.

4.2.44.3 Using e2 studio to write an application with the Transfer Driver

The driver is integrated into the e2 studio ISDE: e2 studio ISDE User Guide.

In e2 studio, create and configure a project and add the drivers:

1) Create the project: Creating a Project.
2) Configure the project: Configuring a Project
3) Add the drivers: Adding Threads and Drivers

The following resources are required for an application that uses the Transfer Driver with the DTC:

Resource	ISDE Tab	Selection
DTC Driver	Threads	**Driver > Transfer > Transfer Driver on r_dtc**. If you add more than one Transfer driver, make sure to create a unique name for each driver in the Name field of the Properties editor.
Interrupts	Threads	The interrupt corresponding to the activation source must be enabled.
DTC Clock	Clocks	See Configuring the DTC Clocks
DTC Pins	Pins	See Configuring the DTC Pins

The following resources are required for an application that uses the Transfer Driver with the DMAC:

Resource	ISDE Tab	Selection
DMAC Driver	Threads	**Driver > Transfer > Transfer Driver on r_dmac**. If you add more than one Transfer driver, make sure to create a unique name for each driver in the Name field of the Properties editor.
Interrupts	Threads	**DMAC > DMACn > DMACn INT** Underflow Interrupt.
DMAC Clock	Clocks	See Configuring the DMAC Clocks
DMAC Pins	Pins	See Configuring the DMAC Pins

Configuring the DTC Clocks

Use the e^2 studio ISDE to configure the DTC clock using the Clocks tab: Configuring Clocks.

The DTC is clocked based on the ICLK frequency. You can set the ICLK frequency using the clock configurator in e^2 studio Configuring Clocks or the CGC Interface at run-time.

Configuring the DTC Pins

The DTC is not associated with any pins.

Configuring the DTC Interrupts

Use the e^2 studio ISDE to configure the DTC interrupts using the ICU tab: Configuring Interrupts.

The activation source interrupt must be enabled to use the DTC.

ATTENTION: Interrupts may be skipped when used with the DTC peripheral with irq set to TRANSFER_IRQ_END.

Configuring the DMAC Clocks

Use the e^2 studio ISDE to configure the DMAC clock using the Clocks tab: Configuring Clocks.

The DMAC is clocked based on the ICLK frequency. You can set the ICLK frequency using the clock configurator in e^2 studio Configuring Clocks or the CGC Interface at run-time.

Configuring the DMAC Pins

The DMAC is not associated with any pins.

Configuring the DMAC Interrupts

Interrupts are not required to use the DMAC. To receive an interrupt when a transfer completes, enable the DMAC > DMACn > DMACn INT interrupt (where n is the DMAC channel number) on the ICU tab of the Project Configurator in e^2 studio.

When the DMACn INT interrupt is enabled in the BSP, the corresponding ISR is defined in the DMAC driver. The ISR calls a user callback function if one was registered in open.

4.2.44.4 Transfer Driver Usage Notes

Normal Mode

In normal mode, a single transfer is triggered each time an activation source event occurs. A single transfer is 1 byte, 2 bytes, or 4 bytes depending on the setting selected in size. Each time a transfer occurs, length is decremented by 1. When length reaches 0, the transfer is complete.

Repeat Mode

In repeat mode, a single transfer is triggered each time an activation source event occurs. A single transfer is 1 byte, 2 bytes, or 4 bytes depending on the setting selected in size. Each time a transfer occurs, length is decremented by 1. When length reaches 0, length is reloaded with its initial value and the transfer restarts. If the repeat area is set to source, the source register is also reloaded with its initial value when the transfer restarts. Alternatively, if the repeat area is set to destination, the destination register is reloaded with its initial value when the transfer restarts.

Block Mode

In block mode, the entire transfer length is transferred each time an activation source event occurs. For example, if a transfer is configured in block mode with timer as the activation source, a 2 byte size, and a 12 byte length, 24 bytes are transferred each time the activation source event occurs. Each time a transfer occurs, length is decremented by 1. When length reaches 0, length is reloaded with its initial value and the transfer restarts. If the repeat area is set to source, the source register is also reloaded with its initial value when the transfer restarts. Alternatively, if the repeat area is set to destination, the destination register is reloaded with its initial value when the transfer restarts.

Address Modes

After each transfer of size (1 byte, 2 bytes, or 4 bytes), the source pointer and destination pointer is adjusted by src_addr_mode and dest_addr_mode respectively. For example, if src_addr_mode is set to TRANSFER_ADDR_MODE_INCREMENTED and size is set to TRANSFER_SIZE_4_BYTES, the p_dest pointer is incremented by 4 (the transfer size) after each transfer. The pointer does not change if set to TRANSFER_ADDR_MODE_FIXED.

Chained Transfers

Chained transfers are only supported by DTC. To use a chained transfer, create an array of transfer_info_t structures. Set chain_mode to TRANSFER_CHAIN_MODE_ENABLED for all transfers except the last transfer. Set p_info to the base of the first structure in the array for transfer_info_t structures.

4.2.44.5 Writing a Transfer Driver Application

The ISDE configures this instance structure when you generate the project in the ISDE generated source file.

```
/* Instance structure to use this module. */

const transfer_instance_t g_transfer =

{

    .p_ctrl = &g_transfer_ctrl,

    .p_cfg = &g_transfer_cfg,

    .p_api = &g_transfer_on_dtc

};
```

To write a Transfer application using the DTC, follow these steps:

1) Configure the module as described above.Once the module is configured, the module related header and configuration files are generated automatically (containing the structure transfer_cfg_t).

2) Open the Transfer instance using the name given to the Transfer during configuration.The appropriate driver is called through the Transfer interface. For a Transfer called g_transfer, this looks like

```
g_transfer.p_api->open(g_transfer.p_ctrl, g_transfer.p_cfg)
```

where g_transfer.p_ctrl and g_transfer.p_cfg are auto-generated after the Transfer configuration step.

3) If Auto Enable was configured to be False, enable the transfer by calling

```
g_transfer.p_api->enable(g_transfer.p_ctrl)
```

or

```
g_transfer.p_api->reset(g_transfer.p_ctrl, p_src, p_dest, length)
```

4) Use other APIs from transfer_api_t as desired.

4.2.44.6 Supported Devices for the Transfer Driver

This driver has been tested on the Synergy microcontroller family S7G2 and S3A7.

The Transfer Driver is designed to support the following families with no changes to the API:

- S124

4.2.44.7 Transfer Driver Files

During the project configuration, the ISDE extracts the files shown in the following table to the /ssp directory.

Module	Directory
HAL Transfer Interface API	synergy/ssp/inc/driver/api/r_transfer_api.h
DTC Instance	synergy/ssp/inc/driver/instances/r_dtc.h
DTC Driver	synergy/ssp/src/driver/r_dtc/r_dtc.c
DMAC Instance	synergy/ssp/inc/driver/instances/r_dmac.h
DMAC Driver	synergy/ssp/src/driver/r_dmac/r_dmac.c

4.2.45 UART Driver

The UART HAL Module is a high-level API for UART applications and is implemented on r_sci_uart. The UART HAL module uses the SCI peripherals on the Synergy MCU. A user-defined callback can be created to manage hardware-handshake and data operation, if needed.

4.2.45.1 UART HAL Module Features

The UART HAL module supports the standard UART protocol. The UART HAL module used in concert with the SCI peripheral in UART mode (UART on SCI) supports the following features (in addition to the standard UART protocol):

- Full-duplex UART communication
  - Simultaneous communication with multiple channels
  - Interrupt-driven data transmission and reception

– Invoking the user-callback function with an event code in the argument
– Baud-rate change at run-time
– Hardware resource locking during UART transaction
– CTS/RTS hardware flow control (with an associated IOPORT pin and supported by user-defined callback function)
– Integration with the DTC transfer module
– Abort in-progress read/write operations

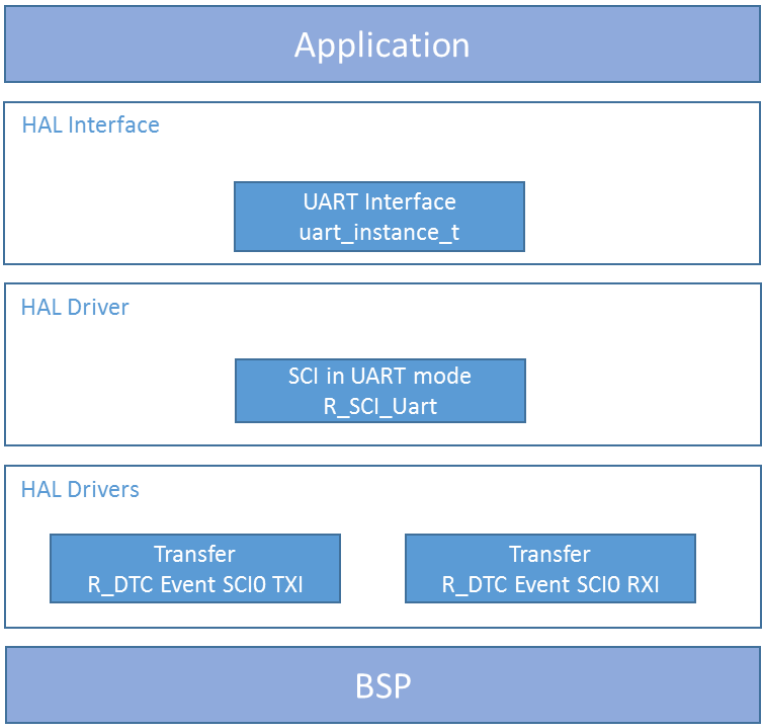


Figure 416: UART HAL Module Block Diagram

4.2.45.2 UART HAL Module APIs Overview

The UART HAL module interface defines APIs for key features such as opening, closing, reading, writing, and setting the baud rate. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows the API summary table.

UART HAL Module API Summary

Function Name	Example API Call and Description
.open	g_uart0.p_api->open(g_uart0.p_ctrl, g_uart0.p_cfg); Open UART device.
.read	g_uart0.p_api->read(g_uart0.p_ctrl, uart0_buf, uart0_rcv_num); Read from UART device. If a transfer instance is used for reception, the received bytes are stored directly in the read input buffer, uart0_buf. When a transfer is complete, the callback is called with event UART_EVENT_RX_COMPLETE. Bytes received outside an active transfer are received in the callback function with event UART_EVENT_RX_CHAR.
.write	g_uart0.p_api->write(g_uart0.p_ctrl, uart0_buf, uart0_send_num) Write to UART device. The write buffer is used until write is complete. Do not overwrite write buffer contents until the write is finished. When the write is complete (all bytes are fully transmitted on the wire), the callback called with event UART_EVENT_TX_COMPLETE.
.baudSet	g_uart0.p_api->baudSet(g_uart0.p_ctrl, (uint32_t)9600); Change baud rate.
.infoGet	g_uart0.p_api->infoGet(g_uart0.p_ctrl, &uart_info); Get the driver specific information.
.close	g_uart0.p_api->close(g_uart0.p_ctrl); Close UART device.
.versionGet	g_uart0.p_api->versionGet(&uart_version); Retrieve the API version with the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Channel operates successfully.
SSP_ERR_IN_USE	Control block has already been opened or channel is being used by another instance.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL or configuration structure is NULL.
SSP_ERR_HW_LOCKED	Channel is locked.
SSP_ERR_INVALID_MODE	Channel is used for non-UART mode or illegal mode is set.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter setting found in the configuration structure. Or source/destination address or data size is invalid against data length.
SSP_ERR_NOT_OPEN	The control block has not been opened.
SSP_ERR_UNSUPPORTED	SCI_UART_CFG_RX_ENABLE is set to 0.

NOTE: Lower-level drivers may return common error codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.45.3 UART HAL Module Operational Overview

The UART HAL Module manages data flow using the standard UART protocol. The high-level APIs are used to read, write, and set the baud rate for the UART interface. In addition, interrupts are typically used to simplify the management of low-level activities.

NOTE: Interrupts need to be enabled for the following functions to operate successfully.

UART on SCI RXI interrupt

The RXI interrupt is used to control the flow of data received from the UART port. When the amount of received data reaches the expected read length, the ISR invokes a user-defined callback (p_callback) with the argument uart_callback_args_t to indicate that the received data is complete. When the External RTS Operation option is enabled, the ISR invokes the UART callback function for the RTS external pin control twice: once at the top of ISR and once at the bottom. You can use the callback function to emulate the RTS function (see the UART on SCI hardware flow-control

section); this interrupt is activated in the open API as long as the reception is enabled in the SCI_UART_CFG_RX_ENABLE configuration parameter.

UART on SCI TXI interrupt

The TXI interrupt handles consecutive transmissions of data to the UART port as requested by the write API. When no data is left in the transmit circular buffer, the ISR deactivates the TXI interrupt and activates the TEI interrupt to handle the last sequence in the data transmission. This interrupt is activated in the write API as long as the transmission is enabled by the SCI_UART_CFG_TX_ENABLE configuration parameter.

UART on SCI TEI interrupt

The TEI interrupt is used to handle a last data transmission to the UART port requested by write API; this interrupt is activated by TXI ISR and deactivates itself. The ISR invokes a user-defined callback (p_callback) with the argument uart_callback_args_t to indicate that the end of data is transmit.

UART on SCI ERI interrupt

The ERI interrupt is used to handle errors that occur in the UART reception. This interrupt is activated in the open API as long as the reception is enabled by the SCI_UART_CFG_RX_ENABLE configuration parameter. The ISR invokes a user-defined callback (p_callback) with the argument uart_callback_args_t to indicate uart_event_t cause of an error.

UART HAL Module Important Operational Notes and Limitations

UART on SCI Hardware Flow Control

The SCI hardware module supports hardware flow-control for only one of the RTS or CTS signals at a time. CTS and RTS are multiplexed on the CTSn/RTSn pin so that one of the hardware flow-control signals can be used exclusively depending on the use-case. The UART HAL module expands this specification and allows control of both the CTS and the RTS signal by enabling an additional pin for the RTS signal. To enable this mode, set the UART on SCI configurations as follows:

- Set SCI_UART_CFG_EXTERNAL_RTS_OPERATION to Enable.
- Set ctsrts_en to CTS (true).
- Specify a user-callback function name to "Name of UART callback function for the RTS external pin control" in p_extpin_ctrl.

The UART on SCI HAL module invokes the user-callback function from the RXI ISR at the top and at the end of processing.

The callback function argument "level" refers to the signal level on the RTS pin for the selected SCI channel.

NOTE: The HAL module does not handle the GPIO pin-initialization or control it; instead, the user needs to initialize the GPIO pin before starting the UART reception.

The following figure shows the timing diagram of CTS/RTS hardware flow-control with an external GPIO pin used as the RTS signal:

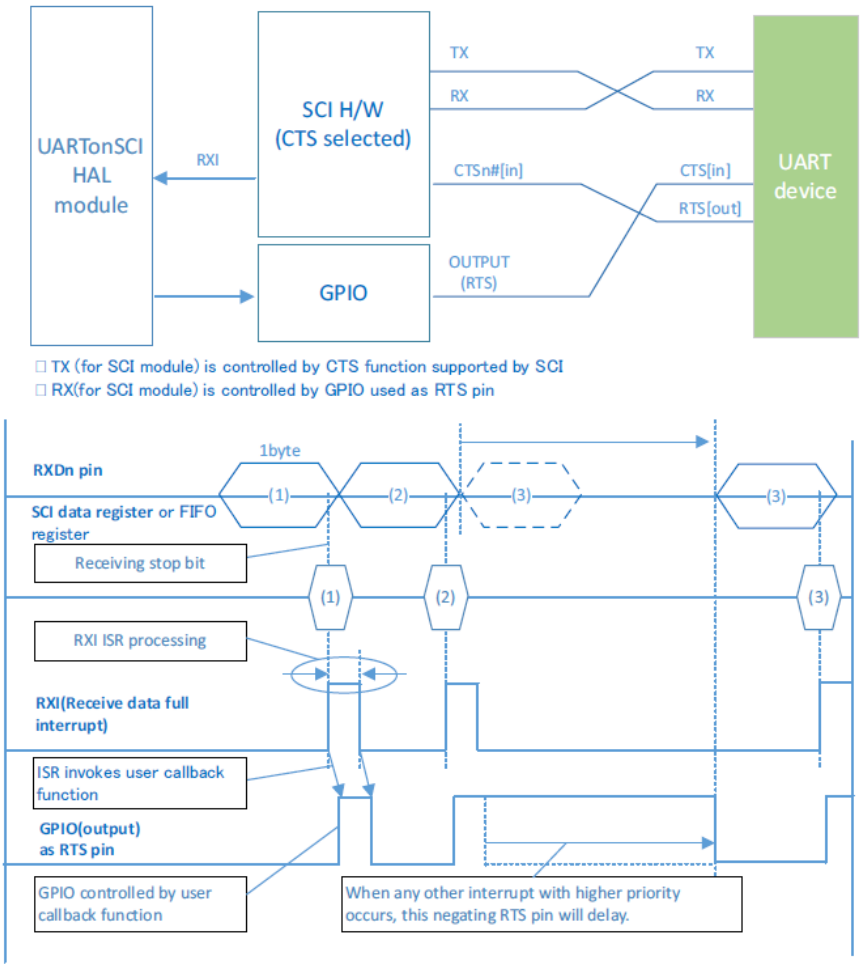


Figure 417: CTS/RTS Hardware Control with an External GPIO

NOTE: The UART on SCI module on the SK-S7G2 board uses PORT8 pin0 (pin P800) and J8 to activate the RS232C port on the RS-232C transceiver. Connect pin 1 and pin 2 of J8. Configure pins P800 as IOPORT pins and set its level for the desired operation.

- The module supports interrupt-based operation but does not support a polled UART mode.
- The module does not support non-buffered UART mode.
- The module does not support Event Link functionality.
- Transfer size must be less than or equal to 64 Kbytes if DTC interface is used for transfer. infoGet API can be used to get the max transfer size allowed

- Reception is still enabled after communicationAbort API is called. Any characters received after abort and before the next call to read, will arrive via the callback function with event UART_EVENT_RX_CHAR.
- There is a 64k limit to the block size that can be sent to the r_sci_uart driver if the DTC is being used. This limit can be retrieved using the infoGet API.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.45.4 Including the UART HAL Module in an Application

This section describes how to include the UART HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the UART Driver to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the UART HAL is g_uart0. This name can be changed in the associated Properties window.)

UART Driver Stack Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_uart0 UART on r_sci_uart	**Threads > HAL/Common Stacks**	Highlight **Threads > HAL/Common Stacks** and select **New Stack > Driver > Connectivity > UART Driver on r_sci_uart**

When the UART HAL module on r_sci_uart is added to a thread as shown in the following figure, the configurator automatically adds the needed lower-level drivers. Any drivers that need additional configuration information will be box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks.

When the mouse is docked to the Red position, the required operations for correcting the configuration will display. Please follow the instructions to enable the SCI Receive Interrupt (RXI), SCI Transmit Interrupt (TXI), and SCI Transmit End Interrupt (TEI) in the Properties window to complete a valid configuration.

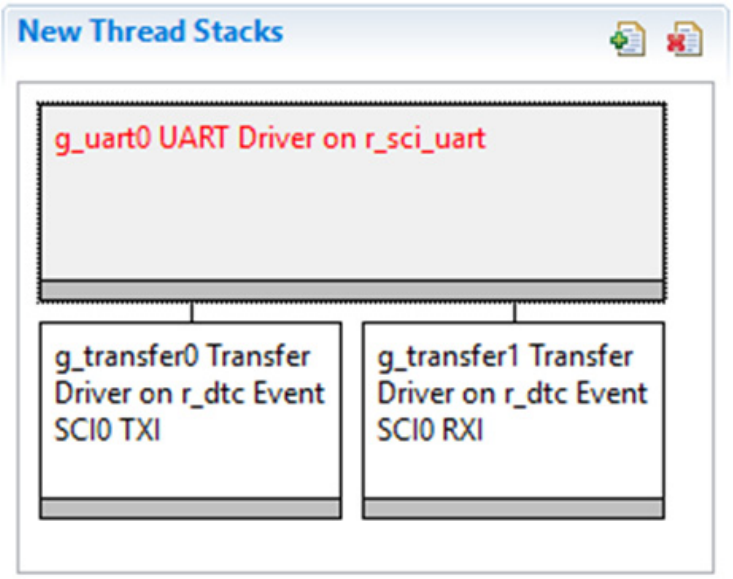


Figure 418: UART HAL Module Stack

4.2.45.5 Configuring the UART HAL Module

The UART HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the properties tab within the SSP configurator, and are shown in the following tables for easy reference.

NOTE: You may want to open your ISDE and create the UART HAL module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for UART HAL Module on r_sci_uart

ISDE Property	Value	Description
External RTS Operation	Enable, Disable Default: Disable	Enable an IOPORT pin to be used as RTS signal. For RTS functionality set this configuration parameter to **Enable** and specify the configuration **Name of UART callback function for the RTS external pin control**.

ISDE Property	Value	Description
Reception	Enable, Disable Default: Enable	Enable or disable UART reception for all UART channels on SCI. Setting this configuration parameter to **Disable** reduces code size because the portion of code for UART reception is not compiled. You cannot set this parameter for individual UART channels.
Transmission	Enable, Disable Default: Enable	Enable or disable UART transmission for all UART channels on SCI. Setting **Disable** to this configuration allows to get smaller code size due to the portion of code for UART transmission is compiled out, however, you can only set **Disable** to this configuration if any other SCI channels which work as UART ports do not perform the transmission.
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_uart0	The name to be used for UART on SCI module control block instance. This name is also used as the prefix of the other variable instances.
Channel	0-9	SCI channel number.
Baud Rate	9600	Baud rate selection.
Data Bits	7 bits, 8, bits, 9 bits Default: 8 bits	UART data bits.
Parity	None, Odd, Even Default: None	UART parity bits.
Stop Bits	1 bit, 2 bits Default: 1 bit	UART stop bits.

ISDE Property	Value	Description
CTS/RTS Selection	CTS (Note that RTS is available when enabling External RTS Operation mode which uses 1 GPIO pin), RTS (CTS is disabled) Default: RTS (CTS is disabled)	Select CTS or RTS for the CTSn/RTSn pin of SCI channel n. The SCI hardware supports either the CTS or the RTS control signal on this pin but not both. For an application that uses both CTS and RTS, select **CTS** for this configuration parameter and enable the configuration **External RTS Operation** specifying the configuration **Name of UART callback function for the RTS external pin control**.
Name of UART callback function to be defined by user	user_uart_callback	Name must be a valid C symbol.
Name of UART callback function for the RTS external pin control to be defined by user	NULL	Name must be a valid C symbol.
Clock Source	Internal Clock, External Clock 8x baudrate, External Clock 16x baudrate Default: Internal Clock	Selection of the clock source to be used in the baud-rate clock generator block.
Baudrate Clock Output from SCK pin	Enable, Disable Default: Disable	Optional setting to output the baud-rate clock on the SCKn pin for the selected channel n.
Start bit detection	Falling Edge, Low Level Default: Falling Edge	Start bit detection mode in the reception, usually set **Falling Edge** to this configuration.
Noise Cancel	Enable, Disable Default: Disable	Enable the digital noise cancellation on RXDn pin. The digital noise filter block in SCI consists of two-stage flip-flop circuits. For detail, refer to the Noise cancellation section in the Renesas Synergy hardware manual.
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bit rate modulation enable selection.

ISDE Property	Value	Description
Receive Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Error interrupt priority selection.

NOTE: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select different noise cancellation settings. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

NOTE: Most of the property settings for lower-level modules are fairly intuitive and usually can be determined by inspection of the associated Properties window from the SSP configurator.

Configuration Settings for the UART HAL Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower-level modules as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

Configuration for the Transfer Driver on r_dtc Event SCI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Set start mode
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section setting
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	1 Bytes	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode)	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 TXI	Activation source selection

ISDE Property	Value	Description
Auto Enable	True, False Default: True	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	ELC Software Event interrupt priority selection

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

Configuration for the Transfer Driver on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Set start mode
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section setting
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	1 Bytes	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection

ISDE Property	Value	Description
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 RXI	Activation source selection
Auto Enable	FALSE	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0(highest), Priority 1-2, Priority 3 (CM4: valid, CM0+: lowest – not valid if using ThreadX), Priority 4-14 (CM4: valid, CM0+? invalid), Priority 15 (CM4: lowest, not valid if using Thread X, CM0: invalid), Disabled Default: Disable	Interrupt priority for ELC SW event

NOTE: The example settings and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

When using the UART with CTS and RTS function simultaneously, the transfer driver cannot be used; please delete all transfer drivers on the low level. After being deleted, the optional transfer driver will display in pink, meaning the driver is recommended but optional, as in the following figure:

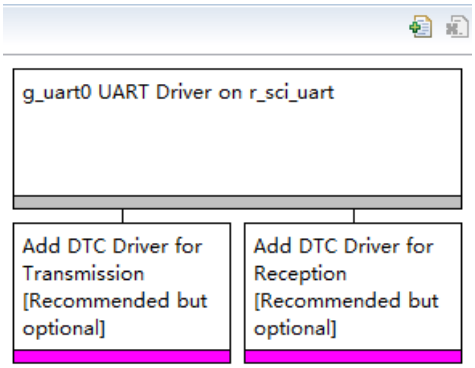


Figure 419: UART Stack with the CTS and RTS Functions

UART HAL Module Clock Configuration

The SCI UART peripheral uses PCLKA as its clock source (PCLKB for S124) or an external clock from the SCKn pin for the selected channel n.

UART HAL Module Pin Configuration

The SCI UART peripheral uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent tables illustrate an example selection for the UART pins.

NOTE: The operation mode selection determines what peripheral signals are available and thus what MCU pins are required.

Pin Selection Sequence for UART HAL Module on SCI

Resource	ISDE Tab	Pin selection Sequence
SCI	Pins	Select **Peripherals > Connectivity: SCI > SCI0**

Pin Configuration Settings for UART HAL Module on SCI

Pin Configuration Property	Value	Description
Pin Group Selection	Mixed, _A Only, _B Only (Default: Mixed)	Pin grouping selection

Pin Configuration Property	Value	Description
Operation Mode	Disabled, Custom, Asynchronous UART, Simple SPI, Simple I2C, Synchronous UART, SmartCard (Default: Simple SPI)	Select Operation Mode for UART on SCI
TXD_MOSI	None, P411, P101 (Default: P411)	TXD Pin
RXD_MISO	None, P410, P100 (Default: P410)	RXD Pin
SCK	None, P412, P102 (Default: P412)	SCK Pin
CTS_RTS_SS	None, P413, P103 (Default: None)	CTS Pin
SDA	Disabled	SDA Pin (when Simple I2C is used)
SCL	Disabled	SCL Pin (when Simple I2C is used)

NOTE: The example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

4.2.45.6 Using the UART HAL Module in an Application

Once the module has been configured and the files generated, the UART HAL module is ready to be used in an application. The typical steps in using the UART HAL module in an application are:

1) Initialize the UART HAL Module using the open API.
2) Set Baud Rate with the baudSet API (if needed.)
3) Read and Write data as needed using the read and write APIs and callbacks.
4) Close the UART HAL module using the close API as needed.

NOTE: Read or Write operations can be aborted using communicationAbort API if required. These common steps are illustrated in a typical operational flow diagram in the following figure:

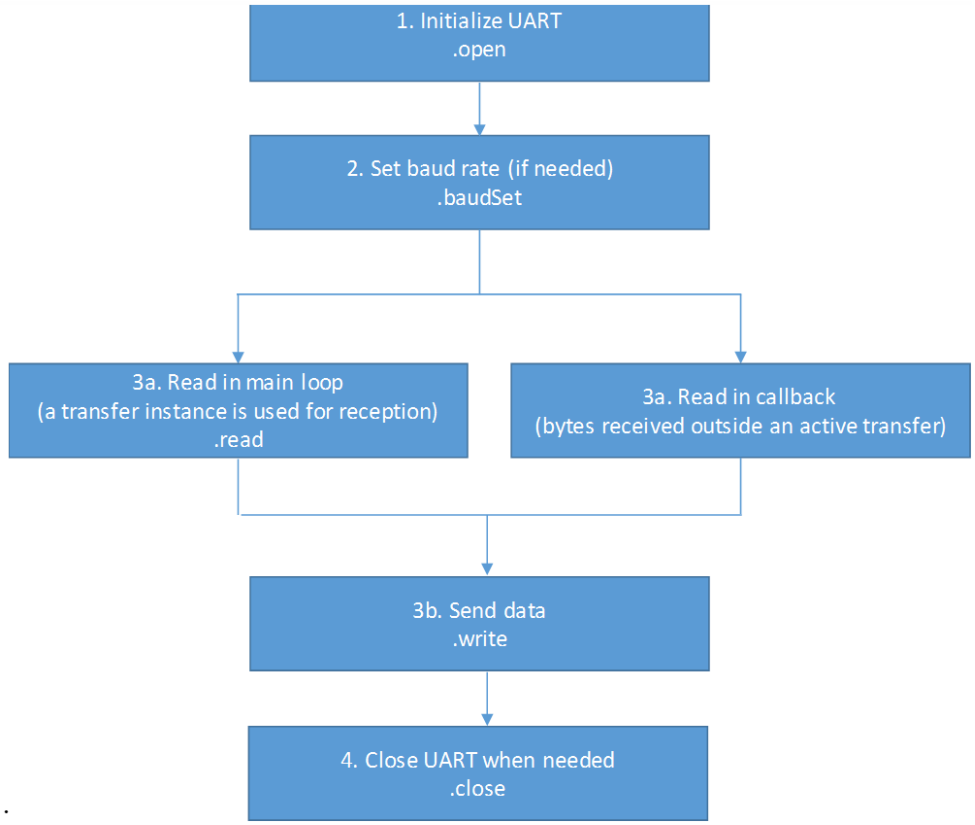


Figure 420: Flow Diagram of a Typical UART HAL Module Application

4.2.46 Watchdog Driver

4.2.46.1 Watchdog Driver

The WDT (Watchdog Timer) HAL module provides high-level APIs for WDT The WDT (Watchdog Timer) HAL module provides high-level APIs for WDT applications and is implemented on r_wdt. The WDT HAL module uses the WDT peripheral on the Synergy MCU. A user-defined callback can be created to respond to event notifications.

4.2.46.2 WDT HAL Module Features

The WDT HAL module has the following key features:

- When the WDT underflows or is refreshed outside of the permitted refresh window, one of the following events can occur:

  - Resetting of the device
  - Generation of an NMI
- Supports the Watchdog Timer (WDT) peripheral, which uses an external clock.
- The WDT can be configured in register start mode through the WDT registers.
- Supports automatic hardware configuration after reset.
- The WDT can be started from the application.

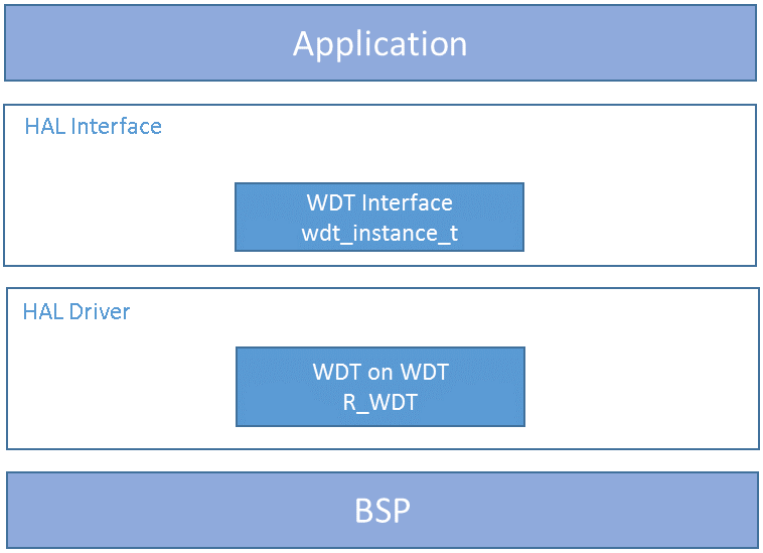


Figure 421:WDT HAL Module Organization, Options, and Stack Implementations

4.2.46.3 WDT HAL Module APIs Overview

The WDT HAL module defines APIs for opening, refreshing, reading, and getting status. A complete list of the available APIs, an example API call, and a short description of each can be found in the following table. A table of status return values follows.

WDT HAL Module API Summary

Function Name	Example API Call and Description
cfgGet	g_wdt0.p_api->cfgGet(g_wdt0.p_ctrl, g_wdt0.p_cfg); Initialize the WDT in register start mode. In auto-start mode with NMI output it registers the NMI callback.
open	g_wdt0.p_api->open(g_wdt0.p_ctrl, g_wdt0.p_cfg); Initialize the WDT in register start mode. In auto-start mode with NMI output it registers the NMI callback.

Function Name	Example API Call and Description
refresh	g_wdt0.p_api->refresh(g_wdt0.p_ctrl); Refresh the watchdog timer.
statusGet	g_wdt0.p_api->statusGet(g_wdt0.p_ctrl, &status); Read the status of the WDT.
statusClear	g_wdt0.p_api->statusClear(g_wdt0.p_ctrl, clear); Clear the status flags of the WDT.
counterGet	g_wdt0.p_api->counterGet(g_wdt0.p_ctrl, &counter); Read the current WDT counter value.
timeoutGet	g_wdt0.p_api->timeoutGet(g_wdt0.p_ctrl, &timeout); Read the watchdog timeout values.
versionGet	g_wdt0.p_api->versionGet(&version); Retrieve the API version using the version pointer.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	Function successfully executed.
SSP_ERR_ASSERTION	Null Pointer(s).
SSP_ERR_INVALID_ARGUMENT	One or more configuration options is invalid.
SSP_ERR_INVALID_MODE	An attempt to open the WDT in register-start mode when the OFS0 register is configured for auto-start mode. Or to open the WDT in auto-start mode when the OSF0 is configured for register start mode.
SSP_ERR_ABORTED	Invalid clock divider for this watchdog

NOTE: Lower level drivers may return Common Error Codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.2.46.4 WDT HAL Module Operational Overview

Synergy MCUs have two watchdog peripherals- the watchdog timer (WDT) and the independent watchdog timer (IWDT). When selecting between them consider these points:

- The WDT can be started from the application.
- The WDT can be configured in register start mode through the WDT registers. The WDT can also be configured by hardware automatically after reset using parameters stored in Option Function Select Register 0 (OFS0).
- The IWDT has its own clock source which improves safety.
- The IWDT is configured by hardware automatically after reset using parameters stored in the Option Function Select Register 0 (OFS0).

4.2.46.5 WDT HAL Module Operation

The WDT HAL module configures the WDT Interface. When the WDT underflows or is refreshed outside of the permitted refresh window, one of the following events can occur:

- Resetting of the device
- Generation of an NMI

The figure below shows an example of the operation of the WDT. When refreshed in the valid refresh period of the counter the timer count value is reset. If the count is allowed to underflow or refresh occurs outside of the valid refresh period, the WDT resets the device or generates an NMI.

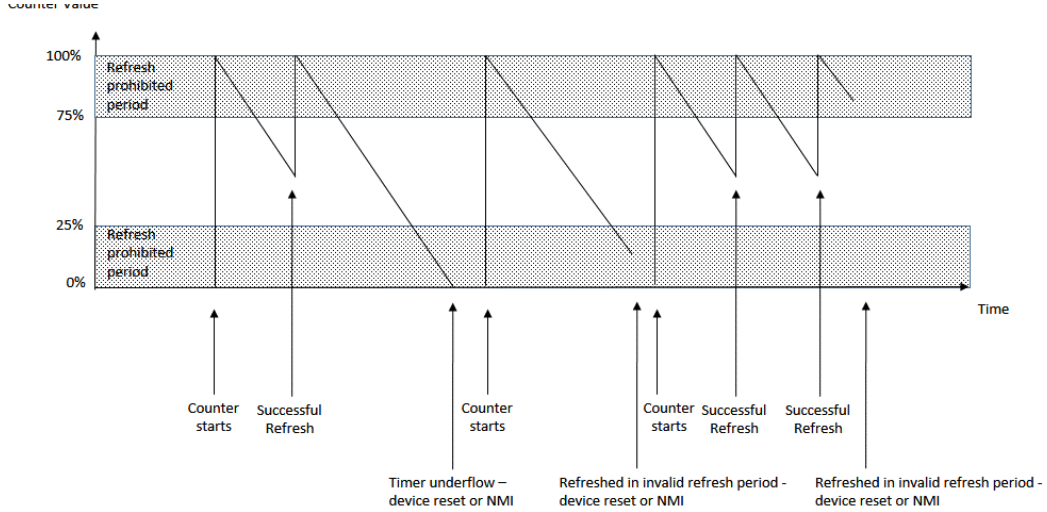


Figure 422: WDT HAL Module Operation

The WDT can be configured in register start mode through the WDT registers. The WDT can also be configured by hardware automatically after reset using parameters stored in Option Function Select Register 0 (OFS0) as shown in the below table.

All series of Synergy microcontrollers have an Option-Setting Memory which can be used to set the operating state of peripherals after a reset. The OFS can be used to set the state of the IWDT, WDT, LVD and CGC HOCO.

The following table details which parameters of the WDT can be configured by the OFS registers:

WDT Parameters

Control	Description
WDT Start Mode Select	Automatically starts the WDT after a Reset in enabled.
WDT Timeout Period	Specifies the WDT timeout (number of clock cycles). 128 cycles 512 cycles 1024 cycles 2048 cycles
WDT-Clock Frequency Division Ratio (The WDT is clocked from PCLKB)	PCLKB / 1 PCLKB / 64 PCLKB / 128 PCLKB / 512 PCLKB / 2048 PCLKB / 8192
WDT Window End Position	25% 50% 75% 100% (no window end position set)
WDT Window Start Position	25% 50% 75% 100% (no window start position set)
WDT Reset Interrupt Request	The WDT can either generate an Interrupt Signal or a Reset signal.
WDT Stop Control	The WDT can continue to count or Stop counting in Low Power Mode.

NOTE: For further information on the contents of the OFS0 register see the Synergy MCU hardware manual.

The OFS register values are set in the Synergy Configuration editor via properties on the BSP tab.

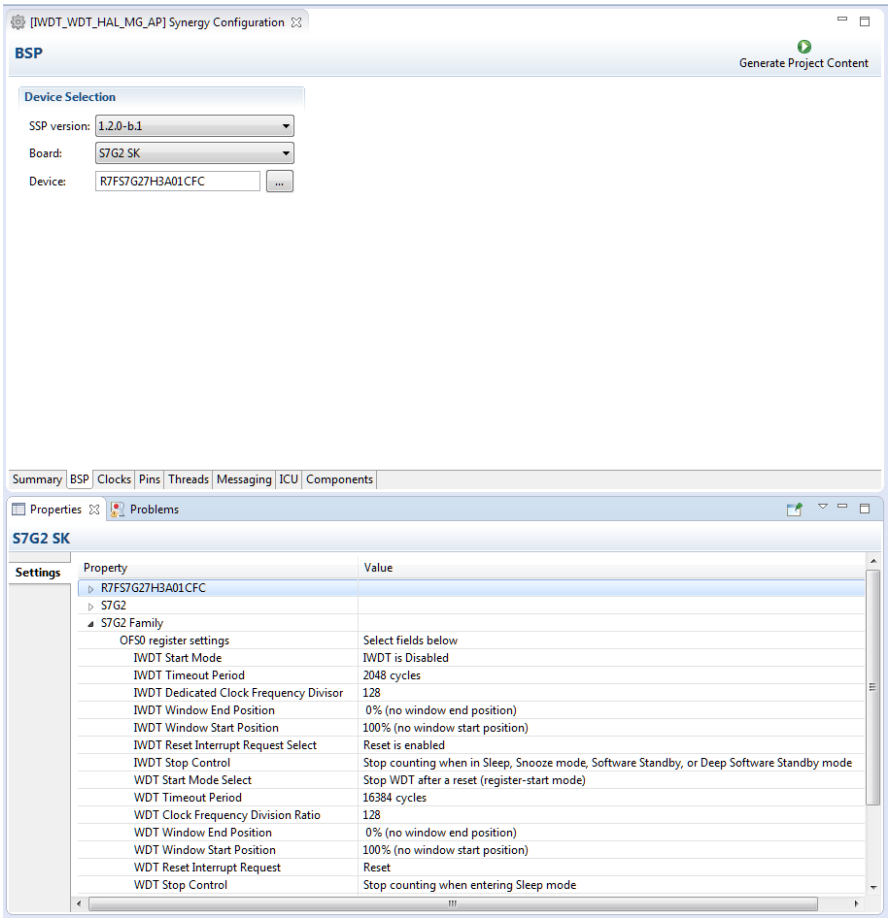


Figure 423: OFS Register Value Settings

4.2.46.6 WDT HAL Module Important Operational Notes and Limitations

4.2.46.7 WDT HAL Module Period Calculation

The WDT operates from PCLKB. Assuming largest parameters for the WDT and a PCLKB of 60 MHz, the time from the last refresh to device reset or NMI generation will be just over 2.2 seconds as detailed below.

PLCKB = 60 MHz

Clock division ratio = PCLKB/8192

Timeout period = 16384 cycles

WDT clock frequency = 60 MHz / 8192 = 7.324 kHz

Cycle time = 1 / 7.324 kHz = 136.53 us

Timeout = 136.53 us x 16384 cycles = 2.23 seconds

4.2.46.8 Triggering DMAC/DTC with the WDT HAL Module

To trigger a transfer of data using the DMAC or DTC peripheral when the WDT counter underflows or when a refresh is attempted outside of the valid refresh period, configure the WDT to generate an NMI and configure the DMAC/DTC transfer with activation_source set to ELC_EVENT_WDT_UNDERFLOW. See the associated User Guide (DMAC, DTC) for further information.

4.2.46.9 Triggering Event Link Controller Events with the WDT HAL Module

The WDT can trigger the start of another peripheral using the Event Link Controller (ELC). Refer to the ELC User Guide for a complete list of available peripherals.

4.2.46.10 WDT HAL Module Limitations

- When using a JLink debugger the WDT counter does not count and therefore will not reset the device or generate an NMI.
- When there is no active task ready to run, ThreadX puts the MCU into sleep mode. If the WDT is configured to stop the counter in low power mode, then your application must restart the timer when used with the ThreadX RTOS.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.2.46.11 Including the WDT HAL Module in an Application

This section describes how to include the WDT HAL module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP based applications.

To add the WDT HAL module to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the WDT HAL module is g_wdt0 as shown in the table below. This name can be changed in the associated properties window.)

Watchdog Timer Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_wdt0 Watchdog Driver on r_wdt	Threads	New Stack>Driver>Monitoring>Watchdog Driver on r_wdt

When the WDT HAL module on r_wdt is added to the thread stack as shown in the figure below, the configurator automatically adds any needed lower level modules. Any drivers that need additional configuration information will be box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

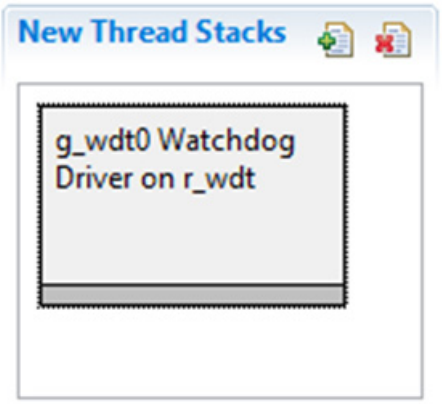


Figure 424: WDT HAL Module Timer Stack

4.2.46.12 Configuring the WDT HAL Module

The WDT HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections (such as interrupts or operating modes) which must be configured for lower level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the properties window of the associated module. Simply select the indicated module and then view the properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+.) This level of detail is not included in the configuration properties tables below, but is easily visible with the ISDE when configuring interrupt priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the following Configuration Table Settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration for the WDT HAL Module

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Include parameter checking code.
Name	g_wdt0	Module Name.
Start Mode	Register, Auto (Default: Register)	Configures the start mode as register start or auto-start.

ISDE Property	Value	Description
Start Watchdog After Configuration	True, False (Default: True)	Controls whether WDT is started during initialization.
Timeout	1024 cycles, 4096 cycles, 8192 cycles, 16384 cycles (Default: 16384 cycles)	WDT timeout period.
Clock Division Ratio	PCLK/4, PCLK/64, PCLK/128, PCLK/512, PCLK/2048, PCLK/8192 (Default: PCLK/8192)	WDT clock divider.
Window Start Position	100% (Window Position Not Specified), 75%, 50%, 25% (Default: 100%)	Permitted refresh period start position.
Window End Position	0% (Window Position Not Specified, 25%, 50%, 75% (Default: 0%)	Permitted refresh period end position.
Reset Control	Reset Output, NMI Generated (Default: Reset Output)	Select whether WDT should reset the MCU or generate an NMI.
Stop Control	WDT Count Enabled in Low Power Mode, WDT Count Disabled in Low Power Mode (Default: Disabled)	Select whether the WDT should stop counting in low power modes.
NMI Callback	NULL	Callback. A user callback function can be registered in open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the IRQn triggers. NSince the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

4.2.46.13 Configuring the Option Function Select Register 0 (OFS0)

All series of Synergy microcontrollers have an Option-Setting Memory which can be used to set the operating state of peripherals after a reset. The OFS can be used to set the state of the IWDT, WDT, LVD and CGC HOCO. See the description in the operational overview section earlier in this document.

4.2.46.14 Configuring the Interrupts for the WDT HAL Module

Configure the WDT interrupts in the same way as configuring the other options for the WDT module. If the WDT is configured to generate an NMI interrupt on underflow or invalid refresh, the interrupt must be enabled in the BSP.

To enable interrupts, set the priority of the CWDT > CWDT NMIUNDF N n. This sets BSP_IRQ_CFG_WDT_UNDERFLOW in ssp_cfg/bsp/bsp_irq_cfg.h to the priority level selected.

When the CWDT NMIUNDF N interrupt is enabled in the BSP, the corresponding ISR will be defined. The ISR will call a user callback function if one was registered in the open API.

4.2.46.15 WDT HAL Module Clock Configuration

The WDT clock is based on the PCLKB frequency. You can set the PCLKB frequency using the clock configurator in the ISDE or using the CGC Interface at run-time. The maximum timeout period with PCLKB running at 60 MHz is approximately 2.2 seconds.

4.2.46.16 WDT HAL Module Pin Configuration

The WDT does not require pins for their operation.

4.2.46.17 Using the WDT HAL Module in an Application

The typical steps in using the WDT HAL module in an application are:

1) Initialize the WDT HAL module in either register start mode or auto-start mode using the open API.
2) Read the configuration of the WDT HAL module in either register start mode or auto start mode with the cfgGet API.
3) Refresh the watchdog timer using the refresh API.
4) Read the WDT status flags using the statusGet API.
5) Clear the status flags and error flags of the WDT HAL module using the statusClear API.
6) Read the current WDT counter value using the counterGet API.
7) Read the watchdog timeout values using the timeoutGet API.

These steps are illustrated in a typical operational flow diagram shown below:

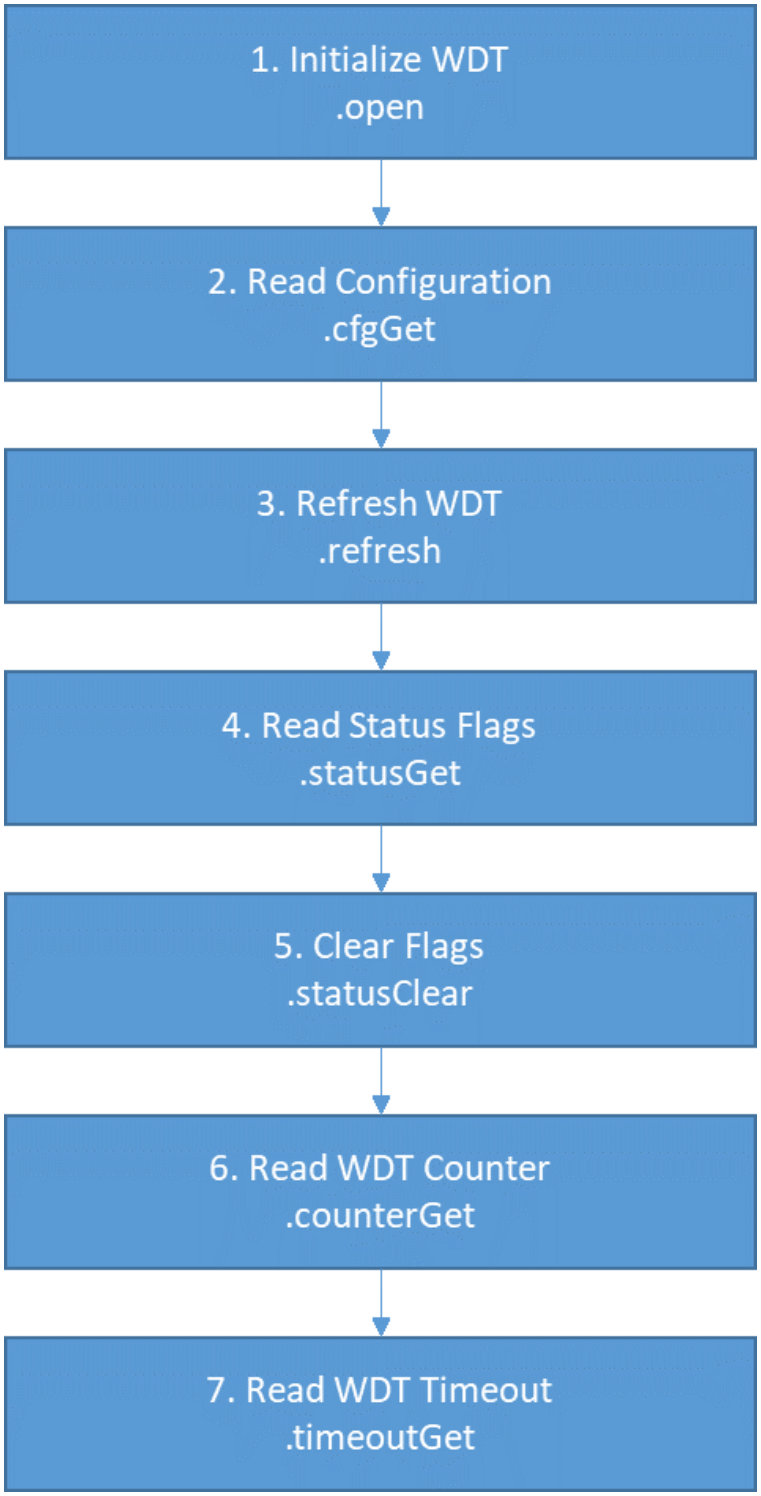


Figure 425: Flow Diagram of a Typical WDT HAL Module Application

4.3 Express Logic Modules

4.3.1 FileX Port Block Media Framework

The FileX Port Block Media Framework module, implemented on sf_el_fx, supports the Express Logic FileX system, a complete FAT format media and file management system for deeply embedded applications. The FileX Port Block Media Framework module provides I/O calls for FileX to access Synergy Media drivers through the Block Media Interface and adaptation layers. The SD/MMC HAL module and the SDIO HAL module are implemented on r_sdmmc and are used to read/write and control SD/MMC media devices as well as SDIO cards. The sf_el_fx module uses the SD/MMC peripherals on the Synergy MCU.

This document is divided into the following sections which can be read independently by a more experienced Synergy developer or in series by developers new to the Synergy Platform.

4.3.1.1 FileX Port Block Media Framework Features

- The FileX Port Block Media Framework module features include:
  - Support for FAT32, FAT16 and FAT12
  - Supports mounting the first FAT partition on a given media
  - Unlimited FileX objects (media, directories, and files)
  - Dynamic FileX object creation/deletion
  - Flexible memory usage
  - Size scales automatically
  - Small footprint
  - Complete integration with ThreadX
- SD card interface
  - Compatible with SD, SDHC, and SDXC formats

  - Supports 1-bit and 4-bit bus width
  - Card detect function when supported by the hardware
  - Write protect support
- eMMC interface
  - Supports 1-bit, 4-bit and 8-bit bus width
- SD bus interface
  - Compatible with SD memory card and SDIO card
  - Transfer bus mode selectable from 4-bit wide bus mode or 1-bit default bus mode
  - Compatible with SD, SDHC, and SDXC formats
- SD and MMC shared
  - DMAC and DTC triggerable by the SBFAI interrupt SD buffer is read and write accessible using the DMAC

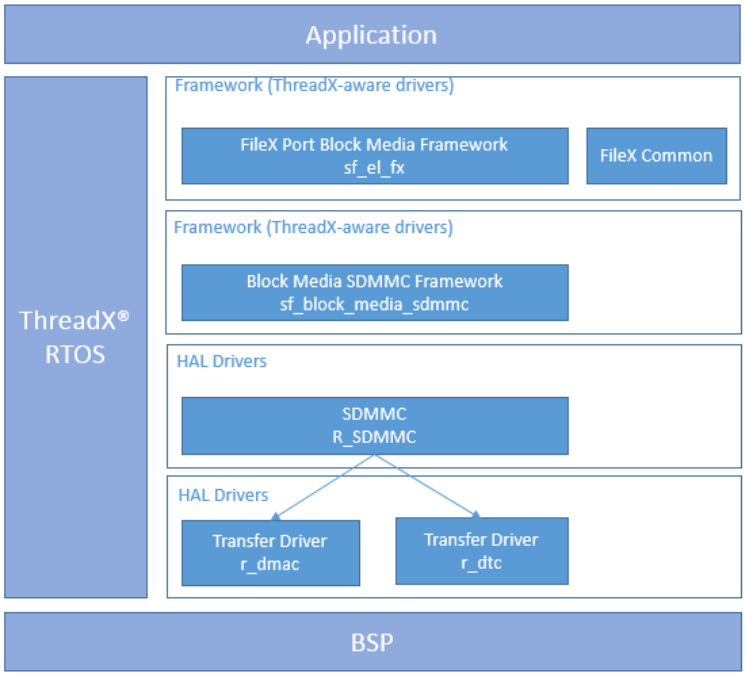


Figure 426: FileX Port Block Media Framework Stack with sf_block_media_sdmmc

4.3.1.2 FileX Port Block Media Framework APIs Overview

The FileX Port Block Media Framework implements the Express Logic FileX operations for file organization and access. The complete list of FileX related APIs is too long to provide here, so only some of the most important are provided in the below table, refer to the FileX User's Manual, available as described in the Reference section of this document.

FileX Port Block Media Framework API Summary (selected examples only)

Function Name	Example API Call and Description
fx_directory_attributes_read	fx_directory_attributes_read(&my_media, "mydir", &attributes); Read the directory attributes from the specified media
fx_file_open	fx_file_open(&my_media, &my_file, "myfile.txt", FX_OPEN_FOR_READ); Open "myfile.txt" file for read
fx_file_create	fx_file_create(&my_media, "myfile.txt"); Create file with name "myfile.txt"
fx_media_read	fx_media_read(&my_media, 22, my_buffer); Read the logic sector (in this example from sector 22) from media specified by &my_media, and places it into the buffer my_buffer

NOTE: For more complete descriptions of operation, status return values and definitions for the function data structures, typedefs, defines, API data, API structures and function variables review the SSP Reference Manual as described in the Reference Section at the end of this document. For FileX API documentation refer to X-Ware and NetX Component Documents for Renesas Synergy found as described in the Reference Section.

4.3.1.3 FileX Port Block Media and R_SDMMC Operational Overviews

FileX Port Block Media Operational Overview

FileX is a complete File Allocation Table (FAT) format media and file management system for deeply embedded applications. FileX supports an unlimited number of media devices at the same time:

- RAM disks
- FLASH managers
- Multiple other physical devices.

FileX supports 12-, 16-, and 32-bit FAT formats and contiguous file allocation. FileX is highly optimized for both size and performance. You can find the API reference for the FileX module in the FileX User's manual available as described in the References section at the end of this document.

FileX Port Block Media Framework Operational Notes

The media must be formatted to either a FAT12, FAT16 or FAT32 filesystem before it can be opened. This can be done prior to inserting the media or at run time.

FileX Block Media Framework Limitations

- SF_EL_FX does not currently support exFAT.
- When using SF_EL_FX with SD/eMMC the FileX block size must be 512 bytes.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

SDMMC HAL Module Operational Overview

The SD/MMC Driver and the SDIO Driver are implemented on r_sdmmc and are used to read/write and control SD/MMC media devices as well as SDIO cards. The SD/MMC module can be used as a standalone driver, or it can be used with FileX or any other compatible file system.

The SDMMC HAL Module supports the following memory devices:

- SDSC (SD Standard Capacity), SDHC (SD High Capacity), SDXC (SD Extended Capacity), and eMMC (embedded Multi Media Card)
  - Supports reading, writing, and erasing SD and eMMC memory devices
  - Supports 1, 4, or 8 bit data bus (8-bit bus is supported for eMMC only)
  - Supports detection of hardware write protection (SD cards only)
  - Automatically selects between backwards compatible mode and high speed SRD mode (eMMC)
- Supports SDIO
  - Supports SDIO single register access (CMD52)
  - Supports SDIO multiple register access (CMD53)
  - Supports SDIO interrupts
- Automatically configures the clock to the maximum clock rate supported by both host (MCU) and device

Interrupt Configurations

When using FileX, sf_el_fx and sf_block_media_sdmmc, the callback is implemented at the sf_block_media_sdmmc layer and does not have to be set up at the SDMMC module level. However, the interrupts and transfer drivers still must be configured. The interrupts required are:

Using SD/MMC with DTC:

- Access Interrupt
- DTC Interrupt

Using SD/MMC with DMAC:

- Access Interrupt
- DMAC Interrupt (in DMAC instance)

Using SDIO with DTC:

- Access Interrupt
- SDIO Interrupt
- DTC Interrupt

Using SDIO with DMAC:

- Access Interrupt
- SDIO Interrupt
- DMAC Interrupt (in DMAC instance)

The Card interrupt is optional, and only available on MCU packages that have the SDHIn CD pin (n = channel number) available on the Pins tab of the Synergy Configuration Tool.

Block Size Configuration

Block size configuration is provided for use with SDIO cards only. For SDIO cards, block size may be configured to 1-512 bytes. Block size must remain at the default 512 bytes for SD cards and e/MMC memory devices.

Card Detection Configuration

See Card Detection Limitations before using card detection in the r_sdmmc driver. If card detection is not available or not desired for the application, Card Detection must be set to Not Used in the Properties for the r_sdmmc instance in the Synergy Configuration Tool. To enable card detection, set Card Detection to CD Pin and Media Type to Card in the Properties for the r_sdmmc instance in the Synergy Configuration Tool.

Access Interrupt Priority

When data transfers are not 4-byte aligned or not a multiple of 4 bytes, a software copy of the block size (up to 512 bytes) is done in the access interrupt. This blocks other interrupts that are a lower or equal priority to the access interrupt until the software copy is complete.

General Timing Notes

Several functions in this driver block. open and erase block until the entire operation is complete. read,write, readIo,writeIo, readIoExt, and writeIoExt block for a single command response cycle. In a multithreaded system, care should be taken to use this driver in a lower priority thread if other threads require a response time faster than the time this driver could block for during one of these function calls.

Timing Notes for Open

The open API completes the entire device identification and configuration process. This involves several command-response cycles at a bus width of 1-bit and a bus speed of 400 kHz or less.

Timing Notes for Read, Write, ReadIoExt, and WriteIoExt

The read and write media (read and write) and extended read and write SDIO APIs (readIoExt and writeIoExt) block until the response is received from the device. The data transfer operation is non-blocking and requires interrupts and a transfer instance, either DMAC or DTC. These APIs return SSP_SUCCESS to indicate that the initial operations have started successfully. The application must wait for a callback with the event SDMMC_EVENT_TRANSFER_COMPLETE or SDMMC_EVENT_TRANSFER_ERROR to indicate completion of the read or write.

Timing Notes for ReadIo and WriteIo

The SDIO readIo and writeIo APIs block until the response is received from the device. The read or write operation is complete when these APIs return.

Timing Notes for Erase

The erase API blocks until the erase operation is complete. This may be several seconds or more depending on how many sectors are being erased.

SDIO Interrupts

Many SDIO cards use level interrupts, meaning the interrupt is not deasserted until the interrupt is cleared on the device. In order to ensure SDIO interrupts are cleared appropriately, one of the following methods are recommended:

- Ensure the SDIO interrupt is cleared on the device before exiting the callback function with the event SDMMC_EVENT_SDIO. If this method is chosen and any SDIO API must be used in the interrupt, the access interrupt must be a higher priority than the SDIO interrupt.
- Disable the SDIO interrupt in the callback function with the event SDMMC_EVENT_SDIO using IoIntEnable. Clear the SDIO interrupt elsewhere in the application, then re-enable SDIO interrupts if desired using IoIntEnable.

SD HALA Compliance

When developing host devices that are compliant with the SD Specifications, the host must comply with the SD Host/Ancillary Product License Agreement (SD HALA).

Data Alignment and Size

Data transfers should be 4-byte aligned and a multiple of 4 bytes in size whenever possible. This recommendation applies to the read(), write(), readIoExt(), and writeIoExt() APIs. When data transfers are 4-byte aligned and a multiple of 4-bytes, the r_sdmmc driver is zero copy and takes full advantage of hardware acceleration by the DMAC or DTC. When data transfers are not 4-byte aligned or not a multiple of 4 bytes an extra CPU interrupt is required for each block transferred (see Access Interrupt Priority).

Card Detection Limitations

Card detection support in the r_sdmmc driver is limited. Card detection is only available after open is complete, and open() cannot be completed unless a card is inserted. Card detection in the r_sdmmc driver is therefore only useful to detect card removal and reinsertion. After reinsertion is detected, the driver must be closed and reopened, and card detection will not be available until the reopen is complete. Card detection is available only on MCU packages that have the SDHIn CD pin (n = channel number) available on the Pins tab of the Synergy Configuration Tool. If the MCU does not have the SDHIn CD pin, or card detection is not desired for the application, card detection must be disabled in the Properties for the r_sdmmc instance in the Synergy Configuration Tool. An alternative to using the card detection feature of the r_sdmmc driver is to use an External IRQ instance and handle card detection at the application layer. If card detection is handled at the application layer, the open should be called after card insertion is detected, and the close should be called after card removal is detected.

Other Limitations

Refer to the most recent SSP Release notes for the most up to date limitations for this module.

4.3.1.4 Including the FileX Port Block Media Framework in an Application

This section describes how to include the FileX Port Block Media Framework in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User Manual available as described in the Reference Section at the end of this document to learn how to manage each of these important steps in creating SSP based applications.

To add the module to an application, simply add it to a thread using the Stacks Selection Sequence given in the table below. (The default name for the modules is shown in the table below. This name can be changed in the associated Properties window). When the driver is added it shows up in the thread as illustrated in the following figure.

Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_el_fx0 FileX Port Block Media Framework on sf_el_fx	**Threads**	**New Stack> Framework > File System> FileX Port Block Media Framework on sf_el_fx**

When the FileX Port Block Media Framework on sf_el_fx is added to the Thread Stack, as shown in the figure below, the configurator automatically adds the needed lower level drivers. Any drivers that need additional configuration information will be box text highlighted in red. Modules with a gray band are individual modules that stand alone. Modules with a blue band are shared or common and need only be added once, since they can be used by multiple stacks. Modules with a pink band can require the selection of lower level drivers. Sometimes these are optional or recommended and this is indicated in the block with the inclusion of this text. If the addition of lower level drivers is required, the module description will include "Add" in the text. Clicking on any pink banded modules will bring up the "New" icon and then will show the possible choices.

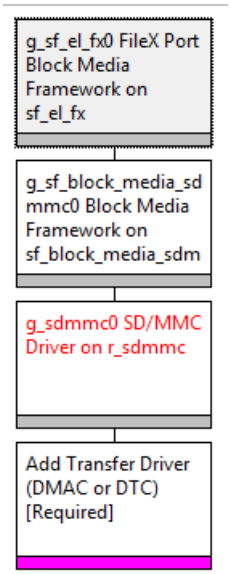


Figure 427: FileX Port Block Media Framework Stack

4.3.1.5 Configuring the FileX Port Block Media Framework Module

The module must be configured by the user for the desired operation. The SSP configuration window will automatically identify, by highlighting the block in red, any required configuration selections, such as Interrupts or operating modes, which must be configured for lower level modules, for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP Configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the Interrupt Priority. This configuration setting is available with the Properties window of the associated module. Simply select the indicated module and then view the

properties window. The Interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the Interrupt Priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the below configuration properties tables, but is easily visible with the ISDE when configuring Interrupt Priority levels.

NOTE: You may want to open your ISDE and create the FileX Port Block Media and explore the property settings in parallel with looking over the Configuration Table Settings given below. This will help orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Configuration Settings for FileX Port Block Media Module

ISDE Property	Setting	Description
Parameter Checking	Enabled, Disabled, BSP (Default: BSP)	Selects if code for parameter checking is to be included in the build
Name	Default: g_sf_el_fx0	FileX Port Block Media framework Module name

Configuration Settings for the Block Media Framework on sf_block_media_sdmmc Module

ISDE Property	Setting	Description
Parameter Checking	Enabled, Disabled, BSP (Default: BSP)	Selects if code for parameter checking is to be included in the build
Name	Default: g_sf_block_media_sdmmc0	Block Media framework Module name
Block size of media in bytes	512	Media Block size

NOTE: The above setting examples and defaults are for a project using the Synergy S7. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower level modules can be desirable. For example, the bus width can be different for different media types. The configurable properties for the lower level stack modules are given in the below sections for completeness and as a reference.

NOTE: Most of the property settings for lower level modules are fairly intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

4.3.1.6 Configuration Settings for the SD/MMC Driver and SDIO Driver

Typically, only a small number of settings must be modified from the default for lower level drivers and these are indicated via the red text in the Thread Stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The table below identifies all the settings within the properties section for the module.

Configuration for the SD/MMC and SDIO Driver

ISDE Property	Setting	Description
Parameter Checking	Enabled, Disabled, BSP (Default: BSP)	Selects if code for parameter checking is to be included in the build
Name	Default: g_sdmmc0	Module name
Channel	0,1 Default 1	Channel of SDMMC peripheral, channel 0 or 1
Media type	Embedded, Card Default: Embedded	Media is a card or an embedded device. This allows to firmware to know whether to look for card insertion/removal and write protect pins.
Bus Width	1 bit, 4 bits, 8 bits	Data bus with as defined by hardware interface. (8 Bits for eMMC only)
Block Size	512	Size of block

ISDE Property	Setting	Description
Callback	NULL (default) Name of user callback function.	(Not required if using FileX) Set to name of user callback function. Provides event that caused interrupt: SDMMC_EVENT_CARD_REMOVED, SDMMC_EVENT_CARD_INSERTED, SDMMC_EVENT_ACCESS, SDMMC_EVENT_SDIO, SDMMC_EVENT_TRANSFER_COMPLETE, SDMMC_EVENT_TRANSFER_ERROR
Access Interrupt Priority	Priority level 0-15 Default: Disabled	Interrupt level priority required for reading, writing, and errors for SD, eMMC and SDIO. Calls the interrupt callback if enabled.
Card Interrupt Priority	Priority level 0-15 Default: Disabled	(Optional) Interrupt for card removal. Automatically closes device when card is removed. Calls the interrupt callback if enabled.
DMA Request Interrupt Priority	Priority level 0-15 Default: Disabled	Interrupt priority level required for SDIO reading, writing, and errors. Calls the interrupt callback if enabled. Not used for memory cards.

NOTE: The above setting examples and defaults are for a project using the Synergy S7. Other MCUs may have different default values and available configuration settings.

SDHI Clock Configuration

The SDHI uses PCLKA for its clock source. There is no need to configure the clock specifically for the SDMMC peripheral, unless you need to optimize the data rate. The SDMMC driver selects the appropriate built-in divider based on

the PCLKA frequency and the maximum clock rate allowed by the SD, SDIO, or eMMC device, obtained at media device initialization.

SDMMC Pin Configuration

Use the e^2 studio pin configurator to configure the I/O pins for SDMMC peripheral. The drive capacity for each pin should be set to "Medium" or "High" for most boards and high-speed memory and SDIO devices. The following tables illustrate the method for selecting the pins within the SSP configuration window and provides an example selection for the module pins.

NOTE: The Operation Mode selection mode determines what peripheral signals are available and thus what MCU pins are required.

Pin Selection Sequence for SDHI Peripheral

Resource	ISDE Tab	Pin selection Sequence
SDHI	**Pins**	Select **Peripherals > Storage:SHDI > SDHI0**

NOTE: The above selection sequence assumes SDHI0 is the desired hardware target for the driver.

Pin Configuration Settings for SDHI Peripheral

Pin Configuration Property	Settings	Description
Operation Mode	Disabled, Custom, SD_MMC 1 bit SD_MMC 4 bit MMC 8 bit (Default: Custom)	Select mode as per application
CLK	None, P413 (Default: P413)	Clock Pin

Pin Configuration Property	Settings	Description
CMD	None, P412 (Default: P412)	Command Pin
DAT0-7	None, PXXX (Default: PXXX)	Data Pin
CD	None, P903 (Default: P903)	Card detection Pin
WP	None, P414 (Default: P414)	Card write protection pin

NOTE: The above example settings are for a project using the Synergy S7G2 and the DK-S7G2 Kit. Other Synergy Kits and other Synergy MCUs may have different available pin configuration settings.

Other Settings

The read and write media (R_SDMMC_Read and R_SDMMC_Write) and extended read and write SDIO functions (R_SDMMC_ReadIoExt and R_SDMMC_WriteIoExt) have been made non-blocking in Release 1.1.0 and now require interrupts and a transfer function, either DMAC or DTC. The read and write functions return SSP_SUCCESS to indicate that the initial operations have started successfully. However, the user application must wait for the user callback and check for event SDMMC_EVENT_TRANSFER_COMPLETE or SDMMC_EVENT_TRANSFER_ERROR to indicate completion of the read or write.

4.3.1.7 Using the FileX Port Block Media Framework Module or SDMMC HAL Module in an Application

The typical steps in using the FileX Port Block Media Framework Module on sf_el_fx in an application are:

1) Initialize the media using the FileX API fx_system_initialize (sf_el_fx calls it automatically if "Auto Initialization" property is set to Enabled in "FileX Common on fx").
2) Optionally, format the media using the FileX API fx_media_format
   - sf_el_fx automatically formats the media during the generated initialization function if "Format media during initialization" property is set to Enabled).
   - If "Filesystem on SDMMC" is enabled the entire media after the hidden sectors will be formatted and the "Total Sectors" configuration in "FileX on Block Media" is ignored.
3) Open the media using the FileX API fx_media_open (sf_el_fx opens the media automatically if "Auto Initialization" is set to Enabled in the FileX on Block Media instance).

4) Create and delete files and directories as required using one of the FileX APIs (for example, fx_file_create, fx_file_delete, fx_directory_create, fx_directory_delete).
5) Read from and write to files on the media as required using one of the FileX API (for example, fx_file_read or fx_file_write).
6) Read from and write to the media directly as required using one of the FileX API (for example, fx_media_read or fx_media_write).
7) Close the physical media using the FileX API fx_media_close.

NOTE: After a successful fx_media_open call all FileX file and directory related APIs can be used. Refer to the FileX User Guide for documentation on all available functions.

These common steps are illustrated in a typical operational flow diagram in the following figure.

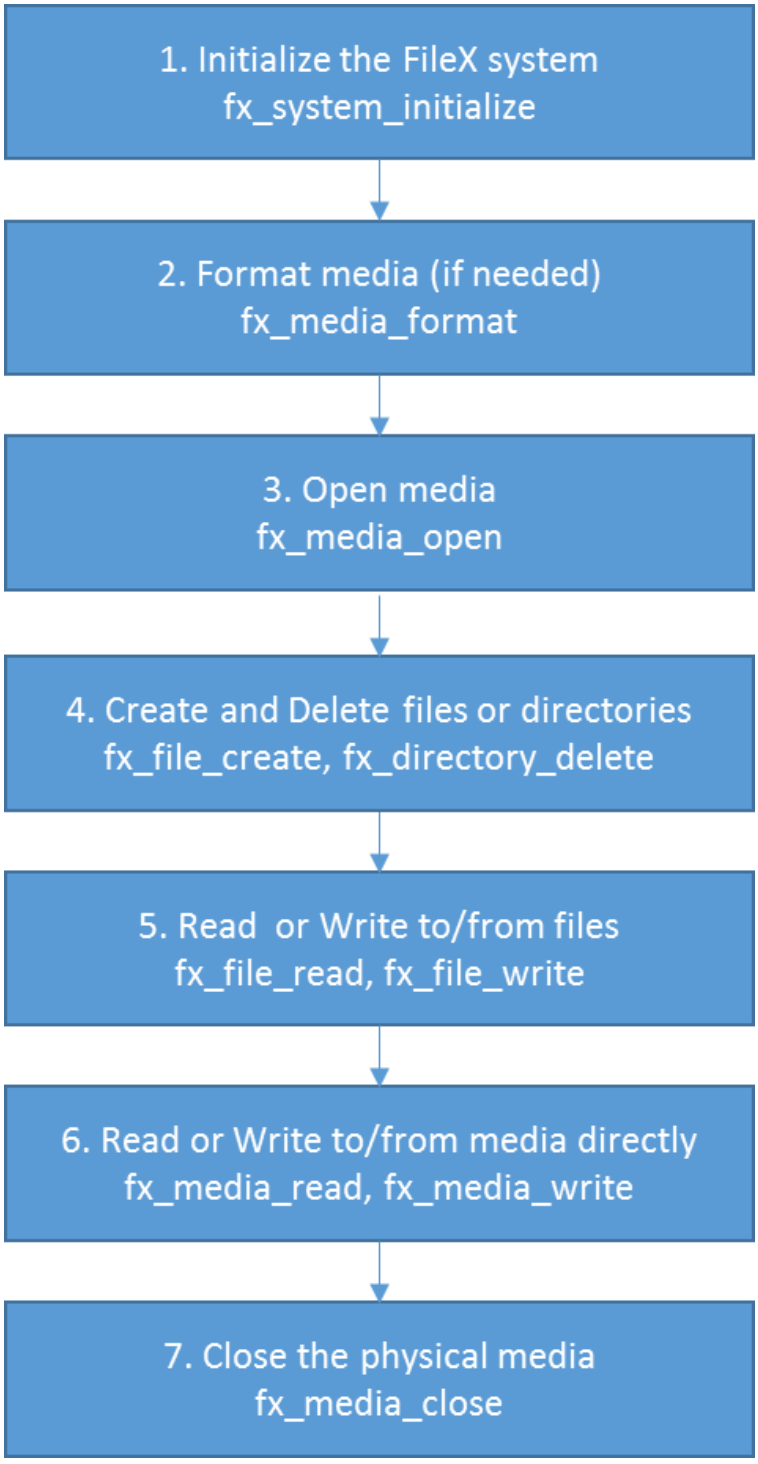


Figure 428: Flow Diagram of a Typical FileX Port Block Media Application

The typical steps for using the r_sdmmc driver with an SD or eMMC memory device are:

1) Open the driver using open.
2) Read data from the card using read or write data to the card using write.
3) Wait for a callback with the event SDMMC_EVENT_TRANSFER_COMPLETE (meaning the operation completed successfully) or SDMMC_EVENT_TRANSFER_ERROR (meaning the operation did not complete successfully) after each time read or write is called.

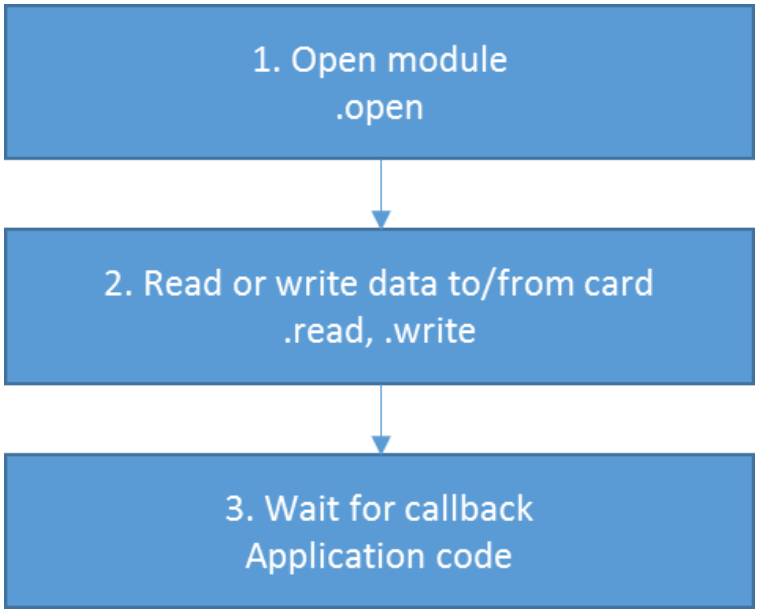


Figure 429: Flow Diagram of a Typical SDMMC SD or eMMC Application

The typical steps for using the r_sdmmc driver with an SDIO card are:

1) Open the driver using open.
2) Set or read back single registers using readIo or writeIo. The operation is complete immediately after these calls and there is no need to wait for a callback.
3) Set or read back multiple registers using readIoExt or writeIoExt. The block size can be changed using control between calls if necessary.
4) Wait for a callback with the event SDMMC_EVENT_TRANSFER_COMPLETE (meaning the operation completed successfully) or SDMMC_EVENT_TRANSFER_ERROR (meaning the operation did not complete successfully) after each time readIoExt or writeIoExt is called.
5) If the card requests an interrupt, the callback is called with the event SDMMC_EVENT_SDIO. Handle the SDIO interrupt as described in the documentation for the card (see the SDIO Interrupts section of this documentation). SDIO interrupts from the card can be enabled or disabled at any time using IoIntEnable.

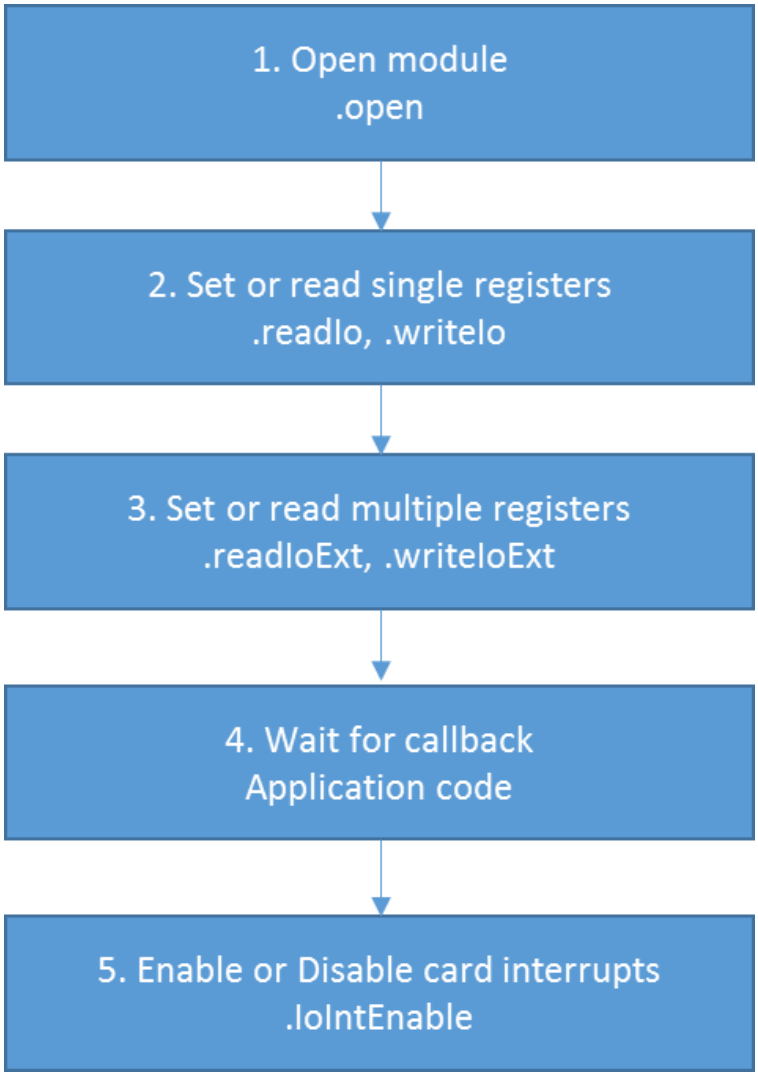


Figure 430: Flow Diagram of a Typical SDMMC SDIO Application

4.3.2 GUIX Port Framework

The Express Logic GUIX Synergy Port Module, SF_EL_GX, is the Express Logic GUIX™ adaptation layer for Synergy MCU groups, which have graphics engines GLCDC, DRW (2DG engine) or a JPEG decode engine. The APIs it provides support graphics hardware engine setup for GUIX and support graphics rendering and displaying accelerated by hardware engines. The module defines full-set of GUIX low-level display driver functions which draw graphics accelerated by DRW (2DG engine) or JPEG, or displays graphics with GLCDC (See the GUIX User Guide Chapter 5: GUIX Display Drivers). The module encourages the hardware acceleration for graphics rendering but also allows software processing without hardware support.

4.3.2.1 GUIX Synergy Port Framework Module Features

The GUIX Synergy Port Framework module includes the following key functions:

- Adapts GUIX to the SSP Framework
- Attaches the SSP Display Interface driver to GUIX Display Driver Interface
- Allows GUIX to draw widgets accelerated by the Synergy D2W (2DG) engine
- Allows GUIX to draw widgets accelerated by the Synergy JPEG engine
- Supports double-buffer toggling control for screen transitions without tearing
- Supports screen rotation (90/180/270 degree)
- Supports various output color formats
  - 32bpp (ARGB8888, RGB-888)
  - 16bpp (RGB565)
  - 8bpp (8bit Palette (CLUT))
- Support for user callback functions

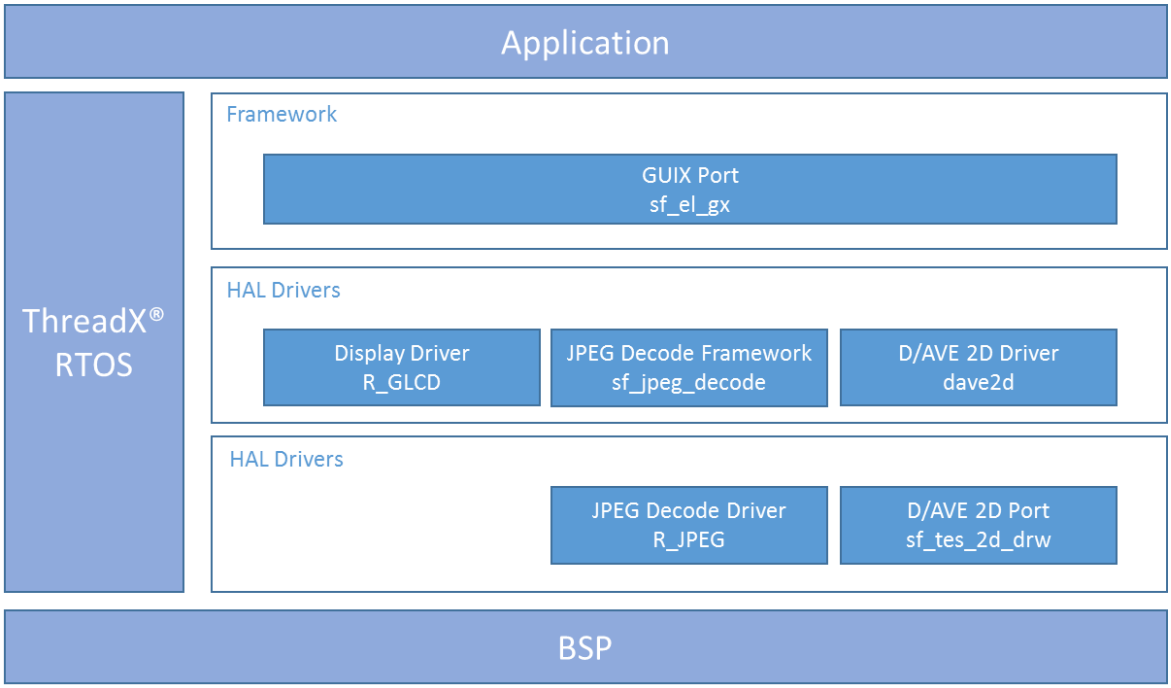


Figure 431: GUIX Synergy Port Framework Organization, Options and Stack Implementations

4.3.2.2 GUIX Synergy Port Framework Module APIs Overview

The GUIX Synergy Port Framework defines APIs for opening, closing, setup and initialization. A complete list of the available APIs, an example API call, and a short description of each can be found in the table below. A table of status return values follows.

GUIX Synergy Port Framework Module API Summary

Function Name	Example API Call and Description
.open	g_sf_el_gx.p_api->open(g_sf_el_gx.p_ctrl, g_sf_el_gx.p_cfg); Opens the SF_EL_GX Module. The API can only be called from a thread. The API passes the configuration pointer to define low-level graphics device drivers and frame buffers and register the user callback function. This function does not actually initialize low-level drivers. Instead, the API setup initializes the low-level drivers. The reason is described in the explanation for setup below.
.close	g_sf_el_gx.p_api->close(g_sf_el_gx.p_ctrl); Closes the SF_EL_GX Module. This API closes the low-level drivers. Normally, the API is not called since GUIX will not be closed once initialized.
.versionGet	g_sf_el_gx.p_api->versionGet(&version); Returns the version of the Module in the version pointer.

Function Name	Example API Call and Description
.setup	gx_studio_display_configure (MAIN_DISPLAY, g_sf_el_gx.p_api->setup, LANGUAGE_ENGLISH, MAIN_DISPLAY_THEME_1, &p_window_root); This is the interface to initialize low-level graphics device drivers and must be passed to GUIX through GUIX (Studio) service call gx_studio_display_configure() as the function pointer. GUIX then calls the API back and, at that moment, the API configures the SSP device drivers based on the configuration passed by open. The reason for this procedure to initialize low-level drivers is that the API has the GUIX-compliant argument (GX_DISPLAY *) type and does not allow applying the detailed configuration of the SSP graphics device drivers generated from e^2 studio.
.canvasInit	g_sf_el_gx.p_api->canvasInit(g_sf_el_gx.p_ctrl, p_window_root); This is the GUIX helper API to determine the memory address of GUIX canvas. The API has an argument with (GX_WINDOW_ROOT *) type and the API provides GUIX the start address of canvas memory, which is needed for the low-level graphics device drivers to draw/display images.

NOTE: For more complete descriptions of operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables review the SSP Users Manuals API References for the associated module.

Status Return Values

Name	Description
SSP_SUCCESS	API call successful.
SSP_ERR_ASSERTION	NULL pointer error happens.
SSP_ERR_IN_USE	SF_EL_GX is in-use.

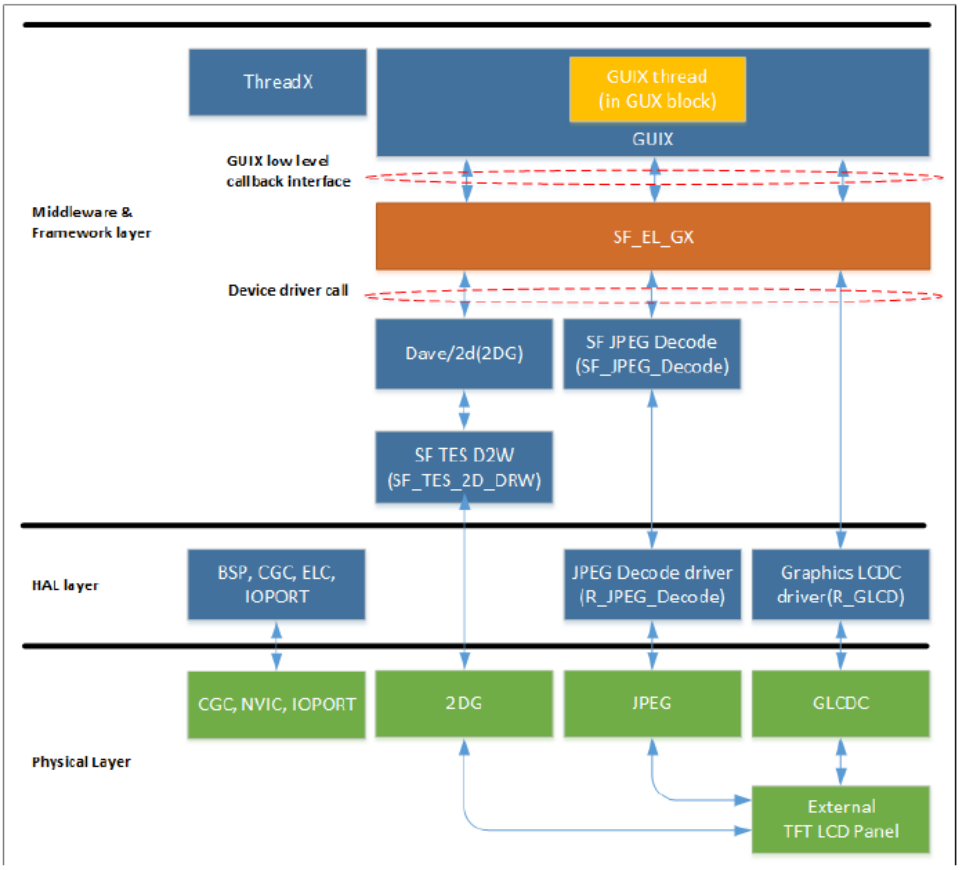
Name	Description
SSP_ERR_INTERNAL	Error happen in Kernel service calls.
SSP_ERR_NOT_OPEN	SF_EL_GX is not opened.
SSP_ERR_TIMEOUT	A task times out (or exceeds retry limit) before completion in display driver.
SSP_ERR_D2D_ERROR_DEINT	Error occurred in D/AVE 2D driver.
GX_SUCCESS	Device driver setup is successfully done.
GX_FAILURE	Device driver setup failed.
SSP_ERR_INVALID_CALL	Function call was made when the driver is not in SF_EL_GX_CONFIGURED state.
SSP_ERR_D2D_RENDERING	The D/AVE 2D returns error at opening a display list buffer
SSP_ERR_INVALID_ARGUEMENT	Invalid non-pointer (e.g. parameter) input
SSP_ERR_UNSUPPORTED	Specified color format is not supported
SSP_ERR_D2D_ERROR_INIT	The D/AVE 2D returns error at the initialization.

NOTE: Lower level drivers may return Common Error Codes. Refer to the SSP User's Manual API References for the associated module for a definition of all relevant status return values.

4.3.2.3 GUIX Synergy Port Framework Module and JPEG Decoder HAL Module Operational Overviews

GUIX Synergy Port Framework Module Overview

The figure below shows the components for a Synergy graphics solution and the flow of graphics data in this solution:



- Module initialization

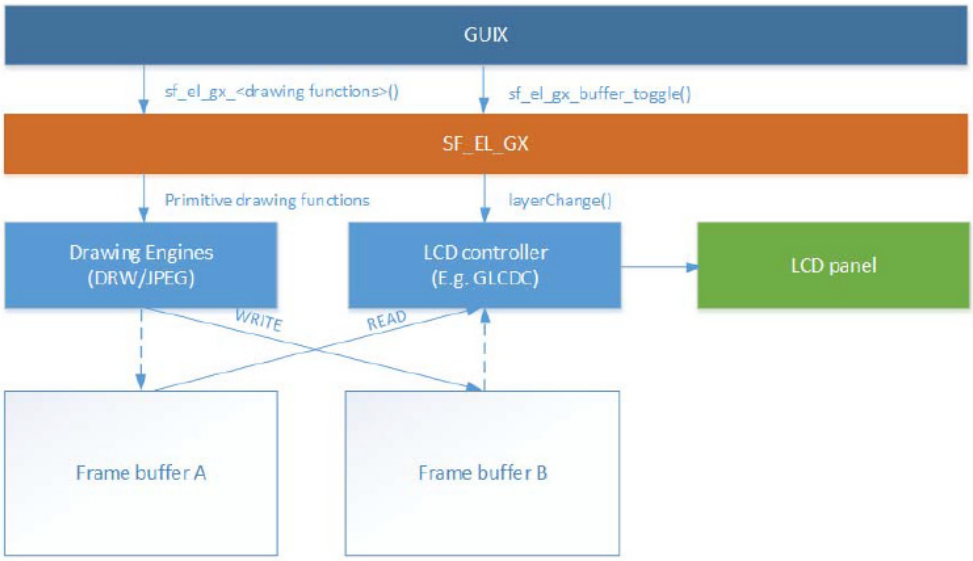
The SF_EL_GX supports the Synergy graphics hardware setup, which is required to run the GUIX system. The module has a dependency with Express Logic GUIX™ and GUIX Studio™ generated code. The GUIX system initialization needs to follow the sequence below as a general guidance.

1) 'open' SF_EL_GX module to initialize SF_EL_UX control block and pass module configurations.
2) Initialize GUIX Display object by GUIX Studio generated API gx_studio_display_configure. Through this API, SF_EL_GX 'setup' API is inputted to GUIX and Synergy graphics hardware setup will complete. Also, the root window initialized by GUIX is outputted to a user application.
3) Initialize the primary memory address of a GUIX Canvas Buffer by canvasInit API.
4) Create the root window by GUIX Studio generated API gx_studio_named_widget_create.
5) Show the root window by GUIX API gx_window_show API.
6) Start the GUIX system by GUIXAPI gx_system_start.

- Ping-Pong Frame Buffer Management

The SF_EL_GX module manages the buffer toggling operation in the graphics system with a ping-pong frame buffer. The figure below shows a ping-pong buffer graphics system managed by the SF_EL_GX module. The module uses GUIX and

the low-level display driver functions to draw an image (2D Drawing engine(DRW) or JPEG) and display the image (DISPLAY module, for example: GLCDC). A design with a single frame buffer is also possible in the SF_EL_GX configuration. However, it is better to use two frame buffers to avoid the tearing issue that could occur in a single frame buffer system.



Synergy 2D Drawing Engine Support

The module can draw graphics image accelerated by 2D Drawing engine (DRW) to get better graphics performance. Users can enable 2D Drawing engine with following configurations. The configuration is available through Synergy Configurator.

- Define GX_USE_SYNEGY_DRW (1) in gx_user.h
- Set DRW (SF_TES_2D_DRW) interrupt priority

Make sure to select "Renesas Synergy" in "Target CPU" setting in the "Configure Project" window and check "Enable Graphics Accelerator" in the "Synergy Advanced Settings" window on the Express Logic GUIX Studio (v5.2.8 or later). For the Pixelmap Output Format in the "Edit Pixel map" window (Right-click on pixelmap->edit settings. This will open the window), select "Compress Output". Do not select "Raw Format". This configuration allows the GUIX Studio to generate Targa RLE formatted encoded image resource data. The 2D Drawing engine hardware can read this format and decode and draw the image on the frame buffer.

Synergy JPEG Support

The module can draw graphics image accelerated by JPEG engine to get better graphics performance if a JPEG encoded image is used as a GUIX image resource. Users can enable JPEG engine with following configurations. The configuration is available through Synergy Configurator.

- Define GX_USE_SYNEGY_JPEG (1) in gx_user.h
- Set JPEG (R_JPEG_DECODE) interrupt priority

Make sure to select "Renesas Synergy" in "Target CPU" setting in the "Configure Project" window and select "Hardware JPEG Decoder" in the "Decoder Types JPEG:" drop-down menu on the Express Logic GUIX Studio (v5.2.8 or later). For the Pixelmap Output Format in the "Exit Pixel map" window (Right-click on pixelmap->edit settings. This will open the window), select "Raw Format". This configuration allows the GUIX Studio to generate raw JPEG encoded image resource data. JPEG hardware can read this format and decode and draw the image on the frame buffer.

GUIX Canvas Buffer

The module supports GUIX Canvas buffer which is used to rotate the graphics screen image. The size of GUIX Canvas Buffer must be exactly same as a frame buffer for the DISPLAY module. Note that, the use of a GUIX Canvas impacts to the graphics performance because of additional graphics image processing being required. Therefore, GUIX Canvas buffer should be only used if the screen rotation is required. Otherwise set NULL to p_canvas to let GUIX draw an image directly to frame buffers.

Screen Rotation

The module supports the screen rotation. The GUIX draws a screen image on a GUIX Canvas buffer as designed in the GUIX Studio, but SF_EL_GX module can perform the screen rotation and draw the rotated screen image on frame buffers. For example, a GUI screen designed as a landscape image in GUIX Studio can be rotated and displayed on a LCD panel with portrait shape dimension. Supported rotation angle is either of 90, 180, 270 degree in counter clockwise way and configured through open. Dynamic screen rotation is not supported. To enable the screen rotation feature, a GUIX Canvas Buffer must be used. GUIX draws screen update on a canvas first and then GUIX Port processes the screen copy to a frame buffer with rotating the image in counter clockwise way. If *Synergy 2D Drawing Engine Support* is enabled (GX_USE_SYNEGY_DRW = 1), the rotation is processed by 2D Drawing engine with texture mapping. If not enabled (GX_USE_SYNEGY_DRW = 0), the rotation is processed by software copy.

Note that, the configuration (rotation_angle = 0) and (p_canvas = non-NULL value) is allowed but should not be done. This configuration just consumes extra bus bandwidth for screen image copy from a GUIX Canvas buffer to frame buffers. Therefore, set NULL to p_canvas to not use a GUIX Canvas buffer.

Size of JPEG Work Buffer

The JPEG work buffer trades off the JPEG decode speed against the buffer size. When a widget on the screen is formatted in JPEG, the JPEG work buffer is used as a temporary storage memory to create the decoded image. If the buffer size is not large enough for decoding an entire image, JPEG decoding is performed in the output buffer streaming mode. BitBLT operation by 2D Drawing engine decodes a piece of JPEG raster image in the buffer, then transfers it to the frame buffer. The minimum size of JPEG work buffer is {(The number of pixels in the horizontal line) x (bpp (bytes per pixel) of the display format) x 8 (lines)}. For instance, if the decoded image is 800 pixels in a horizontal line and RGB565 format, the number is 800 x 2 x 8= 12 800 (byte). If the buffer size was smaller than this number, JPEG decoding will not be processed. To get better throughput, parameter "Size of the JPEG Work Buffer" should be set as much as larger because it improves the JPEG decode throughput. The JPEG output buffer streaming mode repeats partial JPEG decode operations and the repletion comes to be overhead.

D/AVE 2D buffer cache

The D/AVE 2D buffer cache can be enabled or disabled through following configuration in Synergy Configurator. Disable it when images with high resolution and 32 bit ARGB8888 color format are used.

- D/AVE 2D Frame Buffer Cache(Valid if D/AVE 2D Drawing Engine is enabled)

Screen Tearing in Single Buffer Designs

Screen tearing is a visual artifact in video display where a display device shows information from multiple frames in a single screen draw. In general, a system with a single frame buffer can cause the screen tearing issue on a LCD panel. The module allows users to have single frame buffer (set NULL to p_framebuffer_b) but does not care for the screen tearing. It is recommended to have a ping-pong frame buffer system to consist of two frame buffers.

- SF_EL_GX is only applicable for the Synergy MCU with GLCDC (mandatory), 2D Drawing engine or JPEG engine (optional)
- SF_EL_GX does not support a system with more than two frame buffers.
- SF_EL_GX supports only one GUIX canvas system.
- SF_EL_GX makes use of only one layer in DISPAY module.
- Do not access the TES D/AVE 2D module and TES D/AVE 2D Port module directly if GUIX uses the Modules.
- Do not access the JPEG Decode Framework module and JPEG Decode HAL module directly if GUIX uses the Modules.

- Refer to the most recent SSP Release notes for additional limitations when using this module.

JPEG Decoder HAL Module Operational Overview

The JPEG Decode HAL module is a high-level API for JPEG Decode processing implemented on r_jpeg. The module supports the Synergy JPEG Codec peripheral. It has the following features:

- Supports JPEG decompression.
- Supports polling mode that allows an application to wait for JPEG Decoder to complete.
- Supports interrupt mode with user-supplied callback functions.
- Configures parameters such as horizontal and vertical subsample values, horizontal stride, decoded pixel format, input and output data format, and color space.
- Obtains the size of the image prior to decoding it.
- Supports putting coded data in an input buffer and an output buffer to store the decoded image frame.
- Supports streaming coded data into JPEG Decoder module. This feature allows an application to read coded JPEG image from a file or from network without buffering the entire image.
- Configures the number of image lines to decode. This feature enables the application to process the decoded image on the fly without buffering the entire frame.
- Supports the input decoded format YCbCr444, YCbCr422, YCbCr420, YCbCr411.
- Supports the output format ARGB8888, RGB565.
- Returns error when the JPEG image's size, height and width don't meet the requirements.

The JPEG Decoder HAL module can be used in the Input Buffer Streaming mode or JPEG Output Buffer Streaming mode.

Input Buffer Streaming Mode Operational Description

In this scenario the JPEG image data resides on a file, or is received from network. The HAL-layer driver is able to handle this scenario without requiring the input data to be stored in memory first.

Output Buffer Streaming Mode Operational Description

In this scenario the application needs to write the decoded image data to a file or to a network. The HAL-layer driver does not require the application to allocate memory for the entire frame. Instead the application may choose to decode one or more lines at a time. With this feature the amount of memory needed for the output data is greatly reduced.

JPEG Decoder HAL Module Operational Notes

JPEG Decode Callbacks

A user callback function can be registered in the open API. If a user callback function is provided, the callback function will be called from the interrupt service routine (ISR) each time an interrupt happens. The argument of the callback function status can take the enumerated values listed below so that user can identify which event occurred in the decoding procedure.

Event Name	Event Condition
JPEG_DECODE_STATUS_ERROR JPEG	Decode module encountered an error.
JPEG_DECODE_STATUS_IMAGE_SIZE_READY JPEG	Decode obtained the image size of data to be decoded, and paused.

Event Name	Event Condition
JPEG_DECODE_STATUS_INPUT_PAUSE JPEG	Decode paused waiting for more input data.
JPEG_DECODE_STATUS_OUTPUT_PAUSE JPEG	Decode paused after decoded the number of lines specified by user.
JPEG_DECODE_STATUS_DONE JPEG	Decode operation has successfully completed.

NOTE: Since a user callback function is called from an ISR, be careful not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

- The JPEG Decode HAL module does not support JPEG Encode processing.
- Refer to the most recent SSP Release notes for the most up to date limitations for this module.

4.3.2.4 Including the GUIX Framework Module in an Application

This section describes how to include the GUIX Framework module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP based applications.

To add the GUIX Framework module to an application, simply add it to a thread using the Stacks Selection Sequence given in the table below. (The default name for the GUIX Framework is g_sf_el_gx0. This name can be changed in the associated **Properties** window.)

GUIX Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_el_gx0 GUIX Port on sf_el_gx	**Threads**	**New Stack> Framework> Graphics> GUIX Port on sf_el_gx**

When the GUIX Synergy Port Framework module on sf_el_gx is added to the thread stack as shown in the figure below, the configurator automatically adds any needed lower level modules. Any drivers that need additional configuration information will have box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower level drivers; these are either optional or recommended (this is indicated in the block with the inclusion of this text.) If the addition of lower level drivers is required, the module description will include "Add" in the text. Clicking on any Pink banded modules will bring up the "New" icon and then display the possible choices.

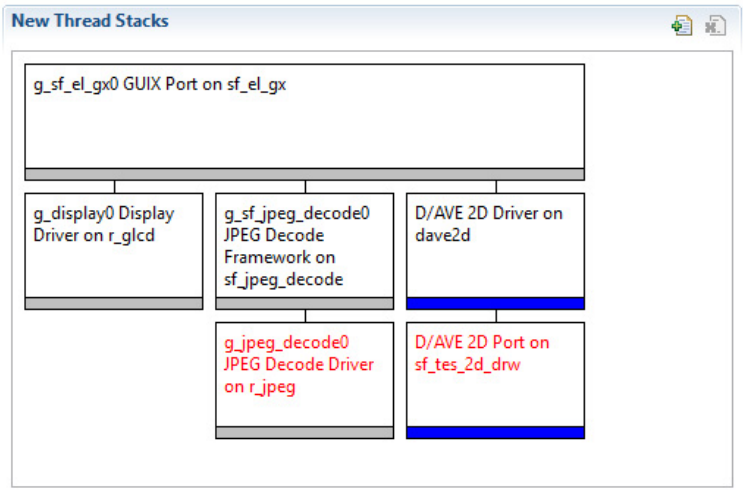


Figure 432: GUIX Synergy Port Framework Module Stack

4.3.2.5 Configuring the GUIX Synergy Port Framework Module

The GUIX Synergy Port Framework module must be configured by you for the desired operation. The SSP configuration window will automatically identify, by highlighting the block in red, any required configuration selections, such as Interrupts or operating modes, which must be configured for lower level modules, for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority. This configuration setting is available with the properties window of the associated module. Simply select the indicated module and then view the properties window. The Interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the Interrupt priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the below configuration properties tables, but is easily visible with the ISDE when configuring Interrupt Priority levels.

NOTE: You may want to open your ISDE and create the module and explore the property settings in parallel with looking over the Configuration Table Settings given below. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration for the GUIX Synergy Port Framework Module on GUIX on gx

ISDE Property	Value	Description
Enable Synergy 2D Drawing Engine Support	Yes, No Default: No	If Synergy 2D Drawing Engine (DRW) Support is enabled, the rotation is processed by Synergy DRW with texture mapping. If not enabled, the rotation is processed by software copy.
Enable Synergy JPEG Support	Yes, No Default: No	Enabling this places restrictions on JPEG image size, width modulus, format, output alignment, etc. It also limited to only decoding the JPEG image into the frame buffer. (The software decoder doesn't have any of those restrictions, but of course it uses more CPU time.)

Configuration for the GUIX Synergy Port Framework Module on sf_el_gx

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_sf_el_gx0	Name of SF_EL_GX instance which will be generated by ISDE. Specify the instance name of this module. Name must be a valid C symbol.
Name of Display Driver Run-time Configuration (Must be a valid symbol)	g_display0_runtime_cfg_bg	Specify the name of run-time configuration for the DISPLAY module you specified in the Synergy Configuration. Name must be a valid C symbol and NULL is not allowed to be set.
Name of Frame Buffer A (Must be a valid symbol)	g_display0_fb_background[0]	Specify the name of frame buffer. A DISPLAY module configuration in the Synergy Configuration contains the name of frame buffer to create. Set the name of frame buffer here. Name must be a valid C symbol and NULL is not allowed to be set.

ISDE Property	Value	Description
Name of Frame Buffer B (NULL allowed if consisting a single frame buffer system)	g_display0_fb_background[1]	Specify the name of another frame buffer. If you want to design your graphics system with a single frame buffer, set NULL to this parameter, or set same frame buffer name with parameter "Name of Frame Buffer A". See Tearing in Single Buffer Designs
Name of User Callback function	NULL	Name of User Callback function invoked by the Module when events happen. It must be a valid C symbol and NULL is allowed.
Screen Rotation Angle (Clockwise)	0, 90, 180, 270 Default: 0	Angle of screen rotation (degree). If non-zero value is selected, screen rotation is enabled and GUIX draws screen image on a frame buffer, rotating the image with the angle in the counter clockwise way.
GUIX Canvas Buffer (required if rotation angle is not zero)	Not used; Used Default: Not used	If enabling the screen rotation, a canvas buffer must be used. The size of canvas buffer must be exactly the same as a frame buffer for the display module.
Size of JPEG Work Buffer (valid if JPEG hardware acceleration enabled)	768000	The JPEG work buffer size in bytes. Value must be a valid integer value and zero is allowed to be set if JPEG acceleration is not used. Larger buffer size shortens the drawing time. See Size of JPEG Work Buffer
Memory section for GUIX Canvas Buffer	sdram, bss, … Default: sdram	Name of memory section where you want to allocate the GUIX Canvas Buffer. Enter a valid section name defined in the linker script file. Name must be a valid C symbol.
Memory section for JPEG Work Buffer	Sdram, bss, … Default: sdram	Name of memory section where you want to allocate the JPEG Work Buffer. Enter a valid section name defined in the linker script file. Name must be a valid C symbol.

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2 Family. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower level modules can be desirable. For example, the it might be useful

to select different screen sizes or input formats. The configurable properties for the lower level stack modules are given in the below sections for completeness and as a reference.

NOTE: Most of the property settings for lower level modules are fairly intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the GUIX Synergy Port Framework Module Low Level Drivers

Typically, only a small number of settings must be modified from the default for lower level drivers and these are indicated with red text in the Thread Stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The table below identifies all the settings within the properties section for the module.

Configuration for the GLCD HAL Module on r_glcd

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_display0	The name to be used for a GLCDC module control block instance. This name is also used as the prefix of the other variable instances.
Name of display callback function to be defined by user	NULL	Name must be a valid C symbol.
Input - Panel clock source select	Internal clock(GLCDCLK), External clock(LCD_EXTCLK) Default: Internal clock (GLCDCLK)	Choose the panel clock source depends on your system.
Input - Graphics screen1	Used, Not used Default: Used	Specify "Used" if the graphics screen N is used. Then the frame buffer named "display_fb_background" for graphics screen1 and "display_fb_foreground" for graphics screen2 is auto-generated by ISDE. If not using either of the graphics screens, specify "Not used". Then the frame buffer is not created. Note that there is no memory read access to the frame buffer when you specify "Not used", which reduces the consumption of bus bandwidth.
Input - Graphics screen1 frame buffer name	fb_background	Custom name for frame buffer.

ISDE Property	Value	Description
Input - Number of Graphics screen1 frame buffer	2	Number of graphics selection.
Input - section where Graphics screen1 frame buffer allocated	sdram	Specify the section name to allocate the frame buffer. This is valid if "Input - Graphics screen1" is set as "Used."
Input - Graphics screen1 input horizontal size	800	Specify the number of horizontal pixels. Default value is the size for an image with 800x480 pixels
Input - Graphics screen1 vertical size	480	Specify the number of vertical pixels. Default value is the size for an image with 800x480 pixels.
Input - Graphics screen1 input horizontal stride (not bytes but pixels)	800	Specify the memory stride for a horizontal line. This value must be specified with the number of pixels, not actual bytes. Typically this parameter is set to same number as parameter 'input horizontal size'. Default value is the size for an image with 800x480 pixels.
Input - Graphics screen1 input format	32bits ARGB888, 32bits RGB888, 16bits RGB565, 16bits ARGB1555, 16bits ARGB4444, CLUT 8, CLUT 4, CLUT 1 Default: 16bits RGB565	Specify the graphics screen Input format. If selecting CLUT formats, you must write CLUT data using clut before performing start. Default setting supports a RGB565 formatted image.
Input - Graphics screen1 input line descending	Used, Not used Default: Not used	Specify "On" if image data descends from the bottom line to the top line in the frame buffer. Usually "Off".
Input - Graphics screen1 input line repeat	On, Off Default: Off	Specify "On" if expecting to repeatedly read a raster image which is smaller than the LCD panel size. Usually "Off". For details, see the description of Line Repeating function.
Input - Graphics screen1 input line repeat times	0	Specify the number of repeating times for a raster image which is read repeatedly in a frame.

ISDE Property	Value	Description
Input - Graphics screen1 layer coordinate X	0	Specify the horizontal offset in pixels of the graphics screen from the background screen.
Input - Graphics screen1 layer coordinate Y	0	Specify the vertical offset in pixels of the graphics screen from the background screen.
Input - Graphics screen1 layer background color alpha	255	Based on the alpha value, either the graphics screen2 (foreground graphics screen) is blended into the graphics screen1 (background graphics screen) or the graphics screen1 is blended into the monochrome background screen.
Input - Graphics screen1 layer background color Red	255	Specify the background color in the graphics screen N.
Input - Graphics screen1 layer background color Green	255	Specify the background color in the graphics screen N.
Input - Graphics screen1 layer background color Blue	255	Specify the background color in the graphics screen N.
Input - Graphics screen1 layer fading control	None, Fade-in, Fade-out Default: None	Specify "On" when performing a fade-in for the graphics screen. The transparent screen changes gradually to opaque. Specify "Off" when performing the fade-out for the graphics screen. The opaque screen changes gradually to transparent. Note that this processing is accelerated by the GLCDC hardware and cannot stop once started. The transition status can be monitored by statusGet.
Input - Graphics screen1 layer fade speed	0	Specify the number of frames for the fading transition to complete.

ISDE Property	Value	Description
Input - Graphics screen2	Used, Not used Default: Not used	Specify "Used" if the graphics screen N is used. Then the frame buffer named "display_fb_background" for graphics screen1 and "display_fb_foreground" for graphics screen2 is auto-generated by ISDE. If not using either of the graphics screens, specify "Not used". Then the frame buffer is not created. Note that there is no memory read access to the frame buffer when you specify "Not used", which reduces the consumption of bus bandwidth.
Input - Graphics screen2 frame buffer name	fb_foreground	Custom name for frame buffer.
Input - Number of Graphics screen2 frame buffer	2	Number of graphics selection.
Input - section where Graphics screen2 frame buffer allocated	sdram	Specify the section name to allocate the frame buffer. This is valid if "Input - Graphics screen1" is set as "Used."
Input - Graphics screen2 input horizontal size	800	Specify the number of horizontal pixels. Default value is the size for an image with 800x480 pixels
Input - Graphics screen2 vertical size	480	Specify the number of vertical pixels. Default value is the size for an image with 800x480 pixels.
Input - Graphics screen2 input horizontal stride (not bytes but pixels)	800	Specify the memory stride for a horizontal line. This value must be specified with the number of pixels, not actual bytes. Typically this parameter is set to same number as parameter 'input horizontal size'. Default value is the size for an image with 800x480 pixels.
Input - Graphics screen2 input format	32bits ARGB888, 32bits RGB888, 16bits RGB565, 16bits ARGB1555, 16bits ARGB4444, CLUT 8, CLUT 4, CLUT 1 Default: 16bits RGB565	Specify the graphics screen Input format. If selecting CLUT formats, you must write CLUT data using clut before performing start. Default setting supports a RGB565 formatted image.

ISDE Property	Value	Description
Input - Graphics screen2 input line descending	On, Off Default: Off	Specify "On" if image data descends from the bottom line to the top line in the frame buffer. Usually "Off".
Input - Graphics screen2 input line repeat	On, Off Default: Off	Specify "On" if expecting to repeatedly read a raster image which is smaller than the LCD panel size. Usually "Off". For details, see the description of Line Repeating function.
Input - Graphics screen2 input line repeat times	0	Specify the number of repeating times for a raster image which is read repeatedly in a frame.
Input - Graphics screen2 layer coordinate X	0	Specify the horizontal offset in pixels of the graphics screen from the background screen.
Input - Graphics screen2 layer coordinate Y	0	Specify the vertical offset in pixels of the graphics screen from the background screen.
Input - Graphics screen2 layer background color alpha	255	Based on the alpha value, either the graphics screen2 (foreground graphics screen) is blended into the graphics screen1 (background graphics screen) or the graphics screen1 is blended into the monochrome background screen.
Input - Graphics screen2 layer background color Red	255	Specify the background color in the graphics screen N.
Input - Graphics screen2 layer background color Green	255	Specify the background color in the graphics screen N.
Input - Graphics screen2 layer background color Blue	255	Specify the background color in the graphics screen N.

ISDE Property	Value	Description
Input - Graphics screen2 layer fading control	None, Fade-in, Fade-out Default: None	Specify "On" when performing a fade-in for the graphics screen. The transparent screen changes gradually to opaque. Specify "Off" when performing the fade-out for the graphics screen. The opaque screen changes gradually to transparent. Note that this processing is accelerated by the GLCDC hardware and cannot stop once started. The transition status can be monitored by statusGet.
Input - Graphics screen2 layer fade speed	0	Specify the number of frames for the fading transition to complete.
Output - Horizontal total cycles	1024	Specify the total cycles in a horizontal line. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Horizontal active video cycles	800	Specify the number of active video cycles in a horizontal line. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Horizontal back porch cycles	46	Specify the number of back porch cycles in a horizontal line. Back porch starts from the beginning of Hsync cycles, which means back porch cycles contain Hsync cycles. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Horizontal sync signal cycles	20	Specify the number of Hsync signal assertion cycles. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches LCD panel on S7G2 PE-HMI1 board.

ISDE Property	Value	Description
Output - Horizontal sync signal polarity	Low active, High active Default: Low active	Select the polarity of Hsync signal to match your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical total lines	525	Specify number of total lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical active video lines	480	Specify the number of active video lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical back porch lines	23	Specify the number of back porch lines in a frame. Back porch starts from the beginning of Vsync lines, which means back porch lines contain Vsync lines. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical sync signal lines	10	Specify the Vsync signal assertion lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical sync signal polarity	Low active, High active Default: Low active	Select the polarity of Vsync signal to match to your system. Default setting matches LCD panel on S7G2 PE-HMI1 board.
Output - Format	24bits RGB888, 18bits RGB666, 16bits RGB565, 8bits serial Default: 24bits RGB888	Specify the graphics screen output format to match to your LCD panel. Default setting matches the LCD panel on S7G2 PE-HMI1 board.

ISDE Property	Value	Description
Output - Endian	Little endian, Big endian Default: Little endian	Select data endian for output signal to LCD panel. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Color order	RGB, BGR Default: RGB	Select data order for output signal to LCD panel. The order of blue and red can be swapped if needed. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Data Enable Signal Polarity	Low active, High active Default: High active	Select the polarity of Data Enable signal to match to your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Sync edge	Rising Edge, Falling Edge Default: Rising Edge	Select the polarity of Sync signals to match to your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Background color alpha channel	255	Specify the background color of the background screens.
Output - Background color R channel	0	Specify the background color of the background screens.
Output - Background color G channel	0	Specify the background color of the background screens.
Output - Background color B channel	0	Specify the background color of the background screens.
CLUT	Used, Not used Default: Not used	Specify "Used" if selecting CLUT formats for a graphics screen input format. Then, a buffer named "CLUT_buffer" for the CLUT source data is generated in the ISDE auto-generated source file.
CLUT - CLUT buffer size	256	Specify the number of entries for the CLUT source data buffer. Each entries consumes 4 bytes (1 word). Words of CLUT source data specified by this parameter are generated in the ISDE auto-generated source file.

ISDE Property	Value	Description
TCON - Hsync pin select	Not used, LCD_TCON0, LCD_TCON1, LCD_TCON2, LCD_TCON3 Default: LCD_TCON0	Select the TCON pin used for the Hsync signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON - Vsync pin select	Not used, LCD_TCON0, LCD_TCON1, LCD_TCON2, LCD_TCON3 Default: LCD_TCON1	Select TCON pin used for Vsync signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON - DataEnable pin select	Not used, LCD_TCON0, LCD_TCON1, LCD_TCON2, LCD_TCON3 Default: LCD_TCON2	Select TCON pin used for DataEnable signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON - Panel clock division ratio	1/1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/12, 1/16, 1/24, 1/32 Default: 1/8	Select the clock source divider value. See the note at bottom of this table about the source clock for the pixel clock.
Color correction - Brightness	Off, On Default: Off	Specify "On" when performing brightness control. If specifying "Off", the setting below does not affect the output color.
Color correction - Brightness R channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each of R, G, B channels.
Color correction - Brightness G channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each of R, G, B channels.
Color correction - Brightness B channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each of R, G, B channels.

ISDE Property	Value	Description
Color correction - Contrast	Off, On Default: Off	Specify "On" when performing contrast control. If specifying "Off", the setting below does not affect the output color.
Color correction - Contrast(gain) R channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set the value for each of R, G, B channels.
Color correction - Contrast(gain) G channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set the value for each of R, G, B channels.
Color correction - Contrast(gain) B channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set the value for each of R, G, B channels.
Color correction - Gamma correction(Red)	Off, On Default: Off	Control for each channel R/G/B. Specify "On" when performing gamma correction for the red channel. If specifying "Off", the settings for gain and threshold do not affect the output color.
Color correction - Gamma gain R[0-15]	0	Set the gain value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma threshold R[0-15]	0	Set the threshold value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as follows: Output color level = Input color level / 1024 (/128).

ISDE Property	Value	Description
Color correction - Gamma correction(Green)	Off, On Default: Off	Control for each channel R/G/B. Specify "On" when performing gamma correction for the red channel. If specifying "Off", the settings for gain and threshold do not affect the output color.
Color correction - Gamma gain G[0-15]	0	Set the gain value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma threshold G[0-15]	0	Set the threshold value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma correction(Blue)	Off, On Default: Off	Control for each channel R/G/B. Specify "On" when performing gamma correction for the red channel. If specifying "Off", the settings for gain and threshold do not affect the output color.
Color correction - Gamma gain B[0-15]	0	Set the gain value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as follows: Output color level = Input color level / 1024 (/128).

ISDE Property	Value	Description
Color correction - Gamma threshold B[0-15]	0	Set the threshold value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Dithering	Off, On Default: Off	Dithering enable. Specify "On" when applying the dither effect to reduce the banding in case of selecting RGB666 or RGB565 output formats. Dithering can be applied when converting. If specified "Off", the settings for dithering below do not affect the output. For details on the dither effect, see Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Mode	Truncate, Round off, 2x2 Pattern Default: Truncate	Specify the dither mode. For detail, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Pattern A	Pattern 00, Pattern 01, Pattern 10, Pattern 11 Default: Pattern 11	Specify the dither pattern for 2X2 pattern mode. Forndetails, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Pattern B	Pattern 00, Pattern 01, Pattern 10, Pattern 11 Default: Pattern 11	Specify the dither pattern for 2X2 pattern mode. Forndetails, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Pattern C	Pattern 00, Pattern 01, Pattern 10, Pattern 11 Default: Pattern 11	Specify the dither pattern for 2X2 pattern mode. Forndetails, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.

ISDE Property	Value	Description
Dithering - Pattern D	Pattern 00, Pattern 01, Pattern 10, Pattern 11 Default: Pattern 11	Specify the dither pattern for 2X2 pattern mode. Forndetails, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Misc - Correction Process Order	Brightness and Contrast then Gamma, Gamma then Brightness and Contrast Default: Brightness and Contrast then Gamma	Specify the color correction processing order if needed.
Line Detect Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	The driver needs valid interrupt priority setting and it won't work if disabled.
Underflow 1 Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	The driver needs valid interrupt priority setting and it won't work if disabled.
Underflow 2 Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	The driver needs valid interrupt priority setting and it won't work if disabled.

Configuration for the JPEG Decode Framework Module on sf_jpeg_decode

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_sf_jpeg_decode0	The name to be used for a JPEG Decode Framework module instance.

Configuration for the JPEG Decode HAL Module on r_jpeg

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_jpeg_decode0	The name to be used for a JPEG Decode module instance.

ISDE Property	Value	Description
Byte Order for Input Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2) Default: Normal Byte order	Specify the byte order for input data. The order is swapped as specified in every 8-byte.

ISDE Property	Value	Description
Byte Order for Output Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2) Default: Normal Byte order	Specify the byte order for output data. The order is swapped as specified in every 8-byte.
Output Data Color Format	Pixel Data RGB565 format, Pixel Data ARGBB888 format Default: Pixel Data RGB565 format	Specify the output data format.
Alpha value to be applied to decoded pixel data (only valid for ARGB8888 format)	255	Specify the alpha value for the output data format (only valid for ARGB8888 format).
Name of user callback function	NULL	Specify the name of user callback function.

ISDE Property	Value	Description
Decompression Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Decompression interrupt priority selection.
Data Transfer Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Data transfer interrupt priority selection.

Table 6: Configuration for the D/AVE 2D Driver on dave2d

ISDE Property	Value	Description
No configurable settings		

Configuration for the D/AVE 2D Port on sf_tes_2d_drw

ISDE Property	Value	Description
Work memory size for display lists in bytes	32768	Work memory size for display lists selection
DRW Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	DRW INT selection

NOTE: The above setting examples and defaults are for a project using the Synergy S7G2 Family. Other MCUs may have

different default values and available configuration settings.

GUIX Synergy Port Framework Module Clock Configuration

The GUIX Synergy Port Module is a logical module and therefore does not require any hardware setting except setting the ARM Cortex-M core SysTick timer.

GUIX Synergy Port Framework Module Pin Configuration

The GUIX Synergy Port Module is a logical module and therefore does not require pin settings.

4.3.2.6 Using the GUIX Framework Module or JPEG Decode HAL Module in an Application

The typical steps in using the GUIX Framework module in an application are (assuming the instance of the GUIX driver is g_sf_el_ux0 and the instance of the SPI driver for the LCD is g_rspi_lcdc):

Step 0. Initialize GUIX with the gx_system_initialize function

Step 1. Initialize GUIX drivers using the open API (g_sf_el_ux0.p_api -> open)

Step 2. Configure the GUIX system with the gx_studio_display_configure function using the setup API

Step 3. Initialize the memory address of the canvas with the canvasInit API (g_sf_el_ux0.p_api -> canvasInit)

Step 4. Create the primary screen with the gx_studio_named_widget_create function

Step 5. Show the root window using the gx_widget_show function

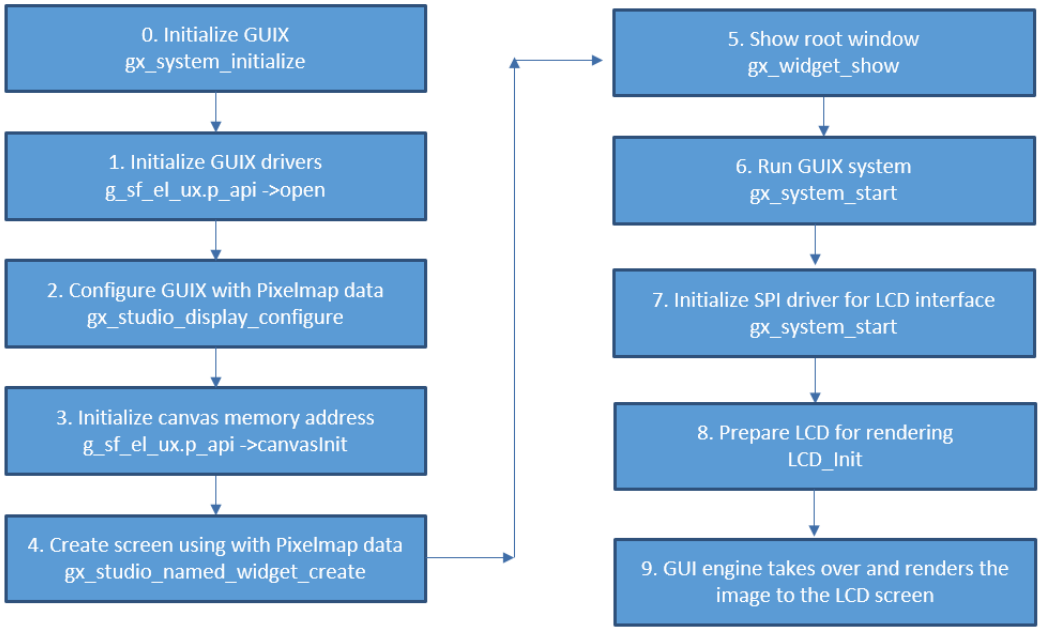
Step 6. Start the GUIX system with the gx_system_start function

Step 7. Initialize the SPI interface with the LCD display with the open API (g_rspi_lcdc.p_api -> open)

Step 8. Initialize the LCD hardware with the LCD_Init call.

Step 9. GUI engine takes over and renders the image on the LCD screen

These common steps are illustrated in a typical operational flow diagram in the following figure:



Once the JPEG Decode HAL module has been configured and the files generated, the JPEG Decode is ready to be used in an application. The typical steps in using the JPEG Decoder HAL module in an application are initializing the JPEG Decode using the open API, configure the horizontal stride, image sub-sample, input buffer and output buffer, once the input and output buffers are set, JPEG codec trigger the decode operation and store the decoded image to the output buffer, statusGet API can be used to poll the status of JPEG operation.

The typical steps in using the JPEG Decode HAL module in an application are:

Step 1. Initialize the JPEG Decode HAL module using open API.

Step 3. Set vertical and horizontal image sub-sample using imageSubsampleSet API.

Step 4. Set the input buffer address (which contains the JPEG image) using inputBufferSet API.

Step 5. Set the output buffer (should be large enough to hold the raw image data) using outputBufferSet API.

Step 6. statusGet API can be used to get the JPEG operation, statusGet API return an enumerated value (described above) to notify user. Status JPEG_DECODE_STATUS_DONE from statusGet API shows that the Decode operation is complete.

Step 7. Operate on the received raw image data as needed by the application.

These common steps are illustrated in a typical operational flow diagram in the following figure:

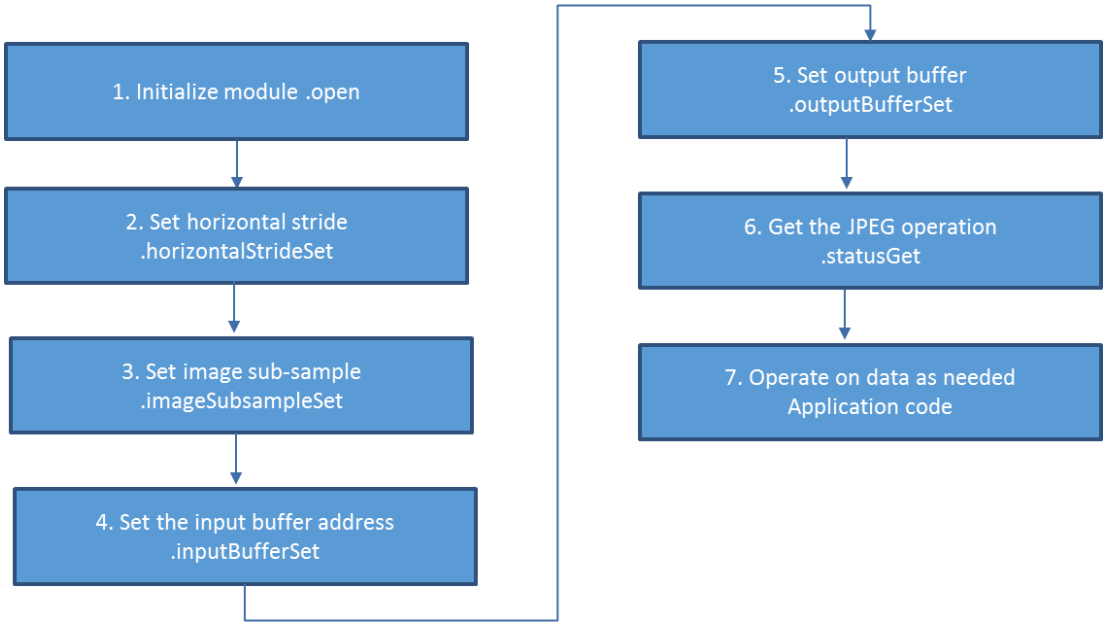


Figure 433: Flow Diagram of a Typical JPEG Decode HAL Module Application

4.3.3 NetX

The Express Logic NetX networking stack (nx) is integrated into the SSP. For more information about NetX, including API reference, refer to the *NetX User Guide*.

4.3.3.1 What Does the Express Logic NetX Module Do?

NetX is a high-performance real-time implementation of the TCP/IP standards designed exclusively for embedded ThreadX-based applications.

4.3.3.2 Using e^2 studio to write an Application with Express Logic NetX

The Express Logic NetX module is integrated into the Synergy Software Package in e^2 studio:e^2 studio ISDE User Guide.

In e^2 studio, create and configure a project and add the drivers:

1) Create the project: Creating a Project.
2) Configure the project: Configuring a Project.
3) Add the drivers: Adding Drivers to a Thread and Configuring the Drivers.

During the configuration of the project, add the following modules in the ISDE: Adding Drivers to a Thread and Configuring the Drivers.

Resource	ISDE Tab	Selection
Express Logic NetX (mandatory)	Threads	**Framework > Networking > NetX on UX**
NetX Driver (mandatory)	Threads	**Framework > Networking > NetX Port <ETHER> on sf_el_nx**
Express Logic NetX Source Code (optional, for advanced configurations or source level debugging support)	Threads	**Framework > Networking > NetX Source**

Adding NetX Application Layer Modules

To add NetX Application Layer modules, go to the **Components** tab of the Synergy Configuration tool (configuration.xml), and select any of the application layer modules:

- nx_auto_ip: Refer to the *NetX Auto IP User Guide*.
- nx_bsd: Refer to the *NetX BSD User Guide*. See Express Logic NetX Limitations section.
- nx_dhcp: Refer to the *NetX Dynamic Host Configuration Protocol for Clients User Guide*.
- nx_dhcp_server: Refer to the *NetX Dynamic Host Configuration Protocol for Servers (NetX DHCP Server) User Guide*.
- nx_dns: Refer to the *NetX DNS (Domain Name System) Client User Guide*.
- nx_ftp_client: Refer to the *NetX File Transfer Protocol (FTP) User Guide*.
- nx_ftp_server: Refer to the *NetX File Transfer Protocol (FTP) User Guide*.
- nx_http_client: Refer to the *NetX Hypertext Transfer Protocol (HTTP) User Guide*.
- nx_http_server: Refer to the *NetX Hypertext Transfer Protocol (HTTP) User Guide*.
- nx_pop3: Refer to the *NetX Post Office Protocol Version 3 Client (NetX POP3 Client) User Guide*.
- nx_ppp: Refer to the *NetX Point-to-Point Protocol (PPP) User Guide*.
- nx_smtp: Refer to the *NetX Simple Mail Transfer Protocol for Clients (NetX SMTP Client) User Guide*.
- nx_sntp: Refer to the *NetX Simple Network Time Protocol (SNTP) Client User Guide*.
- nx_telnet_client: Refer to the *NetX Telnet Protocol (Telnet) User Guide*.
- nx_telnet_server: Refer to the *NetX Telnet Protocol (Telnet) User Guide*.
- nx_tftp_client: Refer to the *NetX Trivial File Transfer Protocol (TFTP) User Guide*.
- nx_tftp_server: Refer to the *NetX Trivial File Transfer Protocol (TFTP) User Guide*.

When the components above are added, a prebuilt library of the application code is added. For each component listed above, there is an analogous component ending in '_src' that contains protected source files. The '_src' component can be added in addition to the prebuilt library module. Do not add the '_src' component without the prebuilt library module.

4.3.3.3 Express Logic NetX Configuration Notes

NetX Source Advanced Configurations

If the NetX Source module is added to the project, the Properties window provides advanced configurations for the NetX source library. Highlight a configuration option to view a description of the option in the bottom left corner of the e^2 studio GUI. If the configuration option is empty, the default value is used. The default values of configuration options are defined in ssp/inc/framework/el/nx/nx_port.h. Refer to the Configuration Options chapter of the *NetX User Guide* for more information.

4.3.3.4 Express Logic NetX Limitations

- Express Logic NetX cannot be used in the same application as Express Logic NetX Duo. Only one or the other can be used per application.
- When using NetX BSD with the GCC compiler, build with the macro _POSIX_SOURCE defined to avoid compilation errors.
- See SSP Release Notes for known limitations.

4.3.3.5 Express Logic NetX Supported Devices

The module is designed to support the following families:

- S7G2

4.3.4 NetX Duo

The Express Logic NetX Duo networking stack (nxd) is integrated into the SSP. For more information about NetX Duo, including API reference, refer to the *NetX Duo User Guide*.

4.3.4.1 What Does the Express Logic NetX Duo Module Do?

NetX Duo is a high-performance real-time implementation of the TCP/IP standards designed exclusively for embedded ThreadX-based applications.

4.3.4.2 Using e^2 studio to write an Application with Express Logic NetX Duo

The Express Logic NetX Duo module is integrated into the Synergy Software Package in e^2 studio:e^2 studio ISDE User Guide.

In e^2 studio, create and configure a project and add the drivers:

1) Create the project: Creating a Project.
2) Configure the project: Configuring a Project.
3) Add the drivers: Adding Drivers to a Thread and Configuring the Drivers.

During the configuration of the project, add the following modules in the ISDE: Adding Drivers to a Thread and Configuring the Drivers.

Resource	ISDE Tab	Selection
Express Logic NetX Duo (mandatory)	Components	**Express Logic > all > nxd**
NetX Driver (mandatory)	Threads	**Framework > Networking > NetX Port <ETHER> on sf_el_nx**
Express Logic NetX Duo Source Code (optional, for advanced configurations or source level debugging support)	Components	**Express Logic > all > nxd_src**

Adding NetX Duo Application Layer Modules

To add NetX Duo Application Layer modules, go to the **Components** tab of the Synergy Configuration tool (configuration.xml), and select any of the application layer modules:

- nxd_auto_ip: Refer to the *NetX Duo Auto IP User Guide*.
- nxd_bsd: Refer to the *NetX Duo BSD User Guide*. See Express Logic NetX Duo Limitations section.
- nxd_dhcp: Refer to the *NetX Duo Dynamic Host Configuration Protocol for Clients User Guide*.
- nxd_dhcp_server: Refer to the *NetX Duo Dynamic Host Configuration Protocol for Servers (NetX Duo DHCP Server) User Guide*.
- nxd_dns: Refer to the *NetX Duo DNS (Domain Name System) Client User Guide*.
- nxd_ftp_client: Refer to the *NetX Duo File Transfer Protocol (FTP) User Guide*.
- nxd_ftp_server: Refer to the *NetX Duo File Transfer Protocol (FTP) User Guide*.
- nxd_http_client: Refer to the *NetX Duo Hypertext Transfer Protocol (HTTP) User Guide*.
- nxd_http_server: Refer to the *NetX Duo Hypertext Transfer Protocol (HTTP) User Guide*.
- nxd_pop3: *Refer to the NetX Duo Post Office Protocol Version 3 Client (NetX Duo POP3 Client) User Guide*.
- nxd_ppp: Refer to the *NetX Duo Point-to-Point Protocol (PPP) User Guide*.
- nxd_smtp: Refer to the *NetX Duo Simple Mail Transfer Protocol for Clients (NetX Duo SMTP Client) User Guide*.
- nxd_sntp: Refer to the *NetX Duo Simple Network Time Protocol (SNTP) Client User Guide*.
- nxd_telnet_client: Refer to the *NetX Duo Telnet Protocol (Telnet) User Guide*.
- nxd_telnet_server: Refer to the *NetX Duo Telnet Protocol (Telnet) User Guide*.
- nxd_tftp_client: Refer to the *NetX Duo Trivial File Transfer Protocol (TFTP) User Guide*.
- nxd_tftp_server: Refer to the *NetX Duo Trivial File Transfer Protocol (TFTP) User Guide*.

When the components above are added, a prebuilt library of the application code is added. For each component listed above, there is an analogous component ending in '_src' that contains protected source files. The '_src' component can be added in addition to the prebuilt library module. Do not add the '_src' component without the prebuilt library module.

4.3.4.3 Express Logic NetX Duo Limitations

- Express Logic NetX Duo cannot be used in the same application as Express Logic NetX. Only one or the other can be used per application.
- When using NetX Duo BSD with the GCC compiler, build with the macro _POSIX_SOURCE defined to avoid compilation errors.
- See SSP Release Notes for known limitations.

4.3.4.4 Express Logic NetX Duo Supported Devices

The module is designed to support the following families:

- S7G2

4.3.5 NetX Duo MQTT Client

The NetX Duo MQTT Client module provides high-level APIs for a Message Queuing Telemetry Transport (MQTT) protocol-based client. The MQTT protocol works on top of TCP/IP and therefore the MQTT client is implemented on top of NetX Duo IP and NetX Duo Packet Pool. NetX Duo IP attaches itself to the appropriate link layer driver such as Ethernet, Wi-Fi or cellular. The MQTT client can optionally connect to an MQTT server over a secure connection and in such case, it uses the service provided by NetX Duo TLS Common.

4.3.5.1 NetX Duo MQTT Client Module Features

- Compliant with OASIS MQTT Version 3.1.1 Oct 29th, 2014. The specification can be found at: http://mqtt.org/
- Provides option to enable/disable TLS for secure communications using NetX Duo Secure in SSP
- Supports Quality of Service (QoS) levels 0 and 1 and provides the ability to choose the levels that can be selected while publishing the message
- Internally buffers and maintains queue of received messages
- Provides mechanism to register callback when a new message is received
- Provides mechanism to register callback when the connection with the broker is terminated.

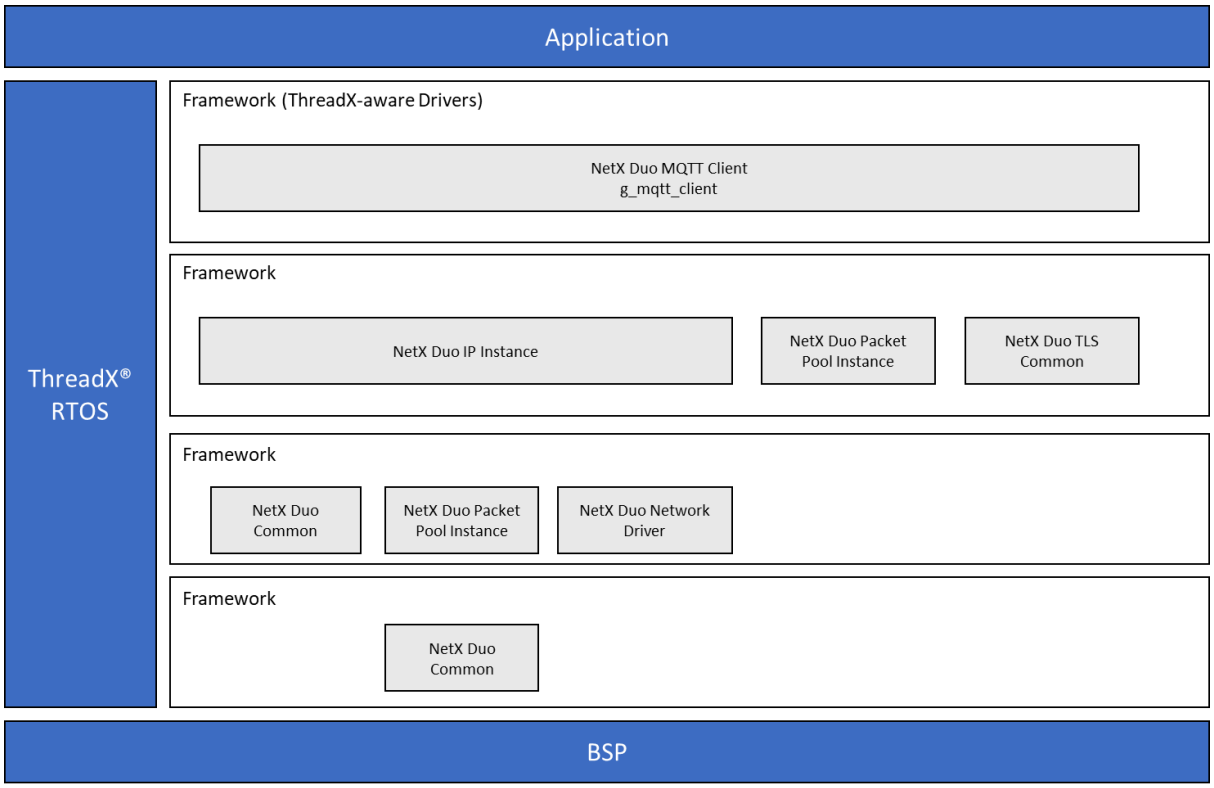


Figure 434: NetX Duo MQTT Client Module Block Diagram

4.3.5.2 NetX Duo MQTT Client Module APIs Overview

A complete list of MQTT APIs, parameters and return values is available from the Synergy Gallery in the X-Ware Component Documents for Renesas Synergy. Here is an overview of the available APIs:

API	Function
nxd_mqtt_client_create	Create MQTT client instance
nxd_mqtt_client_client_login_set	Set MQTT client login username and password
nxd_mqtt_client_connect	Connect MQTT Client to the broker
nxd_mqtt_client_secure_connect	Connect MQTT client to the broker with TLS security
nxd_mqtt_client_publish	Publish a message through the broker.
nxd_mqtt_client_subscribe	Subscribe to a topic
nxd_mqtt_client_unsubscribe	Unsubscribe from a topic

API	Function
nxd_mqtt_client_receive_notify_set	Set MQTT message receive notify callback function
nxd_mqtt_client_message_get	Retrieve a message from the broker
nxd_mqtt_client_disconnect_notify_set	Set MQTT message disconnect notify callback function
nxd_mqtt_client_disconnect	Disconnect MQTT client from the broker
nxd_mqtt_client_delete	Delete the MQTT client instance

4.3.5.3 NetX Duo MQTT Client Module Operational Overview

MQTT (Message Queue Telemetry Transport) is based on a publisher/subscriber model. A client can publish information to other clients through a broker. A client, if interested in a topic, can subscribe to the topic through the broker. A broker is responsible for delivering published messages to its clients who subscribe to the topic. In this publisher/subscriber model, multiple clients may publish data with the same topic. A client will receive a message it publishes if the client subscribes to the same topic.

The following figure provides an overview of the MQTT Client publish/subscribe model

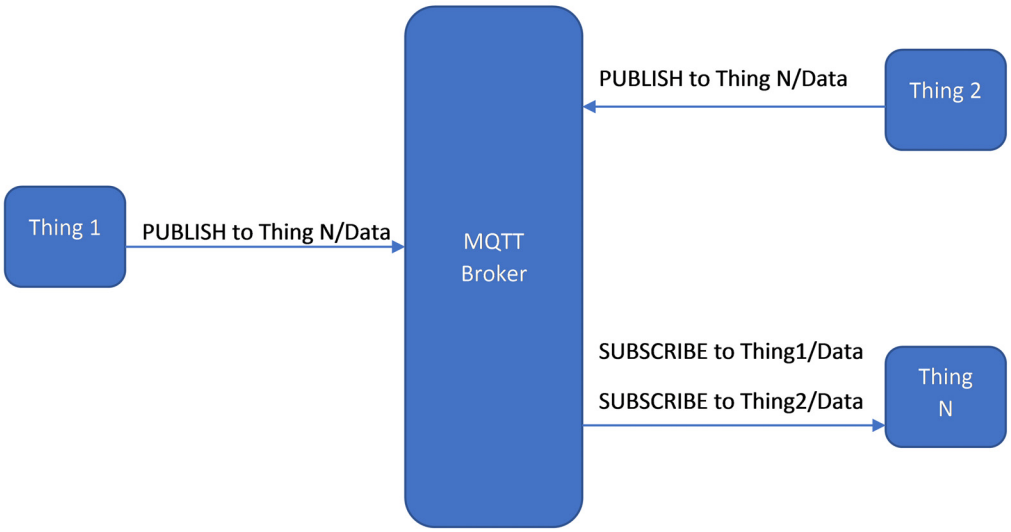


Figure 435: MQTT Client Publish/Subscribe Model

The NetX Duo MQTT client module can be used in the normal mode or the secure mode.

NetX Duo MQTT client module Normal Mode Operational Description

In normal mode, the communication between the MQTT client and broker is not secure.

NetX Duo MQTT client module Secure Mode Operational Description

In Secure mode, the communication between the MQTT client and broker is secured using the TLS protocol. In the thread pane, the TLS protocol is represented by "Add NetX Duo TLS common \[Optional\]" block as shown in the figure below.

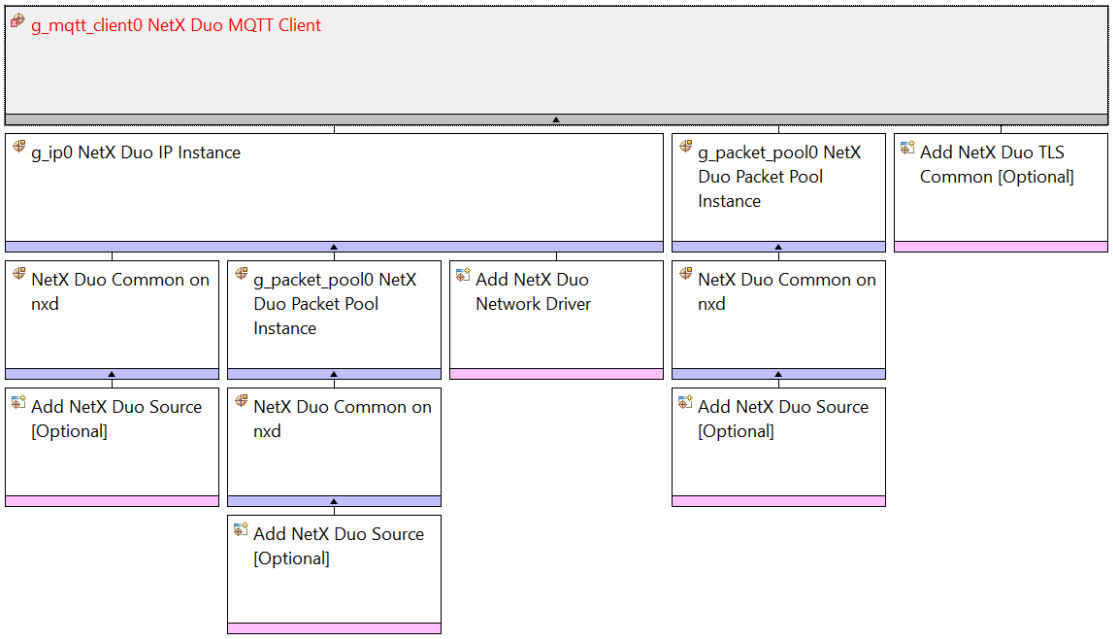


Figure 436: NetX Duo MQTT Client Component thread-pane view

Adding the NetX Duo TLS Common block enables TLS support and internally defines the NX_SECURE_ENABLE macro. The figure below shows the thread pane view of the MQTT client with TLS support enabled.

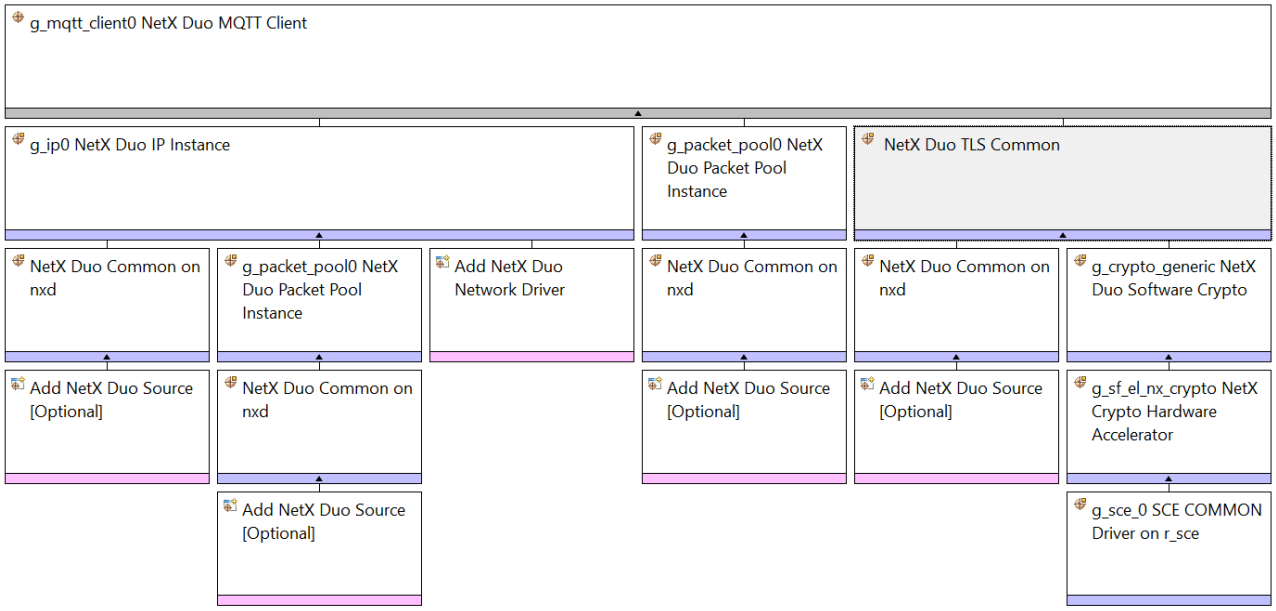


Figure 437: NetX Duo MQTT Client Component with TLS support enabled

NetX Duo MQTT Client Module Operational Notes

To secure the communication between MQTT client and broker, the TLS protocol is required. Please note that MQTT client does not add the NetX Duo TLS Session block. It only adds NetX Duo TLS Common block. This block defines/controls the common properties of NetX Secure. Please refer to the NetX Secure user notes for a detailed description of these properties. The NetX Duo TLS Session block is not added, so code to generate TLS session is not auto-generated. User/Application code should create the TLS session manually. The user code can perform the TLS session creation, configure security parameters and load relevant certificates under the TLS setup callback provided by the nxd_mqtt_client_secure_connect() API.

Before the TLS session is created, the metadata buffer must be allocated. The size of the metadata can be set using the properties pane. Also, the user can use NetX Secure nx_secure_metadata_size_calculate() API to calculate the size of the metadata buffer. Please refer to the NetX Duo MQTT Client user guide for more details. Refer to Using NetX Duo MQTT Client Module in an Application for a description of the callback.

Setting unique Client ID

The MQTT client instance is created using the nxd_mqtt_client_create() API. This API has a unique client ID as one of the parameters which is used by the MQTT broker to uniquely identify the client. The MQTT client component provides properties like Client ID Callback and Client ID Max Length, which are used to get unique client ID. Please refer to the following image.

Module g_mqtt_client0 NetX Duo MQTT Client	
Name	g_mqtt_client0
Name of generated initialization function	mqtt_client_init0
Auto Initialization	Enable
Client ID Callback	mqtt_client_id_callback
Client ID Max Length	12
Client Thread Stack Size	4096
Number of Messages to be stored in memory	1
Client thread priority	2

Figure 438: MQTT Client ID properties

ISDE auto generated code calls this Client ID callback before calling nxd_mqtt_client_create(). The prototype for Client ID callback is as follows:

```
void mqtt_client_id_callback(char * p_client_id, uint32_t * p_client_id_length);
```

Here client_id is an output parameter which will be filled in by this callback function and client_id_length is an input/output parameter.

Refer to Using NetX Duo MQTT Client Module in an Application for a description of the callback.

NetX Duo MQTT Client Limitations

- Refer to the associated SSP Release notes for the limitations on this module.

4.3.5.4 Including the NetX Duo MQTT Client Module in an Application

This section describes how to include the NetX Duo MQTT Client module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few sections of the SSP User's Manual, available as described in the Reference Section at the end of this document to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX Duo MQTT Client module to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the NetX Duo MQTT Client module is g_mqtt_client0. This name can be changed in the associated Properties window.)

TLS Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_mqtt_client NetX Duo MQTT Client on g_mqtt_client	Threads	New Stack> X-Ware> NetX Duo> Protocols > NetX Duo MQTT Client on g_mqtt_client

When the NetX Duo MQTT Client module on g_mqtt_client is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration

information will have box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level drivers; these are either optional or recommended (this is indicated in the block with the inclusion of this text.) If the addition of lower-level drivers is required, the module description will include "Add" in the text. Clicking on any Pink banded modules will bring up the "New" icon and then display the possible choices.

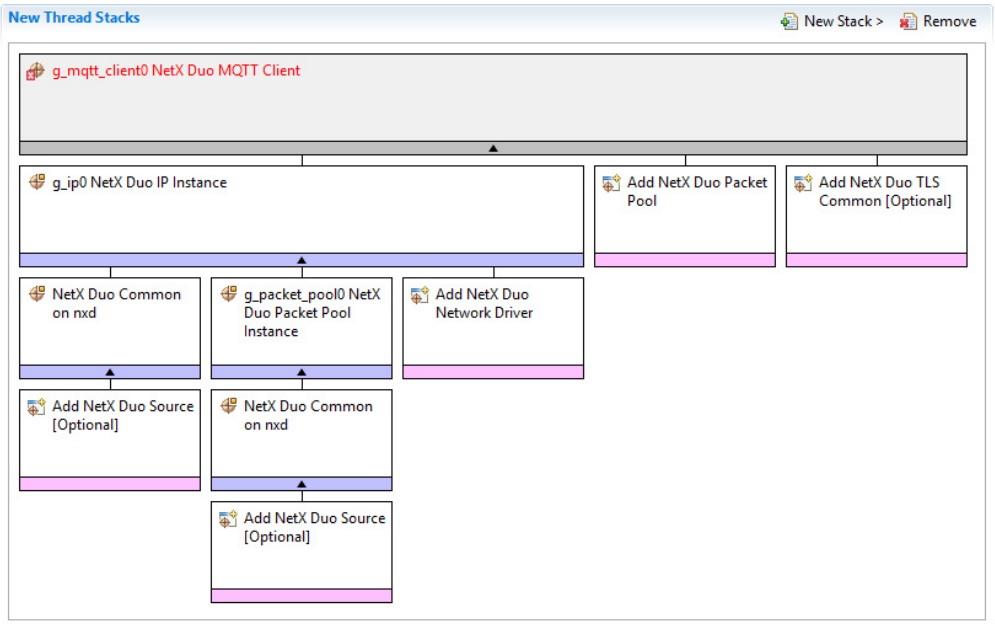


Figure 439: NetX MQTT Client Module Stack

4.3.5.5 Configuring the NetX Duo MQTT Client Module

The NetX Duo MQTT Client module must be configured by you for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority: this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX Duo MQTT Client Module

ISDE Property	Value	Description
Parameter Checking	Enable, Disable, BSP Default: BSP	Add parameter checking code
NX Secure	Enable, Disable Default: Enable	NX secure selection
Topic Name Max Length	12	Topic name max length selection
Message Max Length	32	Message max length selection
Keepalive Timer Rate(s)	1	Keepalive timer rate(s) selection
Ping Timeout Delay(s)	1	Ping timeout delay(s) selection
Socket Timeout (in timer ticks)	0xFFFFFFFF	Specify desired timeout using timer ticks
Name	g_mqtt_client0	Module name
Name of generated initialization function	mqtt_client_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection
Client ID Callback	mqtt_client_id_callback	Client ID callback selection
Client ID Max Length	12	Client ID max length selection
Client Thread Stack Size	4096	Client thread stack size selection
Number of Messages to be stored in memory	1	Number of messages to be stored in memory selection
Client thread priority	2	Client thread priority selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Configuration Settings for the MQTT Client Module Low Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

Configuration Settings for the NetX IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Enable	TCP selection
UDP	Enable	UDP selection

ISDE Property	Value	Description
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection
Link status change callback	NULL	Link status change callback selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Common

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	640	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Port ETHER

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enable or disable the parameter checking
Channel 0 Phy Reset Pin	IOPORT_PORT_09_PIN_03	Channel 0 Phy reset pin selection
Channel 0 MAC Address High Bits	0x00002E09	Channel 0 MAC address high bits selection
Channel 0 MAC Address Low Bits	0x0A0076C7	Channel 0 MAC address low bits selection
Channel 1 Phy Reset Pin	IOPORT_PORT_07_PIN_06	Channel 1 Phy reset pin selection
Channel 1 MAC Address High Bits	0x00002E09	Channel 1 MAC address high bits selection
Channel 1 MAC Address Low Bits	0x0A0076C8	Channel 1 MAC address low bits selection
Number of Receive Buffer Descriptors	8	Number of receive buffer descriptors selection

ISDE Property	Value	Description
Number of Transmit Buffer Descriptors	32	Number of transmit buffer descriptors selection
Ethernet Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Ethernet interrupt priority selection
Name	g_sf_el_nx	Module name
Channel	0	Channel selection
Callback	NULL	Callback selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

4.3.5.6 Using NetX Duo MQTT Client Module in an Application

Once the module has been configured and the files auto generated, the NetX Duo MQTT client Module is ready to be used in an application. Note that the auto generated code includes the initialization function with the name specified under the Name of generated initialization function property. This function internally calls the nxd_mqtt_client_create() API to create a MQTT client instance with the name specified under the Name property. Calls to this initialization function will be enabled or disabled depending the Auto Initialization property value.

The application need to implement the TLS setup callback function.

The application needs to implement the Client ID callback function which will be called by the initialization function prior to calling nxd_mqtt_client_create() API. This client ID callback should return a unique client ID, for example, MAC address.

Once the client instance is created, the typical steps in using the MQTT client module in an application are:

1) The application can connect to the broker by invoking nxd_mqtt_client_connect().
2) After connecting to the broker, the client can subscribe to a MQTT topic by invoking nxd_mqtt_client_subscribe()
3) After connecting to the broker, the client can publish a MQTT topic by invoking nxd_mqtt_client_publish()
4) The Incoming MQTT messages are stored in the receive queue in the MQTT client instance. The Application retrieves these message by invoking nxd_mqtt_client_message_get(). After consuming the data, the application must release the packet
5) To disconnect MQTT client, the application should invoke nxd_mqtt_client_disconnect(). This ensure that TCP connection to the broker is terminated.

6) The application should invoke nxd_mqtt_client_delete() to delete all the resources associated with MQTT client instance.

These common steps are illustrated in a typical operational flow diagram in the following figure:

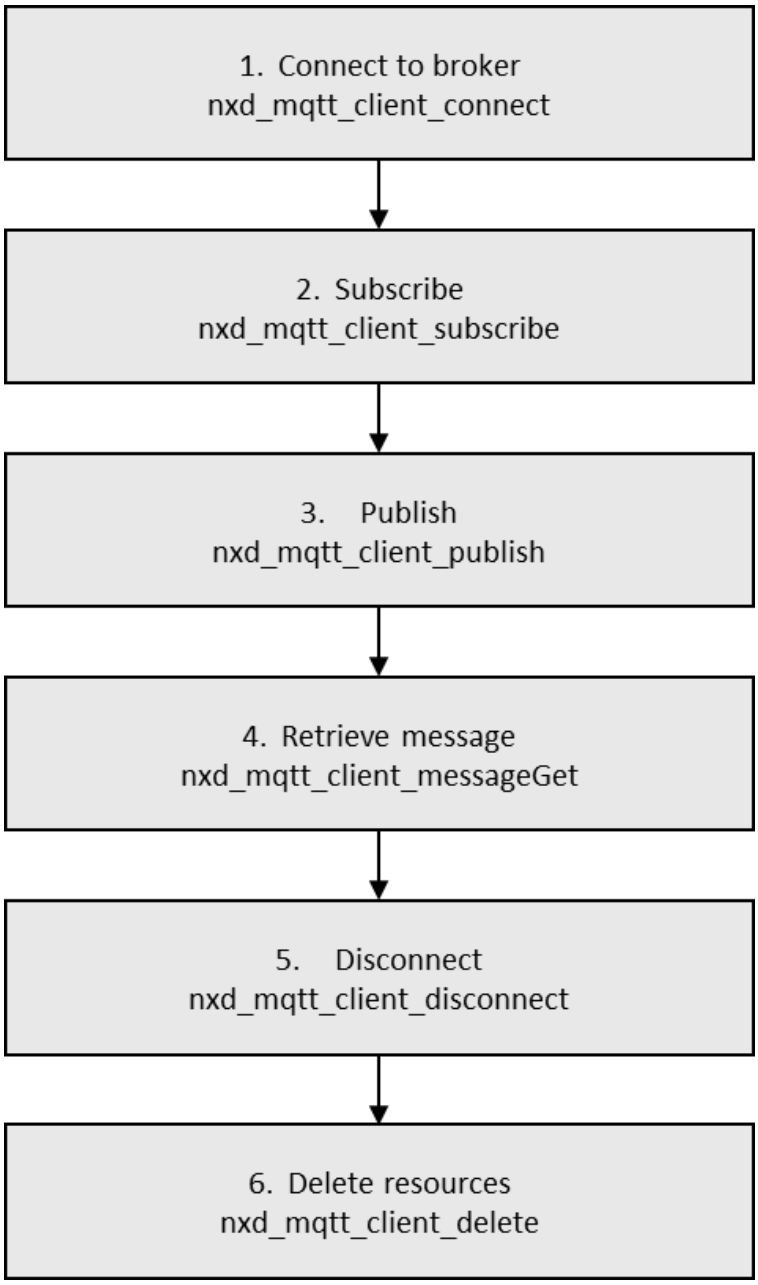


Figure 440: MQTT Client Typical Use Flow Diagram

The following illustrations provide guidance for understanding how to structure the callbacks described for TLS set-up and Client ID processing. Do not copy and paste these illustrations into your own code as they are not meant for that purpose.

Illustration of initialization in TLS setup callback:

```
/* Cipher Suite Variables  */
 extern const NX_SECURE_TLS_CRYPTO nx_crypto_tls_ciphers_synergys7;
 extern const NX_SECURE_TLS_CRYPTO nx_crypto_tls_ciphers;
UINT tls_setup(NXD_MQTT_CLIENT *client_ptr, NX_SECURE_TLS_SESSION *tls_session,
               NX_SECURE_X509_CERT *certificate, NX_SECURE_X509_CERT *trusted_ce
rtificate)
{
    status = nx_secure_tls_session_create(tls_session,
                                          &nx_crypto_tls_ciphers_synergys7,
                                          crypto_metadata,
                                          sizeof(crypto_metadata));
    if (status)
    {
        sprintf(str,"Error in creating TLS Session: 0x%02x\n", status);
        print_to_console(str);
        return(1);
    }
    else
    {
        sprintf(str,"TLS Session has been created. \r\n");
        print_to_console(str);
    }
    /* Allocate space for packet reassembly. */
    status = nx_secure_tls_session_packet_buffer_set(&(client_ptr->nxd_mqtt_tls_
session), tls_packet_buffer, sizeof(tls_packet_buffer));
    /* Check for error.  */
    if (status)
    {
        sprintf(str,"Error in setting Packet Buffer for TLS : %d \r\n",status);
        print_to_console(str);
        return(1);
    }
    else
    {
        sprintf(str,"Packet Buffer for TLS Has been set \r\n");
        print_to_console(str);
    }
    /* Need to allocate space for the certificate coming in from the remote host
. */

    nx_secure_tls_remote_certificate_allocate(tls_session, &remote_certificate,
remote_cert_buffer, sizeof(remote_cert_buffer));
    nx_secure_tls_remote_certificate_allocate(tls_session, &remote_issuer, remot
e_issuer_buffer, sizeof(remote_issuer_buffer));
```

```
    /* Add a CA Certificate to our trusted store for verifying incoming server c
ertificates. */
    nx_secure_x509_certificate_initialize(trusted_certificate, ca_cert_der, ca_c
ert_der_len, NX_NULL, 0, NULL, 0);
    nx_secure_tls_trusted_certificate_add(tls_session, trusted_certificate);

    /* Add the local certificate for client authentication. */
    nx_secure_x509_certificate_initialize(certificate, cert_der, cert_der_len, N
X_NULL, 0, private_key_der, private_key_der_len);
    nx_secure_tls_local_certificate_add(tls_session, certificate);


    /* Add a timestamp function for time checking and timestamps in the TLS hand
shake. */
    nx_secure_tls_timestamp_function_set(tls_session, tls_timestamp_function);

    /* Setup the callback invoked when TLS has a certificate it wants to verify
so we can
       do additional checks not done automatically by TLS. */
    nx_secure_tls_session_certificate_callback_set(tls_session, certificate_veri
fication_callback);

    return NX_SUCCESS;
}
```

Illustration of client Id callback:

```
void mqtt_client_id_callback(char * p_client_id, uint32_t * p_client_id_length)
{
    uint32_t id_length = 0;
    if (*p_client_id_length < sizeof(mac_id))
    {
       id_length = *p_client_id_length;
    }
    else
    {
        id_length = sizeof(mac_id);
    }
    /** Copy MAC address to Client ID and update Client ID length */
    memcpy(p_client_id, mac_id, id_length);
    *p_client_id_length = id_length;
}
```

4.3.6 NetX Duo TLS Session

The NetX Duo TLS Session Module provides high level APIs for Transport Layer Security (TLS) protocol-based client. It uses services provided by the Synergy Crypto Engine (SCE) to carry out hardware-accelerated encryption and decryption.

The NetX Duo TLS Session module is based on ThreadX "NetX Duo Secure" which implements the Secure Socket Layer(SSL) and its replacement Transport Layer Security (TLS) protocol as described in RFCs 2246 (version 1.0), and 5246 (version 1.2). NetX Duo Secure also includes routines for basic X.509 (RFC 5280). NetX Duo Secure is intended for applications using the ThreadX RTOS.

The TLS/SSL protocol provides privacy and reliability between two communicating applications. It has the following basic properties:

- Encryption: The messages exchanged between communicating applications are encrypted to ensure that the connection is private. Symmetric cryptography mechanism such as AES (Advanced Encryption Standard is used for data encryption.
- Authentication: A mechanism to check the peer's identity using certificates
- Integrity: A mechanism to detect message tampering and forgery to ensure that connection is reliable. Message Authentication Code (MAC) such as Secure Hash Algorithm (SHA) is used to ensure message integrity

An Application project that demonstrates the use of TLS as part of an MQTT client is available on the Renesas web site. The "Synergy Enterprise Cloud Toolbox" application project and application note can be found by searching the Renesas web site for R20AN0485.

4.3.6.1 NetX Duo TLS Session Module Features

- RFC 2246- The TLS Protocol Version 1.0
- RFC 5246- The Transport Layer Security (TLS) Protocol Version 1.2
- RFC 5280 X.509 PKI Certificates (v3)
- RFC 3268 Advanced Encryption Standard (AES) Cipher suites for Transport Layer Security (TLS)
- RFC 3447 Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1
- RFC 2104 HMAC: Keyed-Hashing for Message Authentication
- RFC 6234 US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)
- RFC 4279 Pre-Shared Key Cipher suites for TLS
- Supports TLS extensions for:
  - Secure Renegotiation Indication: This extension mitigates a Man-in-the-Middle attack vulnerability that could occur during a renegotiation handshake.
  - Server Name Indication: This extension allows a TLS Client to supply a specific DNS name to a TLS Server, allowing the server to select the correct credentials (assumes the server has multiple identity certificates and network entrypoints).
  - Signature Algorithms: This extension enables a TLS Client to provide a list of acceptable signature and hash algorithms to a TLS Server
- Supports X.509 extensions for:
  - Key Usage: Provides acceptable uses for a certificate's public key in a bitfield

– Extended Key Usage: Provides additional acceptable uses for a certificate's public key using OIDs
– Subject Alternative Name: Provides alternative DNS names that are also represented by the certificate

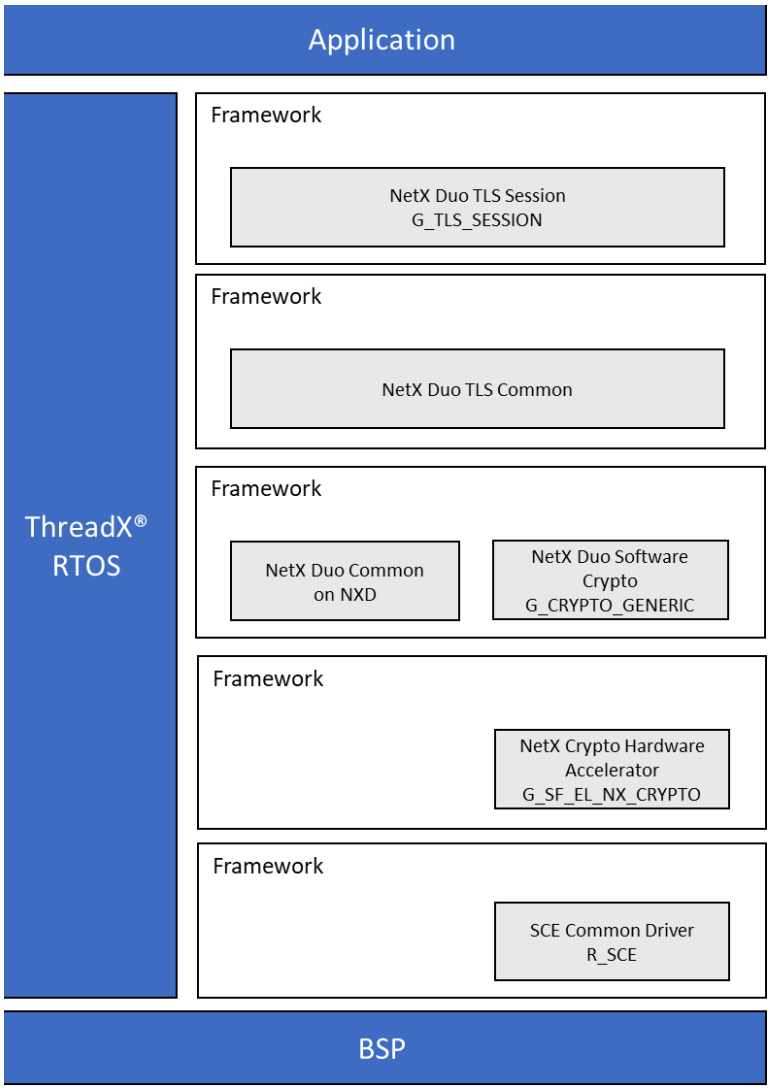


Figure 441: NetX Duo TLS Session Module Block Diagram

4.3.6.2 NetX Duo TLS Session Module APIs Overview

The NetX Duo TLS Support module defines APIs for creating and setting up a TLS security session. A complete list of the available APIs, an example API call, and a short description of each can be found in the table below. A table of status return values follows.

Function Name	Example API Call and Description
nx_secure_tls_local_certificate_add	**nx_secure_tls_local_certificate_add**(tls_session, certificate); Adds an initialized certificate to a TLS session for use as a local identification certificate - the TLS Server certificate for TLS servers, and the Client certificate for TLS clients.
nx_secure_tls_local_certificate_remove	**nx_secure_tls_local_certificate_remove**(tls_session, common_name, common_name_length); Removes a certificate instance from the local certificates list, keyed on the Common Name field.
nx_secure_tls_metadata_size_calculate	**nx_secure_tls_metadata_size_calculate**(cipher_table, metadata_size); Determines the size of the buffer needed by TLS for encryption metadata for a given ciphersuite table
nx_secure_tls_packet_allocate	**nx_secure_tls_packet_allocate**(tls_session, pool_ptr, packet_ptr, wait_option);Allocates a packet for a TLS application such that it allows additional room for the TLS header
nx_secure_tls_remote_ certificate_allocate	**nx_secure_tls_remote_certificate_allocate**(tls_session, certificate, raw_certificate_buffer, buffer_size); Adds an uninitialized certificate instance to a TLS session for the purpose of allocating space for certificates provided by a remote host during a TLS session
nx_secure_tls_session_ certificate_callback_set	**nx_secure_tls_session_certificate_callback_set**(tls_session, session); Sets up a function pointer that TLS will invoke when a certificate is received from a remote host, allowing the application to perform validation checks such as certificate revocation and certificate policy enforcement

Function Name	Example API Call and Description
nx_secure_tls_session_create	**nx_secure_tls_session_create**(session_ptr, cipher_table, metadata_area, metadata_size); Initializes a TLS session control block for later use in establishing a secure TLS session over a TCP socket or other lower-level networking protocol
nx_secure_tls_session_client_verify_disable	**nx_secure_tls_session_client_verify_ disable**(tls_session); Disables Client Certificate Verification for a particular TLS Session which previously had it enabled.
nx_secure_tls_session_client_verify_enable	**nx_secure_tls_session_client_verify_enable**(tls_session); Enables Client Certificate Verification for TLS Server instances. If enabled, the TLS Server will request and verify a remote TLS Client Certificate using all available crypto signature routines.
nx_secure_tls_session_delete	**nx_secure_tls_session_delete**(tls_session); Deletes a TLS session object, returning any resources to the system
nx_secure_tls_session_end	**nx_secure_tls_session_end**(tls_session, wait_option); Ends an active TLS session by sending the TLS CloseNotify alert to the remote host, then waiting for the response CloseNotify before returning.
nx_secure_tls_session_packet_buffer_set	**nx_secure_tls_session_packet_buffer_set**(session_ptr, buffer_ptr, buffer_size); Sets the buffer TLS uses to reassemble incoming messages which may span multiple TCP packets.

Function Name	Example API Call and Description
nx_secure_tls_session_protocol_version_override	**nx_secure_tls_session_protocol_version_override**(tls_session, protocol_version); Overrides the TLS protocol version to use for the TLS session. This allows for a different version of TLS to be utilized even if a newer version is enabled.
nx_secure_tls_session_receive	**nx_secure_tls_session_receive**(tls_session, packet_ptr_ptr, wait_option); Receives data from an active TLS session, handling all decryption and verification before returning the data to the caller in the supplied NX_PACKET structure
nx_secure_tls_session_reset	**nx_secure_tls_session_reset**(session_ptr); Resets a TLS session object, clearing out all data for initialization or re-use.
nx_secure_tls_session_send	**nx_secure_tls_session_send**(tls_session, packet_ptr, wait_option); Sends data using an active TLS session, handling all encryption and hashing before sending data over the established TCP socket connection
nx_secure_tls_session_start	**nx_secure_tls_session_start**(tls_session, tcp_socket, wait_option); Starts a TLS session given a TCP socket. The TCP connection must be established before calling this function or the TLS handshake will fail.
nx_secure_tls_session_time_function_set	**nx_secure_tls_session_time_function_set(**tls_session, time_func_ptr)** **Sets up a function pointer that TLS will invoke when it needs to get the current time, which is used in various TLS handshake messages and for verification of certificates.**

Function Name	Example API Call and Description
**nx_secure_tls_trusted_ certificate_add	**nx_secure_tls_trusted_certificate_add**(tls_session, certificate); Adds an initialized certificate to a TLS session for use as a trusted Root Certificate
**nx_secure_tls_trusted_ certificate_remove	**nx_secure_tls_trusted_certificate_remove**(tls_session, common_name, common_name_length); Removes a certificate instance from the trusted certificates store, keyed on the Common Name field.
nx_secure_tls_remote_certificate_free_all**	Release certificates previously registered with the TLS session.
**nx_secure_tls_psk_add	**nx_secure_tls_psk_add**(tls_session, pre_shared_key, psk_length, psk_identity, identity_length, hint, hint_length); Adds a pre-shared key (PSK) to a TLS session for use with a PSK ciphersuite. The second parameter is the PSK identity used during the TLS handshake to select the proper key.
**nx_secure_tls_psk_find	**nx_secure_tls_psk_find**(tls_session, psk_data, psk_length,psk_identity, identity_length); Finds a pre-shared key (PSK) in a TLS session for use with a PSK ciphersuite. The PSK is found using an "identity hint" that should match a field in the PSK structure in the TLS session.
**nx_secure_tls_client_psk_set	**nx_secure_tls_client_psk_set**(tls_session, pre_shared_key, psk_length, psk_identity, identity_length, hint, hint_length); Sets the pre-shared key (PSK) for a TLS Client in a TLS session control block for use with a remote server that is sing a PSK ciphersuite.
nx_secure_x509_certificate_initialize	Initialize X.509 Certificate for NetX Secure TLS
nx_secure_x509_common_name_dns_check	Check DNS name against X.509 Certificate

Function Name	Example API Call and Description
nx_secure_x509_crl_revocation_ check	Check X.509 Certificate against a supplied Certificate Revocation List

NOTE: ** Requires that the property PSK Cipher Suite of the TLS Common component be enabled.

4.3.6.3 NetX Duo TLS Session Module Operational Overview

TLS uses TCP and provides secure communications for application layer protocols such as HTTP and MQTT. TLS can also be used in 'bare' TCP socket applications to send and receive TCP packets in a secure session to another TCP peer. The module guide for this project uses this simplified application of TLS to demonstrate a TLS Client TCP and TLS Server TCP sockets exchanging data to a TCP peer.

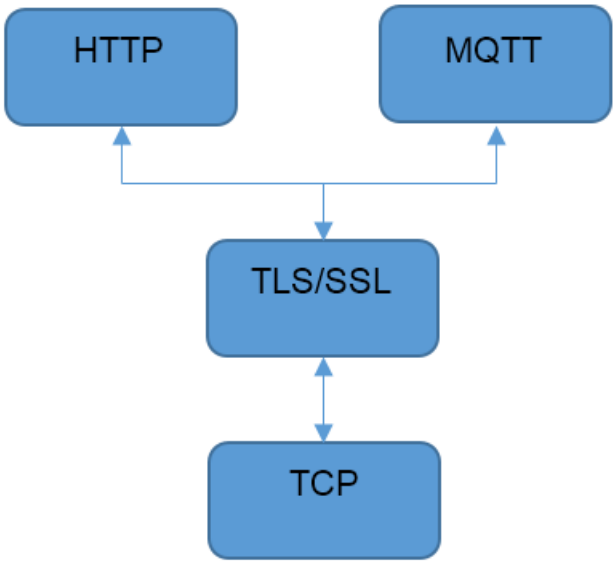


Figure 442: TLS/SSL Layering

TLS does not have a well-known port number. Instead it uses the designated port number of the secure variant of the higher layer protocol. For example, port number 443 for secure HTTP, port number 8883 for MQTT etc.

When a secure connection is established using TLS/SSL, for example using HTTPS, messages are exchanged between the client (which always initiates the connection) and a server. The first set of messages execute a Handshake Protocol after which the client and server can securely send/receive data bidirectionally as shown in the following figure.

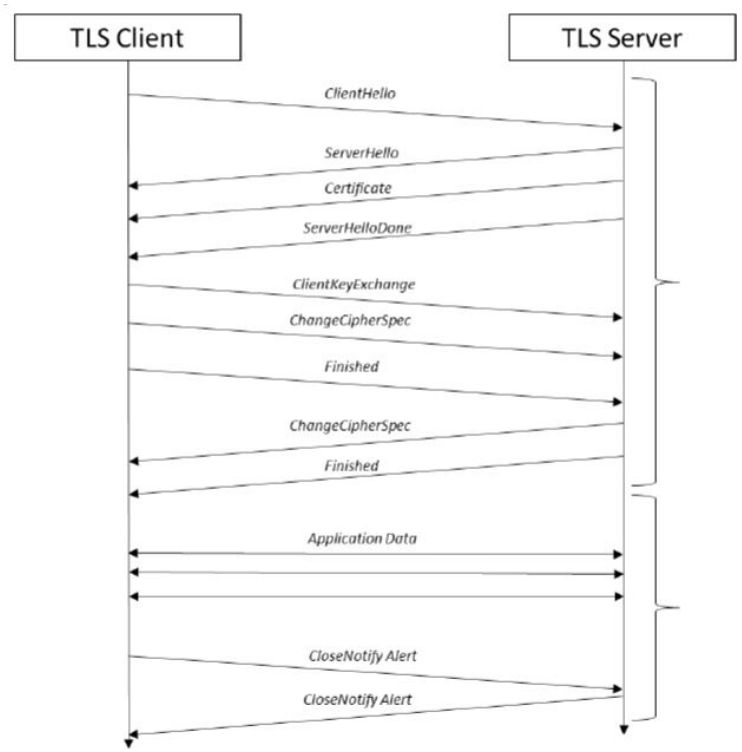


Figure 443: TLS Protocol Sequences

NetX Duo TLS Session Module Operational Notes

- Before the TLS session is created, the metadata buffer must be allocated. The size of meta data can be set using the properties pane. Also, the user can use NetX Duo Secure nx_secure_metadata_size_calculate API to calculate the required size of metadata buffer. The Metadata size is specified in the "Meta data size" of the TLS session component, or in the nx_secure_tls_session_create call; the default value is 4k but to handle most servers, 8k is recommended if the memory space is available.
- To associate a packet reassembly buffer to a TLS session, use the nx_secure_tls_session_packet_buffer_set API. The reassembly buffer is used to place the incoming TLS records which may span multiple TCP packets. If an incoming TLS record is larger than the supplied buffer, the TLS session will end with an error. A reasonable packet buffer size is 6-8k.
- Also, before starting a TLS client session, the application must allocate memory for processing Server certificate data in the nx_secure_tls_remote_certificate_allocate call. A reasonable size for most certificates is 2k. The TLS client application should allow for 2-3 certificates from most servers.
- For any incoming certificate, the NetX Duo Secure TLS will perform basic X.509 path validation.
- Additionally, at each stage in the verification process the expiration date of each certificate is checked against the time provided by the application timestamp function. The nx_secure_tls_session_time_function_set API is used to optionally set up a function pointer for application timestamp function that TLS will invoke when it needs to get the current time. The current time is used in some TLS handshake messages for verification of certificates. If a timestamp function is registered with the TLS session, a timestamp will be used in the generation of the Server or Client Hello, and in verifying the remote certificate.

- Before attempting to reconnect to the same or another TLS server, the TLS client must clear the TLS session. This is most easily done by calling nx_secure_tls_session_end. It is recommended to delete the TLS session and recreate it before making another connection attempt. For applications using SSP 1.3.x, the nx_secure_tls_session_create call should be preceded by a memset call on the TLS session to clear the session completely:

```
memset(tls\_session\_ptr, 0, sizeof(NXD\_SECURE\_TLS\_SESSION));
```

NetX Duo TLS Session Limitations

- Due to the nature of embedded devices, some systems may not have adequate memory to support the maximum TLS record size of 16 KB. NetX Duo TLS Secure can handle 16KB records on devices with sufficient resources.
- NetX Duo Secure performs basic certificate verification only. NetX Duo TLS Secure will perform basic X.509 chain verification on a certificate to assure that the certificate is valid and signed by a trusted Certificate Authority, and can provide the certificate Common Name for the application to compare against the Top-Level Domain Name of the remote host. However, verification of certificate extensions and other data is the responsibility of the application implementer. Refer to the Express Logic NetX Duo TLS Secure User Guide available from the Synergy Gallery for more details.
- Software-based cryptography is processor-intensive and not available. Therefore hardware-based cryptography is used for optimal performance of NetX Duo TLS Secure.
- The Self-Signed Certificates should be disabled in the final application. However, in MQTT Client in SSP 1.3.2 – 1.4.0, this option is automatically enabled even if set to Disabled. To disable this feature, modify the nx_secure_tls.h file in the project synergy\ssp\src\framework\el\nxd_application_layer\nxd_tls_secure directory to comment out NX_SECURE_ALLOW_SELF_SIGNED_CERTIFICATES directly. Make the file read only so it is not overwritten in the e^2 studio environment.
- Refer to the most recent SSP release notes for additional limitations on this module.

4.3.6.4 Including the NetX Duo TLS Session Module in an Application

This section describes how to include the NetX Duo TLS Session module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few sections of the SSP User's Manual, available as described in the Reference Section at the end of this document to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX Duo TLS Session module to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the NetX Duo TLS Session module is g_tls_session0. This name can be changed in the associated Properties window.)

Audio Playback Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_tls_session0 NetX Duo TLS Session on g_tls_session	Threads	New Stack> X-Ware> NetX Duo> Protocols > NetX Duo TLS Session on g_tls_session0

When the NetX Duo TLS Session module on g_tls_session0 is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information will have box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level drivers; these are either optional or recommended (this is indicated in the block with the inclusion of this text.) If the addition of lower-level drivers is required, the module description will include "Add" in the text. Clicking on any Pink banded modules will bring up the "New" icon and then display the possible choices.

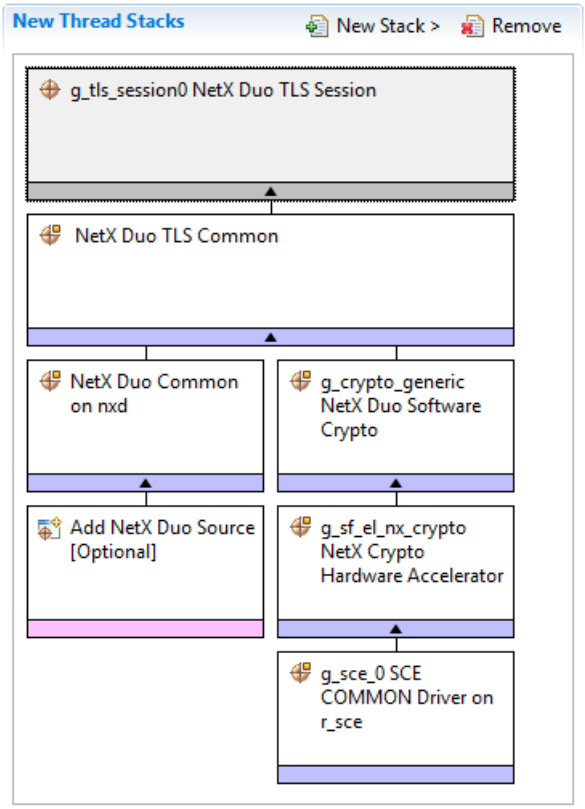


Figure 444: NetX MQTT Client Module Stack

To use TLS Secure with 'bare' TCP sockets, you will need to add an IP instance:

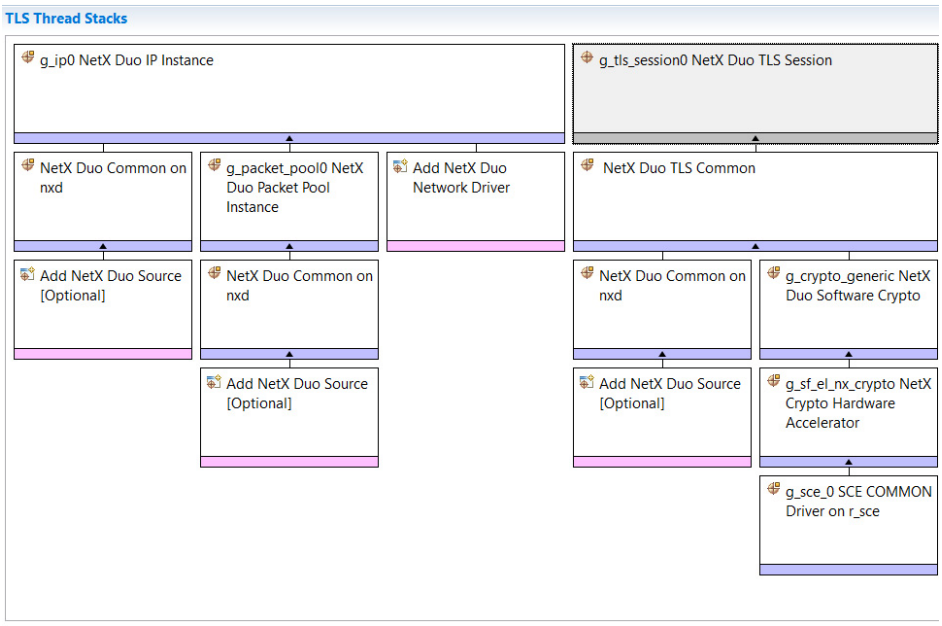


Figure 445: TLS Secure session using NetX Duo sockets directly

4.3.6.5 Configuring the NetX Duo TLS Session Module

The NetX Duo TLS Session module must be configured by you for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority: this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables, but is easily visible with the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Configuration Settings for the NetX Duo TLS Session Module

ISDE Property	Value	Description
Name	g_tls_session0	Module name
Meta data size	4000	Meta data size selection
Name of Timestamp Function	tls_timestamp_callback0	Name of timestamp function
Name of Certificate Verification function	certificate_verification_callback0	Name of certificate verification function
Name of generated initialization function	tls_dtls_session_init0	Name of generated initialization function
Auto initialization	Enable, Disable Default: Enable	Auto initialization selection

Configuration Settings for the NetX Duo TLS Server Session Module

ISDE Property	Value	Description
Name	g_tls_session1	Module name
Meta data size	4000	Meta data size selection
Name of Timestamp Function	NULL	Name of callback for setting timestamp
Name of Certificate Verification Function	NULL	Name of callback for verifying the remote certificate
Name of generated initialization function	tls_dtls_session_init0	Name of generated initialization function selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 MCU Family. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

NOTE: Most of the property settings for lower-level modules are intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

Configuration Settings for the TLS Session Module Lower Level Modules

Typically, only a small number of settings must be modified from the default for lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

Configuration Settings for the NetX TLS Common

ISDE Property	Value	Description
Crypto Engine	Hardware	Crypto engine selection
Self-Signed Certificates	Enable, Disable Default: Disable	Self-signed certificates selection*
PSK Cipher Suite	Enable, Disable Default: Disable	PSK cipher suite selection
X509 Strict Name Compare	Enable, Disable Default: Disable	X509 strict name compare selection
X509 Extended Distinguished Names	Enable, Disable Default: Disable	X509 extended distinguished names selection
Maximum RSA Modulus size (bits)	1024, 2048, 3072, 4096 Default: 4096	Maximum RSA modulus size (bits) selection
TLS v 1.0	Enable, Disable Default: Disable	TLS v 1.0 selection
TLS v 1.1	Enable, Disable Default: Disable	TLS v 1.1 selection

ISDE Property	Value	Description
Server Mode	Enable, Disable Default: Enable	Server mode selection
Client Mode	Enable, Disable Default: Enable	Client mode selection
Name of generated initialization function	nx_secure_common_init	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

NOTE: * This option MUST be disabled. Enabling this option is a serious security risk and is only intended for development and testing. However, in MQTT Client in SSP 1.3.2 – 1.4.0, this option is automatically enabled even if set to Disabled. To disable this feature, modify the nx_secure_tls.h file in the project synergy\ssp\src\framework\el\nxd_application_layer\nxd_tls_secure directory to comment out NX_SECURE_ALLOW_SELF_SIGNED_CERTIFICATES directly. Make the file read only so it is not overwritten in the e^2 studio environment.

Configuration Settings for the NetX Common

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Duo Software Crypto

ISDE Property	Value	Description
Name	g_crypto_generic	Module name
Name	g_packet_pool0	Module name

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Crypto Hardware Accelerator

ISDE Property	Value	Description
Name	g_sf_el_nx_crypto	Module name

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the SCE Common Driver on r_sce

ISDE Property	Value	Description
Name	g_sce_0	Module name
Endian Flag	CRYPTO_WORD_ENDIAN_BIG, CRYPTO_WORD_ENDIAN_LITTLE Default: CRYPT_WORD_ENDIAN_BIG	Endian flag selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Important Note: **Bolded** properties in the next set of tables need to be configured for your system.

Configuration Settings for the NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name

ISDE Property	Value	Description
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
IPv6 Global Address (use commas for separation) *	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
IPv6 Link Local Address (use commas for separation, All zeros means NetX Duo will create the LLA address using the device MAC address) *	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection. This will be configured automatically if left at the default value, based on the device MAC address.
IP Helper Thread Stack Size (bytes)	1024	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable	TCP selection
UDP	Enable, Disable Default: Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection

ISDE Property	Value	Description
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

NOTE: *Only necessary if using IPv6 network connections

Configuration Settings for the NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Configuration Settings for the NetX and NetX Duo Packet Pool Instance g_packet_pool0

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	1532	Packet size selection – certificate packets can be quite large, exceeding the default 640 bytes. So to avoid packet chaining overhead, set the packet payload to the device MTU, which is usually 1518 or so

ISDE Property	Value	Description
Number of Packets in Pool	16	Number of packets in pool selection. For servers with very large certificates, e.g. 4k bytes it might be necessary to increase this number accordingly.
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Configuration Settings for the NetX Port ETHER

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking
Channel 0 Phy Reset Pin	IOPORT_PORT_09_PIN_03	Channel 0 Phy reset pin selection
Channel 0 MAC Address High Bits	0x00002E09	Channel 0 MAC address high bits selection
Channel 0 MAC Address Low Bits	0x0A0076C7	Channel 0 MAC address low bits selection
Channel 1 Phy Reset Pin**	IOPORT_PORT_08_PIN_06	Channel 1 Phy reset pin selection
Channel 1 MAC Address High Bits	0x00002E09	Channel 1 MAC address high bits selection
Channel 1 MAC Address Low Bits	0x0A0076C8	Channel 1 MAC address low bits selection
Number of Receive Buffer Descriptors	8	Number of receive buffer descriptors selection
Number of Transmit Buffer Descriptors	32	Number of transmit buffer descriptors selection

ISDE Property	Value	Description
Ethernet Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled Recommend Priority 3 for most applications	Ethernet interrupt priority selection. Change from Disabled for the network driver to send and receive packets.
Name	g_sf_el_nx	Module name
Channel**	1	Channel selection
Callback	NULL	Callback selection

NOTE: ** This is specific to the SK-S7G2 MCU. The DK-S7G2 default values need not be modified.

For Wi-Fi networks, please refer to the Synergy Wi-Fi Application Project for SK-S7G2 - Application Project on the Renesas Gallery for more information on setting network parameters.

NetX Duo TLS Session Module Clock Configuration

The ETHERC peripheral module uses PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator clock tab prior to a build, or by using the CGC interface at run-time.

NetX Duo TLS Session Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I^2C pins.

NOTE: The selected operation mode determines the peripheral signals available and the MCU pins required.

Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

NOTE: The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII (Default: Disabled)	Select RMII as the Operation Mode for ETHERC1
Pin Group Selection	Mixed, _A only (Default: _A only)	Pin group selection
REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

NOTE: Example settings are for a project using the Synergy S7G2 MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

4.3.6.6 Using NetX Duo TLS Session Module in an Application

Autogenerated code includes the initialization function with the name as specified in the "Name of generated initialization function" property, for example, tls_dlts_session_init0(). The TLS Session instance variable specified using the "Name" property of the TLS Session is passed as input to the initialization function and this instance variable can be passed as input to several APIs of the TLS layer. The initialization function internally calls the nx_secure_tls_session_create () API

to create a TLS session. Calls to the initialization function will be enabled or disabled depending on the "Auto Initialization" property value.

Once the client instance is created, the typical steps in using the NetX Duo TLS Session module in an application are:

1) Packet the reassembly buffer for TLS session using the nx_secure_tls_session_packet_buffer_set API
2) Initialize the root CA certificate using the nx_secure_x509_certificate_initialize API
3) Load the root CA certificate to a trusted store using the nx_secure_tls_trusted_certificate_add API
4) Allocate space for certificates sent by remote server using the nx_secure_tls_remote_certificate_allocate API
5) Start the TLS session using the nx_secure_tls_session_start API
6) Send data over the secure connection using the nx_secure_tls_session_send API
7) Receive data over the secure connection using the nx_secure_tls_session_receive API
8) End the session using the nx_secure_tls_session_end API

These common steps are illustrated in a typical operational flow diagram in the following figure:

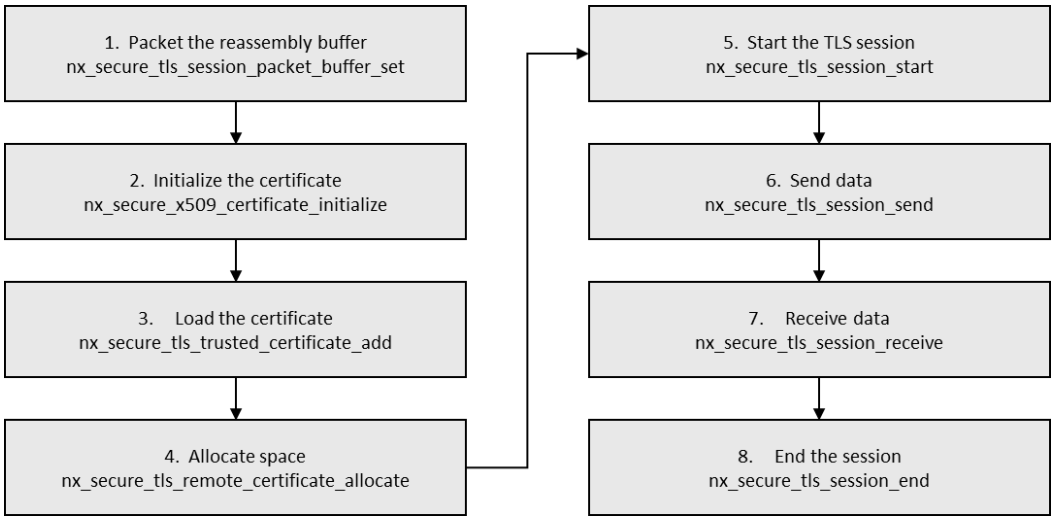


Figure 446: TLS Session Typical Use Flow Diagram

An Application project that demonstrates the use of TLS as part of an MQTT client is available on the Renesas web site. The "Synergy Enterprise Cloud Toolbox" application project and application note can be found by searching the Renesas web site for R20AN0485.

4.3.7 NetX Port Ethernet

The Synergy NetX Port Ether module (sf_el_nx) for NetX and NetX Duo is integrated into the SSP. Its function is to interface the generic NetX and NetX Duo software with the hardware. This module includes the MAC driver, the PHY driver, additional glue logic and utility functions.

NOTE: Unless otherwise stated there is no difference in how this module works in NetX or NetX Duo projects.

4.3.7.1 NetX Port Ether Module Features

- NetX services are implemented as a library, so only code that is needed is added to the project
  - For most applications, this results in an instruction image from between 5k bytes and 30k bytes, with IPV6 and ICMPv6 enabled, the size is from 30k bytes to 45k bytes.
- NetX supports a variety of RFCs including the following:
  - RFC 1112 Host Extensions for IP Multicasting (IGMPv1)
  - RFC 1122 Requirements for Internet Hosts - Communication Layers
  - RFC 2236 Internet Group Management Protocol, Version 2
  - RFC 768 User Datagram Protocol (UDP)
  - RFC 791 Internet Protocol (IP)
  - RFC 792 Internet Control Message Protocol (ICMP)
  - RFC 793 Transmission Control Protocol (TCP)
  - RFC 826 Ethernet Address Resolution Protocol (ARP)
  - RFC 903 Reverse Address Resolution Protocol (RARP)
  - RFC 2460 Internet Protocol v6 (IPv6) Specification (NetX Duo only)
  - RFC 4443 Internet Control Message Protocol (ICMPV6) (NetX Duo only)
  - RFC 4861 Neighbor Discovery for IPv6 (NetX Duo only)
  - RFC 4862 IPv6 Stateless Address Auto Configuration (NetX Duo only)
- Packet-based zero-copy implementation of TCP/IP (buffers are not copied inside NetX as they travel across the stack or from the stack to the user application, freeing up memory and processing cycles for example, significantly improves transmission rates)
- Fast UDP processing

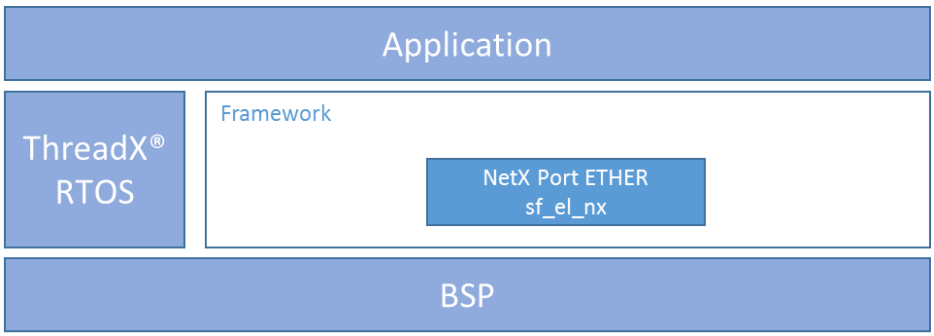


Figure 447: Net X Port Ether Module Organization, Options and Stack Implementations

4.3.7.2 NetX Port Ether Module APIs

The NetX Port Ether module has a narrow API, used internally by NetX and by the module itself. It includes the Ethernet driver entry point (nx_ether_driver_eth0, nx_ether_driver_eth1), the Ethernet interrupt handler and other functions used internally by the module but externally visible.

4.3.7.3 NetX Port Ether Module Operational Overview

The NetX Port Ether module is a high-performance real-time implementation of the NetX Ethernet driver for the Renesas Synergy software and Synergy Ethernet IP.

NOTE: NetX assumes the existence of ThreadX and depends on its thread execution, suspension, periodic timers, and mutual exclusion facilities.

Each IP instance in NetX has a primary interface which is identified by its device driver specified in the nx_ip_create service. The network driver is responsible for handling various NetX requests, including packet transmission, packet reception, and requests for status and control.

For a multi-home system, the IP instance can be configured for multiple interfaces, each with an associated network driver that performs these tasks for the respective interface. The network driver must also handle asynchronous events occurring on the media. Asynchronous events from the media include packet reception, packet transmission completion, and status changes. NetX provides the network driver with several access functions to handle various events. These functions are designed to be called from the interrupt service routine portion of the network driver. For IP networks, the network driver should forward all ARP packets received to the _nx_arp_packet_deferred_receive internal function. All RARP packets should be forwarded to _nx_rarp_packet_deferred_receive internal function. There are two options for IP packets:

If fast dispatch of IP packets is required, incoming IP packets are forwarded to _nx_ip_packet_receive for immediate processing; this greatly improves NetX performance in handling IP packets. Otherwise, IP packets are forwarded to _nx_ip_packet_deferred_receive. This service places the IP packet in the deferred processing queue where it is then handled by the internal IP thread, which results in the least amount of ISR processing time.

The network driver can also defer interrupt processing to run out of the context of the IP thread. In this mode, the ISR shall save the necessary information, call the internal function _nx_ip_driver_deferred_processing, and acknowledge the interrupt controller. This service notifies IP thread to schedule a callback to the device driver to complete the process of the event that causes the interrupt.

Some network controllers are capable of performing TCP/UDP/IP header checksum computation and validation in hardware without taking up valuable CPU resources. To take advantage of the hardware capability feature, NetX provides options to enable or disable various software checksum computation at compilation time, as well as turning on/off checksum computation at run time. See the "NetX Network Drivers" section in the NetX User Guide for more detailed information on writing NetX network drivers.

NetX Port Ether Module Important Operational Notes and Limitations

A summary of some key operational notes is given below. Refer to the *NetX User Guide for the Renesas Synergy™ Platform* and *NetX Duo User Guide for the Renesas Synergy™ Platform* for more details on each of these topics. The source code symbol is provided in addition to the NetX Source property for each feature below. For example, to change the number of physical network interfaces, you can either set the *Maximum Physical Interfaces* property in the NetX Source or NetX Duo Source element, or you can define the source code symbol NX_MAX_PHSYCIAL_INTERFACES directly. In either case, it is still necessary to include the NetX and NetX Duo Source component, generate the project files and to rebuild the NetX library.

NetX Source Level settings

The options below are in the NetX and NetX Duo Source components.

- **Multiple Network Interface Support**NetX supports systems connected to multiple physical devices using a single IP instance. Each physical interface is assigned to an interface control block in the IP instance. Applications wishing to use a multi-home system must define the value for the *Maximum Physical Interfaces* property (or

NX_MAX_PHSYCIAL_INTERFACES) to the number of physical devices attached to the system, and then rebuild the NetX library. If the library is not rebuilt, this has no effect. The default value is 1.

- **Loopback Interface**The loopback interface is a special network interface without a physical link attached to. The loopback interface allows applications to communicate using the IP loopback address 127.0.0.1 To utilize a logical loopback interface, either set the Loopback property to enabled (the default setting) or ensure the symbol NX_DISABLE_LOOPBACK_INTERFACE is not defined.
- **Enabling IPv6 (NetX Duo only)**By default, IPv6 is enabled on the IP instance in NetX Duo. The IP instance can be disabled or enabled for IPv6 by setting the *NetX Duo IPV6 Support* property in the NetX Duo Source component. Other IPv6 related properties may be enabled (checksum on ICMPv6 packets, Duplicate Address Detection, etc). See the *NetX Duo User Guide for the Renesas Synergy™ Platform* for details on IPv6 networking.

Interface Control Blocks: The number of interface control blocks in the IP instance is the number of physical interfaces (defined by NX_MAX_PHYSICAL_INTERFACES) plus the loopback interface if it is enabled. The total number of interfaces is defined in the symbol NX_MAX_IP_INTERFACES. This value should **not** be modified directly. NetX will define this symbol internally based on the NX_MAX_PHSYCIAL_INTERFACES and NX_DISABLE_LOOPBACK_INTERFACE settings.

NetX Port Ether settings

- **Channel**This determines which interface the ethernet driver is applied to. It may need to be modified from the default value of zero. See Configuring the NetX Port Ether Framework Module.
- **Channel 0/1 Phy Reset Pin**This property to be set depends on the value of the Channel property above. It may need to be modified from the default value. See Configuring the NetX Port Ether Framework Module.
- **Channel 0/1 MAC Address High Bits**
- **Channel 0/1 MAC Address Low Bits**These properties set the device MAC address on the respective channel interface at compile time. To set the MAC address at run time, see the description of the *Callback* property below.
- **Dynamic MAC Address Callback**This property defines the user defined callback function which sets MAC address at runtime (network link initialization). The default value is NULL and the MAC address is assigned using the Channel MAC Address High/Low Bits settings.
- **Unknown Ethernet packet receive Callback**This property configures the callback allowing the user to process unsupported/custom EtherType packets. This call-back can be used in conjugation with the nx_ether_custom_packet_send Ethernet API to send and receive custom Ethernet packet types.
- **Name**This property names the instance of the NetX Port ETHER driver. For an IP instance configured with multiple network interfaces, the application must add an instance of the driver and give it a unique name (or compile errors will result). The default name is g_sf_el_nx.
- **Ethernet Interrupt Priority**This property sets the driver interrupt priority. The default value is Priority 4. The dropdown list indicates what are valid and invalid priorities depending on the MCU target.
- **Link status monitoring**This property allows the user to choose the link status monitoring method. There are two options available:
  - PHY Polling: The PHY polling utilizes an internal monitoring thread for auto negotiation to notify the user when a link is lost or established.
  - PHY Interrupt (Uses LINKSTA Pin): The PHY Interrupt method requires the PHY status pin to be connected to the LINKSTA pin of the Ethernet controller and utilize the Ethernet interrupt to notify the user when the link is lost or established. This notification uses the callback registered with the IP instance.
- **Number of Receive Buffer Descriptors**

- **Number of Transmit Buffer Descriptors**These properties set the number of buffer descriptors (BDs) for receiving and transmitting packets. When the Receive BDs are initialized, the driver allocates a packet for each BD. Therefore, the number of receive BDs should not deplete the IP instance packet pool from which it allocates packets.The NetX Port ETHER driver supports packet chaining for both receiving and transmitting packets (where packets are chained in the application layer). If a packet is received on the network that exceeds the size of the IP default packet pool payload, the driver can allocate additional packets and process the incoming packet as a packet chain.
- **Parameter Checking**This is low level error checking for each component of the project hardware layer. By default, it is set to Default (BSP) which means this property inherits the same property in the BSP.

Refer to the most recent SSP release notes for any additional operational limitations for this module.

4.3.7.4 Including the NetX Port Ether Module in an Application

This section describes how to include the NetX Port Ether Module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP based applications.

The NetX Port Ether Framework is used as a lower level module and is not available to add as a separate stack. An example is illustrated in the following figure where a module must be selected to Add NetX Network Driver. The figures following the below diagram show the selection of the NetX Port Ether Framework to complete the stack for NetX and NetX Duo.

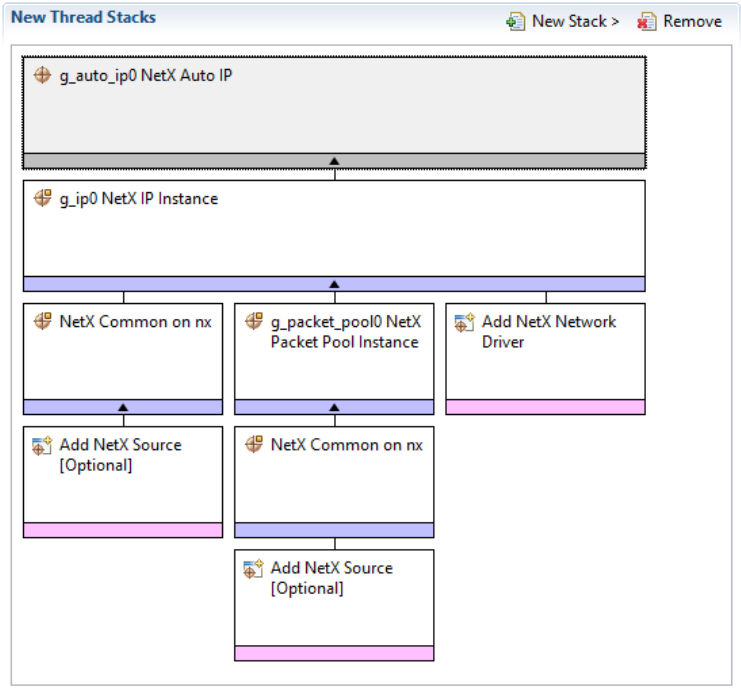


Figure 448: NetX Port Ether is an Option for Add NetX Network Driver

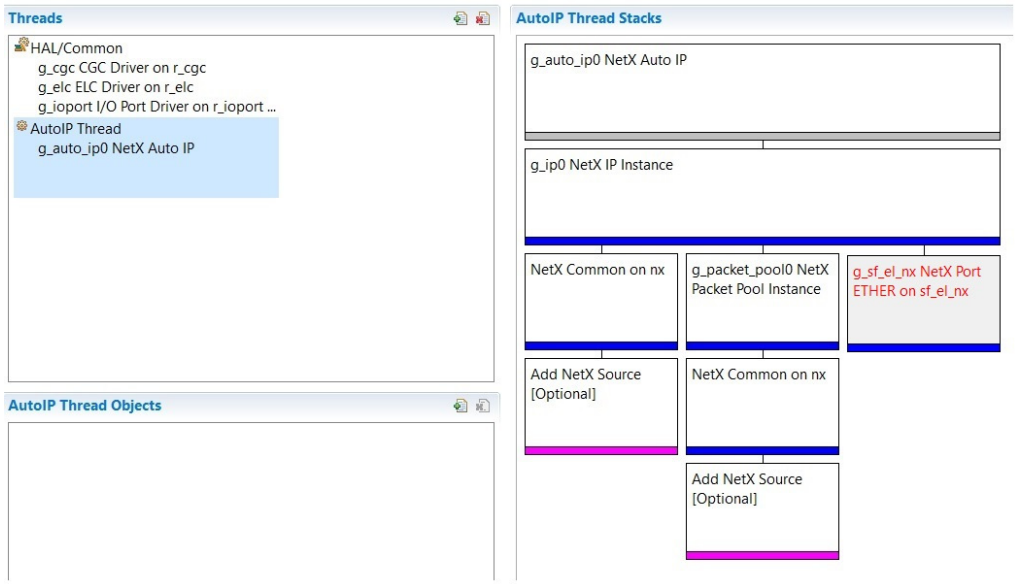


Figure 449: NetX Port Ether Application using NetX Auto IP

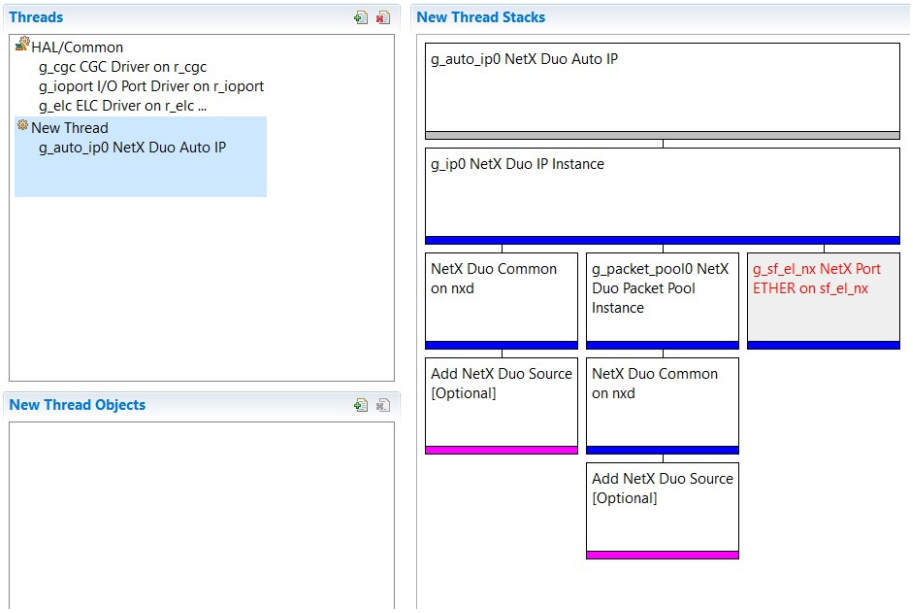


Figure 450: NetX Port Ether Application using NetX Duo Auto IP.

4.3.7.5 Configuring the NetX Port Ether Framework Module

The NetX Port Ether Framework module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) those components which must be configured for

lower level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the property window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user accessible properties are given in the properties tab within the SSP Configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the properties window of the associated module. Simply select the indicated module and then view the properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+.) This level of detail is not included in the configuration properties tables below, but is easily visible with the ISDE when configuring interrupt priority levels.

NOTE: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the Configuration Table Settings given below. This will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Default Configuration Settings for NetX Port Ether Module on sf_el_nx

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled ~~~~Default: BSP	Enable or disable the parameter checking
Channel 0 Phy Reset Pin	IOPORT_PORT_09_PIN_03	Channel 0 Phy reset pin selection
Channel 0 MAC Address High Bits	0x00002E09	Channel 0 MAC address high bits selection
Channel 0 MAC Address Low Bits	0x0A0076C7	Channel 0 MAC address low bits selection
Channel 1 Phy Reset Pin	IOPORT_PORT_07_PIN_06	Channel 1 Phy reset pin selection
Channel 1 MAC Address High Bits	0x00002E09	Channel 1 MAC address high bits selection
Channel 1 MAC Address Low Bits	0x0A0076C8	Channel 1 MAC address low bits selection
Number of Receive Buffer Descriptors	8	Number of receive buffer descriptors selection
Number of Transmit Buffer Descriptors	32	Number of transmit buffer descriptors selection
Ethernet Interrupt Priority	Priority 0 (highest), Priority 1:14, Priority 15 lowest - not valid if using ThreadX)~~~~Default: Priority 12	Ethernet interrupt priority selection

ISDE Property	Value	Description
Link status monitoring method	PHY Interrupt (Uses LINKSTA Pin), PHY Polling~~~~Default: PHY Polling	PHY interrupt requires LINKSTA Pin connection to PHY
Name	g_sf_el_nx	Module name
Channel	0	Channel selection
MAC address change callback	NULL	MAC address change callback selection
Unknown packet receive callback	NULL	Unknown packet receive callback selection

NOTE: The example values and defaults are for a project using the Synergy S7G2 Family. Other MCUs may have different default values and available configuration settings.

4.3.7.6 Using the NetX Port Ether Module in an Application

The NetX Port Ether Module should be used in combination with NetX. The steps performed by the SSP automatically in a NetX application support are:

1) Initialize the system with nx_system_initialize
2) Create a packet pool with nx_packet_pool_create. This creates the IP default packet pool, to be used by the IP instance and the ethernet driver.
3) Create an IP Instance with nx_ip_create
4) Enable ARP with nx_arp_enable
5) Enable Link status change notify call back with nx_ip_link_status_change_notify_set
6) Set gateway IP address for created IP instance nx_ip_gateway_address_set
7) Create an AutoIP instance with nx_auto_ip_create

The steps below are performed directly by the application to set up and run the Auto IP thread task.

1) Check if the network link is enabled with the nx_ip_interface_status_check API.
2) Verify the device does not have an IP address by calling the nx_ip_interface_address_get API.
3) Set the IP address notification callback by calling the nx_ip_address_change_notify API.
4) Start the Auto IP instance with the nx_auto_ip_start API.
5) Wait for the IP address change callback to set the flag that the IP instance has an IP address.
6) Stop the AutoIP task by calling the nx_auto_ip_stop API.

7) Verify the IP instance has a non-zero IP address by calling the nx_ip_interface_address_get API again.

The above common steps are illustrated in the operational flow diagram in the figure below:

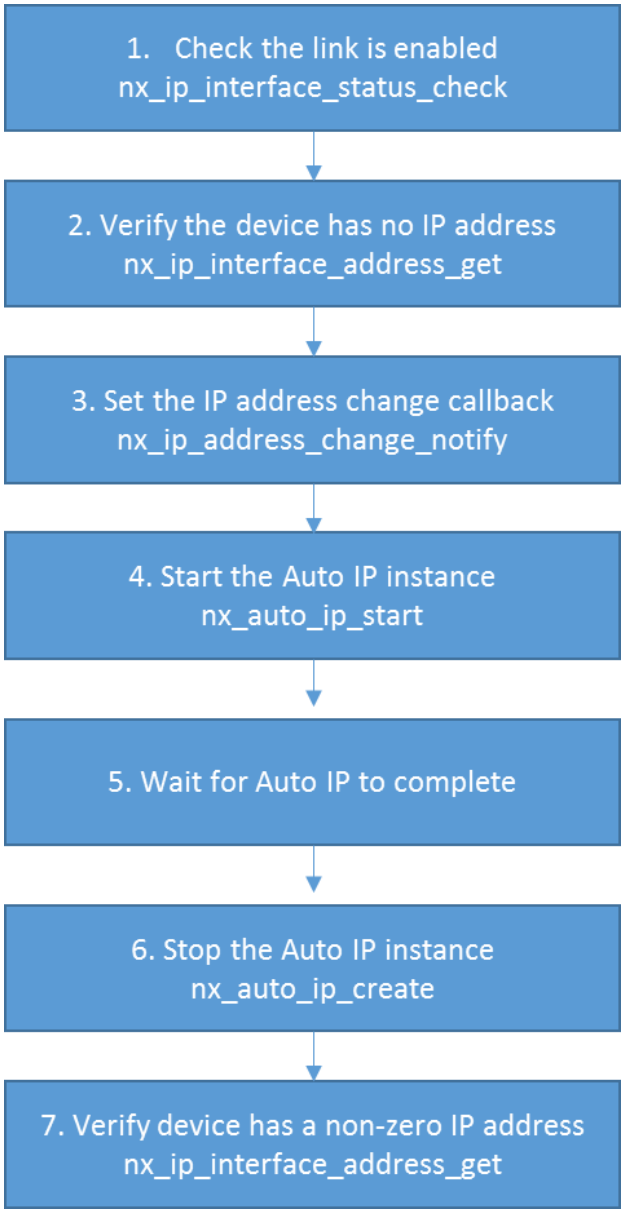


Figure 451: Flow Diagram of a Simple NetX Application using Auto IP

4.3.8 NetX and NetX Duo SNMP Agent

The NetX and NetX Duo SNMP Agent module provides high-level APIs for implementing the SNMP agent for the SSP architecture. It's part of the NetX and NetX Duo application bundle included in the SSP X-Ware integration. This module is MCU independent, so any MCU supports NetX and NetX Duo can implement the SNMP agent.

4.3.8.1 NetX and NetX Duo SNMP Agent Module Features

- The NetX and NetX Duo SNMP agent module is compliant with RFC1155, RFC1157, RFC1215, RFC1901, RFC1905, RFC1906, RFC1907, RFC1908, RFC2571, RFC2572, RFC2574, RFC2575, RFC 3414 and related RFCs.
- The SNMP agent module operates only in UDP. TCP is not supported.
- The SNMP agent module doesn't support Transport Layer Security (TLS) or Datagram Transport Layer Security (DTLS)
- The NetX and NetX Duo SNMP protocol implements SNMP Version 1, 2, and 3. The SNMPv3 implementation supports MD5 and Secure Hash Algorithm 1(SHA-1) authentication, and Data Encryption Standard (DES) encryption. This version of the NetX and NetX Duo SNMP Agent has the following constraints:
  - One SNMP Agent per NetX IP Instance.
  - No support for RMON.
  - SNMP v3 Inform messages are not supported
- Provides a mechanism to register callbacks for handling username, get, set and getnext when creating a SNMP agent.

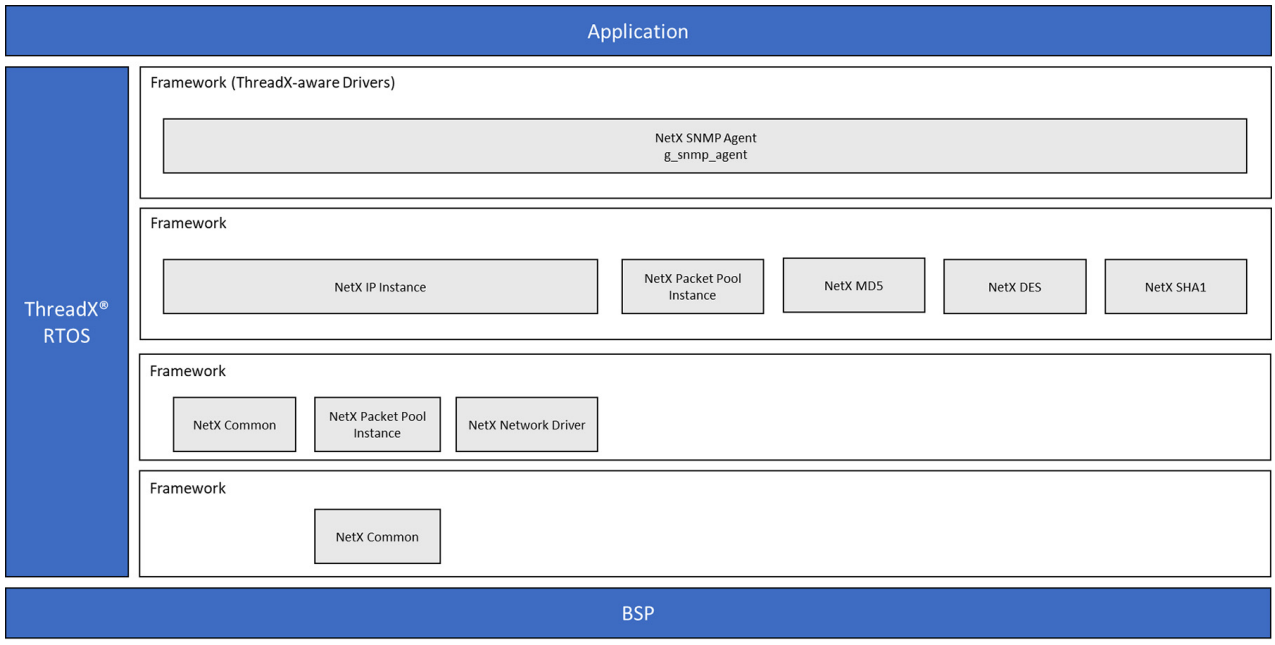


Figure 452: NetX and NetX Duo SNMP Agent Module Block Diagram

4.3.8.2 NetX and NetX Duo SNMP Agent Module APIs Overview

The NetX and NetX Duo SNMP Agent module defines APIs to create, delete and manage SNMP operations. For a list of APIs and a description of their functions and return values refer to the Express Logic SNMP user manual available for download on the Synergy Gallery in the X-Ware zip file.

4.3.8.3 NetX and NetX Duo SNMP Agent Module Operational Overview

The SNMP agent module make use of the underlying NetX/NetX Duo stack to perform the operations. Along with the IP stack, it makes use of NetX MD5, NetX SHA1 and NetX DES modules for authentication and encryption in SNMP v3 operation.

The SNMP agent module can be created with the nx_snmp_agent_create API. For the implementation of the SNMP agent the user need to define the handler functions for username, get, getnext and set operations.

NetX and NetX Duo SNMP Agent Module Important Operational Notes and Limitations

Disabling SNMP Version 1

The SNMP agent module can disable processing of version 1 requests by selecting "Disable" in the configuration properties for the SNMP agent for "SNMP Version 1". By default, this is enabled. When disabled the SNMP agent simply drops the packet with version 1, resulting in a timeout for the SNMP manager.

Disabling SNMP Version 2

The SNMP agent module can disable processing of version 2 requests by selecting "Disable" in the configuration properties for the SNMP agent for "SNMP Version 2". By default, this is enabled. When disabled the SNMP agent simply drops the packet with version 2, resulting in a timeout for the SNMP manager.

Disabling SNMP Version 3

The SNMP agent can disable processing of version 3 requests by selecting "Disable" in the configuration properties for the SNMP agent for "SNMP Version 3". By default, this is enabled. When disabled the SNMP agent simply drops the packet with version 3 resulting in a timeout for the SNMP manager. Enabling SNMP version 3 requires MD5, SHA1 authentication and DES encryption.

- One SNMP Agent per NetX IP Instance.
- No support for RMON
- SNMP v3 Informs are not supported
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4.3.8.4 Including the NetX and NetX Duo SNMP Agent Module in an Application

This section describes how to include the NetX and NetX Duo SNMP Agent Module in an application using the SSP configurator.

NOTE: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX and NetX Duo SNMP Agent Module to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the NetX and NetX Duo SNMP Agent Module is g_sf_snmp_agent0. This name can be changed in the associated Properties window.)

NetX and NetX Duo SNMP Agent Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_snmp_agent0 NetX SNMP Agent	Threads	New Stack> X-Ware> NetX/NetX Duo> Protocols> NetX/NetX Duo SNMP Agent

When the NetX and NetX Duo SNMP Agent module on sf_snmp_agent is added to the thread stack as shown in the following figure, the configurator automatically adds any lower-level drivers that are required. Any drivers that need additional configuration information will be box-text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

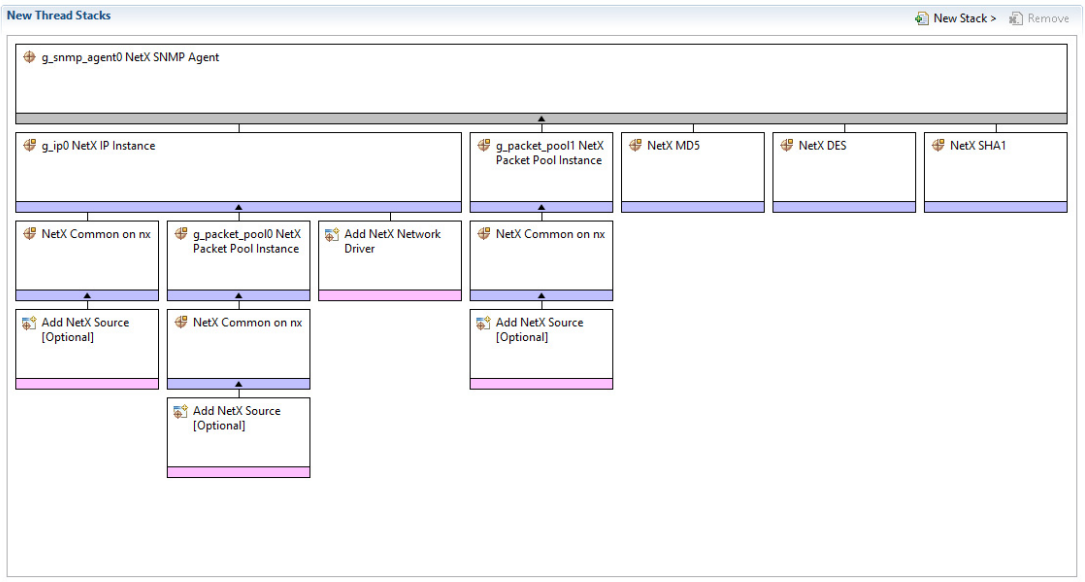


Figure 453: NetX SNMP Agent Module Stack (NetX Duo is similar)

4.3.8.5 Configuring the NetX and NetX Duo SNMP Agent Module

The NetX and NetX Duo SNMP Agent Module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules in order to ensure successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are not available for changes, and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included

in the following configuration properties tables, but is easily visible within the ISDE when configuring interrupt-priority levels.

NOTE: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings; this will help orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with the SSP.

Configuration Settings for the NetX and NetX Duo SNMP Agent Module on sf_snmp_agent

ISDE Property	Value	Description
Internal thread stack size (bytes)	4096	Internal thread stack size selection
SNMP agent priority	16	SNMP agent priority selection
Type of service for SNMP responses	Normal, Minimum Delay, Maximum Delay, Maximum Reliability, Minimum Cost Default: Normal	Type of service for SNMP responses selection
Fragment enable for SNMP PDU requests	Fragment, Don't Fragment Default: Don't Fragment	Fragment enable for SNMP PDU requests selection
SNMP socket time to live	128	SNMP socket time to live selection
Agent timeout	100	Agent timeout selection
Max octet string size	255	Max octet string size selection
Max content string size	32	Max content string size selection
Max User name Size	64	Max User name Size selection
Max security Key Size	64	Max security Key Size selection
Minimum SNMP packet size	560	Minimum SNMP packet size selection (Value must be between 560 to 1500)
UDP port number	161	UDP port number selection (Value must be between 1 to 65535)
Trap destination port	162	Trap destination port selection
Max trap Name Size	64	Max trap Name Size selection

ISDE Property	Value	Description
Max trap Key Size	64	Max trap Key Size selection
Inveral between SNMP packet processing timer ticks	100	Inveral between SNMP packet processing timer ticks selection
SNMP Version 1	Enable, Disable Default: Enable	SNMP Version 1 selection
SNMP Version 2	Enable, Disable Default: Enable	SNMP Version 2 selection
SNMP Version 3	Enable, Disable Default: Enable	SNMP Version 3 selection
Name	g_snmp_agent0	Name selection
Read Community String	public	Read Community String selection (Must be less than 64 chars)
Write Community String	private	Write Community String selection (Must be less than 64 chars)
Name of SNMP Username Handler	sf_snmp0_username_handler	Name of SNMP Username Handler selection
Name of SNMP GET Handler	sf_snmp0_get_handler	Name of SNMP GET Handler selection
Name of SNMP GETNEXT Handler	sf_snmp0_getnext_handler	Name of SNMP GETNEXT Handler selection
Name of SNMP SET Handler	sf_snmp0_set_handler	Name of SNMP SET Handler selection
Name of generated initialization function	snmp_agent_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto Initialization selection

ISDE Property	Value	Description
SNMP agent instance id	0	SNMP agent instance id selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
**IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection
IP Helper Thread Stack Size (bytes)	2048	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	Enable	ARP selection
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Enable	TCP selection
UDP	Enable	UDP selection

ISDE Property	Value	Description
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection
Link status change callback	NULL	Link status change callback selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX MD5

ISDE Property	Value	Description
No configurable settings		

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX DES

ISDE Property	Value	Description
No configurable settings		

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX SHA1

ISDE Property	Value	Description
No configurable settings		

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Common

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	640	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

Configuration Settings for the NetX Port ETHER

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enable or disable the parameter checking
Channel 0 Phy Reset Pin	IOPORT_PORT_09_PIN_03	Channel 0 Phy reset pin selection
Channel 0 MAC Address High Bits	0x00002E09	Channel 0 MAC address high bits selection
Channel 0 MAC Address Low Bits	0x0A0076C7	Channel 0 MAC address low bits selection
Channel 1 Phy Reset Pin	IOPORT_PORT_07_PIN_06	Channel 1 Phy reset pin selection
Channel 1 MAC Address High Bits	0x00002E09	Channel 1 MAC address high bits selection
Channel 1 MAC Address Low Bits	0x0A0076C8	Channel 1 MAC address low bits selection
Number of Receive Buffer Descriptors	8	Number of receive buffer descriptors selection
Number of Transmit Buffer Descriptors	32	Number of transmit buffer descriptors selection
Ethernet Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest-not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Ethernet interrupt priority selection
Name	g_sf_el_nx	Module name
Channel	0	Channel selection
Callback	NULL	Callback selection

NOTE: The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

NetX and NetX Duo SNMP Agent Module Clock Configuration

The NetX and NetX Duo SNMP Agent module ETHERC peripheral uses the PCLKA as its clock source.

To change the clock frequency at run-time, use the CGC Interface.

NetX and NetX Duo SNMP Agent Module Pin Configuration

To use the NetX and NetX Duo SNMP Agent Module, the port pins for the peripheral inputs and outputs must be set in the pin configurator in the ISDE. The following table illustrates the method for selecting the pins within the ISDE configuration window:

Pin Selection Sequence for the NetX and NetX Duo SNMP Agent Module

Resource	ISDE Tab	Pin selection Sequence
ETHERC	Pins	Select Peripherals > Peripherals>Connectivity:ETHERC> ETHERC0/1

4.3.8.6 Using the NetX and NetX Duo SNMP Agent Module in an Application

Once the module has been configured and the files generated, the SNMP module is ready to be used in an application. The typical steps to using the SNMP module in an application are:

1) Create an agent with nx_snmp_agent_create API
2) Define handler functions
    a) username handler function nx_snmp_agent_username_process in user code
    b) get handler function nx_snmp_agent_get_process in user code
    c) getnext handler function nx_snmp_agent_getnext_process in user code
    d) set handler function nx_snmp_agent_set_process in user code
3) Create a structure to map OIDs to action handlers in user code
4) Configure read, write community strings nx_snmp_agent_public_string_set and nx_snmp_agent_private_string_set
5) If the user wants to use SNMPv3, they must create the keys using the nx_snmp_agent_md5_key_create API or the nx_snmp_agent_sha_key_create and associate the keys with authentication, encryption and trap services nx_snmp_agent_authenticate_key_use, nx_snmp_agent_privacy_key_use, nx_snmp_agent_auth_trap_key_use and nx_snmp_agent_priv_trap_key_use
6) Start the SNMP agent using the nx_snmp_agent_start API

These common steps are illustrated in a typical operational flow in the following figure:

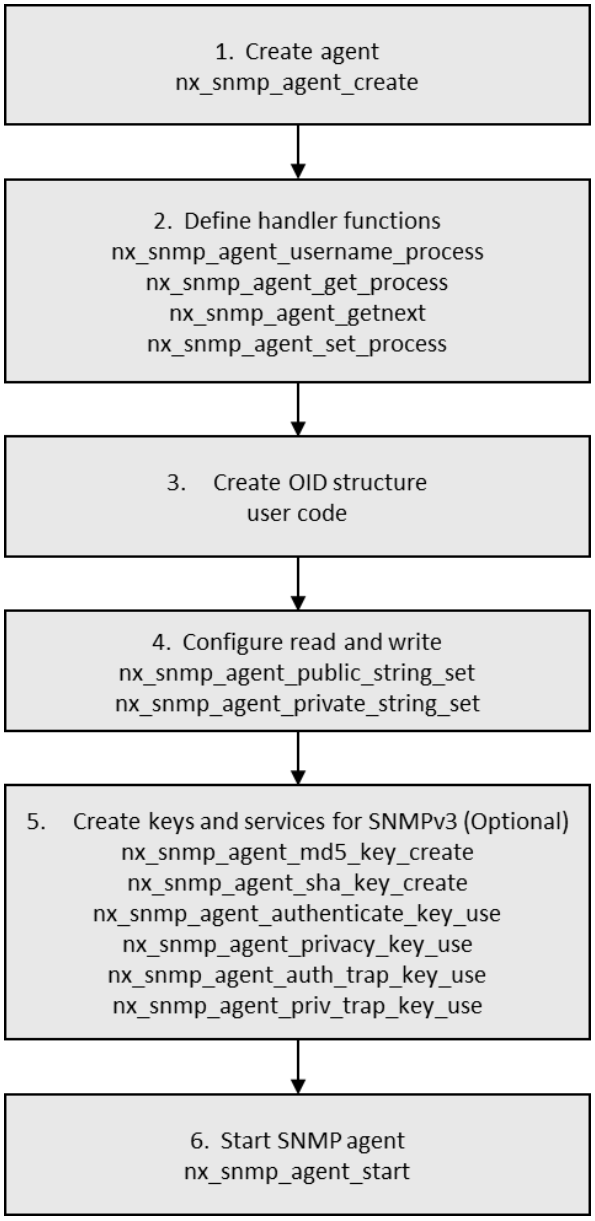


Figure 454: Flow Diagram of a Typical NetX and NetX Duo SNMP Agent Module Application

4.3.9 ThreadX

The Express Logic ThreadX kernel (tx) is integrated into the SSP. For more information about ThreadX, including API reference, refer to the *ThreadX User Guide*.

4.3.9.1 What Does the Express Logic ThreadX Module Do?

The Express Logic ThreadX kernel is the foundation of multi-threaded SSP applications. It provides thread creation and synchronization services, including message queues, counting semaphores, mutexes, event flags, memory block pools, memory byte pools, and application timers.

4.3.9.2 Using e^2 studio to write an Application with Express Logic ThreadX

The Express Logic ThreadX module is integrated into the Synergy Software Package in e^2 studio:e^2 studio ISDE User Guide

In e^2 studio, create and configure a project and add the drivers:

1) Create the project: Creating a Project.
2) Configure the project: Configuring a Project.
3) Add Threads and ThreadX will be added automatically.

During the configuration of the project, ThreadX is added automatically when a new Thread is created. To create a new thread, click **New Thread** in the Threads section of the **Threads** tab of the Synergy Configuration tool (configuration.xml). To add ThreadX source code, add the ThreadX source module as described in the table below.

Resource	ISDE Tab	Selection
Express Logic ThreadX Source Code (optional, for advanced configurations or source level debugging support)	Threads (HAL/Common)	**Framework > RTOS > ThreadX Source**

Adding Objects to Threads

ThreadX objects, including mutexes, semaphores, event flags, and queues, can be added to Threads in the <Thread Name> Objects section, where <Thread Name> represents the name of an application thread. For example, to add a new queue to a thread, highlight the thread and select **New Object > Queue**.

To access the queue from a different thread, include the header file of the thread where the queue is defined. For example, if an application has a thread named main_thread with a queue named message_queue, a file named main_thread.h will be generated with an extern for message_queue. If the application also has a thread named background_thread, and the background_thread requires access to the message_queue, main_thread.h should be included in background_thread_entry.c to ensure background_thread_entry.c has access to the definition of message_queue.

4.3.9.3 Express Logic ThreadX Configuration Notes

IAR Library Support (IAR Only)

Thread-safe support for the IAR tools is easily enabled by making the following setting in the **Properties** of the ThreadX Source Module. In the table, ISDE Property refers to the **Properties** tab name in the e^2 studio ISDE.

ThreadX Source IAR Library Support configuration

ISDE Property	Setting	Description
IAR Library Support	Enabled, Disabled, Default: Disabled	If enabled, defines TX_ENABLE_IAR_LIBRARY_SUPPORT to provide thread safe access to IAR standard library functions such as malloc() and printf().

Also, add the following line to the linker control file (if not already in place):

```
initialize by copy with packing = none { section __DLIB_PERTHREAD };
// Required in a multi-threaded application.
```

@ note Make sure the ThreadX Source module is added to the **Threads** tab (HAL/Common) before modifying TX_ENABLE_IAR_LIBRARY_SUPPORT.

ThreadX Source Advanced Configurations

If the ThreadX Source module is added to the project, the Properties window provides advanced configurations for the ThreadX source library. Highlight a configuration option to view a description of the option in the bottom left corner of the e^2 studio GUI. If the configuration option entry field is empty, the default value will be used. The default values of configuration options are defined in ssp/inc/framework/el/cm<core>_<compiler>/tx_port.h, where <core> is the ARM cortex number of the MCU used ('4' or '0plus') and <compiler> is the compiler used ('gcc' or 'iar'). Refer to the Configuration Options chapter of the *ThreadX User Guide* for more information.

At the end of the advanced properties, there are TX_<COMPONENT>_EXTENSION macros, where <COMPONENT> identifies the type of extension. These extension macros are for advanced use cases only. In most projects, they are not modified. Some cases, such as BSD support for NetX, require use of extension macros and explicitly describe how they should be used.

4.3.9.4 Express Logic ThreadX Limitations

- See SSP Release Notes for known limitations.

4.3.9.5 Express Logic ThreadX Supported Devices

The module is supported on the following families:

- S7G2
- S3A7
- S124

4.3.10 USBX

The Express Logic USBX USB stack (ux) is integrated into the SSP. For more information about the USBX, including API reference, refer to the *USBX User Guide*.

4.3.10.1 What Does the Express Logic USBX Module Do?

USBX supports the two existing USB specifications: 1.1 and 2.0. It is designed to be scalable and will accommodate simple USB topologies with only one connected device as well as complex topologies with multiple devices and cascading hubs. USBX supports both the host side and the device side.

4.3.10.2 Supported USB Classes in Express Logic USBX

The USBX class modules listed below are fully supported in the Synergy Configuration tool. To use these USBX class modules, go to the **Threads** tab of the Synergy Configuration tool (configuration.xml), and select any of the USB class modules:

- ux_device_class_cdc_acm
- ux_device_class_hid
- ux_device_class_storage
- ux_host_class_cdc_acm
- ux_host_class_hid
- ux_host_class_storage
- ux_host_class_video
- ux_host_class_hub

Refer to the *USBX Device Class User Guide* or *USBX Host Class User Guide* for more information about the USBX Class specification.

The USBX class modules listed below are experimental modules for Synergy parts. The experimental modules are not currently supported in Synergy Configuration. To experiment with these USBX class modules, go to the Components tab of the Synergy Configuration tool (configuration.xml), and select any of the USB class modules:

- ux_device_class_cdc_ecm
- ux_device_class_rndis
- ux_host_class_audio
- ux_host_class_gser
- ux_host_class_printer
- ux_host_class_prolific
- ux_host_class_swar
- ux_network_driver

NOTE: Any USB classes shown above are experimental and not yet tested for Synergy parts. Because of that, it is not recommended to use them for product developments.

When the components above are added, a prebuilt library of the application code is added. For each component listed above, there is an analogous component ending in '_src' that contains protected source files. The '_src' component can be added in addition to the prebuilt library module.

4.3.10.3 Using e^2 studio to write an Application with Express Logic USBX

The Express Logic USBX module is integrated into the Synergy Software Package in e^2 studio: e^2 studio ISDE User Guide.

Write an Application with USBX Device Class Stack

To add USBX Device Class modules, go to the **Threads** tab of the Synergy Configuration tool (configuration.xml), and select any of the class modules:

In e^2 studio, create and configure a project and add the drivers:

Step 1. Create the project: Creating a Project.

Step 2. Configure the project: Configuring a Project.

Step 3. Add following either or multiple of USBX Device Class components in Synergy Configuration tool: Adding Drivers to a Thread and Configuring the Drivers.

Resource	ISDE Tab	Selection
USBX Device Class CDC-ACM (mandatory for CDC-ACM)	Threads	**X-Ware > USBX > Device > Classes > CDC-ACM >USBX Device Class CDC ACM**
USBX Device Class HID (mandatory for HID class)	Threads	**X-Ware > USBX > Device > Classes > HID >USBX Device Class HID**
USBX Device Class Mass Storage (mandatory for Mass Storage class)	Threads	**X-Ware > USBX > Device > Classes > Mass Storage >USBX Device Class Mass Storage**

To fill in the USBX class stack in the Synergy Configurator, the following components will be used:

- USBX Device Configuration component (mandatory)
- USBX Interface Configuration component (mandatory)
- USBX on ux (mandatory)

Step 4. Add a USB Port DCD (Device Controller Driver) component to the stack. You may have two choices if the Synergy part has two USB peripherals. Click the component box written as "Add USBX Port DCD". Select **New** and choose one of the USBX Port DCD components (mandatory) below:

- USBX Port DCD on sf_el_ux for USBHS (if available for the parts)
- USBX Port DCD on sf_el_ux for USBFS

Step 5. It is optional but recommended to add Transfer Modules for both of Tx and Rx channels. You may have two choices if the Synergy part has DMAC and DTC. Click the component box "Add Transfer Module for Tx (or Rx)". Select **New** and choose either of Transfer Driver components (optional) below:

- Transfer Driver on r_dmac (if available for the parts)

- Transfer Driver on r_dtc (if available for the parts)(Either of components above is option to use)

Step 6. Users have the option of adding USBX Source if they wish to build from source instead of using the provided USBX library.

Step 7. Configure each component to complete the USB stack configuration. See sections "Express Logic USBX Configuration Notes" and "Express Logic USBX Synergy Port Driver Configuration" in the SF_EL_UX_usage_notes for more inofrmation.

Write an Application with USBX Host Class Stack

The Express Logic USBX module is integrated into the Synergy Software Package in e^2 studio: e^2 studio ISDE User Guide.

In e^2 studio, create and configure a project and add the drivers:

Step 1. Create the project: Creating a Project.

Step 2. Configure the project: Configuring a Project.

Step 3. Add following either or multiple of USBX Host Class components in Synergy Configuration tool: Adding Drivers to a Thread and Configuring the Drivers.

Resource	ISDE Tab	Selection
USBX Host Class CDC-ACM	Threads	**X-Ware > USBX > Host > Classes > CDC-ACM >USBX Host Class CDC ACM**
USBX Host Class HID	Threads	**X-Ware > USBX > Host > Classes > HID >USBX Host Class HID**
USBX Host Class Mass Storage	Threads	**X-Ware > FileX > FileX on USB Mass Storage**
USBX Host Class Video	Threads	**X-Ware > USBX > Host > Classes > Video > USBX Host Class Video**
USBX Host Class HUB	Threads	**X-Ware > USBX > Host > Classes > HUB >USBX Host Class HUB**

To fill in the USBX class stack in the Synergy Configurator, the following components will be used:

- USBX Host Configuration component (mandatory)
- USBX on ux (mandatory)

NOTE: The stack for USBX Host Class Mass Storage is different from the other Class stacks and needs FileX on USB Mass Storage Class on top of it. Select **X-Ware > FileX > FileX on USB Mass Storage** component, not **X-Ware > USBX > Host > Classes > Mass Storage > USBX Host Class Mass Storage**.

Step 4. Add a USB Port HCD (Host Controller Driver) component to the stack. You may have two choices if the Synergy part has two USB peripherals. Click the component box "Add USBX Port HCD". Select **New** and choose one of the USBX Port HCD components (mandatory) listed below:

- USBX Port HCD on sf_el_ux for USBHS (if available for the parts)

- USBX Port HCD on sf_el_ux for USBFS

Step 5. It is optional but recommended to add a Transfer Module for both of Tx and RX channels. You may have two choices if the Synergy part has DMAC and DTC. Click the component box "Add Transfer Module for TX (or RX)". Select **New** and choose one of the Transfer Driver components (optional) listed below:

- Transfer Driver on r_dmac (if available for the parts)
- Transfer Driver on r_dtc (if available for the parts)(Either of components above is option to use)

Step 6. You have the option of adding USBX Source if you wish to build from source instead of using the provided USBX library.

Step 7. Configure each component to complete the USB stack configuration. See sections "Express Logic USBX Configuration Notes" and "Express Logic USBX Synergy Port Driver Configuration" for more information.

Write an Application with Deprecated Modules

In the SSP version 1.2.0 or later, the USBX stack configuration is supported in the Synergy Configuration tool and the few USBX relevant components, which were defined in previous SSP versions are now treated as [DEPRECATED] components. They are kept in the SSP version today to provide you with backward compatibility. Anyone using a new version of SSP and want to keep their existing application code compatible to an existing SSP version, can use [DEPRECATED] components.

In e^2 studio, create and configure a project and add the drivers:

Step 1. Create the project: Creating a Project.

Step 2. Configure the project: Configuring a Project.

Step 3. Add the following components, if required, the same way as was done when using an existing version of SSP: Adding Drivers to a Thread and Configuring the Drivers.

Express Logic USBX Device Class CDC-ACM(option)	**Threads**	**Framework > USB > [DEPRECATED] USBX Device Class CDC-ACM on ux_device_class_cdc_acm**
Express Logic USBX (option)	Threads	**Framework > USB > [DEPRECATED] USBX on ux**

NOTE: Do not use components marked as [DEPRECATED] for new development. Only use these components for existing user applications which were developed with previous SSP releases.

4.3.10.4 Express Logic USBX Configuration Notes

USB Class Stack Configuration Overview

The figure below shows the interface diagram of the USBX device class Stack. The stack consists of one USBX device class components (ux_device_class_xxx) on the top, the USBX (ux) in the middle, and the USBX Port driver (sf_el_ux Device Controller Driver (DCD)) on the bottom of the device class stack. As a recommended option, the SSP Transfer module (r_dmac) supports data transfer between memory and hardware FIFO in Synergy USB peripherals (USBHS or USBFS). To support the USB device stack configuration, there are some components named USBX Device Configuration

and USBX Interface Configuration. These two components do not represent actual software modules in SSP but virtual modules to handle the code generation.

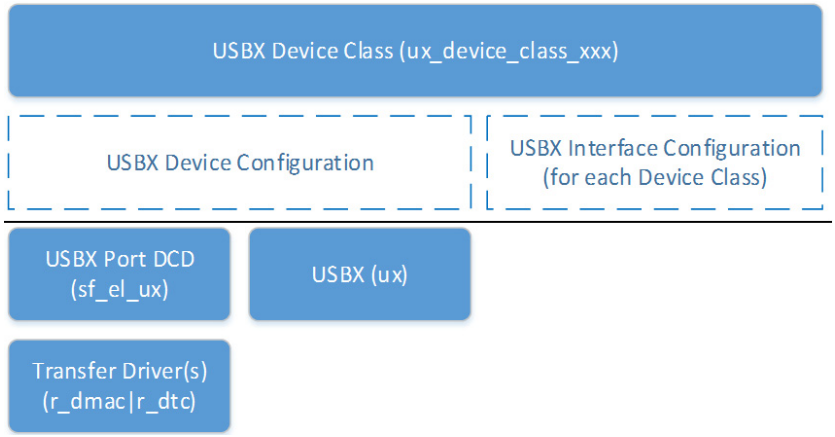


Figure 455: USBX Device Class Stack Configuration

NOTE: Currently r_dtc is not supported for device side driver (only r_dmac is supported)

This figure below shows the interface diagram of the USBX host class Stack. The stack consists of one of the USBX host class components (ux_host_class_xxx) on the top, the USBX (ux) in the middle, and the USBX Port driver component (sf_el_ux Host Controller Driver (HCD)) on the bottom of the host class stack. As a recommended option, the SSP Transfer module (r_dmac or r_dtc) supports data transfer between memory and hardware FIFO in Synergy USB peripherals (USBHS or USBFS). To support the USB host stack configuration, there is a component named USBX host configuration, which is a virtual component to handle the code generation:

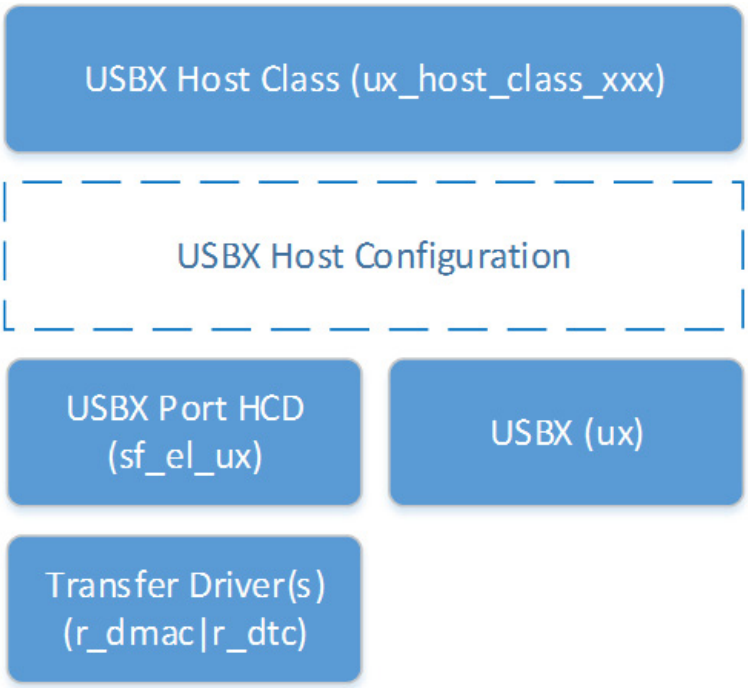


Figure 456: USBX Host Class Stack Configuration

USB Common Configuration

Write an Application with USBX Host Stack

The Express Logic USBX module is integrated into the Synergy Software Package in e^2 studio: e^2 studio ISDE User Guide.

In e^2 studio, create and configure a project and add the drivers:

Step 1. Create the project: Creating a Project.

Step 2. Configure the project: Configuring a Project.

Step 3. Add following either or multiple of USBX Host Class components in Synergy Configuration tool: Adding Drivers to a Thread and Configuring the Drivers.

Resource	ISDE Tab	Selection
USBX Host Configuration	Threads	**X-Ware > USBX > Host > Common>USBX Host Configuration**

To fill in the USBX stack in the Synergy Configurator, the following components will be used:

- USBX on ux (mandatory)
- Add a USB Port HCD (Host Controller Driver) component to the stack. You may have two choices if the Synergy part has two USB peripherals. Click the component box "Add USBX Port HCD". Select **New** and choose one of

the USBX Port HCD components (mandatory) listed below: USBX Port HCD on sf_el_ux for USBHS (if available for the parts)

- USBX Port HCD on sf_el_ux for USBFS

Step 4. It is optional but recommended to add a Transfer Module for both of Tx and RX channels. You may have two choices if the Synergy part has DMAC and DTC. Click the component box "Add Transfer Module for TX (or RX)". You may select **New** and choose other Transfer Driver component (optional) listed below:

- Transfer Driver on r_dmac (if available for the parts)
- Transfer Driver on r_dtc (if available for the parts)(Either of components above is option to use)

Step 5. You have the option of adding USBX Source if you wish to build from source instead of using the provided USBX library.

Step 6. Configure each component to complete the USB stack configuration. See sections "Express Logic USBX Synergy Port Driver Configuration" for more information.

The USBX on ux component has configurations for USBX core library module as shown in the table below.

USBX on ux

Configuration	Settings	Description
USBX Pool Memory Name	Arbitrary symbol name (Default: "g_ux_pool_memory0")	Name must be a valid C symbol.
USBX Pool Memory Size	Any integer value (Default: 18432)	See section "Express Logic USBX Memory Requirements" for the required memory size for each classes.
User Callback for Host Event Notification (Only valid for USB Host)	Arbitrary symbol name (Default: "NULL")	Name must be a valid C symbol. The name of User defined USBX Host event notification can be given to this property.

USB Device Classes Configuration

The USBX Device Class CDC-ACM component has configurations to setup USB Device Class CDC-ACM shown in the table below. It can be configured through the Synergy Configuration tool.

USBX Device Class CDC-ACM configuration

Configuration	Settings	Description
Name	Arbitrary symbol name (Default: "g_ux_device_class_cdc_acm0")	Specify the name of USBX Device CDC-ACM Class module instance. It must be a valid C symbol.

Configuration	Settings	Description
USBX CDC-ACM instance_activate Function Callback	Arbitrary symbol name (Default: "ux_cdc_device_instance_activate")	Specify the name of instance_activate user callback function for the USBX Device CDC-ACM Class module. Name must be a valid C symbol. See the USBX Stack User's Manual *Chapter 5: USBX Device Class Considerations USB Device CDC-ACM Class* for more information about the instance_activate callback function.
USBX CDC-ACM instance_deactivate Function Callback	Arbitrary symbol name (Default: "ux_cdc_device_instance_deactivate ")	Specify the name of instance_deactivate user callback function for the USBX Device CDC-ACM Class module. Name must be a valid C symbol. Refer to the USBX Stack User's Manual *Chapter 5: USBX Device Class Considerations USB Device CDC-ACM Class* for more information about the instance_activate callback function

The USBX Device Class Mass Storage component has configurations to setup USB Device Mass Storage Class shown in the table below. Parameters are available to configure through the Synergy Configuration tool.

USBX Device Class Mass Storage Configuration

Configuration	Settings	Description
Name	Arbitrary symbol name (Default: "g_ux_device_class_storage")	Specify the name of USBX Interface Descriptor for Mass Storage Class. It must be a valid C symbol.
Mass Storage Class Parameter Setup	Auto (Default), Manual	Manual (User Manual Setup if LUN is greater than 1). Simple Auto Setup if LUN is 1.
User Setup Callback (Only valid if Parameter Setup is Auto)	Arbitrary C symbol name (Default: ux_device_class_storage_user_setup)	Specify the name of user callback function to setup the storage parameter setup. This parameter is only valid when the configuration "Mass Storage Class Parameter Setup" is "Auto".

Configuration	Settings	Description
Last LBA of Storage Media	Arbitrary integer value (Default:0)	Specify the last LBA of storage media device (the number of sectors available in the media - 1). This parameter is only valid when the configuration "Mass Storage Class Parameter Setup" is "Auto".
Bytes Per Sector of Storage Media	Arbitrary integer value, which is multiple of 512 (Default: 512).	Specify the sector size of storage media. It can take multiple of 512 such as 512, 4K bytes. This parameter is only valid when the configuration "Mass Storage Class Parameter Setup" is "Auto".
Type of Storage Media	Arbitrary integer value (Default: 0).	Specify the type of storage media device. Typically, the value takes following values. Flash Drive (0), CD-ROM device (5) This parameter is only valid when the configuration "Mass Storage Class Parameter Setup" is "Auto".
Removable Flag of Storage Media	Arbitrary integer value (Default: 0x80).	Specify the Removable Flag value of Storage Media. This parameter is only valid when the configuration "Mass Storage Class Parameter Setup" is "Auto".
Media Read Function Callback	Arbitrary C symbol name (Default: ux_device_msc_media_read)	Specify the C symbol name of Media Read callback for the USBX Device Mass Storage Class. The function is to be called back from the Class library when read access to the USB storage device is requested from user application. Refer USBX Stack User's Manual *Chapter 5: USBX Device Class Considerations USB Device Storage Class* for the function definition.

Configuration	Settings	Description
Media Write Function Callback	Arbitrary C symbol name (Default: ux_device_msc_media_write)	Specify the C symbol name of Media Write callback for the USBX Device Mass Storage Class. The function is to be called back from the Class library when write access to the USB storage device is requested from user application. Refer USBX Stack User's Manual *Chapter 5: USBX Device Class Considerations USB Device Storage Class* for the function definition.
Media Status Function Callback	Arbitrary C symbol name (Default: ux_device_msc_media_status)	Specify the C symbol name of Media Status callback for the USBX Device Mass Storage Class. The function is to be called back from the Class library when status inquiry to the USB storage device is requested from user application. Refer USBX Stack User's Manual *Chapter 5: USBX Device Class ConsiderationsUSB Device Storage Class* for the function definition.
USBX Device Storage Instance_Activate Callback Function	Arbitrary symbol name (Default: "NULL")	Specify the name of instance_activate user callback function for the USBX Device Mass Storage Class module. Name must be a valid C symbol. See the USBX Stack User's Manual *Chapter 5: USBX Device Class Considerations USB Device Storage Class* for more information about the instance_activate callback function.
USBX Device Storage Instance_Deactivate Callback Function	Arbitrary symbol name (Default: "NULL")	Specify the name of instance_deactivate user callback function for the USBX Device Mass Storage Class module. Name must be a valid C symbol. Refer to the USBX Stack User's Manual *Chapter 5: USBX Device Class Considerations USB Device Storage Class* for more information about the instance_activate callback function

The USBX Device Class HID component has the configurations to setup USB Device HID Class shown in the table below. Parameters are available to configure through the Synergy Configuration tool.

USBX Device Class HID Configuration

Configuration	Settings	Description
Name	Arbitrary symbol name (Default: "g_ux_device_class_hid")	Specify the name of USBX Interface Descriptor for HID Class. It must be a valid C symbol.
USBX HID User Callback Function	Arbitrary symbol name (Default: "ux_hid_device_callback")	Specify the name of a user callback function to get an event from HID Class. Name must be a valid C symbol. Refer to the USBX Stack User's Manual *Chapter 5: USBX Device Class Considerations USB Device HID Class* for more information about the user callback function.
USBX Device HID Entry Function	Arbitrary symbol name (Default: "NULL")	Specify the name of user entry function for the USBX Device HID Class module. Name must be a valid C symbol. See the USBX Stack User's Manual *Chapter 5: USBX Device Class Considerations USB Device HID Class* for more information about the user entry function.
USBX Device HID instance_activate Function Callback	Arbitrary symbol name (Default: "NULL")	Specify the name of instance_activate user callback function for the USBX Device HID Class module. Name must be a valid C symbol. See the USBX Stack User's Manual *Chapter 5: USBX Device Class Considerations USB Device HID Class* for more information about the instance_activate callback function.
USBX Device HID instance_deactivate Function Callback	Arbitrary symbol name (Default: "NULL")	Specify the name of instance_deactivate user callback function for the USBX Device HID Class module. Name must be a valid C symbol. Refer to the USBX Stack User's Manual *Chapter 5: USBX Device Class Considerations USB Device HID Class* for more information about the instance_activate callback function

Configuration	Settings	Description
Multiple HID Report support	Enable or Disable (Default)	Set Enable to support multiple HID report. This configuration is used to indicate which data fields are represented in each report structure.

USB Device Descriptor Configuration

The USBX Device Configuration component has configurations shown in the table below to auto-generate the USB Device Descriptor and USB Configuration Descriptor. Refer to http://www.usb.org and download USB 2.0 Specification for more information about the specification of USB Device Descriptor and or Configuration Descriptor.

USBX Device Configuration

Configuration	Settings	Description
Vendor ID	16-bit arbitrary number (Default: 0x045B)	Specify Vendor ID assigned by USB-IF. This configuration is a part of the USB Device Descriptor (**idVendor**).
Product ID	16-bit arbitrary number (Default: 0x0000)	Specify Product ID assigned by manufacturer. This configuration is a part of the Device Descriptor (**idProduct**).
Device Release Number	16-bit arbitrary number (Default: 0x0000)	Specify Device Release Number in binary-coded decimal. This configuration is a part of the USB Device Descriptor (**bcdDevice**).
Index of Manufacturer String Descriptor	Arbitrary number from 0 to 255 (Default: 0x00)	Specify the Index of Manufacturer String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (**iManufacturer**). Set zero if String Descriptor is not used. See section USBX-String-Framework-Configuration for more information.
Index of Product String Descriptor	Arbitrary number from 0 to 255 (Default: 0x00)	Specify the Index of Product String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (**iProduct**). Set zero if String Descriptor is not used. See section "USBX String Framework Configuration" for more information.

Configuration	Settings	Description
Index of Serial Number String Descriptor	Arbitrary number from 0 to 255 (Default: 0x00)	Specify the Index of Serial Number String Descriptor defined in the USBX String Framework. This configuration is a part of the USB Device Descriptor (**iSerialNumber**). Set zero if the String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Class Code	Device(0x00), Communications(CDC) (0x02, Default), HID (0x03), Mass Storage (0x08), Miscellaneous (0xEF), Vendor Specific (0xFF)	Select the USB Device Class Code. This configuration is a part of the USB Configuration Descriptor (**bDeviceClass**).
Index of String Descriptor describing this configuration	Arbitrary number from 0 to 255 (Default: 0x00)	Specify the Index of String Descriptor describing this configuration. This configuration is a part of the USB Configuration Descriptor (**iConfiguration**). Set zero if String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Size of USB Descriptor in bytes for this configuration	0x00 (Default) or value to be set to wTotalLength (calculated by user)	Specify the size of USB Descriptor in bytes. Modify the value for Vendor-specific Class, otherwise you can set zero to calculate the size automatically in the auto-generated code from Synergy Configuration tool. This configuration is a part of the USB Configuration Descriptor (**wTotalLength**).
Number of Interfaces	0x00 (Default) or value to be set to bNumInterfaces (calculated by user).	Specify the Number of interfaces supported by this configuration. Modify the value for Vendor-specific Class, otherwise you can set zero to calculate the value automatically in the auto-generated code from Synergy Configuration tool. This configuration is a part of the USB Configuration Descriptor (**bNumInterfaces**).

Configuration	Settings	Description
Self-Powered	Enable (Default) or Disable	Enable this configuration if your USB Device is a self- powered device. This configuration is a part of the USB Configuration Descriptor (**bmAttributes** bit6)
Remote Wakeup	Enable or Disable (Default)	Enable this configuration if your USB Device supports remote wakeup. This configuration is a part of the USB Configuration Descriptor (**bmAttributes** bit5)
Maximum Power Consumption in 2mA units(0-250)	50 (Default), Integer value from 0 to 250.	Set the maximum power consumption of your device to indicate the amount of bus power required. This configuration is 2mA units, thus, the maximum 500 mA can be specified. This configuration is a part of the USB Configuration Descriptor (**bMaxPower**).
Supported Language Code	16-bit number assigned by Manufacturer (Default: 0x0409)	Specify the Language ID Code. For example, 0x0409 English - United States. This configuration is used for Language ID Framework code generation. See section "USBX Language Framework Configuration" for more information.
Name of USBX String Framework	Arbitrary C language symbol (Default : NULL)	Specify the name of user defined USBX String Framework. This must be a valid C symbol. Set NULL if the String Descriptor is not used. See section "USBX String Framework Configuration" for more information.
Total index number in USB String Descriptor in USBX String Framework	Arbitrary integer number (Default : 0)	Specify the total number of index for String Descriptor. See section "USBX String Framework Configuration" for more information.
Name of USB Language Descriptor	Arbitrary C language symbol (Default : NULL)	Specify the name of user defined USBX Language Framework. This must be a valid C symbol. If '0' is set to the property "Total Number of Language Support", this configuration is ignored. See section "USBX Language Framework Configuration" for more information.

Configuration	Settings	Description
Total Number of Language Support	Arbitrary integer number (Default : 0)	Specify the total number of languages to support. See section "USBX String Framework Configuration" for more information. If '0' is set here, US English (0x0409) is applied as the default language.

The USBX Interface Configuration CDC-ACM component has configurations shown in the table below to setup the USB Interface Descriptor for CDC-ACM. It can be configured through the Synergy Configuration tool. Refer to http://www.usb.org and download USB 2.0 Specification for more information about the specification of USB Interface Descriptor.

USBX Interface Descriptor for CDC-ACM

Configuration	Settings	Description
Name	Arbitrary symbol name(Default: "g_usb_interface_desc_cdcacm_0")	Specify the name of USBX Interface Descriptor for CDC-ACM. It must be a valid C symbol.
Interface Number of Communications Class interface	Arbitrary integer value from 0 to 255 (Default: 0x00)	Specify the index number of Communications Class interface. This configuration is a part of the USB Interface Descriptor (**bInterface**). The number must not be duplicated with the Interface Number of Data Class interface. Also must not be duplicated with any Interface Numbers if your USB device consists of a USB composite device.
Interrupt Transfer Endpoint to use for Communication Class.	End Point N, where N=1 to 9. (Default: End Point 3)	Specify the Endpoint Number of Interrupt Endpoint. It must not be duplicated with ones for the other Endpoints.
Polling period for Interrupt Endpoint (in mS/125us units for FS/HS)	Arbitrary integer value from 1 to 255 (Default: 0x0F)	Specify the Interval for polling Endpoint transfers. This configuration is valid for Interrupt Endpoint and ignored for Bulk Endpoints. Value is in frame counts (1ms units for FS mode and 125us units for HS mode).

Configuration	Settings	Description
Interface Number of Data Class interface	Arbitrary integer value from 0 to 255 (Default: 0x01)	Specify the index number of Data Class interface. This configuration is a part of the USB Interface Descriptor (**bInterface**). The number must not be duplicated with the Interface Number of Communications Class interface. Also must not be duplicated with any Interface Numbers if your USB device consists of a USB composite device.
Bulk In Transfer Endpoint to use for Data Class	End Point N, where N=1 to 9. (Default: End Point 1)	Specify the Endpoint Number of Bulk In Endpoint. It must not be duplicated with ones for the other Endpoints.
Bulk Out Transfer Endpoint to use for Data Class	End Point N, where N=1 to 9. (Default: End Point 2)	Specify the Endpoint Number of Bulk Out Endpoint. It must not be duplicated with ones for the other Endpoints.
Index of String Descriptor Describing Communications Class interface	Arbitrary integer number (Default : 0x00)	Specify the index number of String Descriptor Describing Communications Class interface. This configuration is a part of the USB Interface Descriptor (**iInterface**). Set '0' if do not have String information for the interface.
Index of String Descriptor Describing Data Class interface	Arbitrary integer number (Default : 0x00)	Specify the index number of String Descriptor Describing Data Class interface. This configuration is a part of the USB Interface Descriptor (**iInterface**). Set '0' if do not have String information for the interface.

The USBX Interface Configuration Mass Storage component has configurations shown in the table below to setup the USB Interface Descriptor for Mass Storage Class. Parameters are available to configure through the Synergy Configuration tool. Refer to http://www.usb.org and download USB 2.0 Specification for more information about the specification of USB Interface Descriptor.

USBX Interface Descriptor for Mass Storage Class

Configuration	Settings	Description
Name	Arbitrary symbol name (Default: "g_usb_interface_desciptor_storage0 ")	Specify the name of USBX Interface Descriptor for CDC-ACM. It must be a valid C symbol.

Configuration	Settings	Description
Interface Number of Bulk-Only Data Interface	Arbitrary integer value from 0 to 255 (Default: 0x00)	Specify the index number of Bulk-Only Data interface. This configuration is a part of the USB Interface Descriptor (**bInterface**). The number must not be duplicated with any other Interface Numbers if your USB device consists of a USB composite device.
Endpoint Number of Bulk-Out Transfer	End Point N, where N=1 to 9. (Default: End Point 1)	Specify the Endpoint Number of Bulk Out Endpoint. It must not be duplicated with ones for the other Endpoints.
Endpoint Number of Bulk-In transfer	End Point N, where N=1 to 9. (Default: End Point 2)	Specify the Endpoint Number of Bulk In Endpoint. It must not be duplicated with ones for the other Endpoints.

The USBX Interface Configuration HID component has configurations shown in the table below to setup the USB Interface Descriptor for HID Class. Parameters are available to configure through the Synergy Configuration tool. Refer to http://www.usb.org and download USB 2.0 Specification for more information about the specification of USB Interface Descriptor.

USBX Interface Descriptor for HID Class

Configuration	Settings	Description
Name	Arbitrary symbol name (Default: "g_usb_interface_desc_hid0")	Specify the name of USBX Interface Descriptor for CDC-ACM. It must be a valid C symbol.
Interface Number of HID Class Interface	Arbitrary integer value from 0 to 255 (Default: 0x00)	Specify the index number of HID Class interface. This configuration is a part of the USB Interface Descriptor (**bInterface**). The number must not be duplicated with any other Interface Numbers if your USB device consists of a USB composite device.
Protocol Code	0 (None), 1 (Keyboard, Default), 2 (Mouse)	Specify the HID protocol code from options. This configuration is a part of the USB Interface Descriptor. (**bInterfaceProtocol**). Available protocol codes are 0 (None), 1 (Keyboard, Default), 2 (Mouse).

Configuration	Settings	Description
Endpoint Number of Interrupt-In Transfer	End Point N, where N=1 to 9. (Default: End Point 1)	Specify the Endpoint Number of Interrupt-In Endpoint. It must not be duplicated with ones for the other Endpoints.
Maximum packet size in bytes for Interrupt in Endpoint	Arbitrary integer value from 1 to 1024 (Default: 0x08)	Specify the maximum packet size this endpoint is capable of sending or receiving when this configuration is selected.
Interval for polling Interrupt-In Endpoint (in mS/125us units for FS/HS)	Arbitrary integer value from 1 to 255 (Default: 0x08)	Specify the Interval for polling Endpoint transfers. This configuration is valid for Interrupt-In Endpoint. Value is in frame counts (1ms units for FS mode and 125us units for HS mode).
Interrupt-Out Endpoint (Optional)	Enable or Disable (Default)	This configuration is reserved and currently not used.
Endpoint Number of Interrupt-Out transfer (Optional)	End Point N, where N=1 to 9. (Default: End Point 2)	This configuration is reserved and currently not used.
Maximum packet size in bytes for Interrupt-Out Endpoint (Optional)	Arbitrary integer value from 1 to 1024 (Default: 0x08)	This configuration is reserved and currently not used.
Interval for polling Interrupt-Out Endpoint for data transfers (in mS/125us units for FS/HS) (Optional)	Arbitrary integer value from 1 to 255 (Default: 0x08)	This configuration is reserved and currently not used.

The USBX String Framework is byte stream data to provide USB device information with human readable strings to the USB host device. Users need to define the byte stream data in their application code if required. It is optional and not mandatory to use. See the USBX Device Class User Guide, section *Definition of the Strings of the Device Framework* for more information about the USBX String Framework. Here is an example of USBX String Framework, which consists of three indexes String descriptors.

```
const UCHAR g_usb_string_framework[] =
{
    /* Index #1 (this example shows manufacturer information) */
    (uint8_t) (0x0409),     /* Byte0 Language Code, US English */
    (uint8_t) (0x0409 >> 8),    /* Byte1 Language Code */
    0x01,               /* Byte2 Index */
    7,              /* Byte3 Length */
    'R', 'E', 'N', 'E', 'S', 'A', 'S',

    /* Index #2 (this example shows product information) */
    (uint8_t) (0x0409),     /* Byte0 Language Code, US English */
    (uint8_t) (0x0409 >> 8),    /* Byte1 Language Code */
    0x02,               /* Byte2 Index */
```

```
    10,                 /* Byte3 Length */
    'C', 'O', 'M', ' ', 'D', 'E', 'V', 'I', 'C', 'E',

    /* Index #3 (this example shows Device Serial Number information) */
    (uint8_t) (0x0409),     /* Byte0 Language Code, US English */
    (uint8_t) (0x0409 >> 8),    /* Byte1 Language Code */
    0x03,               /* Byte2 Index */
    4,              /* Byte3 Length */
    '0', '1', '0', '0'
};
```

You can configure the properties of the USBX Device Configuration component as shown in the table below. Refer to for more information about each configuration in the table.

Example of the USBX String Framework Configuration

Configurations for String Descriptor on Synergy Configuration tool	Setting Example
Name of USB String Framework	g_usb_string_framework
Total index number of USB String Descriptor in USBX String Framework	3 (3 indexes)
Index of Manufacturer String Descriptor	1 (Index #1)
Index of Product String Descriptor	2 (Index #2)
Index of Serial Number String Descriptor	3 (Index #3)
Index of String Descriptor describing this configuration	0 (no string information)
Index of String Descriptor Describing Communications Class interface	0 (no string information)

The USBX Languages Framework is byte stream data to support multiple languages. You need to define the byte stream data in your application code if required. It is optional and not mandatory to use. See the USBX Device Class User Guide, section *Definition of the Languages Supported by the Device for each String* for more information. Below is an example of USBX Language Framework, which supports two Languages:

```
const UCHAR g_usb_language_framework[] =
{
    /* US English */
    (uint8_t) (0x0409),
    (uint8_t) (0x0409 >> 8),
    /* Japanese */
    (uint8_t) (0x0411),
    (uint8_t) (0x0411 >> 8),
```

```
};
```

You can configure the properties of USBX Device Configuration component as shown in the table below. See section "USBX Device Configuration for more information.

Example of the USBX Language Framework Configuration

Configurations for String Descriptor on Synergy Configuration tool	Setting Example
Name of USB Language Descriptor	g_usb_language_framework
Total Number of Language Support	2 (2 languages)

USB Host Classes Configuration

The USBX Host Configuration component has a configuration shown in the table below to auto-generate the USBX host stack initialization code.

USBX Host Configuration

Configuration	Settings	Description
Name	Arbitrary symbol name (Default: "g_ux_host_0")	Specify the name of USBX Host Configuration instance. It must be a valid C symbol.

The USBX Host Class CDC-ACM component has configurations to setup the USB Host Class CDC-ACM shown in the table below. It can be configured through the Synergy Configuration tool.

USBX Host Class CDC-ACM configuration

Configuration	Settings	Description
Name	Arbitrary symbol name (Default: "g_ux_host_class_cdc_acm0")	Specify the name of USBX Host CDC-ACM Class module instance. It must be a valid C symbol.

The USBX Host Class Mass Storage component has configurations to setup the USB Host Class Mass Storage shown in the table below. It can be configured through the Synergy Configuration tool.

USBX Host Class Mass Storage configuration

Configuration	Settings	Description
Name	Arbitrary symbol name (Default: "g_ux_host_class_storage0")	Specify the name of USBX Host Mass Storage Class module instance. It must be a valid C symbol.

The USBX Host Class HID component has configurations to setup the USB Host Class HID shown in the table below. It can be configured through the Synergy Configuration tool.

USBX Host Class HID configuration

Configuration	Settings	Description
Name	Arbitrary symbol name (Default: "g_ux_host_class_hid0")	Specify the name of USBX Host Mass HID module instance. It must be a valid C symbol.
HID Client – Keyboard Support	Enable(Default) / Disable	Set Enable to support USB keyboard devices. This configuration registers the USBX HID Keyboard Client.
HID Client – Mouse Support	Enable(Default) / Disable	Set Enable to support USB mouse devices. This configuration registers the USBX HID Mouse Client.
HID Client – Remote Control Support	Enable(Default) / Disable	Set Enable to support USB remote control devices. This configuration registers the USBX Remote Control Client.

The USBX Host Class Video component has configurations to setup the USB Host Class Video shown in the table below. It can be configured through the Synergy Configuration tool.

USBX Host Class Video configuration

Configuration	Settings	Description
Name	Arbitrary symbol name (Default: "g_ux_host_class_video0")	Specify the name of USBX Host Video Class module instance. It must be a valid C symbol.

USBX Port Driver Configuration

The information for the USBX Port Driver, see section "Express Logic USBX Driver Interface."

USBX Source Advanced Configurations

If the USBX Source components are added to Stack view in the Threads tab in the Synergy Configuration tool, the Properties window provides advanced configurations for the USBX source library. Highlight a configuration option to view a description of the option in the bottom left corner of the e^2 studio GUI. If the configuration option is empty, the default value is used. See the Configuration Options chapter of the *USBX User Guide* for more information.

USBX on ux Source Configurations

Configuration	Settings	Description
Maximum Classes	Not valid	UX_MAX_CLASSES This is not valid configuration so deprecated in future release.
Maximum Slave Classes	Not valid	UX_MAX_SLAVE_CLASSES This is not valid configuration so deprecated in future release.
Maximum Slave Interfaces	Value must be greater than 0 or empty (Default 4)	UX_MAX_SLAVE_INTERFACES When set, this value is the maximum number of interfaces in the device framework.
Maximum Host Controllers	Value must be greater than 0 or empty (Default 2)	UX_MAX_HCD When set, this value represents the number of different host controllers available in the system. For USB 1.1 support, this value will usually be 1. For USB 2.0 support, this value can be more than 1. This value represents the number of concurrent host controllers running at the same time. In Synergy implementation, the value can be set to 2 if running both USBHS and USBFS controllers for S7 or S5 parts.
Maximum Devices	Value must be greater than 0 or empty (Default 8)	UX_MAX_DEVICES When set, this value represents the maximum number of devices that can be attached to the USB. Note that this value represents the total number of devices regardless of the number of USB buses in the system.

Configuration	Settings	Description
Maximum Slave Logical Units	Value must be greater than 0 or empty (Default 2)	UX_MAX_SLAVE_LUN When set, this value represents the current number of SCSI logical units represented in the device storage class driver.
Maximum Host Logical Units	Value must be greater than 0 or empty (Default 16)	UX_MAX_HOST_LUN When set, this value represents the maximum number of SCSI logical units represented in the host storage class driver.
Slave Request Control Maximum Length	Value must be greater than 0 or empty (Default 256)	UX_SLAVE_REQUEST_CONTROL_MAX_LENGTH When set, this value represents the maximum number of bytes received on a control endpoint in the device stack. The default is 256 bytes but can be reduced in memory constraint environments.
Enforce Safe Alignment	Enabled/ Disabled(Default)	UX_ENFORCE_SAFE_ALIGNMENT When enabled, this value forces the memory allocation scheme to enforce alignment of memory with the UX_SAFE_ALIGN field.

USBX Device Class Mass Storage Source Configurations

Configuration	Settings	Description
Maximum number of SCSI logical units	Value must be greater than 0 or empty (Default 2)	UX_MAX_SLAVE_LUN This value represents the maximum number of SCSI logical units represented in the device storage class driver.

USBX Host Class Mass Storage Source Configurations

Configuration	Settings	Description
Use FileX stub	Enabled/Disable(Default)	UX_NO_FILEX Enable this option only in cases where FileX is not available to use.

Configuration	Settings	Description
Maximum number of SCSI logical units	Value must be greater than 0 or empty (Default 1)	UX_MAX_HOST_LUN The value represents the maximum number of SCSI logical units represented in the host storage class driver.
Maximum number of storage media instance	Value must be greater than 0 or empty (Default 1)	UX_HOST_CLASS_STORAGE_MAX_MEDIA The value represents the maximum number of storage media instances represented in the host storage class driver.
Storage memory size in bytes for FileX used for data transfer	Value must be greater than 512 or empty (Default 1024)	UX_HOST_CLASS_STORAGE_MEMORY_BUFFER_SIZE The value represents the block of memory in bytes for FileX to use when doing transfers. The value can be changed to save on memory space but should not be smaller than the media sector size. Because USB devices are SCSI devices and there is a great deal of overhead when doing read/writes, it is better to increase it for higher data throughput.
Maximum transfer size in bytes in one BOT data-transport phase	Value must be greater than 512 or empty (Default 1024)	UX_HOST_CLASS_STORAGE_MAX_TRANSFER_SIZEThe value represents maximum size of memory chunk in bytes to send in one data-transport phase in the Bulk-Only Transport protocol. Large size of data transfer request is split into smaller chunks specified by this configuration. It is better to increase it for higher data throughput.
Stack size for the Mass Storage Class internal thread	Value must be greater than 512 or empty (Default 1024)	UX_HOST_CLASS_STORAGE_THREAD_STACK_SIZEThe value represents the stack size in bytes for Mass Storage Class internal thread named ux_storage_thread.
Timeout in millisecond for a BOT transfer request	Value must be greater than 0 or empty (Default 100000)	UX_HOST_CLASS_STORAGE_TRANSFER_TIMEOUT_IN_MS The value represents timeout value in millisecond for a data transfer request in Bulk-Only Transport protocol.

Configuration	Settings	Description
Timeout in millisecond for the status from a command in the Control/Bulk/ Interrupt transport	Value must be greater than 0 or empty (Default 30000)	UX_HOST_CLASS_STORAGE_CBI _STATUS_TIMEOUT_IN_MS The value represents timeout value in millisecond for the status from a command, which is returned by the interrupt endpoint in the Control/Bulk/ Interrupt transport. The transport is mainly used by storage devices that have very slow read/write commands.
Show linkage warning	Enabled(Default)/Disabled	Notification message for users will be shown if "Enabled" option is selected. This is just to warn users possible linkage errors by multiple symbol definitions. Select "Disabled" stops the notification message.

USBX Host Class HID Source Configurations

Configuration	Settings	Description
HID Keyboard Thread Priority	Value must be greater than 0 or empty (Default 20)	UX_THREAD_PRIORITY_KEYBOA RD Define the priority of the HID keyboard thread.
Memory size for HID Report Decompression	Value must be greater than 0 or empty (Default 4096)	UX_HOST_CLASS_HID_DECOMPR ESSION_BUFFERDefine the memory size to build a decompressed report.
Number of Entries for HID Local Usage Item Table	Value must be greater than 0 or empty (Default 1024)	UX_HOST_CLASS_HID_USAGES Define the size of HID local usage item table. One item entry consumes 4bytes.
Show linkage warning	Enabled(Default)/Disabled	Notification message for users will be shown if "Enabled" option is selected. This is just to warn users possible linkage errors by multiple symbol definitions. Select "Disabled" stops the notification message.

No any special property is given for this module.

Device-Only Size Optimization

The Express Logic USBX module is built in device-only mode to reduce code size for Synergy S1 parts. The configuration (UX_SYSTEM_DEVICE_ONLY) is applied automatically if the S1 board is selected in the Synergy Configuration tool BSP tab. Note that following configurations are fixed in the device-only mode.

- UX_THREAD_STACK_SIZE=512
- UX_SLAVE_REQUEST_DATA_MAX_LENGTH=512

4.3.10.5 Express Logic USBX Auto-generated Code Procedures

NOTE: Code samples in this section are subject to change. No any warranty for contents and possible changes.

USBX Device Stack Auto-generated Code Procedures

The code sample below is pseudo code for USBX Device Classes. The function call sequence is auto-generated from the Synergy Configuration tool if one of the USBX Device Class component or multiple of them are added to the Synergy Configuration tool. Auto-generated code is emitted to common_data.c file and handles the feature listed below:

- Generate the USB Device Descriptor.
- Initialize the USBX software contexts in the memory pool.
- Initialize the USBX Device Stacks added in Synergy Configuration tool.
- Initialize Synergy USB controller(s) in device mode. The USBX Device stack supports Many-to-one topology between device classes and a device controller. For instance, Two USBX Device Classes, which consist of a USB composite device can use single USB controller.

  NOTE: The USBX Device stack does not support to use multiple USB controllers simultaneously. Either of USB controllers is used at once even if the Synergy part has multiple USB controllers (like S7G2 parts).

```
// Interrupt vector registering for USBHS or USBFS controller.
SSP_VECTOR_DEFINE_UNIT();

void g_common_init(void)
{
    // Initialize USBX Memory.
    ux_system_initialize ();

    // Initialize USBX Device stack.
    ux_device_stack_initialize ();

    // Register the Device CDC-ACM Class if the class is used.
    ux_device_stack_class_register();

    // Register the Device HID Class if the class is used.
    ux_device_stack_class_register();
```

```
    // Register the Device Mass Storage Class if the class is used.
    ux_device_stack_class_register();

    // Initialize the USB Device Controller. This function calls either of
    // _ux_dcd_synergy_initalize() or
    // _ux_dcd_synergy_initalize_transfer_support()
    ux_dcd_initialize ();
}
```

USBX Host Stack Auto-generated Code Procedures

The code sample below is pseudo code for USBX Host Classes. The function call sequence is auto-generated from the Synergy Configuration tool if one of USBX Host Class component or multiple of them are added to the Synergy Configuration tool. Auto-generated code is emitted to common_data.c file and handles the features listed below:

- Initialize the USBX software contexts in the memory pool.
- Initialize the USBX Host Stacks added in Synergy Configuration tool. Multi-instances of host class stacks are allowed to be built.
- Initialize the USBX HID Clients if the USBX Host HID Class is added in Synergy Configuration tool.
- Initialize Synergy USB controller(s) in host mode. The USBX Host stack supports Many-to-one or Many-to-many topology between host classes and host controllers. For instance, Two USBX Host Classes can use single USB controller, or Two USBX Host Classes can use different USB controllers individually if the Synergy part has multiple USB controllers (like S7G2 parts).
- Get a USBX Host Class container for user application.

```
// Interrupt vector registering for USBHS or USBFS controller.
SSP_VECTOR_DEFINE_UNIT();

void g_common_init(void)
{
    // Initialize FileX (ONLY for Mass Storage Class)
    fx_system_initialize ();

    // Initialize USBX Memory.
    ux_system_initialize ();

    // Initialize the USBX Host stack.
    ux_host_stack_initialize ();

    // Register the HUB Class if the class is used.
    ux_host_stack_class_register();

    // Register the Host CDC-ACM Class if the class is used.
    ux_host_stack_class_register();

    // Register the Host HID Class if the class is used.
    ux_host_stack_class_register();
```

```
    // Register the Host HID Clents if the HID clients are used.
    ux_host_class_hid_clients_register ();

    // Register the Host Mass Storage Class if the class is used.
    ux_host_stack_class_register();

    // Register and initialize the USB Host Controller.
    ux_host_stack_hcd_register ();

    // Get the USBX Host Class Container for registered classes.
    ux_host_stack_class_get ();
}
```

4.3.10.6 Express Logic USBX Application Code Examples

NOTE: Code samples in this section are subject to change. No any warranty for contents and possible changes.

USBX Device Stack User Code Sample

This section describes how user application code to be implemented in a user application thread followed to the auto-generated code, which is described in

Express Logic USBX Auto-generated Code Procedures. User application code for each USBX device class looks different so explained briefly in following sections with code samples.

The user application code follows to auto-generated code needs following processing briefly.

- Get a CDC-ACM instance through USBX CDC-ACM instance_activate Function Callback.
- Use the CDC-ACM instance and perform CDC-ACM communication with USB host device.

The code mock-up sample below shows an example of basic implementation of the USB CDC-ACM application with the USBX Device Class CDC-ACM.

```
// Precondition: The USBX Device Class CDC-ACM component is setup as below.
//   Refer to USBX Configuration Notes for more information.
// - "USBX CDC-ACM instance_activate Function Callback":
//               ux_cdc_device0_instance_activate
// - "USBX CDC-ACM instance_deactivate Function Callback":
//               ux_cdc_device0_instance_deactivate
//

// A pointer to store CDC-ACM device instance.
static UX_SLAVE_CLASS_CDC_ACM   * g_cdc;

// The USBX Device CDC-ACM instance_activate callback function.
void ux_cdc_device0_instance_activate (VOID * cdc_instance)
{
```

```
    /* Save the CDC instance.  */
    g_cdc = (UX_SLAVE_CLASS_CDC_ACM *) cdc_instance;
    tx_semaphore_put(&g_cdc_activate_semaphore0);
}

// The USBX Device CDC-ACM instance_deactivate callback function.
void ux_cdc_device0_instance_deactivate (VOID * cdc_instance)
{
    SSP_PARAMETER_NOT_USED(cdc_instance);
    g_cdc = UX_NULL;
}

// User application thread entry function.
void usb_thread_entry(void)
{
    // Wait until CDC-ACM device instance activated.
    tx_semaphore_get (&g_cdc_activate_semaphore0, TX_WAIT_FOREVER);

    while (1)
    {
        // Use the CDC-ACM instance obtained in instance_activate callback
        // and Perform communications with CDC class.
        // _ux_device_class_cdc_acm_read to read data from a USB Host.
        // _ux_device_class_cdc_acm_write to write data to a USB Host.
    }
}
```

The user application code follows to auto-generated code needs following processing briefly.

- Get the USB device handle by referencing ux_system_slave_device in the USBX System Device Data context named _ux_system_slave structure.
- Polls ux_slave_device_state member in the USB device handle until it changes to UX_DEVICE_CONFIGURED.
- Get a device interface.
- Get a HID instance from the interface.
- Use the HID instance and set HID Events to a USB host.

The code mock-up sample below shows an example of basic implementation of the USB HID application with the USBX Device Class HID.

```
// Precondition: The USBX Device Class HID component is setup as below.
//   See USBX Configuration Notes for the more information.
// - "USBX HID User Callback Function": ux_hid_device_callback
//

// The USBX Device HID User Callback function.
UINT ux_hid_device_callback (UX_SLAVE_CLASS_HID * hid,
                             UX_SLAVE_CLASS_HID_EVENT * hid_event)
{
```

```
    // Do an action against the HID event from a USB host.
    ....

    return UX_SUCCESS;
}

// User application thread entry function.
void usb_thread_entry(void)
{
    UX_SLAVE_DEVICE          * device;
    UX_SLAVE_INTERFACE       * interface;
    UX_SLAVE_CLASS_HID       * hid;
    UX_SLAVE_CLASS_HID_EVENT hid_event;

    // Clear hid_event with zero.
    // ....

    // Get the pointer to the device.
    device = &_ux_system_slave->ux_system_slave_device;

    while (1)
    {
        // Pole device->ux_slave_device_state until it is changed to
        // UX_DEVICE_CONFIGURED.
        // ....

        // Get a device interface. This is a simple case.
        interface = device->ux_slave_device_first_interface;

        // Get a HID instance from the interface.
        hid = interface->ux_slave_interface_class_instance;

        // We can move forward once a device is configured.
        while (1)
        {

            // Define a HID event in hid_event.
            // ....

            // Set the keyboard event to a USB host.
            _ux_device_class_hid_event_set(hid, &hid_event);

            // Do an action whatever you need...
            ....
        }
    }
}
```

The user application code for Device Mass Storage follows to auto-generated code does not need special application code in the user application thread. Everything should be handled in the media access function callback.

The code mock-up sample below shows an example of basic implementation of the USB RAM disk storage application with the USBX Device Class Mass Storage.

```
// Precondition: The USBX Device Mass Storage component is setup as below.
//   See USBX Configuration Notes for more information.
// - User Setup Callback              : Auto
// - Last LBA of Storage Media        : 31
// - Bytes Per Sector of Storage Media: 512
// - Type of Storage Media            : 0x80
// - Media Read Function Callback     : ux_device_msc_media_read
// - Media Write Function Callback    : ux_device_msc_media_write
// - Media Status Function Callback   : ux_device_msc_media_status
//   This setup matches to RAM Disk image "ramdisk" provided in
//   Application Note R30AN0242EUXXX available at https://www.renesas.com/

// RAM disk image.
extern UCHAR    ramdisk[];

// The USBX Device Mass Storage Media Status Function Callback.
UINT    ux_device_msc_media_status (VOID  * storage,
                                    ULONG lun,
                                    ULONG media_id,
                                    ULONG * media_status)
{
    // Do work
    // ....
    return (UX_SUCCESS);
}

// The USBX Device Mass Storage Media Read Function Callback.
UINT    ux_device_msc_media_read (VOID  * storage,
                                  ULONG lun,
                                  UCHAR * data_pointer,
                                  ULONG number_blocks,
                                  ULONG lba,
                                  ULONG * media_status)
{
    // Do work. Following is an example for 512B sector.
    memcpy(data_pointer, ramdisk + (512 * lba), 512 * number_blocks);
    return (UX_SUCCESS);
}

// The USBX Device Mass Storage Media Write Function Callback.
UINT    ux_device_msc_media_write (VOID  * storage,
                                   ULONG lun,
                                   UCHAR * data_pointer,
                                   ULONG number_blocks,
```

```
                                    ULONG lba,
                                    ULONG * media_status)
{
    // Do work. Following is an example for 512B sector.
    memcpy(ramdisk + (512 * lba), data_pointer, 512 * number_blocks);
    return (UX_SUCCESS);
}

// USB Thread entry function
void usb_thread_entry(void)
{
    // Do work if required.
}
```

USBX Host Stack User Code Sample

This section describes how user application code to be implemented in a user application thread followed to the auto-generated code, which is described in

Express Logic USBX Auto-generated Code Procedures. User application code for each USBX host class looks different so explained briefly in following sections with code samples.

The user application code follows to auto-generated code needs following processing briefly.

- Specify **User Callback Host Event Notification (Only valid for USB Host)** property of **USBX on ux** component to arbitrary function name ("NULL" by default).
- Use a CDC-ACM Class instance (UX_HOST_CLASS_CDC_ACM type) obtained in the User Callback Host Event Notification function in user application thread.

The code mock-up below briefly explains procedures required for USB communications with a USB CDC-ACM device.

```
static UX_HOST_CLASS_CDC_ACM * p_cdc_acm_class_instance = NULL;

UINT ux_host_usr_event_notification (ULONG event, UX_HOST_CLASS * host_class,
                                     VOID * instance)
{
  // Check if host_class is for CDC-ACM class.
  if (UX_SUCCESS ==
      _ux_utility_memory_compare (
          _ux_system_host_class_cdc_acm_name,
          host_class,
          _ux_utility_string_length_get(_ux_system_host_class_cdc_acm_name))
  ){
    // Check the event.
    if (UX_DEVICE_INSERTION == event)
    {
      p_cdc_acm_class_instance = (UX_HOST_CLASS_CDC_ACM *) instance;

      if (UX_HOST_CLASS_CDC_DATA_CLASS ==
          p_cdc_acm_class_instance->ux_host_class_cdc_acm_interface
                          ->ux_interface_descriptor.bInterfaceClass)
```

```
      {
        // Return successful completion.
        status = UX_SUCCESS;
      }
      else
      {
        // Check if the DATA class is on the second interface.
        p_cdc_acm_class_instance = p_cdc_acm_class_instance->
                                   ux_host_class_cdc_acm_next_instance;
        if (UX_NULL == p_cdc_acm_class_instance)
        {
          // We did not find a proper data interface.
          p_cdc_acm_class_instance = NULL;
          status = UX_HOST_CLASS_INSTANCE_UNKNOWN;
        }
        else
        {
          // Check this interface.
          if (UX_HOST_CLASS_CDC_DATA_CLASS ==
            p_cdc_acm_class_instance->ux_host_class_cdc_acm_interface
              ->ux_interface_descriptor.bInterfaceClass)
          {
            // Return successful completion.
            status = UX_SUCCESS;
          }
          else
          {
            // We did not find a proper data interface.
            p_cdc_acm_class_instance = NULL;
            status = UX_HOST_CLASS_INSTANCE_UNKNOWN;
          }
        }
      }
    }
  }
  else if (UX_DEVICE_REMOVAL == event)
  {
    p_cdc_acm_class_instance = NULL;
  }
  return status;
}
```

In user application thread entry function, use *p_cdc_acm_class_instance* obtained in the **User Callback Host Event Notification** required for USBX Host Class CDC_ACM APIs.

The user application code follows to auto-generated code needs following processing briefly.

- Specify **User Callback Host Event Notification (Only valid for USB Host)** property of **USBX on ux** component to arbitrary function name ("NULL" by default).

- Use a HID Class instance (UX_HOST_CLASS_HID type) obtained in the User Callback Host Event Notification function in user application thread.
- Get a HID Client instance (UX_HOST_CLASS_HID_CLIENT type) from the HID Class instance in user application thread.
- Get an instance of the HID Keyboard (UX_HOST_CLASS_HID_KEYBOARD type), Mouse (UX_HOST_CLASS_HID_MOUSE type), or Remote Control(UX_HOST_CLASS_HID_REMOTE_CONTROL type) and use it to get HID report in user application thread.

The code mock-up below briefly explains procedures required for USB communications with a USB device.

```
static UX_HOST_CLASS_HID * p_hid_class_instance;

UINT ux_host_usr_event_notification (ULONG event, UX_HOST_CLASS * host_class,
                                     VOID * instance)
{
    // Check if host_class is for HID class.
    if (UX_SUCCESS ==
        _ux_utility_memory_compare (
          _ux_system_host_class_hid_name,
          host_class,
          _ux_utility_string_length_get(_ux_system_host_class_hid_name))
    ){
        // Check the event.
        if (UX_DEVICE_INSERTION == event)
        {
            // Get a USB Host HID Class instance.
            p_hid_class_instance = (UX_HOST_CLASS_HID *) instance;

            // Notify the insertion of a USB HID device.
            tx_event_flags_set (&g_storage_event_flags, …);
        }
        else if (UX_DEVICE_REMOVAL == event)
        {
            p_hid_class_instance = NULL;

            // Notify the removal of a USB HID device.
            tx_event_flags_set (&g_storage_event_flags, …);
        }
    }
    return UX_SUCCESS;
}
```

In user application thread entry function, use *p_hid_class_instance* obtained in the **User Callback Host Event Notification** to get a USBX HID Client Instance.

- Use-case scenario 1:
  - Single USB Mass Storage media is already inserted/connected to the target board before the system boot up.
  - Single USB Mass Storage media is to be inserted before timeout happened.

– NetX/NetX Duo HTTP Server, FTP or TFTP, which require a pointer to the FileX Media Control block is used together.

In this use-case scenario, users can utilize the Auto Media Initialization feature to mount a USB Mass Storage media automatically in the auto-generated code. To enable the feature, set the following configuration properties of **FileX on USB Mass Storage** component.

- Set **Auto Media Initialization** property to "Enable" ("Disable" by default).
- Specify **Name** property ("fx_media0" by default).
- Specify **Name of FileX Media Control block initialization** ("fx_media_init_function0" by default).
- Specify **Timeout ticks for Media Initialization** ("1000" by default). (10 second if ThreadX timer is 10ms per count).
- Specify **User Callback Host Event Notification (Only valid for USB Host)** property of **USBX on ux** component to arbitrary function name ("NULL" by default).

In the Auto Media Initialization mode, user thread will be in suspension in the **FileX Media Control block initialization** function i.e. *fx_media_init_function0* presented in the auto-generated code, then the thread will be resumed when a pointer to the FileX Media Control block i.e. *fx_media0* is obtained. Users can utilize it to access to a USB Mass Storage media through FileX APIs in their application code.

After the USB Mass Storage media is removed, the FileX Media Control block is no longer valid so users need to obtain it again after another device insertion happened. Since **FileX Media Control block initialization** function i.e. *fx_media_init_function0* will not be called automatically at the device insertion second time or later, user application need to call the function to obtain a pointer to the FileX Media Control block by themselves. The function defined in the auto-generated code can be utilized in user application code as a user helper function.

To synchronize user application thread to the USB Mass Storage media insertion or removal timing, use **User Callback Host Event Notification**. Users need to define the callback function in their application code. Here is the mock-up sample of the function.

```
UINT ux_host_usr_event_notification (ULONG event, UX_HOST_CLASS * host_class,
                                     VOID * instance)
{
    // Check if host_class is for Mass Storage class.
    if (UX_SUCCESS ==
        _ux_utility_memory_compare (
          _ux_system_host_class_storage_name,
          host_class,
          _ux_utility_string_length_get(_ux_system_host_class_storage_name))
    ){
        // Check the event.
        if (UX_DEVICE_INSERTION == event)
        {
            // Notify the insertion of a USB Mass Storage media.
            tx_event_flags_set (&g_storage_event_flags, …);
        }
        else if (UX_DEVICE_REMOVAL == event)
        {
            // Notify the removal of a USB Mass Storage media.
            tx_event_flags_set (&g_storage_event_flags, …);
        }
    }
```

```
    return UX_SUCCESS;
}
```

NOTE: Auto Media Initialization feature for the USB Mass Storage media supports simple use-case to support single USB Mass Storage media but would not support multiple Mass Storage medias connected behind a USB HUB.

- Use-case scenario 2:
  - User thread cannot be suspended in the auto-generated code.
  - More complexed use-case than use-case scenario1 to use more than one USB flash drive.

In this use-case scenario, users need to disable Auto Media Initialization feature for the USB Mass Storage media and mount it in their application code on the fly when a device is inserted. Set the following configuration properties to support this use-case scenario.

1) Set **Auto Media Initialization** property of **FileX on USB Mass Storage** component to "Disable" ("Disable" by default).
2) Set **User Callback Host Event Notification (Only valid for USB Host)** property of **USBX on ux** component to arbitrary function name ("NULL" by default).

A FileX Media Control block is available to get in the **User Callback Host Event Notification** function. The code example below is the mock-up sample code.

```
UINT ux_host_usr_event_notification (ULONG event, UX_HOST_CLASS * host_class,
                                     VOID * instance)
{
    UX_HOST_CLASS_STORAGE_MEDIA * p_ux_host_class_storage_media;
    FX_MEDIA                    * p_fx_media;

    // Check if host_class is for Mass Storage class.
    if (UX_SUCCESS ==
        _ux_utility_memory_compare (
          _ux_system_host_class_storage_name,
          host_class,
          _ux_utility_string_length_get(_ux_system_host_class_storage_name))
    ){
        // Check if the event.
        if (UX_DEVICE_INSERTION == event)
        {
            // Get a pointer to the FX_MEDIA for a USB Mass Storage media.
            // p_fx_media can be used for the media access.
            ux_system_host_storage_fx_media_get (
                instance, &p_ux_host_class_storage_media, &p_fx_media);

            // You may want to get the drive volume serial ID to identify
            // the drive. Here is a hint to do that. Note that 'volume' below
            // is name of buffer to store the volume name of drive.
            // fx_media_volume_get (p_fx_media, volume, FX_BOOT_SECTOR);
```

```
            // Check p_fx_media->fx_media_memory_buffer byte 67-70.
            // The 4bytes show the volume serial ID of the drive
            // in case of FAT32 formatted drive.

            // Save the pointer to FX_MEDIA to use it later in user
            // application. The pointer to UX_HOST_CLASS_STORAGE_MEDIA
            // can be saved if required.

            // Notify the insertion of a USB Mass Storage media.
            tx_event_flags_set (&g_storage_event_flags, …);
        }
        else if (UX_DEVICE_REMOVAL == event)
        {
            // Notify the removal of a USB Mass Storage media.
            tx_event_flags_set (&g_storage_event_flags, …);
        }
    }
    return UX_SUCCESS;
}
```

NOTE: The function named *ux_system_host_storage_fx_media_get* is a helper function defined in the auto-generated code to get a pointer to the FileX Media Control block.

Finally, in user application thread entry function, suspend the thread until the UX_DEVICE_INSERTION event is notified by the **User Callback Host Event Notification** function. When the event happened, user thread will be resumed and be able to access to a USB Mass Storage media using a pointer to the FileX Media Control block obtained in the **User Callback Host Event Notification** function.

The steps highlight key points for the USBX UVC configuration and how to write your application thread to control a UVC device

- Add "USBX Host Class Video" component in the Synergy Configurator.
- Select USBX Port HCD on sf_el_ux for USBHS, and configure "FIFO size for Bulk/Isochronous Pipes" property to "1024 bytes" (preferred) or higher value than that.
- Set the "Number of Isochronous Pipes Reserved" to 0, 1, or 2. Note that, PIPE1-PIPE5 in the Synergy USB controllers can be used for the Bulk transfer; PIPE1 and PIPE2 can be used for the Isochronous transfer, that means, PIPE1 and PIPE2 are shared for the Bulk and Isochronous transfers.
  - If "0" is specified, PIPE1 and PIPE2 are shared for the Bulk or Isochronous. If both PIPEs are used for Bulk, no any PIPE allocation would be made for the Isochronous transfer.
  - If "1" is specified, PIPE1 will be secured for the Isochronous transfer. Bulk PIPEs can be assigned to PIPE2-PIPE5. PIPE2 can be assigned for either of Isochronous or Bulk.
  - If "2" is specified, PIPE1 and PIPE2 will be secured for Isochronous. Bulk PIPEs can be assigned to PIPE3-PIPE5.
- Add application code in your application space. Briefly, you will need to implement:
  - USBX Host event notification callback function to get a UVC instance (The callback needs to be specified to "User Callback for Host Event Notification" property of "USBX on ux" component).

  - USBX UVC transfer request done callback. Typically, a semaphore should be put in the function to synchronize your application thread to the timing of video stream data reception.
  - USBX UVC API calls in your thread entry function
- Add application code in your thread entry function to control your UVC device. Briefly, what you will need are:
  - Wait until the USBX Host event notification callback function is called by USBX and you get an instance of UVC instance. When non-NULL value is obtained for your UVC instance, you can proceed following steps in your application thread.
  - Call ux_host_class_video_transfer_callback_set() to register your USBX UVC transfer request done callback function.
  - Call ux_host_class_video_frame_parameters_set() to set video parameters.
  - Call ux_host_class_video_start() to start the UVC device control. You may need to call ux_host_-class_video_ioctl() instead with UX_HOST_CLASS_VIDEO_IOCTL_CHANNEL_START if your UVC device has special interface configuration in the device descriptor (See section 4. Limitations for more detail).
  - Call ux_host_class_video_transfer_buffer_add() with specifying a user buffer to store video stream data. You will receive video stream data from your UVC device by this API call.
  - Suspend your application thread with a semaphore. The semaphore is to be set in the USBX UVC transfer request done callback function.
  - The USBX UVC transfer request done callback function is called when video stream data is received from a UVC device. Perform any required processing in the function and put the semaphore your application thread is suspending.
  - Your application thread is resumed. Process video stream data to retrieve actual portion of video stream. Video stream data you received contains UVC payload header + payload data. For more detail of the UVC sample isochronous transfer, refer to Video Class specification available from USB.org http://www.usb.org/developers/docs/devclass_docs/
  - Continue the procedure #5 to receive video stream data continuously. When stopping the video stream. Call ux_host_class_video_stop()

The code mock-up below briefly explains procedures required for USB communications with a USB CDC-ACM device.

```
static UX_HOST_CLASS_HID * video_host_class;
/* USBX Host event notification callback function */
UINT ux_host_usr_event_notification(ULONG event, UX_HOST_CLASS * host_class, VOI
D * instance)
{
    if (UX_SUCCESS
            == _ux_utility_memory_compare ("ux_host_class_video",
                                           _host_class,
                                           _ux_utility_string_length_get ((UCHAR
 *)"ux_host_class_video")))
    {
    /* Check if there is a device insertion. */
        if (UX_DEVICE_INSERTION == event)
        {
                video_host_class = instance;
```

```
                /* Set the event flag to let application know the device inserti
on. */
                tx_event_flags_set (&g_device_insert_eventflag, …);
        }
        else if(UX_DEVICE_REMOVAL == event)
        {
                video_host_class = NULL;

                /* Clear the event flag in case the camera was removed before th
e application could clear it. */
                tx_event_flags_set (&g_device_insert_eventflag, …);
        }
    }
    return UX_SUCCESS;
}
```

4.3.10.7 Express Logic USBX Special Linker Sections

The USBX Device Stack configuration uses the following special memory sections in the linker script files. The order of memory sections in the linker script needs to consist of the USB Device Descriptor byte stream, which is given to _ux_device_stack_initialize() function; because of this the linker script definitions must not be modified.

Memory section for the USBX Device Descriptor.

Memory section	USB Descriptor to be defined in the section
.usb_device_desc_fs*	The USB Device Descriptor for FS mode
.usb_config_desc_fs*	The USB Configuration Descriptor for FS mode
.usb_interface_desc_fs*	The USB Interface Descriptor for FS mode
.usb_device_desc_hs*	The USB Device Descriptor for HS mode
.usb_config_desc_hs*	The USB Configuration Descriptor for HS mode
.usb_interface_desc_hs*	The USB Interface Descriptor for HS mode

NOTE: Memory sections for HS mode only exists for Synergy Parts, which has the USBHS controller.

4.3.10.8 Express Logic USBX Memory Requirements

The USBX Device stack or USBX Host stack consumes RAM for the control block. The Synergy Configuration tool allocates memory to the USBX memory pool statically in the auto-generate code. The memory consumption is different for each class as shown in the table below. You need to set the appropriate memory size in bytes to the **USBX Pool Memory Size** property of the USBX on ux component in the Synergy Configuration tool in section "USBX on ux Configuration." If multiple classes are used, set total memory size to the property.

Memory (RAM) requirements for USBX memory pool.

USBX class	S1 parts	The other parts
USBX Device Mass Storage (ux_device_class_storage)	5.8KB	19KB
USBX Device CDC-ACM (ux_device_class_cdc_acm)	5.8KB	18KB
USBX Device HID (ux_device_class_hid)	5.8KB	12KB
USBX Host Mass Storage (ux_host_class_storage)	N/A	- Pre-built library: 42KB Source: 39KB + UX_HOST_CLASS_STORAGE_MEMORY_BUFFER_SIZE + UX_HOST_CLASS_STORAGE_THREAD_STACK_SIZE
USBX Host CDC-ACM (ux_host_class_cdc_acm)	N/A	29KB
USBX Host HID (ux_host_class_hid)	N/A	- HID Mouse: 38KB HID Keyboard: 46KB
USBX Host Video (ux_host_class_video)	N/A	- 35KB

NOTE: The information shown in the table above is valid if compiled with default USBX configurations.

NOTE: The memory size of USBX Classes in the table above are of the Pre-built libraries and following configuration was applied for the builds:

- UX_THREAD_STACK_SIZE: 512(bytes) for S1 parts; 2048(bytes) for the other parts

NOTE: The memory size of USBX Host Mass Storage is also shown with the case built with the Source module. It is because the size depends on the configuration UX_HOST_CLASS_STORAGE_MEMORY_BUFFER_SIZE, and would differ much from the Pre-build library's. Note that the size is with following configuration applied.

- UX_THREAD_STACK_SIZE: 2048(bytes)

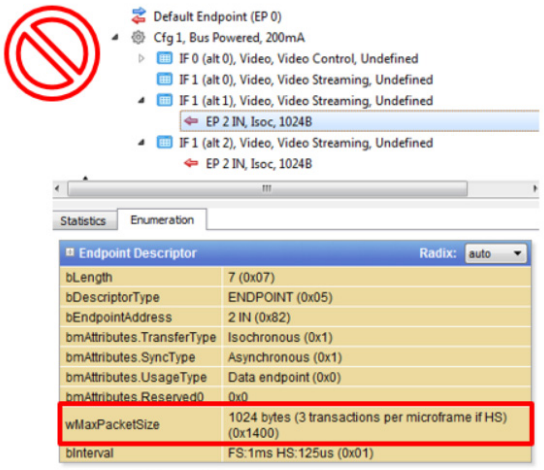
NOTE: The memory size of USBX Host HID shown in the table above is with following configurations (default setting) being made. The size can be reduced if Source module is used.

- UX_HOST_CLASS_HID_DECOMPRESSION_BUFFER: 4096(bytes).
- UX_HOST_CLASS_HID_USAGE: 1024(WORDs).

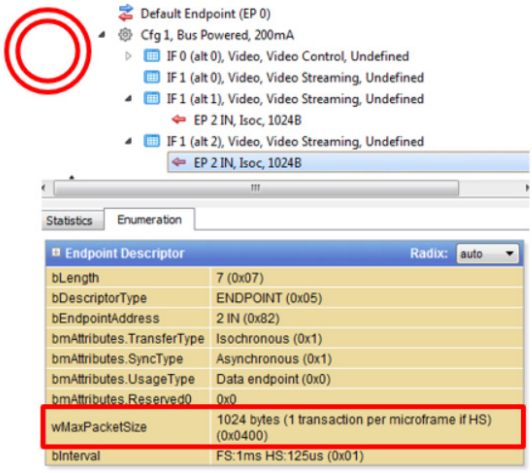
NOTE: The USBX Host Video memory pools size may differ for type of video device attached to the board

4.3.10.9 Express Logic USBX Limitations

- See the SSP Release Notes for known limitations.
- Support for the USB Device vender-specific class is not available.
- The module needs the interrupt of a USB Controller enabled. See section "Logic USBX Synergy Port Framework Limitations" for more information.
- USBX classes (ux_device_class_cdc_ecm, ux_device_class_rndis, ux_host_class_audio, ux_host_class_gser, ux_host_class_printer, ux_host_class_prolific, ux_host_class_swar) and USBX network driver (ux_network_driver) are experimental modules and not yet tested for Synergy parts in this version of SSP. It is not recommended to use them for product developments.
- To specify an Alt. setting in an interface in UVC(USB video class) device, you may need to use the USBX API ux_host_class_video_ioctl() instead of ux_host_class_video_start(), specifying UX_HOST_CLASS_VIDEO_IOCTL_CHANNEL_START for the argument "ioctl_function". You also need to specify the argument parameter with setting the member .ux_host_class_video_parameter_channel_bandwidth_selection to match to the wMaxPacketSize of an isochronous endpoint. For instance, if .ux_host_class_video_parameter_channel_bandwidth_selection is set to 1024 for a UVC device with the device descriptor shown in the figure below, the IF1 Alt.1 will not be selected since it requires high-bandwidth transfer. Instead, IF1 Alt.2 will be selected.



(this will not be selected)



(this will be selected)

Chapter 5 API Reference: Common

You can find error codes and version data structures that are common to the entire Synergy Software Package here.

5.1 Common Error Codes

All SSP code at every layer shares these common error codes.

5.1.1 User-defined enumerations

- ssp_err_t
- ssp_command_t

5.1.2 User-defined macros

- #define SSP_PARAMETER_NOT_USED

  Initial value: `(void) ((p))`

  This macro is used to suppress compiler messages about a parameter not being used in a function. The nice thing about using this implementation is that it does not take any extra RAM or ROM.
- #define SSP_CPP_HEADER

  Initial value:

  Determine if a C++ compiler is being used. If so, ensure that standard C is used to process the API information.
- #define SSP_CPP_FOOTER

  Initial value:
- #define SSP_HEADER

  Initial value: `#define SSP_CPP_HEADER`

  SSP Header and Footer definitions
- #define SSP_FOOTER

  Initial value: `SSP_CPP_FOOTER`

5.1.3 API Data

5.1.3.1 ssp_err_t

`ssp_err_t`

Detailed description

Common error codes

Enumerated values

Name	Description
SSP_SUCCESS	
SSP_ERR_ASSERTION	A critical assertion has failed.
SSP_ERR_INVALID_POINTER	Pointer points to invalid memory location.
SSP_ERR_INVALID_ARGUMENT	Invalid input parameter.
SSP_ERR_INVALID_CHANNEL	Selected channel does not exist.
SSP_ERR_INVALID_MODE	Unsupported or incorrect mode.
SSP_ERR_UNSUPPORTED	Selected mode not supported by this API.
SSP_ERR_NOT_OPEN	Requested channel is not configured or API not open.
SSP_ERR_IN_USE	Channel/peripheral is running/busy.
SSP_ERR_OUT_OF_MEMORY	Allocate more memory in the driver's cfg.h.
SSP_ERR_HW_LOCKED	Hardware is locked.
SSP_ERR_IRQ_BSP_DISABLED	IRQ not enabled in BSP.
SSP_ERR_OVERFLOW	Hardware overflow.
SSP_ERR_UNDERFLOW	Hardware underflow.
SSP_ERR_ALREADY_OPEN	Requested channel is already open in a different configuration.
SSP_ERR_APPROXIMATION	Could not set value to exact result.
SSP_ERR_CLAMPED	Value had to be limited for some reason.
SSP_ERR_INVALID_RATE	Selected rate could not be met.
SSP_ERR_ABORTED	An operation was aborted.
SSP_ERR_NOT_ENABLED	Requested operation is not enabled.
SSP_ERR_TIMEOUT	Timeout error.

Name	Description
SSP_ERR_INVALID_BLOCKS	Invalid number of blocks supplied.
SSP_ERR_INVALID_ADDRESS	Invalid address supplied.
SSP_ERR_INVALID_SIZE	Invalid size/length supplied for operation.
SSP_ERR_WRITE_FAILED	Write operation failed.
SSP_ERR_ERASE_FAILED	Erase operation failed.
SSP_ERR_INVALID_CALL	Invalid function call is made.
SSP_ERR_INVALID_HW_CONDITION	Detected hardware is in invalid condition.
SSP_ERR_INVALID_FACTORY_FLASH	Factory flash is not available on this MCU.
SSP_ERR_INVALID_FMI_DATA	Linked FMI data table is not valid.
SSP_ERR_INVALID_STATE	API or command not valid in the current state.
SSP_ERR_INTERNAL	Internal error. Start of RTOS only error codes
SSP_ERR_WAIT_ABORTED	Wait.
SSP_ERR_FRAMING	Framing error occurs. Start of UART specific
SSP_ERR_BREAK_DETECT	Break signal detects.
SSP_ERR_PARITY	Parity error occurs.
SSP_ERR_RXBUF_OVERFLOW	Receive queue overflow.
SSP_ERR_QUEUE_UNAVAILABLE	Can't open s/w queue.
SSP_ERR_INSUFFICIENT_SPACE	Not enough space in transmission circular buffer.
SSP_ERR_INSUFFICIENT_DATA	Not enough data in receive circular buffer.
SSP_ERR_TRANSFER_ABORTED	The data transfer was aborted. Start of SPI specific
SSP_ERR_MODE_FAULT	Mode fault error.
SSP_ERR_READ_OVERFLOW	Read overflow.
SSP_ERR_SPI_PARITY	Parity error.
SSP_ERR_OVERRUN	Overrun error.

Name	Description
SSP_ERR_CLOCK_INACTIVE	Inactive clock specified as system clock. Start of CGC Specific
SSP_ERR_CLOCK_ACTIVE	Active clock source cannot be modified without stopping first.
SSP_ERR_STABILIZED	Clock has stabilized after its been turned on/off.
SSP_ERR_NOT_STABILIZED	Clock has not stabilized after its been turned on/off.
SSP_ERR_MAIN_OSC_INACTIVE	PLL initialization attempted when main osc is turned off.
SSP_ERR_OSC_STOP_DET_ENABLED	Illegal attempt to stop LOCO when Oscillation stop is enabled.
SSP_ERR_OSC_STOP_DETECTED	The Oscillation stop detection status flag is set.
SSP_ERR_OSC_STOP_CLOCK_ACTIVE	Attempt to clear Oscillation Stop Detect Status with PLL/MAIN_OSC active.
SSP_ERR_CLKOUT_EXCEEDED	Output on target output clock pin exceeds maximum supported limit.
SSP_ERR_USB_MODULE_ENABLED	USB clock configure request with USB Module enabled.
SSP_ERR_HARDWARE_TIMEOUT	A register read or write timed out.
SSP_ERR_PE_FAILURE	Unable to enter Programming mode. Start of FLASH Specific
SSP_ERR_CMD_LOCKED	Peripheral in command locked state.
SSP_ERR_FCLK	FCLK must be >= 4 MHz.
SSP_ERR_INVALID_CAC_REF_CLOCK	Measured clock rate < reference clock rate. Start of CAC Specific
SSP_ERR_CLOCK_GENERATION	Clock cannot be specified as system clock. Start of GLCD Specific
SSP_ERR_INVALID_TIMING_SETTING	Invalid timing parameter.
SSP_ERR_INVALID_LAYER_SETTING	Invalid layer parameter.
SSP_ERR_INVALID_ALIGNMENT	Invalid memory alignment found.
SSP_ERR_INVALID_GAMMA_SETTING	Invalid gamma correction parameter.

Name	Description
SSP_ERR_INVALID_LAYER_FORMAT	Invalid color format in layer.
SSP_ERR_INVALID_UPDATE_TIMING	Invalid timing for register update.
SSP_ERR_INVALID_CLUT_ACCESS	Invalid access to CLUT entry.
SSP_ERR_INVALID_FADE_SETTING	Invalid fade-in/fade-out setting.
SSP_ERR_JPEG_ERR	JPEG error. Start of JPEG Specific
SSP_ERR_JPEG_SOI_NOT_DETECTED	SOI not detected until EOI detected.
SSP_ERR_JPEG_SOF1_TO_SOFF_DETECTED	SOF1 to SOFF detected.
SSP_ERR_JPEG_UNSUPPORTED_PIXEL_FORMAT	Unprovided pixel format detected.
SSP_ERR_JPEG_SOF_ACCURACY_ERROR	SOF accuracy error: other than 8 detected.
SSP_ERR_JPEG_DQT_ACCURACY_ERROR	DQT accuracy error: other than 0 detected.
SSP_ERR_JPEG_COMPONENT_ERROR1	Component error1: the number of SOF0 header components detected is other than 1,3,or 4.
SSP_ERR_JPEG_COMPONENT_ERROR2	Component error2: the number of components differs between SOF0 header and SOS.
SSP_ERR_JPEG_SOF0_DQT_DHT_NOT_DETECTED	SOF0, DQT, and DHT not detected when SOS detected.
SSP_ERR_JPEG_SOS_NOT_DETECTED	SOS not detected: SOS not detected until EOI detected.
SSP_ERR_JPEG_EOI_NOT_DETECTED	EOI not detected (default)
SSP_ERR_JPEG_RESTART_INTERVAL_DATA_NUMBER_ERROR	Restart interval data number error detected.
SSP_ERR_JPEG_IMAGE_SIZE_ERROR	Image size error detected.
SSP_ERR_JPEG_LAST_MCU_DATA_NUMBER_ERROR	Last MCU data number error detected.
SSP_ERR_JPEG_BLOCK_DATA_NUMBER_ERROR	Block data number error detected.
SSP_ERR_JPEG_BUFFERSIZE_NOT_ENOUGH	User provided buffer size not enough.
SSP_ERR_JPEG_UNSUPPORTED_IMAGE_SIZE	JPEG Image size is not aligned with MCU.
SSP_ERR_CALIBRATE_FAILED	Calibration failed. Start of touch panel framework specific

Name	Description
SSP_ERR_IP_HARDWARE_NOT_PRESENT	Requested IP does not exist on this device. Start of FMI specific
SSP_ERR_IP_UNIT_NOT_PRESENT	Requested unit does not exist on this device.
SSP_ERR_IP_CHANNEL_NOT_PRESENT	Requested channel does not exist on this device.
SSP_ERR_NO_MORE_BUFFER	No more buffer found in the memory block pool. Start of Message framework specific
SSP_ERR_ILLEGAL_BUFFER_ADDRESS	Buffer address is out of block memory pool.
SSP_ERR_INVALID_WORKBUFFER_SIZE	Work buffer size is invalid.
SSP_ERR_INVALID_MSG_BUFFER_SIZE	Message buffer size is invalid.
SSP_ERR_TOO_MANY_BUFFERS	Number of buffer is too many.
SSP_ERR_NO_SUBSCRIBER_FOUND	No message subscriber found.
SSP_ERR_MESSAGE_QUEUE_EMPTY	No message found in the message queue.
SSP_ERR_MESSAGE_QUEUE_FULL	No room for new message in the message queue.
SSP_ERR_ILLEGAL_SUBSCRIBER_LISTS	Message subscriber lists is illegal.
SSP_ERR_BUFFER_RELEASED	Buffer has been released.
SSP_ERR_D2D_ERROR_INIT	Dave/2d has an error in the initialization. Start of 2DG Driver specific
SSP_ERR_D2D_ERROR_DEINIT	Dave/2d has an error in the initialization.
SSP_ERR_D2D_ERROR_RENDERING	Dave/2d has an error in the rendering.
SSP_ERR_D2D_ERROR_SIZE	Dave/2d has an error in the rendering.
SSP_ERR_QUEUE_FULL	Queue is full, cannot queue another data. Start of BYTEQ library specific
SSP_ERR_QUEUE_EMPTY	Queue is empty, no data to dequeue.
SSP_ERR_CTSU_SC_OVERFLOW	Sensor count overflowed when performing CTSU scan. User must clear the CTSUSCOVF bit manually.
SSP_ERR_CTSU_RC_OVERFLOW	Reference count overflowed when performing CTSU scan. User must clear the CTSURCOVF bit manually.

Name	Description
SSP_ERR_CTSU_ICOMP	Abnormal TSCAP voltage. User must clear the CTSUICOMP bit manually.
SSP_ERR_CTSU_OFFSET_ADJUSTMENT_FAILED	Auto tuning algorithm failed.
SSP_ERR_CTSU_SAFETY_CHECK_FAILED	Safety check failed
SSP_ERR_CARD_INIT_FAILED	SD card or eMMC device failed to initialize. Start of SDMMC specific
SSP_ERR_CARD_NOT_INSERTED	SD card not installed.
SSP_ERR_SDHI_FAILED	SD peripheral failed to respond properly.
SSP_ERR_READ_FAILED	Data read failed.
SSP_ERR_CARD_NOT_READY	SD card was removed.
SSP_ERR_CARD_WRITE_PROTECTED	Media is write protected.
SSP_ERR_TRANSFER_BUSY	Transfer in progress.
SSP_ERR_MEDIA_FORMAT_FAILED	Media format failed. Start of FX_IO specific
SSP_ERR_MEDIA_OPEN_FAILED	Media open failed.
SSP_ERR_CAN_DATA_UNAVAILABLE	No data available. Start of CAN specific
SSP_ERR_CAN_MODE_SWITCH_FAILED	Switching operation modes failed.
SSP_ERR_CAN_INIT_FAILED	Hardware initialization failed.
SSP_ERR_CAN_TRANSMIT_NOT_READY	Transmit in progress.
SSP_ERR_CAN_RECEIVE_MAILBOX	Mailbox is setup as a receive mailbox.
SSP_ERR_CAN_TRANSMIT_MAILBOX	Mailbox is setup as a transmit mailbox.
SSP_ERR_CAN_MESSAGE_LOST	Receive message has been overwritten or overrun.
SSP_ERR_WIFI_CONFIG_FAILED	WiFi module Configuration failed. Start of SF_WIFI Specific
SSP_ERR_WIFI_INIT_FAILED	WiFi module initialization failed.
SSP_ERR_WIFI_TRANSMIT_FAILED	Transmission failed.
SSP_ERR_WIFI_INVALID_MODE	API called when provisioned in client mode.

Name	Description
SSP_ERR_WIFI_FAILED	WiFi Failed.
SSP_ERR_CELLULAR_CONFIG_FAILED	Cellular module Configuration failed. Start of SF_CELLULAR Specific
SSP_ERR_CELLULAR_INIT_FAILED	Cellular module initialization failed.
SSP_ERR_CELLULAR_TRANSMIT_FAILED	Transmission failed.
SSP_ERR_CELLULAR_FW_UPTODATE	Firmware is uptodate.
SSP_ERR_CELLULAR_FW_UPGRADE_FAILED	Firmware upgrade failed.
SSP_ERR_CELLULAR_FAILED	Cellular Failed.
SSP_ERR_CELLULAR_INVALID_STATE	API Called in invalid state.
SSP_ERR_BLE_FAILED	BLE operation failed. Start of SF_BLE specific
SSP_ERR_BLE_INIT_FAILED	BLE device initialization failed.
SSP_ERR_BLE_CONFIG_FAILED	BLE device configuration failed.
SSP_ERR_BLE_PRF_ALREADY_ENABLED	BLE device Profile already enabled.
SSP_ERR_BLE_PRF_NOT_ENABLED	BLE device not enabled.
SSP_ERR_CRYPTO_CONTINUE	Continue executing function. Start of Crypto specific (0x10000) Refer to sf_cryoto_err.h for Crypto error code.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Hardware resource busy */.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty */.
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Invalid index */.
SSP_ERR_CRYPTO_SCE_RETRY	Retry.
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Verify is failed.
SSP_ERR_CRYPTO_SCE_ALREADY_OPEN	Crypto Module is already opened.
SSP_ERR_CRYPTO_NOT_OPEN	Hardware module is not initialized.
SSP_ERR_CRYPTO_UNKNOWN	Some unknown error occurred */.
SSP_ERR_CRYPTO_NULL_POINTER	Null pointer input as a parameter */.

Name	Description
SSP_ERR_CRYPTO_NOT_IMPLEMENTED	Algorithm/size not implemented */.
SSP_ERR_CRYPTO_RNG_INVALID_PARAM	An invalid parameter is specified */.
SSP_ERR_CRYPTO_RNG_FATAL_ERROR	A fatal error occurred */.
SSP_ERR_CRYPTO_INVALID_SIZE	Size specified is invalid */.
SSP_ERR_CRYPTO_INVALID_STATE	Function used in an valid state */.
SSP_ERR_CRYPTO_ALREADY_OPEN	control block is already opened */
SSP_ERR_CRYPTO_INSTALL_KEY_FAILED	Specified input key is invalid. */.
SSP_ERR_CRYPTO_AUTHENTICATION_FAILED	Authentication failed */.
SSP_ERR_CRYPTO_COMMON_NOT_OPENED	Crypto Framework Common is not opened. Start of SF_CRYPTO specific
SSP_ERR_CRYPTO_HAL_ERROR	Cryoto HAL module returned an error.
SSP_ERR_CRYPTO_KEY_BUF_NOT_ENOUGH	Key buffer size is not enough to generate a key.
SSP_ERR_CRYPTO_BUF_OVERFLOW	Attempt to write data larger than what the buffer can hold.
SSP_ERR_CRYPTO_INVALID_OPERATION_MODE	Invalid operation mode.
SSP_ERR_MESSAGE_TOO_LONG	Message for RSA encryption is too long.
SSP_ERR_RSA_DECRYPTION_ERROR	RSA Decryption error.

5.1.3.2 ssp_command_t

`ssp_command_t`

Detailed description

ioctl commands.

Enumerated values

Name	Description
SSP_COMMAND_GET_SECTOR_COUNT	Get media sector count.
SSP_COMMAND_GET_SECTOR_SIZE	Get sector size.

Name	Description
SSP_COMMAND_GET_BLOCK_SIZE	Get erase block size.
SSP_COMMAND_CTRL_ERASE_SECTOR	Erase sectors.
SSP_COMMAND_GET_WRITE_PROTECTED	Get Write Protection status.
SSP_COMMAND_SET_BLOCK_SIZE	Set block size.

5.2 Version control

5.2.1 ssp_version_t

5.2.1.1 Detailed description

Common version structure

5.2.1.2 Variables

version_id
code_version_minor
code_version_major
api_version_minor
api_version_major

5.2.2 version_id

```
uint32_t ::version_id
```

5.2.2.1 Detailed description

Version id

5.2.3 code_version_minor

```
uint8_t ::code_version_minor
```

5.2.3.1 Brief description

Code minor version.

5.2.4 code_version_major

```
uint8_t ::code_version_major
```

5.2.4.1 Brief description

Code major version.

5.2.5 api_version_minor

```
uint8_t ::api_version_minor
```

5.2.5.1 Brief description

API minor version.

5.2.6 api_version_major

```
uint8_t ::api_version_major
```

5.2.6.1 Brief description

API major version.

5.3 Pack Version Control

5.3.1 ssp_pack_version_t

5.3.1.1 Detailed description

SSP Pack version structure

5.3.1.2 Variables

version_id

build

patch

minor

major

5.3.2 version_id

```
uint32_t ::version_id
```

5.3.2.1 Detailed description

Version id

5.3.3 build

```
uint8_t ::build
```

5.3.3.1 Brief description

Build version of SSP Pack.

5.3.4 patch

```
uint8_t ::patch
```

5.3.4.1 Brief description

Patch version of SSP Pack.

5.3.5 minor

```
uint8_t ::minor
```

5.3.5.1 Brief description

Minor version of SSP Pack.

5.3.6 major

```
uint8_t ::major
```

5.3.6.1 Brief description

Major version of SSP Pack.

Chapter 6 API Reference: Framework Interfaces

6.1 API Reference: Framework Interfaces

The Framework Interface provides common APIs for functional Framework Layer applications. The Framework Interfaces can be implemented by one or more Framework Layer drivers.

- ADC Periodic Framework Interface
- Audio Framework Interface
- Audio Playback Framework Interface
- Audio Recording Framework Interface
- SF BLE Framework Interface
- SF BLE On-Board Profile Framework Interface
- SF BLE Alert Notification Profile Framework Interface
- SF BLE Battery Service Profile Framework Interface
- SF BLE Blood Pressure Profile Framework Interface
- SF BLE Current Time Service Profile Framework Interface
- SF BLE Find Me Profile Framework Interface
- SF BLE HID Over GATT Profile Framework Interface
- SF BLE Heart Rate Profile Framework Interface
- SF BLE Health Thermometer Profile Framework Interface
- SF BLE Immediate Alert Profile Framework Interface
- SF BLE Next DST Change Service Profile Framework Interface
- SF BLE Phone Alert Status Profile Framework Interface
- SF BLE Proximity Profile Framework Interface
- SF BLE Reference Time Update Service Profile Framework Interface
- SF BLE Scan Parameters Service Profile Framework Interface
- SF BLE Time Information Profile Framework Interface
- Block Media Framework Interface
- SF CELLULAR Framework Interface
- SF CELLULAR NSAL Framework Interface
- SF Socket CELLULAR Framework Interface

- Communications Framework Interface
- Console Framework Interface
- SSP Crypto Framework Common Module Interface
- SSP Crypto Cipher Framework Interface
- SSP Crypto HASH Framework Interface
- SSP Crypto TRNG Framework Interface
- GUIX Interface
- External IRQ Framework Interface
- I2C Framework
- JPEG Decode Framework Interface
- Messaging Framework Interface
- Power Profiles Framework Interface
- Power Profiles Framework Interface
- SF Socket WIFI Framework Interface
- SPI Framework Interface
- Thread Monitor Framework Interface
- CTSU Framework Interface
- CTSU Button Framework Interface
- CTSU Slider Framework Interface
- Touch Panel Framework Interface
- SF WIFI Framework Interface
- SF WIFI NSAL Interface
- SF WIFI On-Chip Stack Interface
- SSP Crypto Key Installation Framework Interface
- SSP Crypto Signature Framework Interface
- SF WIFI NSAL on NetX
- SSP Crypto Key Framework Interface
- SF CELLULAR Framework Common Code
- TES D/AVE 2D Port Driver

6.2 ADC Periodic Framework Interface

RTOS-integrated ADC Periodic Framework Interface.

6.2.1 Summary

This is a ThreadX aware generic Periodic ADC sampling framework intended to be used to sample the ADC at periodic intervals, buffer the specified number of samples and then notify the application. The driver will use hardware triggers to allow for time-synchronous sampling. After initial configuration and the scan process is started, the framework uses a hardware timer to trigger an ADC scan in one-shot mode. Each scan can consist of one or more channels. When each scan is completed the ADC interrupt is intercepted by the DTC which moves the result of the scan into the user buffer. Each scan is defined as a sampling iteration and the number of samples generated for each scan will be equal to the number of channels if the channels are sequential eg: channels 1, 2, 3, 4. If the channels are not in sequence, eg: channels 1, 3, 4, 5, then the samples generated by each scan will also include data from the unused channels in between. Thus the second example here will result in 5 samples being stored to the user buffer each time even though only 4 channels have been configured for usage. The user specifies the total number of sample iterations that need to occur before being notified. When the specified number of sampling iterations have occurred and the data for each iteration has been stored into the user buffer, the user is notified via the callback function with an index for the valid data in the buffer and an event indicating that sampling for the specified number of iterations is complete. Unless the user stops the scan process using the stop API call, the scan will continue to be triggered by the timer and data will be written into the user buffer which is treated by the framework as a circular buffer. For this reason, the buffer length must be at least twice the total number of samples that will be generate after all the iterations are completed. In the second example, where there are 5 samples generated for each iteration, if the sample count is set to 3, this will result in 15 samples being available in the buffer before the callback is called. Thus in this example, the buffer length must be set to 30 or larger. The name and length of the buffer is specified via the framework configuration structure.

Implemented by: ADC Interface

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

ADC Periodic Framework Interface description: ADC Periodic Framework

6.2.2 Interface API

sf_adc_periodic_api_t

Function name	Description
.open	Acquires mutex, then initializes driver at the HAL layer
.start	Starts the scan.
.stop	Stops the hardware trigger (timer) from triggering any more ADC scans.
.close	Releases channel mutex and closes channel at HAL layer.
.versionGet	Gets version and stores it in provided pointer p_version.

6.2.3 Data structures

- sf_adc_periodic_callback_args_t
- sf_adc_periodic_cfg_t
- sf_adc_periodic_instance_t

6.2.4 Enumerations

- sf_adc_periodic_event_t

6.2.5 Typedefs

- sf_adc_periodic_ctrl_t

6.2.6 Defines

- #define SF_ADC_PERIODIC_API_VERSION_MAJOR

  Initial value: `(1U)`

  Version of the API defined in this file
- #define SF_ADC_PERIODIC_API_VERSION_MINOR

  Initial value: `(3U)`

6.2.7 API Data

6.2.7.1 sf_adc_periodic_event_t

```
sf_adc_periodic_event_t
```

Detailed description

Options for the callback events.

Enumerated values

Name	Description
SF_ADC_PERIODIC_EVENT_NEW_DATA	New data is available in the buffer.

6.2.7.2 sf_adc_periodic_ctrl_t

```
typedef void sf_adc_periodic_ctrl_t
```

Detailed description

ADC periodic framework control block. Allocate an instance specific control block to pass into the ADC periodic framework API calls. Implemented as

- sf_adc_periodic_instance_ctrl_t

6.2.8 API Structures

6.2.8.1 sf_adc_periodic_callback_args_t

sf_adc_periodic_callback_args_t

Detailed description

ADC callback arguments definitions

Variables

- sf_adc_periodic_event_t event

  Periodic ADC callback event.

- uint32_t buffer_index

  Index to the buffer where the new data is stored.

- void const * p_context

  Placeholder for user data.

- adc_data_size_t * p_data_buffer

  Pointer to the buffer that will store the samples.

- uint32_t num_new_samples

  Number of new samples in the data buffer.

6.2.8.2 sf_adc_periodic_cfg_t

sf_adc_periodic_cfg_t

Detailed description

Configuration for RTOS integrated ADC driver

Variables

- adc_instance_t const *const p_lower_lvl_adc

  Pointer to the ADC instance.

- timer_instance_t const *const p_lower_lvl_timer

  Pointer to the Timer instance.

- transfer_instance_t const *const p_lower_lvl_transfer

  Pointer to the Transfer instance.

- adc_data_size_t * p_data_buffer

  Pointer to the buffer that will store the samples.
- uint32_t data_buffer_length

  Length of the data buffer that will store the samples.
- uint32_t sample_count

  Samples per channel to be buffered before notifying the app.
- elc_event_t scan_trigger

  The hardware trigger that starts the ADC scan.
- void(* p_callback)(sf_adc_periodic_callback_args_t *p_args)

  Callback function.
- void const * p_context

  Placeholder for user data.
- void const * p_extend

  Extension parameter for hardware specific settings.

6.2.8.3 sf_adc_periodic_api_t

sf_adc_periodic_api_t

Detailed description

Framework Periodic ADC API structure. Implementations will use the following API.

6.2.8.4 open

```
ssp_err_t(* sf_adc_periodic_api_t::open) (sf_adc_periodic_ctrl_t *const p_ctrl,
sf_adc_periodic_cfg_t const *const p_cfg)
```

Detailed description

Acquires mutex, then initializes driver at the HAL layer

Table 1: Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a structure allocated by user. Elements initialized here.
p_cfg	in	Pointer to configuration structure. All elements of the structure must be set by user.

Parameter p_ctrl

Definition: sf_adc_periodic_ctrl_t*const p_ctrl

ADC periodic framework control block. Allocate an instance specific control block to pass into the ADC periodic framework API calls. Implemented assf_adc_periodic_instance_ctrl_t

Parameter p_cfg

Definition: sf_adc_periodic_cfg_t const *const p_cfg

Configuration for RTOS integrated ADC driver

- sf_adc_periodic_cfg_t::adc_instance_t

  Pointer to the ADC instance.
- sf_adc_periodic_cfg_t::timer_instance_t

  Pointer to the Timer instance.
- sf_adc_periodic_cfg_t::transfer_instance_t

  Pointer to the Transfer instance.
- sf_adc_periodic_cfg_t::p_data_buffer

  Pointer to the buffer that will store the samples.
- sf_adc_periodic_cfg_t::data_buffer_length

  Length of the data buffer that will store the samples.
- sf_adc_periodic_cfg_t::sample_count

  Samples per channel to be buffered before notifying the app.
- sf_adc_periodic_cfg_t::elc_event_t

  The hardware trigger that starts the ADC scan.
- sf_adc_periodic_cfg_t::p_callback

  Callback function.
- sf_adc_periodic_cfg_t::p_context

  Placeholder for user data.
- sf_adc_periodic_cfg_t::p_extend

  Extension parameter for hardware specific settings.

6.2.8.5 start

```
ssp_err_t(* sf_adc_periodic_api_t::start) (sf_adc_periodic_ctrl_t *const p_ctrl)
```

Detailed description

Starts the scan.

ATTENTION: The driver will enable the ADC to be triggered via timer event; there will be a time delay from the time this function is called to the time the hardware timer count expires and triggers the scan.

Table 2: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block set in SF_ADC_PERIODIC_Open.

Parameter p_ctrl

Definition: `sf_adc_periodic_ctrl_t*const p_ctrl`

ADC periodic framework control block. Allocate an instance specific control block to pass into the ADC periodic framework API calls. Implemented assf_adc_periodic_instance_ctrl_t

6.2.8.6 stop

```
ssp_err_t(* sf_adc_periodic_api_t::stop) (sf_adc_periodic_ctrl_t *const p_ctrl)
```

Detailed description

Stops the hardware trigger (timer) from triggering any more ADC scans.

Table 3: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block set in SF_ADC_PERIODIC_Open.

Parameter p_ctrl

Definition: `sf_adc_periodic_ctrl_t*const p_ctrl`

ADC periodic framework control block. Allocate an instance specific control block to pass into the ADC periodic framework API calls. Implemented assf_adc_periodic_instance_ctrl_t

6.2.8.7 close

```
ssp_err_t(* sf_adc_periodic_api_t::close) (sf_adc_periodic_ctrl_t *const p_ctrl)
```

Detailed description

Releases channel mutex and closes channel at HAL layer.

Table 4: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block set in SF_ADC_PERIODIC_Open.

Parameter p_ctrl

Definition: `sf_adc_periodic_ctrl_t*const p_ctrl`

ADC periodic framework control block. Allocate an instance specific control block to pass into the ADC periodic framework API calls. Implemented assf_adc_periodic_instance_ctrl_t

6.2.8.8 versionGet

```
ssp_err_t(* sf_adc_periodic_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Gets version and stores it in provided pointer p_version.

Table 5: Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

6.2.8.9 sf_adc_periodic_instance_t

sf_adc_periodic_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_adc_periodic_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_adc_periodic_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_adc_periodic_api_t const * p_api

  Pointer to the API structure for this instance.

6.3 Audio Framework Interface

RTOS-integrated Audio Framework Interface.

6.3.1 Summary

The Audio Interface is a ThreadX-aware Audio Framework Interface. The Interface is implemented by the Audio Framework using the Timer driver, the Transfer driver, and a choice of the following drivers for playback: DAC, PWM (to be implemented), or I2S (to be implemented).

Interfaces used:

- Transfer Interface
- DAC Interface
- Timer Interface

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Audio Framework Interface description: Audio Playback Framework

6.3.2 Interface API

sf_audio_playback_api_t

Function name	Description
.open	Configure the audio framework by creating a thread for audio playback and configuring HAL layer drivers used. This function must be called before any other audio functions.
.close	The close API handles cleans up internal driver data.
.start	Play audio. Currently only 16-bit mono PCM buffers are supported.
.pause	Pause audio playback. This stops the peripheral that triggers the DMA/DTC transfer and posts a flag to notify SF_AUDIO_PLAYBACK_Start() to pause any playback in progress.
.stop	Stop audio playback. Causes SF_AUDIO_PLAYBACK_Start() halt playback and return.
.resume	Resume audio playback. Posts a flag to notify SF_AUDIO_PLAYBACK_Start() to restart the peripheral that triggers the DMA/DTC transfer.

Function name	Description
.volumeSet	Set software volume control. Software volume control is applied globally to all streams on the hardware.
.versionGet	Store version information in provided pointer.

6.3.3 Data structures

- sf_audio_playback_data_t
- sf_audio_playback_common_cfg_t
- sf_audio_playback_cfg_t
- sf_audio_playback_instance_t

6.3.4 Enumerations

- sf_audio_playback_status_t

6.3.5 Typedefs

- sf_audio_playback_common_ctrl_t
- sf_audio_playback_ctrl_t

6.3.6 Defines

- #define SF_AUDIO_PLAYBACK_API_VERSION_MAJOR

  Initial value: `(1U)`

  Version of the API defined in this file
- #define SF_AUDIO_PLAYBACK_API_VERSION_MINOR

  Initial value: `(3U)`
- #define SF_AUDIO_PLAYBACK_MESSAGE_WORDS

  Initial value: `((sizeof(sf_message_payload_audio_t) + 3) / 4)`

  Audio playback message size in 4 byte words, rounded up.
- #define SF_AUDIO_PLAYBACK_MAX_VOLUME

  Initial value: `(255)`

  Macro defining the maximum volume.

6.3.7 API Data

6.3.7.1 sf_audio_playback_status_t

```
sf_audio_playback_status_t
```

Detailed description

Audio playback status.

Enumerated values

Name	Description
SF_AUDIO_PLAYBACK_STATUS_STOPPED	Stream is available to be used.
SF_AUDIO_PLAYBACK_STATUS_PAUSED	Stream is paused.
SF_AUDIO_PLAYBACK_STATUS_PLAYING	Stream is playing data.
SF_AUDIO_PLAYBACK_STATUS_WAITING	Stream is between packets, waiting for the next data message.

6.3.7.2 sf_audio_playback_common_ctrl_t

```
typedef void sf_audio_playback_common_ctrl_t
```

Detailed description

Audio playback common control block. Allocate an instance specific control block to pass to open in sf_audio_playback_cfg_t. Implemented as

- sf_audio_playback_common_instance_ctrl_t

6.3.7.3 sf_audio_playback_ctrl_t

```
typedef void sf_audio_playback_ctrl_t
```

Detailed description

Audio playback framework control block. Allocate an instance specific control block to pass into the audio playback framework API calls. Implemented as

- sf_audio_playback_instance_ctrl_t

6.3.8 API Structures

6.3.8.1 sf_audio_playback_data_t

sf_audio_playback_data_t

Detailed description

Audio data for playback.

Variables

- sf_message_header_t header

  Required common members of messaging framework payloads.

- sf_audio_playback_data_type_t type

  Data type. Must be uncompressed.

- uint32_t size_bytes

  Size of data in bytes.

- void const * p_data

  Pointer to data. Data start address must be 4-byte aligned.

- UINT loop_timeout

  ThreadX timeout, select TX_NO_WAIT to play the entire buffer once, TX_WAIT_FOREVER to loop until SF_AUDIO_PLAYBACK_Pause is called from another thread, or any timeout value from (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts to loop until the tick counts expire.

- bool stream_end

  This releases ownership of the stream and allows other threads to post data. Set to true if not more data will be sent as a part of this logical bitstream. Set to false if more packets are being prepared.

6.3.8.2 sf_audio_playback_common_cfg_t

sf_audio_playback_common_cfg_t

Detailed description

Common configuration for RTOS integrated audio framework. Shared by all streams.

Variables

- UINT priority

  Priority of the audio playback thread.

- sf_audio_playback_hw_instance_t const * p_lower_lvl_hw

  Hardware instance.

- sf_message_instance_t const * p_message

  Pointer to messaging framework instance used to post audio messages.

- TX_QUEUE * p_queue

  Pointer to the messaging framework queue specified for this audio stream. Must be subscribed to the SF_MESSAGE_EVENT_CLASS_AUDIO event class.

- void const * p_extend

  Implementation specific extension configuration.

6.3.8.3 sf_audio_playback_cfg_t

sf_audio_playback_cfg_t

Detailed description

Per stream configuration for RTOS integrated audio framework.

Variables

- void(* p_callback)(sf_message_callback_args_t *p_args)

  Callback called when playback of a buffer passed to sf_audio_playback_api_t::start is complete. Set to NULL for no callback.

- sf_audio_playback_common_ctrl_t * p_common_ctrl

  Pointer to the hardware control block used by this stream.

- sf_audio_playback_common_cfg_t const * p_common_cfg

  Pointer to common configurations shared by all streams using the same hardware.

- uint8_t class_instance

  Class instance used to identify the stream to the messaging framework.

6.3.8.4 sf_audio_playback_api_t

sf_audio_playback_api_t

Detailed description

Audio playback API structure. Audio playback implementations use the following API.

6.3.8.5 open

```
ssp_err_t(* sf_audio_playback_api_t::open) (sf_audio_playback_ctrl_t *const
p_ctrl, sf_audio_playback_cfg_t const *const p_cfg)
```

Brief description

Configure the audio framework by creating a thread for audio playback and configuring HAL layer drivers used. This function must be called before any other audio functions.

Detailed description

Implemented as

- SF_AUDIO_PLAYBACK_Open

Table 6: Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a device structure allocated by user. The device control structure is initialized in this function.

Table 6: Parameters (Continued)

Name	Direction	Description
p_cfg	in	Pointer to configuration structure. All elements of the structure must be set by user.

Parameter p_ctrl

Definition: `sf_audio_playback_ctrl_t*const p_ctrl`

Audio playback framework control block. Allocate an instance specific control block to pass into the audio playback framework API calls. Implemented assf_audio_playback_instance_ctrl_t

Parameter p_cfg

Definition: `sf_audio_playback_cfg_t const *const p_cfg`

Per stream configuration for RTOS integrated audio framework.

- `sf_audio_playback_cfg_t`::`p_callback`

  Callback called when playback of a buffer passed to sf_audio_playback_api_t::start is complete. Set to NULL for no callback.

- `sf_audio_playback_cfg_t`::`sf_audio_playback_common_ctrl_t`

  Pointer to the hardware control block used by this stream.

- `sf_audio_playback_cfg_t`::`sf_audio_playback_common_cfg_t`

  Pointer to common configurations shared by all streams using the same hardware.

- `sf_audio_playback_cfg_t`::`class_instance`

  Class instance used to identify the stream to the messaging framework.

6.3.8.6 close

```
ssp_err_t(* sf_audio_playback_api_t::close) (sf_audio_playback_ctrl_t *const p_ctrl)
```

Brief description

The close API handles cleans up internal driver data.

Detailed description

Implemented as

- SF_AUDIO_PLAYBACK_Close

Table 7: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for audio driver.

Parameter p_ctrl

Definition: `sf_audio_playback_ctrl_t*const p_ctrl`

Audio playback framework control block. Allocate an instance specific control block to pass into the audio playback framework API calls. Implemented assf_audio_playback_instance_ctrl_t

6.3.8.7 start

```
ssp_err_t(* sf_audio_playback_api_t::start) (sf_audio_playback_ctrl_t *const
p_ctrl, sf_audio_playback_data_t *const p_data, UINT const timeout)
```

Brief description

Play audio. Currently only 16-bit mono PCM buffers are supported.

Detailed description

Implemented as

- SF_AUDIO_PLAYBACK_Start

NOTE: Call SF_MESSAGE_Open to configure the messaging framework control block and queues with the parameters specified in sf_audio_playback_cfg_t::p_message and sf_audio_playback_cfg_t::p_queue.

Table 8: Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to device control block initialized in Open call for audio driver.
p_data	in	Pointer to data, description, timeout values, and synchronization options.

Table 8: Parameters (Continued)

Name	Direction	Description
timeout	in	ThreadX timeout, represents the maximum amount of time to wait to post to the audio queue. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout values from 0x00000001 through 0xFFFFFFFE in ThreadX tick counts.

Parameter p_ctrl

Definition: `sf_audio_playback_ctrl_t*const p_ctrl`

Audio playback framework control block. Allocate an instance specific control block to pass into the audio playback framework API calls. Implemented assf_audio_playback_instance_ctrl_t

Parameter p_data

Definition: `sf_audio_playback_data_t*const p_data`

Audio data for playback.

- `sf_audio_playback_data_t::header`

  Required common members of messaging framework payloads.

- `sf_audio_playback_data_t::type`

  Data type. Must be uncompressed.

- `sf_audio_playback_data_t::size_bytes`

  Size of data in bytes.

- `sf_audio_playback_data_t::p_data`

  Pointer to data. Data start address must be 4-byte aligned.

- `sf_audio_playback_data_t::loop_timeout`

  ThreadX timeout, select TX_NO_WAIT to play the entire buffer once, TX_WAIT_FOREVER to loop until SF_AUDIO_PLAYBACK_Pause is called from another thread, or any timeout value from (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts to loop until the tick counts expire.

- `sf_audio_playback_data_t::stream_end`

  This releases ownership of the stream and allows other threads to post data. Set to true if not more data will be sent as a part of this logical bitstream. Set to false if more packets are being prepared.

Parameter timeout

`const`

6.3.8.8 pause

```
ssp_err_t(* sf_audio_playback_api_t::pause) (sf_audio_playback_ctrl_t *const
p_ctrl)
```

Brief description

Pause audio playback. This stops the peripheral that triggers the DMA/DTC transfer and posts a flag to notify SF_AUDIO_PLAYBACK_Start to pause any playback in progress.

Detailed description

Implemented as

- SF_AUDIO_PLAYBACK_Pause

NOTE: Call SF_AUDIO_PLAYBACK_Start before using this function. Calling SF_AUDIO_PLAYBACK_Pause before SF_AUDIO_PLAYBACK_Start has no effect and does not return an error code.

Table 9: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for audio driver.

Parameter p_ctrl

Definition: `sf_audio_playback_ctrl_t*const p_ctrl`

Audio playback framework control block. Allocate an instance specific control block to pass into the audio playback framework API calls. Implemented assf_audio_playback_instance_ctrl_t

6.3.8.9 stop

```
ssp_err_t(* sf_audio_playback_api_t::stop) (sf_audio_playback_ctrl_t *const
p_ctrl)
```

Brief description

Stop audio playback. Causes SF_AUDIO_PLAYBACK_Start halt playback and return.

Detailed description

Implemented as

- SF_AUDIO_PLAYBACK_Stop

NOTE: Call SF_AUDIO_PLAYBACK_Start before using this function. Calling SF_AUDIO_PLAYBACK_Stop before SF_AUDIO_PLAYBACK_Start has no effect and does not return an error code.

Table 10: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for audio driver.

Parameter p_ctrl

Definition: `sf_audio_playback_ctrl_t*const p_ctrl`

Audio playback framework control block. Allocate an instance specific control block to pass into the audio playback framework API calls. Implemented assf_audio_playback_instance_ctrl_t

6.3.8.10 resume

```
ssp_err_t(* sf_audio_playback_api_t::resume) (sf_audio_playback_ctrl_t *const p_ctrl)
```

Brief description

Resume audio playback. Posts a flag to notify SF_AUDIO_PLAYBACK_Start to restart the peripheral that triggers the DMA/DTC transfer.

Detailed description

Implemented as

- SF_AUDIO_PLAYBACK_Resume

NOTE: Call SF_AUDIO_PLAYBACK_Pause before using this function. Calling SF_AUDIO_PLAYBACK_Resume before SF_AUDIO_PLAYBACK_Pause has no effect and does not return an error code.

Table 11: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for audio driver.

Parameter p_ctrl

Definition: `sf_audio_playback_ctrl_t*const p_ctrl`

Audio playback framework control block. Allocate an instance specific control block to pass into the audio playback framework API calls. Implemented assf_audio_playback_instance_ctrl_t

6.3.8.11 volumeSet

```
ssp_err_t(* sf_audio_playback_api_t::volumeSet) (sf_audio_playback_ctrl_t *const
p_ctrl, uint8_t const volume)
```

Brief description

Set software volume control. Software volume control is applied globally to all streams on the hardware.

Detailed description

Implemented as

- SF_AUDIO_PLAYBACK_VolumeSet

ATTENTION: Software volume control reduces resolution and may require extra memory and processing bandwidth.

Table 12: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for audio driver.
volume	in	Volume level requested. Valid range is from 0 (muted, which will stop playback) to 255 (maximum volume, default on open).

Parameter p_ctrl

Definition: `sf_audio_playback_ctrl_t*const p_ctrl`

Audio playback framework control block. Allocate an instance specific control block to pass into the audio playback framework API calls. Implemented assf_audio_playback_instance_ctrl_t

Parameter volume

`uint8_t`

6.3.8.12 versionGet

```
ssp_err_t(* sf_audio_playback_api_t::versionGet) (ssp_version_t *const p_version)
```

Brief description

Store version information in provided pointer.

Detailed description

Implemented as

- SF_AUDIO_PLAYBACK_VersionGet

Table 13: Parameters

Name	Direction	Description
p_version	in	Pointer to device control block initialized in Open call for UART driver.

Parameter p_version

6.3.8.13 sf_audio_playback_instance_t

sf_audio_playback_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_audio_playback_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_audio_playback_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_audio_playback_api_t const * p_api

  Pointer to the API structure for this instance.

6.4 Audio Playback Framework Interface

RTOS-integrated Audio Playback Framework Interface.

6.4.1 Summary

Audio playback driver to play buffers of audio data.

Implemented by: DAC Audio Playback Framework

Audio Framework Interface description: Audio Playback Framework

6.4.2 Interface API

sf_audio_playback_hw_api_t

Function name	Description
.open	Open a device channel for read/write and control.
.start	Start audio playback hardware.
.stop	Stop audio playback hardware.
.play	Play audio buffer.
.dataTypeGet	Stores expected data type in provided pointer p_data_type.
.close	Close the audio driver.
.versionGet	Return the version of the driver.

6.4.3 Data structures

- sf_audio_playback_data_type_t
- sf_audio_playback_hw_callback_args_t
- sf_audio_playback_hw_cfg_t
- sf_audio_playback_hw_instance_t

6.4.4 Typedefs

- sf_audio_playback_hw_ctrl_t

6.4.5 Defines

- #define SF_AUDIO_PLAYBACK_HW_API_VERSION_MAJOR
  Initial value: `(1U)`
- #define SF_AUDIO_PLAYBACK_HW_API_VERSION_MINOR
  Initial value: `(2U)`

6.4.6 API Data

6.4.6.1 sf_audio_playback_hw_ctrl_t

```
typedef void sf_audio_playback_hw_ctrl_t
```

Detailed description

Audio playback hardware control block. Allocate an instance specific control block to pass into the audio playback hardware API calls. Implemented as

- sf_audio_playback_hw_dac_instance_ctrl_t
- sf_audio_playback_hw_i2s_instance_ctrl_t

6.4.7 API Structures

6.4.7.1 sf_audio_playback_data_type_t

sf_audio_playback_data_type_t

Detailed description

Audio data type.

Variables

- uint8_t scale_bits_max

  Maximum data resolution in bits.

- bool is_signed

  Set to 1 for signed samples, or 0 for unsigned samples.

6.4.7.2 sf_audio_playback_hw_callback_args_t

sf_audio_playback_hw_callback_args_t

Detailed description

Callback function parameter data

Variables

- void * p_context

  Placeholder for user data. Set in sf_audio_playback_hw_api_t::open function in sf_audio_playback_hw_cfg_t.

6.4.7.3 sf_audio_playback_hw_cfg_t

sf_audio_playback_hw_cfg_t

Detailed description

Audio playback driver configuration.

Variables

- void(* p_callback)(sf_audio_playback_hw_callback_args_t *p_args)

  Callback called when play is complete. Set to NULL for no callback.
- void * p_context

  Placeholder for user data. Passed to the user callback in sf_audio_playback_hw_callback_args_t.
- void const * p_extend

  Hardware dependent configuration.

6.4.7.4 sf_audio_playback_hw_api_t

sf_audio_playback_hw_api_t

Detailed description

Audio playback API definition.

6.4.7.5 open

```
(* sf_audio_playback_hw_api_t::open) (sf_audio_playback_hw_ctrl_t *const p_ctrl,
sf_audio_playback_hw_cfg_t const *const p_cfg)
```

Detailed description

Open a device channel for read/write and control. Implemented as

- SF_AUDIO_PLAYBACK_HW_DAC_Open

Table 14: Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to memory allocated for control block.
p_cfg	in	Pointer to the hardware configurations.

Parameter p_ctrl

Definition: `sf_audio_playback_hw_ctrl_t*const p_ctrl`

Audio playback hardware control block. Allocate an instance specific control block to pass into the audio playback hardware API calls. Implemented
assf_audio_playback_hw_dac_instance_ctrl_tsf_audio_playback_hw_i2s_instance_ctrl_t

Parameter p_cfg

Definition: `sf_audio_playback_hw_cfg_t const *const p_cfg`

Audio playback driver configuration.

- sf_audio_playback_hw_cfg_t::p_callback

  Callback called when play is complete. Set to NULL for no callback.

- sf_audio_playback_hw_cfg_t::p_context

  Placeholder for user data. Passed to the user callback in sf_audio_playback_hw_callback_args_t.

- sf_audio_playback_hw_cfg_t::p_extend

  Hardware dependent configuration.

6.4.7.6 start

```
(* sf_audio_playback_hw_api_t::start) (sf_audio_playback_hw_ctrl_t *const p_ctrl)
```

Detailed description

Start audio playback hardware. Implemented as

- SF_AUDIO_PLAYBACK_HW_DAC_Start

Table 15: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block initialized in open.

Parameter p_ctrl

Definition: sf_audio_playback_hw_ctrl_t*const p_ctrl

Audio playback hardware control block. Allocate an instance specific control block to pass into the audio playback hardware API calls. Implemented
assf_audio_playback_hw_dac_instance_ctrl_tsf_audio_playback_hw_i2s_instance_ctrl_t

6.4.7.7 stop

```
(* sf_audio_playback_hw_api_t::stop) (sf_audio_playback_hw_ctrl_t *const p_ctrl)
```

Detailed description

Stop audio playback hardware. Implemented as

- SF_AUDIO_PLAYBACK_HW_DAC_Stop

Table 16: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block initialized in open.

Parameter p_ctrl

Definition: `sf_audio_playback_hw_ctrl_t*const p_ctrl`

Audio playback hardware control block. Allocate an instance specific control block to pass into the audio playback hardware API calls. Implemented assf_audio_playback_hw_dac_instance_ctrl_tsf_audio_playback_hw_i2s_instance_ctrl_t

6.4.7.8 play

```
(* sf_audio_playback_hw_api_t::play) (sf_audio_playback_hw_ctrl_t *const p_ctrl,
int16_t const *const p_buffer, uint32_t length)
```

Detailed description

Play audio buffer. Implemented as

- SF_AUDIO_PLAYBACK_HW_DAC_Play

Table 17: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block initialized in open.
p_buffer	in	Pointer to buffer with PCM samples to play. Data must be scaled for audio playback hardware.
length	in	Length of data in p_buffer.

Parameter p_ctrl

Definition: `sf_audio_playback_hw_ctrl_t*const p_ctrl`

Audio playback hardware control block. Allocate an instance specific control block to pass into the audio playback hardware API calls. Implemented assf_audio_playback_hw_dac_instance_ctrl_tsf_audio_playback_hw_i2s_instance_ctrl_t

Parameter p_buffer

Parameter length

uint32_t

6.4.7.9 dataTypeGet

(* sf_audio_playback_hw_api_t::dataTypeGet) (sf_audio_playback_hw_ctrl_t *const p_ctrl, sf_audio_playback_data_type_t *const p_data_type)

Detailed description

Stores expected data type in provided pointer p_data_type. Implemented as

- SF_AUDIO_PLAYBACK_HW_DAC_DataTypeGet

Table 18: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block initialized in open.
p_data_type	out	Pointer to audio sample data type required by hardware.

Parameter p_ctrl

Definition: sf_audio_playback_hw_ctrl_t*const p_ctrl

Audio playback hardware control block. Allocate an instance specific control block to pass into the audio playback hardware API calls. Implemented
assf_audio_playback_hw_dac_instance_ctrl_tsf_audio_playback_hw_i2s_instance_ctrl_t

Parameter p_data_type

Definition: sf_audio_playback_data_type_t*const p_data_type

Audio data type.

- sf_audio_playback_data_type_t::scale_bits_max

  Maximum data resolution in bits.

- sf_audio_playback_data_type_t::is_signed

  Set to 1 for signed samples, or 0 for unsigned samples.

6.4.7.10 close

(* sf_audio_playback_hw_api_t::close) (sf_audio_playback_hw_ctrl_t *const p_ctrl)

Detailed description

Close the audio driver. Implemented as

- SF_AUDIO_PLAYBACK_HW_DAC_Close

Table 19: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block initialized in open.

Parameter p_ctrl

Definition: `sf_audio_playback_hw_ctrl_t*const p_ctrl`

Audio playback hardware control block. Allocate an instance specific control block to pass into the audio playback hardware API calls. Implemented
assf_audio_playback_hw_dac_instance_ctrl_tsf_audio_playback_hw_i2s_instance_ctrl_t

6.4.7.11 versionGet

```
(* sf_audio_playback_hw_api_t::versionGet) ( *const p_version)
```

Detailed description

Return the version of the driver. Implemented as

- SF_AUDIO_PLAYBACK_HW_DAC_VersionGet

Table 20: Parameters

Name	Direction	Description
p_version	out	Pointer to variable that will be populated with version information.

Parameter p_version

6.4.7.12 sf_audio_playback_hw_instance_t

sf_audio_playback_hw_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_audio_playback_hw_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_audio_playback_hw_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_audio_playback_hw_api_t const * p_api

  Pointer to the API structure for this instance.

6.5 Audio Recording Framework Interface

RTOS-integrated Audio Recording Framework Interface.

6.5.1 Summary

The Audio Record Interface is a ThreadX-aware Interface for Audio Recording. The Interface is implemented by the ADC Audio Recording Framework using the ADC periodic Framework driver for recording. The interface is implemented by the I2S Audio Recording Framework using the I2S driver for recording.

Interfaces used:

- ADC Periodic Framework
- I2S Interface

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

6.5.2 Interface API

sf_audio_record_api_t

Function name	Description
.open	Initializes audio recording framework.
.start	Starts audio recording.
.stop	Stops audio recording.

Function name	Description
.infoGet	Gets channel information(Mono/Stereo).
.close	Releases channel mutex and closes channel at HAL layer.
.versionGet	Gets version and stores it in provided pointer p_version.

6.5.3 Data structures

- sf_audio_record_info_t
- sf_audio_record_callback_args_t
- sf_audio_record_cfg_t
- sf_audio_record_instance_t

6.5.4 Enumerations

- sf_audio_record_channel_t
- sf_audio_record_data_size_t
- sf_audio_record_event_t

6.5.5 Typedefs

- sf_audio_record_ctrl_t

6.5.6 Defines

- #define SF_AUDIO_RECORD_API_VERSION_MAJOR

  Initial value: `(1U)`

  Version of the API defined in this file
- #define SF_AUDIO_RECORD_API_VERSION_MINOR

  Initial value: `(2U)`

6.5.7 API Data

6.5.7.1 sf_audio_record_channel_t

```
sf_audio_record_channel_t
```

Detailed description

Definition of audio recording mode.

Enumerated values

Name	Description
SF_AUDIO_RECORD_CHANNEL_MONO	Support Mono Channel.
SF_AUDIO_RECORD_CHANNEL_STEREO	Support Stereo Channel.

6.5.7.2 sf_audio_record_data_size_t

```
sf_audio_record_data_size_t
```

Detailed description

Definition of audio recording data sample size.

Enumerated values

Name	Description
SF_AUDIO_RECORD_DATA_SIZE_8BIT	data width in the sample is 8bit
SF_AUDIO_RECORD_DATA_SIZE_16BIT	data width in the sample is 16bit

6.5.7.3 sf_audio_record_event_t

```
sf_audio_record_event_t
```

Detailed description

Definition of events for audio recording.

Enumerated values

Name	Description
SF_AUDIO_RECORD_EVENT_NEW_DATA	New data is available in the buffer.

6.5.7.4 sf_audio_record_ctrl_t

```
typedef void sf_audio_record_ctrl_t
```

Detailed description

Audio record framework control block. Allocate an instance specific control block to pass into the audio record framework API calls. Implemented as

- sf_audio_record_adc_instance_ctrl_t
- sf_audio_record_i2s_instance_ctrl_t

6.5.8 API Structures

6.5.8.1 sf_audio_record_info_t

sf_audio_record_info_t

Detailed description
Variables

- sf_audio_record_channel_t channel_info

6.5.8.2 sf_audio_record_callback_args_t

sf_audio_record_callback_args_t

Detailed description
Variables

- sf_audio_record_event_t event

  Audio callback event.
- uint32_t buffer_index

  Index to the buffer where the new data is stored.
- void const * p_context

  Placeholder for user data.

6.5.8.3 sf_audio_record_cfg_t

sf_audio_record_cfg_t

Detailed description

Configuration for audio recording framework

Variables

- sf_audio_record_data_size_t capture_data_size

  Size of data in the sample 8 or 16 bit.
- uint32_t sampling_rate_hz

  Sampling rate for audio capture.
- void * p_capture_data_buffer

  Pointer to the buffer that will store the samples
- uint32_t capture_data_buffer_size

  total size of buffer configured by user to store samples

- uint32_t sample_count

  Samples per channel to be buffered before notifying the user via callback
- void(* p_callback)(sf_audio_record_callback_args_t *p_args)

  Callback function.
- void const * p_context

  Placeholder for user data.
- void const * p_extend

  Extension parameter for hardware specific settings.

6.5.8.4 sf_audio_record_api_t

sf_audio_record_api_t

Detailed description

Framework Audio Recording API structure. Implementations will use the following API.

6.5.8.5 open

```
ssp_err_t(* sf_audio_record_api_t::open) (sf_audio_record_ctrl_t *const p_ctrl,
sf_audio_record_cfg_t const *const p_cfg)
```

Brief description

Initializes audio recording framework.

Detailed description

Implemented as

- SF_AUDIO_RECORD_ADC_Open
- SF_AUDIO_RECORD_I2S_Open

Table 21: Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a structure allocated by user. Elements initialized here.
p_cfg	in	Pointer to configuration structure. All elements of the structure must be set by user.

Parameter p_ctrl

Definition: `sf_audio_record_ctrl_t*const p_ctrl`

Audio record framework control block. Allocate an instance specific control block to pass into the audio record framework API calls. Implemented assf_audio_record_adc_instance_ctrl_tsf_audio_record_i2s_instance_ctrl_t

Parameter p_cfg

Definition: sf_audio_record_cfg_t const *const p_cfg

Configuration for audio recording framework

- sf_audio_record_cfg_t::sf_audio_record_data_size_t

  Size of data in the sample 8 or 16 bit.

  Enumerated as:

  – SF_AUDIO_RECORD_DATA_SIZE_8BIT

  – SF_AUDIO_RECORD_DATA_SIZE_16BIT
- sf_audio_record_cfg_t::sampling_rate_hz

  Sampling rate for audio capture.
- sf_audio_record_cfg_t::p_capture_data_buffer

  Pointer to the buffer that will store the samples
- sf_audio_record_cfg_t::capture_data_buffer_size

  total size of buffer configured by user to store samples
- sf_audio_record_cfg_t::sample_count

  Samples per channel to be buffered before notifying the user via callback
- sf_audio_record_cfg_t::p_callback

  Callback function.
- sf_audio_record_cfg_t::p_context

  Placeholder for user data.
- sf_audio_record_cfg_t::p_extend

  Extension parameter for hardware specific settings.

6.5.8.6 start

```
ssp_err_t(* sf_audio_record_api_t::start) (sf_audio_record_ctrl_t *const p_ctrl)
```

Brief description

Starts audio recording.

Detailed description

Implemented as

- SF_AUDIO_RECORD_ADC_Start
- SF_AUDIO_RECORD_I2S_Start

Table 22: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block set

Parameter p_ctrl

Definition: `sf_audio_record_ctrl_t*const p_ctrl`

Audio record framework control block. Allocate an instance specific control block to pass into the audio record framework API calls. Implemented assf_audio_record_adc_instance_ctrl_tsf_audio_record_i2s_instance_ctrl_t

6.5.8.7 stop

```
ssp_err_t(* sf_audio_record_api_t::stop) (sf_audio_record_ctrl_t *const p_ctrl)
```

Brief description

Stops audio recording.

Detailed description

Implemented as

- SF_AUDIO_RECORD_ADC_Stop
- SF_AUDIO_RECORD_I2S_Stop

Table 23: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block .

Parameter p_ctrl

Definition: `sf_audio_record_ctrl_t*const p_ctrl`

Audio record framework control block. Allocate an instance specific control block to pass into the audio record framework API calls. Implemented assf_audio_record_adc_instance_ctrl_tsf_audio_record_i2s_instance_ctrl_t

6.5.8.8 infoGet

```
ssp_err_t(* sf_audio_record_api_t::infoGet) (sf_audio_record_ctrl_t *const
p_ctrl, sf_audio_record_info_t *p_info)
```

Brief description

Gets channel information(Mono/Stereo).

Detailed description

Implemented as

- SF_AUDIO_RECORD_ADC_InfoGet
- SF_AUDIO_RECORD_I2S_InfoGet

Table 24: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block .

Parameter p_ctrl

Definition: `sf_audio_record_ctrl_t*const p_ctrl`

Audio record framework control block. Allocate an instance specific control block to pass into the audio record framework API calls. Implemented assf_audio_record_adc_instance_ctrl_tsf_audio_record_i2s_instance_ctrl_t

6.5.8.9 close

```
ssp_err_t(* sf_audio_record_api_t::close) (sf_audio_record_ctrl_t *const p_ctrl)
```

Brief description

Releases channel mutex and closes channel at HAL layer.

Detailed description

Implemented as

- SF_AUDIO_RECORD_ADC_Close
- SF_AUDIO_RECORD_I2S_Close

Table 25: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block .

Parameter p_ctrl

Definition: `sf_audio_record_ctrl_t*const p_ctrl`

Audio record framework control block. Allocate an instance specific control block to pass into the audio record framework API calls. Implemented assf_audio_record_adc_instance_ctrl_tsf_audio_record_i2s_instance_ctrl_t

6.5.8.10 versionGet

```
ssp_err_t(* sf_audio_record_api_t::versionGet) (ssp_version_t *const p_version)
```

Brief description

Gets version and stores it in provided pointer p_version.

Detailed description

Implemented as

- SF_AUDIO_RECORD_ADC_VersionGet
- SF_AUDIO_RECORD_I2S_VersionGet

Table 26: Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

6.5.8.11 sf_audio_record_instance_t

sf_audio_record_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_audio_record_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_audio_record_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_audio_record_api_t const * p_api

  Pointer to the API structure for this instance.

6.6 BLE Framework API

6.6.1 SF BLE Framework Interface

RTOS-integrated SF BLE Framework Interface.

6.6.1.1 Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Framework.

6.6.1.2 Interface API

sf_ble_api_t

Function name	Description
.open	Initializes the interface for data transfers. Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.
.close	De-initialize the interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.
.infoGet	Get BLE module information like chipset information and RSSI value.
.provisionGet	Reads the current BLE Provisioning information.
.provisionSet	Provisions the BLE Driver and set bonding and security modes as provisioned.
.scan	Scans for available BLE devices and return the list to the caller.
.advertisementStart	Make the device discoverable by broadcasting device information to remote devices.
.advertisementStop	Stop the device from being discoverable.
.whitelistAdd	Add specified devices to whitelist.
.whitelistDel	Remove specified devices from whitelist.
.bondingStart	Initiate bonding process with remote BLE device and exchange security keys if enabled.
.bondingResponse	Respond to the bonding request from the remote BLE device. Send bonding response on reception of bonding indication.
.startEncryption	Start encryption over a connection.
.connect	Connect to a remote BLE device.
.listen	Listen for connection request from remote device.
.authorization	Indicates that the specified remote device has been authorized by user.

Function name	Description
.disconnect	Terminate connection with remote BLE device.
.setTxPower	Sets the transmit power for the procedure specified by the connection handle.
.gattAddCustomProfiles	Add Custom Profile to GATT Database. This function is called with list of service and characteristics which is to be added to GATT database. Services and characteristics which were previously added will be removed and newly passed services and characteristics will be added.If services and characteristics were previously added and now those services and characteristics are to be removed and no new services and characteristics are to be added , then call this API with service and characteristics length as zero
.gattServiceDiscovery	Perform service discovery used by GATT client.Perform service discovery on remote GATT server and get results.
.gattCharDiscovery	Perform characteristics discovery used by GATT client.Perform characteristics discovery on remote GATT server and get results.
.gattCharDescDiscovery	Perform characteristics descriptor discovery used by GATT client.
.gattCharRead	Perform read characteristic used by GATT client.Read characteristic value from remote GATT server.
.gattCharWrite	Perform write characteristic used by GATT client.Write characteristic value on remote GATT server.
.gattCharExecuteWrite	Perform execute write on all pending write operations, used by GATT client.Execute or cancel all prepared writes of characteristics on remote GATT server.
.gattCharWriteLocal	Perform local characteristic write used by GATT server.Writes local characteristic value on GATT server.
.gattSendNotify	Send notification to GATT client, used by GATT server.
.gattSendIndicate	Send indication to GATT client, used by GATT server.Send indication to remote GATT client.
.gattWriteResponse	Send response to write operation received by GATT client, used by GATT server.Send response for write request received from remote GATT client.
.versionGet	Gets version and stores it in provided pointer p_version.

6.6.1.3 Data structures

- sf_ble_addr_t
- sf_ble_adv_data_t
- sf_ble_event_info_t
- sf_ble_sec_info_t
- sf_ble_provisioning_t
- sf_ble_scan_info_t
- sf_ble_bonding_start_t
- sf_ble_bonding_response_t
- sf_ble_sm_tk_info_t
- sf_ble_sm_tk_ind_t
- sf_ble_addr_verify_ind_t
- sf_ble_sm_key_ind_t
- sf_ble_sm_enc_info_t
- sf_ble_sec_enc_start_ind_t
- sf_ble_scan_response_data_t
- sf_ble_adv_info_t
- sf_ble_scan_t
- sf_ble_connection_t
- sf_ble_disconnect_t
- sf_ble_chipset_info_t
- sf_ble_info_t
- sf_ble_bonding_req_ind_t
- sf_ble_connect_info_t
- sf_ble_set_tx_pwr_info_t
- sf_ble_uuid_t
- sf_ble_svc_attribute_t
- sf_ble_char_attribute_t
- sf_ble_gatt_attr_event_t
- sf_ble_service_discovery_rsp_t
- sf_ble_service_discovery_req_t
- sf_ble_char_discovery_req_t
- sf_ble_char_discovery_rsp_t
- sf_ble_char_desc_discovery_rsp_t

- sf_ble_char_multiple_read_req_t
- sf_ble_char_read_req_t
- sf_ble_char_multiple_read_rsp_t
- sf_ble_char_read_by_uuid_rsp_t
- sf_ble_char_read_by_handle_rsp_t
- sf_ble_char_read_rsp_t
- sf_ble_char_write_req_t
- sf_ble_gatt_notif_ind_event_data_t
- sf_ble_write_cmd_event_data_t
- sf_ble_ctrl_t
- sf_ble_cfg_t
- sf_ble_instance_t

6.6.1.4 Enumerations

- sf_ble_event_t
- sf_ble_gap_role_t
- sf_ble_addr_type_t
- sf_ble_scan_mode_t
- sf_ble_bonding_mode_t
- sf_ble_sec_mode_t
- sf_ble_disc_type_t
- sf_ble_conn_type_t
- sf_ble_adv_type_t
- sf_ble_adv_filt_type_t
- sf_ble_duplicate_filter_t
- sf_ble_init_filt_type_t
- sf_ble_adv_chnl_map_t
- sf_ble_iocap_t
- sf_ble_auth_type_t
- sf_ble_disconnect_reason_t
- sf_ble_key_dist_t
- sf_ble_tx_pwr_state_t
- sf_ble_char_property_t
- sf_ble_service_discovery_t

- sf_ble_char_discovery_t
- sf_ble_uuid_length_t
- sf_ble_char_read_t
- sf_ble_char_write_t
- sf_ble_execute_write_t
- sf_ble_write_response_t
- sf_ble_attribute_error_code_t
- sf_ble_char_attr_permissions_t
- sf_ble_adv_event_type_t

6.6.1.5 Typedefs

- sf_ble_callback_t
- sf_ble_conn_handle_t

6.6.1.6 Defines

- #define SF_BLE_API_VERSION_MAJOR

  Initial value: `(1U)`

  Major Version of the API defined in this file
- #define SF_BLE_API_VERSION_MINOR

  Initial value: `(2U)`

  Minor Version of the API defined in this file
- #define SF_BLE_MAX_BLE_ADV_DATA_LEN

  Initial value: `(100U)`

  BLE advertising data length.
- #define SF_BLE_MAX_NAME_LEN

  Initial value: `(66U)`

  Maximum length of name for BLE device.
- #define SF_BLE_MAX_GAP_NAME_LEN

  Initial value: `(23U)`

  Maximum length of GAP name.
- #define SF_BLE_ADDR_LEN

  Initial value: `(6U)`

  BLE address length.

- #define SF_BLE_SEC_KEY_LEN

  Initial value: `(0x10U)`

  BLE Security Key length.
- #define SF_BLE_128BITS_UUID_LENGTH

  Initial value: `(16U)`

  128 bit UUID length
- #define SF_BLE_RAND_NUM_LENGTH

  Initial value: `(8U)`

  Rand Number length.
- #define SF_BLE_TRUE

  Initial value: `(1U)`

  Boolean True Condition.
- #define SF_BLE_FALSE

  Initial value: `(0U)`

  Boolean False Condition.
- #define SF_BLE_MAX_MULTI_CHAR_READ_CNT

  Initial value: `(4U)`

  Maximum characteristics count for multiple read
- #define SF_BLE_MAX_CHAR_UUID_READ_CNT

  Initial value: `(10U)`

  Maximum characteristics count in result of characteristic read by UUID
- #define SF_BLE_GATT_LEN_UNDEF

  Initial value: `(0xFF)`

  Variable length characteristic value which can not be defined
- #define SF_BLE_ADV_DATA_LEN

  Initial value: `(0x1F)`

  BLE Advertising Data Length
- #define SF_BLE_GATT_PRI_SERVICE

  Initial value: `(0x2800U)`

  BLE GATT Service type Primary
- #define SF_BLE_GATT_INCLUDE_SERVICE

  Initial value: `(0x2802U)`

  BLE GATT Service type Included

- #define SF_BLE_GATT_CHAR_DECLARE

  Initial value: `(0x2803U)`

  BLE GATT Declare Characteristics
- #define SF_BLE_TX_POWER_CONNECTION_HANDLE

  Initial value: `(0xFFFFU)`

  BLE connection handle value when TX power is set

6.6.1.7 API Data

sf_ble_event_t

`sf_ble_event_t`

Detailed description

BLE events

Enumerated values

Name	Description
SF_BLE_EVENT_NONE	BLE user event none.
SF_BLE_EVENT_BONDING_INDICATION	BLE user event indicating reception of bonding request.
SF_BLE_EVENT_CONNECTION_COMP	BLE user event indicating connection completion.
SF_BLE_EVENT_DISCONNECT_COMP	BLE user event indicating disconnect.
SF_BLE_EVENT_SM_TK_REQ_IND	BLE user event indicating request for Temporary key.
SF_BLE_EVENT_SM_KEY_IND	BLE user event indicating received key from user.
SF_BLE_EVENT_SM_CHK_BD_ADDR_REQ	BLE user event indicating validation of remote address.
SF_BLE_EVENT_SM_ENC_START_IND	BLE user event indicating encryption started.
SF_BLE_EVENT_GATT_NOTIFICATION	BLE user event for Notification from peer.
SF_BLE_EVENT_GATT_INDICATION	BLE user event for Indication from peer.
SF_BLE_EVENT_GATT_WRITE_CMD_INDICATION	BLE user event for Write command from peer.
SF_BLE_EVENT_GATT_RESPONSE_TIMEOUT	BLE user event for GATT operation timeout.
SF_BLE_EVENT_GATT_ATTR_WRITE_REQ	BLE user event for Remote Write request on user Attributes.

Name	Description
SF_BLE_EVENT_GATT_ATTR_READ_REQ	BLE user event for Remote Read request on user Attributes.
SF_BLE_EVENT_GATT_ATTR_WRITE_COMPLETE	BLE user event Indicating Remote Write complete for user Attributes.
SF_BLE_EVENT_GATT_ATTR_WRITE_CCCD_REQ	BLE user event for Remote CCCD Write request on user Attributes.
SF_BLE_EVENT_GATT_ATTR_WRITE_CCCD_COMPLETE	BLE user event Indicating Remote CCCD Write complete for user Attributes.

sf_ble_gap_role_t

`sf_ble_gap_role_t`

Detailed description

GAP role

Enumerated values

Name	Description
SF_BLE_GAP_ROLE_MASTER	GAP role master/central.
SF_BLE_GAP_ROLE_SLAVE	GAP role slave/peripheral.
SF_BLE_GAP_ROLE_OBSERVER	GAP role Observer.
SF_BLE_GAP_ROLE_BROADCASTER	GAP role Broadcaster.

sf_ble_addr_type_t

`sf_ble_addr_type_t`

Detailed description

BLE address type

Enumerated values

Name	Description
SF_BLE_ADDR_TYPE_PUBLIC	BLE address type public.
SF_BLE_ADDR_TYPE_RANDOM	BLE address type random.

sf_ble_scan_mode_t

`sf_ble_scan_mode_t`

Detailed description

BLE scan mode

Enumerated values

Name	Description
SF_BLE_SCAN_MODE_PASSIVE	BLE scan mode passive.
SF_BLE_SCAN_MODE_ACTIVE	BLE scan mode active.

sf_ble_bonding_mode_t

`sf_ble_bonding_mode_t`

Detailed description

BLE bonding mode

Enumerated values

Name	Description
SF_BLE_BONDING_MODE_NONBONDABLE	BLE bonding not supported.
SF_BLE_BONDING_MODE_BONDABLE	BLE bonding supported.

sf_ble_sec_mode_t

`sf_ble_sec_mode_t`

Detailed description

BLE security mode

Enumerated values

Name	Description
SF_BLE_SEC_MODE1_LVL1_NO_SEC	BLE no security.
SF_BLE_SEC_MODE1_LVL2_NOAUTH_PAIR_ENC	BLE no authentication pairing encryption.
SF_BLE_SEC_MODE1_LVL3_AUTH_PAIR_ENC	Authenticated pairing with encryption.
SF_BLE_SEC_MODE1_LVL4_AUTHLE_PAIR_ENC	Authenticated LE Secure Connections pairing with encryption.

Name	Description
SF_BLE_SEC_MODE2_LVL1_NOAUTH_DATA_SIGNED	Unauthenticated pairing with data signing.
SF_BLE_SEC_MODE2_LVL2_AUTH_DATA_SIGNED	Authentication pairing with data signing.

sf_ble_disc_type_t

`sf_ble_disc_type_t`

Detailed description

BLE discovery type

Enumerated values

Name	Description
SF_BLE_DISC_TYPE_NON_DISCOVERABLE	BLE non-discoverable.
SF_BLE_DISC_TYPE_GENERAL_DISCOVERABLE	BLE discoverable.
SF_BLE_DISC_TYPE_LIMITED_DISCOVERABLE	BLE limited discoverable.

sf_ble_conn_type_t

`sf_ble_conn_type_t`

Detailed description

BLE connection type

Enumerated values

Name	Description
SF_BLE_CONN_TYPE_NON_CONNECTABLE	Connection type connection not allowed.
SF_BLE_CONN_TYPE_UNDIRECTED_CONNECTABLE	Connection type undirected.
SF_BLE_CONN_TYPE_DIRECTED_CONNECTABLE	Connection type directed.

sf_ble_adv_type_t

`sf_ble_adv_type_t`

Detailed description

BLE advertisement type

Enumerated values

Name	Description
SF_BLE_ADV_TYPE_CONN_UNDIR	Connectable Undirected advertising.
SF_BLE_ADV_TYPE_CONN_DIR_HIGH_DUTY	Connectable high duty cycle directed advertising.
SF_BLE_ADV_TYPE_DISC_UNDIR	Discoverable undirected advertising.
SF_BLE_ADV_TYPE_NONCONN_UNDIR	Non-connectable undirected advertising.
SF_BLE_ADV_TYPE_CONN_DIR_LOW_DUTY	Connectable low duty cycle directed advertising.

sf_ble_adv_filt_type_t

`sf_ble_adv_filt_type_t`

Detailed description

BLE advertisement filter type

Enumerated values

Name	Description
SF_BLE_ADV_FILT_TYPE_ALLOW_ALL	Filter type allow all.
SF_BLE_ADV_FILT_TYPE_ALLOW_SCAN_WLIST_CON_ANY	Filter type scan only whitelist entries, connect to any.
SF_BLE_ADV_FILT_TYPE_ALLOW_SCAN_ANY_CON_WLIST	Filter type scan any, connect to whitelist entries only.
SF_BLE_ADV_FILT_TYPE_ALLOW_WLIST_ONLY	Filter type allow whitelist only for both scan and connection.

sf_ble_duplicate_filter_t

`sf_ble_duplicate_filter_t`

Detailed description

BLE Duplicate Filter Configuration

Enumerated values

Name	Description
SF_BLE_DUPLICATE_FILTER_ENABLE	Enable Duplicate filter in Scan.

Name	Description
SF_BLE_DUPLICATE_FILTER_DISABLE	Disbale Duplicate filter in Scan.

sf_ble_init_filt_type_t

`sf_ble_init_filt_type_t`

Detailed description

BLE initial filter type

Enumerated values

Name	Description
SF_BLE_INIT_FILT_TYPE_IGNORE_WLIST	Ignore whitelist.
SF_BLE_INIT_FILT_TYPE_USE_WLIST	Use whitelist.

sf_ble_adv_chnl_map_t

`sf_ble_adv_chnl_map_t`

Detailed description

BLE advertisement channel map

Enumerated values

Name	Description
SF_BLE_ADV_CHNL_37	Enable channel 37 use.
SF_BLE_ADV_CHNL_38	Enable channel 38 use.
SF_BLE_ADV_CHNL_39	Enable channel 39 use.
SF_BLE_ADV_CHNL_ALL_CHANNELS	all channels(37, 38 and 39) enabled

sf_ble_iocap_t

`sf_ble_iocap_t`

Detailed description

BLE System IO Capabilities

Enumerated values

Name	Description
SF_BLE_IO_CAP_DISPLAY_ONLY	Display Only.
SF_BLE_IO_CAP_DISPLAY_YES_NO	Display Yes No.
SF_BLE_IO_CAP_KB_ONLY	Keyboard Only.
SF_BLE_IO_CAP_NO_INPUT_NO_OUTPUT	No Input No Output.
SF_BLE_IO_CAP_KB_DISPLAY	Keyboard Display.

sf_ble_auth_type_t

`sf_ble_auth_type_t`

Detailed description

BLE security authorization requirement

Enumerated values

Name	Description
SF_BLE_AUTH_TYPE_NONE	No Security.
SF_BLE_AUTH_TYPE_UNAUTHENTICATED_NO_MITM	No MITM.
SF_BLE_AUTH_TYPE_AUTHENTICATED_MITM	MITM Protected.

sf_ble_disconnect_reason_t

`sf_ble_disconnect_reason_t`

Detailed description

BLE security authorization requirement

Enumerated values

Name	Description
SF_BLE_DISCONNECT_REASON_LOCAL_HOST	Disconnected by local machine.
SF_BLE_DISCONNECT_REASON_REMOTE_USER	Disconnected by remote device.
SF_BLE_DISCONNECT_REASON_ERR	Error in Connection so disconnected.

sf_ble_key_dist_t

`sf_ble_key_dist_t`

Detailed description

BLE security key distribution type

Enumerated values

Name	Description
SF_BLE_KEY_DIST_NONE	No Keys to distribute.
SF_BLE_KEY_DIST_ENCKEY	Encryption key in distribution.
SF_BLE_KEY_DIST_IDKEY	IRK (ID key)in distribution.
SF_BLE_KEY_DIST_SIGNKEY	CSRK(Signature key) in distribution.

sf_ble_tx_pwr_state_t

`sf_ble_tx_pwr_state_t`

Detailed description

BLE Transmit power state

Enumerated values

Name	Description
SF_BLE_TX_PWR_STATE_NORMAL	Functionality disabled.
SF_BLE_TX_PWR_STATE_ADAPT_NEAR	RF low-power mode.
SF_BLE_TX_PWR_STATE_ADAPT_MIDDLE	RF normal mode.
SF_BLE_TX_PWR_STATE_ADAPT_FAR	RF high-performance mode.

sf_ble_char_property_t

`sf_ble_char_property_t`

Detailed description

Characteristic property

Enumerated values

Name	Description
SF_BLE_CHAR_PROPERTY_BCAST	Permits broadcasts of the Characteristic Value.
SF_BLE_CHAR_PROPERTY_RD	Permits reads of the Characteristic Value.
SF_BLE_CHAR_PROPERTY_WR_NO_RESP	Permits writes of the Characteristic Value without response.
SF_BLE_CHAR_PROPERTY_WR	Permits writes of the Characteristic Value with response.
SF_BLE_CHAR_PROPERTY_NTF	Permits notifications of the Characteristic Value without acknowledgment.
SF_BLE_CHAR_PROPERTY_IND	Permits indications of a Characteristic Value with acknowledgment.
SF_BLE_CHAR_PROPERTY_AUTH	Permits signed writes to the Characteristic Value.
SF_BLE_CHAR_PROPERTY_EXT_PROP	Additional characteristic properties.

sf_ble_service_discovery_t

`sf_ble_service_discovery_t`

Detailed description

Service discovery type

Enumerated values

Name	Description
SF_BLE_SERVICE_DISCOVERY_PRIMARY_ALL	Discover all primary services.
SF_BLE_SERVICE_DISCOVERY_PRIMARY_UUID	Discover primary service by UUID.
SF_BLE_SERVICE_DISCOVERY_INCLUDED	Discover includes services.

sf_ble_char_discovery_t

`sf_ble_char_discovery_t`

Detailed description

Characteristics discovery type

Enumerated values

Name	Description
SF_BLE_CHAR_DISCOVERY_ALL	Discover all characteristics.
SF_BLE_CHAR_DISCOVERY_UUID	Discover characteristics by UUID.

sf_ble_uuid_length_t

`sf_ble_uuid_length_t`

Detailed description

UUID length

Enumerated values

Name	Description
SF_BLE_UUID_LENGTH_16BITS	16 bit UUID
SF_BLE_UUID_LENGTH_32BITS	32 bit UUID
SF_BLE_UUID_LENGTH_128BITS	128 bit UUID

sf_ble_char_read_t

`sf_ble_char_read_t`

Detailed description

Characteristics read type

Enumerated values

Name	Description
SF_BLE_CHAR_READ_BY_HANDLE	Read single characteristic by handle.
SF_BLE_CHAR_READ_BY_UUID	Read single characteristic by UUID.
SF_BLE_CHAR_READ_LONG_BY_HANDLE	Read single long characteristic by handle.
SF_BLE_CHAR_DESC_READ_BY_HANDLE	Read single characteristic descriptor by handle.
SF_BLE_CHAR_DESC_READ_LONG_BY_HANDLE	Read single long characteristic descriptor by handle.
SF_BLE_CHAR_READ_MULTIPLE_BY_HANDLES	Read multiple characteristics by handles.

sf_ble_char_write_t

`sf_ble_char_write_t`

Detailed description

Characteristics write type

Enumerated values

Name	Description
SF_BLE_CHAR_WRITE_NO_RESPONSE	Write single characteristic with no response.
SF_BLE_CHAR_WRITE	Write single characteristic with response.
SF_BLE_CHAR_WRITE_LONG	Write single long characteristic with response.
SF_BLE_CHAR_DESC_WRITE	Write single characteristic descriptor with response.
SF_BLE_CHAR_DESC_WRITE_LONG	Write single long characteristic descriptor with response.

sf_ble_execute_write_t

`sf_ble_execute_write_t`

Detailed description

Characteristic execute write flag

Enumerated values

Name	Description
SF_BLE_EXECUTE_WRITE_FALSE	Execute write false i.e. cancel write operations.
SF_BLE_EXECUTE_WRITE_TRUE	Execute write true i.e. execute write operations.

sf_ble_write_response_t

`sf_ble_write_response_t`

Detailed description

Characteristic execute write flag

Enumerated values

Name	Description
SF_BLE_WRITE_RESPONSE_NOT_REQUIRED	Write response not required.

Name	Description
SF_BLE_WRITE_RESPONSE_REQUIRED	Write response required.

sf_ble_attribute_error_code_t

`sf_ble_attribute_error_code_t`

Detailed description

BLE Attribute write response code

Enumerated values

Name	Description
SF_BLE_ATTRIBUTE_ERR_SUCCESS	Success.
SF_BLE_ATTRIBUTE_ERR_INVALID_HANDLE	Invalid handle.
SF_BLE_ATTRIBUTE_ERR_WRITE_NOT_PERMITTED	Writing is not permitted.
SF_BLE_ATTRIBUTE_ERR_INSUFF_AUTHEN	Authentication required for the request.
SF_BLE_ATTRIBUTE_ERR_REQUEST_NOT_SUPPORTED	Unsupported request.
SF_BLE_ATTRIBUTE_ERR_INVALID_OFFSET	Invalid offset.
SF_BLE_ATTRIBUTE_ERR_INSUFF_AUTHOR	Authorization required for the request.
SF_BLE_ATTRIBUTE_ERR_ATTRIBUTE_NOT_FOUND	The attribute could not be found.
SF_BLE_ATTRIBUTE_ERR_ATTRIBUTE_NOT_LONG	The attribute is not long enough.
SF_BLE_ATTRIBUTE_ERR_INVALID_ATTRIBUTE_VAL_LEN	Invalid attribute value size.
SF_BLE_ATTRIBUTE_ERR_INSUFF_ENC	Encryption required for the request.
SF_BLE_ATTRIBUTE_ERR_OUT_OF_RANGE	Out of Range.

sf_ble_char_attr_permissions_t

`sf_ble_char_attr_permissions_t`

Detailed description

BLE GATT Attribute characteristics permission

Enumerated values

Name	Description
SF_BLE_ATT_PERMISSIONS_ALLACCESS	Full access.
SF_BLE_ATT_PERMISSIONS_READ_AUTHENTICATION	Read with authentication.
SF_BLE_ATT_PERMISSIONS_READ_ENCRYPTION	Read with encryption.
SF_BLE_ATT_PERMISSIONS_READ_AUTHORIZATION	Read with authorization.
SF_BLE_ATT_PERMISSIONS_READ_FORBIDDEN	Read forbidden.
SF_BLE_ATT_PERMISSIONS_WRITE_AUTHENTICATION	Write with authentication.
SF_BLE_ATT_PERMISSIONS_WRITE_ENCRYPTION	Write with encryption.
SF_BLE_ATT_PERMISSIONS_WRITE_AUTHORIZATION	Write with authorization.
SF_BLE_ATT_PERMISSIONS_WRITE_FORBIDDEN	Write forbidden.
SF_BLE_ATT_PERMISSIONS_NOACCESS	No access.

sf_ble_adv_event_type_t

`sf_ble_adv_event_type_t`

Detailed description

BLE advertising event type

Enumerated values

Name	Description
SF_BLE_ADV_EVENT_TYPE_CONN_UND_ADV	Connectable undirected advertising.
SF_BLE_ADV_EVENT_TYPE_CONN_DIR_ADV	Connectable directed advertising.
SF_BLE_ADV_EVENT_TYPE_SCANNABLE_UND_ADV	Scannable undirected advertising.
SF_BLE_ADV_EVENT_TYPE_NON_CONN_UND_ADV	Non connectable undirected advertising.
SF_BLE_ADV_EVENT_TYPE_SCAN_RESPONSE	Scan response.

sf_ble_callback_t

```
typedef void(* sf_ble_callback_t) (sf_ble_event_info_t *ev)
```

Detailed description

BLE Callback Type

sf_ble_conn_handle_t

```
typedef uint16_t sf_ble_conn_handle_t
```

Detailed description

BLE Connection Handle

6.6.1.8 API Structures

sf_ble_addr_t

sf_ble_addr_t

Detailed description

BLE address

Variables

- uint8_t addr[SF_BLE_ADDR_LEN]

  6-byte array address value

sf_ble_adv_data_t

sf_ble_adv_data_t

Detailed description

BLE Advertising data structure

Variables

- uint8_t data[SF_BLE_ADV_DATA_LEN]

  Advertising data bytes array.

- uint8_t adv_data_length

  Advertising data length.

sf_ble_event_info_t

sf_ble_event_info_t

Detailed description

BLE event info

Variables

- void * p_data

  Data for BLE event.

- uint16_t event

  Event type.

sf_ble_sec_info_t

sf_ble_sec_info_t

Detailed description

Security Related Configuration

Variables

- sf_ble_sec_mode_t sec_mode

  security mode

- uint8_t id_key[SF_BLE_SEC_KEY_LEN]

  GAP Identity Resolving Security key.

- uint8_t csrk_key[SF_BLE_SEC_KEY_LEN]

  GAP Connection Signature Resolving Security key.

- uint8_t ltk_key[SF_BLE_SEC_KEY_LEN]

  GAP Connection Long term Security key.

- uint8_t rand_num[SF_BLE_RAND_NUM_LENGTH]

  GAP Connection Random number for Long term Security key.

- uint16_t ediv

  GAP Connection Encrypted Diversifier for Long term Security key.

sf_ble_provisioning_t

sf_ble_provisioning_t

Detailed description

BLE provisioning information

Variables

- uint8_t gap_name[SF_BLE_MAX_GAP_NAME_LEN]

  GAP name.

- uint8_t bcast_mode

  broadcast mode

- sf_ble_bonding_mode_t bonding_mode

  bonding mode

- sf_ble_sec_info_t sec_info

  Security information.

- sf_ble_gap_role_t gap_role

  GAP role (master/slave)

- sf_ble_callback_t p_ble_callback

  GAP user event callback that runs in driver thread context. Application should make sure callback logic is as minimal as possible without any blocking calls

sf_ble_scan_info_t

sf_ble_scan_info_t

Detailed description

BLE scan information structure

Variables

- uint16_t total_scan_duration

  BLE total scan duration in milliseconds.

- sf_ble_scan_mode_t scan_mode

  BLE scan mode.

- sf_ble_disc_type_t discovery_type

  Specifies discovery type, Used only in active scan.

- sf_ble_addr_type_t address_type

  BLE address type.

- sf_ble_adv_filt_type_t filt_policy

  Scan Filter policy.

- sf_ble_duplicate_filter_t duplicate_filt

  Duplicate filter configuration.

sf_ble_bonding_start_t

sf_ble_bonding_start_t

Detailed description

BLE Bonding Start information

Variables

- sf_ble_iocap_t iocap

  IO capabilities.

- uint8_t key_size

  Encryption key size.

- sf_ble_key_dist_t ikey_dist

  Initiator key distribution.

- sf_ble_key_dist_t rkey_dist

  Responder key distribution.

sf_ble_bonding_response_t

sf_ble_bonding_response_t

Detailed description

Bonding Response Parameter

Variables

- uint8_t accept

  accept or reject bonding

- sf_ble_iocap_t io_cap

  IO capabilities.

- uint8_t max_key_size

  Max key size.

- sf_ble_key_dist_t ikeys

  Initiator key distribution.

- sf_ble_key_dist_t rkeys

  Responder key distribution.

sf_ble_sm_tk_info_t

sf_ble_sm_tk_info_t

Detailed description

Security structures BLE Temporary Key information

Variables

- sf_ble_conn_handle_t conhdl

  Connection handle.

- uint8_t disp_en

  Whether to enable display.

sf_ble_sm_tk_ind_t

sf_ble_sm_tk_ind_t

Detailed description

BLE Temporary Key indication

Variables

- sf_ble_sm_tk_info_t tk_info

  input parameter,information to app about temporary key request

- uint8_t tk_req_status

  output parameter: request status, application has to set this in callback event whether request is valid or not
- uint8_t tk_key[SF_BLE_SEC_KEY_LEN]

  application has to set this in callback event temporary key for encryption

sf_ble_addr_verify_ind_t

sf_ble_addr_verify_ind_t

Detailed description

BLE Bluetooth address verification indication

Variables

- uint8_t bd_addr[SF_BLE_ADDR_LEN]

  input parameter specifying bluetooth address of remote device
- sf_ble_addr_type_t addr_type

  input parameter specifying Address type of remote BLE device
- uint8_t accept_addr

  output parameter: application has to set this parameter in callback if it accepts this address

sf_ble_sm_key_ind_t

sf_ble_sm_key_ind_t

Detailed description

BLE Received Key information

Variables

- uint8_t conn_idx

  connection index
- sf_ble_key_dist_t key_code

  type of security key
- uint16_t ediv

  BLE Security EDIV.
- uint8_t rand_num[SF_BLE_RAND_NUM_LENGTH]

  Random number for security.
- uint8_t ltk_key[SF_BLE_SEC_KEY_LEN]

  BLE long term security key.

sf_ble_sm_enc_info_t

sf_ble_sm_enc_info_t

Detailed description

BLE Start Encryption information

Variables

- sf_ble_conn_handle_t conn_idx

  connection index
- uint16_t ediv

  BLE Security EDIV.
- uint8_t rand_num[SF_BLE_RAND_NUM_LENGTH]

  Random number for security.
- uint8_t ltk_key[SF_BLE_SEC_KEY_LEN]

  BLE long term security key.

sf_ble_sec_enc_start_ind_t

sf_ble_sec_enc_start_ind_t

Detailed description

Encryption Start Indication Event for user

Variables

- sf_ble_conn_handle_t conhdl

  Connection handle of the remote device.
- uint8_t status

  Result of encryption start, 0 = SUCCESS else error.
- uint8_t key_size

  Key Size.
- sf_ble_auth_type_t auth_type

  Security properties.
- uint8_t bonding_status

  Bonding Status, 0 = Unbonded, 1 = Bonded.

sf_ble_scan_response_data_t

sf_ble_scan_response_data_t

Detailed description

Scan Response Data for advertising

Variables

- uint8_t scan_response_data[SF_BLE_ADV_DATA_LEN]

  Advertising data bytes array.

- uint8_t scan_response_data_length

  Advertising data length.

sf_ble_adv_info_t

sf_ble_adv_info_t

Detailed description

BLE advertisement information

Variables

- sf_ble_disc_type_t disc_mode

  Discovery mode.
- sf_ble_conn_type_t conn_mode

  Connection mode.
- sf_ble_adv_type_t adv_type

  Advertisement type.
- uint16_t adv_intv_min

  Minimum interval for advertising.
- uint16_t adv_intv_max

  Maximum interval for advertising.
- sf_ble_addr_type_t own_addr_type

  Own address type.
- sf_ble_addr_type_t direct_addr_type

  Direct connection address type.
- uint8_t direct_bd_addr[SF_BLE_ADDR_LEN]

  Direct connection BLE address.
- sf_ble_adv_chnl_map_t adv_chnl_map

  Advertising channel map.
- sf_ble_adv_filt_type_t adv_filt_policy

  Advertising filter policy.
- sf_ble_scan_response_data_t scan_response_data

  Scan Response information, will be advertised only for active scan.
- sf_ble_adv_data_t adv_data

  Advertising data.

sf_ble_scan_t

sf_ble_scan_t

Detailed description

BLE scan information

Variables

- sf_ble_addr_type_t addr_type

  Address type of remote BLE device.
- uint8_t bd_addr[SF_BLE_ADDR_LEN]

  Remote BLE address.
- uint16_t data_len

  Scan data length.
- uint8_t data[SF_BLE_MAX_BLE_ADV_DATA_LEN]

  Scan data.
- int16_t rssi

  RSSI value.
- sf_ble_adv_event_type_t event_type

  Advertising event type.

sf_ble_connection_t

sf_ble_connection_t

Detailed description

BLE connection information

Variables

- sf_ble_init_filt_type_t init_filt_type

  Connection filter type.
- sf_ble_addr_type_t addr_type

  BLE address type.
- uint8_t bd_addr[SF_BLE_ADDR_LEN]

  BLE address.

sf_ble_disconnect_t

sf_ble_disconnect_t

Detailed description

BLE Disconnect Event Information

Variables

- sf_ble_disconnect_reason_t reason

  Disconnection reason.

- uint8_t status

  Disconnect status if initiated by local host.
- sf_ble_conn_handle_t conhdl

  Connection handle of the remote device.

sf_ble_chipset_info_t

sf_ble_chipset_info_t

Detailed description

BLE chipset information

Variables

- uint32_t version

  chipset version
- uint8_t bd_addr[SF_BLE_ADDR_LEN]

  BLE address.

sf_ble_info_t

sf_ble_info_t

Detailed description

BLE module information

Variables

- sf_ble_chipset_info_t chipset

  Chipset information.
- uint16_t rssi

  RSSI value.

sf_ble_bonding_req_ind_t

sf_ble_bonding_req_ind_t

Detailed description

BLE bonding request indication information

Variables

- sf_ble_addr_t bd_addr

  BLE address.
- uint8_t index

  Connection index.
- uint8_t auth_req

  Authentication request type.

- uint8_t io_cap

  IO capability.
- uint8_t oob_data_flg

  Indicating if OOB data is present.
- uint8_t max_enc_size

  Maximum encryption key size.
- uint8_t ikey_dist

  Type of key distributed by the initiator.
- uint8_t rkey_dist

  Type of key distributed by the responder.

sf_ble_connect_info_t

sf_ble_connect_info_t

Detailed description

BLE connection information

Variables

- uint8_t status

  Confirmation status.
- uint8_t reserved

  Reserved.
- sf_ble_conn_handle_t conhdl

  Connection handle.
- uint8_t peer_addr_type

  Peer address type.
- sf_ble_addr_t peer_addr

  Peer BT address.
- uint8_t reserved2

  Reserved.
- uint16_t con_interval

  Connection interval.
- uint16_t con_latency

  Connection latency.
- uint16_t sup_to

  Link supervision time-out.

- uint8_t clk_accuracy

  Clock accuracy.
- uint8_t reserved3

  Reserved.

sf_ble_set_tx_pwr_info_t

sf_ble_set_tx_pwr_info_t

Detailed description

BLE set TX power information structure

Variables

- int8_t power_lvl

  TX power level in dBm.
- sf_ble_tx_pwr_state_t state

  Operating state to set the transmit power level.

sf_ble_uuid_t

sf_ble_uuid_t

Detailed description

Union holding either 16-bit/32-bit/128-bit UUID

Variables

- uint16_t uuid16

  16 bit UUID
- uint32_t uuid32

  32 bit UUID
- uint8_t uuid128[SF_BLE_128BITS_UUID_LENGTH]

  128 bit UUID
- union{} union{}

  See source code for definition of this member.
- sf_ble_uuid_length_t uuid_length

  Service UUID length.

sf_ble_svc_attribute_t

sf_ble_svc_attribute_t

Detailed description

BLE GATT Service Attributes

Variables

- uint16_t attr_handle

  Service Handle.
- uint16_t attr_type

  Attribute type such as SF_BLE_GATT_PRI_SERVICE, SF_BLE_GATT_INCLUDE_SERVICE.
- uint16_t parent_svc_handle

  parent_svc_handle is service handle of parent service which is already registered

  If service is included service,
- uint8_t * p_attr_value

  Service UUID Pointer.
- sf_ble_uuid_length_t attr_value_len

  UUID Length.

sf_ble_char_attribute_t

sf_ble_char_attribute_t

Detailed description

BLE GATT Characteristics Attributes

Variables

- sf_ble_svc_attribute_t * p_service

  Service to which this characteristics belongs.
- sf_ble_uuid_t attr_uuid

  UUID of characteristics value.
- uint16_t attr_declare_handle

  Characteristics handle.
- uint16_t attr_declare_type

  Characteristics declare type SF_BLE_GATT_CHAR_DECLARE.
- uint16_t attr_value_handle

  Characteristics value handle.
- sf_ble_char_attr_permissions_t attr_perm

  Characteristics permission.
- sf_ble_char_property_t attr_properties

  Characteristics properties.
- uint8_t * p_attr_value

  Characteristics value data.

- uint8_t attr_value_len

  Characteristics value data length.

sf_ble_gatt_attr_event_t

sf_ble_gatt_attr_event_t

Detailed description

BLE GATT Characteristics Remote Event data, passed in provision callback for GATT Custom profile

Variables

- uint16_t attr_handle

  Attribute handle for which event is generated.

- uint16_t offset

  The offset at which the client is willing to write.

- uint8_t const * p_value

  The value at which the client is willing to write.

- uint16_t length

  The amount of bytes that the client is willing to write as from this offset.

- ssp_err_t response

  User will update this variable, Pass SSP_SUCCESS if user accepts the remote request.

sf_ble_service_discovery_rsp_t

sf_ble_service_discovery_rsp_t

Detailed description

Service discovery result

Variables

- uint16_t service_handle

  Service handle.

- sf_ble_uuid_t uuid

  Service UUID.

- uint16_t start_handle

  Start handle of sub-attributes handle range.

- uint16_t end_handle

  End handle of sub-attributes handle range.

sf_ble_service_discovery_req_t

sf_ble_service_discovery_req_t

Detailed description

Service discovery request

Variables

- sf_ble_uuid_t uuid

  Service UUID.
- uint16_t start_handle

  Discovery start handle.
- uint16_t end_handle

  Discovery end handle.
- sf_ble_service_discovery_t discovery_type

  Service discovery type.

sf_ble_char_discovery_req_t

sf_ble_char_discovery_req_t

Detailed description

Characteristic discovery request

Variables

- sf_ble_uuid_t uuid

  Characteristic UUID.
- uint16_t start_handle

  Discovery start handle.
- uint16_t end_handle

  Discovery end handle.
- sf_ble_char_discovery_t discovery_type

  Characteristic discovery type.

sf_ble_char_discovery_rsp_t

sf_ble_char_discovery_rsp_t

Detailed description

Characteristic discovery result

Variables

- uint16_t char_handle

  Characteristic handle.
- sf_ble_uuid_t uuid

  Characteristic UUID.

- uint16_t value_handle

  Characteristic value handle.
- sf_ble_char_property_t property

  Characteristic property.

sf_ble_char_desc_discovery_rsp_t

sf_ble_char_desc_discovery_rsp_t

Detailed description

Characteristic descriptor discovery result

Variables

- uint16_t desc_handle

  Characteristic descriptor handle.
- sf_ble_uuid_t uuid

  Characteristic descriptor UUID.

sf_ble_char_multiple_read_req_t

sf_ble_char_multiple_read_req_t

Detailed description

Multiple Characteristic descriptor read request

Variables

- uint16_t handles[SF_BLE_MAX_MULTI_CHAR_READ_CNT]

  Characteristic handles for multiple read.
- uint8_t expected_result_size[SF_BLE_MAX_MULTI_CHAR_READ_CNT]

  Expected result length in bytes.

sf_ble_char_read_req_t

sf_ble_char_read_req_t

Detailed description

Read Characteristic request

Variables

- sf_ble_char_read_t char_read_type

  Characteristic read type.
- uint16_t handle

  Characteristic value or descriptor handle.
- sf_ble_uuid_t uuid

  Characteristic UUID.

- uint16_t offset

  Offset for long Characteristic value.
- sf_ble_char_multiple_read_req_t * p_mul_read_req

  Pointer to multiple read Characteristic request.
- uint16_t handles_cnt

  Characteristic handles count for multiple read.

sf_ble_char_multiple_read_rsp_t

sf_ble_char_multiple_read_rsp_t

Detailed description

Read multiple Characteristic response

Variables

- uint8_t * p_data[SF_BLE_MAX_MULTI_CHAR_READ_CNT]

  Pointer to Characteristic value.
- uint16_t data_len[SF_BLE_MAX_MULTI_CHAR_READ_CNT]

  Characteristic value length.

sf_ble_char_read_by_uuid_rsp_t

sf_ble_char_read_by_uuid_rsp_t

Detailed description

Read Characteristic by UUID response

Variables

- uint16_t handle[SF_BLE_MAX_CHAR_UUID_READ_CNT]

  Characteristic handles.
- uint8_t * p_data[SF_BLE_MAX_CHAR_UUID_READ_CNT]

  Pointer to Characteristic value.
- uint16_t data_len[SF_BLE_MAX_CHAR_UUID_READ_CNT]

  Characteristic value length.

sf_ble_char_read_by_handle_rsp_t

sf_ble_char_read_by_handle_rsp_t

Detailed description

Read Characteristic by handle response

Variables

- uint8_t * p_data

  Pointer to Characteristic value.

- uint16_t data_len

  Characteristic value length.

sf_ble_char_read_rsp_t

Detailed description

Read Characteristic response

Variables

p_char_read_by_handle_rsp

p_char_read_by_uuid_rsp

p_char_multiple_read_rsp

p_char_read_by_handle_rsp

`sf_ble_char_read_by_handle_rsp_t::p_char_read_by_handle_rsp`

Brief description

Response for read Characteristic by handle.

p_char_read_by_uuid_rsp

`sf_ble_char_read_by_uuid_rsp_t::p_char_read_by_uuid_rsp`

Brief description

Response for read Characteristic by UUID.

p_char_multiple_read_rsp

`sf_ble_char_multiple_read_rsp_t::p_char_multiple_read_rsp`

Brief description

Response for read multiple Characteristics.

sf_ble_char_write_req_t

sf_ble_char_write_req_t

Detailed description

Write Characteristic request

Variables

- sf_ble_char_write_t char_write_type

  Characteristic write type.

- uint16_t handle

  Characteristic value or descriptor handle.

- uint8_t * p_data

  Pointer to data.

- uint16_t offset

  Offset for long Characteristic value.
- uint16_t data_length

  Data length.
- sf_ble_execute_write_t auto_execute

  Automatic execute write flag.

sf_ble_gatt_notif_ind_event_data_t

sf_ble_gatt_notif_ind_event_data_t

Detailed description

Notification/Indication event data

Variables

- uint16_t handle

  Characteristic value or descriptor handle.
- uint8_t * p_data

  Pointer to data.
- uint16_t data_length

  Data length.

sf_ble_write_cmd_event_data_t

sf_ble_write_cmd_event_data_t

Detailed description

Write command event data

Variables

- uint16_t handle

  Characteristic value or descriptor handle.
- uint16_t offset

  Offset for long Characteristic value.
- sf_ble_write_response_t response_required

  Write response required or not.
- uint8_t * p_data

  Pointer to data.
- uint16_t data_length

  Data length.

sf_ble_ctrl_t

sf_ble_ctrl_t

Detailed description

BLE Framework control structure

Variables

- void * p_driver_handle

  Storage for information needed for each BLE device driver in the system.

sf_ble_cfg_t

sf_ble_cfg_t

Detailed description

BLE configuration information

Variables

- uint8_t bd_addr[SF_BLE_ADDR_LEN]

  BLE address.
- sf_ble_addr_type_t own_addr_type

  self address type
- uint8_t max_slaves

  Maximum slaves allowed to be connected.
- uint8_t update_bd_addr

  Set this to true to update bluetooth address during SF_BLE_Open.
- uint16_t scan_interval

  BLE scan interval for receiving advertisement.
- uint16_t scan_window

  Period of time during which advertising data is received at the scan interval.
- uint16_t disc_time

  Duration for which the device remain discoverable.
- uint16_t con_interval

  Interval for transmitting and receiving data periodically after connection establishment.
- uint16_t slave_latency

  Period of time during which data is transmitted and received at the connection interval.
- uint16_t sup_timeout

  Link loss time-out.
- void const * p_extend

  Instance specific configuration.

sf_ble_api_t

sf_ble_api_t

Detailed description

Framework API structure. Implementations will use the following API.

open

```
ssp_err_t(* sf_ble_api_t::open) (sf_ble_ctrl_t *const p_ctrl, const sf_ble_cfg_t *p_cfg)
```

Brief description

Initializes the interface for data transfers.

Detailed description

Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.

Table 27: Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to user-provided storage for the control block.
p_cfg	in	Pointer to BLE configuration structure.

Parameter p_ctrl

Definition: sf_ble_ctrl_t

BLE Framework control structure

Parameter p_cfg

Definition: sf_ble_cfg_t sf_ble_cfg_t *p_cfg

BLE configuration information

- sf_ble_cfg_t::bd_addr

  BLE address.

- sf_ble_cfg_t::sf_ble_addr_type_t

  self address type

  Enumerated as:

  – SF_BLE_ADDR_TYPE_PUBLIC

  – SF_BLE_ADDR_TYPE_RANDOM

- sf_ble_cfg_t::max_slaves

  Maximum slaves allowed to be connected.

- sf_ble_cfg_t::update_bd_addr

  Set this to true to update bluetooth address during SF_BLE_Open.

- sf_ble_cfg_t::scan_interval

  BLE scan interval for receiving advertisement.

- sf_ble_cfg_t::scan_window

  Period of time during which advertising data is received at the scan interval.

- sf_ble_cfg_t::disc_time

  Duration for which the device remain discoverable.

- sf_ble_cfg_t::con_interval

  Interval for transmitting and receiving data periodically after connection establishment.

- sf_ble_cfg_t::slave_latency

  Period of time during which data is transmitted and received at the connection interval.

- sf_ble_cfg_t::sup_timeout

  Link loss time-out.

- sf_ble_cfg_t::p_extend

  Instance specific configuration.

close

```
ssp_err_t(* sf_ble_api_t::close) (sf_ble_ctrl_t *const p_ctrl)
```

Brief description

De-initialize the interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.

Detailed description

Table 28: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.

Parameter p_ctrl

Definition: sf_ble_ctrl_t

BLE Framework control structure

infoGet

```
ssp_err_t(* sf_ble_api_t::infoGet) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, sf_ble_info_t *p_ble_info)
```

Brief description

Get BLE module information like chipset information and RSSI value.

Detailed description

Table 29: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.
p_handle	in	Pointer to connection handle
p_ble_info	out	Pointer to module information

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

Parameter p_handle

Definition: `sf_ble_conn_handle_t*p_handle`

BLE Connection Handle

Parameter p_ble_info

Definition: `sf_ble_info_t*p_ble_info`

BLE module information

- `sf_ble_info_t::chipset`

  Chipset information.

- `sf_ble_info_t::rssi`

  RSSI value.

provisionGet

```
ssp_err_t(* sf_ble_api_t::provisionGet) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_provisioning_t *p_ble_provisioning)
```

Brief description

Reads the current BLE Provisioning information.

Detailed description

Table 30: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.

Table 30: Parameters (Continued)

Name	Direction	Description
p_ble_provisioning	out	current provisioning information

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

Parameter p_ble_provisioning

Definition: `sf_ble_provisioning_t*p_ble_provisioning`

BLE provisioning information

- `sf_ble_provisioning_t::gap_name`

  GAP name.

- `sf_ble_provisioning_t::bcast_mode`

  broadcast mode

- `sf_ble_provisioning_t::bonding_mode`

  bonding mode

- `sf_ble_provisioning_t::sec_info`

  Security information.

- `sf_ble_provisioning_t::gap_role`

  GAP role (master/slave)

- `sf_ble_provisioning_t::p_ble_callback`

  GAP user event callback that runs in driver thread context. Application should make sure callback logic is as minimal as possible without any blocking calls

provisionSet

```
ssp_err_t(* sf_ble_api_t::provisionSet) (sf_ble_ctrl_t *const p_ctrl, const
sf_ble_provisioning_t *p_ble_provisioning)
```

Brief description

Provisions the BLE Driver and set bonding and security modes as provisioned.

Detailed description

Table 31: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.

Table 31: Parameters (Continued)

Name	Direction	Description
p_ble_provisioning	in	Pointer to BLE provisioning structure

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

Parameter p_ble_provisioning

Definition: `sf_ble_provisioning_t`sf_ble_provisioning_t *p_ble_provisioning

BLE provisioning information

- `sf_ble_provisioning_t::gap_name`

  GAP name.

- `sf_ble_provisioning_t::bcast_mode`

  broadcast mode

- `sf_ble_provisioning_t::bonding_mode`

  bonding mode

- `sf_ble_provisioning_t::sec_info`

  Security information.

- `sf_ble_provisioning_t::gap_role`

  GAP role (master/slave)

- `sf_ble_provisioning_t::p_ble_callback`

  GAP user event callback that runs in driver thread context. Application should make sure callback logic is as minimal as possible without any blocking calls

scan

```
ssp_err_t(* sf_ble_api_t::scan) (sf_ble_ctrl_t *const p_ctrl, sf_ble_scan_t
*p_scan, uint8_t *p_cnt, sf_ble_scan_info_t *p_scan_info)
```

Brief description

Scans for available BLE devices and return the list to the caller.

Detailed description

Table 32: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.

Table 32: Parameters (Continued)

Name	Direction	Description
p_scan	out	Pointer to scan information structure
p_cnt	inout	Pointer to number of BLE devices scanned
p_scan_info	in	Pointer to scan information structure

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

Parameter p_scan

Definition: `sf_ble_scan_t*p_scan`

BLE scan information

- `sf_ble_scan_t::addr_type`
  Address type of remote BLE device.
- `sf_ble_scan_t::bd_addr`
  Remote BLE address.
- `sf_ble_scan_t::data_len`
  Scan data length.
- `sf_ble_scan_t::data`
  Scan data.
- `sf_ble_scan_t::rssi`
  RSSI value.
- `sf_ble_scan_t::event_type`
  Advertising event type.

Parameter p_cnt

`uint8_t`

Parameter p_scan_info

Definition: `sf_ble_scan_info_t*p_scan_info`

BLE scan information structure

- `sf_ble_scan_info_t::total_scan_duration`
  BLE total scan duration in milliseconds.
- `sf_ble_scan_info_t::scan_mode`
  BLE scan mode.

- sf_ble_scan_info_t::discovery_type

  Specifies discovery type, Used only in active scan.

- sf_ble_scan_info_t::address_type

  BLE address type.

- sf_ble_scan_info_t::filt_policy

  Scan Filter policy.

- sf_ble_scan_info_t::duplicate_filt

  Duplicate filter configuration.

advertisementStart

```
ssp_err_t(* sf_ble_api_t::advertisementStart) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_adv_info_t *const p_advt_info)
```

Brief description

Make the device discoverable by broadcasting device information to remote devices.

Detailed description

Table 33: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.
p_advt_info	in	Pointer to advertisement information structure

Parameter p_ctrl

Definition: sf_ble_ctrl_t

BLE Framework control structure

Parameter p_advt_info

Definition: sf_ble_adv_info_t*const p_advt_info

BLE advertisement information

- sf_ble_adv_info_t::disc_mode

  Discovery mode.

- sf_ble_adv_info_t::conn_mode

  Connection mode.

- sf_ble_adv_info_t::adv_type

  Advertisement type.

- sf_ble_adv_info_t::adv_intv_min
  Minimum interval for advertising.
- sf_ble_adv_info_t::adv_intv_max
  Maximum interval for advertising.
- sf_ble_adv_info_t::own_addr_type
  Own address type.
- sf_ble_adv_info_t::direct_addr_type
  Direct connection address type.
- sf_ble_adv_info_t::direct_bd_addr
  Direct connection BLE address.
- sf_ble_adv_info_t::adv_chnl_map
  Advertising channel map.
- sf_ble_adv_info_t::adv_filt_policy
  Advertising filter policy.
- sf_ble_adv_info_t::scan_response_data
  Scan Response information, will be advertised only for active scan.
- sf_ble_adv_info_t::adv_data
  Advertising data.

advertisementStop

```
ssp_err_t(* sf_ble_api_t::advertisementStop) (sf_ble_ctrl_t *const p_ctrl)
```

Brief description

Stop the device from being discoverable.

Detailed description

Table 34: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.

Parameter p_ctrl

Definition: sf_ble_ctrl_t

BLE Framework control structure

whitelistAdd

```
ssp_err_t(* sf_ble_api_t::whitelistAdd) (sf_ble_ctrl_t *const p_ctrl, const
uint8_t *p_bd_addr)
```

Brief description

Add specified devices to whitelist.

Detailed description

Table 35: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.
p_bd_addr	in	Pointer to BLE address

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

Parameter p_bd_addr

`uint8_t`

whitelistDel

```
ssp_err_t(* sf_ble_api_t::whitelistDel) (sf_ble_ctrl_t *const p_ctrl, const
uint8_t *p_bd_addr)
```

Brief description

Remove specified devices from whitelist.

Detailed description

Table 36: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.
p_bd_addr	in	Pointer to BLE address

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

Parameter p_bd_addr

uint8_t

bondingStart

```
ssp_err_t(* sf_ble_api_t::bondingStart) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, const uint8_t *p_bd_addr, sf_ble_bonding_start_t
*p_bonding_start)
```

Brief description

Initiate bonding process with remote BLE device and exchange security keys if enabled.

Detailed description

Table 37: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.
p_handle	in	Pointer to connection handle.
p_bd_addr	in	Pointer to BLE address

Parameter p_ctrl

Definition: sf_ble_ctrl_t

BLE Framework control structure

Parameter p_handle

Definition: sf_ble_conn_handle_t*p_handle

BLE Connection Handle

Parameter p_bd_addr

uint8_t

bondingResponse

```
ssp_err_t(* sf_ble_api_t::bondingResponse) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, const uint8_t *p_bd_addr,
sf_ble_bonding_response_t *p_bonding_resp)
```

Brief description

Respond to the bonding request from the remote BLE device. Send bonding response on reception of bonding indication.

Detailed description

Table 38: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.
p_handle	in	Pointer to connection handle
p_bd_addr	in	Pointer to BLE address
p_bonding_resp	in	Pointer to Bonding address

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

Parameter p_handle

Definition: `sf_ble_conn_handle_t*p_handle`

BLE Connection Handle

Parameter p_bd_addr

`uint8_t`

Parameter p_bonding_resp

Definition: `sf_ble_bonding_response_t*p_bonding_resp`

Bonding Response Parameter

- `sf_ble_bonding_response_t::accept`
  accept or reject bonding
- `sf_ble_bonding_response_t::io_cap`
  IO capabilities.
- `sf_ble_bonding_response_t::max_key_size`
  Max key size.
- `sf_ble_bonding_response_t::ikeys`
  Initiator key distribution.
- `sf_ble_bonding_response_t::rkeys`
  Responder key distribution.

startEncryption

```
ssp_err_t(* sf_ble_api_t::startEncryption) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_sm_enc_info_t const *p_enc_info)
```

Brief description

Start encryption over a connection.

Detailed description

Table 39: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.
p_enc_info	in	information for starting encryption

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

Parameter p_enc_info

Definition: `sf_ble_sm_enc_info_tconst *p_enc_info`

BLE Start Encryption information

- `sf_ble_sm_enc_info_t::conn_idx`
  connection index
- `sf_ble_sm_enc_info_t::ediv`
  BLE Security EDIV.
- `sf_ble_sm_enc_info_t::rand_num`
  Random number for security.
- `sf_ble_sm_enc_info_t::ltk_key`
  BLE long term security key.

connect

```
ssp_err_t(* sf_ble_api_t::connect) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_connection_t const *const p_conn, sf_ble_conn_handle_t *p_handle)
```

Brief description

Connect to a remote BLE device.

Detailed description

Table 40: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.

Table 40: Parameters (Continued)

Name	Direction	Description
p_conn	in	Pointer to connection information
p_handle	out	Pointer to connection handle

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

Parameter p_conn

Definition: `sf_ble_connection_tconst *const p_conn`

BLE connection information

- `sf_ble_connection_t::init_filt_type`
  Connection filter type.
- `sf_ble_connection_t::addr_type`
  BLE address type.
- `sf_ble_connection_t::bd_addr`
  BLE address.

Parameter p_handle

Definition: `sf_ble_conn_handle_t*p_handle`

BLE Connection Handle

listen

```
ssp_err_t(* sf_ble_api_t::listen) (sf_ble_ctrl_t *const p_ctrl)
```

Brief description

Listen for connection request from remote device.

Detailed description

Table 41: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

authorization

```
ssp_err_t(* sf_ble_api_t::authorization) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle)
```

Brief description

Indicates that the specified remote device has been authorized by user.

Detailed description

Table 42: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.
p_handle	in	Pointer to connection handle.

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

Parameter p_handle

Definition: `sf_ble_conn_handle_t*p_handle`

BLE Connection Handle

disconnect

```
ssp_err_t(* sf_ble_api_t::disconnect) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle)
```

Brief description

Terminate connection with remote BLE device.

Detailed description

Table 43: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.
p_handle	in	Pointer to connection handle

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

Parameter p_handle

Definition: sf_ble_conn_handle_t*p_handle

BLE Connection Handle

setTxPower

```
ssp_err_t(* sf_ble_api_t::setTxPower) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *const p_handle, sf_ble_set_tx_pwr_info_t *p_tx_power_info)
```

Brief description

Sets the transmit power for the procedure specified by the connection handle.

Detailed description

Table 44: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.
p_handle	in	Pointer to connection handle. If transmit power is to be set before advertisement or connecting to remote device then pass connection handle as SF_BLE_TX_POWER_CONNECTION_HANDLE value else pass connection handle of remote device
p_tx_power_info	in	Pointer to TX power information

Parameter p_ctrl

Definition: sf_ble_ctrl_t

BLE Framework control structure

Parameter p_handle

Definition: sf_ble_conn_handle_t*const p_handle

BLE Connection Handle

Parameter p_tx_power_info

Definition: sf_ble_set_tx_pwr_info_t*p_tx_power_info

BLE set TX power information structure

- sf_ble_set_tx_pwr_info_t::power_lvl

  TX power level in dBm.

- sf_ble_set_tx_pwr_info_t::state

  Operating state to set the transmit power level.

gattAddCustomProfiles

```
ssp_err_t(* sf_ble_api_t::gattAddCustomProfiles) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_svc_attribute_t *p_svc_attr, uint32_t svc_attr_len,
sf_ble_char_attribute_t *p_char_attr, uint32_t char_attr_len)
```

Brief description

Add Custom Profile to GATT Database.

Detailed description

This function is called with list of service and characteristics which is to be added to GATT database. Services and characteristics which were previously added will be removed and newly passed services and characteristics will be added.

If services and characteristics were previously added and now those services and characteristics are to be removed and no new services and characteristics are to be added , then call this API with service and characteristics length as zero

Table 45: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure
p_svc_attr	in	List of Services to add
svc_attr_len	in	No of elements in services list
p_char_attr	in	List of Characteristics to add
char_attr_len	in	No of elements in Characteristics list

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

Parameter p_svc_attr

Definition: `sf_ble_svc_attribute_t*p_svc_attr`

BLE GATT Service Attributes

- `sf_ble_svc_attribute_t::attr_handle`

  Service Handle.
- `sf_ble_svc_attribute_t::attr_type`

  Attribute type such as SF_BLE_GATT_PRI_SERVICE, SF_BLE_GATT_INCLUDE_SERVICE.
- `sf_ble_svc_attribute_t::parent_svc_handle`

  parent_svc_handle is service handle of parent service which is already registered
- `sf_ble_svc_attribute_t::p_attr_value`

  Service UUID Pointer.

- sf_ble_svc_attribute_t::attr_value_len

  UUID Length.

Parameter svc_attr_len

uint32_t

Parameter p_char_attr

Definition: sf_ble_char_attribute_t*p_char_attr

BLE GATT Characteristics Attributes

- sf_ble_char_attribute_t::p_service

  Service to which this characteristics belongs.
- sf_ble_char_attribute_t::attr_uuid

  UUID of characteristics value.
- sf_ble_char_attribute_t::attr_declare_handle

  Characteristics handle.
- sf_ble_char_attribute_t::attr_declare_type

  Characteristics declare type SF_BLE_GATT_CHAR_DECLARE.
- sf_ble_char_attribute_t::attr_value_handle

  Characteristics value handle.
- sf_ble_char_attribute_t::attr_perm

  Characteristics permission.
- sf_ble_char_attribute_t::attr_properties

  Characteristics properties.
- sf_ble_char_attribute_t::p_attr_value

  Characteristics value data.
- sf_ble_char_attribute_t::attr_value_len

  Characteristics value data length.

Parameter char_attr_len

uint32_t

gattServiceDiscovery

```
ssp_err_t(* sf_ble_api_t::gattServiceDiscovery) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, sf_ble_service_discovery_req_t const *const
p_sf_ble_svc_dscv_req, sf_ble_service_discovery_rsp_t *const
p_sf_ble_svc_dscv_rsp, uint32_t *const p_rsp_cnt)
```

Brief description

Perform service discovery used by GATT client.Perform service discovery on remote GATT server and get results.

Detailed description

Table 46: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure
p_handle	in	Connection handle
p_sf_ble_svc_dscv_req	in	Pointer to service discovery request
p_sf_ble_svc_dscv_rsp	out	Pointer to service discovery response
p_rsp_cnt	inout	Input Size specifying maximum number of service discovery results which can be stored in response, output specifying number of service discovery results stored in response

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

Parameter p_handle

Definition: `sf_ble_conn_handle_t*p_handle`

BLE Connection Handle

Parameter p_sf_ble_svc_dscv_req

Definition: `sf_ble_service_discovery_req_tconst *const p_sf_ble_svc_dscv_req`

Service discovery request

- `sf_ble_service_discovery_req_t::uuid`
  Service UUID.
- `sf_ble_service_discovery_req_t::start_handle`
  Discovery start handle.
- `sf_ble_service_discovery_req_t::end_handle`
  Discovery end handle.
- `sf_ble_service_discovery_req_t::discovery_type`
  Service discovery type.

Parameter p_sf_ble_svc_dscv_rsp

Definition: `sf_ble_service_discovery_rsp_t*const p_sf_ble_svc_dscv_rsp`

Service discovery result

- sf_ble_service_discovery_rsp_t::service_handle

  Service handle.

- sf_ble_service_discovery_rsp_t::uuid

  Service UUID.

- sf_ble_service_discovery_rsp_t::start_handle

  Start handle of sub-attributes handle range.

- sf_ble_service_discovery_rsp_t::end_handle

  End handle of sub-attributes handle range.

Parameter p_rsp_cnt

uint32_t

gattCharDiscovery

```
ssp_err_t(* sf_ble_api_t::gattCharDiscovery) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, sf_ble_char_discovery_req_t const *const
p_sf_ble_char_dscv_req, sf_ble_char_discovery_rsp_t *const
p_sf_ble_char_dscv_rsp, uint32_t *const p_rsp_cnt)
```

Brief description

Perform characteristics discovery used by GATT client.Perform characteristics discovery on remote GATT server and get results.

Detailed description

Table 47: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure
p_handle	in	Connection handle
p_sf_ble_char_dscv_req	in	Pointer to characteristics discovery request
p_sf_ble_char_dscv_rsp	out	Pointer to characteristics discovery response
p_rsp_cnt	inout	Input Size specifying maximum number of characteristics discovery results which can be stored in response, output specifying number of characteristics discovery results stored in response

Parameter p_ctrl

Definition: sf_ble_ctrl_t

BLE Framework control structure

Parameter p_handle

Definition: sf_ble_conn_handle_t*p_handle

BLE Connection Handle

Parameter p_sf_ble_char_dscv_req

Definition: sf_ble_char_discovery_req_tconst *const p_sf_ble_char_dscv_req

Characteristic discovery request

- sf_ble_char_discovery_req_t::uuid

  Characteristic UUID.

- sf_ble_char_discovery_req_t::start_handle

  Discovery start handle.

- sf_ble_char_discovery_req_t::end_handle

  Discovery end handle.

- sf_ble_char_discovery_req_t::discovery_type

  Characteristic discovery type.

Parameter p_sf_ble_char_dscv_rsp

Definition: sf_ble_char_discovery_rsp_t*const p_sf_ble_char_dscv_rsp

Characteristic discovery result

- sf_ble_char_discovery_rsp_t::char_handle

  Characteristic handle.

- sf_ble_char_discovery_rsp_t::uuid

  Characteristic UUID.

- sf_ble_char_discovery_rsp_t::value_handle

  Characteristic value handle.

- sf_ble_char_discovery_rsp_t::property

  Characteristic property.

Parameter p_rsp_cnt

uint32_t

gattCharDescDiscovery

```
ssp_err_t(* sf_ble_api_t::gattCharDescDiscovery) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, uint16_t start_handle, uint16_t end_handle,
sf_ble_char_desc_discovery_rsp_t *const p_sf_ble_chardesc_dscv_rsp, uint32_t
*const p_rsp_cnt)
```

Brief description

Perform characteristics descriptor discovery used by GATT client.

Detailed description

Table 48: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure
p_handle	in	Connection handle
start_handle	in	Start handle from set of handle ranges to be used in discovery
end_handle	in	End handle from set of handle ranges to be used in discovery
p_sf_ble_chardesc_dscv_rsp	out	Pointer to characteristics descriptor discovery response
p_rsp_cnt	inout	Input Size specifying maximum number of characteristics descriptor discovery results which can be stored in response, output specifying number of characteristics descriptor discovery results stored in response

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

Parameter p_handle

Definition: `sf_ble_conn_handle_t*p_handle`

BLE Connection Handle

Parameter start_handle

`uint16_t`

Parameter end_handle

`uint16_t`

Parameter p_sf_ble_chardesc_dscv_rsp

Definition: `sf_ble_char_desc_discovery_rsp_t*const p_sf_ble_chardesc_dscv_rsp`

Characteristic descriptor discovery result

- `sf_ble_char_desc_discovery_rsp_t::desc_handle`
  Characteristic descriptor handle.
- `sf_ble_char_desc_discovery_rsp_t::uuid`
  Characteristic descriptor UUID.

Parameter p_rsp_cnt

`uint32_t`

gattCharRead

```
ssp_err_t(* sf_ble_api_t::gattCharRead) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, sf_ble_char_read_req_t const *const
p_char_read_req, sf_ble_char_read_rsp_t *const p_char_read_rsp)
```

Brief description

Perform read characteristic used by GATT client.Read characteristic value from remote GATT server.

Detailed description

Table 49: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure
p_handle	in	Connection handle
p_char_read_req	in	Pointer to characteristic read request
p_char_read_rsp	out	Pointer to characteristic read response

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

Parameter p_handle

Definition: `sf_ble_conn_handle_t*p_handle`

BLE Connection Handle

Parameter p_char_read_req

Definition: `sf_ble_char_read_req_tconst *const p_char_read_req`

Read Characteristic request

- `sf_ble_char_read_req_t::char_read_type`

  Characteristic read type.

- `sf_ble_char_read_req_t::handle`

  Characteristic value or descriptor handle.

- `sf_ble_char_read_req_t::uuid`

  Characteristic UUID.

- sf_ble_char_read_req_t::offset

  Offset for long Characteristic value.

- sf_ble_char_read_req_t::p_mul_read_req

  Pointer to multiple read Characteristic request.

- sf_ble_char_read_req_t::handles_cnt

  Characteristic handles count for multiple read.

Parameter p_char_read_rsp

Definition: sf_ble_char_read_rsp_t*const p_char_read_rsp

gattCharWrite

```
ssp_err_t(* sf_ble_api_t::gattCharWrite) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, sf_ble_char_write_req_t const *const
p_char_write_req)
```

Brief description

Perform write characteristic used by GATT client.Write characteristic value on remote GATT server.

Detailed description

Table 50: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure
p_handle	in	Connection handle
p_char_write_req	in	Pointer to characteristic write request

Parameter p_ctrl

Definition: sf_ble_ctrl_t

BLE Framework control structure

Parameter p_handle

Definition: sf_ble_conn_handle_t*p_handle

BLE Connection Handle

Parameter p_char_write_req

Definition: sf_ble_char_write_req_tconst *const p_char_write_req

Write Characteristic request

- sf_ble_char_write_req_t::char_write_type

  Characteristic write type.

- sf_ble_char_write_req_t::handle

  Characteristic value or descriptor handle.
- sf_ble_char_write_req_t::p_data

  Pointer to data.
- sf_ble_char_write_req_t::offset

  Offset for long Characteristic value.
- sf_ble_char_write_req_t::data_length

  Data length.
- sf_ble_char_write_req_t::auto_execute

  Automatic execute write flag.

gattCharExecuteWrite

```
ssp_err_t(* sf_ble_api_t::gattCharExecuteWrite) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, sf_ble_execute_write_t execute_flag)
```

Brief description

Perform execute write on all pending write operations, used by GATT client.Execute or cancel all prepared writes of characteristics on remote GATT server.

Detailed description

Table 51: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure
p_handle	in	Connection handle
execute_flag	in	Flag specifying whether to execute or cancel pending writes

Parameter p_ctrl

Definition: sf_ble_ctrl_t

BLE Framework control structure

Parameter p_handle

Definition: sf_ble_conn_handle_t*p_handle

BLE Connection Handle

Parameter execute_flag

Definition: sf_ble_execute_write_texecute_flag

Characteristic execute write flag

gattCharWriteLocal

```
ssp_err_t(* sf_ble_api_t::gattCharWriteLocal) (sf_ble_ctrl_t *const p_ctrl,
uint16_t char_handle, uint16_t data_length, uint8_t *const p_data)
```

Brief description

Perform local characteristic write used by GATT server.Writes local characteristic value on GATT server.

Detailed description

Table 52: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure
char_handle	in	Characteristic handle
data_length	in	Length of data to write
p_data	in	Pointer to data

Parameter p_ctrl

Definition: sf_ble_ctrl_t

BLE Framework control structure

Parameter char_handle

uint16_t

Parameter data_length

uint16_t

Parameter p_data

uint8_t

gattSendNotify

```
ssp_err_t(* sf_ble_api_t::gattSendNotify) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, uint16_t char_handle)
```

Brief description

Send notification to GATT client, used by GATT server.

Detailed description

Table 53: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure

Table 53: Parameters (Continued)

Name	Direction	Description
p_handle	in	Connection handle
char_handle	in	Characteristic handle whose value will be notified

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

Parameter p_handle

Definition: `sf_ble_conn_handle_t*p_handle`

BLE Connection Handle

Parameter char_handle

`uint16_t`

gattSendIndicate

```
ssp_err_t(* sf_ble_api_t::gattSendIndicate) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, uint16_t char_handle)
```

Brief description

Send indication to GATT client, used by GATT server.Send indication to remote GATT client.

Detailed description

Table 54: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure
p_handle	in	Connection handle
char_handle	in	Characteristic handle whose value will be indicated

Parameter p_ctrl

Definition: `sf_ble_ctrl_t`

BLE Framework control structure

Parameter p_handle

Definition: `sf_ble_conn_handle_t*p_handle`

BLE Connection Handle

Parameter char_handle

uint16_t

gattWriteResponse

```
ssp_err_t(* sf_ble_api_t::gattWriteResponse) (sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, uint16_t handle, sf_ble_attribute_error_code_t
error_code)
```

Brief description

Send response to write operation received by GATT client, used by GATT server.Send response for write request received from remote GATT client.

Detailed description

Table 55: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure
p_handle	in	Connection handle
handle	in	Characteristic handle used for write operation
error_code	in	Characteristic write operation error code to be sent in response

Parameter p_ctrl

Definition: sf_ble_ctrl_t

BLE Framework control structure

Parameter p_handle

Definition: sf_ble_conn_handle_t*p_handle

BLE Connection Handle

Parameter handle

uint16_t

Parameter error_code

versionGet

```
ssp_err_t(* sf_ble_api_t::versionGet) (ssp_version_t *const p_version)
```

Brief description

Gets version and stores it in provided pointer p_version.

Detailed description

Table 56: Parameters

Name	Direction	Description
p_version	out	pointer to memory location to return version number

Parameter p_version

sf_ble_instance_t

sf_ble_instance_t

Detailed description

BLE instance

Variables

- sf_ble_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_ble_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_ble_api_t const * p_api

  Pointer to the API structure for this instance.

6.6.2 SF BLE On-Board Profile Framework Interface

RTOS-integrated SF BLE On-Board Profile Framework Interface.

6.6.2.1 Summary

This SSP Interface provides access to the ThreadX-aware SF BLE On-Board Profile Framework.

6.6.2.2 Interface API

sf_ble_onboard_profile_api_t

Function name	Description
.open	Initializes the interface for data transfers. Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.

Function name	Description
.close	De-initialize the interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.
.onbpEnable	Enable On-Board Profile Enables On-Board profile on given connection handle with specified security type. Registers user callback for the profile.
.onbpDisable	Disable On-Board Profile Disables On-Board profile on given connection handle and unregisters user callback.
.onbpServerWriteData	Update Server Local Database Update Local GATT database of Profile Server.
.onbpServerSendNotification	Send notification from Server Sends Notification which will be data specific to On-Board Profile to Client.
.onbpServerSendIndication	Send Indication from Server Sends Indication which contains profile specific data to client.
.onbpClientWriteCCCD	Writes CCCD configuration in Server This API writes CCCD configuration of Server and which enables or disables notification or indication from server.
.onbpClientWriteChar	Writes GATT characteristics with data passed Writes the GATT characteristics in the Server with the data passed.
.onbpClientReadChar	Reads GATT characteristics from Server Reads GATT characteristics from the Server for the profile specified.
.versionGet	Gets version and stores it in provided pointer p_version.

6.6.2.3 Data structures

- sf_ble_onboard_profile_cccd_changed_t
- sf_ble_attr_info_t
- sf_ble_long_attr_info_t
- sf_ble_onboard_profile_cfg_t
- sf_ble_onboard_profile_ctrl_t
- sf_ble_onboard_profile_instance_t

6.6.2.4 Enumerations

- sf_onbp_t

- sf_ble_prf_sec_t
- sf_ble_onbp_char_t
- sf_ble_cccd_val_t

6.6.2.5 Typedefs

- sf_ble_profile_callback_t

6.6.2.6 Defines

- #define SF_BLE_ONBOARD_PROFILE_API_VERSION_MAJOR

  Initial value: `(1U)`

  SSP BSP Include files Framework Include files Major Version of the API defined in this file
- #define SF_BLE_ONBOARD_PROFILE_API_VERSION_MINOR

  Initial value: `(1U)`

  Minor Version of the API defined in this file
- #define SF_BLE_ATTM_MAX_VALUE

  Initial value: `(0x18)`

  GATT Attribute Length

6.6.2.7 API Data

sf_onbp_t

`sf_onbp_t`

Detailed description

BLE On-Board Profile types

Enumerated values

Name	Description
SF_ONBP_ANS_SERVER	Alert Notification Service Server.
SF_ONBP_ANS_CLIENT	Alert Notification Service Client.
SF_ONBP_BAS_SERVER	Battery Service Server.
SF_ONBP_BPS_SERVER	Blood Pressure Service Server.
SF_ONBP_BPS_CLIENT	Blood Pressure Service Client.

Name	Description
SF_ONBP_CTS_SERVER	Current Time Service Server.
SF_ONBP_DIS_SERVER	Device Information Service Server.
SF_ONBP_FMP_SERVER	Find Me Service Server.
SF_ONBP_FMP_CLIENT	Find Me Service Client.
SF_ONBP_HTP_SERVER	Health Thermometer Service Server.
SF_ONBP_HTP_CLIENT	Health Thermometer Service Client.
SF_ONBP_HRS_SERVER	Heart Rate Service Server.
SF_ONBP_HRS_CLIENT	Heart Rate Service Client.
SF_ONBP_HID_SERVER	HID Over GATT Service Server.
SF_ONBP_HID_BHOST_CLIENT	HID Over GATT Boot Host Service Client.
SF_ONBP_HID_RHOST_CLIENT	HID Over GATT Report Host Service Client.
SF_ONBP_IMA_SERVER	Immediate Alert Service Server.
SF_ONBP_LLS_SERVER	Link Loss Service Server.
SF_ONBP_LLS_CLIENT	Link Loss Service Client.
SF_ONBP_NDCS_SERVER	Next Daylight Savings Change Service Server.
SF_ONBP_PAPS_SERVER	Phone Alert Service Server.
SF_ONBP_PAPS_CLIENT	Phone Alert Service Client.
SF_ONBP_PXP_SERVER	Proximity Service Server.
SF_ONBP_PXP_CLIENT	Proximity Service Client.
SF_ONBP_RTUS_SERVER	Reference Time Update Service Server.
SF_ONBP_SCPS_SERVER	Scan Parameter Service Server.
SF_ONBP_SCPS_CLIENT	Scan Parameter Service Client.
SF_ONBP_TIP_SERVER	Time Information Service Server.
SF_ONBP_TIP_CLIENT	Time Information Service Client.

Name	Description
SF_ONBP_TXP_SERVER	Transmit Power Service Server.

sf_ble_prf_sec_t

`sf_ble_prf_sec_t`

Detailed description

BLE Profile Security type

Enumerated values

Name	Description
SF_BLE_PRF_SEC_NONE	No Security.
SF_BLE_PRF_SEC_UNAUTH	Unauthenticated Pairing.
SF_BLE_PRF_SEC_AUTH	Authenticated pairing.
SF_BLE_PRF_SEC_AUTZ	Requires Authorized.
SF_BLE_PRF_SEC_ENC	Encrypted communication.

sf_ble_onbp_char_t

`sf_ble_onbp_char_t`

Detailed description

On-Board Profile GATT Characteristics Code

Enumerated values

Name	Description
SF_BLE_ONBP_CHAR_HRP_HRCP	Heart Rate Control Point Characteristics (Property: Write), Refer sf_ble_prf_hrp_api_hrcp_t. Heart Rate Profile
SF_BLE_ONBP_CHAR_HRP_CCCD_NTF_HRMEAS	Heart Rate Measurement CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_hrp_api_hrmeas_t.
SF_BLE_ONBP_CHAR_HRP_BSL	Heart Rate Body Sensor Location Characteristics (Property: Read)
SF_BLE_ONBP_CHAR_ANP_SUP_NEW_ALERT	Supported New Alert Category Characteristics (Property: Read) Alert Notification Profile

Name	Description
SF_BLE_ONBP_CHAR_ANP_CCCD_NTF_NEW_ALERT	New Alert CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_anp_api_new_alert_ntf_t.
SF_BLE_ONBP_CHAR_ANP_SUP_UNREAD_ALERT	Supported Unread Alert Category Characteristics (Property: Read)
SF_BLE_ONBP_CHAR_ANP_CCCD_NTF_UNREAD_ALERT_STATUS	Unread Alert Status CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_anp_api_unread_alert_ntf_t.
SF_BLE_ONBP_CHAR_ANP_ALERT_NOTIFICATION_CTRL_POINT	Alert Notification Control Point (Property: Write), Refer sf_ble_anp_ancp_t.
SF_BLE_ONBP_CHAR_DIS_MANUF	Device Information Service Manufacturer Name String (Property: Read) Device Information Service Profile
SF_BLE_ONBP_CHAR_DIS_MODEL	Device Information Service Model Number String (Property: Read)
SF_BLE_ONBP_CHAR_DIS_SERNB	Device Information Service Serial number String (Property: Read)
SF_BLE_ONBP_CHAR_DIS_HWREV	Device Information Service HW Revision String (Property: Read)
SF_BLE_ONBP_CHAR_DIS_FWREV	Device Information Service Fw Revision String (Property: Read)
SF_BLE_ONBP_CHAR_DIS_SWREV	Device Information Service SW Revision String (Property: Read)
SF_BLE_ONBP_CHAR_DIS_SYSID	Device Information Service system ID (Property: Read)
SF_BLE_ONBP_CHAR_DIS_IEEE	Device Information Service IEEE Certification (Property: Read)
SF_BLE_ONBP_CHAR_DIS_PNPID	Device Information PNPID, Used in services like HOGP (Property: Read)
SF_BLE_ONBP_CHAR_IAS_ALERT_LEVEL	Alert Level (Property: Write), Refer sf_ble_prf_ias_alert_type_t. Immediate Alert Service Profile
SF_BLE_ONBP_CHAR_SCPS_CCCD_NTF_SCAN_REFRESH	Scan Refresh CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_prf_scps_scan_refresh_t. Scan Parameters Service Profile

Name	Description
SF_BLE_ONBP_CHAR_SCPS_SCAN_INTERVAL_WINDOW	Scan Interval Window (Property: Write), Refer sf_ble_prf_scps_scan_intv_t.
SF_BLE_ONBP_CHAR_CTS_CCCD_NTF_CURRENT_TIME	Current Time CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_cts_curr_time_ntf_t. Current Time Service Profile
SF_BLE_ONBP_CHAR_CTS_CCCD_NTF_CURRENT_TIME_VALUE	Used to read data for current time through read characteristics.
SF_BLE_ONBP_CHAR_CTS_LOCAL_TIME_INFORMATION	Local Time Information (Property: Read)
SF_BLE_ONBP_CHAR_CTS_REFERENCE_TIME_INFORMATION	Reference Time Information (Property: Read)
SF_BLE_ONBP_CHAR_NDCS_TIME_WITH_DST	Time with DST (Property: Read) Next DST Change Service Profile
SF_BLE_ONBP_CHAR_TIME_UPDATE_CONTROL_POINT	Time Update Control Point (Property: Write), Refer sf_ble_prf_rtus_time_updt_state_t. Reference Time Update Service Profile
SF_BLE_ONBP_CHAR_TIME_UPDATE_STATE	Time Update State (Property: Read)
SF_BLE_ONBP_CHAR_CCCD_NTF_CURRENT_TIME	Current Time CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_bas_battery_lvl_ntf_t. Time Information Service Profile
SF_BLE_ONBP_CHAR_CCCD_NTF_BATTERY_LEVEL	Battery Level (Property: NTF, Read, Write), Refer sf_ble_prf_bas_battery_lvl_t. Battery Service Profile
SF_BLE_ONBP_CHAR_PXPM_SET_ALERT	PXP Alert char. Alert Level Service Profile
SF_BLE_ONBP_CHAR_PXPM_GET_ALERT_LVL	PXP Get Alert Level.
SF_BLE_ONBP_CHAR_PXPM_GET_TX_POWER_LVL	PXP Get TX Power Level.
SF_BLE_ONBP_CHAR_HTP_TEMP_MEAS_IND	HTPC temp. measurement indication CCCD Characteristics (Property: IND, Read, Write), Refer. Health Thermometer Profile
SF_BLE_ONBP_CHAR_HTP_INTERM_TEMP_NTF	HTPC intermediate temp. notification CCCD Characteristics (Property: NTF, Read, Write), Refer.
SF_BLE_ONBP_CHAR_HTP_MEAS_INTV_IND	HTPC temp. measurement interval indication CCCD Characteristics (Property: IND, Read, Write), Refer.

Name	Description
SF_BLE_ONBP_CHAR_HTP_TEMP_TYPE	HTPC temp. temperature type(Property: Read)
SF_BLE_ONBP_CHAR_HTP_MEAS_INTV	HTPC temp. measurement interval.
SF_BLE_ONBP_CHAR_HTP_MEAS_INTV_RANGE	HTPC temp. measurement interval valid range.
SF_BLE_ONBP_CHAR_CCCD_NTF_REPORT_INPUT	Report Input CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_prf_hid_report_desc_t. HID Over GATT Service Profile
SF_BLE_ONBP_CHAR_CCCD_NTF_KB_INPUT_REPORT	Keyboard Input Report CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_prf_hid_report_desc_t.
SF_BLE_ONBP_CHAR_CCCD_NTF_KB_INPUT_REPORT_VALUE	Used to read Keyboard Input record value through read characteristics.
SF_BLE_ONBP_CHAR_CCCD_NTF_KB_OUTPUT_REPORT_VALUE	Used to read Keyboard Output record value through read characteristics.
SF_BLE_ONBP_CHAR_CCCD_NTF_MOUSE_INPUT_REPORT	Mouse Input Report CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_prf_hid_report_desc_t.
SF_BLE_ONBP_CHAR_CCCD_NTF_MOUSE_INPUT_REPORT_VALUE	Used to read Mouse Input record value through read characteristics.
SF_BLE_ONBP_CHAR_PROTOCOL_MODE	HID boot host protocol mode (Property: Read, Write), Refer sf_ble_prf_hid_protocol_mode_t.
SF_BLE_ONBP_CHAR_CONTROL_POINT	HID report host write control point (Property: Read, Write), Refer sf_ble_prf_hid_ctrl_point_val_t.
SF_BLE_ONBP_CHAR_BLP_CCCD_NTF_INTERM_CUFPRS	Read Cuff Pressure measurement CCCD Characteristics (Property: NTF, Read, Write), Refer sf_ble_blp_meas_info_t. Blood Pressure profile
SF_BLE_ONBP_CHAR_BLP_CCCD_IND_BLDPRS_MEAS	Blood Pressure Measurement CCCD Characteristics (Property: IND, Read, Write), Refer sf_ble_blp_meas_info_t.
SF_BLE_ONBP_CHAR_BLP_RD_BLS_BF	Read Blood Pressure Feature.
SF_BLE_ONBP_CHAR_CCCD_NTF_ALERT_STATUS	Alert Status (Property: NTF, Read, Write), Refer sf_ble_prf_alert_status. Phone Alert Status Service Profile
SF_BLE_ONBP_CHAR_CCCD_NTF_ALERT_STATUS_VALUE	Used to read Alert Status value through read characteristics.

Name	Description
SF_BLE_ONBP_CHAR_CCCD_NTF_RINGER_SETTING	Ringer Settings (Property: NTF, Read, Write), Refer sf_ble_prf_ringer_setting_t.
SF_BLE_ONBP_CHAR_CCCD_NTF_RINGER_SETTING_VALUE	Used to read Ringer Settings value through read characteristics.
SF_BLE_ONBP_CHAR_RINGER_CONTROL_POINT	PAPS Ringer control point write characteristics.

sf_ble_cccd_val_t

```
sf_ble_cccd_val_t
```

Detailed description

Value for BLE CCCD Configuration

Enumerated values

Name	Description
SF_BLE_CCCD_VAL_STOP_NTFIND	Stop Notification Indication.
SF_BLE_CCCD_VAL_START_NTF	Start Notification.
SF_BLE_CCCD_VAL_START_IND	Start Indication.

sf_ble_profile_callback_t

```
typedef void(* sf_ble_profile_callback_t) (sf_ble_event_info_t *ev)
```

Detailed description

User callback type for profile

6.6.2.8 API Structures

sf_ble_onboard_profile_cccd_changed_t

sf_ble_onboard_profile_cccd_changed_t

Detailed description

BLE Profile CCCD Changed indication data

Variables

- sf_ble_conn_handle_t conn_handle

  Connection handle.

- sf_ble_onbp_char_t char_code

  CCCD type that has been changed by the remote node.

- sf_ble_cccd_val_t cccd_val

  CCCD value.
- uint8_t inst_idx

  Instance index, Applicable for HOGP.

sf_ble_attr_info_t

sf_ble_attr_info_t

Detailed description

BLE Profile Read Char data

Variables

- uint8_t len

  data length
- uint8_t data[SF_BLE_ATTM_MAX_VALUE]

  data

sf_ble_long_attr_info_t

sf_ble_long_attr_info_t

Detailed description

BLE Profile Read Char data (Long type)

Variables

- uint8_t val_len

  size of the value data
- uint8_t reserved

  Reserved.
- uint16_t attr_hdl

  Attribute handle.
- uint8_t value[SF_BLE_ATTM_MAX_VALUE]

  actual value pairs

sf_ble_onboard_profile_cfg_t

sf_ble_onboard_profile_cfg_t

Detailed description

BLE On-Board Profile configuration information

Variables

- sf_ble_instance_t const * p_low_lvl_ble

  Low level BLE Interface.

- void * p_extend

  Extended configuration.

sf_ble_onboard_profile_ctrl_t

sf_ble_onboard_profile_ctrl_t

Detailed description

BLE On-Board Profile Framework control structure

Variables

- sf_ble_instance_t * p_low_lvl_ble

  Low level BLE Framework information needed by On-Board Profile.

sf_ble_onboard_profile_api_t

sf_ble_onboard_profile_api_t

Detailed description

BLE Configuration, Control and API structures Framework API structure. Implementations will use the following API.

open

```
ssp_err_t(* sf_ble_onboard_profile_api_t::open) (sf_ble_onboard_profile_ctrl_t
*const p_ctrl, const sf_ble_onboard_profile_cfg_t *p_cfg)
```

Brief description

Initializes the interface for data transfers.

Detailed description

Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.

Table 57: Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to user-provided storage for the control block.
p_cfg	in	Pointer to BLE configuration structure.

Parameter p_ctrl

Definition: `sf_ble_onboard_profile_ctrl_t`

BLE On-Board Profile Framework control structure

Parameter p_cfg

Definition: `sf_ble_onboard_profile_cfg_t sf_ble_onboard_profile_cfg_t *p_cfg`

BLE On-Board Profile configuration information

- sf_ble_onboard_profile_cfg_t::sf_ble_instance_t

  Low level BLE Interface.

- sf_ble_onboard_profile_cfg_t::p_extend

  Extended configuration.

close

```
ssp_err_t(* sf_ble_onboard_profile_api_t::close) (sf_ble_onboard_profile_ctrl_t
*const p_ctrl)
```

Brief description

De-initialize the interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.

Detailed description

Table 58: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the BLE module.

Parameter p_ctrl

Definition: sf_ble_onboard_profile_ctrl_t

BLE On-Board Profile Framework control structure

onbpEnable

```
ssp_err_t(* sf_ble_onboard_profile_api_t::onbpEnable)
(sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle,
sf_onbp_t profile, sf_ble_profile_callback_t p_prf_cb, sf_ble_prf_sec_t sec)
```

Brief description

Enable On-Board Profile Enables On-Board profile on given connection handle with specified security type. Registers user callback for the profile.

Detailed description

Table 59: Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to control structure for BLE
p_handle	in	Pointer to connection handle
profile	in	Profile type to enable

Table 59: Parameters (Continued)

Name	Direction	Description
p_prf_cb	in	User callback for Profile
sec	in	Security type for profile

Parameter p_ctrl

Definition: `sf_ble_onboard_profile_ctrl_t`

BLE On-Board Profile Framework control structure

Parameter p_handle
Parameter profile

Definition: `sf_onbp_tprofile`

BLE On-Board Profile types

Parameter p_prf_cb

Definition: `sf_ble_profile_callback_tp_prf_cb`

User callback type for profile

Parameter sec

onbpDisable

```
ssp_err_t(* sf_ble_onboard_profile_api_t::onbpDisable)
(sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle,
sf_onbp_t profile)
```

Brief description

Disable On-Board Profile Disables On-Board profile on given connection handle and unregisters user callback.

Detailed description

Table 60: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure for BLE
p_handle	in	Pointer to connection handle
profile	in	Profile type to disable

Parameter p_ctrl

Definition: `sf_ble_onboard_profile_ctrl_t`

BLE On-Board Profile Framework control structure

Parameter p_handle
Parameter profile

Definition: sf_onbp_tprofile

BLE On-Board Profile types

onbpServerWriteData

```
ssp_err_t(* sf_ble_onboard_profile_api_t::onbpServerWriteData)
(sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle,
sf_onbp_t profile, sf_ble_onbp_char_t characteristics, const void *p_data)
```

Brief description

Update Server Local Database Update Local GATT database of Profile Server.

Detailed description

Table 61: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure for BLE
p_handle	in	Pointer to connection handle
profile	in	Profile type
characteristics	in	Profile characteristics
p_data	in	Pointer to data

Parameter p_ctrl

Definition: sf_ble_onboard_profile_ctrl_t

BLE On-Board Profile Framework control structure

Parameter p_handle
Parameter profile

Definition: sf_onbp_tprofile

BLE On-Board Profile types

Parameter characteristics

Definition: sf_ble_onbp_char_tcharacteristics

On-Board Profile GATT Characteristics Code

Parameter p_data

const

onbpServerSendNotification

```
ssp_err_t(* sf_ble_onboard_profile_api_t::onbpServerSendNotification)
(sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle,
sf_onbp_t profile, sf_ble_onbp_char_t characteristics, const void *p_data)
```

Brief description

Send notification from Server Sends Notification which will be data specific to On-Board Profile to Client.

Detailed description

Table 62: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure for BLE
p_handle	in	Pointer to connection handle
profile	in	Profile type
characteristics	in	Profile characteristics
p_data	in	Pointer to data

Parameter p_ctrl

Definition: `sf_ble_onboard_profile_ctrl_t`

BLE On-Board Profile Framework control structure

Parameter p_handle
Parameter profile

Definition: `sf_onbp_tprofile`

BLE On-Board Profile types

Parameter characteristics

Definition: `sf_ble_onbp_char_tcharacteristics`

On-Board Profile GATT Characteristics Code

Parameter p_data

`const`

onbpServerSendIndication

```
ssp_err_t(* sf_ble_onboard_profile_api_t::onbpServerSendIndication)
(sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle,
sf_onbp_t profile, sf_ble_onbp_char_t characteristics, const void *p_data)
```

Brief description

Send Indication from Server Sends Indication which contains profile specific data to client.

Detailed description

Table 63: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure for BLE

Table 63: Parameters (Continued)

Name	Direction	Description
p_handle	in	Pointer to connection handle
profile	in	Profile type
characteristics	in	Profile characteristics
p_data	in	Pointer to data

Parameter p_ctrl

Definition: `sf_ble_onboard_profile_ctrl_t`

BLE On-Board Profile Framework control structure

Parameter p_handle
Parameter profile

Definition: `sf_onbp_tprofile`

BLE On-Board Profile types

Parameter characteristics

Definition: `sf_ble_onbp_char_tcharacteristics`

On-Board Profile GATT Characteristics Code

Parameter p_data

`const`

onbpClientWriteCCCD

```
ssp_err_t(* sf_ble_onboard_profile_api_t::onbpClientWriteCCCD)
(sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle,
sf_onbp_t profile, sf_ble_onbp_char_t cccd_char, sf_ble_cccd_val_t cccd_val)
```

Brief description

Writes CCCD configuration in Server This API writes CCCD configuration of Server and which enables or disables notification or indication from server.

Detailed description

Table 64: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure for BLE
p_handle	in	Pointer to connection handle
profile	in	Parameter_Description

Table 64: Parameters (Continued)

Name	Direction	Description
cccd_char	in	CCCD Code
cccd_val	in	Configuration data of CCCD

Parameter p_ctrl

Definition: sf_ble_onboard_profile_ctrl_t

BLE On-Board Profile Framework control structure

Parameter p_handle

Parameter profile

Definition: sf_onbp_tprofile

BLE On-Board Profile types

Parameter cccd_char

Definition: sf_ble_onbp_char_tcccd_char

On-Board Profile GATT Characteristics Code

Parameter cccd_val

onbpClientWriteChar

```
ssp_err_t(* sf_ble_onboard_profile_api_t::onbpClientWriteChar)
(sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle,
sf_onbp_t profile, sf_ble_onbp_char_t characteristics, const void *p_data)
```

Brief description

Writes GATT characteristics with data passed Writes the GATT characteristics in the Server with the data passed.

Detailed description

Table 65: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure for BLE
p_handle	in	Pointer to connection handle
profile	in	Profile type
characteristics	in	GATT characteristics code
p_data	in	Pointer to data

Parameter p_ctrl

Definition: sf_ble_onboard_profile_ctrl_t

BLE On-Board Profile Framework control structure

Parameter p_handle
Parameter profile

Definition: sf_onbp_tprofile

BLE On-Board Profile types

Parameter characteristics

Definition: sf_ble_onbp_char_tcharacteristics

On-Board Profile GATT Characteristics Code

Parameter p_data

const

onbpClientReadChar

```
ssp_err_t(* sf_ble_onboard_profile_api_t::onbpClientReadChar)
(sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle,
sf_onbp_t profile, sf_ble_onbp_char_t characteristics)
```

Brief description

Reads GATT characteristics from Server Reads GATT characteristics from the Server for the profile specified.

Detailed description

Table 66: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure for BLE
p_handle	in	Pointer to connection handle
profile	in	Profile type
characteristics	in	Profile characteristics

Parameter p_ctrl

Definition: sf_ble_onboard_profile_ctrl_t

BLE On-Board Profile Framework control structure

Parameter p_handle
Parameter profile

Definition: sf_onbp_tprofile

BLE On-Board Profile types

Parameter characteristics

Definition: sf_ble_onbp_char_tcharacteristics

On-Board Profile GATT Characteristics Code

versionGet

```
ssp_err_t(* sf_ble_onboard_profile_api_t::versionGet) (ssp_version_t *const
p_version)
```

Brief description

Gets version and stores it in provided pointer p_version.

Detailed description

Table 67: Parameters

Name	Direction	Description
p_version	out	pointer to memory location to return version number

Parameter p_version

sf_ble_onboard_profile_instance_t

sf_ble_onboard_profile_instance_t

Detailed description

Instance structure

Variables

- sf_ble_onboard_profile_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_ble_onboard_profile_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_ble_onboard_profile_api_t const * p_api

  Pointer to the API structure for this instance.

6.6.3 SF BLE Alert Notification Profile Framework Interface

RTOS-integrated SF BLE Alert Notification Profile Framework Interface.

6.6.3.1 Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Alert Notification Profile Framework.

6.6.3.2 Interface API

sf_ble_anp_api_new_alert_t

Function name	Description
.category_id	Category ID.
.alert_num	Number of New Alert.
.text_size	Text Size.
.text	Actual Text.

sf_ble_anp_api_new_alert_ntf_t

Function name	Description
.conhdl	Connection handle.
.new_alert	New Alert data.

sf_ble_anp_api_unread_alert_t

Function name	Description
.category_id	Category ID.
.unread_count	Number of Unread Alert.

sf_ble_anp_api_unread_alert_ntf_t

Function name	Description
.conhdl	Connection handle.
.alert	Unread Alert data.

6.6.3.3 Data structures

- sf_ble_anp_ancp_t
- sf_ble_anp_ancp_change_t
- sf_ble_anp_api_new_alert_t
- sf_ble_anp_api_new_alert_ntf_t

- sf_ble_anp_api_unread_alert_t
- sf_ble_anp_api_unread_alert_ntf_t

6.6.3.4 Enumerations

- sf_ble_prf_anpc_event_t
- sf_ble_prf_anps_event_t
- sf_ble_prf_anp_category_id
- sf_ble_prf_anp_cmd_id_t

6.6.3.5 Defines

- #define SF_BLE_ANP_ALT_TEXT_MAX

  Initial value: `(18U)`

  Buffer size of data for Alert Notification

6.6.3.6 API Data

sf_ble_prf_anpc_event_t

`sf_ble_prf_anpc_event_t`

Detailed description

Profile Client user events

Enumerated values

Name	Description
SF_BLE_PRF_ANPC_EVENT_NONE	Event not supported.
SF_BLE_PRF_ANPC_EVENT_NEW_ALT_NTF	New Alert Data received event, Refer sf_ble_anp_api_new_alert_ntf_t.
SF_BLE_PRF_ANPC_EVENT_UNREAD_ALT_NTF	Unread Alert Data received event, Refer sf_ble_anp_api_unread_alert_ntf_t.
SF_BLE_PRF_ANPC_EVENT_READ_CHAR_RES	Read Char Complete Event.

sf_ble_prf_anps_event_t

`sf_ble_prf_anps_event_t`

Detailed description

Alert notification server profile user events

Enumerated values

Name	Description
SF_BLE_PRF_ANPS_EVENT_NONE	Event not supported.
SF_BLE_PRF_ANPS_EVENT_ALT_NF_CP_IND	Alert Notification Control Point Changed indication, Refer sf_ble_anp_ancp_change_t.
SF_BLE_PRF_ANPS_EVENT_CCCD_NTF_IND	CCCD Notification Setting change event.

sf_ble_prf_anp_category_id

`sf_ble_prf_anp_category_id`

Detailed description

Alert Notification Control Point Category ID

Enumerated values

Name	Description
SF_BLE_PRF_ANP_CATEGORY_ID_SIMPLE_ALERT	Simple Alert: General text alert or non-text alert.
SF_BLE_PRF_ANP_CATEGORY_ID_EMAIL	Email Alert.
SF_BLE_PRF_ANP_CATEGORY_ID_NEWS	News feeds Alert.
SF_BLE_PRF_ANP_CATEGORY_ID_CALL	Incoming Call Alert.
SF_BLE_PRF_ANP_CATEGORY_ID_MISSED_CALL	Missed Call Alert.
SF_BLE_PRF_ANP_CATEGORY_ID_SMS_MMS	SMS/MMS Message Alert.
SF_BLE_PRF_ANP_CATEGORY_ID_VOICE_MAIL	Voice Mail Alert.
SF_BLE_PRF_ANP_CATEGORY_ID_SCHEDULE	Alert occurred on calendar, planner.
SF_BLE_PRF_ANP_CATEGORY_ID_HIGH_PRIORITY_ALERT	High Prioritized Alert.
SF_BLE_PRF_ANP_CATEGORY_ID_INSTANT_MESSAGE	Incoming Instant Messages.
SF_BLE_PRF_ANP_CATEGORY_ID_ALL	All Supported Categories.

sf_ble_prf_anp_cmd_id_t

`sf_ble_prf_anp_cmd_id_t`

Detailed description

Alert Notification Control Point Command ID

Enumerated values

Name	Description
SF_BLE_PRF_ANP_CMD_ID_NEW_ALERT_ENABLE	Enable New Incoming Alert Notification.
SF_BLE_PRF_ANP_CMD_ID_UNREAD_ALERT_ENABLE	Enable Unread Category Status Notification.
SF_BLE_PRF_ANP_CMD_ID_NEW_ALERT_DISABLE	Disable New Incoming Alert Notification.
SF_BLE_PRF_ANP_CMD_ID_UNREAD_ALERT_DISABLE	Disable Unread Category Status Notification.
SF_BLE_PRF_ANP_CMD_ID_NEW_ALERT_NTF_REQ	Notify New Incoming Alert immediately.
SF_BLE_PRF_ANP_CMD_ID_UNREAD_ALERT_NTF_REQ	Notify Unread Category Status immediately.

6.6.3.7 API Structures

sf_ble_anp_ancp_t

sf_ble_anp_ancp_t

Detailed description

Write Characteristics data for Alert Notification Control Point

Variables

- sf_ble_prf_anp_cmd_id_t command_id

  Command type.

- sf_ble_prf_anp_category_id category_id

  Category on which to act.

sf_ble_anp_ancp_change_t

sf_ble_anp_ancp_change_t

Detailed description

Alert Notification Control Point Change Indication for Sensor

Variables

- sf_ble_conn_handle_t conhdl

  Connection handle.

- sf_ble_anp_ancp_t control_point_value

  Control point value.

sf_ble_anp_api_new_alert_t

sf_ble_anp_api_new_alert_t

Detailed description

Notification Data for New Alert, also used for sending notification from server

category_id

`sf_ble_prf_anp_category_id::category_id`

Brief description

Category ID.

alert_num

`uint8_t sf_ble_anp_api_new_alert_t::alert_num`

Brief description

Number of New Alert.

text_size

`uint8_t sf_ble_anp_api_new_alert_t::text_size`

Brief description

Text Size.

text

`uint8_t sf_ble_anp_api_new_alert_t::text[SF_BLE_ANP_ALT_TEXT_MAX]`

Brief description

Actual Text.

sf_ble_anp_api_new_alert_ntf_t

sf_ble_anp_api_new_alert_ntf_t

Detailed description

Notification Data received for New Alert

conhdl

`sf_ble_conn_handle_t::conhdl`

Brief description

Connection handle.

new_alert

`sf_ble_anp_api_new_alert_t::new_alert`

Brief description

New Alert data.

sf_ble_anp_api_unread_alert_t

sf_ble_anp_api_unread_alert_t

Detailed description

Notification Data for Unread Alert, also used for sending notification from server

category_id

`sf_ble_prf_anp_category_id::category_id`

Brief description

Category ID.

unread_count

`uint8_t sf_ble_anp_api_unread_alert_t::unread_count`

Brief description

Number of Unread Alert.

sf_ble_anp_api_unread_alert_ntf_t

sf_ble_anp_api_unread_alert_ntf_t

Detailed description

Notification Data received for Unread Alert

conhdl

`sf_ble_conn_handle_t::conhdl`

Brief description

Connection handle.

alert

`sf_ble_anp_api_unread_alert_t::alert`

Brief description

Unread Alert data.

6.6.4 SF BLE Battery Service Profile Framework Interface

RTOS-integrated SF BLE Battery Service Profile Framework Interface.

6.6.4.1 Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Battery Service Profile Framework.

6.6.4.2 Data structures

- sf_ble_bas_battery_lvl_ntf_t

6.6.4.3 Variables

- sf_ble_prf_bas_battery_lvl_t

6.6.4.4 sf_ble_prf_bas_battery_lvl_t

```
SSP_HEADER typedef uint8_t sf_ble_prf_bas_battery_lvl_t
```

Detailed description

Battery Level

6.6.4.5 API Structures

sf_ble_bas_battery_lvl_ntf_t

sf_ble_bas_battery_lvl_ntf_t

Detailed description

Control Point Change Indication for Sensor

Variables

- sf_ble_conn_handle_t conhdl

  Connection handle.
- sf_ble_prf_bas_battery_lvl_t batt_lvl

  Battery Level.

6.6.5 SF BLE Blood Pressure Profile Framework Interface

RTOS-integrated SF BLE Blood Pressure Profile Framework Interface.

6.6.5.1 Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Blood Pressure Profile Framework.

6.6.5.2 Data structures

- sf_ble_blp_meas_info_t
- sf_ble_blp_meas_recv_data_t

6.6.5.3 Enumerations

- sf_ble_prf_blpc_event_t
- sf_ble_prf_blps_event_t

6.6.5.4 API Data

sf_ble_prf_blpc_event_t

`sf_ble_prf_blpc_event_t`

Detailed description

Profile user client events

Enumerated values

Name	Description
SF_BLE_PRF_BLPC_EVENT_NONE	Event not supported.
SF_BLE_PRF_BLPC_EVENT_MEAS_NTF_IND	BLE user event indicating BLPC profile measurement notification.
SF_BLE_PRF_BLPC_EVENT_WRITE_CHAR_RES	BLE user event indicating BLPC profile read char response.
SF_BLE_PRF_BLPC_EVENT_READ_CHAR_RES	BLE user event indicating BLPC write char response.

sf_ble_prf_blps_event_t

`sf_ble_prf_blps_event_t`

Detailed description

Profile user server events

Enumerated values

Name	Description
SF_BLE_PRF_BLPS_EVENT_NONE	Event not supported.
SF_BLE_PRF_BLPS_EVENT_NTFIND_IND	BLE user event indicating notification indication.

6.6.5.5 API Structures

sf_ble_blp_meas_info_t

sf_ble_blp_meas_info_t

Detailed description

Blood Pressure Measurements Info

Variables

- uint8_t flag_stable_meas

  Stable or intermediary type of measurements. Set to 0 if intermediate measurement.
- uint8_t flags

  flags
- int16_t press_val1

  blood pressure value - Systolic or cuff pressure. Cuff pressure has to be filled here
- int16_t press_val2

  blood pressure value - Diastolic or subfield1
- int16_t press_val3

  blood pressure value - MAP or subfield2
- sf_ble_prf_cts_date_time_t stamp

  time stamp
- int16_t rate

  pulse rate
- uint8_t id

  user ID
- uint8_t reserved

  Reserved.
- uint16_t meas_sts

  measurement status

sf_ble_blp_meas_recv_data_t

sf_ble_blp_meas_recv_data_t

Detailed description

Blood Pressure Measurements Data Received from Server

Variables

- uint16_t conhdl

  Connection handle.
- sf_ble_onbp_char_t char_code

  Characteristic code.
- sf_ble_blp_meas_info_t meas_info

  BLP measurement data.

6.6.6 SF BLE Current Time Service Profile Framework Interface

RTOS-integrated SF BLE Current Time Service Profile Framework Interface.

6.6.6.1 Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Current Time Service Profile Framework.

6.6.6.2 Data structures

- sf_ble_prf_cts_date_time_t
- sf_ble_prf_cts_curr_time_t
- sf_ble_cts_local_time_t
- sf_ble_cts_ref_time_t
- sf_ble_cts_curr_time_ntf_t

6.6.6.3 API Structures

sf_ble_prf_cts_date_time_t

sf_ble_prf_cts_date_time_t

Detailed description

Current Time Service Date Time information

Variables

- uint16_t year

  Year value.
- uint8_t month

  Month value.
- uint8_t day

  Day value.
- uint8_t hour

  Hour value.
- uint8_t min

  Minute value.
- uint8_t sec

  Second value.
- uint8_t reserved

  Reserved.

sf_ble_prf_cts_curr_time_t

sf_ble_prf_cts_curr_time_t

Detailed description

Current time information with Date Time

Variables

- sf_ble_prf_cts_date_time_t stamp

  Date time info.
- uint8_t day_of_week

  Day of week.
- uint8_t fractions256

  Fraction value.
- uint8_t adjust_reason

  Adjust reason.
- uint8_t reserved

  Reserved.

sf_ble_cts_local_time_t

sf_ble_cts_local_time_t

Detailed description

Local Time Information structure

Variables

- int8_t time_zone

  Time Zone.
- uint8_t dst_offset

  DST Offset.

sf_ble_cts_ref_time_t

sf_ble_cts_ref_time_t

Detailed description

Reference Time Information structure

Variables

- uint8_t time_source

  Source of time.
- uint8_t accuracy

  Estimated accuracy of time compared to original time source.

- uint8_t days_since_update

  Days that passed since time was updated successfully from time source.
- uint8_t hours_since_update

  Time that passed since time was updated successfully from time source.

sf_ble_cts_curr_time_ntf_t

sf_ble_cts_curr_time_ntf_t

Detailed description

Notification Data of Current Time received from server

Variables

- sf_ble_conn_handle_t conhdl

  Connection handle.
- sf_ble_prf_cts_curr_time_t current_time

  heart rate measurement data

6.6.7 SF BLE Find Me Profile Framework Interface

RTOS-integrated SF BLE Find Me Profile Framework Interface.

6.6.7.1 Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Find Me Profile Framework.

6.6.7.2 API Data

sf_ble_prf_fmpt_event_t

`sf_ble_prf_fmpt_event_t`

Detailed description

Profile Server user events

Enumerated values

Name	Description
SF_BLE_PRF_FMPT_EVENT_NONE	BLE event unsupported.
SF_BLE_PRF_FMPT_EVENT_ALERT_LVL_CHANGED	BLE user event Alert level changed from locator, Refer sf_ble_ias_alert_lvl_change_t.

6.6.8 SF BLE HID Over GATT Profile Framework Interface

RTOS-integrated SF BLE HID Over GATT Profile Framework Interface.

6.6.8.1 Summary

This SSP Interface provides access to the ThreadX-aware SF BLE HID Over GATT Profile Framework.

6.6.8.2 Data structures

- sf_ble_prf_value_t
- sf_ble_prf_hid_report_desc_t
- sf_ble_prf_hid_report_ind_t
- sf_ble_prf_hid_change_event_t
- sf_ble_prf_dis_pnpid_t

6.6.8.3 Enumerations

- sf_ble_prf_hidd_event_t
- sf_ble_prf_hid_event_t
- sf_ble_hidd_device_type_t

6.6.8.4 Typedefs

- sf_ble_prf_hid_protocol_mode_t
- sf_ble_prf_hid_ctrl_point_val_t

6.6.8.5 Defines

- #define SF_BLE_PRF_HIDS_REPORT_MAX

  Initial value: `(32U)`

  Maximum Number of reports in HID

6.6.8.6 API Data

sf_ble_prf_hidd_event_t

`sf_ble_prf_hidd_event_t`

Detailed description

Profile Server user events

Enumerated values

Name	Description
SF_BLE_PRF_HIDD_EVENT_NONE	Event not supported.
SF_BLE_PRF_HIDD_EVENT_REPORT_IND	Report value updated from Boot Host or Report Host, Refer sf_ble_prf_hid_report_ind_t.
SF_BLE_PRF_HIDD_EVENT_CFG_IND	CCCD change indication from Boot Host or Report Host.
SF_BLE_PRF_HIDD_EVENT_PROTO_MODE_CHG_EVT	protocol mode change event from Boot Host or Report Host, Refer sf_ble_prf_hid_change_event_t
SF_BLE_PRF_HIDD_EVENT_REPORT_EVT	report value update event from Boot Host or Report Host, Refer sf_ble_prf_hid_report_ind_t
SF_BLE_PRF_HIDD_EVENT_CP_CHANGED_EVT	suspend event from Report Host, Refer sf_ble_prf_hid_change_event_t

sf_ble_prf_hid_event_t

`sf_ble_prf_hid_event_t`

Detailed description

Profile Client user events

Enumerated values

Name	Description
SF_BLE_PRF_HID_EVENT_NONE	Event not supported.
SF_BLE_HID_BHOST_EVENT_REPORT_NTF	Report value received from HID device, Refer sf_ble_prf_hid_report_ind_t. HID BHOST
SF_BLE_HID_BHOST_EVENT_READ_CHAR_RESP	read char response received from HID device
SF_BLE_HID_RHOST_EVENT_REPORT_NTF	Report value received from HID device, Refer sf_ble_prf_hid_report_ind_t. HID RHOST
SF_BLE_HID_RHOST_EVENT_BATTERY_LVL_NTF	Battery level received from HID device, Refer sf_ble_bas_battery_lvl_ntf_t.
SF_BLE_HID_RHOST_EVENT_READ_CHAR_RESP	Read char response received from HID device.
SF_BLE_HID_RHOST_EVENT_READ_LONG_CHAR_RESP	Long char read response received from HID device.

sf_ble_hidd_device_type_t

`sf_ble_hidd_device_type_t`

Detailed description

HID Device types

Enumerated values

Name	Description
SF_BLE_HIDD_HID_DEVICE	HID Device type.
SF_BLE_HIDD_BOOT_KEYBOARD	Boot Keyboard type.
SF_BLE_HIDD_BOOT_MOUSE	Boot Mouse type.

sf_ble_prf_hid_protocol_mode_t

`typedef uint8_t sf_ble_prf_hid_protocol_mode_t`

Detailed description

Protocol Mode Characteristics

sf_ble_prf_hid_ctrl_point_val_t

`typedef uint8_t sf_ble_prf_hid_ctrl_point_val_t`

Detailed description

HID Control Point Characteristics

6.6.8.7 API Structures

sf_ble_prf_value_t

Detailed description

HID Profile value changed by the client

Variables

protocol_mode_val

control_point_val

protocol_mode_val

`sf_ble_prf_hid_protocol_mode_t::protocol_mode_val`

Brief description

Protocol Mode.

control_point_val

`sf_ble_prf_hid_ctrl_point_val_t::control_point_val`

Brief description

HID Control Point.

sf_ble_prf_hid_report_desc_t

sf_ble_prf_hid_report_desc_t

Detailed description

HID Report structure

Variables

- sf_ble_hidd_device_type_t device_type

  Device type.
- uint8_t report_type

  Report type.
- uint8_t value[SF_BLE_PRF_HIDS_REPORT_MAX]

  Report values.
- uint16_t value_size

  Report size.

sf_ble_prf_hid_report_ind_t

sf_ble_prf_hid_report_ind_t

Detailed description

HID Report Indication structure

Variables

- uint16_t conhdl

  Connection handle.
- uint8_t inst_idx

  Instance Index.
- uint8_t reserved

  Reserved.
- sf_ble_prf_hid_report_desc_t report

  Report received from either BHOST or RHOST.

sf_ble_prf_hid_change_event_t

sf_ble_prf_hid_change_event_t

Detailed description

HID Profile Values Change structure

Variables

- uint16_t conhdl

  Connection handle.
- uint8_t inst_idx

  Instance Index.
- sf_ble_prf_value_t ble_prf_value

  HID Profile value changed by the client.

sf_ble_prf_dis_pnpid_t

sf_ble_prf_dis_pnpid_t

Detailed description

Structure to set the current Device Information PnP Id characteristic value

Variables

- uint8_t vendorIdSource

  Vendor ID source.
- uint16_t vendorId

  Vendor ID.
- uint16_t productId

  Product ID.
- uint16_t productVersion

  Version of Product.

6.6.9 SF BLE Heart Rate Profile Framework Interface

RTOS-integrated SF BLE Heart Rate Profile Framework Interface.

6.6.9.1 Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Heart Rate Profile Framework.

6.6.9.2 Interface API

sf_ble_hrp_api_hrmeas_t

Function name	Description
.flags	HRP bit Flags.
.rr_interval_num	number of RR Interval

Function name	Description
.heart_rate_measure	Heart Rate measurement value.
.energy_expended	Energy Expended in Kilo Joules from last time it was reset.
.rr_interval	RR interval(s)

sf_ble_hrp_api_meas_ntf_t

Function name	Description
.conhdl	Connection handle.
.measurements_info	heart rate measurement data

6.6.9.3 Data structures

- sf_ble_hrp_api_hrmeas_t
- sf_ble_hrp_api_meas_ntf_t
- sf_ble_hrp_cp_change_t

6.6.9.4 Enumerations

- sf_ble_prf_hrpc_event_t
- sf_ble_prf_hrps_event_t

6.6.9.5 Typedefs

- sf_ble_prf_hrp_api_hrcp_t

6.6.9.6 Defines

- #define SF_BLE_PRF_HRP_API_RR_INTERVAL_BUFF_LEN

  Initial value: `(0x9U)`

  Heart Rate RR Interval Buffer Length

6.6.9.7 API Data

sf_ble_prf_hrpc_event_t

`sf_ble_prf_hrpc_event_t`

Detailed description

Profile Client user events

Enumerated values

Name	Description
SF_BLE_PRF_HRPC_EVENT_NONE	Event not supported.
SF_BLE_PRF_HRPC_EVENT_MEAS_NTF	Heart Rate Measurement data received event, Refer sf_ble_hrp_api_meas_ntf_t.
SF_BLE_PRF_HRPC_EVENT_READ_CHAR_RES	Read Char Complete Event.

sf_ble_prf_hrps_event_t

`sf_ble_prf_hrps_event_t`

Detailed description

Profile Server user events

Enumerated values

Name	Description
SF_BLE_PRF_HRPS_EVENT_NONE	Event not supported.
SF_BLE_PRF_HRPS_EVENT_NTF_CHG_IND	CCCD Notification Setting Change Event.
SF_BLE_PRF_HRPS_EVENT_HRCP_CHG_IND	Heart Rate Control Point Changed Event, Refer sf_ble_hrp_cp_change_t.

sf_ble_prf_hrp_api_hrcp_t

`typedef uint16_t sf_ble_prf_hrp_api_hrcp_t`

Detailed description

Write Characteristics data for Heart Rate Control Point

6.6.9.8 API Structures

sf_ble_hrp_api_hrmeas_t

sf_ble_hrp_api_hrmeas_t

Detailed description

Notification Data for Heart Rate Measurement, also used for sending notification from server

flags

`uint8_t sf_ble_hrp_api_hrmeas_t::flags`

Brief description

HRP bit Flags.

rr_interval_num

`uint8_t sf_ble_hrp_api_hrmeas_t::rr_interval_num`

Brief description

number of RR Interval

heart_rate_measure

`uint16_t sf_ble_hrp_api_hrmeas_t::heart_rate_measure`

Brief description

Heart Rate measurement value.

energy_expended

`uint16_t sf_ble_hrp_api_hrmeas_t::energy_expended`

Brief description

Energy Expended in Kilo Joules from last time it was reset.

rr_interval

`uint16_t sf_ble_hrp_api_hrmeas_t::rr_interval[SF_BLE_PRF_HRP_API_RR_INTERVAL_BUFF_LEN]`

Brief description

RR interval(s)

sf_ble_hrp_api_meas_ntf_t

sf_ble_hrp_api_meas_ntf_t

Detailed description

Notification Data of Heart Rate measurement received from sensor

conhdl

`sf_ble_conn_handle_t::conhdl`

Brief description

Connection handle.

measurements_info

`sf_ble_hrp_api_hrmeas_t::measurements_info`

Brief description

heart rate measurement data

sf_ble_hrp_cp_change_t

sf_ble_hrp_cp_change_t

Detailed description

Control Point Change Indication for Sensor

Variables

- sf_ble_conn_handle_t conhdl

  Connection handle.
- sf_ble_prf_hrp_api_hrcp_t control_point_value

  Control point value.

6.6.10 SF BLE Health Thermometer Profile Framework Interface

RTOS-integrated SF BLE Health Thermometer Profile Framework Interface.

6.6.10.1 Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Health Thermometer Profile Framework.

6.6.10.2 Data structures

- sf_ble_prf_htp_temp_info_t
- sf_ble_prf_htp_temp_info_ind_t
- st_sf_ble_prf_htp_meas_intv_val_t

6.6.10.3 Enumerations

- sf_ble_event_prf_htpt_t
- sf_ble_event_prf_htpc_t
- sf_ble_prf_htp_meas_state_t
- sf_ble_prf_htp_temp_type_t

6.6.10.4 API Data

sf_ble_event_prf_htpt_t

`sf_ble_event_prf_htpt_t`

Detailed description

HTP thermometer (server) user events

Enumerated values

Name	Description
SF_BLE_EVENT_PRF_HTPT_NONE	Event not supported.
SF_BLE_EVENT_PRF_HTPT_MEAS_INTV_CHG_IND	Thermometer measurement interval characteristic change indication.
SF_BLE_EVENT_PRF_HTPT_CFG_IND	BLE user event indicating HTPT profile CCCD change indication.

sf_ble_event_prf_htpc_t

`sf_ble_event_prf_htpc_t`

Detailed description

HTP Collector (client) user events

Enumerated values

Name	Description
SF_BLE_EVENT_PRF_HTPC_NONE	Event not supported.
SF_BLE_EVENT_PRF_HTPC_TEMP_IND	BLE user event indicating HTPC profile temperature value indication.
SF_BLE_EVENT_PRF_HTPC_MEAS_INTV_IND	BLE user event indicating HTPC profile measurement interval value indication.
SF_BLE_PRF_HTPC_EVENT_READ_CHAR_RES	BLE user event for read char response.

sf_ble_prf_htp_meas_state_t

`sf_ble_prf_htp_meas_state_t`

Detailed description

HTP Collector (client) user events

Enumerated values

Name	Description
SF_BLE_PRF_HTP_MEAS_STATE_IN_PROGRESS	Measurement in progress.
SF_BLE_PRF_HTP_MEAS_STATE_COMPLETE	Measurement is complete.

sf_ble_prf_htp_temp_type_t

`sf_ble_prf_htp_temp_type_t`

Detailed description

HTP Temperature Types

Enumerated values

Name	Description
SF_BLE_PRF_HTP_TYPE_ARMPIT	Temperature type Armpit.
SF_BLE_PRF_HTP_TYPE_BODY	Temperature type Body.
SF_BLE_PRF_HTP_TYPE_EAR	Temperature type Ear.
SF_BLE_PRF_HTP_TYPE_FINGER	Temperature type Finger.
SF_BLE_PRF_HTP_TYPE_GASTROINTESTINAL	Temperature type Gastrointestinal.
SF_BLE_PRF_HTP_TYPE_MOUTH	Temperature type Mouth.
SF_BLE_PRF_HTP_TYPE_RECTUM	Temperature type Rectum.
SF_BLE_PRF_HTP_TYPE_TOE	Temperature type Toe.
SF_BLE_PRF_HTP_TYPE_TYMPANUM	Temperature type Tympanum.

6.6.10.5 API Structures

sf_ble_prf_htp_temp_info_t

sf_ble_prf_htp_temp_info_t

Detailed description

Health Thermometer Info

Variables

- uint8_t flag_stable_meas

  Stable or intermediary type of temperature.
- uint8_t flags

  flags
- int32_t temp_val

  temp value
- sf_ble_prf_cts_date_time_t stamp

  time stamp

- sf_ble_prf_htp_temp_type_t type
  type
- uint8_t reserved
  Reserved.

sf_ble_prf_htp_temp_info_ind_t

sf_ble_prf_htp_temp_info_ind_t

Detailed description

Thermometer Information Indication

Variables

- uint16_t conhdl
  Connection handle.
- sf_ble_prf_htp_temp_info_t temp_info
  Thermometer info.

st_sf_ble_prf_htp_meas_intv_val_t

st_sf_ble_prf_htp_meas_intv_val_t

Detailed description

Measurement Interval value

Variables

- uint16_t conhdl
  Connection handle.
- uint16_t intv
  Interval value.

6.6.11 SF BLE Immediate Alert Profile Framework Interface

RTOS-integrated SF BLE Immediate Alert Profile Framework Interface.

6.6.11.1 Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Immediate Alert Profile Framework.

6.6.11.2 Data structures

- sb_ble_prf_ias_set_alert_t
- sf_ble_prf_ias_alert_lvl_change_t

6.6.11.3 Enumerations

- sf_ble_prf_ias_alert_type_t
- sf_ble_prf_ias_svc_code_t

6.6.11.4 API Data

sf_ble_prf_ias_alert_type_t

`sf_ble_prf_ias_alert_type_t`

Detailed description

Alert Level Values

Enumerated values

Name	Description
SF_BLE_PRF_ALERT_TYPE_NONE	No Alert.
SF_BLE_PRF_ALERT_TYPE_MILD	Mild Alert.
SF_BLE_PRF_ALERT_TYPE_HIGH	High Alert.

sf_ble_prf_ias_svc_code_t

`sf_ble_prf_ias_svc_code_t`

Detailed description

SVC codes

Enumerated values

Name	Description
SF_BLE_SVC_SET_LK_LOSS_ALERT	Code for LLS Alert Level Char.
SF_BLE_SVC_SET_IMMDT_ALERT	Code for IAS Alert Level Char.

6.6.11.5 API Structures

sb_ble_prf_ias_set_alert_t

sb_ble_prf_ias_set_alert_t

Detailed description

Alert level and type

Variables

- sf_ble_prf_ias_svc_code_t svc_code

  SVC code.
- sf_ble_prf_ias_alert_type_t lvl

  Alert level.

sf_ble_prf_ias_alert_lvl_change_t

sf_ble_prf_ias_alert_lvl_change_t

Detailed description

Alert Level Change Indication for Server

Variables

- sf_ble_conn_handle_t conhdl

  Connection handle.
- sf_ble_prf_ias_alert_type_t alert_lvl

  Control point value.

6.6.12 SF BLE Next DST Change Service Profile Framework Interface

RTOS-integrated SF BLE Next DST Change Service Profile Framework Interface.

6.6.12.1 Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Next DST Change Service Profile Framework.

6.6.12.2 Data structures

- sf_ble_prf_ndcs_time_dst_t

6.6.12.3 API Structures

sf_ble_prf_ndcs_time_dst_t

sf_ble_prf_ndcs_time_dst_t

Detailed description

Next Time with DST Information structure

Variables

- sf_ble_prf_cts_date_time_t stamp

  Current time stamp.
- uint8_t dst_offset

  DST Offset.

- uint8_t reserved

  Reserved.

6.6.13 SF BLE Phone Alert Status Profile Framework Interface

RTOS-integrated SF BLE Phone Alert Status Profile Framework Interface.

6.6.13.1 Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Phone Alert Status Profile Framework.

6.6.13.2 Data structures

- sf_ble_prf_ringer_cp_change_t
- sf_ble_prf_ringer_setting_ntf_t
- sf_ble_prf_alert_status_ntf_t

6.6.13.3 Enumerations

- sf_ble_prf_papsc_event_t
- sf_ble_prf_papss_event_t
- sf_ble_prf_ringer_cp_t
- sf_ble_prf_ringer_setting_t
- sf_ble_prf_alert_setting_t

6.6.13.4 Typedefs

- sf_ble_prf_alert_status

6.6.13.5 API Data

sf_ble_prf_papsc_event_t

`sf_ble_prf_papsc_event_t`

Detailed description

Profile Client user events

Enumerated values

Name	Description
SF_BLE_PRF_PAPSC_EVENT_NONE	Event not supported.

Name	Description
SF_BLE_PRF_PAPSC_EVENT_READ_CHAR_RES	BLE user event indicating PAPS target alert indication received from locator.
SF_BLE_PRF_PAPSC_EVENT_RINGER_SETTING_IND	BLE user event indicating PAPS ringer setting indication, Refer sf_ble_prf_ringer_setting_ntf_t.
SF_BLE_PRF_PAPSC_EVENT_ALERT_STATUS_RES	BLE user event indicating PAPS alert status response, Refer sf_ble_prf_alert_status_ntf_t.

sf_ble_prf_papss_event_t

```
sf_ble_prf_papss_event_t
```

Detailed description

Profile Server user events

Enumerated values

Name	Description
SF_BLE_PRF_PAPSS_EVENT_NONE	Event not supported.
SF_BLE_PRF_PAPSS_EVENT_RINGER_CP_IND	BLE user event indicating ringer control point indication, Refer sf_ble_prf_ringer_cp_change_t.
SF_BLE_PRF_PAPSS_EVENT_CFG_NTF_IND	BLE user event indicating notification indication.

sf_ble_prf_ringer_cp_t

```
sf_ble_prf_ringer_cp_t
```

Detailed description

Ringer Control Point value

Enumerated values

Name	Description
SF_BLE_PRF_RINGER_CP_SILENT_MODE	Silent Mode.
SF_BLE_PRF_RINGER_CP_MUTE_ONCE	Mute Once.
SF_BLE_PRF_RINGER_CP_CANCEL_SILENT_MODE	Cancel Silent Mode.

sf_ble_prf_ringer_setting_t

```
sf_ble_prf_ringer_setting_t
```

Detailed description

Ringer Setting Value

Enumerated values

Name	Description
SF_BLE_PRF_RINGER_SETTING_SILENT	Ringer Silent.
SF_BLE_PRF_RINGER_SETTING_NORMAL	Ringer Normal.

sf_ble_prf_alert_setting_t

```
sf_ble_prf_alert_setting_t
```

Detailed description

PASP Alert status

Enumerated values

Name	Description
SF_BLE_PRF_ALERT_NONE	No active alert.
SF_BLE_PRF_ALERT_RINGER	Ringer State is active.
SF_BLE_PRF_ALERT_VIBRATOR	Vibrate State is active.
SF_BLE_PRF_ALERT_DISPLAY	Display Alert Status State is active.

sf_ble_prf_alert_status

```
typedef uint8_t sf_ble_prf_alert_status
```

Detailed description

PASP Alert Status

6.6.13.6 API Structures

sf_ble_prf_ringer_cp_change_t

sf_ble_prf_ringer_cp_change_t

Detailed description

Ringer Control Point value changed indication

Variables

- sf_ble_conn_handle_t conhdl

  Connection handle.

- sf_ble_prf_ringer_cp_t ringer_cp

  Ringer control point value.

sf_ble_prf_ringer_setting_ntf_t

sf_ble_prf_ringer_setting_ntf_t

Detailed description

Ringer Setting notification received data

Variables

- sf_ble_conn_handle_t conhdl

  Connection handle.

- sf_ble_prf_ringer_setting_t ringer_setting

  Ringer control point value.

sf_ble_prf_alert_status_ntf_t

sf_ble_prf_alert_status_ntf_t

Detailed description

Alert Status notification received data

Variables

- sf_ble_conn_handle_t conhdl

  Connection handle.

- sf_ble_prf_alert_status alert_status

  Ringer control point value.

6.6.14 SF BLE Proximity Profile Framework Interface

RTOS-integrated SF BLE Proximity Profile Framework Interface.

6.6.14.1 Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Proximity Profile Framework.

6.6.14.2 API Data

sf_ble_prf_pxpr_event_t

`sf_ble_prf_pxpr_event_t`

Detailed description

PXP reporter (server) user events

Enumerated values

Name	Description
SF_BLE_PRF_PXPR_EVENT_NONE	unsupported event
SF_BLE_PRF_PXPR_EVENT_ALT_IND	BLE user event indicating PXPR profile alert indication.
SF_BLE_PRF_PXPR_EVENT_CMD_DISALLOWED_IND	command disallowed error received

sf_ble_prf_pxpm_event_t

`sf_ble_prf_pxpm_event_t`

Detailed description

PXP monitor (client) user events

Enumerated values

Name	Description
SF_BLE_PRF_PXPM_EVENT_NONE	unsupported event
SF_BLE_PRF_PXPM_EVENT_READ_CHAR_RESP	BLE user event indicating PXPR profile alert indication.
SF_BLE_PRF_PXPM_EVENT_CMD_DISALLOWED_IND	command disallowed error received
SF_BLE_PRF_PXPM_EVENT_ERROR_IND	error received for command issued

6.6.15 SF BLE Reference Time Update Service Profile Framework Interface

RTOS-integrated SF BLE Reference Time Update Service Profile Framework Interface.

6.6.15.1 Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Reference Time Update Service Profile Framework.

6.6.15.2 Data structures

- sf_ble_prf_rtus_time_updt_state_t
- sf_ble_tip_cp_change_t

6.6.15.3 Variables

- sf_ble_prf_tip_time_ctrl_point_t

6.6.15.4 sf_ble_prf_tip_time_ctrl_point_t

```
SSP_HEADER typedef uint8_t sf_ble_prf_tip_time_
```

Detailed description

Time Update Control point

6.6.15.5 API Structures

sf_ble_prf_rtus_time_updt_state_t

sf_ble_prf_rtus_time_updt_state_t

Detailed description

Reference Time Update State structure

Variables

- uint8_t current_state

  Current state of Reference time.
- uint8_t update_result

  Result of update.

sf_ble_tip_cp_change_t

sf_ble_tip_cp_change_t

Detailed description

Time Update Control Point Change Indication for Server

Variables

- sf_ble_conn_handle_t conhdl

  Connection handle.
- sf_ble_prf_tip_time_ctrl_point_t control_point_value

  Time Update Control point value.

6.6.16 SF BLE Scan Parameters Service Profile Framework Interface

RTOS-integrated SF BLE Scan Parameters Service Profile Framework Interface.

6.6.16.1 Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Scan Parameters Service Profile Framework.

6.6.16.2 Data structures

- sf_ble_prf_scps_scan_intv_t

- sf_ble_scps_scan_intv_change_t

6.6.16.3 Enumerations

- sf_ble_prf_scps_event_t

6.6.16.4 Typedefs

- sf_ble_prf_scps_scan_refresh_t

6.6.16.5 API Data

sf_ble_prf_scps_event_t

```
sf_ble_prf_scps_event_t
```

Detailed description

Profile Server user events

Enumerated values

Name	Description
SF_BLE_PRF_SCPS_EVENT_NONE	Event not supported.
SF_BLE_PRF_SCPS_EVENT_INDNTF_IND	BLE Notification Setting Change Indication.
SF_BLE_PRF_SCPS_EVENT_CHG_EVT	BLE Scan Interval Changed Indication, Refer sf_ble_scps_scan_intv_change_t.

sf_ble_prf_scps_scan_refresh_t

```
typedef uint8_t sf_ble_prf_scps_scan_refresh_t
```

Detailed description

Scan Refresh Data value

6.6.16.6 API Structures

sf_ble_prf_scps_scan_intv_t

sf_ble_prf_scps_scan_intv_t

Detailed description

Scan interval window characteristic data

Variables

- uint16_t le_scan_interval

  scan interval value
- uint16_t le_scan_window

  scan window value

sf_ble_scps_scan_intv_change_t

sf_ble_scps_scan_intv_change_t

Detailed description

Scan interval window Change Indication for Sensor

Variables

- sf_ble_conn_handle_t conhdl

  Connection handle.
- sf_ble_prf_scps_scan_intv_t scan_interval_window_val

  Scan Interval window.

6.6.17 SF BLE Time Information Profile Framework Interface

RTOS-integrated SF BLE Time Information Profile Framework Interface.

6.6.17.1 Summary

This SSP Interface provides access to the ThreadX-aware SF BLE Time Information Profile Framework.

6.6.17.2 Data structures

- sf_ble_prf_tip_write_data_t

6.6.17.3 Enumerations

- sf_ble_prf_tipc_event_t
- sf_ble_prf_tips_event_t

6.6.17.4 API Data

sf_ble_prf_tipc_event_t

`sf_ble_prf_tipc_event_t`

Detailed description

Time Profile Client events

Enumerated values

Name	Description
SF_BLE_PRF_TIPC_EVENT_NONE	Event not supported.
SF_BLE_PRF_TIPC_EVENT_READ_CHAR_RES	Read Char Complete Event.
SF_BLE_PRF_TIPC_EVENT_CURR_TIME_NTF	Current Time information received, Refer sf_ble_cts_curr_time_ntf_t.

sf_ble_prf_tips_event_t

`sf_ble_prf_tips_event_t`

Detailed description

Time Profile Server events

Enumerated values

Name	Description
SF_BLE_PRF_TIPS_EVENT_NONE	Event not supported.
SF_BLE_PRF_TIPS_EVENT_NTF_IND	CCCD Notification Received event.
SF_BLE_PRF_TIPS_EVENT_CP_IND	Time Update control point changed, Refer sf_ble_prf_tip_time_ctrl_point_t.

6.6.17.5 API Structures

sf_ble_prf_tip_write_data_t

Detailed description

Write Local Database Information

Variables

current_time

local_time

ref_time

next_dst

update_state

current_time

`sf_ble_prf_cts_curr_time_t::current_time`

Brief description

Current Time Information.

local_time

sf_ble_cts_local_time_t::local_time

Brief description

Local Time Information.

ref_time

sf_ble_cts_ref_time_t::ref_time

Brief description

Reference Time Information.

next_dst

sf_ble_prf_ndcs_time_dst_t::next_dst

Brief description

Next DST Information.

update_state

sf_ble_prf_rtus_time_updt_state_t::update_state

Brief description

Update Status Information.

6.7 Block Media Framework Interface

RTOS-integrated File system Interface to access Synergy block media devices.

The interface provides an adaption layer from the FileX I/O to block media devices.

6.7.1 Summary

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

See also FileX Interface description: FileX Port Block Media Framework

6.7.2 Interface API

sf_block_media_api_t

Function name	Description
.open	Open a device channel for read/write and control.
.read	Read data from a media channel.
.write	Write data to a media channel.
.ioctl	Send control commands to and receives the status from the media port.
.close	Close the open media channel.
.versionGet	Return the version of the driver.

6.7.3 Data structures

- sf_block_media_cfg_t
- sf_block_media_instance_t

6.7.4 Typedefs

- sf_block_media_ctrl_t

6.7.5 Defines

- #define BLOCK_MEDIA_API_VERSION_MAJOR

  Initial value: `(1U)`
- #define BLOCK_MEDIA_API_VERSION_MINOR

  Initial value: `(3U)`

6.7.6 API Data

6.7.6.1 sf_block_media_ctrl_t

```
typedef void sf_block_media_ctrl_t
```

Detailed description

Block media framework control block. Allocate an instance specific control block to pass into the block media framework API calls. Implemented as

- sf_block_media_sdmmc_instance_ctrl_t

- sf_block_media_ram_instance_ctrl_t
- sf_block_media_qspi_instance_ctrl_t

6.7.7 API Structures

6.7.7.1 sf_block_media_cfg_t

sf_block_media_cfg_t

Detailed description

Interface Configuration

Variables

- uint32_t block_size

  Block size in bytes.
- void const * p_extend

  Instance dependent configuration.

6.7.7.2 sf_block_media_api_t

sf_block_media_api_t

Detailed description

Shared Interface definition for Block Media

6.7.7.3 open

```
(* sf_block_media_api_t::open) (sf_block_media_ctrl_t *p_ctrl,
sf_block_media_cfg_t const *const p_cfg)
```

Detailed description

Open a device channel for read/write and control. Implemented as

- SF_Block_Media_SDMMC_Open
- SF_BLOCK_MEDIA_RAM_Open
- SF_BLOCK_MEDIA_QSPI_Open

Table 68: Parameters

Name	Direction	Description
p_cfg	in	Pointer to the media configuration structure for a channel.

Parameter p_cfg

Definition: sf_block_media_cfg_t const *const p_cfg

Interface Configuration

- sf_block_media_cfg_t::block_size

  Block size in bytes.

- sf_block_media_cfg_t::p_extend

  Instance dependent configuration.

6.7.7.4 read

```
(* sf_block_media_api_t::read) (sf_block_media_ctrl_t *p_ctrl, uint8_t *const
p_dest, uint32_t const start_sector, uint32_t const sector_count)
```

Detailed description

Read data from a media channel. Implemented as

- SF_Block_Media_SDMMC_Read
- SF_BLOCK_MEDIA_RAM_Read
- SF_BLOCK_MEDIA_QSPI_Read

Table 69: Parameters

Name	Direction	Description
p_cfg	in	Pointer to the media configuration structure for a channel.
p_dest	in	Destination address to read data out.
start_sector	in	Beginning sector address to read.
sector_count	in	Number of sectors to read.

Parameter p_cfg
Parameter p_dest

uint8_t

Parameter start_sector

uint32_t

Parameter sector_count

uint32_t

6.7.7.5 write

```
(* sf_block_media_api_t::write) (sf_block_media_ctrl_t *p_ctrl, uint8_t const
*const p_src, uint32_t const start_sector, uint32_t const sector_count)
```

Detailed description

Write data to a media channel. Implemented as

- SF_Block_Media_SDMMC_Write
- SF_BLOCK_MEDIA_RAM_Write
- SF_BLOCK_MEDIA_QSPI_Write

Table 70: Parameters

Name	Direction	Description
p_cfg	in	Pointer to the media configuration structure for a channel.
p_src	in	Source address of data for writing.
start_sector	in	Beginning sector address to write to.
sector_count	in	Number of sectors to write.

Parameter p_cfg
Parameter p_src

`uint8_t`

Parameter start_sector

`uint32_t`

Parameter sector_count

`uint32_t`

6.7.7.6 ioctl

```
(* sf_block_media_api_t::ioctl) (sf_block_media_ctrl_t *p_ctrl, const command,
void *p_data)
```

Detailed description

Send control commands to and receives the status from the media port. Implemented as

- SF_Block_Media_SDMMC_Control
- SF_BLOCK_MEDIA_RAM_Control
- SF_BLOCK_MEDIA_QSPI_Control

Table 71: Parameters

Name	Direction	Description
p_cfg	in	Pointer to the media configuration structure for a channel.
command	in	Command to execute.
p_data	inout	Void pointer to data in or out.

Parameter p_cfg
Parameter command
Parameter p_data

`const`

6.7.7.7 close

```
(* sf_block_media_api_t::close) (sf_block_media_ctrl_t *p_ctrl)
```

Detailed description

Close the open media channel. Implemented as

- SF_Block_Media_SDMMC_Close
- SF_BLOCK_MEDIA_RAM_Close
- SF_BLOCK_MEDIA_QSPI_Close

Table 72: Parameters

Name	Direction	Description
p_cfg	in	Pointer to the media configuration structure for a channel.

Parameter p_cfg

6.7.7.8 versionGet

```
(* sf_block_media_api_t::versionGet) ( *const p_version)
```

Detailed description

Return the version of the driver. Implemented as

- SF_Block_Media_SDMMC_VersionGet
- SF_BLOCK_MEDIA_RAM_VersionGet
- SF_BLOCK_MEDIA_QSPI_VersionGet

Table 73: Parameters

Name	Direction	Description
p_cfg	in	Pointer to the media configuration structure for a channel.
p_version	out	Memory address to return version information to.

Parameter p_cfg
Parameter p_version

6.7.7.9 sf_block_media_instance_t

sf_block_media_instance_t

Detailed description

Interface Instance

Variables

- sf_block_media_ctrl_t * p_ctrl

  Block media pointer to device driver control structure.
- sf_block_media_cfg_t const * p_cfg

  Block media pointer to device driver configuration structure.
- sf_block_media_api_t const * p_api

  Block media pointer to device driver api structure.

6.8 SF CELLULAR Framework Interface

RTOS-integrated SF CELUULAR Framework Interface.

6.8.1 Summary

This SSP Interface provides access to the ThreadX-aware SF CELLULAR Framework.

6.8.2 Interface API

sf_cellular_api_t

Function name	Description
.open	Initializes and enables the Cellular module.
.close	Disables the Cellular module.
.provisioningGet	Pointer to function to Get the Cellular module provisioning information.
.provisioningSet	Pointer to function to Set the Cellular module's provisioning information.
.infoGet	Reads the Cellular module's information.
.statisticsGet	Returns statistics information of Cellular module.
.transmit	Passes packet buffer to PPP stack for transmission.
.versionGet	Gets version and stores it in provided pointer p_version.
.networkConnect	Initiates the Data connection.
.networkDisconnect	Terminates the Data connection.
.networkStatusGet	Get Network Status information.
.simPinSet	Set SIM Pin.
.simLock	Locks SIM.
.simUnlock	Unlocks SIM.
.simIDGet	Gets the SIM ID.
.fotaCheck	Checks for Available Firmware upgrade.
.fotaStart	Starts the Firmware upgrade.
.fotaStop	Stops the Firmware upgrade.
.reset	Reset cellular module. This reset() API will only work when module is opened.

6.8.3 Data structures

- sf_cellular_provisioning_t
- sf_cellular_ctrl_t

- sf_cellular_stats_t
- sf_cellular_info_t
- sf_cellular_network_status_t
- sf_cellular_callback_args_t
- sf_cellular_op_t
- sf_cellular_at_cmd_set_t
- sf_cellular_cfg_t
- sf_cellular_instance_t

6.8.4 Enumerations

- sf_cellular_op_select_mode_t
- sf_cellular_timezone_update_mode_t
- sf_cellular_event_t
- sf_cellular_reset_type_t
- sf_cellular_airplane_mode_t
- sf_cellular_pdp_type_t
- sf_cellular_op_name_format_t
- sf_cellular_auth_type_t
- sf_cellular_at_cmd_index_t

6.8.5 Defines

- #define SF_CELLULAR_API_VERSION_MAJOR

  Initial value: `(1U)`

  Major Version of the API defined in this file
- #define SF_CELLULAR_API_VERSION_MINOR

  Initial value: `(1U)`

  Minor Version of the API defined in this file
- #define SF_CELLULAR_MAX_OPERATOR_NAME_LEN

  Initial value: `(32U)`

  Maximum length for Operator name.
- #define SF_CELLULAR_MAX_PREFFERED_OPERATOR_COUNT

  Initial value: `(5U)`

  Maximum number of preferred operator.

- #define SF_CELLULAR_IMEI_LEN

  Initial value: `(16U)`

  Maximum length of IMEI number.
- #define SF_CELLULAR_FWVERSION_LEN

  Initial value: `(32U)`

  Maximum length of Firmware Version.
- #define SF_CELLULAR_MAX_STRING_LEN

  Initial value: `(32U)`

  Maximum string length.
- #define SF_CELLULAR_CHIPSET_LEN

  Initial value: `(16U)`

  Maximum length of Chipset info.
- #define SF_CELLULAR_MFG_NAME_LEN

  Initial value: `(16U)`

  Maximum length of manufacturer.
- #define SF_CELLULAR_CID_LEN

  Initial value: `(16U)`

  Maximum length of CID.
- #define SF_CELLULAR_IMSI_LEN

  Initial value: `(24U)`

  Maximum length of IMSI.
- #define SF_CELLULAR_IP_ADDR_LEN

  Initial value: `(128U)`

  Maximum lenfth of IP address.
- #define SF_CELLULAR_TRUE

  Initial value: `(1U)`

  Logical Value TRUE for Cellular
- #define SF_CELLULAR_FALSE

  Initial value: `(0U)`

  Logical Value FALSE for Cellular

6.8.6 API Data

6.8.6.1 sf_cellular_op_select_mode_t

`sf_cellular_op_select_mode_t`

Detailed description

Operator selection mode

Enumerated values

Name	Description
SF_CELLULAR_OP_SELECT_MODE_AUTO	Automatic Operator selection.
SF_CELLULAR_OP_SELECT_MODE_MANUAL	Manual Operator selection.
SF_CELLULAR_OP_SELECT_MODE_DEREGISTER	De-register from the network.
SF_CELLULAR_OP_SELECT_MODE_MANUAL_FALLBACK	Manual with fallback to automatic.

6.8.6.2 sf_cellular_timezone_update_mode_t

`sf_cellular_timezone_update_mode_t`

Detailed description

Timezone update mode

Enumerated values

Name	Description
SF_CELLULAR_TIMEZONE_UPDATE_AUTO_DISABLE	Disable automatic time zone update.
SF_CELLULAR_TIMEZONE_UPDATE_AUTO_ENABLE	Enable automatic time zone update.

6.8.6.3 sf_cellular_event_t

`sf_cellular_event_t`

Detailed description

Cellular Framework event codes

Enumerated values

Name	Description
SF_CELLULAR_EVENT_RX	Packet received event.
SF_CELLULAR_EVENT_PROVISIONSET	Provisioning Set event.

6.8.6.4 sf_cellular_reset_type_t

`sf_cellular_reset_type_t`

Detailed description

Cellular Module reset type

Enumerated values

Name	Description
SF_CELLULAR_RESET_TYPE_SOFT	Soft reset module using AT command.
SF_CELLULAR_RESET_TYPE_HARD	Hard reset module by toggling Reset Pin.

6.8.6.5 sf_cellular_airplane_mode_t

`sf_cellular_airplane_mode_t`

Detailed description

Airplane mode

Enumerated values

Name	Description
SF_CELLULAR_AIRPLANE_MODE_OFF	Airplane mode disabled.
SF_CELLULAR_AIRPLANE_MODE_ON	Airplane mode enabled.

6.8.6.6 sf_cellular_pdp_type_t

`sf_cellular_pdp_type_t`

Detailed description

PDP type

Enumerated values

Name	Description
SF_CELLULAR_PDP_TYPE_IP	Internet protocol.
SF_CELLULAR_PDP_TYPE_PPP	Point to point protocol.
SF_CELLULAR_PDP_TYPE_IPV6	Internet protocol, version 6.
SF_CELLULAR_PDP_TYPE_IPV4V6	Virtual introduced to handle dual stack UE capability.

6.8.6.7 sf_cellular_op_name_format_t

`sf_cellular_op_name_format_t`

Detailed description

Cellular operator name format

Enumerated values

Name	Description
SF_CELLULAR_OP_NAME_FORMAT_LONG	Long alphanumeric.
SF_CELLULAR_OP_NAME_FORMAT_SHORT	Short alphanumeric.
SF_CELLULAR_OP_NAME_FORMAT_NUMERIC	Numeric.

6.8.6.8 sf_cellular_auth_type_t

`sf_cellular_auth_type_t`

Detailed description

Cellular authentication type

Enumerated values

Name	Description
SF_CELLULAR_AUTH_TYPE_NONE	No authentication.
SF_CELLULAR_AUTH_TYPE_PAP	PAP.
SF_CELLULAR_AUTH_TYPE_CHAP	CHAP.

6.8.6.9 sf_cellular_at_cmd_index_t

`sf_cellular_at_cmd_index_t`

Detailed description

Enumeration for AT command index

Enumerated values

Name	Description
SF_CELLULAR_AT_CMD_INDEX_AT	Index for Command AT.
SF_CELLULAR_AT_CMD_INDEX_ATZ0	Index for Command ATZ0.
SF_CELLULAR_AT_CMD_INDEX_AT_CREG_SET_0	Index for Command to set AT+CREG.
SF_CELLULAR_AT_CMD_INDEX_AT_CMEE_SET_0	Index for Command to set AT+CMEE.
SF_CELLULAR_AT_CMD_INDEX_AT_ECHO	Index for Command ATE.
SF_CELLULAR_AT_CMD_INDEX_AT_SAVE	Index for Command AT&W.
SF_CELLULAR_AT_CMD_INDEX_AT_CREG	Index for Command AT+CREG.
SF_CELLULAR_AT_CMD_INDEX_AT_CPIN_STATUS_GET	Index for Command to get status of SIM lock.
SF_CELLULAR_AT_CMD_INDEX_AT_ENTER_CPIN	Index for Command to unlock SIM.
SF_CELLULAR_AT_CMD_INDEX_AT_CPIN_SET	Index for Command to set SIM PIN.
SF_CELLULAR_AT_CMD_INDEX_AT_CGDCONT_SET	Index for Command to set AT+CGDCOND.
SF_CELLULAR_AT_CMD_INDEX_AT_CPOL_SET	Index for Command to set AT+CPOL.
SF_CELLULAR_AT_CMD_INDEX_AT_COPS_AUTO_SET	Index for Command to set AUTO AT+COPS.
SF_CELLULAR_AT_CMD_INDEX_AT_COPS_MANUAL_SET	Index for Command to set Manual AT+COPS.
SF_CELLULAR_AT_CMD_INDEX_AT_AIRPLANE_OFF	Index for Command to set Airplane mode OFF.
SF_CELLULAR_AT_CMD_INDEX_AT_AIRPLANE_ON	Index for Command to set Airplane mode ON.
SF_CELLULAR_AT_CMD_INDEX_AT_CONTEXT_ACTIVE	Index for Command to activate context.

Name	Description
SF_CELLULAR_AT_CMD_INDEX_AT_CONTEXT_DEACTIVE	Index for Command to deactivate context.
SF_CELLULAR_AT_CMD_INDEX_AT_CGDATA_ACTIVE	Index for Command to activate Data mode.
SF_CELLULAR_AT_CMD_INDEX_AT_CGDATA_DEACTIVE	Index for Command to deactivate Data mode.
SF_CELLULAR_AT_CMD_INDEX_AT_CSQ_GET	Index for Command to get signal quality.
SF_CELLULAR_AT_CMD_INDEX_AT_VER_GET	Index for Command to get Modem stack Version.
SF_CELLULAR_AT_CMD_INDEX_AT_CHIPSET_GET	Index for Command to get chipset details.
SF_CELLULAR_AT_CMD_INDEX_AT_IMEI_GET	Index for Command to get IMEI number.
SF_CELLULAR_AT_CMD_INDEX_AT_MANF_NAME_GET	Index for Command to get manufacturer name.
SF_CELLULAR_AT_CMD_INDEX_AT_SIMID_GET	Index for Command to get SIM Card ID.
SF_CELLULAR_AT_CMD_INDEX_AT_NET_TYPE_STATUS_GET	Index for Command to get network type information.
SF_CELLULAR_AT_CMD_INDEX_AT_NET_STATUS_GET	Index for Command to get network status information.
SF_CELLULAR_AT_CMD_INDEX_AT_LOCK_SIM	Index for Command to lock SIM card.
SF_CELLULAR_AT_CMD_INDEX_AT_UNLOCK_SIM	Index for Command to unlock SIM card.
SF_CELLULAR_AT_CMD_INDEX_AT_ENTER_DATA_MODE	Index for Command to enter data mode.
SF_CELLULAR_AT_CMD_INDEX_AT_EXIT_DATA_MODE	Index for Command to exit data mode.
SF_CELLULAR_AT_CMD_INDEX_AT_USERNAME_SET	Index for Command to set username.
SF_CELLULAR_AT_CMD_INDEX_AT_PASSWORD_SET	Index for Command to set password.
SF_CELLULAR_AT_CMD_INDEX_AT_AUTH_TYPE_SET	Index for Command to set authorisation type.

Name	Description
SF_CELLULAR_AT_CMD_INDEX_AT_AUTO_TIME_UPDATE_ENABLE_SET	Index for Command to enable auto time update.
SF_CELLULAR_AT_CMD_INDEX_AT_AUTO_TIME_UPDATE_DISABLE_SET	Index for Command to disable auto time update.
SF_CELLULAR_AT_CMD_INDEX_AT_EMPTY_APN_SET	Index for Command to set empty APN.
SF_CELLULAR_AT_CMD_INDEX_AT_GET_IP_ADDR	Index for Command to get IP address.

6.8.7 API Structures

6.8.7.1 sf_cellular_provisioning_t

sf_cellular_provisioning_t

Detailed description

Cellular Provisioning information structure

Variables

- uint8_t apn[SF_CELLULAR_MAX_STRING_LEN]

  Access Point Name.
- sf_cellular_auth_type_t auth_type

  Authentication type: PAP/CHAP.
- uint8_t username[SF_CELLULAR_MAX_STRING_LEN]

  User name used for authentication.
- uint8_t password[SF_CELLULAR_MAX_STRING_LEN]

  Password used for authentication.
- sf_cellular_airplane_mode_t airplane_mode

  Airplane mode.
- uint8_t context_id

  Context ID to be used for connection.
- sf_cellular_pdp_type_t pdp_type

  PDP Type for Context.

6.8.7.2 sf_cellular_ctrl_t

sf_cellular_ctrl_t

Detailed description

Cellular Framework control structure

Variables

- void * p_driver_handle

  Stores information required by underlying Cellular device driver.

6.8.7.3 sf_cellular_stats_t

sf_cellular_stats_t

Detailed description

The statistic and error counters for this instance

Variables

- uint32_t rx_bytes

  Bytes received successfully.

- uint32_t tx_bytes

  Bytes transmitted successfully.

- uint32_t rx_err

  Bytes receive errors.

- uint32_t tx_err

  Bytes transmit errors.

6.8.7.4 sf_cellular_info_t

sf_cellular_info_t

Detailed description

Cellular Driver Information

Variables

- uint8_t mfg_name[SF_CELLULAR_MFG_NAME_LEN]

  Manufacturer name.

- uint8_t chipset[SF_CELLULAR_CHIPSET_LEN]

  Pointer to string showing Cellular chipset/driver information.

- uint8_t fw_version[SF_CELLULAR_FWVERSION_LEN]

  Cellular firmware version.

- uint8_t imei[SF_CELLULAR_IMEI_LEN]

  IMEI number.

- uint16_t rssi

  Received signal strength indication.
- uint16_t ber

  Bit rate error.
- uint8_t ip_addr[SF_CELLULAR_IP_ADDR_LEN]

  IP address.

6.8.7.5 sf_cellular_network_status_t

sf_cellular_network_status_t

Detailed description

Network Status information

Variables

- uint16_t country_code

  Country code.
- uint16_t operator_code

  Operator code.
- int16_t rssi

  RSSI.
- uint8_t cid[SF_CELLULAR_CID_LEN]

  Cell ID.
- uint8_t imsi[SF_CELLULAR_IMSI_LEN]

  IMSI.
- uint8_t op_name[SF_CELLULAR_MAX_OPERATOR_NAME_LEN]

  Operator name.
- uint8_t service_domain

  Service Domain.
- uint8_t active_band

  Active Band.

6.8.7.6 sf_cellular_callback_args_t

sf_cellular_callback_args_t

Detailed description

Callback structure for Cellular driver to get the data receive notification

Variables

- sf_cellular_event_t event

  Event Code.
- uint8_t * p_data

  Pointer to received data.
- uint32_t length

  Receive Data Length.
- void const * p_context

  Context Provided to user during callback.

6.8.7.7 sf_cellular_op_t

sf_cellular_op_t

Detailed description

Preferred operator selection structure

Variables

- sf_cellular_op_name_format_t op_name_format

  Cellular operator name format.
- uint8_t op_name[SF_CELLULAR_MAX_OPERATOR_NAME_LEN]

  Cellular operator name.

6.8.7.8 sf_cellular_at_cmd_set_t

sf_cellular_at_cmd_set_t

Detailed description

Structure defining AT commands parameters

Variables

- uint8_t * p_cmd

  AT Command.
- uint8_t * p_success_resp

  Success response string.
- uint16_t max_resp_length

  Maximum length of expected response.
- uint8_t retry

  Retry count.
- uint16_t retry_delay

  Delay between AT command retry.

6.8.7.9 sf_cellular_cfg_t

sf_cellular_cfg_t

Detailed description

Define the Cellular configuration parameters

Variables

- sf_cellular_op_select_mode_t op_select_mode

  Cellular Operator selection mode.

- sf_cellular_op_t op

  Cellular operator. Valid when operator selection mode is manual.

- uint16_t num_pref_ops

  Number of preferred operators in the pref_ops array.

- sf_cellular_op_t pref_ops[SF_CELLULAR_MAX_PREFFERED_OPERATOR_COUNT]

  Array of structures describing preferred operators.

- sf_cellular_timezone_update_mode_t tz_upd_mode

  TimeZone update mode policy.

- uint8_t * p_sim_pin

  SIM Pin.

- uint8_t * p_puk_pin

  PUK Pin.

- ssp_err_t(* p_prov_callback)(sf_cellular_callback_args_t *p_args)

  Pointer to provisioning callback function, used in NSAL

- void(* p_recv_callback)(sf_cellular_callback_args_t *p_args)

  This is the receive callback function used by NetX which will take a data packet from the Cellular module and hand it over to NetX for further processing.

- void const * p_context

  User defined context passed into callback function.

- void const * p_extend

  Instance specific configuration.

- sf_cellular_at_cmd_set_t const * p_cmd_set

  Instance specific command set.

6.8.7.10 sf_cellular_api_t

sf_cellular_api_t

Detailed description

Cellular Framework API structure.

6.8.7.11 open

```
ssp_err_t(* sf_cellular_api_t::open) (sf_cellular_ctrl_t *p_ctrl,
sf_cellular_cfg_t const *const p_cfg)
```

Brief description

Initializes and enables the Cellular module.

Detailed description

Table 74: Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to user-provided storage for the control block.
p_cfg	in	Pointer to Cellular configuration structure.

Parameter p_ctrl

Definition: `sf_cellular_ctrl_t`

Cellular Framework control structure

Parameter p_cfg

Definition: `sf_cellular_cfg_t const *const p_cfg`

Define the Cellular configuration parameters

- `sf_cellular_cfg_t`::`sf_cellular_op_select_mode_t`

  Cellular Operator selection mode.

  Enumerated as:

  – SF_CELLULAR_OP_SELECT_MODE_AUTO

  – SF_CELLULAR_OP_SELECT_MODE_MANUAL

  – SF_CELLULAR_OP_SELECT_MODE_DEREGISTER

  – SF_CELLULAR_OP_SELECT_MODE_MANUAL_FALLBACK

- `sf_cellular_cfg_t`::`sf_cellular_op_t`

  Cellular operator. Valid when operator selection mode is manual.

- `sf_cellular_cfg_t`::`num_pref_ops`

  Number of preferred operators in the pref_ops array.

- sf_cellular_cfg_t::sf_cellular_op_t

  Array of structures describing preferred operators.
- sf_cellular_cfg_t::sf_cellular_timezone_update_mode_t

  TimeZone update mode policy.

  Enumerated as:

  – SF_CELLULAR_TIMEZONE_UPDATE_AUTO_DISABLE

  – SF_CELLULAR_TIMEZONE_UPDATE_AUTO_ENABLE
- sf_cellular_cfg_t::p_sim_pin

  SIM Pin.
- sf_cellular_cfg_t::p_puk_pin

  PUK Pin.
- sf_cellular_cfg_t::ssp_err_t

  Pointer to provisioning callback function, used in NSAL
- sf_cellular_cfg_t::p_recv_callback

  This is the receive callback function used by NetX which will take a data packet from the Cellular module and hand it over to NetX for further processing.
- sf_cellular_cfg_t::p_context

  User defined context passed into callback function.
- sf_cellular_cfg_t::p_extend

  Instance specific configuration.
- sf_cellular_cfg_t::sf_cellular_at_cmd_set_t

  Instance specific command set.

6.8.7.12 close

```
ssp_err_t(* sf_cellular_api_t::close) (sf_cellular_ctrl_t *const p_ctrl)
```

Brief description

Disables the Cellular module.

Detailed description

Table 75: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Cellular module. .

Parameter p_ctrl

Definition: sf_cellular_ctrl_t

Cellular Framework control structure

6.8.7.13 provisioningGet

```
ssp_err_t(* sf_cellular_api_t::provisioningGet) (sf_cellular_ctrl_t *const
p_ctrl, sf_cellular_provisioning_t *const p_cellular_provisioning)
```

Brief description

Pointer to function to Get the Cellular module provisioning information.

Detailed description

Table 76: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Cellular module.
p_cellular_provisioning	out	Pointer to Cellular provisioning structure.

Parameter p_ctrl

Definition: sf_cellular_ctrl_t

Cellular Framework control structure

Parameter p_cellular_provisioning

Definition: sf_cellular_provisioning_t*const p_cellular_provisioning

Cellular Provisioning information structure

- sf_cellular_provisioning_t::apn

  Access Point Name.
- sf_cellular_provisioning_t::auth_type

  Authentication type: PAP/CHAP.
- sf_cellular_provisioning_t::username

  User name used for authentication.
- sf_cellular_provisioning_t::password

  Password used for authentication.
- sf_cellular_provisioning_t::airplane_mode

  Airplane mode.

- sf_cellular_provisioning_t::context_id

  Context ID to be used for connection.

- sf_cellular_provisioning_t::pdp_type

  PDP Type for Context.

6.8.7.14 provisioningSet

```
ssp_err_t(* sf_cellular_api_t::provisioningSet) (sf_cellular_ctrl_t *const
p_ctrl, sf_cellular_provisioning_t const *const p_cellular_provisioning)
```

Brief description

Pointer to function to Set the Cellular module's provisioning information.

Detailed description

Table 77: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Cellular module.
p_cellular_provisioning	in	Pointer to Cellular provisioning structure.

Parameter p_ctrl

Definition: sf_cellular_ctrl_t

Cellular Framework control structure

Parameter p_cellular_provisioning

Definition: sf_cellular_provisioning_tconst *const p_cellular_provisioning

Cellular Provisioning information structure

- sf_cellular_provisioning_t::apn

  Access Point Name.

- sf_cellular_provisioning_t::auth_type

  Authentication type: PAP/CHAP.

- sf_cellular_provisioning_t::username

  User name used for authentication.

- sf_cellular_provisioning_t::password

  Password used for authentication.

- sf_cellular_provisioning_t::airplane_mode

  Airplane mode.

- sf_cellular_provisioning_t::context_id

  Context ID to be used for connection.

- sf_cellular_provisioning_t::pdp_type

  PDP Type for Context.

6.8.7.15 infoGet

```
ssp_err_t(* sf_cellular_api_t::infoGet) (sf_cellular_ctrl_t *const p_ctrl,
sf_cellular_info_t *const p_cellular_info)
```

Brief description

Reads the Cellular module's information.

Detailed description

Table 78: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Cellular module.
p_cellular_info	out	Pointer to Cellular info structure.

Parameter p_ctrl

Definition: sf_cellular_ctrl_t

Cellular Framework control structure

Parameter p_cellular_info

Definition: sf_cellular_info_t*const p_cellular_info

Cellular Driver Information

- sf_cellular_info_t::mfg_name

  Manufacturer name.

- sf_cellular_info_t::chipset

  Pointer to string showing Cellular chipset/driver information.

- sf_cellular_info_t::fw_version

  Cellular firmware version.

- sf_cellular_info_t::imei

  IMEI number.

- sf_cellular_info_t::rssi

  Received signal strength indication.

- sf_cellular_info_t::ber

  Bit rate error.

- sf_cellular_info_t::ip_addr

  IP address.

6.8.7.16 statisticsGet

```
ssp_err_t(* sf_cellular_api_t::statisticsGet) (sf_cellular_ctrl_t *const p_ctrl,
sf_cellular_stats_t *const p_cellular_device_stats)
```

Brief description

Returns statistics information of Cellular module.

Detailed description

Table 79: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Cellular module.
p_cellular_device_stats	out	Pointer to Cellular statistics information structure.

Parameter p_ctrl

Definition: sf_cellular_ctrl_t

Cellular Framework control structure

Parameter p_cellular_device_stats

Definition: sf_cellular_stats_t*const p_cellular_device_stats

The statistic and error counters for this instance

- sf_cellular_stats_t::rx_bytes

  Bytes received successfully.

- sf_cellular_stats_t::tx_bytes

  Bytes transmitted successfully.

- sf_cellular_stats_t::rx_err

  Bytes receive errors.

- sf_cellular_stats_t::tx_err

  Bytes transmit errors.

6.8.7.17 transmit

```
ssp_err_t(* sf_cellular_api_t::transmit) (sf_cellular_ctrl_t *const p_ctrl,
uint8_t *const p_buf, uint32_t length)
```

Brief description

Passes packet buffer to PPP stack for transmission.

Detailed description

Table 80: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Cellular module.
p_buf	in	Pointer to packet buffer to transmit
length	in	Length of packet buffer

Parameter p_ctrl

Definition: `sf_cellular_ctrl_t`

Cellular Framework control structure

Parameter p_buf

`uint8_t`

Parameter length

`uint32_t`

6.8.7.18 versionGet

```
ssp_err_t(* sf_cellular_api_t::versionGet) (ssp_version_t *const p_version)
```

Brief description

Gets version and stores it in provided pointer p_version.

Detailed description

Table 81: Parameters

Name	Direction	Description
p_version	out	pointer to memory location to return version number

Parameter p_version

6.8.7.19 networkConnect

```
ssp_err_t(* sf_cellular_api_t::networkConnect) (sf_cellular_ctrl_t *const p_ctrl)
```

Brief description

Initiates the Data connection.

Detailed description

Table 82: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Cellular module.

Parameter p_ctrl

Definition: sf_cellular_ctrl_t

Cellular Framework control structure

6.8.7.20 networkDisconnect

```
ssp_err_t(* sf_cellular_api_t::networkDisconnect) (sf_cellular_ctrl_t *const
p_ctrl)
```

Brief description

Terminates the Data connection.

Detailed description

Table 83: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Cellular module.

Parameter p_ctrl

Definition: sf_cellular_ctrl_t

Cellular Framework control structure

6.8.7.21 networkStatusGet

```
ssp_err_t(* sf_cellular_api_t::networkStatusGet) (sf_cellular_ctrl_t *const
p_ctrl, sf_cellular_network_status_t *p_network_status)
```

Brief description

Get Network Status information.

Detailed description

Table 84: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Cellular module.
p_network_status	out	Pointer to Network Status structure

Parameter p_ctrl

Definition: `sf_cellular_ctrl_t`

Cellular Framework control structure

Parameter p_network_status

Definition: `sf_cellular_network_status_t*p_network_status`

Network Status information

- `sf_cellular_network_status_t::country_code`
  Country code.
- `sf_cellular_network_status_t::operator_code`
  Operator code.
- `sf_cellular_network_status_t::rssi`
  RSSI.
- `sf_cellular_network_status_t::cid`
  Cell ID.
- `sf_cellular_network_status_t::imsi`
  IMSI.
- `sf_cellular_network_status_t::op_name`
  Operator name.
- `sf_cellular_network_status_t::service_domain`
  Service Domain.
- `sf_cellular_network_status_t::active_band`
  Active Band.

6.8.7.22 simPinSet

```
ssp_err_t(* sf_cellular_api_t::simPinSet) (sf_cellular_ctrl_t *const p_ctrl,
uint8_t *const p_old_pin, uint8_t *const p_new_pin)
```

Brief description

Set SIM Pin.

Detailed description

Table 85: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Cellular module.
p_old_pin	in	Pointer to char array containing current 4 digit pin.
p_new_pin	in	Pointer to char array containing new 4 digit pin.

Parameter p_ctrl

Definition: `sf_cellular_ctrl_t`

Cellular Framework control structure

Parameter p_old_pin

`uint8_t`

Parameter p_new_pin

`uint8_t`

6.8.7.23 simLock

```
ssp_err_t(* sf_cellular_api_t::simLock) (sf_cellular_ctrl_t *const p_ctrl,
uint8_t *const p_pin)
```

Brief description

Locks SIM.

Detailed description

Table 86: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Cellular module.
p_pin	in	PIN number to lock the SIM

Parameter p_ctrl

Definition: sf_cellular_ctrl_t

Cellular Framework control structure

Parameter p_pin

uint8_t

6.8.7.24 simUnlock

```
ssp_err_t(* sf_cellular_api_t::simUnlock) (sf_cellular_ctrl_t *const p_ctrl,
uint8_t *const p_pin)
```

Brief description

Unlocks SIM.

Detailed description

Table 87: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Cellular module.
p_pin	in	PIN number to unlock the SIM

Parameter p_ctrl

Definition: sf_cellular_ctrl_t

Cellular Framework control structure

Parameter p_pin

uint8_t

6.8.7.25 simIDGet

```
ssp_err_t(* sf_cellular_api_t::simIDGet) (sf_cellular_ctrl_t *const p_ctrl,
uint8_t *p_sim_id)
```

Brief description

Gets the SIM ID.

Detailed description

Table 88: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Cellular module.

Table 88: Parameters (Continued)

Name	Direction	Description
p_sim_id	out	SIM ID

Parameter p_ctrl

Definition: `sf_cellular_ctrl_t`

Cellular Framework control structure

Parameter p_sim_id

`uint8_t`

6.8.7.26 fotaCheck

```
ssp_err_t(* sf_cellular_api_t::fotaCheck) (sf_cellular_ctrl_t *const p_ctrl, void *p_fotacheck)
```

Brief description

Checks for Available Firmware upgrade.

Detailed description

Table 89: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Cellular module.
p_fotacheck	in	Pointer to fota check specific data structure

Parameter p_ctrl

Definition: `sf_cellular_ctrl_t`

Cellular Framework control structure

Parameter p_fotacheck

`const`

6.8.7.27 fotaStart

```
ssp_err_t(* sf_cellular_api_t::fotaStart) (sf_cellular_ctrl_t *const p_ctrl, void *p_fotastart)
```

Brief description

Starts the Firmware upgrade.

Detailed description

Table 90: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Cellular module.
p_fotastart	in	Pointer to fota start specific data structure

Parameter p_ctrl

Definition: `sf_cellular_ctrl_t`

Cellular Framework control structure

Parameter p_fotastart

`const`

6.8.7.28 fotaStop

```
ssp_err_t(* sf_cellular_api_t::fotaStop) (sf_cellular_ctrl_t *const p_ctrl, void *p_fotastop)
```

Brief description

Stops the Firmware upgrade.

Detailed description

Table 91: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Cellular module.
p_fotastop	in	Pointer to fota stop specific data structure

Parameter p_ctrl

Definition: `sf_cellular_ctrl_t`

Cellular Framework control structure

Parameter p_fotastop

`const`

6.8.7.29 reset

```
ssp_err_t(* sf_cellular_api_t::reset) (sf_cellular_ctrl_t *const p_ctrl,
sf_cellular_reset_type_t reset_type)
```

Brief description

Reset cellular module. This reset API will only work when module is opened.

Detailed description

Table 92: Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Cellular module.
reset_type	in	Reset type

Parameter p_ctrl

Definition: sf_cellular_ctrl_t

Cellular Framework control structure

Parameter reset_type

6.8.7.30 sf_cellular_instance_t

sf_cellular_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_cellular_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_cellular_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_cellular_api_t const * p_api

  Pointer to the API structure for this instance.

6.9 SF CELLULAR NSAL Framework Interface

RTOS-integrated SF CELUULAR NSAL Framework Interface.

6.9.1 Summary

This SSP Interface provides access to the ThreadX-aware SF CELLULAR NSAL Framework.

6.9.2 Functions

- sf_cellular_deinit

6.9.3 Data structures

- sf_cellular_nsal_cfg_t

6.9.4 sf_cellular_deinit

```
ssp_err_t sf_cellular_deinit (  NX_IP_DRIVER * driver_req_ptr ,
 sf_cellular_instance_t  const * p_cellular_instance ,  sf_cellular_nsal_cfg_t
* p_cellular_nsal_cfg )
```

6.9.4.1 Brief description

De-initialize the Cellular NSAL, deletes PPP interface, disconnect the network and close cellular framework.

6.9.4.2 Detailed description

Function Prototypes

Table 93: Parameters

Name	Direction	Description
p_driver_req_ptr	in	Pointer to NetX IP driver
p_cellular_instance	in	Pointer to Cellular Framework instance
p_cellular_nsal_cfg	in	Pointer to Cellular NSAL configuration structure

Table 94: Return values

Name	Description
SSP_SUCCESS	Successful

Table 94: Return values (Continued)

Name	Description
SSP_ERR_CELLULAR_FAILED	Failed to de-init Cellular Framework
SSP_ERR_NOT_OPEN	Not opened

6.9.4.3 Function steps

- Disconnect Network
- Close Cellular Driver
- Return an error to NetX.

6.9.5 API Structures

6.9.5.1 sf_cellular_nsal_cfg_t

sf_cellular_nsal_cfg_t

Detailed description

Define the NSAL configuration parameters

Variables

- uint8_t * p_ppp_stack

  PPP Stack.
- uint32_t ppp_stack_size

  PPP Stack size.
- UINT priority

  PPP Thread Priority.
- NX_PPP * p_ppp

  NetX PPP Interface.
- NX_IP * p_ip

  NetX IP Interface.
- NX_PACKET_POOL * p_ppp_packet_pool

  NetX Packet pool for internal.
- void(* p_ppp_invalid_packet_cb)(NX_PACKET *p_packet_ptr)

  Callback handler for Invalid packet.

- void(* p_ppp_send_byte)(UCHAR byte)

  PPP Send byte callback function.
- void(* p_link_down_cb)(NX_PPP *p_ppp_ptr)

  PPP Link down notification callback.

  Link Notification callback function
- void(* p_link_up_cb)(NX_PPP *p_ppp_ptr)

  PPP Link up notification callback.
- sf_cellular_auth_type_t auth_type

  Authentication Type.
- UINT(* p_chap_get_challenge_cb)(CHAR *p_rand_value, CHAR *p_id, CHAR *p_name)

  Get challenge notification callback.

  CHAP Callback Function
- UINT(* p_chap_get_responder_cb)(CHAR *p_system, CHAR *p_name, CHAR *p_secret)

  Get Responder notification callback.
- UINT(* p_chap_get_verify_cb)(CHAR *p_system, CHAR *p_name, CHAR *p_secret)

  Get Chap verification callback.
- UINT(* p_pap_generate_login)(CHAR *p_name, CHAR *p_password)

  PAP Authentication generate login callback.

  PAP Callback Function
- UINT(* p_pap_verify_login)(CHAR *p_name, CHAR *p_password)

  PAP authentication verification callback.
- uint32_t local_ip

  Local IP Address.
- uint32_t peer_ip

  Peer IP Address.
- void const * p_extend

  Instance specific configuration.

6.10 SF CELLULAR Framework Common Code

SF_Cellular Framework API Common Code.

Cellular Framework header files Framework header files for this package HAL Layer communication interface header file

6.10.1 Functions

- SF_CELLULAR_COMMON_Open
- SF_CELLULAR_COMMON_Close
- SF_CELLULAR_COMMON_InfoGet
- SF_CELLULAR_COMMON_StatisticsGet
- SF_CELLULAR_COMMON_Transmit
- SF_CELLULAR_COMMON_ProvisioningGet
- SF_CELLULAR_COMMON_ProvisioningSet
- SF_CELLULAR_COMMON_NetworkConnect
- SF_CELLULAR_COMMON_NetworkConnectWithCGDATA
- SF_CELLULAR_COMMON_NetworkDisconnect
- SF_CELLULAR_COMMON_NetworkStatusGet
- SF_CELLULAR_COMMON_SimPinSet
- SF_CELLULAR_COMMON_SimLock
- SF_CELLULAR_COMMON_SimUnlock
- SF_CELLULAR_COMMON_SimIDGet
- SF_CELLULAR_COMMON_FotaCheck
- SF_CELLULAR_COMMON_FotaStart
- SF_CELLULAR_COMMON_FotaStop

6.10.2 SF_CELLULAR_COMMON_Open

```
ssp_err_t SF_CELLULAR_COMMON_Open ( sf_cellular_ctrl_t   * p_ctrl ,
 sf_cellular_cfg_t   const *const p_cfg )
```

6.10.2.1 Brief description

Initialize Cellular driver.

6.10.2.2 Detailed description

Implements open Initializes Cellular Driver and Configure the parameters as per the p_cfg Update global variables for future use.

Table 95: Parameters

Name	Direction	Description
p_ctrl	out	Cellular control block
p_cfg	in	Cellular configuration structure

Table 96: Return values

Name	Description
SSP_SUCCESS	Driver initialization successfully.
SSP_ERR_ALREADY_OPEN	Cellular Driver is already opened.
SSP_ERR_CELLULAR_CONFIG_FAILED	Cellular module Configuration failed
SSP_ERR_CELLULAR_INIT_FAILED	Cellular module initialization failed
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Driver initialization failed
SSP_ERR_IN_USE	Device already in used

6.10.2.3 Function steps

- Get Mutex Lock
- Set Cellular configuration

6.10.3 SF_CELLULAR_COMMON_Close

```
ssp_err_t SF_CELLULAR_COMMON_Close ( sf_cellular_ctrl_t  *const p_ctrl )
```

6.10.3.1 Brief description

Stop Cellular driver functionality.

6.10.3.2 Detailed description

Implements close This function deactivates the PDP context and Update global variables for future use

Table 97: Parameters

Name	Direction	Description
p_ctrl	in	Cellular control block

Table 98: Return values

Name	Description
SSP_SUCCESS	Cellular Driver stop successfully.
SSP_ERR_NOT_OPEN	Device is not opened.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Driver un-initialization failed
SSP_ERR_IN_USE	Device already in used
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

6.10.3.3 Function steps

- Get cellular information
- Get Mutex Lock

6.10.4 SF_CELLULAR_COMMON_InfoGet

```
ssp_err_t SF_CELLULAR_COMMON_InfoGet ( sf_cellular_ctrl_t  *const p_ctrl ,
 sf_cellular_info_t  *const p_cellular_info )
```

6.10.4.1 Brief description

Get Cellular module information.

6.10.4.2 Detailed description

Implements infoGet Get Cellular module information like chipset/driver information, RSSI, noise level, link quality

Table 99: Parameters

Name	Direction	Description
p_ctrl	in	Cellular control block
p_cellular_info	in	Cellular information structure

Table 100:Return values

Name	Description
SSP_SUCCESS	Successfully get the Cellular information
SSP_ERR_NOT_OPEN	Driver not opened.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Failed reading Cellular information
SSP_ERR_IN_USE	Device already in used
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

6.10.4.3 Function steps

- Get cellular information
- Get Mutex Lock

6.10.5 SF_CELLULAR_COMMON_StatisticsGet

```
ssp_err_t SF_CELLULAR_COMMON_StatisticsGet ( sf_cellular_ctrl_t  *const p_ctrl ,
 sf_cellular_stats_t  *const p_cellular_device_stats )
```

6.10.5.1 Brief description

Get the interface statistics.

6.10.5.2 Detailed description

Implements statisticsGet Collect the statistics information of Cellular interface

Table 101:Parameters

Name	Direction	Description
p_ctrl	in	Cellular control block
p_cellular_device_stats	in	Cellular statistic structure

Table 102:Return values

Name	Description
SSP_SUCCESS	Successfully get the Statistics information.
SSP_ERR_UNSUPPORTED	Functionality is not supported.
SSP_ERR_NOT_OPEN	Device not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Failed Reading statistics information
SSP_ERR_IN_USE	Device already in used
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

6.10.5.3 Function steps

- Get cellular information
- Get Mutex Lock

6.10.6 SF_CELLULAR_COMMON_Transmit

```
ssp_err_t SF_CELLULAR_COMMON_Transmit ( sf_cellular_ctrl_t  *const p_ctrl ,
  uint8_t *const p_buf ,   uint32_t length )
```

6.10.6.1 Brief description

Passes packet buffer to PPP stack for transmission.

6.10.6.2 Detailed description

Implements transmit Send packet buffer to PPP stack

Table 103:Parameters

Name	Direction	Description
p_ctrl	in	Cellular control block
p_buf	in	transmit byte buffer pointer
length	in	Length of data

Table 104:Return values

Name	Description
SSP_SUCCESS	Successfully send the packet buffer.
SSP_ERR_NOT_OPEN	Device not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Transmitting Data failed
SSP_ERR_IN_USE	Device already in used
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

6.10.6.3 Function steps

- Get cellular information
- Get Mutex Lock

6.10.7 SF_CELLULAR_COMMON_ProvisioningGet

```
ssp_err_t SF_CELLULAR_COMMON_ProvisioningGet ( sf_cellular_ctrl_t
*const p_ctrl ,  sf_cellular_provisioning_t  *const p_cellular_provisioning )
```

6.10.7.1 Brief description

Reads the provisioning information.

6.10.7.2 Detailed description

Implements provisioningGet Reads Cellular's provisioning information

Table 105:Parameters

Name	Direction	Description
p_ctrl	in	Cellular control block
p_cellular_provisioning	out	Cellular provisioning structure

Table 106:Return values

Name	Description
SSP_SUCCESS	Successfully read the provisioning information.
SSP_ERR_NOT_OPEN	Device not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Reading provisioning information failed
SSP_ERR_IN_USE	Device already in used
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

6.10.7.3 Function steps

- Get cellular information
- Get Mutex Lock
- Copy provisioning info to driver structure

6.10.8 SF_CELLULAR_COMMON_ProvisioningSet

```
ssp_err_t SF_CELLULAR_COMMON_ProvisioningSet ( sf_cellular_ctrl_t
*const p_ctrl ,  sf_cellular_provisioning_t  const
*const p_cellular_provisioning )
```

6.10.8.1 Brief description

Sets the provisioning information.

6.10.8.2 Detailed description

Implements provisioningSet Sets Cellular's provisioning information

Table 107:Parameters

Name	Direction	Description
p_ctrl	in	Cellular control block
p_cellular_provisioning	in	Cellular provisioning structure

Table 108:Return values

Name	Description
SSP_SUCCESS	Successfully set the provisioning information.
SSP_ERR_NOT_OPEN	Device not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Provisioning configuration failed
SSP_ERR_IN_USE	Device already in used
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

6.10.8.3 Function steps

- Get cellular information
- Get Mutex Lock

6.10.9 SF_CELLULAR_COMMON_NetworkConnect

```
ssp_err_t SF_CELLULAR_COMMON_NetworkConnect ( sf_cellular_ctrl_t
*const p_ctrl )
```

6.10.9.1 Brief description

Establishes the Data connection.

6.10.9.2 Detailed description

Implements networkConnect Establishes the Network Connection

Table 109:Parameters

Name	Direction	Description
p_ctrl	in	Cellular control block

Table 110:Return values

Name	Description
SSP_SUCCESS	Successfully establishes the Network connection
SSP_ERR_NOT_OPEN	Device not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Failed to establish the Network Connection
SSP_ERR_IN_USE	Device already in used
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

6.10.9.3 Function steps

- Get cellular information
- Get Mutex Lock
- Enter Data Mode

6.10.10 SF_CELLULAR_COMMON_NetworkConnectWithCGDATA

```
ssp_err_t SF_CELLULAR_COMMON_NetworkConnectWithCGDATA ( sf_cellular_ctrl_t
*const p_ctrl )
```

6.10.10.1 Brief description

Establishes the Data connection using AT+CGDATA command. Limitation: This API activates only one PDP Context Id even though AT+CGDATA command actually supports activation of multiple Context Ids in single command.

6.10.10.2 Detailed description

Implements networkConnect Establishes the Network Connection

Table 111:Parameters

Name	Direction	Description
p_ctrl	in	Cellular control block

Table 112:Return values

Name	Description
SSP_SUCCESS	Successfully establishes the Network connection
SSP_ERR_NOT_OPEN	Device not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_CELLULAR_FAILED	Failed to establish the Network Connection
SSP_ERR_IN_USE	Device already in used
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

6.10.10.3 Function steps

- Get cellular information
- Get Mutex Lock
- Enter Data Mode

6.10.11 SF_CELLULAR_COMMON_NetworkDisconnect

```
ssp_err_t SF_CELLULAR_COMMON_NetworkDisconnect ( sf_cellular_ctrl_t
*const p_ctrl )
```

6.10.11.1 Brief description

Terminates the connection.

6.10.11.2 Detailed description

Implements networkDisconnect

Table 113:Parameters

Name	Direction	Description
p_ctrl	in	Cellular control block

Table 114:Return values

Name	Description
SSP_SUCCESS	Successfully disconnect the connection
SSP_ERR_NOT_OPEN	Cellular driver is not opened
SSP_ERR_CELLULAR_FAILED	Failed to terminate the network connection
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

6.10.11.3 Function steps

- Get cellular information
- Get Mutex Lock
- Exit Data Mode

6.10.12 SF_CELLULAR_COMMON_NetworkStatusGet

```
ssp_err_t SF_CELLULAR_COMMON_NetworkStatusGet ( sf_cellular_ctrl_t
*const p_ctrl ,  sf_cellular_network_status_t  * p_network_status )
```

6.10.12.1 Brief description

Get Network Status information.

6.10.12.2 Detailed description

Implements networkStatusGet

Table 115:Parameters

Name	Direction	Description
p_ctrl	in	Cellular control block
p_network_status	out	Cellular network structure

Table 116:Return values

Name	Description
SSP_SUCCESS	Successfully read the Network status information
SSP_ERR_NOT_OPEN	Cellular driver is not opened
SSP_ERR_CELLULAR_FAILED	Failed reading Network Status information.
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

6.10.12.3 Function steps

- Get cellular information
- Get Mutex Lock

6.10.13 SF_CELLULAR_COMMON_SimPinSet

```
ssp_err_t SF_CELLULAR_COMMON_SimPinSet ( sf_cellular_ctrl_t  *const p_ctrl ,
  uint8_t *const p_old_pin ,   uint8_t *const p_new_pin )
```

6.10.13.1 Brief description

Set the SIM PIN.

6.10.13.2 Detailed description

Implements sf_cellular_api_t::simSetPin

Table 117:Parameters

Name	Direction	Description
p_ctrl	in	Cellular control block
p_old_pin	in	Old SIM Pin
p_new_pin	in	New SIM Pin

Table 118:Return values

Name	Description
SSP_SUCCESS	Successfully set SIM Pin
SSP_ERR_NOT_OPEN	Cellular driver is not opened
SSP_ERR_CELLULAR_FAILED	Failed to set the SIM Pin
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

6.10.13.3 Function steps

- Get cellular information
- Get Mutex Lock

6.10.14 SF_CELLULAR_COMMON_SimLock

```
ssp_err_t SF_CELLULAR_COMMON_SimLock ( sf_cellular_ctrl_t  *const p_ctrl ,
  uint8_t *const p_pin )
```

6.10.14.1 Brief description

Lock the SIM.

6.10.14.2 Detailed description

Implements simLock

Table 119:Parameters

Name	Direction	Description
p_ctrl	in	Cellular control block
p_pin	in	SIM Pin

Table 120:Return values

Name	Description
SSP_SUCCESS	Successfully lock the SIM
SSP_ERR_NOT_OPEN	Cellular driver is not opened
SSP_ERR_CELLULAR_FAILED	Failed to Lock the SIM
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

6.10.14.3 Function steps

- Get cellular information
- Get Mutex Lock

6.10.15 SF_CELLULAR_COMMON_SimUnlock

```
ssp_err_t SF_CELLULAR_COMMON_SimUnlock ( sf_cellular_ctrl_t  *const p_ctrl ,
  uint8_t *const p_pin )
```

6.10.15.1 Brief description

Unlock the SIM.

6.10.15.2 Detailed description

Implements simUnlock

Table 121:Parameters

Name	Direction	Description
p_ctrl	in	Cellular control block
p_pin	in	SIM Pin

Table 122:Return values

Name	Description
SSP_SUCCESS	Successfully unlock the SIM
SSP_ERR_NOT_OPEN	Cellular driver is not opened
SSP_ERR_CELLULAR_FAILED	Failed to unlock the SIM
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

6.10.15.3 Function steps

- Get cellular information
- Get Mutex Lock

6.10.16 SF_CELLULAR_COMMON_SimIDGet

```
ssp_err_t SF_CELLULAR_COMMON_SimIDGet ( sf_cellular_ctrl_t  *const p_ctrl ,
  uint8_t * p_sim_id )
```

6.10.16.1 Brief description

Get SIM ID.

6.10.16.2 Detailed description

Implements sf_cellular_api_t::simGetID Get SIM ID

Table 123:Parameters

Name	Direction	Description
p_ctrl	in	Cellular control block
p_sim_id	out	SIM ID

Table 124:Return values

Name	Description
SSP_SUCCESS	Successfully get the SIM ID.
SSP_ERR_CELLULAR_FAILED	Failed to get the SIM ID
SSP_ERR_NOT_OPEN	Device is not opened
SSP_ERR_ASSERTION	Argument NULL is passed
SSP_ERR_IN_USE	Device already in use
SSP_ERR_CELLULAR_INVALID_STATE	Module in Data mode can't send AT command

6.10.16.3 Function steps

- Get cellular information

6.10.17 SF_CELLULAR_COMMON_FotaCheck

```
ssp_err_t SF_CELLULAR_COMMON_FotaCheck ( sf_cellular_ctrl_t  *const p_ctrl ,
  void * p_fotacheck )
```

6.10.17.1 Brief description

Checks for available firmware upgrade.

6.10.17.2 Detailed description

Implements fotaCheck Checks for available firmware upgrade.

Table 125:Parameters

Name	Direction	Description
p_ctrl	in	Cellular control block
p_fotacheck	in	Fota check specific structure

Table 126:Return values

Name	Description
SSP_ERR_UNSUPPORTED	Functionality not supported

6.10.18 SF_CELLULAR_COMMON_FotaStart

```
ssp_err_t SF_CELLULAR_COMMON_FotaStart ( sf_cellular_ctrl_t  *const p_ctrl ,
  void * p_fotastart )
```

6.10.18.1 Brief description

Perform the firmware upgrade.

6.10.18.2 Detailed description

Implements fotaStart Perform the firmware upgrade.

Table 127:Parameters

Name	Direction	Description
p_ctrl	in	Cellular control block
p_fotastart	in	Fota start specific structure

Table 128:Return values

Name	Description
SSP_ERR_UNSUPPORTED	Functionality not supported

6.10.19 SF_CELLULAR_COMMON_FotaStop

```
ssp_err_t SF_CELLULAR_COMMON_FotaStop ( sf_cellular_ctrl_t  *const p_ctrl ,
  void * p_fotastop )
```

6.10.19.1 Brief description

Stop firmware upgrade.

6.10.19.2 Detailed description

Implements fotaStop Stop firmware upgrade

Table 129:Parameters

Name	Direction	Description
p_ctrl	in	Cellular control block
p_fotastop	in	Fota stop specific structure

Table 130:Return values

Name	Description
SSP_ERR_UNSUPPORTED	Functionality not supported

6.11 Communications Framework Interface

RTOS-integrated communications Framework Interface.

Implemented by:

- UART Framework Instance - UART implementation
- USB Communication Framework - USBX CDC ACM device implementation
- Telnet Communication Framework - NetX telnet server implementation
- Telnet Communication Framework - NetX telnet server with shared IP Instance implementation

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

See also Framework Communications Interface description: Communications Framework

6.11.1 Interface API

sf_comms_api_t

Function name	Description
.open	Initialize communications driver.
.close	Clean up communications driver.
.read	Read data from communications driver. This call will return after the number of bytes requested is read or if a timeout occurs while waiting for access to the driver.
.write	Write data to communications driver. This call will return after all bytes are written or if a timeout occurs while waiting for access to the driver.
.lock	Lock the communications driver. Reserve exclusive access to the communications driver.
.unlock	Unlock the communications driver. Release exclusive access to the communications driver.
.versionGet	Store the driver version in the provided p_version.

6.11.2 Data structures

- sf_comms_cfg_t
- sf_comms_instance_t

6.11.3 Enumerations

- sf_comms_lock_t

6.11.4 Typedefs

- sf_comms_ctrl_t

6.11.5 Defines

- #define SF_COMMS_API_VERSION_MAJOR

  Initial value: `(1U)`

  Version of the API defined in this file
- #define SF_COMMS_API_VERSION_MINOR

  Initial value: `(5U)`

6.11.6 API Data

6.11.6.1 sf_comms_lock_t

```
sf_comms_lock_t
```

Detailed description

Communications locks

Enumerated values

Name	Description
SF_COMMS_LOCK_TX	Lock Transmit.
SF_COMMS_LOCK_RX	Lock Receive.
SF_COMMS_LOCK_ALL	Lock Transmit and Receive.

6.11.6.2 sf_comms_ctrl_t

```
typedef void sf_comms_ctrl_t
```

Detailed description

Communications framework control block. Allocate an instance specific control block to pass into the communications framework API calls. Implemented as

- sf_console_instance_ctrl_t

6.11.7 API Structures

6.11.7.1 sf_comms_cfg_t

sf_comms_cfg_t

Detailed description

Configuration for RTOS integrated communications driver

Variables

- void const * p_extend

  Pointer to lower level communications control structure.

6.11.7.2 sf_comms_api_t

sf_comms_api_t

Detailed description

Framework communications API structure. Implementations will use the following API.

6.11.7.3 open

```
ssp_err_t(* sf_comms_api_t::open) (sf_comms_ctrl_t *const p_ctrl, sf_comms_cfg_t
const *const p_cfg)
```

Detailed description

Initialize communications driver.

Table 131:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a control structure allocated by user. The control structure is initialized in this function.
p_cfg	in	Pointer to configuration structure. All elements of the structure must be set by user.

Parameter p_ctrl

Definition: `sf_comms_ctrl_t*const p_ctrl`

Communications framework control block. Allocate an instance specific control block to pass into the communications framework API calls. Implemented assf_console_instance_ctrl_t

Parameter p_cfg

Definition: `sf_comms_cfg_t const *const p_cfg`

Configuration for RTOS integrated communications driver

- sf_comms_cfg_t::p_extend

  Pointer to lower level communications control structure.

6.11.7.4 close

```
ssp_err_t(* sf_comms_api_t::close) (sf_comms_ctrl_t *const p_ctrl)
```

Detailed description

Clean up communications driver.

Table 132:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for communications driver.

Parameter p_ctrl

Definition: sf_comms_ctrl_t*const p_ctrl

Communications framework control block. Allocate an instance specific control block to pass into the communications framework API calls. Implemented assf_console_instance_ctrl_t

6.11.7.5 read

```
ssp_err_t(* sf_comms_api_t::read) (sf_comms_ctrl_t *const p_ctrl, uint8_t *const
p_dest, uint32_t const bytes, UINT const timeout)
```

Detailed description

Read data from communications driver. This call will return after the number of bytes requested is read or if a timeout occurs while waiting for access to the driver.

Table 133:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for communications driver.
p_dest	in	Destination address to read data out
bytes	in	Read data length

Table 133:Parameters (Continued)

Name	Direction	Description
timeout	in	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

Parameter p_ctrl

Definition: `sf_comms_ctrl_t*const p_ctrl`

Communications framework control block. Allocate an instance specific control block to pass into the communications framework API calls. Implemented assf_console_instance_ctrl_t

Parameter p_dest

`uint8_t`

Parameter bytes

`uint32_t`

Parameter timeout

`const`

6.11.7.6 write

```
ssp_err_t(* sf_comms_api_t::write) (sf_comms_ctrl_t *const p_ctrl, uint8_t const
*const p_src, uint32_t const bytes, UINT const timeout)
```

Detailed description

Write data to communications driver. This call will return after all bytes are written or if a timeout occurs while waiting for access to the driver.

Table 134:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for communications driver.
p_src	in	Source address to read data out from
bytes	in	Write data length

Table 134:Parameters (Continued)

Name	Direction	Description
timeout	in	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

Parameter p_ctrl

Definition: `sf_comms_ctrl_t*const p_ctrl`

Communications framework control block. Allocate an instance specific control block to pass into the communications framework API calls. Implemented assf_console_instance_ctrl_t

Parameter p_src

`uint8_t`

Parameter bytes

`uint32_t`

Parameter timeout

`const`

6.11.7.7 lock

```
ssp_err_t(* sf_comms_api_t::lock) (sf_comms_ctrl_t *const p_ctrl, sf_comms_lock_t
lock_type, UINT timeout)
```

Detailed description

Lock the communications driver. Reserve exclusive access to the communications driver.

Table 135:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for communications driver.
lock_type	in	Locking type, transmission channel or reception channel

Table 135:Parameters (Continued)

Name	Direction	Description
timeout	in	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

Parameter p_ctrl

Definition: `sf_comms_ctrl_t*const p_ctrl`

Communications framework control block. Allocate an instance specific control block to pass into the communications framework API calls. Implemented assf_console_instance_ctrl_t

Parameter lock_type

Definition: `sf_comms_lock_tlock_type`

Communications locks

Parameter timeout

`const`

6.11.7.8 unlock

```
ssp_err_t(* sf_comms_api_t::unlock) (sf_comms_ctrl_t *const p_ctrl,
sf_comms_lock_t lock_type)
```

Detailed description

Unlock the communications driver. Release exclusive access to the communications driver.

Table 136:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for communications driver.
lock_type	in	Locking type, transmission channel or reception channel

Parameter p_ctrl

Definition: `sf_comms_ctrl_t*const p_ctrl`

Communications framework control block. Allocate an instance specific control block to pass into the communications framework API calls. Implemented assf_console_instance_ctrl_t

Parameter lock_type

Definition: sf_comms_lock_tlock_type

Communications locks

6.11.7.9 versionGet

```
ssp_err_t(* sf_comms_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Store the driver version in the provided p_version.

Table 137:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for communications driver.
p_version	in	Pointer to memory version to be stored.

Parameter p_ctrl

Parameter p_version

6.11.7.10 sf_comms_instance_t

sf_comms_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_comms_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- sf_comms_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- sf_comms_api_t const * p_api

  Pointer to the API structure for this instance.

6.12 Console Framework Interface

RTOS-integrated Console Framework Interface.

6.12.1 Summary

This module is a ThreadX-aware Console Framework.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

See also Console Interface description Console Framework

6.12.2 Interface API

sf_console_api_t

Function name	Description
.open	This function configures the console. This function must be called before any other console functions.
.close	The close API handles cleans up internal driver data.
.prompt	Prints prompt string from menu, waits for input, parses input based on menu, and calls callback function if a command is identified.
.parse	Looks for input string in menu, and calls callback function if found.
.read	Reads data into the destination byte by byte and echos input to the console. Backspace, delete, and left/right arrow keys supported. Read completes when a line ending CR, CR+LF, or CR+NULL is received, or when the input exceeds the number of bytes input. If the buffer overflows SF_CONSOLE_MAX_INPUT_LENGTH, read will return an error code.
.write	The write API gets mutex object and handles UART data transmission at UART HAL layer. gets event flag to synchronize to completion of data transfer.

Function name	Description
.argumentFind	Finds a command line argument in an input string and returns the index of the character immediately following the argument and any string numbers converted to integers.
.versionGet	Stores version information in provided pointer.

6.12.3 Data structures

- sf_console_callback_args_t
- sf_console_command_t
- sf_console_menu_t
- sf_console_cfg_t
- sf_console_instance_t

6.12.4 Typedefs

- sf_console_ctrl_t
- sf_console_cb_args_t

6.12.5 Defines

- #define SF_CONSOLE_API_VERSION_MAJOR

  Initial value: `(1U)`

  Version of the API defined in this file
- #define SF_CONSOLE_API_VERSION_MINOR

  Initial value: `(4U)`
- #define SF_CONSOLE_HELP_COMMAND

  Initial value: `((uint8_t *) "?")`

  Command to print each command and help in menu
- #define SF_CONSOLE_MENU_PREVIOUS_COMMAND

  Initial value: `((uint8_t *) "^")`

  Previous command

- #define SF_CONSOLE_ROOT_MENU_COMMAND

  Initial value: `((uint8_t *) "~")`

  Root menu command

- #define SF_CONSOLE_CALLBACK_NEXT_FUNCTION

  Initial value: `((void(*)(sf_console_callback_args_t * p_args)) 0x70000000)`

  Use this macro to access the next menu layer from this command.

6.12.6 API Data

6.12.6.1 sf_console_ctrl_t

```
typedef void sf_console_ctrl_t
```

Detailed description

Console framework control block. Allocate an instance specific control block to pass into the console framework API calls. Implemented as

- sf_console_instance_ctrl_t

6.12.6.2 sf_console_cb_args_t

```
typedef sf_console_callback_args_t sf_console_cb_args_t
```

Detailed description

DEPRECATED definition, please use sf_console_callback_args_t instead.

6.12.7 API Structures

6.12.7.1 sf_console_callback_args_t

sf_console_callback_args_t

Detailed description

Console callback arguments

Variables

- sf_console_ctrl_t * p_ctrl

  Pointer to console that received the command that caused this callback.

- uint8_t const * p_remaining_string

  String remaining after parsing command.

- uint8_t const * context

  Pointer to user provided data.

- uint32_t bytes

  The number of bytes remaining in the input string.

6.12.7.2 sf_console_command_t

sf_console_command_t

Detailed description

Console command structure, used to create a console menu with associated callbacks.

Variables

- uint8_t * command

  Command string.

- uint8_t * help

  Description of command.

- void(* callback)(sf_console_callback_args_t *p_args)

  Callback to call when command is selected.

- void const * context

  User provided context passed into callback.

6.12.7.3 sf_console_menu_t

sf_console_menu_t

Detailed description

Console menu structure.

Variables

- struct st_sf_console_menu const * menu_prev

  Previous menu.

- uint8_t const * menu_name

  Menu name, used as a prompt.

- uint32_t num_commands

  Number of commands in this menu.

- sf_console_command_t const * command_list

  Pointer to an array of commands of length num_commands.

6.12.7.4 sf_console_cfg_t

sf_console_cfg_t

Detailed description

Configuration for RTOS integrated console framework.

Variables

- sf_comms_instance_t const * p_comms

  Pointer to communications driver instance.

- sf_console_menu_t const * p_initial_menu

  First menu to print during Open.

- bool echo

  Whether to echo input commands to transmitter.

- bool autostart

  If true, prompt will occur with p_initial_menu after initialization.

6.12.7.5 sf_console_api_t

sf_console_api_t

Detailed description

Console framework API structure. Console implementations will use the following API.

6.12.7.6 open

```
(* sf_console_api_t::open) (sf_console_ctrl_t *const p_ctrl, sf_console_cfg_t
const *const p_cfg)
```

Brief description

This function configures the console. This function must be called before any other console functions.

Detailed description

Implemented as

- SF_CONSOLE_Open

Table 138:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a device structure allocated by user. The device control structure is initialized in this function.
p_cfg	in	Pointer to configuration structure. All elements of the structure must be set by user.

Parameter p_ctrl

Definition: sf_console_ctrl_t*const p_ctrl

Console framework control block. Allocate an instance specific control block to pass into the console framework API calls. Implemented assf_console_instance_ctrl_t

Parameter p_cfg

Definition: sf_console_cfg_t const *const p_cfg

Configuration for RTOS integrated console framework.

- sf_console_cfg_t::sf_comms_instance_t

  Pointer to communications driver instance.

- sf_console_cfg_t::sf_console_menu_t

  First menu to print during Open.

- sf_console_cfg_t::echo

  Whether to echo input commands to transmitter.

- sf_console_cfg_t::autostart

  If true, prompt will occur with p_initial_menu after initialization.

6.12.7.7 close

(* sf_console_api_t::close) (sf_console_ctrl_t *const p_ctrl)

Brief description

The close API handles cleans up internal driver data.

Detailed description

Implemented as

- SF_CONSOLE_Close

Table 139:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for UART driver.

Parameter p_ctrl

Definition: sf_console_ctrl_t*const p_ctrl

Console framework control block. Allocate an instance specific control block to pass into the console framework API calls. Implemented assf_console_instance_ctrl_t

6.12.7.8 prompt

```
(* sf_console_api_t::prompt) (sf_console_ctrl_t *const p_ctrl, sf_console_menu_t
const *const p_menu, UINT const timeout)
```

Brief description

Prints prompt string from menu, waits for input, parses input based on menu, and calls callback function if a command is identified.

Detailed description

Implemented as

- SF_CONSOLE_Prompt

Table 140:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for UART driver.
p_menu	in	Set to NULL to stay on current menu maintained by the console framework. To change menus, pass a pointer to the new menu.
timeout	in	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

Parameter p_ctrl

Definition: `sf_console_ctrl_t*const p_ctrl`

Console framework control block. Allocate an instance specific control block to pass into the console framework API calls. Implemented assf_console_instance_ctrl_t

Parameter p_menu

Definition: `sf_console_menu_tconst *const p_menu`

Console menu structure.

- `sf_console_menu_t::menu_prev`

  Previous menu.

- `sf_console_menu_t::menu_name`

  Menu name, used as a prompt.

- sf_console_menu_t::num_commands

  Number of commands in this menu.

- sf_console_menu_t::command_list

  Pointer to an array of commands of length num_commands.

Parameter timeout

const

6.12.7.9 parse

```
(* sf_console_api_t::parse) (sf_console_ctrl_t *const p_ctrl, sf_console_menu_t
const *const p_cmd_list, uint8_t const *const p_input, uint32_t const bytes)
```

Brief description

Looks for input string in menu, and calls callback function if found.

Detailed description

Implemented as

- SF_CONSOLE_Parse

Table 141:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for UART driver.
p_cmd_list	in	Pointer to a menu of valid input commands for this prompt
p_input	in	Pointer to a null terminated string to search for in the command list
bytes	in	Length of the input string.

Parameter p_ctrl

Definition: sf_console_ctrl_t*const p_ctrl

Console framework control block. Allocate an instance specific control block to pass into the console framework API calls. Implemented assf_console_instance_ctrl_t

Parameter p_cmd_list

Definition: sf_console_menu_tconst *const p_cmd_list

Console menu structure.

- sf_console_menu_t::menu_prev

  Previous menu.

- sf_console_menu_t::menu_name

  Menu name, used as a prompt.

- sf_console_menu_t::num_commands

  Number of commands in this menu.

- sf_console_menu_t::command_list

  Pointer to an array of commands of length num_commands.

Parameter p_input

`uint8_t`

Parameter bytes

`uint32_t`

6.12.7.10 read

```
(* sf_console_api_t::read) (sf_console_ctrl_t *const p_ctrl, uint8_t *const
p_dest, uint32_t const bytes, uint32_t const timeout)
```

Brief description

Reads data into the destination byte by byte and echos input to the console. Backspace, delete, and left/right arrow keys supported. Read completes when a line ending CR, CR+LF, or CR+NULL is received, or when the input exceeds the number of bytes input. If the buffer overflows SF_CONSOLE_MAX_INPUT_LENGTH, read will return an error code.

Detailed description

Implemented as

- SF_CONSOLE_Read

Table 142:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for UART driver.
p_dest	in	Destination address to read data out
bytes	in	Read data length

Table 142:Parameters (Continued)

Name	Direction	Description
timeout	in	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

Parameter p_ctrl

Definition: `sf_console_ctrl_t*const p_ctrl`

Console framework control block. Allocate an instance specific control block to pass into the console framework API calls. Implemented assf_console_instance_ctrl_t

Parameter p_dest

`uint8_t`

Parameter bytes

`uint32_t`

Parameter timeout

`uint32_t`

6.12.7.11 write

```
(* sf_console_api_t::write) (sf_console_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const timeout)
```

Brief description

The write API gets mutex object and handles UART data transmission at UART HAL layer. gets event flag to synchronize to completion of data transfer.

Detailed description

Implemented as

- SF_CONSOLE_Write

Table 143:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for UART driver.

Table 143:Parameters (Continued)

Name	Direction	Description
p_src	in	Pointer to a NULL terminated string. Length must be less than SF_CONSOLE_MAX_WRITE_LENGTH.
timeout	in	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

Parameter p_ctrl

Definition: `sf_console_ctrl_t*const p_ctrl`

Console framework control block. Allocate an instance specific control block to pass into the console framework API calls. Implemented assf_console_instance_ctrl_t

Parameter p_src

`uint8_t`

Parameter timeout

`uint32_t`

6.12.7.12 argumentFind

```
(* sf_console_api_t::argumentFind) (uint8_t const *const p_arg, uint8_t const
*const p_str, int32_t *const p_index, int32_t *const p_data)
```

Brief description

Finds a command line argument in an input string and returns the index of the character immediately following the argument and any string numbers converted to integers.

Detailed description

Implemented as

- SF_CONSOLE_ArgumentFind

Table 144:Parameters

Name	Direction	Description
p_arg	in	Pointer to argument to find.
p_src	in	Pointer to source string to find the argument in.

Table 144:Parameters (Continued)

Name	Direction	Description
p_index	out	Pointer to location to store index. Set to -1 if argument is not found in input string. Pass NULL if index is not requested.
p_data	out	Pointer to location to store data following the argument. Set to -1 if argument is not found in input string. Pass NULL if data is not requested.

Parameter p_arg

`uint8_t`

Parameter p_src
Parameter p_index
Parameter p_data

6.12.7.13 versionGet

```
(* sf_console_api_t::versionGet) ( *const p_version)
```

Brief description

Stores version information in provided pointer.

Detailed description

Implemented as

- SF_CONSOLE_VersionGet

Table 145:Parameters

Name	Direction	Description
p_version	out	Code and API version used stored here.

Parameter p_version

6.12.7.14 sf_console_instance_t

sf_console_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_console_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_console_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_console_api_t const * p_api

  Pointer to the API structure for this instance.

6.13 SSP Crypto Framework Common Module Interface

Interface definition for Synergy Crypto Framework module.
Interface definition for Synergy Crypto Key Framework module.
Interface definition for Synergy Crypto HASH Framework module.
Interface definition for Synergy Crypto Framework Common Module.
Interface definition for Synergy Crypto Cipher Framework module.

6.13.1 Summary

This is the Interface of SF_CRYPTO Framework module.
SF_CRYPTO Interface description: Crypto Framework
SF_CRYPTO_CIPHER Interface description: Crypto Framework

6.13.2 Summary

This is the Interface of SF_CRYPTO Framework module.
SF_CRYPTO Interface description: Crypto Framework

6.13.3 Summary

This is the Interface of SF_CRYPTO KEY Framework module.
SF_CRYPTO_HASH Interface description: Crypto Framework

6.13.4 Summary

This is the Interface of SF_CRYPTO KEY Framework module.
SF_CRYPTO_KEY Interface description: Crypto Framework

6.13.5 Functions

- SF_CRYPTO_Open
- SF_CRYPTO_Close
- SF_CRYPTO_Lock
- SF_CRYPTO_Unlock
- SF_CRYPTO_StatusGet
- SF_CRYPTO_VersionGet

6.13.6 Typedefs

- sf_crypto_ctrl_t

6.13.7 Defines

- #define SF_CRYPTO_API_VERSION_MAJOR

  Initial value: `(1U)`

  The API version of SSP Crypto Framework Common Module
- #define SF_CRYPTO_API_VERSION_MINOR

  Initial value: `(1U)`
- #define SF_CRYPTO_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  The API version of SSP Crypto Framework Common Module
- #define SF_CRYPTO_CODE_VERSION_MINOR

  Initial value: `(1U)`
- #define SF_CRYPTO_CIPHER_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  The API version of SSP Crypto Cipher Framework
- #define SF_CRYPTO_CIPHER_CODE_VERSION_MINOR

  Initial value: `(0U)`
- #define NUM_BYTES_IN_WORD

  Initial value: `(4U)`
- #define SF_CRYPTO_CIPHER_API_VERSION_MAJOR

  Initial value: `(1U)`

  The API version of SSP Crypto CIPHER Framework

- #define SF_CRYPTO_CIPHER_API_VERSION_MINOR

  Initial value: `(0U)`
- #define SF_CRYPTO_CIPHER_AES_128_XTS_KEY_SIZE

  Initial value: `(2 * (#define AES128_SECRET_KEY_SIZE_BYTES))`

  Macros for AES operations
- #define SF_CRYPTO_CIPHER_AES_256_XTS_KEY_SIZE

  Initial value: `(2 * (#define AES256_SECRET_KEY_SIZE_BYTES))`
- #define SF_CRYPTO_CIPHER_AES_GCM_TAG_LENGTH_16_BYTES

  Initial value: `(16U)`

  AES GCM tag of length 16 bytes.
- #define SF_CRYPTO_CIPHER_AES_IV_LENGTH_12_BYTES

  Initial value: `(12U)`

  IV for AES operations - 16 bytes.
- #define SF_CRYPTO_CIPHER_AES_GCM_IV_PAD_4_BYTES

  Initial value: `(4U)`

  4 byte padding for 96-bit IV.
- #define SF_CRYPTO_CIPHER_AES_IV_LENGTH_16_BYTES

  Initial value: `(16U)`

  IV for AES operations - 16 bytes.
- #define SF_CRYPTO_CIPHER_AES_BLOCK_SIZE_BYTES

  Initial value: `(16U)`

  AES block size = 16 bytes.
- #define SF_CRYPTO_CIPHER_BYTES_PER_WORD

  Initial value: `(4U)`

  number of bytes in a WORD.
- #define SF_CRYPTO_CIPHER_AES_BLOCK_SIZE_IN_WORDS

  Initial value: `(#define SF_CRYPTO_CIPHER_AES_BLOCK_SIZE_BYTES/#define SF_CRYPTO_CIPHER_BYTES_PER_WORD)`
- #define SF_CRYPTO_CIPHER_RSA_1024_MODULUS_BITS

  Initial value: `(1024U)`

  Modulus size of RSA 1024-bit key. Macros for RSA operations EB = ENCRYPTION_BLOCK PS = Padding String PKCS_1_5 = RSAES-PKCS1-v1_5

- #define SF_CRYPTO_CIPHER_RSA_2048_MODULUS_BITS

  Initial value: `(2048U)`

  Modulus size of RSA 2048-bit key.
- #define SF_CRYPTO_PKCS_1_5_EB_START_BYTE

  Initial value: `(0U)`

  Encryption Block start byte = 00.
- #define SF_CRYPTO_PKCS_1_5_BT_00

  Initial value: `(0U)`

  Encryption Block Type (BT) = 00.
- #define SF_CRYPTO_PKCS_1_5_BT_01

  Initial value: `(1U)`

  Encryption Block Type (BT) = 01.
- #define SF_CRYPTO_PKCS_1_5_BT_02

  Initial value: `(2U)`

  Encryption Block Type (BT) = 02.
- #define SF_CRYPTO_PKCS_1_5_EB_DATA_SEPARATOR

  Initial value: `(0U)`

  EB Data separator (between PS and Data).
- #define SF_CRYPTO_PKCS_1_5_EB_START_BYTE_LENGTH

  Initial value: `(1U)`

  Encryption Block Start Byte length.
- #define SF_CRYPTO_PKCS_1_5_EB_BT_BYTE_LENGTH

  Initial value: `(1U)`

  Encryption Block Block Type field length.
- #define SF_CRYPTO_PKCS_1_5_EB_PS_MIN_LENGTH

  Initial value: `(8U)`

  Encryption Block Padding String (PS) min length.
- #define SF_CRYPTO_PKCS_1_5_EB_DATA_SEPARATOR_LENGTH

  Initial value: `(1U)`

  EB Data separator (between PS and Data)length.
- #define SF_CRYPTO_PKCS_1_5_EB_OVERHEAD

  Initial value: `(#define SF_CRYPTO_PKCS_1_5_EB_START_BYTE_LENGTH + \ #define SF_CRYPTO_PKCS_1_5_EB_BT_BYTE_LENGTH + \ #define`

```
SF_CRYPTO_PKCS_1_5_EB_PS_MIN_LENGTH + \ #define
SF_CRYPTO_PKCS_1_5_EB_DATA_SEPARATOR_LENGTH)
```

Overhead for formatting the Encryption Block, in number of bytes

- #define SF_CRYPTO_KEY_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  The API version of SSP Crypto Framework

- #define SF_CRYPTO_KEY_CODE_VERSION_MINOR

  Initial value: `(1U)`

6.13.8 SF_CRYPTO_Open

```
ssp_err_t SF_CRYPTO_Open ( sf_crypto_ctrl_t  *const p_api_ctrl ,
 sf_crypto_cfg_t  const *const p_cfg )
```

6.13.8.1 Brief description

SSP Crypto Framework Common Open operation.

6.13.8.2 Detailed description

Table 146:Parameters

Name	Direction	Description
p_api_ctrl	inout	Pointer to a Crypto framework control block
p_cfg	in	Pointer to a Crypto framework configuration structure

Table 147:Return values

Name	Description
SSP_SUCCESS	Crypto framework was successfully opened.
SSP_ERR_ASSERTION	NULL pointer is passed.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.
SSP_ERR_CRYPTO_HAL_ERROR	Crypto HAL driver retuned an error.

6.13.9 SF_CRYPTO_Close

```
ssp_err_t SF_CRYPTO_Close ( sf_crypto_ctrl_t  *const p_api_ctrl )
```

6.13.9.1 Brief description

SSP Crypto Framework Common Close operation.

6.13.9.2 Detailed description

Table 148:Parameters

Name	Direction	Description
p_api_ctrl	inout	Pointer to a Crypto framework control block

Table 149:Return values

Name	Description
SSP_SUCCESS	Module was successfully closed.
SSP_ERR_NOT_OPEN	Module has not opened.
SSP_ERR_ASSERTION	NULL pointer is passed as an input parameter.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.

6.13.10 SF_CRYPTO_Lock

```
ssp_err_t SF_CRYPTO_Lock ( sf_crypto_ctrl_t  *const p_api_ctrl )
```

6.13.10.1 Brief description

Locks the module. This API is utilized for locking shared resources.

6.13.10.2 Detailed description

Table 150:Parameters

Name	Direction	Description
p_api_ctrl	inout	Pointer to a Crypto framework control block

Table 151:Return values

Name	Description
SSP_SUCCESS	Module resources are successfully locked.
SSP_ERR_TIMEOUT	Unable to get ownership of the mutex within the specified time.
SSP_ERR_INTERNAL	Thread suspension was aborted. Critical error.
SSP_ERR_ASSERTION	NULL pointer is passed.as an input parameter.
SSP_ERR_NOT_OPEN	The module is not yet opened.

6.13.10.3 Function steps

- Check if the module has been opened. If not, return error.

6.13.11 SF_CRYPTO_Unlock

```
ssp_err_t SF_CRYPTO_Unlock ( sf_crypto_ctrl_t  *const p_api_ctrl )
```

6.13.11.1 Brief description

Unlocks the module. This API is utilized for unlocking shared resources.

6.13.11.2 Detailed description

Table 152:Parameters

Name	Direction	Description
p_api_ctrl	inout	Pointer to a Crypto framework control block

Table 153:Return values

Name	Description
SSP_SUCCESS	Module resources are successfully unlocked.
SSP_ERR_ASSERTION	NULL pointer is passed as an input parameter.
SSP_ERR_INTERNAL	Mutex is not owned by a caller thread.
SSP_ERR_NOT_OPEN	The module is not yet opened.

6.13.12 SF_CRYPTO_StatusGet

```
ssp_err_t SF_CRYPTO_StatusGet ( sf_crypto_ctrl_t  *const p_api_ctrl ,
 sf_crypto_state_t  * p_status )
```

6.13.12.1 Brief description

Gets the Crypto Common Framework module status.

6.13.12.2 Detailed description

Table 154:Parameters

Name	Direction	Description
p_api_ctrl	in	Pointer to a Crypto framework control block
p_status	out	Memory location to store module status.

Table 155:Return values

Name	Description
SSP_SUCCESS	Status returned successfully.
SSP_ERR_ASSERTION	The parameter p_status is NULL.
SSP_ERR_CRYPTO_COMMON_NOT_OPENED	This common module is not yet opened.

6.13.12.3 Function steps

- Check if the module has a valid / known status else return error.
- Other Crypto Framework modules will propogate this error to the caller. It will be helpful to explicitly indicate that the common module is not opened instead of the generic SSP_ERR_NOT_OPEN which may imply that the module from which the call is made is not open.

6.13.13 SF_CRYPTO_VersionGet

```
ssp_err_t SF_CRYPTO_VersionGet ( ssp_version_t  *const p_version )
```

6.13.13.1 Brief description

Gets the version of Crypto Common Framework module.

6.13.13.2 Detailed description

Table 156:Parameters

Name	Direction	Description
p_version	out	Pointer to the memory to store the version information.

Table 157:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

6.13.14 API Data

6.13.14.1 sf_crypto_key_type_t

`sf_crypto_key_type_t`

Detailed description

Supported key types

Enumerated values

Name	Description
SF_CRYPTO_KEY_TYPE_RSA_PLAIN_TEXT	RSA Key pair in standard format and plain text.
SF_CRYPTO_KEY_TYPE_RSA_CRT_PLAIN_TEXT	RSA Key pair in CRT format and plain text.
SF_CRYPTO_KEY_TYPE_RSA_WRAPPED	RSA Key pair public key in plain text and wrapped standard format private key.
SF_CRYPTO_KEY_TYPE_AES_WRAPPED	Wrapped AES key.
SF_CRYPTO_KEY_TYPE_AES_PLAIN_TEXT	AES Plain text key.
SF_CRYPTO_KEY_TYPE_ECC_PLAIN_TEXT	ECC Key pair in standard format and plain text.
SF_CRYPTO_KEY_TYPE_ECC_WRAPPED	ECC Key pair public key in plain text and wrapped standard format private key.
SF_CRYPTO_KEY_TYPE_ENCRYPTED_RSA_PRIVATE_KEY	RSA private key in encrypted format for installation.
SF_CRYPTO_KEY_TYPE_ENCRYPTED_AES_KEY	AES key in encrypted format for installation.
SF_CRYPTO_KEY_TYPE_ENCRYPTED_ECC_PRIVATE_KEY	ECC private key in encrypted format for installation.

6.13.14.2 sf_crypto_key_size_t

`sf_crypto_key_size_t`

Detailed description

Supported key sizes

Enumerated values

Name	Description
SF_CRYPTO_KEY_SIZE_RSA_1024	RSA 1024-bit key.
SF_CRYPTO_KEY_SIZE_RSA_2048	RSA 2048-bit key.
SF_CRYPTO_KEY_SIZE_AES_128	AES 128-bit key for CBC, CTR, ECB, GCM chaining modes.
SF_CRYPTO_KEY_SIZE_AES_XTS_128	AES 128-bit key for XTS chaining mode only.
SF_CRYPTO_KEY_SIZE_AES_192	AES 192-bit key for CBC, CTR, ECB, GCM chaining modes.
SF_CRYPTO_KEY_SIZE_AES_256	AES 256-bit key for CBC, CTR, ECB, GCM chaining modes.
SF_CRYPTO_KEY_SIZE_AES_XTS_256	AES 256-bit key for XTS chaining mode only.
SF_CRYPTO_KEY_SIZE_ECC_192	ECC 192-bit key.
SF_CRYPTO_KEY_SIZE_ECC_256	ECC 256-bit key.

6.13.14.3 sf_crypto_state_t

`sf_crypto_state_t`

Detailed description

State codes for the SSP Crypto Framework Common Module

Enumerated values

Name	Description
SF_CRYPTO_CLOSED	The module is closed.
SF_CRYPTO_OPENED	The module is opened.

6.13.14.4 sf_crypto_event_t

`sf_crypto_event_t`

Detailed description

Event code for the SSP Crypto Framework Common Module. This event code is all reserved for the future use.

Enumerated values

Name	Description
SF_CRYPTO_EVENT_PROCEDURE_DONE	Crypto hardware procedure done.
SF_CRYPTO_EVENT_ERROR	Error occurred.

6.13.14.5 sf_crypto_close_option_t

`sf_crypto_close_option_t`

Detailed description

SF_CRYPTO Close option. The module executes close operation if any SF_CRYPTO_XXX modules have already closed if SF_CRYPTO_CLOSE_OPTION_DEFAULT option is specified. The module performs close operation regardless of any SF_CRYPTO_XXX module status if SF_CRYPTO_CLOSE_OPTION_FORCE_CLOSE is specified.

Enumerated values

Name	Description
SF_CRYPTO_CLOSE_OPTION_DEFAULT	Close the module if no any SF_CRYPTO_XXX modules opened.
SF_CRYPTO_CLOSE_OPTION_FORCE_CLOSE	Close the module regardless of SF_CRYPTO_XXX modules status.

6.13.14.6 sf_crypto_cipher_state_t

`sf_crypto_cipher_state_t`

Detailed description

States the SSP Crypto Cipher Framework module can go through.

Enumerated values

Name	Description
SF_CRYPTO_CIPHER_STATE_CLOSED	The Cipher module is closed.
SF_CRYPTO_CIPHER_STATE_OPENED	The Cipher module is opened.
SF_CRYPTO_CIPHER_STATE_INITIALIZED	The cipher operation is initialized.
SF_CRYPTO_CIPHER_STATE_UPDATED	The cipher operation is updated.
SF_CRYPTO_CIPHER_STATE_FINALIZED	The cipher operation is finalized.

6.13.14.7 sf_crypto_ctrl_t

```
typedef void sf_crypto_ctrl_t
```

Detailed description

SSP Crypto Framework Common Module control block. Allocate an instance specific control block to pass into the SSP Crypto Framework Common Module API calls. Implemented as

- sf_crypto_instance_ctrl_t

6.13.15 Extensions

6.13.15.1 sf_crypto_data_handle_t

sf_crypto_data_handle_t

Detailed description

A structure to handle data among Crypto Framework modules

Variables

- uint8_t * p_data

  Pointer to data.

- uint32_t data_length

  The length of data pointed by p_data.

6.13.15.2 sf_crypto_callback_args_t

sf_crypto_callback_args_t

Detailed description

Callback arguments for the SSP Crypto Framework Common Module

Variables

- sf_crypto_event_t event

  Event code of the low level hardware.

- ssp_err_t error

  Error code if SF_CRYPTO_EVENT_ERROR.

6.13.15.3 sf_crypto_cfg_t

sf_crypto_cfg_t

Detailed description

Configuration structure for the SSP Crypto Framework Common Module

Variables

- uint32_t wait_option

  Wait option for RTOS service calls.
- crypto_instance_t * p_lower_lvl_crypto

  Pointer to a low-level Crypto engine HAL driver instance.
- void const * p_extend

  Extension parameter for hardware specific settings.
- void const * p_context

  Placeholder for user data.
- void * p_memory_pool

  Byte pool address.
- uint32_t memory_pool_size

  Byte pool size.
- sf_crypto_close_option_t close_option

  Close option.

6.13.15.4 sf_crypto_api_t

sf_crypto_api_t

Detailed description

Shared Interface definition for the SSP Crypto Framework Common Module

6.13.15.5 sf_crypto_instance_t

sf_crypto_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_crypto_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_crypto_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_crypto_api_t const * p_api

  Pointer to the API structure for this instance.

6.13.15.6 sf_crypto_instance_ctrl_t

sf_crypto_instance_ctrl_t

Detailed description

SSP Crypto Framework Common Module instance control block

Variables

- sf_crypto_state_t status

  Module status.

- TX_MUTEX mutex

  Mutex used in the Crypto Framework.

- TX_SEMAPHORE semaphore

  Semaphore used in the Crypto Framework (Reserve)

- TX_BYTE_POOL byte_pool

  Byte pool used in the Crypto Framework.

- uint32_t wait_option

  Wait time option used for RTOS service calls.

- uint32_t open_counter

  Counter to keep the number of SF_CRYPTO_XXX opened.

- void * p_lower_lvl_crypto

  Pointer to a low-level Crypto engine HAL driver instance.

- void(* p_callback)(*p_args)

  Pointer to callback function.

- void * p_context

  Pointer to a context.

- sf_crypto_close_option_t close_option

  Close option.

6.13.15.7 sf_crypto_cipher_instance_ctrl_t

sf_crypto_cipher_instance_ctrl_t

Detailed description

SSP Crypto Cipher Framework instance control block. DO NOT INITIALIZE. Initialization occurs when SF_CRYPTO_CIPHER_Open is called

Variables

- sf_crypto_key_type_t key_type

  Key type.

- sf_crypto_key_size_t key_size

  Key size.

- sf_crypto_cipher_mode_t cipher_chaining_mode
  Chaining mode specified for the cipher operation.
- sf_crypto_cipher_state_t status
  Module status.
- crypto_algorithm_type_t cipher_algorithm_type
  Cipher algorithm for the keys selected.
- sf_crypto_instance_ctrl_t * p_lower_lvl_fwk_common_ctrl
  Pointer to the Crypto Framework Common instance.
- sf_crypto_api_t * p_lower_lvl_fwk_common_api
  Pointer to the Crypto Framework Common API.
- sf_crypto_trng_instance_ctrl_t * p_lower_lvl_sf_crypto_trng_ctrl
  Pointer to the Crypto TRNG API.
- sf_crypto_trng_api_t * p_sf_crypto_trng_api
  Pointer to the Crypto TRNG API.
- void * p_hal_ctrl
  Pointer to HAL control structure for the Cipher operation.
- void * p_hal_api
  Pointer to HAL API structure for the cipher algorithm.
- void * p_cipher_context_buffer
  Cipher context buffer after DWORD alignment.

6.13.15.8 sf_crypto_key_instance_ctrl_t

sf_crypto_key_instance_ctrl_t

Detailed description

SSP Crypto Key Framework instance control block

Variables

- sf_crypto_key_state_t status
  Module status.
- sf_crypto_key_type_t key_type
  Key type.
- sf_crypto_key_size_t key_size
  Key size.
- sf_crypto_data_handle_t domain_params
  Domain parameters structure with pointer to data and length.

- sf_crypto_data_handle_t generator_point

  Generator Point structure with pointer to data and length.
- sf_crypto_instance_ctrl_t * p_fwk_common_ctrl

  Pointer to the Crypto Framework Common instance.
- sf_crypto_api_t * p_fwk_common_api

  Pointer to the Crypto Framework Common instance.
- void * p_hal_ctrl

  pointer to Crypto module control structure
- void * p_hal_api

  pointer to Crypto module API structure

6.14 SSP Crypto Cipher Framework Interface

Interface definition for Synergy Crypto Cipher Framework module.

Interface declaration for Synergy Crypto Key Framework module.

Interface declaration for Synergy Crypto Cipher Framework module for RSA.

Interface declaration for Synergy Crypto Cipher Framework module for AES.

6.14.1 Summary

This is a ThreadX aware Interface of SF_CRYPTO_CIPHER Framework module which provides encryption and decryption operations for AES and RSA algorithms.

SF_CRYPTO_CIPHER Framework Interface description: Crypto Framework

6.14.2 Functions

- sf_crypto_cipher_aes_hal_open
- sf_crypto_cipher_aes_hal_close
- sf_crypto_cipher_aes_interface_get
- sf_crypto_cipher_aes_fill_interface_key_type
- sf_crypto_cipher_aes_fill_interface_key_size
- sf_crypto_cipher_aes_fill_interface_chaining_mode
- sf_crypto_cipher_instance_aes_memory_allocate
- sf_crypto_cipher_is_key_type_aes
- sf_crypto_cipher_aes_get_plain_text_key_size_bytes
- sf_crypto_cipher_aes_get_wrapped_key_size_bytes

- sf_crypto_cipher_aes_get_key_size_bytes
- sf_crypto_cipher_aes_validate_init_gcm_params
- sf_crypto_cipher_aes_validate_init_params_iv
- sf_crypto_cipher_aes_validate_init_padding_scheme
- sf_crypto_cipher_aes_validate_init_params
- sf_crypto_cipher_aes_init_gcm_params
- sf_crypto_cipher_aes_init
- sf_crypto_cipher_aes_check_fill_partial_block
- sf_crypto_cipher_aes_fill_partial_block
- sf_crypto_cipher_aes_instance_gcm_memory_allocate
- sf_crypto_cipher_aes_update_helper_gcm_aad_update
- sf_crypto_cipher_aes_update
- sf_crypto_cipher_aes_check_update_params
- sf_crypto_cipher_aes_validate_update_params_context
- sf_crypto_cipher_aes_validate_aad_update_params_context
- sf_crypto_cipher_aes_aad_update
- sf_crypto_cipher_aes_process_aad_partial_block
- sf_crypto_cipher_aes_process_partial_block
- sf_crypto_cipher_aes_gcm_process_partial_block
- sf_crypto_cipher_aes_encrypt_final
- sf_crypto_cipher_aes_decrypt_final_check_input_length
- sf_crypto_cipher_aes_decrypt_final
- sf_crypto_cipher_aes_is_last_decrypt_block
- sf_crypto_cipher_aes_decrypt_last_block
- sf_crypto_cipher_aes_helper_decrypt_last_block
- sf_crypto_cipher_is_last_block_with_padding
- sf_crypto_cipher_aes_helper_decrypt_final
- sf_crypto_cipher_aes_gcm_helper_decrypt_final
- sf_crypto_cipher_aes_handle_encrypt_padding
- sf_crypto_cipher_aes_get_gcm_tag
- sf_crypto_cipher_aes_set_gcm_tag
- sf_crypto_cipher_aes_compute_verify_gcm_tag
- sf_crypto_cipher_aes_helper_validate_final_params
- sf_crypto_cipher_aes_validate_encrypt_final_out_buffer

- sf_crypto_cipher_aes_validate_encrypt_final_params
- sf_crypto_cipher_aes_validate_decrypt_final_params
- sf_crypto_cipher_aes_hal_encrypt_decrypt
- sf_crypto_cipher_aes_final
- sf_crypto_cipher_aes_context_buffer_pointer_data_zeroise
- sf_crypto_cipher_instance_aes_memory_deallocate
- sf_crypto_cipher_aes_zeroise_context_buffer
- sf_crypto_cipher_instance_aes_gcm_memory_release
- sf_crypto_cipher_initialize_aes_instance
- sf_crypto_cipher_deinitialize_aes_instance
- SF_CRYPTO_CIPHER_RSA_EB_SIZE_BYTES
- SF_CRYPTO_CIPHER_RSA_PKCS_1_5_EB_DATA_SIZE_BYTES
- sf_crypto_cipher_rsa_hal_open
- sf_crypto_cipher_rsa_hal_close
- sf_crypto_cipher_rsa_interface_get
- sf_crypto_cipher_rsa_private_free_context_buffer_key
- sf_crypto_cipher_rsa_private_free_context_buffer
- sf_crypto_cipher_instance_rsa_memory_allocate
- sf_crypto_cipher_is_key_type_rsa
- sf_crypto_cipher_rsa_init
- sf_crypto_cipher_rsa_update
- sf_crypto_cipher_rsa_helper_pkcs_1_5_encode_block
- sf_crypto_cipher_rsa_encrypt_final
- sf_crypto_cipher_rsa_hal_encrypt_decrypt
- sf_crypto_cipher_rsa_pkcs_1_5_encode_block
- sf_crypto_cipher_rsa_decrypt_final
- sf_crypto_cipher_rsa_pkcs_1_5_decode_block
- sf_crypto_cipher_rsa_validate_encrypt_final_out_params
- sf_crypto_cipher_rsa_final_input_buffer_check
- sf_crypto_cipher_rsa_final
- sf_crypto_cipher_rsa_context_buffer_pointer_data_zeroise
- sf_crypto_cipher_instance_rsa_memory_deallocate
- sf_crypto_cipher_rsa_zeroise_context_buffer
- sf_crypto_cipher_get_rsa_pvt_key_size_bytes

- sf_crypto_cipher_get_rsa_public_key_size_bytes
- sf_crypto_cipher_get_rsa_modulus_size_bits
- sf_crypto_cipher_get_rsa_block_size_bytes
- sf_crypto_cipher_rsa_get_modulus_block_sizes
- sf_crypto_cipher_rsa_encrypt_update
- sf_crypto_cipher_rsa_decrypt_update
- sf_crypto_cipher_rsa_init_validate_padding_scheme
- sf_crypto_cipher_rsa_init_validate
- sf_crypto_cipher_initialize_rsa_instance
- sf_crypto_cipher_deinitialize_rsa_instance
- SF_CRYPTO_CIPHER_Open
- SF_CRYPTO_CIPHER_Close
- SF_CRYPTO_CIPHER_VersionGet
- SF_CRYPTO_CIPHER_CipherInit
- SF_CRYPTO_CIPHER_CipherUpdate
- SF_CRYPTO_CIPHER_CipherAadUpdate
- SF_CRYPTO_CIPHER_CipherFinal

6.14.3 Typedefs

- sf_crypto_cipher_algorithm_init_params_t
- sf_crypto_cipher_ctrl_t

6.14.4 sf_crypto_cipher_aes_hal_open

```
ssp_err_t sf_crypto_cipher_aes_hal_open ( sf_crypto_cipher_instance_ctrl_t
* p_ctrl ,  sf_crypto_cipher_cfg_t  const *const p_cfg )
```

6.14.4.1 Brief description

Subroutine to open a Crypto AES HAL module. This function is called by SF_CRYPTO_CIPHER_Open.

6.14.4.2 Detailed description

Table 158:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used and set
p_cfg	in	Pointer to the cipher framework module configuration parameters. in the open() call.

Table 159:Return values

Name	Description
SSP_SUCCESS	AES HAL module is successfully opened.
SSP_ERR_OUT_OF_MEMORY	Failed to allocate memory to store AES HAL module control block.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.
See	Common Error Codes or HAL driver for other possible return codes or causes. This function calls: open

6.14.4.3 Function steps

- Get a Crypto common control block and the interface.
- Allocate memory for a Crypto HAL control block in the byte pool.
- Get a AES interface instance.
- Set Crypto HAL API instance with the control common hardware api.
- Open the Crypto HAL AES module.

6.14.5 sf_crypto_cipher_aes_hal_close

```
ssp_err_t sf_crypto_cipher_aes_hal_close ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl )
```

6.14.5.1 Brief description

Subroutine to close a Crypto AES HAL module. This function is called by SF_CRYPTO_CIPHER_Close.

6.14.5.2 Detailed description

End of function sf_crypto_cipher_aes_hal_open

Table 160:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the cipher framework module control block used and set in the open() call.

Table 161:Return values

Name	Description
SSP_SUCCESS	AES HAL module is successfully closed.
See	Common Error Codes or HAL driver for other possible return codes or causes. This function calls: close

6.14.5.3 Function steps

- Release the control block memory back to byte pool

6.14.6 sf_crypto_cipher_aes_interface_get

```
 sf_crypto_cipher_aes_interface_get ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl )
```

6.14.6.1 Brief description

Subroutine to get a pointer to the AES HAL API instance.

6.14.6.2 Detailed description

End of function sf_crypto_cipher_aes_hal_close

Table 162:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a Cipher framework control block, whose p_hal_api is filled with HAL AES interface. This will be NULL, for MCUs /feature/ configuration parameters not supported

6.14.6.3 Function steps

- Get a Crypto common control block and the HAL instance.
- Check the AES key type and size and get an appropriate API instance.
- Get the HAL API instance for a selected algorithm type.

6.14.7 sf_crypto_cipher_aes_fill_interface_key_type

```
sf_crypto_cipher_aes_fill_interface_key_type ( sf_crypto_cipher_instance_ctrl_t
 *const p_ctrl ,   crypto_interface_get_param_t * param )
```

6.14.7.1 Brief description

Subroutine to fill the key type parameter in the crypto_interface_get_param_t structure.

6.14.7.2 Detailed description

End of function sf_crypto_cipher_aes_interface_get

Table 163:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used and set in the open() call. *
param	in	Pointer to a crypto_interface_get_param_t structure to get the lower level interface.

6.14.8 sf_crypto_cipher_aes_fill_interface_key_size

```
sf_crypto_cipher_aes_fill_interface_key_size ( sf_crypto_cipher_instance_ctrl_t
 *const p_ctrl ,   crypto_interface_get_param_t * param )
```

6.14.8.1 Brief description

Subroutine to fill the key size parameter in the crypto_interface_get_param_t structure. This function is called by SF_CRYPTO_CIPHER_Open.

6.14.8.2 Detailed description

Table 164:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a Crypto Key Framework module control block.
param	in	Pointer to a crypto_interface_get_param_t structure to get the lower level interface.

6.14.9 sf_crypto_cipher_aes_fill_interface_chaining_mode

```
 sf_crypto_cipher_aes_fill_interface_chaining_mode ( sf_crypto_cipher_instance_c
trl_t  *const p_ctrl ,   crypto_interface_get_param_t * param )
```

6.14.9.1 Brief description

Subroutine to fill the chaining mode parameter in the crypto_interface_get_param_t structure. This function is called by SF_CRYPTO_CIPHER_Open.

6.14.9.2 Detailed description

Table 165:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a Crypto Key Framework module control block.

Table 165:Parameters (Continued)

Name	Direction	Description
param	in	Pointer to a crypto_interface_get_param_t structure to get the lower level interface.

6.14.10 sf_crypto_cipher_instance_aes_memory_allocate

```
ssp_err_t sf_crypto_cipher_instance_aes_memory_allocate ( sf_crypto_cipher_instance_ctrl_t  *const p_ctrl )
```

6.14.10.1 Brief description

Subroutine to allocate memory required for the context buffer for AES cipher operation. A contiguous block for the entire memory requirement for AES instance may not be available in the byte pool if it is fragmented. Hence memory is allocated in chunks to get the most out of the byte pool.

6.14.10.2 Detailed description

Table 166:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a Crypto Key Framework module control block.

Table 167:Return values

Name	Description
SSP_SUCCESS	Memory allocation was successful.
SSP_ERR_TIMEOUT	Was unable to allocate the memory within the specified time to wait.
SSP_ERR_OUT_OF_MEMORY	Requested size is zero or larger than the pool.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error. *

6.14.10.3 Function steps

- Allocate memory for the AES context buffer. Extra 3 bytes are allocated to get the buffer WORD aligned
- Allocate memory for IV in the context buffer Extra 3 bytes are allocated to get the buffer WORD aligned
- Allocate memory for the key in the context buffer Extra 3 bytes are allocated to get the buffer WORD aligned
- Allocate memory for the partial block in the context buffer Extra 3 bytes are allocated to get the buffer WORD aligned.
- Allocate memory required for GCM related elements in the context buffer

6.14.11 sf_crypto_cipher_is_key_type_aes

```
sf_crypto_cipher_is_key_type_aes ( sf_crypto_key_type_t  key_type )
```

6.14.11.1 Brief description

Subroutine to check if the key type enum provided is a valid AES key type for cipher operation.

6.14.11.2 Detailed description

End of function sf_crypto_cipher_instance_aes_memory_allocate

Table 168:Parameters

Name	Direction	Description
key_type	in	The key type to be tested.

Table 169:Return values

Name	Description
true	The key is a valid AES key type.
false	The key is NOT a valid AES key type.

6.14.11.3 Function steps

- Check the key type and and determine the algorithm.

6.14.12 sf_crypto_cipher_aes_get_plain_text_key_size_bytes

```
ssp_err_t sf_crypto_cipher_aes_get_plain_text_key_size_bytes ( sf_crypto_key_size_t  key_size ,   uint32_t * key_size_bytes )
```

6.14.12.1 Brief description

Subroutine to get AES plain text key size in bytes given the key size enums.

6.14.12.2 Detailed description

End of function sf_crypto_cipher_is_key_type_aes

Table 170:Parameters

Name	Direction	Description
key_size	in	The key size enum to be used in calculating the key size in bytes.

Table 171:Return values

Name	Description
key_size_bytes	The AES key size in bytes. 0 if any of the inputs are invalid.

6.14.13 sf_crypto_cipher_aes_get_wrapped_key_size_bytes

```
ssp_err_t sf_crypto_cipher_aes_get_wrapped_key_size_bytes ( sf_crypto_key_size_t  key_size ,   uint32_t * key_size_bytes )
```

6.14.13.1 Brief description

Subroutine to get AES wrapped key size in bytes given the key size enums.

6.14.13.2 Detailed description

End of function sf_crypto_cipher_aes_get_plain_text_key_size_bytes

Table 172:Parameters

Name	Direction	Description
key_size	in	The key size enum to be used in calculating the key size in bytes.

Table 173:Return values

Name	Description
key_size_bytes	The AES key size in bytes. 0 if any of the inputs are invalid.

6.14.14 sf_crypto_cipher_aes_get_key_size_bytes

```
ssp_err_t sf_crypto_cipher_aes_get_key_size_bytes ( sf_crypto_key_type_t  key_ty
pe ,  sf_crypto_key_size_t  key_size ,   uint32_t * key_size_bytes )
```

6.14.14.1 Brief description

Subroutine to get AES key size in bytes given the key type and key size enums.

6.14.14.2 Detailed description

End of function sf_crypto_cipher_aes_get_wrapped_key_size_bytes

Table 174:Parameters

Name	Direction	Description
key_type	in	The key type enum to be used in calculating the key size in bytes.
key_size	in	The key size enum to be used in calculating the key size in bytes.

Table 175:Return values

Name	Description
key_size_bytes	The AES key size in bytes. 0 if any of the inputs are invalid.

6.14.15 sf_crypto_cipher_aes_validate_init_gcm_params

```
ssp_err_t sf_crypto_cipher_aes_validate_init_gcm_params ( sf_crypto_cipher_aes_init_params_t  * p_aes_params )
```

6.14.15.1 Brief description

Subroutine to validate the input parameters for the AES cipher Init operation.

6.14.15.2 Detailed description

End of function sf_crypto_cipher_aes_get_key_size_bytes

Table 176:Parameters

Name	Direction	Description
p_aes_params	in	Pointer to the AES algorithm specific parameter struct.

Table 177:Return values

Name	Description
SSP_SUCCESS	All of the input parameters are validated successfully.
SSP_ERR_INVALID_ARGUMENT	An input for the required cipher operation is invalid.

6.14.15.3 Function steps

- GCM supports 96-bit IV and 128-bit IV
- Tag is necessary for both encrypt and decrypt operations

6.14.16 sf_crypto_cipher_aes_validate_init_params_iv

```
ssp_err_t sf_crypto_cipher_aes_validate_init_params_iv ( sf_crypto_cipher_instance_ctrl_t  *const p_ctrl ,  sf_crypto_cipher_aes_init_params_t  * p_aes_params )
```

6.14.16.1 Brief description

Subroutine to validate the IV for the AES cipher Init operation for modes other than ECB and GCM.

6.14.16.2 Detailed description

Table 178:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_aes_params	in	Pointer to the algorithm specific parameters.

Table 179:Return values

Name	Description
SSP_SUCCESS	All of the input parameters are validated successfully.
SSP_ERR_INVALID_ARGUMENT	An input for the required cipher operation is invalid.

6.14.17 sf_crypto_cipher_aes_validate_init_padding_scheme

```
ssp_err_t sf_crypto_cipher_aes_validate_init_padding_scheme ( sf_crypto_cipher_instance_ctrl_t  *const p_ctrl ,
 sf_crypto_cipher_padding_scheme_t  padding_scheme )
```

6.14.17.1 Brief description

Subroutine to validate the padding scheme for AES cipher Init operation.

6.14.17.2 Detailed description

End of function sf_crypto_cipher_aes_validate_init_params_iv

Table 180:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
padding_scheme	in	The padding scheme to be used for the cipher operations.

Table 181:Return values

Name	Description
SSP_SUCCESS	All of the input parameters are validated successfully.
SSP_ERR_INVALID_ARGUMENT	An input for the required cipher operation is invalid.

6.14.17.3 Function steps

- Expect NO_PADDING for CTR and XTS - we expect multiple of block size For GCM it will be internally padded with 0, so user has to choose no padding.

6.14.18 sf_crypto_cipher_aes_validate_init_params

```
ssp_err_t sf_crypto_cipher_aes_validate_init_params ( sf_crypto_cipher_instance_
ctrl_t  *const p_ctrl ,  sf_crypto_cipher_op_mode_t  cipher_operation_mode ,
 sf_crypto_data_handle_t  const *const p_key ,
 sf_crypto_cipher_aes_init_params_t  * p_aes_params )
```

6.14.18.1 Brief description

Subroutine to validate the input parameters for the AES cipher Init operation.

6.14.18.2 Detailed description

End of function sf_crypto_cipher_aes_validate_init_padding_scheme

Table 182:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
cipher_operation_mode	in	The cipher operation mode - encrypt / decrypt.
p_key	in	Pointer to the key to be used for the cipher operations.
p_aes_params	in	Pointer to the algorithm specific parameters.

Table 183:Return values

Name	Description
SSP_SUCCESS	All of the input parameters are validated successfully.
SSP_ERR_INVALID_ARGUMENT	An input for the required cipher operation is invalid.

6.14.18.3 Function steps

- Cipher operation mode is already validated before this function is called.
- Validate key length
- Validate padding scheme
- Validate algorithm specific parameters - IV for all modes except ECB and GCM ECB does not need IV and IV length for GCM will be tested in the next call to a GCM specific test.
- Validate parameters / lengths specific to GCM mode

6.14.19 sf_crypto_cipher_aes_init_gcm_params

```
ssp_err_t sf_crypto_cipher_aes_init_gcm_params ( sf_crypto_cipher_instance_ctrl_
t  *const p_ctrl ,  sf_crypto_cipher_op_mode_t  cipher_operation_mode ,
 sf_crypto_data_handle_t  const *const p_key ,
 sf_crypto_cipher_algorithm_init_params_t  * p_algorithm_specific_params )
```

6.14.19.1 Brief description

Subroutine to init the AES GCM mode of operation with parameters from the AES cipherInit operation.

6.14.19.2 Detailed description

End of function sf_crypto_cipher_validate_aes_init_params

Table 184:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
cipher_operation_mode	in	The cipher operation mode - encrypt / decrypt.
p_key	in	Pointer to the key to be used for the cipher operations.
p_algorithm_specific_params	in	Pointer to the algorithm specific parameters.

Table 185:Return values

Name	Description
SSP_SUCCESS	All of the input parameters are validated successfully.
Others	Crypto HAL module returned an error or resource lock/unlock was failed. See Common Error Codes or HAL driver for other possible return codes or causes. This function calls: open

6.14.19.3 Function steps

- These unused common params are already handled in the calling routine
- Store the tag address supplied by the user, in the context buffer.
- For GCM, the work buffer in the HAL has to be initialized and that is done through the open call because there is no HW interaction and only the work buffer in the control block is cleaned up.

6.14.20 sf_crypto_cipher_aes_init

```
ssp_err_t sf_crypto_cipher_aes_init ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl ,  sf_crypto_cipher_op_mode_t  cipher_operation_mode ,
 sf_crypto_data_handle_t  const *const p_key ,
 sf_crypto_cipher_algorithm_init_params_t  * p_algorithm_specific_params )
```

6.14.20.1 Brief description

Subroutine to initialize the AES cipher operation.

6.14.20.2 Detailed description

Table 186:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
cipher_operation_mode	in	The cipher operation mode - encrypt / decrypt.
p_key	in	Pointer to the key to be used for the cipher operations.
p_algorithm_specific_params	in	Pointer to the algorithm specific parameters.

Table 187:Return values

Name	Description
SSP_SUCCESS	All of the input parameters are validated successfully.
SSP_ERR_INVALID_ARGUMENT	An input for the required cipher operation is invalid.

6.14.20.3 Function steps

- All buffers within the context buffer are allocated at Open. Now just zeroise their contents.
- Copy the init parameters to the context buffer

6.14.21 sf_crypto_cipher_aes_check_fill_partial_block

```
sf_crypto_cipher_aes_check_fill_partial_block ( sf_crypto_cipher_instance_ctrl_
t  *const p_ctrl ,  uint32_t * fill_bytes ,  sf_crypto_data_handle_t  const
*const p_data_in )
```

6.14.21.1 Brief description

Subroutine to fill partial data block in the context buffer for the AES cipher operation. First checks if there is any data in the partial block. Only if there is any data, fills the partial block with the data from the input.

6.14.21.2 Detailed description

End of function sf_crypto_cipher_aes_init

Table 188:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
fill_bytes	inout	Pointer to the number of bytes required to fill the buffer. On output, contains the number of bytes used.
p_data_in	in	Pointer to input data output from the cipher operation.

6.14.21.3 Function steps

- Check if there is data in the partial block buffer. If so fill it from the input data
- Calculate the number of bytes to fill the partial block.
- Update the size of the data in the partial block.

6.14.22 sf_crypto_cipher_aes_fill_partial_block

```
sf_crypto_cipher_aes_fill_partial_block (  uint8_t * p_fill_buffer ,  uint32_t
* p_num_bytes_in_buffer ,  uint32_t * fill_bytes ,  sf_crypto_data_handle_t
const *const p_data_in )
```

6.14.22.1 Brief description

Subroutine to fill any AES partial block for the AES cipher operation. Fills the buffer only if there is any data present in the buffer.

6.14.22.2 Detailed description

Table 189:Parameters

Name	Direction	Description
p_fill_buffer	in	Pointer to the cbuffer to be filled.
p_num_bytes_in_buffer	inout	Pointer to the number of bytes already present in the buffer.
fill_bytes	inout	Pointer to the number of bytes required to fill the buffer.
p_data_in	in	Pointer to data handle of the input data from the user.

6.14.22.3 Function steps

- Calculate the number of bytes to fill the buffer.
- Update the size of the data in the partial block.

6.14.23 sf_crypto_cipher_aes_instance_gcm_memory_allocate

```
ssp_err_t sf_crypto_cipher_aes_instance_gcm_memory_allocate ( sf_crypto_cipher_i
nstance_ctrl_t  *const p_ctrl )
```

6.14.23.1 Brief description

Subroutine to allocate memory for GCM parameters in the context buffer.

6.14.23.2 Detailed description

Table 190:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.

Table 191:Return values

Name	Description
SSP_SUCCESS	Memory was alocated successfully.
SSP_ERR_TIMEOUT	Was unable to allocate the memory within the specified time to wait.
SSP_ERR_OUT_OF_MEMORY	Requested size is zero or larger than the pool.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.

6.14.23.3 Function steps

- Allocate memory for GCM tag in the context buffer. Extra 3 bytes are allocated to get the buffer WORD aligned
- Allocate memory for partial AAD in the context buffer. Extra 3 bytes are allocated to get the buffer WORD aligned
- Release the memory allocated for GCM Tag

6.14.24 sf_crypto_cipher_aes_update_helper_gcm_aad_update

```
ssp_err_t sf_crypto_cipher_aes_update_helper_gcm_aad_update ( sf_crypto_cipher_instance_ctrl_t  *const p_ctrl )
```

6.14.24.1 Brief description

Subroutine to update check and update AAD for cipher AES operation. If the mode is GCM and there is any partial AAD not processed yet, it will be processed.

6.14.24.2 Detailed description

Table 192:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.

Table 193:Return values

Name	Description
SSP_SUCCESS	AAD check / operation was completed successfully.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.24.3 Function steps

- Now that all the data in the partial block is processed, clean it up and set the size to 0

6.14.25 sf_crypto_cipher_aes_update

```
ssp_err_t sf_crypto_cipher_aes_update ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in ,
 sf_crypto_data_handle_t  *const p_data_out )
```

6.14.25.1 Brief description

Subroutine to update the cipher AES operation.

6.14.25.2 Detailed description

Table 194:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_data_in	inout	Pointer to the input data structure - has the pointer to input data and the data length
p_data_out	inout	Pointer to the output data structure - has the pointer to output data and the data length.

Table 195:Return values

Name	Description
SSP_SUCCESS	Cipher operation was updated successfully.
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.25.3 Function steps

- If GCM check if there is any partial AAD remaining. This is to be processed only if the cipher is in initialized state. Once the update is done AAD calculations should not be done. if so call to update AAD
- Set the data out length before starting cipher operation.
- Update the number of bytes remaining in the input buffer to be processed.
- If data is block length, process it. That is encrypt / decrypt it.
- Check if the remaining input data is a multiple of block size
- Update the number of bytes written to the output buffer.
- Update the number of bytes remaining in the input buffer to be processed.
- If any bytes remain that is less than the block size, copy to the partial buffer
- Update the size of the data in the partial block

6.14.26 sf_crypto_cipher_aes_check_update_params

```
ssp_err_t sf_crypto_cipher_aes_check_update_params ( sf_crypto_cipher_instance_c
trl_t  *const p_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in ,
 sf_crypto_data_handle_t  *const p_data_out )
```

6.14.26.1 Brief description

Subroutine to check the parameters for the cipherUpdate operation.

6.14.26.2 Detailed description

End of function sf_crypto_cipher_aes_update

Table 196:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_data_in	in	Pointer to the data handle of the input data
p_data_out	in	Pointer to the data handle of the output data

Table 197:Return values

Name	Description
SSP_SUCCESS	Parameters were validated successfully.
SSP_ERR_INVALID_ARGUMENT	At least one of the buffer is set to NULL.
SSP_ERR_INVALID_SIZE	The output buffer sizes is invalid.

6.14.26.3 Function steps

- Check if the output buffer length is sufficient to hold the cipher text.
- validate the context buffer params too.

6.14.27 sf_crypto_cipher_aes_validate_update_params_context

```
ssp_err_t sf_crypto_cipher_aes_validate_update_params_context ( sf_crypto_cipher
_instance_ctrl_t  *const p_ctrl )
```

6.14.27.1 Brief description

Subroutine to validate the context buffer parameters for the cipherUpdate operation.

6.14.27.2 Detailed description

Table 198:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.

Table 199:Return values

Name	Description
SSP_SUCCESS	Parameters were validated successfully.
SSP_ERR_ASSERTION	At least one of the buffer is set to NULL.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.28 sf_crypto_cipher_aes_validate_aad_update_params_context

```
ssp_err_t sf_crypto_cipher_aes_validate_aad_update_params_context ( sf_crypto_cipher_instance_ctrl_t  *const p_ctrl )
```

6.14.28.1 Brief description

Subroutine to validate the context buffer parameters for the cipherAadUpdate operation.

6.14.28.2 Detailed description

End of function sf_crypto_cipher_aes_validate_update_params

Table 200:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.

Table 201:Return values

Name	Description
SSP_SUCCESS	Parameters were validated successfully.
SSP_ERR_ASSERTION	At least one of the buffer is set to NULL.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.29 sf_crypto_cipher_aes_aad_update

```
ssp_err_t sf_crypto_cipher_aes_aad_update ( sf_crypto_cipher_instance_ctrl_t
* p_ctrl ,  sf_crypto_data_handle_t  const *const p_aad )
```

6.14.29.1 Brief description

Subroutine to update the AAD for the cipher operation.

6.14.29.2 Detailed description

End of function sf_crypto_cipher_aes_validate_aad_update_params_context

Table 202:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_aad	in	Pointer to the input data structure - has the pointer to input AAD and the data length

Table 203:Return values

Name	Description
SSP_SUCCESS	AAD was updated successfully.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.29.3 Function steps

- Check if there is data in the aad block buffer. If so first fill it from the input data
- Update the number of bytes remaining in the input buffer to be processed.
- If data is block length, encrypt it.
- Check if the input data is a multiple of block size
- Update the number of bytes remaining in the input buffer to be processed.
- If any bytes remain that is less than the block size, copy to the partial buffer
- Update the size of the data in the partial block

6.14.30 sf_crypto_cipher_aes_process_aad_partial_block

```
sf_crypto_cipher_aes_process_aad_partial_block ( sf_crypto_cipher_instance_ctrl
_t  *const p_ctrl )
```

6.14.30.1 Brief description

Subroutine to update the partial block of AAD for the cipher operation. Any partial data in the context buffer will be zero padded internally and updated.

6.14.30.2 Detailed description

End of function sf_crypto_cipher_aes_aad_update

Table 204:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.

Table 205:Return values

Name	Description
SSP_SUCCESS	AAD was updated successfully.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.31 sf_crypto_cipher_aes_process_partial_block

```
ssp_err_t sf_crypto_cipher_aes_process_partial_block ( sf_crypto_cipher_instance
_ctrl_t  *const p_ctrl ,  sf_crypto_data_handle_t  *const p_data_out )
```

6.14.31.1 Brief description

Subroutine to update the partial block of input data for the cipher operation. This calls the lower level HAL APIs to process the data.

6.14.31.2 Detailed description

End of function sf_crypto_cipher_process_aes_aad_partial_block

Table 206:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_data_out	in	Pointer to data handle of teh output data.

Table 207:Return values

Name	Description
SSP_SUCCESS	Data was processed successfully.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.32 sf_crypto_cipher_aes_gcm_process_partial_block

```
 sf_crypto_cipher_aes_gcm_process_partial_block ( sf_crypto_cipher_instance_ctrl
_t  *const p_ctrl ,  sf_crypto_data_handle_t  *const p_data_out )
```

6.14.32.1 Brief description

Subroutine to process the partial block of input data for the AES GCM cipher operation. This calls the lower level HAL APIs to process the data.

6.14.32.2 Detailed description

End of function sf_crypto_cipher_process_aes_partial_block

Table 208:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_data_out	in	Pointer to data handle of teh output data.

Table 209:Return values

Name	Description
SSP_SUCCESS	Data was processed successfully.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.33 sf_crypto_cipher_aes_encrypt_final

```
ssp_err_t sf_crypto_cipher_aes_encrypt_final ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in ,
 sf_crypto_data_handle_t  *const p_data_out )
```

6.14.33.1 Brief description

Subroutine to finalize the cipher encrypt operation.

6.14.33.2 Detailed description

End of function sf_crypto_cipher_aes_gcm_process_partial_block

Table 210:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.

Table 210:Parameters (Continued)

Name	Direction	Description
p_data_in	inout	Pointer to the input data structure - has the pointer to input data and the data length
p_data_out	inout	Pointer to the output data structure - has the pointer to output data and the data length.

Table 211:Return values

Name	Description
SSP_SUCCESS	Cipher operation was finalized successfully.
SSP_ERR_INVALID_ARGUMENT	Input data is invalid. •
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.33.3 Function steps

- Validate the input parameters
- Process the input data
- Handle padding for ECB, CBC modes
- Add padding and encrypt the last block.
- If there is a remaining partial block process it. For GCM zero padding is automatic
- Now that the possible AAD update and plain text encryption is done, get the GCM tag

6.14.34 sf_crypto_cipher_aes_decrypt_final_check_input_length

```
ssp_err_t sf_crypto_cipher_aes_decrypt_final_check_input_length ( sf_crypto_cipher_instance_ctrl_t  *const p_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in )
```

6.14.34.1 Brief description

Subroutine to check if the total data length to be decrypted is a multiple of the AES block size.

6.14.34.2 Detailed description

End of function sf_crypto_cipher_aes_encrypt_final

Table 212:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_data_in	inout	Pointer to the input data structure - has the pointer to input data and the data length

Table 213:Return values

Name	Description
SSP_SUCCESS	Cipher operation was finalized successfully.
SSP_ERR_INVALID_ARGUMENT	Input data is invalid.

6.14.34.3 Function steps

- If remaining length is not a multiple of block size, return error

6.14.35 sf_crypto_cipher_aes_decrypt_final

```
ssp_err_t sf_crypto_cipher_aes_decrypt_final ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in ,
 sf_crypto_data_handle_t  *const p_data_out )
```

6.14.35.1 Brief description

Subroutine to finalize the cipher decrypt operation.

6.14.35.2 Detailed description

Table 214:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_data_in	inout	Pointer to the input data structure - has the pointer to input data and the data length
p_data_out	inout	Pointer to the output data structure - has the pointer to output data and the data length.

Table 215:Return values

Name	Description
SSP_SUCCESS	Cipher operation was finalized successfully.
SSP_ERR_INVALID_ARGUMENT	Input data is invalid. •
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.35.3 Function steps

- Validate the input parameters
- Process the input data
- If GCM check if there is any partial AAD remaining, if so call to update AAD
- If GCM set the tag
- For GCM compute and verify the tag - return error to indicate if the data is valid or not.

6.14.36 sf_crypto_cipher_aes_is_last_decrypt_block

```
 sf_crypto_cipher_aes_is_last_decrypt_block ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl ,   uint32_t remaining_length )
```

6.14.36.1 Brief description

Subroutine to check whether the the input is the last block for the cipher decrypt operation.

6.14.36.2 Detailed description

End of function sf_crypto_cipher_aes_decrypt_final

Table 216:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
remaining_length	in	Remaining bytes in the input data to be decrypted.

Table 217:Return values

Name	Description
true	Yes, it is the last block.
false	No, it is not the last block.

6.14.37 sf_crypto_cipher_aes_decrypt_last_block

```
ssp_err_t sf_crypto_cipher_aes_decrypt_last_block ( sf_crypto_cipher_instance_ct
rl_t  *const p_ctrl ,   uint8_t * p_input_data ,   uint8_t * p_output_buffer ,
  uint32_t * num_data_bytes )
```

6.14.37.1 Brief description

Subroutine to decrypt the last block of the input data for the cipher decrypt operation.

6.14.37.2 Detailed description

Table 218:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_input_data	in	Pointer to the the input data buffer.
p_output_buffer	inout	Pointer to the output data buffer.
num_data_bytes	inout	Length of the actual data (block minus the padding) is populated on successful output.

Table 219:Return values

Name	Description
SSP_SUCCESS	Cipher operation was finalized successfully.
SSP_ERR_INVALID_ARGUMENT	Input data is invalid.
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.37.3 Function steps

- Decrypt the last block into the buffer provided.
- Strip the padding bytes
- Pick the last byte which is the pad byte
- Check backwards to ensure the number of pad_bytes is the real padding

6.14.38 sf_crypto_cipher_aes_helper_decrypt_last_block

```
ssp_err_t sf_crypto_cipher_aes_helper_decrypt_last_block ( sf_crypto_cipher_inst
ance_ctrl_t  *const p_ctrl ,  uint32_t input_data_length ,  uint8_t
* p_input_data ,  sf_crypto_data_handle_t  *const p_data_out ,
  uint32_t data_out_offset )
```

6.14.38.1 Brief description

helper function to finalize the cipher decrypt operation. It decrypts the last block of data strips the padding bytes and writes the actual data to the output data buffer at the given offset. Also updates the data length on output.

6.14.38.2 Detailed description

End of function sf_crypto_cipher_aes_decrypt_last_block

Table 220:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module instance control block.
input_data_length	in	Length of the input data
p_input_data	in	Pointer to the input data buffer.
p_data_out	inout	Pointer to the output data structure - has the pointer to output data and the data length on input.
data_out_offset	in	Offset at which the data has to be output into p_data_out.p_data.

Table 221:Return values

Name	Description
SSP_SUCCESS	Cipher operation was finalized successfully.
SSP_ERR_INVALID_ARGUMENT	Input data is invalid. •
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.39 sf_crypto_cipher_is_last_block_with_padding

```
 sf_crypto_cipher_is_last_block_with_padding ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl ,   uint32_t remaining_input_length )
```

6.14.39.1 Brief description

helper function to finalize the cipher decrypt operation. This checks if it is the last block to be decrypted and if padding is enabled.

6.14.39.2 Detailed description

Table 222:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
remaining_input_length	in	The number of bytes remaining to be processed in the input data.

Table 223:Return values

Name	Description
padded_block	true if the remaining length is the last block and is padded false otherwise.

6.14.40 sf_crypto_cipher_aes_helper_decrypt_final

```
ssp_err_t sf_crypto_cipher_aes_helper_decrypt_final ( sf_crypto_cipher_instance_
ctrl_t  *const p_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in ,
 sf_crypto_data_handle_t  *const p_data_out )
```

6.14.40.1 Brief description

helper function to finalize the cipher decrypt operation.

6.14.40.2 Detailed description

Table 224:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_data_in	inout	Pointer to the input data structure - has the pointer to input data and the data length
p_data_out	inout	Pointer to the output data structure - has the pointer to output data and the data length.

Table 225:Return values

Name	Description
SSP_SUCCESS	Cipher operation was finalized successfully.
SSP_ERR_INVALID_ARGUMENT	Input data is invalid. •
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.40.3 Function steps

- This will be updated once the data is written to the buffer.
- Check if there is data in the partial block buffer. If so first fill it from the input data
- Update the number of bytes remaining in the input buffer to be processed.
- If data is block length within the partial block in the context buffer, then decrypt it.
- This is last block with padding buffered in the context buffer and we are done,so return
- update the number of bytes written to the output buffer.
- Now that all the data in the partial block is processed, clean it up and set the size to 0
- Check if there is remaining input data

- no more data to process, so return .
- This is last block with padding from the input data stream and we are done,so return
- There are multiple blocks to be decrypted but the last one is padded. So decrypt all but the last one.
- Process the input data stream with num blocks
- update the number of bytes written to the output buffer.
- Update the number of bytes remaining in the input buffer to be processed.
- This is last block with padding from the input data stream and we are done,so return

6.14.41 sf_crypto_cipher_aes_gcm_helper_decrypt_final

```
ssp_err_t sf_crypto_cipher_aes_gcm_helper_decrypt_final ( sf_crypto_cipher_instance_ctrl_t  *const p_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in ,  sf_crypto_data_handle_t  *const p_data_out )
```

6.14.41.1 Brief description

helper function to finalize the AES cipher decrypt operation for GCM .

6.14.41.2 Detailed description

End of function sf_crypto_cipher_aes_helper_decrypt_final

Table 226:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_data_in	inout	Pointer to the input data structure - has the pointer to input data and the data length
p_data_out	inout	Pointer to the output data structure - has the pointer to output data and the data length.

Table 227:Return values

Name	Description
SSP_SUCCESS	Cipher operation was finalized successfully.

Table 227:Return values (Continued)

Name	Description
SSP_ERR_INVALID_ARGUMENT	Input data is invalid. •
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.41.3 Function steps

- This will be updated once the data is written to the buffer.
- Check if there is data in the partial block buffer. If so first fill it from the input data
- Update the number of bytes remaining in the input buffer to be processed.
- If there is a remaining partial block process it. For GCM zero padding is automatic
- update the number of bytes written to the output buffer.
- Now that all the data in the partial block is processed, clean it up and set the size to 0
- If remaining length is 0, we are done else we have more data to process
- update the number of bytes written to the output buffer.

6.14.42 sf_crypto_cipher_aes_handle_encrypt_padding

```
ssp_err_t sf_crypto_cipher_aes_handle_encrypt_padding ( sf_crypto_cipher_instanc
e_ctrl_t  *const p_ctrl ,  sf_crypto_data_handle_t  *const p_data_out )
```

6.14.42.1 Brief description

Subroutine to handle the PKCS#7 padding scheme for the cipher encrypt operation.

6.14.42.2 Detailed description

End of function sf_crypto_cipher_aes_gcm_helper_decrypt_final

Table 228:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.

Table 228:Parameters (Continued)

Name	Direction	Description
p_data_out	inout	Pointer to the output data structure - has the pointer to output data and the data length.

Table 229:Return values

Name	Description
SSP_SUCCESS	Cipher operation was finalized successfully.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.42.3 Function steps

- Add PKCS#7 padding

6.14.43 sf_crypto_cipher_aes_get_gcm_tag

```
ssp_err_t sf_crypto_cipher_aes_get_gcm_tag ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl )
```

6.14.43.1 Brief description

Subroutine to get the GCM tag after the the cipher encrypt operation.

6.14.43.2 Detailed description

End of function sf_crypto_cipher_aes_handle_encrypt_padding

Table 230:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used

Table 231:Return values

Name	Description
SSP_SUCCESS	The GCM tag was created successfully.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.44 sf_crypto_cipher_aes_set_gcm_tag

```
ssp_err_t sf_crypto_cipher_aes_set_gcm_tag ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl )
```

6.14.44.1 Brief description

Subroutine to set the GCM tag for the the cipher decrypt operation.

6.14.44.2 Detailed description

End of function sf_crypto_cipher_aes_get_gcm_tag

Table 232:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.

Table 233:Return values

Name	Description
SSP_SUCCESS	The GCM tag was created successfully.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.45 sf_crypto_cipher_aes_compute_verify_gcm_tag

```
ssp_err_t sf_crypto_cipher_aes_compute_verify_gcm_tag ( sf_crypto_cipher_instanc
e_ctrl_t  *const p_ctrl )
```

6.14.45.1 Brief description

Subroutine to compute and verify the GCM tag for after the cipher decrypt operation.

6.14.45.2 Detailed description

End of function sf_crypto_cipher_aes_set_gcm_tag

Table 234:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.

Table 235:Return values

Name	Description
SSP_SUCCESS	The GCM tag was computed and verified successfully.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.46 sf_crypto_cipher_aes_helper_validate_final_params

```
ssp_err_t sf_crypto_cipher_aes_helper_validate_final_params ( sf_crypto_cipher_instance_ctrl_t  *const p_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in )
```

6.14.46.1 Brief description

Subroutine to validate the parameters for the AES cipher final operation. This function only checks if any input data is received for the final operation. *.

6.14.46.2 Detailed description

End of function sf_crypto_cipher_aes_compute_verify_gcm_tag

Table 236:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call. in the open() call.
p_data_in	inout	Pointer to the input data structure - has the pointer to input data and the data length

Table 237:Return values

Name	Description
SSP_SUCCESS	The parameters were verified successfully.
SSP_ERR_INVALID_ARGUMENT	No input data was provided either to the cipherUpdate() or the cipherFinal operation.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.46.3 Function steps

- GCM is the only mode that can take no input data but just create a tag For other modes: If no input data is provided in update operation and none in final, return error. If the status is SF_CRYPTO_CIPHER_STATE_INITIALIZED, it means this is a one shot final operation and no data was provided through the Update call.

6.14.47 sf_crypto_cipher_aes_validate_encrypt_final_out_buffer

```
ssp_err_t sf_crypto_cipher_aes_validate_encrypt_final_out_buffer ( sf_crypto_dat
a_handle_t  *const p_data_out ,   uint32_t required_length )
```

6.14.47.1 Detailed description

End of function sf_crypto_cipher_validate_aes_final_params

6.14.48 sf_crypto_cipher_aes_validate_encrypt_final_params

```
ssp_err_t sf_crypto_cipher_aes_validate_encrypt_final_params ( sf_crypto_cipher_
instance_ctrl_t  *const p_ctrl ,  sf_crypto_data_handle_t  const
*const p_data_in ,  sf_crypto_data_handle_t  *const p_data_out )
```

6.14.48.1 Brief description

Subroutine to validate the parameters for the final encrypt operation.

6.14.48.2 Detailed description

Table 238:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_data_in	inout	Pointer to the input data structure - has the pointer to input data and the data length
p_data_out	inout	Pointer to the output data structure - has the pointer to output data and the data length

Table 239:Return values

Name	Description
SSP_SUCCESS	The parameters were verified successfully.
SSP_ERR_INVALID_ARGUMENT	Input data is not a multiple of block size.
SSP_ERR_INVALID_SIZE	Output data buffer is inadequate to hold the output data.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.48.3 Function steps

- GCM is the only mode that gets zero padded internally. Unless a padding scheme is selected, for rest of the modes the data has to be a multiple of the block size.
- Check if the output buffer is big enough
- If a padding scheme is selected and mode is not GCM, whether or not there are partial bytes remaining, padding will be applied.

6.14.49 sf_crypto_cipher_aes_validate_decrypt_final_params

```
ssp_err_t sf_crypto_cipher_aes_validate_decrypt_final_params ( sf_crypto_cipher_
instance_ctrl_t  *const p_ctrl ,  sf_crypto_data_handle_t  const
*const p_data_in ,  sf_crypto_data_handle_t  *const p_data_out )
```

6.14.49.1 Brief description

Subroutine to validate the parameters for the final decrypt operation.

6.14.49.2 Detailed description

End of function sf_crypto_cipher_validate_aes_encrypt_final_params

Table 240:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_data_in	inout	Pointer to the input data structure - has the pointer to input data and the data length
p_data_out	inout	Pointer to the output data structure - has the pointer to output data and the data length

Table 241:Return values

Name	Description
SSP_SUCCESS	The parameters were verified successfully.
SSP_ERR_INVALID_ARGUMENT	input data is not a multiple of block size.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.49.3 Function steps

- Except for GCM - For decryption, the data must be a multiple of block size

- Check if the output buffer is big enough. Note: If there is partial data in GCM, it will be zero padded internally for calculations but the data returned will be the same size as the input data.

6.14.50 sf_crypto_cipher_aes_hal_encrypt_decrypt

```
ssp_err_t sf_crypto_cipher_aes_hal_encrypt_decrypt ( sf_crypto_cipher_instance_ctrl_t  *const p_ctrl ,   uint32_t * p_key ,   uint32_t * p_iv ,
  uint32_t num_words ,   uint32_t * p_input ,   uint32_t * p_output )
```

6.14.50.1 Brief description

Subroutine calls the HAL encrypt / decrypt APIs as pointed to by the HAL interface in the control block. It is assumed that the control block is already populated with the interface for the HAL AES APIs. *.

6.14.50.2 Detailed description

End of function sf_crypto_cipher_validate_aes_decrypt_final_params

Table 242:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the Cipher framework module instance control block.
p_key	in	Pointer to the key to be used for the cipher operation.
p_iv	in	Pointer to the IV to be used for the cipher operation.
num_words	in	Number of words of input data for the cipher operation.
p_input	in	Pointer to the input data for the cipher operation.
p_output	in	Pointer to the IV to output data for the cipher operation.

Table 243:Return values

Name	Description
SSP_SUCCESS	The cipher update operation was successful.

Table 243:Return values (Continued)

Name	Description
SSP_ERR_UNSUPPORTED	The module does not support the algorithm bbased on the key type specified by the user.
Others	Crypto HAL module returned an error or resource lock/unlock was failed. See Common Error Codes for other possible return codes.

6.14.51 sf_crypto_cipher_aes_final

```
ssp_err_t sf_crypto_cipher_aes_final ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in ,
 sf_crypto_data_handle_t  *const p_data_out )
```

6.14.51.1 Brief description

Sub-routine for Crypto Cipher Framework to call the appropriate routine for final encryption based on the algorithm for the cipherFinal operation. This function is called by SF_CRYPTO_CIPHER_CipherFinal.

6.14.51.2 Detailed description

End of function sf_crypto_cipher_aes_hal_encrypt_decrypt

Table 244:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the Cipher framework module instance control block.
p_data_in	in	Pointer to the input data buffer and the length of the input data.
p_data_out	inout	Pointer to the output data buffer and the buffer length on input. If data is output, the length of the data will be updated.

Table 245:Return values

Name	Description
SSP_SUCCESS	The cipher update operation was successful.

Table 245:Return values (Continued)

Name	Description
SSP_ERR_UNSUPPORTED	The module does not support the algorithm bbased on the key type specified by the user.
Others	Crypto HAL module returned an error or resource lock/unlock was failed. See Common Error Codes for other possible return codes.

6.14.51.3 Function steps

- Validate that there is some input data provided / processed before the finalizing operation Applies to both encrypt and decrypt operations. AES GCM is the only exception.

6.14.52 sf_crypto_cipher_aes_context_buffer_pointer_data_zeroise

```
 sf_crypto_cipher_aes_context_buffer_pointer_data_zeroise ( sf_crypto_cipher_instance_ctrl_t  *const p_ctrl )
```

6.14.52.1 Brief description

A subroutine to zeroise the memory pointed to by the fields within the context buffer. NOTE: the buffer sizes pointed to by the original allocation should NOT be cleared. Those sizes will be used when freeing up the memory.

6.14.52.2 Detailed description

End of function sf_crypto_cipher_aes_final

Table 246:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the Cipher framework module instance control block.

6.14.53 sf_crypto_cipher_instance_aes_memory_deallocate

```
ssp_err_t sf_crypto_cipher_instance_aes_memory_deallocate ( sf_crypto_cipher_instance_ctrl_t  *const p_ctrl )
```

6.14.53.1 Brief description

A subroutine to release the memory allocated for context buffer. All the data pointed to by the context buffer is first zeroised and freed. Next all elements of the context buffer are zeroised and pointers are set to NULL. Lastly the context buffer itself is freed.

6.14.53.2 Detailed description

End of function sf_crypto_cipher_aes_context_buffer_pointer_data_zeroise

Table 247:Parameters

Name	Direction	Description
p_ctrl	inout	The instance control block for the Cipher module.

Table 248:Return values

Name	Description
SSP_SUCCESS	The memory for this instance was freed successfully.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.

6.14.53.3 Function steps

- First zeroise the data pointed to from the context buffer and then release the memory resources used for this instance
- Release the memory allocated for Partial Block
- Release the memory allocated for Key
- Release the memory allocated for IV
- Release the memory allocated for AAD partial Block and tag
- Zeroise the context buffer and set all pointers to NULL
- Release the memory allocated for the context buffer.

6.14.54 sf_crypto_cipher_aes_zeroise_context_buffer

```
 sf_crypto_cipher_aes_zeroise_context_buffer ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl )
```

6.14.54.1 Brief description

A subroutine to init / reset the context buffer. All elements are zeroised and pointers are set to NULL.

6.14.54.2 Detailed description

End of function sf_crypto_cipher_instance_aes_memory_deallocate

Table 249:Parameters

Name	Direction	Description
p_ctrl	inout	The instance control block for the Cipher module.

6.14.55 sf_crypto_cipher_instance_aes_gcm_memory_release

```
ssp_err_t sf_crypto_cipher_instance_aes_gcm_memory_release ( sf_crypto_cipher_instance_ctrl_t  *const p_ctrl )
```

6.14.55.1 Brief description

A subroutine to free the memory allocated for the GCM data in the context buffer. All elements are zeroised and pointers are set to NULL.

6.14.55.2 Detailed description

End of function sf_crypto_cipher_aes_zeroise_context_buffer

Table 250:Parameters

Name	Direction	Description
p_ctrl	inout	The instance control block for the Cipher module.

Table 251:Return values

Name	Description
SSP_SUCCESS	The GCM data in the context buffer for this instance was freed successfully.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.

6.14.56 sf_crypto_cipher_initialize_aes_instance

```
ssp_err_t sf_crypto_cipher_initialize_aes_instance ( sf_crypto_cipher_instance_ctrl_t  * p_ctrl ,  sf_crypto_cipher_cfg_t  const *const p_cfg )
```

6.14.56.1 Brief description

Allocates memory for the instance and opens the underlying HAL instance.

6.14.56.2 Detailed description

End of function sf_crypto_cipher_instance_aes_gcm_memory_release

Table 252:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to Crypto Cipher Framework instance control block structure.
p_cfg	in	Pointer to the configuration structure for Cipher module .

Table 253:Return values

Name	Description
SSP_SUCCESS	The module instantiated successfully.
SSP_ERR_TIMEOUT	Was unable to allocate the memory within the specified time to wait.
SP_ERR_OUT_OF_MEMORY	Requested size is zero or larger than the pool.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.
Others	Crypto HAL module returned an error or resource lock/unlock was failed. See Common Error Codes or HAL driver for other possible return codes or causes. This function calls: sf_crypto_cipher_instance_memory_allocate sf_crypto_cipher_hal_open

6.14.57 sf_crypto_cipher_deinitialize_aes_instance

```
ssp_err_t sf_crypto_cipher_deinitialize_aes_instance ( sf_crypto_cipher_instance
_ctrl_t  * p_ctrl )
```

6.14.57.1 Brief description

closes the underlying HAL instance and releases the memory for the instance.

6.14.57.2 Detailed description

End of function sf_crypto_cipher_initialize_aes_instance

Table 254:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to Crypto Cipher Framework instance control block structure.

Table 255:Return values

Name	Description
SSP_SUCCESS	The module instantiated successfully.
SSP_ERR_TIMEOUT	Was unable to allocate the memory within the specified time to wait.
SP_ERR_OUT_OF_MEMORY	Requested size is zero or larger than the pool.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.
Others	Crypto HAL module returned an error or resource lock/unlock was failed. See Common Error Codes or HAL driver for other possible return codes or causes. This function calls: sf_crypto_cipher_instance_memory_allocate sf_crypto_cipher_hal_open

6.14.58 SF_CRYPTO_CIPHER_RSA_EB_SIZE_BYTES

```
SF_CRYPTO_CIPHER_RSA_EB_SIZE_BYTES (  uint32_t key_size_bits )
```

6.14.58.1 Brief description

Routine to calculate the Encryption block size based on the key size.

6.14.58.2 Detailed description

Table 256:Parameters

Name	Direction	Description
key_size_bits	in	key size in bits.

encryption block size in bytes

6.14.59 SF_CRYPTO_CIPHER_RSA_PKCS_1_5_EB_DATA_SIZE_BYTES

```
SF_CRYPTO_CIPHER_RSA_PKCS_1_5_EB_DATA_SIZE_BYTES (  uint32_t key_size_bits )
```

6.14.59.1 Brief description

Routine to calculate the Encryption block size based on the key size.

6.14.59.2 Detailed description

Table 257:Parameters

Name	Direction	Description
key_size_bits	in	key size in bits.

encryption block size in bytes for PKCS#1 v1.5 encoding

6.14.60 sf_crypto_cipher_rsa_hal_open

```
ssp_err_t sf_crypto_cipher_rsa_hal_open ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl ,  sf_crypto_cipher_cfg_t  const *const p_cfg )
```

6.14.60.1 Brief description

Subroutine to open a Crypto RSA HAL module. This function is called by SF_CRYPTO_CIPHER_Open.

6.14.60.2 Detailed description

Table 258:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a Crypto Key Framework module control block
p_cfg	inout	Pointer to a Crypto Key Framework module configuration structure

Table 259:Return values

Name	Description
SSP_SUCCESS	RSA HAL module is successfully opened.
SSP_ERR_OUT_OF_MEMORY	Failed to allocate memory to store RSA HAL module control block.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.
Others	RSA HAL module returned an error. See Common Error Codes for other possible return codes.

6.14.60.3 Function steps

- Get a Crypto common control block and the interface.
- Allocate memory for a Crypto HAL control block from the byte pool.
- Set Crypto HAL API instance with the control common hardware api.
- Open the Crypto HAL RSA module.

6.14.61 sf_crypto_cipher_rsa_hal_close

```
ssp_err_t sf_crypto_cipher_rsa_hal_close ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl )
```

6.14.61.1 Brief description

Subroutine to close a Crypto RSA HAL module. This function is called by SF_CRYPTO_CIPHER_Close.

6.14.61.2 Detailed description

Table 260:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to a Crypto Key Framework module control block

Table 261:Return values

Name	Description
SSP_SUCCESS	RSA HAL module is successfully closed.
Others	RSA HAL module returned an error. See Common Error Codes for other possible return codes.

6.14.61.3 Function steps

- Close the Crypto HAL driver.
- Release the control block memory back to byte pool

6.14.62 sf_crypto_cipher_rsa_interface_get

```
 sf_crypto_cipher_rsa_interface_get ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl )
```

6.14.62.1 Brief description

Subroutine to get a RSA HAL API instance. This function is called by sf_crypto_cipher_open_rsa().

6.14.62.2 Detailed description

Table 262:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a Key framework control block, whose p_hal_api filled with HAL RSA interface. This indicates NULL, for not supported MCUs

6.14.62.3 Function steps

- Get a Crypto common control block and the HAL instance.
- Check the RSA key type.
- Check the RSA key type.
- Get the HAL API instance for a selected algorithm type.

6.14.63 sf_crypto_cipher_rsa_private_free_context_buffer_key

```
ssp_err_t sf_crypto_cipher_rsa_private_free_context_buffer_key ( sf_crypto_ciphe
r_instance_ctrl_t  *const p_ctrl )
```

6.14.63.1 Brief description

Subroutine to free memory allocated for the key in the context buffer for RSA cipher operation.

6.14.63.2 Detailed description

Table 263:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a Crypto Key Framework module control block.

Table 264:Return values

Name	Description
SSP_SUCCESS	Memory allocation was successful.
SSP_ERR_TIMEOUT	Was unable to allocate the memory within the specified time to wait.
SSP_ERR_OUT_OF_MEMORY	Requested size is zero or larger than the pool.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error. *

6.14.63.3 Function steps

- Release the memory allocated for the RSA Context Buffer

6.14.64 sf_crypto_cipher_rsa_private_free_context_buffer

```
ssp_err_t sf_crypto_cipher_rsa_private_free_context_buffer ( sf_crypto_cipher_instance_ctrl_t  *const p_ctrl )
```

6.14.64.1 Brief description

Subroutine to free memory allocated for the context buffer for RSA cipher operation.

6.14.64.2 Detailed description

Table 265:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a Crypto Key Framework module control block.

Table 266:Return values

Name	Description
SSP_SUCCESS	Memory allocation was successful.
SSP_ERR_TIMEOUT	Was unable to allocate the memory within the specified time to wait.
SSP_ERR_OUT_OF_MEMORY	Requested size is zero or larger than the pool.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error. *

6.14.64.3 Function steps

- Release the memory allocated for the RSA Context Buffer

6.14.65 sf_crypto_cipher_instance_rsa_memory_allocate

```
ssp_err_t sf_crypto_cipher_instance_rsa_memory_allocate ( sf_crypto_cipher_instance_ctrl_t  *const p_ctrl )
```

6.14.65.1 Brief description

Subroutine to allocate memory required for the context buffer for RSA cipher operation.

6.14.65.2 Detailed description

End of function sf_crypto_cipher_rsa_memory_free_context_buffer

Table 267:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a Crypto Key Framework module control block.

Table 268:Return values

Name	Description
SSP_SUCCESS	Memory allocation was successful.
SSP_ERR_TIMEOUT	Was unable to allocate the memory within the specified time to wait.
SSP_ERR_OUT_OF_MEMORY	Requested size is zero or larger than the pool.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.
SSP_ERR_INTERNAL	An internal error occurred during processing.

6.14.65.3 Function steps

- Allocate memory for the RSA context buffer. Extra 3 bytes are allocated to get the buffer WORD aligned
- Zeroise the context buffer and set all pointers to NULL
- Allocate memory for key in the context buffer Extra 3bytes are allocated to get the buffer WORD aligned
- Allocate memory for partial block. Extra 3bytes are allocated to get the buffer WORD aligned.
- Not checking error here as we are already sending a mem alloc error. Free the memory for the key and then the context buffer itself.

6.14.66 sf_crypto_cipher_is_key_type_rsa

```
sf_crypto_cipher_is_key_type_rsa ( sf_crypto_key_type_t  key_type )
```

6.14.66.1 Brief description

Subroutine to check if key type is RSA.

6.14.66.2 Detailed description

Table 269:Parameters

Name	Direction	Description
key_type	in	Key type enum.

Table 270:Return values

Name	Description
SSP_SUCCESS	Memory allocation was successful.
SSP_ERR_TIMEOUT	Was unable to allocate the memory within the specified time to wait.
SSP_ERR_OUT_OF_MEMORY	Requested size is zero or larger than the pool.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error. *

6.14.66.3 Function steps

- Check the key type is valid for RSA operations.

6.14.67 sf_crypto_cipher_rsa_init

```
ssp_err_t sf_crypto_cipher_rsa_init ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl ,  sf_crypto_cipher_op_mode_t  cipher_operation_mode ,
 sf_crypto_data_handle_t  const *const p_key ,
 sf_crypto_cipher_algorithm_init_params_t  * p_algorithm_specific_params )
```

6.14.67.1 Brief description

Routine to handle the cipherInit RSA operation.

6.14.67.2 Detailed description

Table 271:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
cipher_operation_mode	in	
p_key	in	
p_algorithm_specific_params	in	

Table 272:Return values

Name	Description
SSP_SUCCESS	Cipher operation was updated successfully.
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.67.3 Function steps

- All buffers within the context buffer are allocated at Open. Now just zeroise their contents.
- Copy data into the control block / context buffer.

6.14.68 sf_crypto_cipher_rsa_update

```
ssp_err_t sf_crypto_cipher_rsa_update ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in ,
 sf_crypto_data_handle_t  *const p_data_out )
```

6.14.68.1 Brief description

Subroutine to update the cipher RSA operation. This algorithm is only suitable for messages of limited length. The total number of input bytes processed during encryption may not be more than k-11, where k is the RSA key's modulus size in bytes. The encryption block(EB) during encryption with a Public key is built as follows: EB = 00 || 02 || PS || 00 || M :: M (input bytes) is the plaintext message :: PS is an octet string of length k-3-||M|| of pseudo random nonzero octets. The length of PS must be at least 8 octets. :: k is the RSA modulus size.

6.14.68.2 Detailed description

Table 273:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_data_in	in	Pointer to the input data structure - has the pointer to input data and the data length
p_data_out	inout	Pointer to the output data structure - has the pointer to output data and the buffer length on input. If data is filled the length is updated on output.

Table 274:Return values

Name	Description
SSP_SUCCESS	Cipher operation was updated successfully. •
SSP_ERR_UNSUPPORTED	Unknown cipher operation mode was passed in.
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.68.3 Function steps

- input parameters validation - done at the entry nothing to do here no output on update - so no checks for output buffer / length collect input up to the block size for encryption when no padding is selected. collect data up to block size -11 for encryption when padding is selected collect input up to the block size for decryption if it exceeds the above return error
- Copy the data into the partial block

6.14.69 sf_crypto_cipher_rsa_helper_pkcs_1_5_encode_block

```
ssp_err_t sf_crypto_cipher_rsa_helper_pkcs_1_5_encode_block ( sf_crypto_cipher_instance_ctrl_t  *const p_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in )
```

6.14.69.1 Brief description

Helper routine to encode the PKCS#1 v1.5 encoding.

6.14.69.2 Detailed description

Table 275:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_data_in	inout	Pointer to the input data structure - has the pointer to input data and the data length

Table 276:Return values

Name	Description
SSP_SUCCESS	Cipher operation was updated successfully.
SSP_ERR_MESSAGE_TOO_LONG	The input data / message is too long for the given operation.
SSP_ERR_INTERNAL	An internal error occurred during processing.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.69.3 Function steps

- Copy any input data into the partial block.

6.14.70 sf_crypto_cipher_rsa_encrypt_final

```
ssp_err_t sf_crypto_cipher_rsa_encrypt_final ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in ,
 sf_crypto_data_handle_t  *const p_data_out )
```

6.14.70.1 Brief description

Subroutine to finalize the cipher RSA encrypt operation.

6.14.70.2 Detailed description

Table 277:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_data_in	inout	Pointer to the input data structure - has the pointer to input data and the data length
p_data_out	inout	Pointer to the output data structure - has the pointer to output data and the data length.

Table 278:Return values

Name	Description
SSP_SUCCESS	Cipher operation was updated successfully.
SSP_ERR_MESSAGE_TOO_LONG	The input data / message is too long for the given operation.
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.
SSP_ERR_UNSUPPORTED	Unknown padding scheme supplied.
SSP_ERR_INTERNAL	An internal error occurred during processing.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.70.3 Function steps

- At this point we have a block which has to be encrypted by the HAL API

6.14.71 sf_crypto_cipher_rsa_hal_encrypt_decrypt

```
ssp_err_t sf_crypto_cipher_rsa_hal_encrypt_decrypt ( sf_crypto_cipher_instance_ctrl_t  *const p_ctrl ,   uint32_t * p_key ,   uint32_t * p_domain ,
  uint32_t num_words ,   uint32_t * p_input ,   uint32_t * p_output )
```

6.14.71.1 Brief description

Subroutine to call the HAL RSA encrypt/decrypt API.

6.14.71.2 Detailed description

Table 279:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_key	in	Pointer to the ket used for the cipher operation.
p_domain	in	Pointer to the domain parameters for the cipher operation - NULL for RSA.
num_words	in	Number of words supplied as the input data.
p_input	in	Pointer to the input data for the cipher operation.
p_output	in	Pointer to the output data for the cipher operation.

Table 280:Return values

Name	Description
SSP_SUCCESS	Cipher operation was updated successfully.

Table 280:Return values (Continued)

Name	Description
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.72 sf_crypto_cipher_rsa_pkcs_1_5_encode_block

```
ssp_err_t sf_crypto_cipher_rsa_pkcs_1_5_encode_block ( sf_crypto_cipher_instance
_ctrl_t  *const p_ctrl )
```

6.14.72.1 Brief description

Subroutine to format the Encryption Block as per the PKCS#1 v1.5 for RSA encrypt operation. The partial data block in the context buffer will be formatted in place to avoid allocating extra memory from the byte pool or creating a block on the stack.

6.14.72.2 Detailed description

Table 281:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used

Table 282:Return values

Name	Description
SSP_SUCCESS	Cipher operation was updated successfully.
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.72.3 Function steps

- message_end points to the last byte of the message to be encoded.
- Get the encryption block size.
- buffer_end points to the last byte of the EB to be encoded.

- move the data to the end of the buffer to format the Encryption Block EB in place in the context buffer EB = 00 || BT || PS || 00 || D
- Now get random numbers to fill the Padding String PS Also ensure that the random numbers so not contain any zeroes.

6.14.73 sf_crypto_cipher_rsa_decrypt_final

```
ssp_err_t sf_crypto_cipher_rsa_decrypt_final ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in ,
 sf_crypto_data_handle_t  *const p_data_out )
```

6.14.73.1 Brief description

Subroutine to finalize the cipher RSA decrypt operation.

6.14.73.2 Detailed description

Table 283:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the cipher framework module control block used in the open() call.
p_data_in	inout	Pointer to the input data structure - has the pointer to input data and the data length
p_data_out	inout	Pointer to the output data structure - has the pointer to output data and the data length.

Table 284:Return values

Name	Description
SSP_SUCCESS	Cipher operation was updated successfully.
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.
SSP_ERR_INTERNAL	An internal error occurred during processing.
See	Common Error Codes or HAL driver for other possible return codes or causes.

6.14.73.3 Function steps

- Copy any input data into the partial block.
- At this point we have a block which has to be decrypted by the HAL API
- Before copying data into the output buffer ensure that the output buffer is of sufficient size.

6.14.74 sf_crypto_cipher_rsa_pkcs_1_5_decode_block

```
ssp_err_t sf_crypto_cipher_rsa_pkcs_1_5_decode_block ( sf_crypto_cipher_instance
_ctrl_t  *const p_ctrl )
```

6.14.74.1 Function steps

- Check if the Encoded Message format is correct
- check if the data separator is at least 8 octets away or the Padding string is at least eight octets long.

6.14.75 sf_crypto_cipher_rsa_validate_encrypt_final_out_params

```
ssp_err_t sf_crypto_cipher_rsa_validate_encrypt_final_out_params ( sf_crypto_cip
her_instance_ctrl_t  *const p_ctrl ,  sf_crypto_data_handle_t
*const p_data_out )
```

6.14.75.1 Brief description

Sub-routine for Crypto Cipher Framework to validate the output data bufferfor final RSA encryption.

6.14.75.2 Detailed description

Table 285:Return values

Name	Description
SSP_SUCCESS	Validation operation was successful.
SSP_ERR_INVALID_SIZE	The out buffer is inadequate to hold the output data.
SSP_ERR_INTERNAL	An internal error occurred during processing.

6.14.75.3 Function steps

- Get the encryption block size.
- If the output data buffer length is insufficient to hold the encryption data return error.

6.14.76 sf_crypto_cipher_rsa_final_input_buffer_check

```
ssp_err_t sf_crypto_cipher_rsa_final_input_buffer_check ( sf_crypto_cipher_instance_ctrl_t  *const p_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in )
```

6.14.76.1 Brief description

Sub-routine for Crypto Cipher Framework to call the appropriate routine for final encryption based on the algorithm for the cipherFinal operation. This function is called by SF_CRYPTO_CIPHER_CipherFinal.

6.14.76.2 Detailed description

Table 286:Return values

Name	Description
SSP_SUCCESS	The cipher update operation was successful.
SSP_ERR_INVALID_ARGUMENT	One or more input parameters are invalid.

6.14.76.3 Function steps

- If input data is NULL and no data has been buffered through an update operation, return error This applies to both encryption and decryption operations. Hence check at the top before branching.
- Check if input data length is non zero but buffer is NULL

6.14.77 sf_crypto_cipher_rsa_final

```
ssp_err_t sf_crypto_cipher_rsa_final ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in ,
 sf_crypto_data_handle_t  *const p_data_out )
```

6.14.77.1 Brief description

Routine to process the final encryption / decryption operation for RSA. This function is called by SF_CRYPTO_CIPHER_CipherFinal.

6.14.77.2 Detailed description

End of function sf_crypto_cipher_rsa_final_input_buffer_check

Table 287:Return values

Name	Description
SSP_SUCCESS	The cipher update operation was successful.
SSP_ERR_UNSUPPORTED	The module does not support the algorithm bbased on the key type specified by the user.
Others	Crypto HAL module returned an error or resource lock/unlock was failed. See Common Error Codes for other possible return codes.

6.14.77.3 Function steps

- Clean up the data in the context buffer before exiting.

6.14.78 sf_crypto_cipher_rsa_context_buffer_pointer_data_zeroise

```
 sf_crypto_cipher_rsa_context_buffer_pointer_data_zeroise ( sf_crypto_cipher_ins
tance_ctrl_t  *const p_ctrl )
```

6.14.78.1 Detailed description

A subroutine to zeroise the memory pointed to by the fields within the context buffer. NOTE: the buffer sizes pointed to by the original allocation should NOT be cleared. That size will be used when freeing up the memory.

Table 288:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the Cipher framework module control block.

6.14.79 sf_crypto_cipher_instance_rsa_memory_deallocate

```
ssp_err_t sf_crypto_cipher_instance_rsa_memory_deallocate ( sf_crypto_cipher_ins
tance_ctrl_t  *const p_ctrl )
```

6.14.79.1 Detailed description

A subroutine to release the memory allocated for context buffer. All the data pointed to by the context buffer is first zeroised and freed. Next all elements of the context buffer are zeroised and pointers are set to NULL.

Table 289:Parameters

Name	Direction	Description
p_ctrl	inout	The control block for the Cipher module.

6.14.79.2 Function steps

- First zeroise the data pointed to from the context buffer and then release the memory resources used for this instance
- Release the memory allocated for Partial Block
- Release the memory allocated for Key
- Zeroise the context buffer and set all pointers to NULL
- Release the memory allocated for the context buffer

6.14.80 sf_crypto_cipher_rsa_zeroise_context_buffer

```
 sf_crypto_cipher_rsa_zeroise_context_buffer ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl )
```

6.14.80.1 Detailed description

A subroutine to init / reset the RSA context buffer. All elements are zeroised and pointers are set to NULL.

Table 290:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the Cipher control block structure passed in at open().

6.14.81 sf_crypto_cipher_get_rsa_pvt_key_size_bytes

```
ssp_err_t sf_crypto_cipher_get_rsa_pvt_key_size_bytes ( sf_crypto_key_type_t  ke
y_type ,  sf_crypto_key_size_t  key_size ,  uint32_t * key_size_bytes )
```

6.14.81.1 Detailed description

End of function sf_crypto_cipher_rsa_zeroise_context_buffer A subroutine to calculate the RSA key size in bytes.

Table 291:Parameters

Name	Direction	Description
key_type	in	The enum for key type.
key_size	in	The enum for key size.
key_size_bytes	inout	The size of the private key in bytes.

Table 292:Return values

Name	Description
SSP_SUCCESS	Key size in bytes returned successfully.
SSP_ERR_INTERNAL	Internal error occurred during processing.
SSP_ERR_INVALID_ARGUMENT	One of the input arguments is invalid.

6.14.82 sf_crypto_cipher_get_rsa_public_key_size_bytes

```
ssp_err_t sf_crypto_cipher_get_rsa_public_key_size_bytes ( sf_crypto_key_size_t
 key_size ,   uint32_t * key_size_bytes )
```

6.14.82.1 Detailed description

A subroutine to calculate the RSA Public key size in bytes.

Table 293:Parameters

Name	Direction	Description
key_size	in	enum for key size.
key_size_bytes	inout	key size in bytes filled on return.

Table 294:Return values

Name	Description
SSP_SUCCESS	Public key is computed and returned successfully.

Table 294:Return values (Continued)

Name	Description
SSP_ERR_INTERNAL	An error occurred during processing.

6.14.83 sf_crypto_cipher_get_rsa_modulus_size_bits

```
ssp_err_t sf_crypto_cipher_get_rsa_modulus_size_bits ( sf_crypto_key_size_t  key
_size ,   uint32_t * key_size_bits )
```

6.14.83.1 Detailed description

A subroutine to calculate the RSA modulus size in bits.

Table 295:Parameters

Name	Direction	Description
key_size	in	The key size of enum type sf_crypto_key_size_t
key_size_bits	inout	The RSA Modulus size in bits for the given RSA Key size.

Table 296:Return values

Name	Description
SSP_SUCCESS	The modulus size is returned successfully.
SSP_ERR_INVALID_ARGUMENT	The key_size input param is invalid.

6.14.84 sf_crypto_cipher_get_rsa_block_size_bytes

```
ssp_err_t sf_crypto_cipher_get_rsa_block_size_bytes ( sf_crypto_key_size_t  key_
size ,   uint32_t * block_size_bytes )
```

6.14.84.1 Detailed description

A subroutine to calculate the RSA encryption block size / modulus size in bytes. The block size is same as the size of the RSA modulus.

Table 297:Parameters

Name	Direction	Description
key_size	in	The key size of enum type sf_crypto_key_size_t

Table 298:Return values

Name	Description
block_size_bytes	The RSA Block size in bytes for the given RSA Key size.

6.14.85 sf_crypto_cipher_rsa_get_modulus_block_sizes

```
ssp_err_t sf_crypto_cipher_rsa_get_modulus_block_sizes ( sf_crypto_cipher_instan
ce_ctrl_t  *const p_ctrl ,   uint32_t * key_size_bits ,   uint32_t
* block_size_bytes )
```

6.14.86 sf_crypto_cipher_rsa_encrypt_update

```
ssp_err_t sf_crypto_cipher_rsa_encrypt_update ( sf_crypto_cipher_instance_ctrl_t
  *const p_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in ,
 sf_crypto_data_handle_t  *const p_data_out )
```

6.14.86.1 Detailed description

A routine to handle the cipherUpdate - encrypt operation for RSA.

Input parameters validation - done at the entry nothing to do here No output on update - so no checks for output buffer / length Collect input up to the block size for encryption when no padding is selected. Collect data up to block size -11 for encryption when padding is selected if it exceeds the above return error

Table 299:Parameters

Name	Direction	Description
p_ctrl	inout	The control block for the Cipher module.
p_data_in	in	The pointer to the input data (structure with pointer to data and data length.)

Table 299:Parameters (Continued)

Name	Direction	Description
p_data_out	in	The pointer to the input data (structure with pointer to data and data length.)

Table 300:Return values

Name	Description
SSP_ERR_INTERNAL	Internal error occurred during processing.
SSP_SUCCESS	Update operation was completed successfully.

6.14.86.2 Function steps

- since no data is output, set output data length to 0.

6.14.87 sf_crypto_cipher_rsa_decrypt_update

```
ssp_err_t sf_crypto_cipher_rsa_decrypt_update ( sf_crypto_cipher_instance_ctrl_t
  *const p_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in ,
 sf_crypto_data_handle_t  *const p_data_out )
```

6.14.87.1 Detailed description

End of function sf_crypto_cipher_rsa_encrypt_update A routine to handle the cipherUpdate - decrypt operation for RSA.

Input parameters validation - done at the entry nothing to do here No output on update - so no checks for output buffer / length collect input up to the block size for decryption. if it exceeds the above return error

Table 301:Parameters

Name	Direction	Description
p_ctrl	inout	The control block for the Cipher module.
p_data_in	in	The pointer to the input data (structure with pointer to data and data length).

Table 301:Parameters (Continued)

Name	Direction	Description
p_data_out	in	The pointer to the input data (structure with pointer to data and data length).

Table 302:Return values

Name	Description
SSP_SUCCESS	The validation of the init parameters for the RSA cipher operation was successful.
SSP_ERR_MESSAGE_TOO_LONG	The input data / message is too long for the given operation..
SSP_ERR_INVALID_CALL	This call is invalid and no operation can occur.

6.14.88 sf_crypto_cipher_rsa_init_validate_padding_scheme

```
ssp_err_t sf_crypto_cipher_rsa_init_validate_padding_scheme ( sf_crypto_cipher_padding_scheme_t  padding_scheme )
```

6.14.88.1 Detailed description

End of function sf_crypto_cipher_rsa_decrypt_update A subroutine to validate the parameters for the RSA padding scheme.

Table 303:Parameters

Name	Direction	Description
padding_scheme	in	Specifies RSA padding scheme

Table 304:Return values

Name	Description
SSP_SUCCESS	The validation of the init parameters for the RSA cipher operation was successful.
SSP_ERR_INVALID_ARGUMENT	At least one of the input parameters are invalid.

6.14.89 sf_crypto_cipher_rsa_init_validate

```
ssp_err_t sf_crypto_cipher_rsa_init_validate ( sf_crypto_cipher_instance_ctrl_t
*const p_ctrl ,  sf_crypto_cipher_op_mode_t  cipher_operation_mode ,
 sf_crypto_data_handle_t  const *const p_key ,
 sf_crypto_cipher_rsa_init_params_t  * p_rsa_params )
```

6.14.89.1 Detailed description

End of function sf_crypto_cipher_rsa_init_validate_padding_scheme A subroutine to validate the parameters for the RSA cipherInit operation.

Table 305:Parameters

Name	Direction	Description
p_ctrl	inout	The control block for the Cipher module.
cipher_operation_mode	in	The operation mode of enum type sf_crypto_cipher_op_mode_t.
p_key	in	The key for the subsequent cipher operation.
p_rsa_params	in	The parameters specific to the RSA algorithm used for the Cipher operation.

Table 306:Return values

Name	Description
SSP_SUCCESS	The validation of the init parameters for the RSA cipher operation was successful.
SSP_ERR_INVALID_ARGUMENT	At least one of the input parameters are invalid.

6.14.89.2 Function steps

- Validate padding scheme
- Decryption is only supported with the private key.
- Encryption is only supported with the public key.
- Validate key length and buffer

6.14.90 sf_crypto_cipher_initialize_rsa_instance

```
ssp_err_t sf_crypto_cipher_initialize_rsa_instance ( sf_crypto_cipher_instance_ctrl_t  * p_ctrl ,  sf_crypto_cipher_cfg_t  const *const p_cfg )
```

6.14.90.1 Brief description

Allocates memory for the instance and opens the underlying HAL instance.

6.14.90.2 Detailed description

Table 307:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to Crypto Cipher Framework instance control block structure.
p_cfg	in	Pointer to the configuration structure for Cipher module .

Table 308:Return values

Name	Description
SSP_SUCCESS	The module instantiated successfully.
SSP_ERR_TIMEOUT	Was unable to allocate the memory within the specified time to wait.
SP_ERR_OUT_OF_MEMORY	Requested size is zero or larger than the pool.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.
Others	Crypto HAL module returned an error or resource lock/unlock was failed. See Common Error Codes or HAL driver for other possible return codes or causes. This function calls: sf_crypto_cipher_instance_memory_allocate sf_crypto_cipher_hal_open

6.14.91 sf_crypto_cipher_deinitialize_rsa_instance

```
ssp_err_t sf_crypto_cipher_deinitialize_rsa_instance ( sf_crypto_cipher_instance_ctrl_t  * p_ctrl )
```

6.14.91.1 Brief description

closes the underlying HAL instance and releases the memory for the instance.

6.14.91.2 Detailed description

Table 309:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to Crypto Cipher Framework instance control block structure.

Table 310:Return values

Name	Description
SSP_SUCCESS	The module instantiated successfully.
SSP_ERR_TIMEOUT	Was unable to allocate the memory within the specified time to wait.
SP_ERR_OUT_OF_MEMORY	Requested size is zero or larger than the pool.
SSP_ERR_INTERNAL	RTOS service returned a unexpected error.
Others	Crypto HAL module returned an error or resource lock/unlock was failed. See Common Error Codes or HAL driver for other possible return codes or causes. This function calls: sf_crypto_cipher_instance_memory_allocate sf_crypto_cipher_hal_open

6.14.91.3 Function steps

- First close the Framework TRNG instance.
- Then close the HAL RSA instance.

6.14.92 SF_CRYPTO_CIPHER_Open

```
ssp_err_t SF_CRYPTO_CIPHER_Open ( sf_crypto_cipher_ctrl_t  *const p_api_ctrl ,
 sf_crypto_cipher_cfg_t  const *const p_cfg )
```

6.14.92.1 Brief description

SSP Crypto Cipher Framework Open operation.

6.14.92.2 Detailed description

The SF_CRYTO_CIPHER_Open function: Allocates memory required for the cipher operations based on the configuration parameters. Acquires lock for the shared crypto resources. Gets the interface to the HAL driver based on the config parameters. Calls the .open function of the HAL API. On successful open, the module status is updated as such. The shared resources are unlocked before exit.

Table 311:Return values

Name	Description
SSP_SUCCESS	The module was successfully opened.
SSP_ERR_ASSERTION	One or more input parameters maybe NULL.
SSP_ERR_INVALID_ARGUMENT	An invalid argument is used.
SSP_ERR_CRYPTO_COMMON_NOT_OPENED	Crypto Framework Common Module has yet been opened.
SSP_ERR_ALREADY_OPEN	The module has been already opened.
SSP_ERR_UNSUPPORTED	The module does not support the key type specified by user.
Others	Crypto HAL module returned an error or resource lock/unlock failed.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use a mutex or to create an internal thread. See Common Error Codes for other possible return codes.

6.14.92.3 Function steps

- Get a Crypto common control block and the interface.
- Get the pointers to TRNG control and API structures from the instance pointed to by the config structure.
- Check if the Crypto Framework has been opened. If not yet opened, return an error.
- Check if the Cipher module has been already opened. If opened, return an error.
- Validate the cfg parameters
- Fill the control block with the validated cfg parameters
- Determine and fill the algorithm type in the control block.

- Note: sf_crypto.lock() is not called because the HAL driver does not access HW during open. The HAL interfaceGet API is also called which does not access HW either. If HW will be accessed through the HAL driver, lock should be acquired.
- Allocate memory resources used by this framework instance and open HAL driver

6.14.93 SF_CRYPTO_CIPHER_Close

```
ssp_err_t SF_CRYPTO_CIPHER_Close ( sf_crypto_cipher_ctrl_t  *const p_api_ctrl )
```

6.14.93.1 Brief description

SSP Crypto Cipher Framework Close operation.

6.14.93.2 Detailed description

End of function SF_CRYPTO_CIPHER_Open

Table 312:Return values

Name	Description
SSP_SUCCESS	The module was successfully closed.
SSP_ERR_ASSERTION	One or more input parameters maybe NULL.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.
Others	Crypto HAL module returned an error or resource lock/unlock was failed. See Common Error Codes for other possible return codes.

6.14.93.3 Function steps

- Get a Crypto common control block and the interface.
- Check if the Crypto Framework Cipher module has been opened. If not yet opened, return an error.
- Note: sf_crypto.lock() is not called because the HAL driver does not access HW during close. If HW will be accessed through the HAL driver, lock should be acquired.
- Free the memory allocated for this instance and close the HAL driver.

6.14.94 SF_CRYPTO_CIPHER_VersionGet

```
ssp_err_t SF_CRYPTO_CIPHER_VersionGet ( ssp_version_t  *const p_version )
```

6.14.94.1 Brief description

Gets driver version based on compile time macros.

6.14.94.2 Detailed description

End of function SF_CRYPTO_CIPHER_Close

Table 313:Return values

Name	Description
SSP_SUCCESS	The version is returned successfully.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

6.14.95 SF_CRYPTO_CIPHER_CipherInit

```
ssp_err_t SF_CRYPTO_CIPHER_CipherInit ( sf_crypto_cipher_ctrl_t
*const p_api_ctrl ,  sf_crypto_cipher_op_mode_t  cipher_operation_mode ,
 sf_crypto_data_handle_t  const *const p_key ,
 sf_crypto_cipher_algorithm_init_params_t  *const p_algorithm_specific_params )
```

6.14.95.1 Brief description

SSP Crypto Framework Cipher Init operation. The input parameters initialize the cipher operation. Some of the parameters are algorithm specific. The input parameters are validated and then copied into the context buffer. Refer to sf_crypto_cipher_aes_init_params_t or sf_crypto_cipher_rsa_init_params_t.

6.14.95.2 Detailed description

End of function SF_CRYPTO_KEY_VersionGet

Table 314:Return values

Name	Description
SSP_SUCCESS	The module is initialized for cipher operation successfully.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.
SSP_ERR_INVALID_ARGUMENT	One of the input parameters is invalid.

Table 314:Return values (Continued)

Name	Description
SSP_ERR_UNSUPPORTED	The module does not support the key type specified by user.
Others	See Common Error Codes for other possible return codes.

6.14.95.3 Function steps

- Validate cipher operation mode
- Check if the module can transition to the initialized state.
- Note: sf_crypto.lock() is not called because the HAL driver does not access HW during init. If HW will be accessed through the HAL driver, lock should be acquired.
- Do algorithm specific init
- Mark the module status as 'Initialized'.

6.14.96 SF_CRYPTO_CIPHER_CipherUpdate

```
ssp_err_t SF_CRYPTO_CIPHER_CipherUpdate ( sf_crypto_cipher_ctrl_t
*const p_api_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in ,
 sf_crypto_data_handle_t  *const p_data_out )
```

6.14.96.1 Brief description

SSP Crypto Cipher Framework - Encrypts / Decrypts the input data and writes the cipher-text or plain-text to the output buffer. Can be called multiple times for chunks of data.

6.14.96.2 Detailed description

End of function SF_CRYPTO_CIPHER_CipherInit

Table 315:Return values

Name	Description
SSP_SUCCESS	The function updated a cipher operation successfully.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.

Table 315:Return values (Continued)

Name	Description
SSP_ERR_UNSUPPORTED	Key types / algorithms are not supported for the MCU part.
SSP_ERR_INVALID_CALL	Function call was made if the module state had not yet transitioned to SF_CRYPTO_CIPHER_STATE_INITIALIZED.
Others	Crypto HAL module returned an error or resource lock/unlock was failed. See Common Error Codes for other possible return codes.

NOTE:

NOTE: 1. This is a blocking call.

NOTE: 2. The input and output buffers have to be WORD aligned.

NOTE: 3. On encryption and decryption operations, block alignment considerations may require that the number of

NOTE: bytes written into output buffer be larger or smaller than input data length or even 0.

NOTE: 4. The output data area must not partially overlap the input data area such that the input data is modified before

NOTE: it is used else incorrect output may result.

NOTE: 5. If the length of input data is 0 this method does nothing.

NOTE: 6. For all Cipher operations cipherFinal() must be called to finalize the operation.

NOTE: 7. For RSA Operations, no data is output unless cipherFinal() is called.

6.14.96.3 Function steps

- If input data length is 0 - do nothing return success.
- The status will be set to 'updated' only when data is processed.
- Get a Crypto Framework common control block and the interface.
- Check if the Crypto Cipher Framework is in the correct state to transition to do the update operation. If module is not opened or initialized, return an error.
- Acquire the lock from the common module to access the Crypto HAL driver.
- If update processing has succeeded return the error from unlock.

6.14.97 SF_CRYPTO_CIPHER_CipherAadUpdate

```
ssp_err_t SF_CRYPTO_CIPHER_CipherAadUpdate ( sf_crypto_cipher_ctrl_t
*const p_api_ctrl ,  sf_crypto_data_handle_t  const *const p_aad )
```

6.14.97.1 Brief description

Updates Additional Authenticated Data. This is applicable only to AES GCM. Can be called multiple times for chunks of data.

6.14.97.2 Detailed description

End of function SF_CRYPTO_CIPHER_CipherUpdate

Table 316:Return values

Name	Description
SSP_SUCCESS	The function updated the AAD successfully.
SSP_ERR_ASSERTION	NULL is passed through an argument.

Table 316:Return values (Continued)

Name	Description
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.
SSP_ERR_UNSUPPORTED	Key types / algorithms are not supported for the MCU part.
SSP_ERR_INVALID_CALL	Function call was made if the module state had not yet transitioned to SF_CRYPTO_CIPHER_STATE_INITIALIZED.
Others	Crypto HAL module returned an error or resource lock/unlock was failed. See Common Error Codes for other possible return codes.

NOTE:

NOTE: 1. This is a blocking call.

NOTE: 2. This function has to be called before any call to cipherUpdate / cipherFinal is made.

NOTE: 3. The data buffer must be WORD aligned.

6.14.97.3 Function steps

- do nothing - similar to the cipherUpdate API
- Get a Crypto Framework common control block and the interface.
- Check if the Crypto Cipher Framework is in the correct state to execute this operation. If module is not opened or initialized, return an error.
- Acquire the lock from the common module to access the Crypto HAL driver.
- AAD is optional so no need to track if it is updated.
- Unlock the module.
- if update processing has succeeded return the error from unlock.

6.14.98 SF_CRYPTO_CIPHER_CipherFinal

```
ssp_err_t SF_CRYPTO_CIPHER_CipherFinal ( sf_crypto_cipher_ctrl_t
*const p_api_ctrl ,  sf_crypto_data_handle_t  const *const p_data_in ,
 sf_crypto_data_handle_t  *const p_data_out )
```

6.14.98.1 Brief description

SSP Crypto Cipher Framework - Generates encrypted / decrypted output from all/last input data. This function processes any remaining input data buffered by one or more calls to the cipherUpdate()API as well as input data supplied in the input parameter. This function must be invoked to complete a cipher operation.

6.14.98.2 Detailed description

End of function SF_CRYPTO_CIPHER_CipherAadUpdate

Table 317:Return values

Name	Description
SSP_SUCCESS	The module updated a message Digest successfully.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.
SSP_ERR_UNSUPPORTED	Key types / algorithms are not supported for the MCU part.
SSP_ERR_INVALID_CALL	Function call was made if the module state had not yet transitioned to SF_CRYPTO_CIPHER_STATE_INITIALIZED or SF_CRYPTO_CIPHER_STATE_UPDATED.
Others	Crypto HAL module returned an error or resource lock/unlock was failed. See Common Error Codes for other possible return codes.

NOTE:

NOTE: 1. This is a blocking call.

NOTE: 2. On encryption and decryption operations, block alignment considerations may require that the number of

NOTE: bytes written into output buffer be larger or smaller than input data length or even 0.

NOTE: 3. The output data area must not partially overlap the input data area such that the input data is modified before

NOTE: it is used else incorrect output may result.

NOTE: 4. Except for GCM, if the length of input data is 0 and no input was provided through update, an error is returned.

NOTE: 5. On decryption operations the padding bytes are not written to p_data_out.p_data.

NOTE: 6. The input and output buffers must be WORD aligned.

6.14.98.3 Function steps

- Get a Crypto Framework common control block and the interface.
- Check if the Crypto Cipher Framework is in the correct state to execute this operation. If module is not opened or initialized, return an error.
- Acquire the lock from the common module to access to Crypto HAL driver.
- Mark the module status as 'Finalized'.
- Unlock the module.
- if final processing has succeeded return the error from unlock.

6.14.99 API Data

6.14.99.1 sf_crypto_cipher_mode_t

`sf_crypto_cipher_mode_t`

Detailed description

AES modes for the SSP Crypto Cipher Framework

Enumerated values

Name	Description
SF_CRYPTO_CIPHER_MODE_ECB	Electronic Code Book chaining mode, default for RSA.
SF_CRYPTO_CIPHER_MODE_CBC	Cipher Block Chaining.
SF_CRYPTO_CIPHER_MODE_CTR	Counter Mode.
SF_CRYPTO_CIPHER_MODE_XTS	XEX-based tweaked-codebook mode with ciphertext stealing.
SF_CRYPTO_CIPHER_MODE_GCM	Galois Counter Mode.

6.14.99.2 sf_crypto_cipher_op_mode_t

`sf_crypto_cipher_op_mode_t`

Detailed description

Operating mode for Cipher APIs

Enumerated values

Name	Description
SF_CRYPTO_CIPHER_OP_MODE_ENCRYPT	The operating mode is set to encryption.
SF_CRYPTO_CIPHER_OP_MODE_DECRYPT	The operating mode is set to decryption.

6.14.99.3 sf_crypto_cipher_padding_scheme_t

`sf_crypto_cipher_padding_scheme_t`

Detailed description

Padding mode to be used for for Cipher operation - encrypting/ decrypting input data

Enumerated values

Name	Description
SF_CRYPTO_CIPHER_PADDING_SCHEME_NO_PADDING	No padding scheme.
SF_CRYPTO_CIPHER_PADDING_SCHEME_PKCS7	PKCS#7 padding scheme - applicable only for AES operations.
SF_CRYPTO_CIPHER_PADDING_SCHEME_PKCS1_1_5	PKCS#1 v1.5 padding scheme - applicable only for RSA operations.

6.14.99.4 sf_crypto_cipher_algorithm_init_params_t

```
typedef void sf_crypto_cipher_algorithm_init_params_t
```

Detailed description

Algorithm specific parameters. Allocate an algorithm specific block to pass into the cipherInit API call. Implemented as

- sf_crypto_cipher_aes_init_params_t for AES
- sf_crypto_cipher_rsa_init_params_t for RSA.

6.14.99.5 sf_crypto_cipher_ctrl_t

```
typedef void sf_crypto_cipher_ctrl_t
```

Detailed description

SSP Crypto Cipher framework control block. Allocate an instance specific control block to pass into the SSP Crypto framework Cipher API calls. Implemented as

- sf_crypto_cipher_instance_ctrl_t

6.14.100 Extensions

6.14.100.1 sf_crypto_cipher_aes_init_params_t

sf_crypto_cipher_aes_init_params_t

Detailed description

AES Algorithm specific parameters for cipher operations

Variables

- sf_crypto_cipher_padding_scheme_t padding_scheme
- sf_crypto_data_handle_t * p_iv

  pointer to IV for the AES operation.

- sf_crypto_data_handle_t * p_auth_tag

  Pointer to the GCM Authentication Tag. Only tag length of SF_CRYPTO_CIPHER_AES_GCM_TAG_LENGTH_16_BYTES is supported.

6.14.100.2 sf_crypto_cipher_rsa_init_params_t

sf_crypto_cipher_rsa_init_params_t

Detailed description

RSA Algorithm specific parameters for cipher operations

Variables

- sf_crypto_cipher_padding_scheme_t padding_scheme

6.14.100.3 sf_crypto_cipher_cfg_t

sf_crypto_cipher_cfg_t

Detailed description

Configuration structure for the SSP Crypto Cipher framework Cipher chaining mode for RSA operations is not applicable and can be set to ECB

Variables

- sf_crypto_key_type_t key_type

  Key type for cipher operation.

- sf_crypto_key_size_t key_size

  Key size for cipher operation.

- sf_crypto_cipher_mode_t cipher_chaining_mode

  Chaining mode specified for the cipher operation.

- sf_crypto_instance_t * p_lower_lvl_crypto_common

  Pointer to a Crypto Framework common instance.

- sf_crypto_trng_instance_t * p_lower_lvl_crypto_trng

  Pointer to a Crypto Framework TRNG instance.

- void const * p_extend

  Future extension for hardware specific settings.

6.14.100.4 sf_crypto_cipher_api_t

sf_crypto_cipher_api_t

Detailed description

Shared Interface definition for the SSP Crypto Cipher framework module

6.14.100.5 sf_crypto_cipher_instance_t

sf_crypto_cipher_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_crypto_cipher_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_crypto_cipher_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_crypto_cipher_api_t const * p_api

  Pointer to the API structure for this instance.

6.15 SSP Crypto HASH Framework Interface

Interface definition for Synergy Crypto HASH Framework module.

6.15.1 Summary

This is the Interface of SF_CRYPTO_HASH Framework module.
SF_CRYPTO_HASH Interface description:

6.15.2 Typedefs

- sf_crypto_hash_t
- sf_crypto_hash_ctrl_t

6.15.3 Defines

- #define SF_CRYPTO_HASH_API_VERSION_MAJOR

  Initial value: `(1U)`

  The API version of SSP Crypto HASH Framework
- #define SF_CRYPTO_HASH_API_VERSION_MINOR

  Initial value: `(2U)`

- #define SF_CRYPTO_HASH_MESSAGE_DIGEST_SIZE_MD5

  Initial value: `(16U)`

  Message Digest size for SHA1. Message Digest size for each HASH algorithm in bytes
- #define SF_CRYPTO_HASH_MESSAGE_DIGEST_SIZE_SHA1

  Initial value: `(20U)`

  Message Digest size for SHA1.
- #define SF_CRYPTO_HASH_MESSAGE_DIGEST_SIZE_SHA224

  Initial value: `(28U)`

  Message Digest size for SHA224.
- #define SF_CRYPTO_HASH_MESSAGE_DIGEST_SIZE_SHA256

  Initial value: `(32U)`

  Message Digest size for SHA256.

6.15.4 API Data

6.15.4.1 sf_crypto_hash_state_t

`sf_crypto_hash_state_t`

Detailed description

State codes for the SSP SSP Crypto HASH Framework

Enumerated values

Name	Description
SF_CRYPTO_HASH_CLOSED	The module is closed.
SF_CRYPTO_HASH_OPENED	The module is opened. The initial message digest is not yet generated.
SF_CRYPTO_HASH_DIGEST_INITIALIZED	Message digest is initialized.
SF_CRYPTO_HASH_DIGEST_UPDATED	Message digest is updated.

6.15.4.2 sf_crypto_hash_type_t

`sf_crypto_hash_type_t`

Detailed description

HASH algorithm types for the SSP SSP Crypto HASH Framework

Enumerated values

Name	Description
SF_CRYPTO_HASH_ALGORITHM_MD5	MD5 algorithm type.
SF_CRYPTO_HASH_ALGORITHM_SHA1	SHA-1 algorithm type.
SF_CRYPTO_HASH_ALGORITHM_SHA224	SHA-224 algorithm type.
SF_CRYPTO_HASH_ALGORITHM_SHA256	SHA-256 algorithm type.

6.15.4.3 sf_crypto_hash_t

```
typedef sf_crypto_data_handle_t sf_crypto_hash_t
```

6.15.4.4 sf_crypto_hash_ctrl_t

```
typedef void sf_crypto_hash_ctrl_t
```

Detailed description

SSP Crypto framework control block. Allocate an instance specific control block to pass into the SSP Crypto framework API calls. Implemented as

- sf_crypto_instance_ctrl_t

6.15.5 Extensions

6.15.5.1 sf_crypto_hash_context_t

sf_crypto_hash_context_t

Detailed description

HASH internal context structure for a message digest

Variables

- uint8_t * p_message_digest

  Intermediate digest stored buffer - WORD aligned.

- uint8_t * p_message_digest_org

  Originally allocated buffer - may not be WORD aligned.

- uint8_t * p_message_buffer

  Intermediate message data stored buffer - - WORD aligned.
- uint8_t * p_message_buffer_org

  IOriginally allocated buffer - may not be WORD aligned.
- uint64_t message_bytes

  Number of bytes from user data processed.
- uint32_t message_bytes_buffered

  Number of bytes buffered in the message data stored buffer.

6.15.5.2 sf_crypto_hash_callback_args_t

sf_crypto_hash_callback_args_t

Detailed description

Callback arguments for the SSP Crypto HASH framework

Variables

- ssp_err_t error

  Error code if SF_CRYPTO_EVENT_ERROR.

6.15.5.3 sf_crypto_hash_cfg_t

sf_crypto_hash_cfg_t

Detailed description

Configuration structure for the SSP SSP Crypto HASH framework

Variables

- sf_crypto_hash_type_t hash_type

  HASH algorithm type.
- sf_crypto_instance_t * p_lower_lvl_crypto_common

  Pointer to a Crypto Framework common instance.
- hash_instance_t * p_lower_lvl_instance

  pointer to HASH lower-level module instance
- void * p_extend

  Pointer to an optional configuration for Crypto HAL module.

6.15.5.4 sf_crypto_hash_api_t

sf_crypto_hash_api_t

Detailed description

Shared Interface definition for the SSP SSP Crypto framework

6.15.5.5 sf_crypto_hash_instance_t

sf_crypto_hash_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_crypto_hash_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_crypto_hash_cfg_t * p_cfg

  Pointer to the configuration structure for this instance.
- sf_crypto_hash_api_t const * p_api

  Pointer to the API structure for this instance.

6.16 SSP Crypto TRNG Framework Interface

Interface definition for Synergy Crypto TRNG Framework module.
Interface declaration for Synergy Crypto TRNG Framework module.

6.16.1 Summary

This is the Interface of SF_CRYPTO_TRNG Framework module.
SF_CRYPTO_TRNG Interface description: Crypto Framework

6.16.2 Summary

This is the Interface declarations of SF_CRYPTO_TRNG Framework module.
SF_CRYPTO_TRNG Interface declaration: Crypto Framework

6.16.3 Functions

- SF_CRYPTO_TRNG_Open
- SF_CRYPTO_TRNG_Close
- SF_CRYPTO_TRNG_RandomNumberGenerate
- SF_CRYPTO_TRNG_VersionGet

6.16.4 Variables

- sf_crypto_trng_version
- g_sf_crypto_trng_api

6.16.5 Typedefs

- sf_crypto_trng_ctrl_t

6.16.6 Defines

- #define SF_CRYPTO_TRNG_API_VERSION_MAJOR

  Initial value: `(1U)`

  The API version of SSP Crypto Framework
- #define SF_CRYPTO_TRNG_API_VERSION_MINOR

  Initial value: `(0U)`
- #define WORDS_IN_SINGLE_HW_PROC_CALL

  Initial value: `(4)`
- #define BYTES_IN_WORD

  Initial value: `(4)`
- #define BYTES_IN_SINGLE_HW_PROC_CALL

  Initial value: `((BYTES_IN_WORD) * (WORDS_IN_SINGLE_HW_PROC_CALL))`
- #define EXTRA_BUFFER_SIZE_WORDS

  Initial value: `((BYTES_IN_SINGLE_HW_PROC_CALL) / (BYTES_IN_WORD))`

6.16.7 SF_CRYPTO_TRNG_Open

```
ssp_err_t SF_CRYPTO_TRNG_Open ( sf_crypto_trng_ctrl_t  *const p_api_ctrl ,
 sf_crypto_trng_cfg_t  const *const p_cfg )
```

6.16.7.1 Brief description

SSP Crypto TRNG Framework Open operation.

6.16.7.2 Detailed description

Table 318:Return values

Name	Description
SSP_SUCCESS	The module was successfully opened
SSP_ERR_ASSERTION	One or more input parameters are NULL.
SSP_ERR_CRYPTO_COMMON_NOT_OPENED	Crypto framework common module has not been opened before calling this API.
SSP_ERR_ALREADY_OPEN	The module has already been opened.
Others	Crypto HAL module returned an error or resource lock/unlock failed. See Common Error Codes for other possible return codes.

6.16.7.3 Function steps

- Note: sf_crypto.lock() is not called because the HAL driver does not access HW during open. In the future, if HW will be accessed through the HAL driver; lock should be acquired.

6.16.8 SF_CRYPTO_TRNG_Close

```
ssp_err_t SF_CRYPTO_TRNG_Close ( sf_crypto_trng_ctrl_t  *const p_api_ctrl )
```

6.16.8.1 Brief description

SSP Crypto TRNG Framework Close operation.

6.16.8.2 Detailed description

Table 319:Return values

Name	Description
SSP_SUCCESS	The module was successfully closed

Table 319:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	One or more input parameters are NULL.
SSP_ERR_CRYPTO_COMMON_NOT_OPENED	Crypto Common module has not been opened.
SSP_ERR_NOT_OPEN	The module has not been opened before calling this API.
Others	Crypto HAL module returned an error or resource lock/unlock failed. See Common Error Codes for other possible return codes.

6.16.8.3 Function steps

- Note: sf_crypto.lock() is not called because the HAL driver does not access HW during close. In the future, if HW will be accessed through the HAL driver; lock should be acquired.

6.16.9 SF_CRYPTO_TRNG_RandomNumberGenerate

```
ssp_err_t SF_CRYPTO_TRNG_RandomNumberGenerate ( sf_crypto_trng_ctrl_t
*const p_api_ctrl ,  sf_crypto_data_handle_t  *const p_random_number_buff )
```

6.16.9.1 Brief description

SSP Crypto TRNG Framework True Random Generation operation.

6.16.9.2 Detailed description

Table 320:Return values

Name	Description
SSP_SUCCESS	The module was successfully closed.
SSP_ERR_NOT_OPEN	The module has not been opened before calling this API.
SSP_ERR_ASSERTION	One or more input parameters are NULL.
Others	Crypto HAL module returned an error or resource lock/unlock failed. See Common Error Codes for other possible return codes.

6.16.10 SF_CRYPTO_TRNG_VersionGet

```
ssp_err_t SF_CRYPTO_TRNG_VersionGet ( ssp_version_t  *const p_version )
```

6.16.10.1 Brief description

Sets TRNG Framework Code and API version based on compile time macros.

6.16.10.2 Detailed description

Table 321:Return values

Name	Description
SSP_SUCCESS	Successful close
SSP_ERR_ASSERTION	The parameter p_version is NULL.

6.16.11 g_sf_crypto_trng_api

```
sf_crypto_trng_api_t::g_sf_crypto_trng_api
```

6.16.11.1 Detailed description

Filled in Interface API structure for this Instance.

6.16.11.2 Initialized as

```
g_sf_crypto_trng_api=
{
    .open                       = SF_CRYPTO_TRNG_Open,
    .close                      = SF_CRYPTO_TRNG_Close,
    .randomNumberGenerate       = SF_CRYPTO_TRNG_RandomNumberGenerate,
    .versionGet                 = SF_CRYPTO_TRNG_VersionGet
}
```

6.16.12 API Data

6.16.12.1 sf_crypto_trng_state_t

```
sf_crypto_trng_state_t
```

Detailed description

State codes for the SSP Crypto TRNG framework

Enumerated values

Name	Description
SF_CRYPTO_TRNG_CLOSED	Crypto TRNG Framework Module module is closed.
SF_CRYPTO_TRNG_OPENED	Crypto TRNG Framework Module module is opened.

6.16.12.2 sf_crypto_trng_ctrl_t

```
typedef void sf_crypto_trng_ctrl_t
```

Detailed description

SSP Crypto TRNG framework control block. Implemented as

- sf_crypto_trng_ctrl_t

6.16.13 Extensions

6.16.13.1 sf_crypto_trng_cfg_t

sf_crypto_trng_cfg_t

Detailed description

Configuration structure for the SSP Crypto TRNG framework

Variables

- sf_crypto_instance_t * p_lower_lvl_common

  Pointer to a Crypto Framework common instance.
- trng_instance_t * p_lower_lvl_instance

  Pointer to Crypto TRNG HAL instance.
- void * p_extend

  Pointer to an optional configuration for HW specific settings.

6.16.13.2 sf_crypto_trng_api_t

sf_crypto_trng_api_t

Detailed description

Shared Interface definition for the SSP Crypto framework

6.16.13.3 sf_crypto_trng_instance_t

sf_crypto_trng_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_crypto_trng_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- sf_crypto_trng_cfg_t * p_cfg

  Pointer to the configuration structure for this instance.

- sf_crypto_trng_api_t const * p_api

  Pointer to the API structure for this instance.

6.17 SSP Crypto Key Framework Interface

Interface declaration for Synergy Crypto Key Framework module for AES.
Interface declaration for Synergy Crypto Key Framework module.
Interface declaration for Synergy Crypto Key Framework module for RSA.
Interface declaration for Synergy Crypto Key Framework module for ECC.

6.18 SSP Crypto Key Installation Framework Interface

Interface definition for Synergy Crypto Key Installation Framework module.

6.18.1 Functions

- SF_CRYPTO_KEY_INSTALLATION_Open
- SF_CRYPTO_KEY_INSTALLATION_Close
- SF_CRYPTO_KEY_INSTALLATION_KeyInstall
- SF_CRYPTO_KEY_INSTALLATION_VersionGet

6.18.2 Defines

- #define SF_CRYPTO_KEY_INSTALLATION_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  The API version of SSP Crypto Key Installation Framework

- #define SF_CRYPTO_KEY_INSTALLATION_CODE_VERSION_MINOR

  Initial value: `(0U)`

6.18.3 SF_CRYPTO_KEY_INSTALLATION_Open

```
ssp_err_t SF_CRYPTO_KEY_INSTALLATION_Open ( sf_crypto_key_installation_ctrl_t
*const p_api_ctrl , sf_crypto_key_installation_cfg_t const *const p_cfg )
```

6.18.3.1 Brief description

SSP Crypto Key Installation Framework Open operation.

6.18.3.2 Detailed description

Table 322:Return values

Name	Description
SSP_SUCCESS	The module was successfully opened.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_CRYPTO_COMMON_NOT_OPENED	Crypto Framework Common Module has yet been opened.
SSP_ERR_ALREADY_OPEN	The module has been already opened.
SSP_ERR_UNSUPPORTED	The module does not support the key type specified by user.
Others	Crypto HAL module returned an error. See Common Error Codes for other possible return codes.

6.18.3.3 Function steps

- Note: sf_crypto.lock() is not called because the HAL driver does not access HW during open. The HAL interfaceGet API is also called which does not access HW either. In the future, if HW will be accessed through the HAL driver; lock should be acquired.

6.18.4 SF_CRYPTO_KEY_INSTALLATION_Close

```
ssp_err_t SF_CRYPTO_KEY_INSTALLATION_Close ( sf_crypto_key_installation_ctrl_t
*const p_api_ctrl )
```

6.18.4.1 Brief description

SSP Crypto Key Installation Framework Close operation.

6.18.4.2 Detailed description

Table 323:Return values

Name	Description
SSP_SUCCESS	The module was successfully closed.
SSP_ERR_ASSERTION	NULL is passed through the argument.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.
Others	Crypto HAL module returned an error. See Common Error Codes for other possible return codes.

6.18.4.3 Function steps

- Note: sf_crypto.lock() is not called because the HAL driver does not access HW during close. In the future, if HW will be accessed through the HAL driver; lock should be acquired.

6.18.5 SF_CRYPTO_KEY_INSTALLATION_KeyInstall

```
ssp_err_t SF_CRYPTO_KEY_INSTALLATION_KeyInstall (  sf_crypto_key_installation_ct
rl_t *const p_api_ctrl ,  sf_crypto_data_handle_t  *const p_user_key_input ,
 sf_crypto_data_handle_t  *const p_install_key_input ,  sf_crypto_data_handle_t
* p_key_data_out )
```

6.18.5.1 Brief description

SSP Crypto Framework Key Installation operation.

6.18.5.2 Detailed description

Table 324:Return values

Name	Description
SSP_SUCCESS	The module created a key successfully.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.
SSP_ERR_UNSUPPORTED	The module does not support the key type specified by user.

Table 324:Return values (Continued)

Name	Description
Others	Crypto HAL module returned an error or resource lock/unlock was failed. See Common Error Codes for other possible return codes.

6.18.6 SF_CRYPTO_KEY_INSTALLATION_VersionGet

```
ssp_err_t SF_CRYPTO_KEY_INSTALLATION_VersionGet ( ssp_version_t
*const p_version )
```

6.18.6.1 Brief description

Gets driver version based on compile time macros.

6.18.6.2 Detailed description

Table 325:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

6.18.7 Extensions

6.18.7.1 sf_crypto_key_installation_instance_ctrl_t

sf_crypto_key_installation_instance_ctrl_t

Detailed description

SSP Crypto Key Installation Framework instance control block

Variables

- sf_crypto_key_installation_state_t status

  Module status.
- sf_crypto_key_type_t key_type

  Type of key to be installed.
- sf_crypto_key_size_t key_size

  Size of key to be installed.

- sf_crypto_instance_ctrl_t * p_lower_lvl_common_ctrl

  Pointer to the Crypto Framework Common instance.
- sf_crypto_api_t * p_lower_lvl_common_api

  Pointer to the Crypto Framework Common instance.
- void * p_lower_lvl_instance

  Pointer to HAL KeyInstall Crypto module instance structure.

6.19 SSP Crypto Signature Framework Interface

Interface definition for Synergy Crypto Signature Framework module.
Common declaration for Synergy Crypto Signature Framework module.
Interface declaration for Synergy Crypto Signature Framework module.
Interface declaration for Synergy Crypto Signature Framework module - RSA specific.
SF_CRYPTO_SIGNATURE Interface description: Crypto Framework

6.19.1 Functions

- sf_crypto_signature_key_size_config_rsa
- sf_crypto_signature_open_rsa
- sf_crypto_signature_close_rsa
- sf_crypto_signature_context_init_rsa
- sf_crypto_signature_sign_update_rsa
- sf_crypto_signature_verify_update_rsa
- sf_crypto_signature_sign_final_rsa
- sf_crypto_signature_verify_final_rsa
- SF_CRYPTO_SIGNATURE_Open
- SF_CRYPTO_SIGNATURE_Close
- SF_CRYPTO_SIGNATURE_ContextInit
- SF_CRYPTO_SIGNATURE_SignUpdate
- SF_CRYPTO_SIGNATURE_VerifyUpdate
- SF_CRYPTO_SIGNATURE_SignFinal
- SF_CRYPTO_SIGNATURE_VerifyFinal
- SF_CRYPTO_SIGNATURE_VersionGet
- sf_crypto_signature_validate_sign_operation_state_transition
- sf_crypto_signature_validate_verify_operation_state_transition

6.19.2 Defines

- #define SF_CRYPTO_SIGNATURE_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  The API version of SSP Crypto Signature Framework
- #define SF_CRYPTO_SIGNATURE_CODE_VERSION_MINOR

  Initial value: `(0U)`

6.19.3 sf_crypto_signature_key_size_config_rsa

```
ssp_err_t sf_crypto_signature_key_size_config_rsa ( sf_crypto_signature_cfg_t
const *const p_cfg )
```

6.19.3.1 Brief description

Function for Crypto Signature Framework to check configuration params - Key size. This function is called by sf_crypto_signature_validate_config().

6.19.3.2 Detailed description

Table 326:Parameters

Name	Direction	Description
p_cfg	in	Pointer to sf_crypto_signature_cfg_t configuration structure.

Table 327:Return values

Name	Description
SSP_SUCCESS	Valid RSA Key size.
SSP_ERR_INVALID_ARGUMENT	Invalid RSA Key size.

6.19.4 sf_crypto_signature_open_rsa

```
ssp_err_t sf_crypto_signature_open_rsa ( sf_crypto_signature_instance_ctrl_t
*const p_ctrl ,  sf_crypto_signature_cfg_t  const *const p_cfg )
```

6.19.4.1 Brief description

Function for Crypto Signature Framework to open the RSA HAL driver. This function is called by sf_crypto_signature_hal_open().

6.19.4.2 Detailed description

Table 328:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to Crypto Signature Framework instance control block structure.
p_cfg	in	Pointer to sf_crypto_signature_cfg_t configuration structure.

Table 329:Return values

Name	Description
SSP_SUCCESS	The module was successfully opened.
SSP_ERR_INTERNAL	Critical internal error.
Others	Crypto RSA module returned an error. See Common Error Codes for other possible return codes.

6.19.5 sf_crypto_signature_close_rsa

```
ssp_err_t sf_crypto_signature_close_rsa ( sf_crypto_signature_instance_ctrl_t
* p_ctrl )
```

6.19.5.1 Brief description

Function for Crypto Signature Framework to close the RSA HAL driver. This function is called by sf_crypto_signature_hal_close().

6.19.5.2 Detailed description

Table 330:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to Crypto Signature Framework instance control block structure.

Table 331:Return values

Name	Description
SSP_SUCCESS	The module was successfully closed.
SSP_ERR_INTERNAL	Critical internal error.
Others	Crypto RSA module returned an error. See Common Error Codes for other possible return codes.

6.19.6 sf_crypto_signature_context_init_rsa

```
ssp_err_t sf_crypto_signature_context_init_rsa ( sf_crypto_signature_instance_ctrl_t  *const p_ctrl ,   sf_crypto_signature_mode_t operation_mode ,
  sf_crypto_signature_algorithm_init_params_t
*const p_algorithm_specific_params ,  sf_crypto_data_handle_t  const
*const p_key )
```

6.19.6.1 Brief description

Function for Crypto Signature Framework to initialize the Signature framework module context. This function is called by sf_crypto_signature_hal_context_init().

6.19.6.2 Detailed description

Table 332:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to Crypto Signature Framework instance control block structure.

Table 332:Parameters (Continued)

Name	Direction	Description
operation_mode	in	Perform Sign Or Verify operation.
p_algorithm_specific_params	in	Algorithm specific params.
p_key	in	Private key if sign operation is to be performed. Public key if verify operation is to be performed.

Table 333:Return values

Name	Description
SSP_SUCCESS	The module context was successfully initialized.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_INVALID_CALL	Invalid call to this API.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_INVALID_OPERATION_MODE	Invalid operation mode requested.
Others	Crypto RSA module returned an error. Crypto Hash Framework module returned an error. See Common Error Codes for other possible return codes.

6.19.7 sf_crypto_signature_sign_update_rsa

```
ssp_err_t sf_crypto_signature_sign_update_rsa ( sf_crypto_signature_instance_ctrl_t  *const p_ctrl ,  sf_crypto_data_handle_t  const *const p_message )
```

6.19.7.1 Brief description

Function for Crypto Signature Framework to perform sign update operation. This function is called by sf_crypto_signature_hal_sign_update().

6.19.7.2 Detailed description

Table 334:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to Crypto Signature Framework instance control block structure.
p_message	in	Pointer to data handle containing update data and its length.

Table 335:Return values

Name	Description
SSP_SUCCESS	Sign update was performed successfully.
SSP_ERR_ASSERTION	Critical internal error.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_BUF_OVERFLOW	Update data exceeded the block size.
SSP_ERR_CRYPTO_INVALID_OPERATION_MODE	Invalid operation mode requested.
Others	Crypto RSA module returned an error. Crypto Hash Framework module returned an error. See Common Error Codes for other possible return codes.

6.19.8 sf_crypto_signature_verify_update_rsa

```
ssp_err_t sf_crypto_signature_verify_update_rsa ( sf_crypto_signature_instance_ctrl_t  *const p_ctrl ,  sf_crypto_data_handle_t  const *const p_message )
```

6.19.8.1 Brief description

Function for Crypto Signature Framework to perform verify update operation. This function is called by sf_crypto_signature_hal_verify_update().

6.19.8.2 Detailed description

Table 336:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to Crypto Signature Framework instance control block structure.
p_message	in	Pointer to data handle containing message in appropriate format and its length.

Table 337:Return values

Name	Description
SSP_SUCCESS	Verify update operation was performed successfully.
SSP_ERR_ASSERTION	Critical internal error.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_BUF_OVERFLOW	Update data exceeded the block size.
SSP_ERR_CRYPTO_INVALID_OPERATION_MODE	Invalid operation mode requested.
Others	Crypto RSA module returned an error. Crypto Hash Framework module returned an error. See Common Error Codes for other possible return codes.

6.19.9 sf_crypto_signature_sign_final_rsa

```
ssp_err_t sf_crypto_signature_sign_final_rsa ( sf_crypto_signature_instance_ctrl
_t  *const p_ctrl ,  sf_crypto_data_handle_t  const *const p_message ,
 sf_crypto_data_handle_t  *const p_dest )
```

6.19.9.1 Brief description

Function for Crypto Signature Framework to perform sign final operation. This function is called by sf_crypto_signature_hal_sign_final().

6.19.9.2 Detailed description

Table 338:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to Crypto Signature Framework instance control block structure.
p_message	in	Pointer to data handle containing last block of data and its length. If there is no more data to be passed this param can be set to NULL.
p_dest	inout	Pointer to data handle containing pointer to a buffer for storing signature. The data_length of this handle must be populated with the buffer length. Upon successful return this data_length will be updated with the number of bytes written to this buffer.

Table 339:Return values

Name	Description
SSP_SUCCESS	Sign Final operation was performed successfully.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_CRYPTO_INVALID_SIZE	Not enough space to store signature.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_BUF_OVERFLOW	Update data exceeded the block size.
Others	Crypto RSA module returned an error. Crypto Hash Framework module returned an error. See Common Error Codes for other possible return codes.

6.19.10 sf_crypto_signature_verify_final_rsa

```
ssp_err_t sf_crypto_signature_verify_final_rsa ( sf_crypto_signature_instance_ctrl_t  *const p_ctrl ,  sf_crypto_data_handle_t  const *const p_signature ,
 sf_crypto_data_handle_t  const *const p_message )
```

6.19.10.1 Brief description

Function for Crypto Signature Framework to perform verify final operation. This function is called by sf_crypto_signature_hal_verify_final().

6.19.10.2 Detailed description

Table 340:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to Crypto Signature Framework instance control block structure.
p_signature	in	Pointer to data handle containing signature and its length.
p_message	in	Pointer to data handle containing message in appropriate format and its length.

Table 341:Return values

Name	Description
SSP_SUCCESS	Verify Final operation was performed successfully.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_CRYPTO_INVALID_SIZE	Invalid signature length.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_BUF_OVERFLOW	Update data exceeded the block size.
Others	Crypto RSA module returned an error. Crypto Hash Framework module returned an error. See Common Error Codes for other possible return codes.

6.19.11 SF_CRYPTO_SIGNATURE_Open

```
ssp_err_t SF_CRYPTO_SIGNATURE_Open (  sf_crypto_signature_ctrl_t
*const p_api_ctrl ,  sf_crypto_signature_cfg_t  const *const p_cfg )
```

6.19.11.1 Brief description

SSP Crypto Signature Framework Open operation.

6.19.11.2 Detailed description

Table 342:Return values

Name	Description
SSP_SUCCESS	The module was successfully opened.
SSP_ERR_ASSERTION	NULL is passed through an argument.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_CRYPTO_COMMON_NOT_OPENED	Crypto Framework Common Module has yet been opened.
SSP_ERR_ALREADY_OPEN	The module has been already opened.
SSP_ERR_UNSUPPORTED	The module does not support the key type specified by user.
Others	Crypto HAL module returned an error or resource lock/unlock was failed. See Common Error Codes for other possible return codes.

6.19.11.3 Function steps

- Note: sf_crypto.lock() is not called because the HAL driver does not access HW during open. The HAL interfaceGet API is also called which does not access HW either. If HW will be accessed through the HAL driver, lock should be acquired.

6.19.12 SF_CRYPTO_SIGNATURE_Close

```
ssp_err_t SF_CRYPTO_SIGNATURE_Close (  sf_crypto_signature_ctrl_t
*const p_api_ctrl )
```

6.19.12.1 Brief description

SSP Crypto Signature Framework Close operation.

6.19.12.2 Detailed description

Table 343:Return values

Name	Description
SSP_SUCCESS	The module was successfully closed.
SSP_ERR_ASSERTION	NULL is passed through the argument.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_NOT_OPEN	The module has yet been opened. Call Open API first.
Others	Crypto HAL module returned an error or resource lock/unlock was failed. See Common Error Codes for other possible return codes.

6.19.12.3 Function steps

- Note: sf_crypto.lock() is not called because the HAL driver does not access HW during close. If HW will be accessed through the HAL driver, lock should be acquired.

6.19.13 SF_CRYPTO_SIGNATURE_ContextInit

```
ssp_err_t SF_CRYPTO_SIGNATURE_ContextInit (  sf_crypto_signature_ctrl_t
*const p_api_ctrl ,   sf_crypto_signature_mode_t operation_mode ,
  sf_crypto_signature_algorithm_init_params_t
*const p_algorithm_specific_params ,  sf_crypto_data_handle_t  const
*const p_key )
```

6.19.13.1 Brief description

SSP Crypto Signature Framework Context Initialization operation.

6.19.13.2 Detailed description

Table 344:Return values

Name	Description
SSP_SUCCESS	The module context was successfully initialized.
SSP_ERR_ASSERTION	NULL is passed through the argument.

Table 344:Return values (Continued)

Name	Description
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_INVALID_CALL	Invalid call to this API.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_INVALID_OPERATION_MODE	Invalid operation mode requested.
Others	Crypto HAL module returned an error or resource lock/unlock was failed. Crypto Hash Framework module returned an error. See Common Error Codes for other possible return codes.

6.19.13.3 Function steps

- Note: sf_crypto.lock() is not called because the HAL driver does not access HW during init. If HW will be accessed through the HAL driver, lock should be acquired.

6.19.14 SF_CRYPTO_SIGNATURE_SignUpdate

```
ssp_err_t SF_CRYPTO_SIGNATURE_SignUpdate (  sf_crypto_signature_ctrl_t
*const p_api_ctrl ,  sf_crypto_data_handle_t  const *const p_message )
```

6.19.14.1 Brief description

SSP Crypto Signature Framework Signature Update operation.

6.19.14.2 Detailed description

Table 345:Return values

Name	Description
SSP_SUCCESS	Sign update was performed successfully.
SSP_ERR_ASSERTION	NULL is passed through the argument.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_INVALID_CALL	Invalid call to this API.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.

Table 345:Return values (Continued)

Name	Description
SSP_ERR_CRYPTO_BUF_OVERFLOW	Update data exceeded the block size.
SSP_ERR_CRYPTO_INVALID_OPERATION_MODE	Invalid operation mode requested.
Others	Crypto HAL module returned an error or resource lock/unlock was failed. Crypto Hash Framework module returned an error. See Common Error Codes for other possible return codes.

6.19.15 SF_CRYPTO_SIGNATURE_VerifyUpdate

```
ssp_err_t SF_CRYPTO_SIGNATURE_VerifyUpdate (  sf_crypto_signature_ctrl_t
*const p_api_ctrl ,  sf_crypto_data_handle_t  const *const p_message )
```

6.19.15.1 Brief description

SSP Crypto Signature Framework Signature-Verification Update operation.

6.19.15.2 Detailed description

Table 346:Return values

Name	Description
SSP_SUCCESS	Verify update operation was performed successfully.
SSP_ERR_ASSERTION	NULL is passed through the argument.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_INVALID_CALL	Invalid call to this API.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_BUF_OVERFLOW	Update data exceeded the block size.
SSP_ERR_CRYPTO_INVALID_OPERATION_MODE	Invalid operation mode requested.
Others	Crypto HAL module returned an error. resource lock/unlock was failed. Crypto Hash Framework module returned an error. See Common Error Codes for other possible return codes.

6.19.16 SF_CRYPTO_SIGNATURE_SignFinal

```
ssp_err_t SF_CRYPTO_SIGNATURE_SignFinal ( sf_crypto_signature_ctrl_t
*const p_api_ctrl , sf_crypto_data_handle_t const *const p_message ,
 sf_crypto_data_handle_t *const p_dest )
```

6.19.16.1 Brief description

SSP Crypto Signature Framework Signature Final operation.

6.19.16.2 Detailed description

Table 347:Return values

Name	Description
SSP_SUCCESS	Sign Final operation was performed successfully.
SSP_ERR_ASSERTION	NULL is passed through the argument.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_INVALID_CALL	Invalid call to this API.
SSP_ERR_CRYPTO_INVALID_SIZE	Not enough space to store signature.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_BUF_OVERFLOW	Update data exceeded the block size.
Others	Crypto HAL module returned an error. Crypto Hash Framework module returned an error. See Common Error Codes for other possible return codes.

6.19.17 SF_CRYPTO_SIGNATURE_VerifyFinal

```
ssp_err_t SF_CRYPTO_SIGNATURE_VerifyFinal ( sf_crypto_signature_ctrl_t
*const p_api_ctrl , sf_crypto_data_handle_t const *const p_signature ,
 sf_crypto_data_handle_t const *const p_message )
```

6.19.17.1 Brief description

SSP Crypto Signature Framework Signature-Verification Update operation.

6.19.17.2 Detailed description

Table 348:Return values

Name	Description
SSP_SUCCESS	Verify final operation is performed successfully.
SSP_ERR_ASSERTION	NULL is passed through the argument.
SSP_ERR_INTERNAL	Critical internal error.
SSP_ERR_INVALID_CALL	Invalid call to this API.
SSP_ERR_CRYPTO_INVALID_SIZE	Invalid signature length.
SSP_ERR_UNSUPPORTED	Invalid Hash module request.
SSP_ERR_CRYPTO_BUF_OVERFLOW	Update data exceeded the block size.
Others	Crypto HAL module returned an error. Crypto Hash Framework module returned an error. See Common Error Codes for other possible return codes.

6.19.18 SF_CRYPTO_SIGNATURE_VersionGet

```
ssp_err_t SF_CRYPTO_SIGNATURE_VersionGet ( ssp_version_t  *const p_version )
```

6.19.18.1 Brief description

Gets driver version based on compile time macros.

6.19.18.2 Detailed description

Table 349:Return values

Name	Description
SSP_SUCCESS	Function returned successfully.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

6.19.19 sf_crypto_signature_validate_sign_operation_state_transition

```
ssp_err_t sf_crypto_signature_validate_sign_operation_state_transition ( sf_cryp
to_signature_instance_ctrl_t  * p_ctrl ,
 sf_crypto_signature_operation_state_t  next_state )
```

6.19.19.1 Brief description

SSP Crypto Signature Framework State transition validation for sign operation.

6.19.19.2 Detailed description

Table 350:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to Crypto Signature Framework instance control block structure.
next_state	in	Requested next state.

Table 351:Return values

Name	Description
SSP_SUCCESS	Valid call to the calling API.
SSP_ERR_INVALID_CALL	Invalid call to the calling API.

6.19.20 sf_crypto_signature_validate_verify_operation_state_transition

```
ssp_err_t sf_crypto_signature_validate_verify_operation_state_transition ( sf_cr
ypto_signature_instance_ctrl_t  * p_ctrl ,
 sf_crypto_signature_operation_state_t  next_state )
```

6.19.20.1 Brief description

SSP Crypto Signature Framework State transition validation for verify operation.

6.19.20.2 Detailed description

Table 352:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to Crypto Signature Framework instance control block structure.
next_state	in	Requested next state.

Table 353:Return values

Name	Description
SSP_SUCCESS	Valid call to the calling API.
SSP_ERR_INVALID_CALL	Invalid call to the calling API.

6.19.21 API Data

6.19.21.1 sf_crypto_signature_state_t

`sf_crypto_signature_state_t`

Detailed description

State codes for the SSP Crypto Signature framework module. Once the module is opened successfully, then the state is transition to OPENED state. After sign/verify operations, the Signature framework module must be closed with CLOSED state.

Enumerated values

Name	Description
SF_CRYPTO_SIGNATURE_CLOSED	The Signature module is closed.
SF_CRYPTO_SIGNATURE_OPENED	The Signature module is opened.

6.19.21.2 sf_crypto_signature_operation_state_t

`sf_crypto_signature_operation_state_t`

Detailed description

Internal state codes for the SSP Crypto Signature framework module.

Enumerated values

Name	Description
SF_CRYPTO_SIGNATURE_OPERATION_STATE_OPEN	Module opened to perform sign/verify operation.
SF_CRYPTO_SIGNATURE_OPERATION_STATE_SIGN_INITIALIZED	Context is initialized for Sign Operation.
SF_CRYPTO_SIGNATURE_OPERATION_STATE_SIGN_UPDATED	Sign operation is in progress.
SF_CRYPTO_SIGNATURE_OPERATION_STATE_SIGN_FINALIZED	Sign operation has been completed.
SF_CRYPTO_SIGNATURE_OPERATION_STATE_VERIFY_INITIALIZED	Context is initialized for Verify Operation.
SF_CRYPTO_SIGNATURE_OPERATION_STATE_VERIFY_UPDATED	Verify operation is in progress.
SF_CRYPTO_SIGNATURE_OPERATION_STATE_VERIFY_FINALIZED	Verify operation has been completed.

6.19.22 Extensions

6.19.22.1 sf_crypto_signature_context_t

sf_crypto_signature_context_t

Detailed description

Internal SSP Crypto Signature framework module context.

Variables

- sf_crypto_signature_mode_t operation_mode

  Operating mode. (Sign / Verify operation)
- sf_crypto_signature_algorithm_init_params_t * p_aglorithm_specific_params

  Algorithm specific parameters. OR hold formatted input data.

- sf_crypto_data_handle_t buffer

  Internal buffer to format input data
- uint8_t * p_key_data

  Buffer to hold private key in case of Sign Operations. OR. Buffer to hold public key in case of Verify Operations.
- uint32_t key_data_length

  Length of key data.

6.19.22.2 sf_crypto_signature_instance_ctrl_t

sf_crypto_signature_instance_ctrl_t

Detailed description

SSP Crypto Signature Framework instance control block

Variables

- sf_crypto_signature_state_t status

  Module status.
- sf_crypto_key_type_t key_type

  Key type.
- sf_crypto_key_size_t key_size

  Key size.
- sf_crypto_signature_operation_state_t operation_state

  Internal Operation state.
- sf_crypto_signature_context_t operation_context

  Context for sign / verify operations.
- sf_crypto_instance_ctrl_t * p_lower_lvl_common_ctrl

  Pointer to the Crypto Framework Common instance
- sf_crypto_api_t * p_lower_lvl_common_api

  Pointer to the Crypto Framework API instance
- void * p_hal_ctrl

  pointer to Crypto module control structure
- void * p_hal_api

  pointer to Crypto module API structure
- sf_crypto_hash_instance_t * p_lower_lvl_sf_crypto_hash

  pointer to Crypto Framework Hash instance

6.19.22.3 sf_crypto_signature_api_t

Detailed description

Shared Interface definition for the SSP Crypto Signature framework

6.20 GUIX Interface

Interface definition for Adapting Express Logic GUIX for Synergy graphics drivers.

6.20.1 Summary

This is the Interface of SF_EL_GX Framework module which ties Synergy graphics device drivers to GUIX. The interface provides following driver adaptation for GUIX:

- DISPLAY driver for displaying image on LCD(e.g. GLCDC)
- Dave/2d driver for image rendering by 2DG engine
- JPEG driver for the image rendering by JPEG engine
- Software image rendering with no hardware acceleration

Implemented by: GUIX Synergy Port

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

GUIX Interface description: GUIX Port Framework

6.20.2 Interface API

sf_el_gx_api_t

Function name	Description
.open	Open SSP GUIX adaptation framework.
.close	Close SSP GUIX adaptation framework.
.versionGet	Get version.
.setup	Express Logic GUIX Driver setup entry.

Function name	Description
.canvasInit	Canvas initialization. Set the memory address of the initial canvas.

6.20.3 Data structures

- sf_el_gx_callback_args_t
- sf_el_gx_cfg_t
- sf_el_gx_instance_t

6.20.4 Enumerations

- sf_el_gx_state_t
- sf_el_gx_device_t
- sf_el_gx_event_t

6.20.5 Typedefs

- sf_el_gx_ctrl_t

6.20.6 Defines

- #define SF_EL_GX_API_VERSION_MAJOR

  Initial value: `(1U)`

  The API version of GUIX integrated driver framework
- #define SF_EL_GX_API_VERSION_MINOR

  Initial value: `(5U)`

6.20.7 API Data

6.20.7.1 sf_el_gx_state_t

`sf_el_gx_state_t`

Detailed description

State codes for the SSP GUIX adaptation framework

Enumerated values

Name	Description
SF_EL_GX_CLOSED	
SF_EL_GX_OPENED	
SF_EL_GX_CONFIGURED	

6.20.7.2 sf_el_gx_device_t

`sf_el_gx_device_t`

Detailed description

Low level device code for the GUIX

Enumerated values

Name	Description
SF_EL_GX_DEVICE_NONE	Non hardware.
SF_EL_GX_DEVICE_DISPLAY	Display device.
SF_EL_GX_DEVICE_DRW	2D Graphics Engine
SF_EL_GX_DEVICE_JPEG	JPEG Decoder.

6.20.7.3 sf_el_gx_event_t

`sf_el_gx_event_t`

Detailed description

Display event codes

Enumerated values

Name	Description
SF_EL_GX_EVENT_ERROR	Low level driver error occurs.
SF_EL_GX_EVENT_DISPLAY_VSYNC	Display interface VSYNC.
SF_EL_GX_EVENT_UNDERFLOW	Display interface underflow.

6.20.7.4 sf_el_gx_ctrl_t

```
typedef void sf_el_gx_ctrl_t
```

Detailed description

GUIX adaptation framework control block. Allocate an instance specific control block to pass into the GUIX adaptation framework API calls. Implemented as

- sf_el_gx_instance_ctrl_t

6.20.8 API Structures

6.20.8.1 sf_el_gx_callback_args_t

sf_el_gx_callback_args_t

Detailed description

Callback arguments for the SSP GUIX adaptation framework

Variables

- sf_el_gx_device_t device

  Device code.

- sf_el_gx_event_t event

  Event code of the low level hardware.

- uint32_t error

  Error code if SF_EL_GX_EVENT_ERROR.

6.20.8.2 sf_el_gx_cfg_t

sf_el_gx_cfg_t

Detailed description

Configuration structure for the SSP GUIX adaptation framework

Variables

- display_instance_t * p_display_instance

  Pointer to a display instance.

- display_runtime_cfg_t * p_display_runtime_cfg

  Pointer to a runtime display configuration.

- void * p_canvas

  Pointer to a canvas(reserved)

- void * p_framebuffer_a

  Pointer to a frame buffer(A)

- void * p_framebuffer_b

  Pointer to a frame buffer(B)
- void(* p_callback)(sf_el_gx_callback_args_t *p_args)

  Pointer to callback function.
- void * p_context

  Pointer to a context.
- void * p_jpegbuffer

  Pointer to a JPEG work buffer.
- uint32_t jpegbuffer_size

  Size of a JPEG work buffer.
- uint16_t rotation_angle

  Screen rotation angle(0/90/270)
- void * p_sf_jpeg_decode_instance

  Pointer to a JPEG framework instance.
- _Bool dave2d_buffer_cache_enabled

  D/AVE 2D buffer cache enabled/disabled.

6.20.8.3 sf_el_gx_api_t

sf_el_gx_api_t

Detailed description

Shared Interface definition for the SSP GUIX adaptation framework

6.20.8.4 open

```
ssp_err_t(* sf_el_gx_api_t::open) (sf_el_gx_ctrl_t *const p_ctrl, sf_el_gx_cfg_t
const *const p_cfg)
```

Detailed description

Open SSP GUIX adaptation framework. Implemented as

- SF_EL_GX_Open

Table 354:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to SF_EL_GX control block structure. Must be declared by user. Value set here.

Table 354:Parameters (Continued)

Name	Direction	Description
p_cfg	in	Pointer to SF_EL_GX configuration structure. All elements of this structure must be set by user.

Parameter p_ctrl

Definition: `sf_el_gx_ctrl_t*const p_ctrl`

GUIX adaptation framework control block. Allocate an instance specific control block to pass into the GUIX adaptation framework API calls. Implemented assf_el_gx_instance_ctrl_t

Parameter p_cfg

Definition: `sf_el_gx_cfg_t const *const p_cfg`

Configuration structure for the SSP GUIX adaptation framework

- `sf_el_gx_cfg_t`::`display_instance_t`

  Pointer to a display instance.

- `sf_el_gx_cfg_t`::`display_runtime_cfg_t`

  Pointer to a runtime display configuration.

- `sf_el_gx_cfg_t`::`p_canvas`

  Pointer to a canvas(reserved)

- `sf_el_gx_cfg_t`::`p_framebuffer_a`

  Pointer to a frame buffer(A)

- `sf_el_gx_cfg_t`::`p_framebuffer_b`

  Pointer to a frame buffer(B)

- `sf_el_gx_cfg_t`::`p_callback`

  Pointer to callback function.

- `sf_el_gx_cfg_t`::`p_context`

  Pointer to a context.

- `sf_el_gx_cfg_t`::`p_jpegbuffer`

  Pointer to a JPEG work buffer.

- `sf_el_gx_cfg_t`::`jpegbuffer_size`

  Size of a JPEG work buffer.

- `sf_el_gx_cfg_t`::`rotation_angle`

  Screen rotation angle(0/90/270)

- `sf_el_gx_cfg_t`::`p_sf_jpeg_decode_instance`

  Pointer to a JPEG framework instance.

- sf_el_gx_cfg_t::dave2d_buffer_cache_enabled

  D/AVE 2D buffer cache enabled/disabled.

6.20.8.5 close

```
ssp_err_t(* sf_el_gx_api_t::close) (sf_el_gx_ctrl_t *const p_ctrl)
```

Detailed description

Close SSP GUIX adaptation framework. Implemented as

- SF_EL_GX_Close

Table 355:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to SF_EL_GX control block structure.

Parameter p_ctrl

Definition: sf_el_gx_ctrl_t*const p_ctrl

GUIX adaptation framework control block. Allocate an instance specific control block to pass into the GUIX adaptation framework API calls. Implemented assf_el_gx_instance_ctrl_t

6.20.8.6 versionGet

```
ssp_err_t(* sf_el_gx_api_t::versionGet) (ssp_version_t *p_version)
```

Detailed description

Get version. Implemented as

- SF_EL_GX_VersionGet

Table 356:Parameters

Name	Direction	Description
p_version	in	Pointer to the memory to store the version information.

Parameter p_version

6.20.8.7 setup

```
UINT(* sf_el_gx_api_t::setup) (GX_DISPLAY *p_display)
```

Detailed description

Express Logic GUIX Driver setup entry. Implemented as

- SF_EL_GX_Setup

Table 357:Parameters

Name	Direction	Description
p_display	in	Pointer to GUIX display driver setup function.

Parameter p_display

`const`

6.20.8.8 canvasInit

```
ssp_err_t(* sf_el_gx_api_t::canvasInit) (sf_el_gx_ctrl_t *const p_ctrl,
GX_WINDOW_ROOT *p_window_root)
```

Detailed description

Canvas initialization. Set the memory address of the initial canvas. Implemented as

- SF_EL_GX_CanvasInit

Table 358:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to SF_EL_GX control block structure.
p_window_root	in	Pointer to GUIX root window context.

Parameter p_ctrl

Definition: `sf_el_gx_ctrl_t*const p_ctrl`

GUIX adaptation framework control block. Allocate an instance specific control block to pass into the GUIX adaptation framework API calls. Implemented assf_el_gx_instance_ctrl_t

Parameter p_window_root

`const`

6.20.8.9 sf_el_gx_instance_t

sf_el_gx_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_el_gx_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- sf_el_gx_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- sf_el_gx_api_t const * p_api

  Pointer to the API structure for this instance.

6.21 External IRQ Framework Interface

RTOS-integrated External IRQ Framework Interface.

6.21.1 Summary

This module is a ThreadX-aware external IRQ Framework Interface for external inputs such as switches or other binary signals. The Interface is implemented by External IRQ Framework.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

External IRQ Framework Interface description: External IRQ Framework

6.21.2 Interface API

sf_external_irq_api_t

Function name	Description
.open	Acquire mutex, then handle driver initialization at the HAL layer.
.wait	Wait for the next external interrupt expiration, then return.
.versionGet	Get version and store it in provided pointer p_version.
.close	Release channel mutex and close channel at HAL layer.

6.21.3 Data structures

- sf_external_irq_cfg_t
- sf_external_irq_instance_t

6.21.4 Enumerations

- sf_external_irq_event_t

6.21.5 Typedefs

- sf_external_irq_ctrl_t

6.21.6 Defines

- #define SF_EXTERNAL_IRQ_API_VERSION_MAJOR

  Initial value: `(1U)`
- #define SF_EXTERNAL_IRQ_API_VERSION_MINOR

  Initial value: `(2U)`

6.21.7 API Data

6.21.7.1 sf_external_irq_event_t

```
sf_external_irq_event_t
```

Detailed description

Options for what should happen when the external interrupt expires.

Enumerated values

Name	Description
SF_EXTERNAL_IRQ_EVENT_NONE	Nothing happens during expiration. Can be used for data transfer.
SF_EXTERNAL_IRQ_EVENT_SEMAPHORE_PUT	Posts to internal semaphore. Select this if using SF_EXTERNAL_IRQ_Wait.

6.21.7.2 sf_external_irq_ctrl_t

```
typedef void sf_external_irq_ctrl_t
```

Detailed description

External interrupt control block. Allocate an instance specific control block to pass into the external interrupt framework API calls. Implemented as

- sf_external_irq_instance_ctrl_t

6.21.8 API Structures

6.21.8.1 sf_external_irq_cfg_t

sf_external_irq_cfg_t

Detailed description

Configuration for RTOS integrated external interrupt driver

Variables

- external_irq_instance_t const * p_lower_lvl_irq

  All info needed to work with lower layer

- sf_external_irq_event_t event

  Select what happens when the external IRQ is triggered.

6.21.8.2 sf_external_irq_api_t

sf_external_irq_api_t

Detailed description

External IRQ framework API structure. External IRQ implementations use the following API.

6.21.8.3 open

```
ssp_err_t(* sf_external_irq_api_t::open) (sf_external_irq_ctrl_t *const p_ctrl,
sf_external_irq_cfg_t const *const p_cfg)
```

Detailed description

Acquire mutex, then handle driver initialization at the HAL layer. Implemented as

- SF_EXTERNAL_IRQ_Open

Table 359:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a structure allocated by user. The device control structure is initialized in this function.

Table 359:Parameters (Continued)

Name	Direction	Description
p_cfg	in	Pointer to configuration structure. All elements of the structure must be set by user.

Parameter p_ctrl

Definition: `sf_external_irq_ctrl_t*const p_ctrl`

External interrupt control block. Allocate an instance specific control block to pass into the external interrupt framework API calls. Implemented assf_external_irq_instance_ctrl_t

Parameter p_cfg

Definition: `sf_external_irq_cfg_t const *const p_cfg`

Configuration for RTOS integrated external interrupt driver

- `sf_external_irq_cfg_t`::`external_irq_instance_t`

  All info needed to work with lower layer

- `sf_external_irq_cfg_t`::`sf_external_irq_event_t`

  Select what happens when the external IRQ is triggered.

  Enumerated as:

  – SF_EXTERNAL_IRQ_EVENT_NONE

  – SF_EXTERNAL_IRQ_EVENT_SEMAPHORE_PUT

6.21.8.4 wait

```
ssp_err_t(* sf_external_irq_api_t::wait) (sf_external_irq_ctrl_t *const p_ctrl,
ULONG const timeout)
```

Detailed description

Wait for the next external interrupt expiration, then return.

NOTE: Call SF_EXTERNAL_IRQ_Open to configure the external IRQ before using this function. During SF_EXTERNAL_IRQ_Open, set sf_external_irq_cfg_t::sf_external_irq_event_t to SF_EXTERNAL_IRQ_EVENT_SEMAPHORE_PUT.

Implemented as

- SF_EXTERNAL_IRQ_Wait

Table 360:Parameters

Name	Direction	Description
p_ctrl	in	Handle set in SF_EXTERNAL_IRQ_Open.
timeout	in	ThreadX timeout. Select TX_NO_WAIT, a value in system clock counts between 1 and 0xFFFFFFFF, or TX_WAIT_FOREVER.

Parameter p_ctrl

Definition: `sf_external_irq_ctrl_t*const p_ctrl`

External interrupt control block. Allocate an instance specific control block to pass into the external interrupt framework API calls. Implemented assf_external_irq_instance_ctrl_t

Parameter timeout

`const`

6.21.8.5 versionGet

```
ssp_err_t(* sf_external_irq_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Get version and store it in provided pointer p_version. Implemented as

- SF_EXTERNAL_IRQ_VersionGet

Table 361:Parameters

Name	Direction	Description
p_version	out	Code and API version used stored here.

Parameter p_version

6.21.8.6 close

```
ssp_err_t(* sf_external_irq_api_t::close) (sf_external_irq_ctrl_t *const p_ctrl)
```

Detailed description

Release channel mutex and close channel at HAL layer. Implemented as

- SF_EXTERNAL_IRQ_Close

Table 362:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for this external interrupt.

Parameter p_ctrl

Definition: `sf_external_irq_ctrl_t*const p_ctrl`

External interrupt control block. Allocate an instance specific control block to pass into the external interrupt framework API calls. Implemented assf_external_irq_instance_ctrl_t

6.21.8.7 sf_external_irq_instance_t

sf_external_irq_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_external_irq_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- sf_external_irq_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- sf_external_irq_api_t const * p_api

  Pointer to the API structure for this instance.

6.22 I2C Framework

RTOS-integrated I2C Framework Interface.

6.22.1 Summary

This is a ThreadX-aware I2C interface. It can be implemented by several hardware peripherals at the HAL layer through the I2C interface I2C Master Interface.

The connection to the HAL layer is established by passing in a driver structure in SF_I2C_Open.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers

- Using SSP Modules

SPI Framework Interface description: I2C Framework

6.22.2 Interface API

sf_i2c_api_t

Function name	Description
.open	Open a designated I2C device on a bus.
.read	Receive data from I2C device.
.write	Transmit data to I2C device.
.reset	Abort any in-progress transfer and force the I2C peripheral into a ready state. This function safely terminates any in-progress I2C transfer with the device. If a transfer is aborted, the user is be notified via callback with an abort event. Since the callback is optional, this function also returns a specific error code in this situation.
.close	Disable the I2C device designated by the control handle. Close the RTOS services used by the bus if no devices are connected to the bus.
.lock	Lock the bus for a device. Locking allows devices to reserve a bus to themselves for a given period of time (i.e. between lock and unlock). This allows devices to complete several reads and writes on the bus without interrupt, which is required in some instances.
.unlock	Unlock the bus from a particular device and make it available for other devices. This allows other devices to use bus for reads and writes on the bus.
.version	Get I2C framework version.

6.22.3 Data structures

- sf_i2c_bus_t
- sf_i2c_cfg_t
- sf_i2c_instance_t

6.22.4 Enumerations

- sf_i2c_dev_state_t

6.22.5 Typedefs

- sf_i2c_ctrl_t

6.22.6 Defines

- #define SF_I2C_API_VERSION_MAJOR

  Initial value: `(1U)`

  Includes driver interface.
- #define SF_I2C_API_VERSION_MINOR

  Initial value: `(3U)`

6.22.7 API Data

6.22.7.1 sf_i2c_dev_state_t

```
sf_i2c_dev_state_t
```

Detailed description

SF I2C device state

Enumerated values

Name	Description
SF_I2C_DEV_STATE_CLOSED	I2C device is closed.
SF_I2C_DEV_STATE_OPENED	I2C device is opened.

6.22.7.2 sf_i2c_ctrl_t

```
typedef void sf_i2c_ctrl_t
```

Detailed description

I2C framework control block. Allocate an instance specific control block to pass into the I2C framework API calls. Implemented as

- sf_i2c_instance_ctrl_t

6.22.8 API Structures

6.22.8.1 sf_i2c_bus_t

sf_i2c_bus_t

Detailed description

Data structure defining a I2C bus.

Variables

- uint8_t channel

  Channel.
- TX_MUTEX * p_lock_mutex

  Lock mutex handle for this channel.
- TX_EVENT_FLAGS_GROUP * p_sync_eventflag

  Pointer to the event flag object for I2C data transfer.
- sf_i2c_ctrl_t ** pp_curr_ctrl

  Current device using the bus (by switching the address)
- uint8_t * p_bus_name

  User-supplied name to identify the bus. Useful for debugging.
- i2c_api_master_t const * p_lower_lvl_api

  Pointer to I2C HAL interface to be used in the framework.
- uint8_t device_count

  Number of devices on the bus; initialize to 0.
- sf_i2c_ctrl_t ** pp_curr_bus_ctrl

  Device configured on the bus (low level configuration)

6.22.8.2 sf_i2c_cfg_t

sf_i2c_cfg_t

Detailed description

Configuration for Framework I2C driver

Variables

- sf_i2c_bus_t * p_bus

  Bus used by the device.
- i2c_cfg_t const * p_lower_lvl_cfg

  Pointer to I2C HAL configuration.

6.22.8.3 sf_i2c_api_t

sf_i2c_api_t

Detailed description

Shared Interface definition for I2C Framework

6.22.8.4 open

```
ssp_err_t(* sf_i2c_api_t::open) (sf_i2c_ctrl_t *p_ctrl, sf_i2c_cfg_t const *const p_cfg)
```

Brief description

Open a designated I2C device on a bus.

Detailed description

Implemented as

- SF_I2C_Open

Table 363:Parameters

Name	Direction	Description
p_ctrl	out	Control handle for I2C framework driver context for a device (Value returns from this function). This value must be cleared by user.
p_cfg	in	I2C configuration includes I2C bus and low level configuration

Parameter p_ctrl

Definition: `sf_i2c_ctrl_t*p_ctrl`

I2C framework control block. Allocate an instance specific control block to pass into the I2C framework API calls. Implemented assf_i2c_instance_ctrl_t

Parameter p_cfg

Definition: `sf_i2c_cfg_t const *const p_cfg`

Configuration for Framework I2C driver

- `sf_i2c_cfg_t`::`sf_i2c_bus_t`

  Bus used by the device.

- `sf_i2c_cfg_t`::`i2c_cfg_t`

  Pointer to I2C HAL configuration.

6.22.8.5 read

```
ssp_err_t(* sf_i2c_api_t::read) (sf_i2c_ctrl_t *const p_ctrl, uint8_t *const
p_dest, uint32_t const bytes, bool const restart, uint32_t const timeout)
```

Brief description

Receive data from I2C device.

Detailed description

Implemented as

- SF_I2C_Read

Table 364:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to previously opened I2C SF control structure.
p_dest	in	Pointer to location to store read data.
bytes	in	Number of bytes to read.
restart	in	Indicates whether the restart condition should be issued after reading.
timeout	in	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

Parameter p_ctrl

Definition: `sf_i2c_ctrl_t*const p_ctrl`

I2C framework control block. Allocate an instance specific control block to pass into the I2C framework API calls. Implemented assf_i2c_instance_ctrl_t

Parameter p_dest

`uint8_t`

Parameter bytes

`uint32_t`

Parameter restart

`const`

Parameter timeout

`uint32_t`

6.22.8.6 write

```
ssp_err_t(* sf_i2c_api_t::write) (sf_i2c_ctrl_t *const p_ctrl, uint8_t *const
p_src, uint32_t const bytes, bool const restart, uint32_t const timeout)
```

Brief description

Transmit data to I2C device.

Detailed description

Implemented as

- SF_I2C_Write

Table 365:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to previously opened I2C control structure.
p_src	in	Pointer to location to get write data.
bytes	in	Number of bytes to write.
restart	in	Indicates whether the restart condition should be issued after writing.
timeout	in	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

Parameter p_ctrl

Definition: `sf_i2c_ctrl_t*const p_ctrl`

I2C framework control block. Allocate an instance specific control block to pass into the I2C framework API calls. Implemented assf_i2c_instance_ctrl_t

Parameter p_src

`uint8_t`

Parameter bytes

`uint32_t`

Parameter restart

`const`

Parameter timeout

`uint32_t`

6.22.8.7 reset

```
ssp_err_t(* sf_i2c_api_t::reset) (sf_i2c_ctrl_t *const p_ctrl, uint32_t const timeout)
```

Brief description

Abort any in-progress transfer and force the I2C peripheral into a ready state.

Detailed description

Implemented as

- SF_I2C_Reset

This function safely terminates any in-progress I2C transfer with the device. If a transfer is aborted, the user is be notified via callback with an abort event. Since the callback is optional, this function also returns a specific error code in this situation.

Table 366:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to device control block initialized in Open call for I2C driver.
timeout	in	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

Parameter p_ctrl

Definition: `sf_i2c_ctrl_t*const p_ctrl`

I2C framework control block. Allocate an instance specific control block to pass into the I2C framework API calls. Implemented assf_i2c_instance_ctrl_t

Parameter timeout

`uint32_t`

6.22.8.8 close

```
ssp_err_t(* sf_i2c_api_t::close) (sf_i2c_ctrl_t *const p_ctrl)
```

Brief description

Disable the I2C device designated by the control handle. Close the RTOS services used by the bus if no devices are connected to the bus.

Detailed description

Implemented as

- SF_I2C_Close

Table 367:Parameters

Name	Direction	Description
p_ctrl	in	Control handle for I2C framework driver context for a device

Parameter p_ctrl

Definition: `sf_i2c_ctrl_t*const p_ctrl`

I2C framework control block. Allocate an instance specific control block to pass into the I2C framework API calls. Implemented assf_i2c_instance_ctrl_t

6.22.8.9 lock

```
ssp_err_t(* sf_i2c_api_t::lock) (sf_i2c_ctrl_t *const p_ctrl)
```

Brief description

Lock the bus for a device. Locking allows devices to reserve a bus to themselves for a given period of time (i.e. between lock and unlock). This allows devices to complete several reads and writes on the bus without interrupt, which is required in some instances.

Detailed description

Implemented as

- SF_I2C_Lock

Table 368:Parameters

Name	Direction	Description
p_ctrl	in	Control handle for I2C framework driver context for a device

Parameter p_ctrl

Definition: `sf_i2c_ctrl_t*const p_ctrl`

I2C framework control block. Allocate an instance specific control block to pass into the I2C framework API calls. Implemented assf_i2c_instance_ctrl_t

6.22.8.10 unlock

```
ssp_err_t(* sf_i2c_api_t::unlock) (sf_i2c_ctrl_t *const p_ctrl)
```

Brief description

Unlock the bus from a particular device and make it available for other devices. This allows other devices to use bus for reads and writes on the bus.

Detailed description

Implemented as

- SF_I2C_Unlock

Table 369:Parameters

Name	Direction	Description
p_ctrl	in	Control handle for I2C framework driver context for a device

Parameter p_ctrl

Definition: `sf_i2c_ctrl_t*const p_ctrl`

I2C framework control block. Allocate an instance specific control block to pass into the I2C framework API calls. Implemented assf_i2c_instance_ctrl_t

6.22.8.11 version

```
ssp_err_t(* sf_i2c_api_t::version) (ssp_version_t *const p_version)
```

Brief description

Get I2C framework version.

Detailed description

Implemented as

- SF_I2C_VersionGet

Table 370:Parameters

Name	Direction	Description
p_ctrl	in	Handle for I2C framework control block for a device

Parameter p_ctrl

6.22.8.12 sf_i2c_instance_t

sf_i2c_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_i2c_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- sf_i2c_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- sf_i2c_api_t const * p_api

  Pointer to the API structure for this instance.

6.23 JPEG Decode Framework Interface

RTOS-integrated JPEG Decode Framework Interface.

6.23.1 Summary

This is a ThreadX aware generic JPEG decoding framework for run-time JPEG decode applications. It can be implemented by either hardware or software. For Synergy parts, the interface is implemented by the on-chip JPEG decoding engine. The connection to the HAL layer is established by passing in a driver structure in SF_JPEG_Decode_Open.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Framework JPEG Decode Interface description: JPEG Decode Framework

6.23.2 Interface API

sf_jpeg_decode_api_t

Function name	Description
.open	Acquire mutex, then handle driver initialization at the HAL layer. This function releases mutex before it returns to the caller.
.inputBufferSet	Feed data into JPEG codec module.
.outputBufferSet	Read processed data from JPEG codec module.
.linesDecodedGet	Obtain number of lines JPEG codec decoded.
.horizontalStrideSet	Configure the horizontal stride value.

Function name	Description
.imageSubsampleSet	Configure the horizontal and vertical subsample values. This allows an application to reduce the size of the decoded image.
.wait	Wait for current JPEG codec operation to finish
.statusGet	Obtain JPEG codec status
.imageSizeGet	Obtain the size of the image. This function is only useful for decoding a JPEG image.
.pixelFormatGet	Obtain the pixel format of the image. This function is only useful for decoding a JPEG image.
.close	Closes JPEG codec device. Un-finished codec operation is interrupted, and output data are discarded.
.versionGet	Gets version and stores it in provided pointer p_version.

6.23.3 Data structures

- sf_jpeg_decode_cfg_t
- sf_jpeg_decode_instance_t

6.23.4 Typedefs

- sf_jpeg_decode_ctrl_t

6.23.5 Defines

- #define SF_JPEG_DECODE_API_VERSION_MAJOR

  Initial value: `(1U)`

  Version of the API defined in this file
- #define SF_JPEG_DECODE_API_VERSION_MINOR

  Initial value: `(2U)`

6.23.6 API Data

6.23.6.1 sf_jpeg_decode_ctrl_t

```
typedef void sf_jpeg_decode_ctrl_t
```

Detailed description

JPEG decode framework control block. Allocate an instance specific control block to pass into the JPEG decode framework API calls. Implemented as

- sf_jpeg_decode_instance_ctrl_t

6.23.7 API Structures

6.23.7.1 sf_jpeg_decode_cfg_t

sf_jpeg_decode_cfg_t

Detailed description

Configuration for RTOS integrated JPEG driver

Variables

- jpeg_decode_instance_t const * p_lower_lvl_jpeg_decode

  Pointer to a driver structure that implements this interface. Pre-configured driver structures are located in r_jpeg_-decode.c and extern'ed in r_jpeg_decode.h.

6.23.7.2 sf_jpeg_decode_api_t

sf_jpeg_decode_api_t

Detailed description

JPEG Decode API structure. Implementations will use the following API.

6.23.7.3 open

```
(* sf_jpeg_decode_api_t::open) (sf_jpeg_decode_ctrl_t *const p_ctrl,
sf_jpeg_decode_cfg_t const *const p_cfg)
```

Detailed description

Acquire mutex, then handle driver initialization at the HAL layer. This function releases mutex before it returns to the caller.

Table 371:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a structure allocated by user. Elements initialized here.

Table 371:Parameters (Continued)

Name	Direction	Description
p_cfg	in	Pointer to configuration structure. All elements of the structure must be set by user.

Parameter p_ctrl

Definition: `sf_jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode framework control block. Allocate an instance specific control block to pass into the JPEG decode framework API calls. Implemented assf_jpeg_decode_instance_ctrl_t

Parameter p_cfg

Definition: `sf_jpeg_decode_cfg_t const *const p_cfg`

Configuration for RTOS integrated JPEG driver

- `sf_jpeg_decode_cfg_t`::`jpeg_decode_instance_t`

  Pointer to a driver structure that implements this interface. Pre-configured driver structures are located in r_jpeg_-decode.c and extern'ed in r_jpeg_decode.h.

6.23.7.4 inputBufferSet

```
(* sf_jpeg_decode_api_t::inputBufferSet) (sf_jpeg_decode_ctrl_t *const p_ctrl,
void *const p_buffer, uint32_t const buffer_size)
```

Detailed description

Feed data into JPEG codec module.

Table 372:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block initialized in SF_JPEG_Decode_Open.
p_buffer	in	Buffer contains data to be processed by the JPEG codec module. The buffer starting address must be 8-byte aligned
buffer_size	in	Size of the data buffer, must be multiple of 8 bytes

Parameter p_ctrl

Definition: `sf_jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode framework control block. Allocate an instance specific control block to pass into the JPEG decode framework API calls. Implemented assf_jpeg_decode_instance_ctrl_t

Parameter p_buffer

`const`

Parameter buffer_size

`uint32_t`

6.23.7.5 outputBufferSet

```
(* sf_jpeg_decode_api_t::outputBufferSet) (sf_jpeg_decode_ctrl_t *const p_ctrl,
void *p_buffer, uint32_t buffer_size)
```

Detailed description

Read processed data from JPEG codec module.

Table 373:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block initialized in SF_JPEG_Decode_Open.
p_buffer	in	User-supplied buffer space to hold output from JPEG codec module. The buffer starting address must be 8-byte aligned
buffer_size	in	Size of the output data buffer

Parameter p_ctrl

Definition: `sf_jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode framework control block. Allocate an instance specific control block to pass into the JPEG decode framework API calls. Implemented assf_jpeg_decode_instance_ctrl_t

Parameter p_buffer

`const`

Parameter buffer_size

`uint32_t`

6.23.7.6 linesDecodedGet

```
(* sf_jpeg_decode_api_t::linesDecodedGet) (sf_jpeg_decode_ctrl_t *const p_ctrl,
uint32_t *const p_lines)
```

Detailed description

Obtain number of lines JPEG codec decoded.

Table 374:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block initialized in SF_JPEG_Decode_Open.
p_lines	out	Number of lines decoded into the output buffer.

Parameter p_ctrl

Definition: `sf_jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode framework control block. Allocate an instance specific control block to pass into the JPEG decode framework API calls. Implemented assf_jpeg_decode_instance_ctrl_t

Parameter p_lines

`uint32_t`

6.23.7.7 horizontalStrideSet

```
(* sf_jpeg_decode_api_t::horizontalStrideSet) (sf_jpeg_decode_ctrl_t *const
p_ctrl, uint32_t horizontal_stride)
```

Detailed description

Configure the horizontal stride value.

Table 375:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block initialized in SF_JPEG_Decode_Open.
horizontal_stride	out	Set the horizontal stride value, in pixels

Parameter p_ctrl

Definition: `sf_jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode framework control block. Allocate an instance specific control block to pass into the JPEG decode framework API calls. Implemented assf_jpeg_decode_instance_ctrl_t

Parameter horizontal_stride

uint32_t

6.23.7.8 imageSubsampleSet

(* sf_jpeg_decode_api_t::imageSubsampleSet) (sf_jpeg_decode_ctrl_t *const p_ctrl, horizontal_subsample, vertical_subsample)

Detailed description

Configure the horizontal and vertical subsample values. This allows an application to reduce the size of the decoded image.

Table 376:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block initialized in SF_JPEG_Decode_Open.
horizontal_subsample	in	Set the horizontal subsample value
vertical_subsample	in	Set the vertical subsample value

Parameter p_ctrl

Definition: sf_jpeg_decode_ctrl_t*const p_ctrl

JPEG decode framework control block. Allocate an instance specific control block to pass into the JPEG decode framework API calls. Implemented assf_jpeg_decode_instance_ctrl_t

Parameter horizontal_subsample

Parameter vertical_subsample

6.23.7.9 wait

(* sf_jpeg_decode_api_t::wait) (sf_jpeg_decode_ctrl_t *const p_ctrl, *const p_status, uint32_t timeout)

Detailed description

Wait for current JPEG codec operation to finish

Table 377:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block initialized in SF_JPEG_Decode_Open.
p_status	out	Status of current JPEG codec module
timeout	out	Amount of time (in ThreadX ticks) to wait

Parameter p_ctrl

Definition: `sf_jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode framework control block. Allocate an instance specific control block to pass into the JPEG decode framework API calls. Implemented assf_jpeg_decode_instance_ctrl_t

Parameter p_status

Parameter timeout

`uint32_t`

6.23.7.10 statusGet

```
(* sf_jpeg_decode_api_t::statusGet) (sf_jpeg_decode_ctrl_t *const p_ctrl, *const
p_status)
```

Detailed description

Obtain JPEG codec status

Table 378:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block initialized in SF_JPEG_Decode_Open.
p_status	out	Status of current JPEG codec module

Parameter p_ctrl

Definition: `sf_jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode framework control block. Allocate an instance specific control block to pass into the JPEG decode framework API calls. Implemented assf_jpeg_decode_instance_ctrl_t

Parameter p_status

6.23.7.11 imageSizeGet

```
(* sf_jpeg_decode_api_t::imageSizeGet) (sf_jpeg_decode_ctrl_t *const p_ctrl,
uint16_t *p_horizontal_size, uint16_t *p_vertical_size)
```

Detailed description

Obtain the size of the image. This function is only useful for decoding a JPEG image.

Table 379:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block initialized in SF_JPEG_Decode_Open.
p_horizontal_size	out	Width of the image, in pixels
p_vertical_size	out	Height of the image, in pixels

Parameter p_ctrl

Definition: `sf_jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode framework control block. Allocate an instance specific control block to pass into the JPEG decode framework API calls. Implemented assf_jpeg_decode_instance_ctrl_t

Parameter p_horizontal_size

`uint16_t`

Parameter p_vertical_size

`uint16_t`

6.23.7.12 pixelFormatGet

```
(* sf_jpeg_decode_api_t::pixelFormatGet) (sf_jpeg_decode_ctrl_t *const p_ctrl,
*const p_color_space)
```

Detailed description

Obtain the pixel format of the image. This function is only useful for decoding a JPEG image.

Table 380:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block initialized in SF_JPEG_Decode_Open.
p_color_space	out	Color space of the image

Parameter p_ctrl

Definition: `sf_jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode framework control block. Allocate an instance specific control block to pass into the JPEG decode framework API calls. Implemented assf_jpeg_decode_instance_ctrl_t

Parameter p_color_space

6.23.7.13 close

```
(* sf_jpeg_decode_api_t::close) (sf_jpeg_decode_ctrl_t *const p_ctrl)
```

Detailed description

Closes JPEG codec device. Un-finished codec operation is interrupted, and output data are discarded.

Table 381:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block set in SF_JPEG_Decode_Open.

Parameter p_ctrl

Definition: `sf_jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode framework control block. Allocate an instance specific control block to pass into the JPEG decode framework API calls. Implemented assf_jpeg_decode_instance_ctrl_t

6.23.7.14 versionGet

```
(* sf_jpeg_decode_api_t::versionGet) ( *const p_version)
```

Detailed description

Gets version and stores it in provided pointer p_version.

Table 382:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

6.23.7.15 sf_jpeg_decode_instance_t

sf_jpeg_decode_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_jpeg_decode_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_jpeg_decode_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_jpeg_decode_api_t const * p_api

  Pointer to the API structure for this instance.

6.24 Messaging Framework Interface

RTOS-integrated Messaging Framework Interface.

6.24.1 Summary

This module is a ThreadX-aware Messaging Framework. This Interface is implemented by Messaging Framework.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Messaging Interface description: Messaging Framework

6.24.2 Interface API

sf_message_api_t

Function name	Description
.open	Initialize message framework. Initiate the messaging framework control block, configure the work memory corresponding to the configuration parameters.
.close	Finalize message framework.
.bufferAcquire	Acquire buffer for message passing from the block.
.bufferRelease	Release buffer obtained from SF_MESSAGE_BufferAcquire.
.post	Post message to the subscribers.
.pend	Pend message.
.versionGet	Get the version of the messaging framework.

6.24.3 Data structures

- sf_message_header_t
- sf_message_instance_range_t
- sf_message_subscriber_t
- sf_message_subscriber_list_t
- sf_message_callback_args_t
- sf_message_post_err_t
- sf_message_buffer_ctrl_t
- sf_message_cfg_t
- sf_message_acquire_cfg_t
- sf_message_post_cfg_t
- sf_message_instance_t

6.24.4 Enumerations

- sf_message_state_t
- sf_message_callback_event_t

- sf_message_priority_t
- sf_message_release_option_t

6.24.5 Typedefs

- sf_message_ctrl_t

6.24.6 Defines

- #define SF_MESSAGE_API_VERSION_MAJOR

  Initial value: `(1U)`

  Version of the API defined in this file
- #define SF_MESSAGE_API_VERSION_MINOR

  Initial value: `(2U)`

6.24.7 API Data

6.24.7.1 sf_message_state_t

`sf_message_state_t`

Detailed description

Messaging framework state

Enumerated values

Name	Description
SF_MESSAGE_STATE_CLOSED	Messaging framework is closed.
SF_MESSAGE_STATE_OPENED	Messaging framework is opened.

6.24.7.2 sf_message_callback_event_t

`sf_message_callback_event_t`

Detailed description

Messaging callback response

Enumerated values

Name	Description
SF_MESSAGE_CALLBACK_EVENT_ACK	ACK response.
SF_MESSAGE_CALLBACK_EVENT_NAK	NAK response.

6.24.7.3 sf_message_priority_t

`sf_message_priority_t`

Detailed description

Messaging framework state

Enumerated values

Name	Description
SF_MESSAGE_PRIORITY_NORMAL	Gives a message to be sent normal priority.
SF_MESSAGE_PRIORITY_HIGH	Gives a message to be sent higher priority than previous ones.

6.24.7.4 sf_message_release_option_t

`sf_message_release_option_t`

Detailed description

Messaging option

Enumerated values

Name	Description
SF_MESSAGE_RELEASE_OPTION_NONE	No option.
SF_MESSAGE_RELEASE_OPTION_FORCED_RELEASE	Buffer forced release option.
SF_MESSAGE_RELEASE_OPTION_ACK	ACK response (note if both ACK and NAK are set at same time, NAK.

Name	Description
SF_MESSAGE_RELEASE_OPTION_NAK	NAK response.

6.24.7.5 sf_message_ctrl_t

```
typedef void sf_message_ctrl_t
```

Detailed description

Message framework control block. Allocate an instance specific control block to pass into the message framework API calls. Implemented as

- sf_message_instance_ctrl_t

6.24.8 API Structures

6.24.8.1 sf_message_header_t

sf_message_header_t

Detailed description

Message header definition

Variables

- uint32_t event
- uint32_t class_code

  Event class code.
- uint32_t class_instance

  Event class instance number.
- uint32_t code

  Event code.
- struct{} event_b

  See source code for definition of this member.
- union{} union{}

  See source code for definition of this member.

6.24.8.2 sf_message_instance_range_t

sf_message_instance_range_t

Detailed description

Subscriber lists definitions

Variables

- uint8_t start

  Start of the event class instance range.

- uint8_t end

  End of the event class instance range.

6.24.8.3 sf_message_subscriber_t

sf_message_subscriber_t

Detailed description

Message subscriber

Variables

- TX_QUEUE * p_queue

  Pointer to the message queue for subscriber thread.

- sf_message_instance_range_t instance_range

  Range of the event class instance to receive message.

6.24.8.4 sf_message_subscriber_list_t

sf_message_subscriber_list_t

Detailed description

Message subscriber list

Variables

- sf_message_event_class_t event_class

  Event class code.

- uint16_t number_of_nodes

  Number of nodes in the subscriber group.

- sf_message_subscriber_t ** pp_subscriber_group

  Subscriber group for the event class.

6.24.8.5 sf_message_callback_args_t

sf_message_callback_args_t

Detailed description

Message framework callback parameters

Variables

- sf_message_callback_event_t event

  Event code.

- void const * p_context

  Context provided to user during callback.

6.24.8.6 sf_message_post_err_t

sf_message_post_err_t

Detailed description

Post error information structure

Variables

- TX_QUEUE * p_queue

  Queue.

6.24.8.7 sf_message_buffer_ctrl_t

sf_message_buffer_ctrl_t

Detailed description

Buffer control block structure

Variables

- void(* p_callback)(sf_message_callback_args_t *)

  Optional user callback function.

- void const * p_context

  Context provided to user during callback.

- sf_message_buffer_ctrl_t::st_buffer_ctrl_flagstruct flag_b

6.24.8.8 sf_message_buffer_ctrl_t::st_buffer_ctrl_flag

sf_message_buffer_ctrl_t::st_buffer_ctrl_flag

Brief description

< Flags

Detailed description

Variables

- uint32_t semaphore

  Counting semaphore to prevent a buffer from being released.
- uint32_t buffer_keep

  Buffer keep request.
- uint32_t nak_response

  NAK (ORed logic for multiple subscribers)
- uint32_t reserved

  Reserved bits.
- uint32_t in_use

  Buffer in-use.

6.24.8.9 sf_message_cfg_t

sf_message_cfg_t

Detailed description

Messaging framework configuration structure definition

Variables

- void * p_work_memory_start

  Start address of the memory area.
- uint32_t work_memory_size_bytes

  Size of working memory area in bytes.
- uint32_t buffer_size

  Bytes of the message block.
- sf_message_subscriber_list_t ** pp_subscriber_lists

  Pointer array to the subscriber lists.
- uint8_t * p_block_pool_name

  Pointer to the block pool name.

6.24.8.10 sf_message_acquire_cfg_t

sf_message_acquire_cfg_t

Detailed description

Messaging framework Post API function configuration structure definition

Variables

- bool buffer_keep

  Buffer keep option.

6.24.8.11 sf_message_post_cfg_t

sf_message_post_cfg_t

Detailed description

Messaging framework Acquire API function configuration structure definition

Variables

- sf_message_priority_t priority

  Message priority.

- void(* p_callback)(sf_message_callback_args_t *)

  User callback function.

- void const * p_context

  Context provided to user during callback.

6.24.8.12 sf_message_api_t

sf_message_api_t

Detailed description

Messaging Framework API structure. Implementations will use the following API.

6.24.8.13 open

```
ssp_err_t(* sf_message_api_t::open) (sf_message_ctrl_t *const p_ctrl,
sf_message_cfg_t const *const p_cfg)
```

Detailed description

Initialize message framework. Initiate the messaging framework control block, configure the work memory corresponding to the configuration parameters. Implemented as

- SF_MESSAGE_Open

Table 383:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the messaging control block
p_cfg	in	Pointer to configuration structure

Parameter p_ctrl

Definition: `sf_message_ctrl_t*const p_ctrl`

Message framework control block. Allocate an instance specific control block to pass into the message framework API calls. Implemented assf_message_instance_ctrl_t

Parameter p_cfg

Definition: `sf_message_cfg_t const *const p_cfg`

Messaging framework configuration structure definition

- `sf_message_cfg_t`::`p_work_memory_start`

  Start address of the memory area.

- `sf_message_cfg_t`::`work_memory_size_bytes`

  Size of working memory area in bytes.

- `sf_message_cfg_t`::`buffer_size`

  Bytes of the message block.

- `sf_message_cfg_t`::`sf_message_subscriber_list_t`

  Pointer array to the subscriber lists.

- `sf_message_cfg_t`::`p_block_pool_name`

  Pointer to the block pool name.

6.24.8.14 close

```
ssp_err_t(* sf_message_api_t::close) (sf_message_ctrl_t *const p_ctrl)
```

Detailed description

Finalize message framework. Implemented as

- SF_MESSAGE_Close

Table 384:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the messaging control block

Parameter p_ctrl

Definition: `sf_message_ctrl_t*const p_ctrl`

Message framework control block. Allocate an instance specific control block to pass into the message framework API calls. Implemented assf_message_instance_ctrl_t

6.24.8.15 bufferAcquire

```
ssp_err_t(* sf_message_api_t::bufferAcquire) (sf_message_ctrl_t const *const
p_ctrl, sf_message_header_t **pp_buffer, sf_message_acquire_cfg_t const *const
p_acquire_cfg, uint32_t const wait_option)
```

Detailed description

Acquire buffer for message passing from the block. Implemented as

- SF_MESSAGE_BufferAcquire

Table 385:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the messaging control block
pp_buffer	inout	Pointer to the pointer to the allocated buffer memory
p_acquire_cfg	in	Pointer to the buffer acquisition configuration
wait_option	in	Wait option (TX_NO_WAIT, TX_WAIT_FOREVER or numerical values)

Parameter p_ctrl

Definition: `sf_message_ctrl_tconst *const p_ctrl`

Message framework control block. Allocate an instance specific control block to pass into the message framework API calls. Implemented assf_message_instance_ctrl_t

Parameter pp_buffer

Definition: sf_message_header_t**pp_buffer

Message header definition

- sf_message_header_t::event
- sf_message_header_t::class_code

  Event class code.
- sf_message_header_t::class_instance

  Event class instance number.
- sf_message_header_t::code

  Event code.
- sf_message_header_t::event_b
- sf_message_header_t::struct{}

Parameter p_acquire_cfg

Definition: sf_message_acquire_cfg_t const *const p_acquire_cfg

Messaging framework Post API function configuration structure definition

- sf_message_acquire_cfg_t::buffer_keep

  Buffer keep option.

Parameter wait_option

uint32_t

6.24.8.16 bufferRelease

```
ssp_err_t(* sf_message_api_t::bufferRelease) (sf_message_ctrl_t *const p_ctrl,
sf_message_header_t *const p_buffer, sf_message_release_option_t const option)
```

Detailed description

Release buffer obtained from SF_MESSAGE_BufferAcquire. Implemented as

- SF_MESSAGE_BufferRelease

Table 386:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the messaging control block
p_buffer	in	Pointer to the buffer allocated by SF_MESSAGE_BufferAcquire
option	in	Buffer release option (SF_MESSAGE_RELEASE_OPTION_NONE, SF_MESSAGE_RELEASE_OPTION_ACK, SF_MESSAGE_RELEASE_OPTION_NAK, SF_MESSAGE_RELEASE_OPTION_FORCED_RELEASE)

Parameter p_ctrl

Definition: `sf_message_ctrl_t*const p_ctrl`

Message framework control block. Allocate an instance specific control block to pass into the message framework API calls. Implemented assf_message_instance_ctrl_t

Parameter p_buffer

Definition: `sf_message_header_t*const p_buffer`

Message header definition

- `sf_message_header_t::event`
- `sf_message_header_t::class_code`

  Event class code.
- `sf_message_header_t::class_instance`

  Event class instance number.
- `sf_message_header_t::code`

  Event code.
- `sf_message_header_t::event_b`
- `sf_message_header_t::struct{}`

Parameter option

6.24.8.17 post

```
ssp_err_t(* sf_message_api_t::post) (sf_message_ctrl_t *const p_ctrl,
sf_message_header_t const *const p_buffer, sf_message_post_cfg_t const *const
p_post_cfg, sf_message_post_err_t *const p_post_err, uint32_t const wait_option)
```

Detailed description

Post message to the subscribers. Implemented as

- SF_MESSAGE_Post

Table 387:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the messaging control block
p_buffer	in	Pointer to the buffer allocated by SF_MESSAGE_BufferAcquire
p_post_cfg	in	Pointer to the message post configuration
wait_option	in	Wait option (TX_NO_WAIT, TX_WAIT_FOREVER or numerical values)

Parameter p_ctrl

Definition: `sf_message_ctrl_t*const p_ctrl`

Message framework control block. Allocate an instance specific control block to pass into the message framework API calls. Implemented assf_message_instance_ctrl_t

Parameter p_buffer

Definition: `sf_message_header_tconst *const p_buffer`

Message header definition

- `sf_message_header_t::event`
- `sf_message_header_t::class_code`

  Event class code.
- `sf_message_header_t::class_instance`

  Event class instance number.

- sf_message_header_t::code

  Event code.

- sf_message_header_t::event_b

- sf_message_header_t::struct{}

Parameter p_post_cfg

Definition: sf_message_post_cfg_t const *const p_post_cfg

Messaging framework Acquire API function configuration structure definition

- sf_message_post_cfg_t::sf_message_priority_t

  Message priority.

  Enumerated as:

  – SF_MESSAGE_PRIORITY_NORMAL

  – SF_MESSAGE_PRIORITY_HIGH

- sf_message_post_cfg_t::p_callback

  User callback function.

- sf_message_post_cfg_t::p_context

  Context provided to user during callback.

Parameter wait_option

uint32_t

6.24.8.18 pend

```
ssp_err_t(* sf_message_api_t::pend) (sf_message_ctrl_t const *const p_ctrl,
TX_QUEUE const *const p_queue, sf_message_header_t **pp_buffer, uint32_t const
wait_option)
```

Detailed description

Pend message. Implemented as

- SF_MESSAGE_Pend

Table 388:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the messaging control block

Table 388:Parameters (Continued)

Name	Direction	Description
p_queue	in	Pointer to a threadX message queue object
pp_buffer	inout	Pointer to the pointer to the buffer where message is stored.
wait_option	in	Wait option (TX_NO_WAIT, TX_WAIT_FOREVER or numerical values)

Parameter p_ctrl

Definition: `sf_message_ctrl_tconst *const p_ctrl`

Message framework control block. Allocate an instance specific control block to pass into the message framework API calls. Implemented assf_message_instance_ctrl_t

Parameter p_queue

`const`

Parameter pp_buffer

Definition: `sf_message_header_t**pp_buffer`

Message header definition

- `sf_message_header_t::event`
- `sf_message_header_t::class_code`

  Event class code.
- `sf_message_header_t::class_instance`

  Event class instance number.
- `sf_message_header_t::code`

  Event code.
- `sf_message_header_t::event_b`
- `sf_message_header_t::struct{}`

Parameter wait_option

`uint32_t`

6.24.8.19 versionGet

```
ssp_err_t(* sf_message_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Get the version of the messaging framework. Implemented as

- SF_MESSAGE_VersionGet

Table 389:Parameters

Name	Direction	Description
p_version	in	Pointer to the memory where to store the version number

Parameter p_version

6.24.8.20 sf_message_instance_t

sf_message_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_message_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_message_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_message_api_t const * p_api

  Pointer to the API structure for this instance.

6.25 Power Profiles Framework Interface

Power Profiles Framework Interface.

6.25.1 Summary

This framework allows an application to place itself in one of several available Low Power configurations. The application may make API calls that will place it into a low power sleep mode from which an external interrupt, or periodic RTC interrupt may awaken it.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Framework Power Profiles Interface description: Power Profiles Framework

6.25.2 Interface API

sf_power_profiles_api_t

Function name	Description
.open	Acquires mutex, then initialize the lower layer drivers at the HAL layer
.sleep	Places the MCU in Software Standby mode.
.close	Releases channel mutex and closes channel at HAL layer.
.versionGet	Gets version and stores it in provided pointer p_version.

6.25.3 Data structures

- sf_power_profiles_callback_args_t
- sf_power_profiles_ctrl_t
- sf_power_profiles_cfg_t
- sf_power_profiles_instance_t

6.25.4 Enumerations

- sf_power_profiles_mode_t
- sf_power_profiles_event_t

6.25.5 Defines

- #define SF_POWER_PROFILES_API_VERSION_MAJOR

  Initial value: `(1U)`

  Version of the API defined in this file
- #define SF_POWER_PROFILES_API_VERSION_MINOR

  Initial value: `(2U)`

6.25.6 API Data

6.25.6.1 sf_power_profiles_mode_t

`sf_power_profiles_mode_t`

Detailed description

Options for the callback events.

Enumerated values

Name	Description
SF_POWER_PROFILES_MODE_RUN	Normal startup mode (No sleep)
SF_POWER_PROFILES_MODE_RTC	Wakeup using RTC.
SF_POWER_PROFILES_MODE_EXTERNAL	Wakeup from an external source.

6.25.6.2 sf_power_profiles_event_t

`sf_power_profiles_event_t`

Detailed description

Options for the callback events.

Enumerated values

Name	Description
SF_POWER_PROFILES_EVENT_PRE_SLEEP	Callback just before entering sleep mode.
SF_POWER_PROFILES_EVENT_POST_SLEEP	Callback just after waking up.

6.25.7 API Structures

6.25.7.1 sf_power_profiles_callback_args_t

sf_power_profiles_callback_args_t

Detailed description

Power profiles callback arguments definitions

Variables

- sf_power_profiles_event_t event

  Power profiles callback event.

- void * p_context

  Placeholder for user data.

6.25.7.2 sf_power_profiles_ctrl_t

sf_power_profiles_ctrl_t

Detailed description

Channel control block. DO NOT INITIALIZE. Initialization occurs when SF_POWER_PROFILES_Open is called

Variables

- uint32_t open

  Used by driver to check if pointer to control block is valid.

- ioport_cfg_t const * p_wake_ioport_pin_tbl

  Pointer to ioport settings for wakeup.

- ioport_cfg_t const * p_sleep_ioport_pin_tbl

  Pointer to ioport settings for sleep.

- sf_power_profiles_mode_t operating_mode

  Power profile mode to use.

- lpm_api_t const * p_api_lpm

  Pointer to lower level Low Power driver function pointers.

- rtc_api_t const * p_api_rtc

  Pointer to lower level RTC driver function pointers.

- rtc_ctrl_t * p_ctrl_rtc

  Pointer to lower level RTC driver control block.

- void(* p_callback)(sf_power_profiles_callback_args_t *p_args)

  Callback function.

- void * p_context

  Placeholder for user data.

6.25.7.3 sf_power_profiles_cfg_t

sf_power_profiles_cfg_t

Detailed description

Configuration for RTOS integrated Power Profiles driver

Variables

- ioport_cfg_t const *const p_wake_ioport_pin_tbl

  Pointer to ioport settings for wakeup.

- ioport_cfg_t const *const p_sleep_ioport_pin_tbl

  Pointer to ioport settings for sleep.

- sf_power_profiles_mode_t operating_mode

  Power profile mode to use.

- bool retain_output_signals

  (Future implementation:) True (Default) ==> address bus and bus control signals retain the output state False ==> signals are set to high-impedance state

- lpm_instance_t const *const p_lower_lvl_lpm

  Pointer to the LPM instance.

- rtc_instance_t const *const p_lower_lvl_rtc

  Pointer to the RTC instance (if any)

- void(* p_callback)(sf_power_profiles_callback_args_t *p_args)

  Callback function.

- void * p_context

  Placeholder for user data.

6.25.7.4 sf_power_profiles_api_t

sf_power_profiles_api_t

Detailed description

Framework Power Profiles API structure. Implementations will use the following API.

6.25.7.5 open

```
ssp_err_t(* sf_power_profiles_api_t::open) (sf_power_profiles_ctrl_t *const
p_ctrl, sf_power_profiles_cfg_t const *const p_cfg)
```

Detailed description

Acquires mutex, then initialize the lower layer drivers at the HAL layer Implemented as

- SF_POWER_PROFILES_Open

Table 390:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a structure allocated by user. Elements initialized here.
p_cfg	in	Pointer to configuration structure. All elements of the structure must be set by user.

Parameter p_ctrl

Definition: `sf_power_profiles_ctrl_t`

Channel control block. DO NOT INITIALIZE. Initialization occurs when SF_POWER_PROFILES_Open is called

Parameter p_cfg

Definition: `sf_power_profiles_cfg_t const *const p_cfg`

Configuration for RTOS integrated Power Profiles driver

- `sf_power_profiles_cfg_t`::`ioport_cfg_t`

  Pointer to ioport settings for wakeup.

- `sf_power_profiles_cfg_t`::`ioport_cfg_t`

  Pointer to ioport settings for sleep.

- `sf_power_profiles_cfg_t`::`sf_power_profiles_mode_t`

  Power profile mode to use.

  Enumerated as:

  – SF_POWER_PROFILES_MODE_RUN

  – SF_POWER_PROFILES_MODE_RTC

  – SF_POWER_PROFILES_MODE_EXTERNAL

- `sf_power_profiles_cfg_t`::`retain_output_signals`

  (Future implementation:) True (Default) ==> address bus and bus control signals retain the output state False ==> signals are set to high-impedance state

- `sf_power_profiles_cfg_t`::`lpm_instance_t`

  Pointer to the LPM instance.

- `sf_power_profiles_cfg_t`::`rtc_instance_t`

  Pointer to the RTC instance (if any)

- sf_power_profiles_cfg_t::p_callback

  Callback function.

- sf_power_profiles_cfg_t::p_context

  Placeholder for user data.

6.25.7.6 sleep

```
ssp_err_t(* sf_power_profiles_api_t::sleep) (sf_power_profiles_ctrl_t *const
p_ctrl)
```

Detailed description

Places the MCU in Software Standby mode. Implemented as

- SF_POWER_PROFILES_Sleep

NOTE: If the Power Profiles operating mode has been set to SF_POWER_PROFILES_MODE_RTC, then the MCU is configured to enter a low power mode that also allows the RTC and it's associated clock to run. It is up to the application to configure the RTC such that it interrupts at a specified interval.

Table 391:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block set in SF_POWER_PROFILES_Open.

Parameter p_ctrl

Definition: sf_power_profiles_ctrl_t

Channel control block. DO NOT INITIALIZE. Initialization occurs when SF_POWER_PROFILES_Open is called

6.25.7.7 close

```
ssp_err_t(* sf_power_profiles_api_t::close) (sf_power_profiles_ctrl_t *const
p_ctrl)
```

Detailed description

Releases channel mutex and closes channel at HAL layer. Implemented as

- SF_POWER_PROFILES_Close

Table 392:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block set in SF_POWER_PROFILES_Open.

Parameter p_ctrl

Definition: `sf_power_profiles_ctrl_t`

Channel control block. DO NOT INITIALIZE. Initialization occurs when SF_POWER_PROFILES_Open is called

6.25.7.8 versionGet

```
ssp_err_t(* sf_power_profiles_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Gets version and stores it in provided pointer p_version.

Table 393:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

6.25.7.9 sf_power_profiles_instance_t

sf_power_profiles_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_power_profiles_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_power_profiles_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_power_profiles_api_t const * p_api

  Pointer to the API structure for this instance.

6.26 Power Profiles Framework Interface

Power Profiles Framework Interface.

6.26.1 Summary

This framework allows an application to apply power profiles at runtime. There are 2 types of profiles: Run and Low Power. Applying a Run profile will change things like the system clock and IOPORT settings. The MCU will continue to run during this process and will not be put into a low power mode. Applying a Low Power profile will put the MCU into a low power mode. Which low power mode is used is specified by the LPMv2 instance used. IOPORT settings can also be specified which will be applied before entering the low power mode and after waking up.

The Deep Software Standby low power mode, available on some MCUs, will reset the MCU when the woken up. In this case the callback will not be called and the IOPORT configuration will not be applied after waking up.

This framework can be used with, or without, an RTOS.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Framework Power Profiles Interface description: Power Profiles Framework

6.26.2 Interface API

sf_power_profiles_v2_api_t

Function name	Description
.open	Initializes the framework.
.runApply	Applies a Run profile.
.lowPowerApply	Applies a Low Power profile.
.close	Closes the framework.
.versionGet	Gets version and stores it in provided pointer p_version.

6.26.3 Data structures

- sf_power_profiles_v2_callback_args_t
- sf_power_profiles_v2_ctrl_t
- sf_power_profiles_v2_cfg_t

- sf_power_profiles_v2_run_cfg_t
- sf_power_profiles_v2_low_power_cfg_t
- sf_power_profiles_v2_instance_t

6.26.4 Enumerations

- sf_power_profiles_v2_event_t

6.26.5 Defines

- #define SF_POWER_PROFILES_V2_API_VERSION_MAJOR

  Initial value: `(2U)`

  Version of the API defined in this file
- #define SF_POWER_PROFILES_V2_API_VERSION_MINOR

  Initial value: `(1U)`

6.26.6 API Data

6.26.6.1 sf_power_profiles_v2_event_t

`sf_power_profiles_v2_event_t`

Detailed description

Options for the callback events.

Enumerated values

Name	Description
SF_POWER_PROFILES_V2_EVENT_PRE_LOW_POWER	Callback just before entering low power mode.
SF_POWER_PROFILES_V2_EVENT_POST_LOW_POWER	Callback just after exiting the low power mode.

6.26.7 API Structures

6.26.7.1 sf_power_profiles_v2_callback_args_t

sf_power_profiles_v2_callback_args_t

Detailed description

Power profiles callback arguments definitions

Variables

- sf_power_profiles_v2_event_t event

  Power profiles callback event.

- void * p_context

  Placeholder for user data.

6.26.7.2 sf_power_profiles_v2_ctrl_t

sf_power_profiles_v2_ctrl_t

Detailed description

Common control block. DO NOT INITIALIZE. Initialization occurs when SF_POWER_PROFILES_V2_Open is called

Variables

- uint32_t open

  Used by driver to check if pointer to control block is valid.

6.26.7.3 sf_power_profiles_v2_cfg_t

sf_power_profiles_v2_cfg_t

Detailed description

Initialization configuration

Variables

- void const * p_extend

  Pointer to additional settings (not currently in use)

6.26.7.4 sf_power_profiles_v2_run_cfg_t

sf_power_profiles_v2_run_cfg_t

Detailed description

Run profile configuration

Variables

- ioport_cfg_t const * p_ioport_pin_tbl

  Pointer to IOPORT settings

- cgc_clocks_cfg_t const * p_clock_cfg

  Pointer to a CGC configuration

- void const * p_extend

  Pointer to additional settings

6.26.7.5 sf_power_profiles_v2_low_power_cfg_t

sf_power_profiles_v2_low_power_cfg_t

Detailed description

Low Power profile configuration

Variables

- ioport_cfg_t const * p_ioport_pin_tbl_exit

  Pointer to IOPORT settings to apply after exiting the low power mode
- ioport_cfg_t const * p_ioport_pin_tbl_enter

  Pointer to IOPORT settings to apply before entering low power mode
- lpmv2_instance_t const * p_lower_lvl_lpm

  Pointer to an LPMv2 instance
- void(* p_callback)(sf_power_profiles_v2_callback_args_t *p_args)

  Callback function
- void * p_context

  Placeholder for user data
- void const * p_extend

  Pointer to additional settings

6.26.7.6 sf_power_profiles_v2_api_t

sf_power_profiles_v2_api_t

Detailed description

Framework Power Profiles v2 API structure. Implementations will use the following API.

6.26.7.7 open

```
ssp_err_t(* sf_power_profiles_v2_api_t::open) (sf_power_profiles_v2_ctrl_t *const
p_ctrl, sf_power_profiles_v2_cfg_t const *const p_cfg)
```

Detailed description

Initializes the framework. Implemented as

- SF_POWER_PROFILES_V2_Open

Table 394:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a structure allocated by user. Elements initialized here.

Table 394:Parameters (Continued)

Name	Direction	Description
p_cfg	in	Pointer to configuration structure. Elements of the structure must be set by user.

Parameter p_ctrl

Definition: sf_power_profiles_v2_ctrl_t

Common control block. DO NOT INITIALIZE. Initialization occurs when SF_POWER_PROFILES_V2_Open is called

Parameter p_cfg

Definition: sf_power_profiles_v2_cfg_t const *const p_cfg

Initialization configuration

- sf_power_profiles_v2_cfg_t::p_extend

  Pointer to additional settings (not currently in use)

6.26.7.8 runApply

```
ssp_err_t(* sf_power_profiles_v2_api_t::runApply) (sf_power_profiles_v2_ctrl_t
*const p_ctrl, sf_power_profiles_v2_run_cfg_t const *const p_cfg)
```

Detailed description

Applies a Run profile. Implemented as

- SF_POWER_PROFILES_V2_RunApply

Table 395:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block set in SF_POWER_PROFILES_V2_Open.
p_cfg	in	Pointer to configuration structure. Elements of the structure must be set by user.

Parameter p_ctrl

Definition: sf_power_profiles_v2_ctrl_t

Common control block. DO NOT INITIALIZE. Initialization occurs when SF_POWER_PROFILES_V2_Open is called

Parameter p_cfg

Definition: sf_power_profiles_v2_run_cfg_t const *const p_cfg

Run profile configuration

- sf_power_profiles_v2_run_cfg_t::ioport_cfg_t

  Pointer to IOPORT settings
- sf_power_profiles_v2_run_cfg_t::cgc_clocks_cfg_t

  Pointer to a CGC configuration
- sf_power_profiles_v2_run_cfg_t::p_extend

  Pointer to additional settings

6.26.7.9 lowPowerApply

```
ssp_err_t(* sf_power_profiles_v2_api_t::lowPowerApply)
(sf_power_profiles_v2_ctrl_t *const p_ctrl, sf_power_profiles_v2_low_power_cfg_t
const *const p_cfg)
```

Detailed description

Applies a Low Power profile. Implemented as

- SF_POWER_PROFILES_V2_LowPowerApply

Table 396:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block set in SF_POWER_PROFILES_V2_Open.
p_cfg	in	Pointer to configuration structure. Elements of the structure must be set by user.

Parameter p_ctrl

Definition: sf_power_profiles_v2_ctrl_t

Common control block. DO NOT INITIALIZE. Initialization occurs when SF_POWER_PROFILES_V2_Open is called

Parameter p_cfg

Definition: sf_power_profiles_v2_low_power_cfg_t const *const p_cfg

Low Power profile configuration

- sf_power_profiles_v2_low_power_cfg_t::ioport_cfg_t

  Pointer to IOPORT settings to apply after exiting the low power mode
- sf_power_profiles_v2_low_power_cfg_t::ioport_cfg_t

  Pointer to IOPORT settings to apply before entering low power mode
- sf_power_profiles_v2_low_power_cfg_t::lpmv2_instance_t

  Pointer to an LPMv2 instance

- sf_power_profiles_v2_low_power_cfg_t::p_callback

  Callback function

- sf_power_profiles_v2_low_power_cfg_t::p_context

  Placeholder for user data

- sf_power_profiles_v2_low_power_cfg_t::p_extend

  Pointer to additional settings

6.26.7.10 close

```
ssp_err_t(* sf_power_profiles_v2_api_t::close) (sf_power_profiles_v2_ctrl_t
*const p_ctrl)
```

Detailed description

Closes the framework. Implemented as

- SF_POWER_PROFILES_V2_Close

Table 397:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block set in SF_POWER_PROFILES_V2_Open.

Parameter p_ctrl

Definition: sf_power_profiles_v2_ctrl_t

Common control block. DO NOT INITIALIZE. Initialization occurs when SF_POWER_PROFILES_V2_Open is called

6.26.7.11 versionGet

```
ssp_err_t(* sf_power_profiles_v2_api_t::versionGet) (ssp_version_t *const
p_version)
```

Detailed description

Gets version and stores it in provided pointer p_version. Implemented as

- SF_POWER_PROFILES_V2_VersionGet

Table 398:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

6.26.7.12 sf_power_profiles_v2_instance_t

sf_power_profiles_v2_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_power_profiles_v2_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_power_profiles_v2_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_power_profiles_v2_api_t const * p_api

  Pointer to the API structure for this instance.

6.27 SF Socket WIFI Framework Interface

RTOS-integrated SF Socket WIFI Framework Interface.

6.27.1 Summary

This SSP Interface provides access OnChip stack BSD Socket API.

6.27.2 Functions

- socket
- close
- bind
- listen
- connect
- accept
- send
- recv
- sendto
- recvfrom
- setsockopt
- getsockopt
- select

6.27.3 Interface API

sf_socket_api_t

Function name	Description
.open	Pointer to function which initializes the network interface for data transfers. Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.
.close	Pointer to function which un-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.
.versionGet	Gets version and stores it in provided pointer p_version.

6.27.4 Data structures

- in_addr
- sockaddr
- sockaddr_in
- sf_socket_ctrl_t
- sf_socket_cfg_t
- sf_socket_instance_t

6.27.5 Typedefs

- socklen_t

6.27.6 Defines

- #define SF_SOCKET_WIFI_API_VER_MAJOR

  Initial value: `(1U)`

  Major Version of the API defined in this file
- #define SF_SOCKET_WIFI_API_VER_MINOR

  Initial value: `(0U)`

  Minor Version of the API defined in this file

6.27.7 socket

```
socket ( int domain , int type , int protocol )
```

6.27.7.1 Detailed description

API which creates socket

Table 399:Parameters

Name	Direction	Description
domain	in	Socket family
type	in	Socket type
protocol	in	Protocol type

6.27.8 close

```
close ( int socket_fd )
```

6.27.8.1 Detailed description

API which closes socket

Table 400:Parameters

Name	Direction	Description
socket_fd	in	Local socket

6.27.9 bind

```
bind ( int socket_fd , sockaddr const struct * p_local_sock_addr ,
socklen_t addrlen )
```

6.27.9.1 Detailed description

Bind socket to interface which is identified by IP address

Table 401:Parameters

Name	Direction	Description
socket_fd	in	Local socket
p_local_sock_addr	in	Pointer to local socket address
addrlen	in	Size of sock address structure

6.27.10 listen

```
listen (  int sockfd ,   int backlog )
```

6.27.10.1 Detailed description

Listen for tcp connection. Set socket in listen mode for tcp connection.

Table 402:Parameters

Name	Direction	Description
sockfd	in	Local socket
backlog	in	Max number of connection queue.

6.27.11 connect

```
connect (  int sockfd ,  sockaddr const struct * p_serv_addr ,
socklen_t  addrlen )
```

6.27.11.1 Detailed description

Establish TCP connection with remote socket

Table 403:Parameters

Name	Direction	Description
sockfd	in	Local socket
p_serv_addr	in	Pointer to remote socket address

Table 403:Parameters (Continued)

Name	Direction	Description
addrlen	in	Size of sock address structure

6.27.12 accept

```
accept (  int sockfd ,  sockaddr struct * p_cliaddr ,  socklen_t  * p_addrlen )
```

6.27.12.1 Detailed description

Accept connection request from remote.

Table 404:Parameters

Name	Direction	Description
sockfd	in	Local socket
p_cliaddr	out	Pointer to remote socket address which trying to connect
p_addrlen	out	Pointer to address length of client socket address

6.27.13 send

```
send (  int sockfd ,  const void * p_buf ,  size_t length ,  int flags )
```

6.27.13.1 Detailed description

Send data to remote socket.

Table 405:Parameters

Name	Direction	Description
sockfd	in	Local socket
p_buf	in	Pointer to data buffer
length	in	Data buffer length
flags	in	Socket flags

6.27.14 recv

```
recv ( int sockfd , void * p_buf , size_t length , int flags )
```

6.27.14.1 Detailed description

Receive data from remote socket.

Table 406:Parameters

Name	Direction	Description
sockfd	in	Local socket
p_buf	out	Pointer to data buffer where data will be received
length	in	Maximum length of data which can be received
flags	in	Socket flags

6.27.15 sendto

```
sendto ( int sockfd , const void * p_buf , size_t length , int flags ,
sockaddr const struct * p_dest_addr , socklen_t addrlen )
```

6.27.15.1 Detailed description

Send data to remote socket.

Table 407:Parameters

Name	Direction	Description
sockfd	in	Local socket
p_buf	in	Pointer to data buffer to sent
length	in	Data buffer length
flags	in	Socket flag
p_dest_addr	in	Pointer to remote socket address where to send data

Table 407:Parameters (Continued)

Name	Direction	Description
addrlen	in	Length of socket address structure

6.27.16 recvfrom

```
recvfrom (  int sockfd ,   void * p_buf ,   size_t length ,   int flags ,
sockaddr struct * p_src_addr ,  socklen_t  * p_addrlen )
```

6.27.16.1 Detailed description

Receive data from remote socket.

Table 408:Parameters

Name	Direction	Description
sockfd	in	Local socket
p_buf	out	Pointer to data buffer where data will be received
length	in	Maximum length of data which can be received
flags	in	Socket flag
p_src_addr	out	Pointer to remote socket address which has sent data
p_addrlen	out	Length of socket address structure

6.27.17 setsockopt

```
 setsockopt (  int sockfd ,   int level ,   int optname ,   const void
* p_optval ,  socklen_t  optlen )
```

6.27.17.1 Detailed description

Set Socket options.

Table 409:Parameters

Name	Direction	Description
sockfd	in	Local socket
level	in	Sockets API level
optname	in	Option to be set
p_optval	in	Option value to be set
optlen	in	Length of option value

6.27.18 getsockopt

```
getsockopt (  int sockfd ,   int level ,   int optname ,   void * p_optval ,
socklen_t  * p_optlen )
```

6.27.18.1 Detailed description

Get Socket options.

Table 410:Parameters

Name	Direction	Description
sockfd	in	Local socket
level	in	Sockets API level
optname	in	Option to be get
p_optval	out	Option value to be get
p_optlen	in	Length of option value

6.27.19 select

```
 select (  int nfds ,   fd_set * p_readfds ,   fd_set * p_writefds ,   fd_set
* p_exceptfds ,   struct timeval * p_timeout )
```

6.27.19.1 Detailed description

Wait on a given socket for specified amount of time. In case of any activity e.g. arrival of packet it comes out of wait.

Table 411:Parameters

Name	Direction	Description
nfds	in	Max fd
p_readfds	in	Pointer to fd_set to check whether data is available for read
p_writefds	in	Pointer to fd_set to check whether data is available for write
p_exceptfds	in	Pointer to fd_set to check whether exceptional condition occurred
p_timeout	in	Wait time in milliseconds

6.27.20 API Data

6.27.20.1 socklen_t

```
typedef int32_t socklen_t
```

Detailed description

Socket Structure Length

6.27.21 API Structures

6.27.21.1 in_addr

in_addr

Detailed description

IP address used by sockaddr

Socket Internet Address structure

Variables

- unsigned long s_addr

  load with inet_aton()

  Load with inet_aton()
- ULONG s_addr

6.27.21.2 sockaddr

sockaddr

Detailed description

Socket Address information

Socket Internet Address structure with port and family

Variables

- short sin_family

  Address family.
- unsigned short sin_port

  Port number.
- in_addrstruct sin_addr

  IP Address.
- char sin_zero[8]

  zero this if you want to

  Zero this if you want to.
- USHORT sa_family
- UCHAR sa_data[14]

6.27.21.3 sockaddr_in

sockaddr_in

Detailed description

Socket address, Internet style.

Variables

- uint16_t sin_family

  Internet Protocol (AF_INET)
- uint16_t sin_port

  Address port (16 bits)
- in_addrstruct sin_addr

  Internet address (32 bits)
- int8_t sin_zero[8]

  Not used.

  Not used structure member.
- USHORT sin_family
- USHORT sin_port

- CHAR sin_zero[8]

6.27.21.4 sf_socket_ctrl_t

sf_socket_ctrl_t

Detailed description

Socket Interface control structure

Variables

- sf_wifi_onchip_stack_instance_t * p_lower_lvl_onchip_wifi

  low level wifi interface

6.27.21.5 sf_socket_cfg_t

sf_socket_cfg_t

Detailed description

Socket Interface configuration structure

Variables

- sf_wifi_onchip_stack_instance_t * p_lower_lvl_onchip_wifi

  Pointer to SF on-chip stack instance.

- void * p_extend

  Extended configuration.

6.27.21.6 sf_socket_api_t

sf_socket_api_t

Detailed description

Socket Interface API

6.27.21.7 open

```
(* sf_socket_api_t::open) (sf_socket_ctrl_t *p_ctrl, sf_socket_cfg_t const *const
p_cfg)
```

Brief description

Pointer to function which initializes the network interface for data transfers.

Detailed description

Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.

Table 412:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to user-provided storage for the control block.
p_cfg	in	Pointer to configuration structure.

Parameter p_ctrl

Definition: `sf_socket_ctrl_t`

Socket Interface control structure

Parameter p_cfg

Definition: `sf_socket_cfg_t const *const p_cfg`

Socket Interface configuration structure

- `sf_socket_cfg_t`::`sf_wifi_onchip_stack_instance_t`

  Pointer to SF on-chip stack instance.

- `sf_socket_cfg_t`::`p_extend`

  Extended configuration.

6.27.21.8 close

```
(* sf_socket_api_t::close) (sf_socket_ctrl_t *const p_ctrl)
```

Brief description

Pointer to function which un-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.

Detailed description

Table 413:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the control block

Parameter p_ctrl

Definition: `sf_socket_ctrl_t`

Socket Interface control structure

6.27.21.9 versionGet

```
(* sf_socket_api_t::versionGet) ( *const p_version)
```

Brief description

Gets version and stores it in provided pointer p_version.

Detailed description

Table 414:Parameters

Name	Direction	Description
p_version	out	pointer to memory location to return version number

Parameter p_version

6.27.21.10 sf_socket_instance_t

sf_socket_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_socket_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_socket_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_socket_api_t const * p_api

  Pointer to the API structure for this instance.

6.28 SF Socket CELLULAR Framework Interface

RTOS-integrated SF Socket Cellular Framework Interface.

6.28.1 Summary

This SSP Interface provides access OnChip stack BSD Socket API.

6.28.2 Functions

- socket
- close
- bind
- listen
- connect
- accept
- send
- recv
- sendto
- recvfrom
- setsockopt
- getsockopt
- select

6.28.3 Interface API

sf_cellular_socket_api_t

Function name	Description
.open	Pointer to function which initializes the network interface for data transfers. Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.
.close	Pointer to function which un-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.
.versionGet	Gets version and stores it in provided pointer p_version.

6.28.4 Data structures

- in_addr
- sockaddr
- sockaddr_in
- sf_cellular_socket_ctrl_t

- sf_cellular_socket_cfg_t
- sf_cellular_socket_instance_t

6.28.5 Typedefs

- socklen_t

6.28.6 Defines

- #define SF_CELLULAR_SOCKET_API_VERSION_MAJOR

  Initial value: `(1U)`

  SF Cellular Socket APIs Major Version
- #define SF_CELLULAR_SOCKET_API_VERSION_MINOR

  Initial value: `(0U)`

  SF Cellular Socket APIs Minor Version

6.28.7 socket

```
socket (  int domain ,   int type ,   int protocol )
```

6.28.7.1 Detailed description

API which creates socket

Table 415:Parameters

Name	Direction	Description
domain	in	Socket family
type	in	Socket type
protocol	in	Protocol type

6.28.8 close

```
close (  int socket_fd )
```

6.28.8.1 Detailed description

API which closes socket

Table 416:Parameters

Name	Direction	Description
socket_fd	in	Local socket

6.28.9 bind

```
bind (  int socket_fd ,  sockaddr const struct * p_local_sock_addr ,
socklen_t  addrlen )
```

6.28.9.1 Detailed description

Bind socket to interface which is identified by IP address

Table 417:Parameters

Name	Direction	Description
socket_fd	in	Local socket
p_local_sock_addr	in	Pointer to local socket address
addrlen	in	Size of sock address structure

6.28.10 listen

```
listen (  int sockfd ,   int backlog )
```

6.28.10.1 Detailed description

Listen for tcp connection. Set socket in listen mode for tcp connection.

Table 418:Parameters

Name	Direction	Description
sockfd	in	Local socket
backlog	in	Max number of connection queue.

6.28.11 connect

```
connect ( int sockfd , sockaddr const struct * p_serv_addr ,
socklen_t addrlen )
```

6.28.11.1 Detailed description

Establish TCP connection with remote socket

Table 419:Parameters

Name	Direction	Description
sockfd	in	Local socket
p_serv_addr	in	Pointer to remote socket address
addrlen	in	Size of sock address structure

6.28.12 accept

```
accept ( int sockfd , sockaddr struct * p_cliaddr , socklen_t * p_addrlen )
```

6.28.12.1 Detailed description

Accept connection request from remote.

Table 420:Parameters

Name	Direction	Description
sockfd	in	Local socket
p_cliaddr	out	Pointer to remote socket address which trying to connect
p_addrlen	out	Pointer to address length of client socket address

6.28.13 send

```
send ( int sockfd , const void * p_buf , size_t length , int flags )
```

6.28.13.1 Detailed description

Send data to remote socket.

Table 421:Parameters

Name	Direction	Description
sockfd	in	Local socket
p_buf	in	Pointer to data buffer
length	in	Data buffer length
flags	in	Socket flags

6.28.14 recv

```
recv ( int sockfd , void * p_buf , size_t length , int flags )
```

6.28.14.1 Detailed description

Receive data from remote socket.

Table 422:Parameters

Name	Direction	Description
sockfd	in	Local socket
p_buf	out	Pointer to data buffer where data will be received
length	in	Maximum length of data which can be received
flags	in	Socket flags

6.28.15 sendto

```
sendto ( int sockfd , const void * p_buf , size_t length , int flags ,
sockaddr const struct * p_dest_addr , socklen_t addrlen )
```

6.28.15.1 Detailed description

Send data to remote socket.

Table 423:Parameters

Name	Direction	Description
sockfd	in	Local socket
p_buf	in	Pointer to data buffer to sent
length	in	Data buffer length
flags	in	Socket flag
p_dest_addr	in	Pointer to remote socket address where to send data
addrlen	in	Length of socket address structure

6.28.16 recvfrom

```
recvfrom (  int sockfd ,   void * p_buf ,   size_t length ,   int flags ,
sockaddr struct * p_src_addr ,  socklen_t  * p_addrlen )
```

6.28.16.1 Detailed description

Receive data from remote socket.

Table 424:Parameters

Name	Direction	Description
sockfd	in	Local socket
p_buf	out	Pointer to data buffer where data will be received
length	in	Maximum length of data which can be received
flags	in	Socket flag
p_src_addr	out	Pointer to remote socket address which has sent data

Table 424:Parameters (Continued)

Name	Direction	Description
p_addrlen	out	Length of socket address structure

6.28.17 setsockopt

```
 setsockopt (  int sockfd ,   int level ,   int optname ,   const void
* p_optval ,  socklen_t  optlen )
```

6.28.17.1 Detailed description

Set Socket options.

Table 425:Parameters

Name	Direction	Description
sockfd	in	Local socket
level	in	Sockets API level
optname	in	Option to be set
p_optval	in	Option value to be set
optlen	in	Length of option value

6.28.18 getsockopt

```
getsockopt (  int sockfd ,   int level ,   int optname ,   void * p_optval ,
socklen_t  * p_optlen )
```

6.28.18.1 Detailed description

Get Socket options.

Table 426:Parameters

Name	Direction	Description
sockfd	in	Local socket
level	in	Sockets API level

Table 426:Parameters (Continued)

Name	Direction	Description
optname	in	Option to be get
p_optval	out	Option value to be get
p_optlen	in	Length of option value

6.28.19 select

```
 select (  int nfds ,   fd_set * p_readfds ,   fd_set * p_writefds ,   fd_set
* p_exceptfds ,   struct timeval * p_timeout )
```

6.28.19.1 Detailed description

Wait on a given socket for specified amount of time. In case of any activity e.g. arrival of packet it comes out of wait.

Table 427:Parameters

Name	Direction	Description
nfds	in	Max fd
p_readfds	in	Pointer to fd_set to check whether data is available for read
p_writefds	in	Pointer to fd_set to check whether data is available for write
p_exceptfds	in	Pointer to fd_set to check whether exceptional condition occurred
p_timeout	in	Wait time in milliseconds

6.28.20 API Data

6.28.20.1 socklen_t

```
typedef int32_t socklen_t
```

Detailed description

Socket address Length

6.28.21 API Structures

6.28.21.1 in_addr

in_addr

Detailed description

IP address used by sockaddr

Socket Internet Address structure

Variables

- unsigned long s_addr

  load with inet_aton()

  Load with inet_aton()
- ULONG s_addr

6.28.21.2 sockaddr

sockaddr

Detailed description

Socket Address information

Socket Internet Address structure with port and family

Variables

- short sin_family

  Address family.
- unsigned short sin_port

  Port number.
- in_addrstruct sin_addr

  IP Address.
- char sin_zero[8]

  zero this if you want to

  Zero this if you want to.
- USHORT sa_family
- UCHAR sa_data[14]

6.28.21.3 sockaddr_in

sockaddr_in

Detailed description

Socket address, Internet style.

Variables

- uint16_t sin_family

  Internet Protocol (AF_INET)
- uint16_t sin_port

  Address port (16 bits)
- in_addrstruct sin_addr

  Internet address (32 bits)
- int8_t sin_zero[8]

  Not used.

  Not used structure member.
- USHORT sin_family
- USHORT sin_port
- CHAR sin_zero[8]

6.28.21.4 sf_cellular_socket_ctrl_t

sf_cellular_socket_ctrl_t

Detailed description

Socket Interface control structure

Variables

- sf_cellular_instance_t * p_lower_lvl_cellular

  low level cellular interface

6.28.21.5 sf_cellular_socket_cfg_t

sf_cellular_socket_cfg_t

Detailed description

Socket Interface configuration structure

Variables

- sf_cellular_instance_t * p_lower_lvl_cellular

  Pointer to SF on-chip stack instance.

- void * p_extend

  Extended configuration.

6.28.21.6 sf_cellular_socket_api_t

sf_cellular_socket_api_t

Detailed description

Socket Interface API

6.28.21.7 open

```
(* sf_cellular_socket_api_t::open) (sf_cellular_socket_ctrl_t *p_ctrl,
sf_cellular_socket_cfg_t const *const p_cfg)
```

Brief description

Pointer to function which initializes the network interface for data transfers.

Detailed description

Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.

Table 428:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the control block for the Cellular module Socket interface.
p_cfg	in	Pointer to Cellular Socket interface configuration structure.

Parameter p_ctrl

Definition: `sf_cellular_socket_ctrl_t`

Socket Interface control structure

Parameter p_cfg

Definition: `sf_cellular_socket_cfg_t const *const p_cfg`

Socket Interface configuration structure

- sf_cellular_socket_cfg_t::sf_cellular_instance_t

  Pointer to SF on-chip stack instance.

- sf_cellular_socket_cfg_t::p_extend

  Extended configuration.

6.28.21.8 close

```
(* sf_cellular_socket_api_t::close) (sf_cellular_socket_ctrl_t *const p_ctrl)
```

Brief description

Pointer to function which un-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.

Detailed description

Table 429:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the control block for the Cellular module Socket interface.

Parameter p_ctrl

Definition: sf_cellular_socket_ctrl_t

Socket Interface control structure

6.28.21.9 versionGet

```
(* sf_cellular_socket_api_t::versionGet) ( *const p_version)
```

Brief description

Gets version and stores it in provided pointer p_version.

Detailed description

Table 430:Parameters

Name	Direction	Description
p_version	out	Pointer to SSP Version structure

Parameter p_version

6.28.21.10 sf_cellular_socket_instance_t

sf_cellular_socket_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_cellular_socket_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_cellular_socket_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_cellular_socket_api_t const * p_api

  Pointer to the API structure for this instance.

6.29 SPI Framework Interface

RTOS-integrated SPI Framework Interface.

6.29.1 Summary

This SSP Interface provides access to the ThreadX-aware SPI Framework. The Interface is implemented by the SPI Framework.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

SPI Framework Interface description: SPI Framework

6.29.2 Interface API

sf_spi_api_t

Function name	Description
.open	Open a designated SPI device on a bus.
.read	Receive data from SPI device.
.write	Transmit data to SPI device.
.writeRead	Simultaneously transmit data to an SPI device while receiving data from an SPI device (full duplex). The writeread API gets mutex object, handles SPI data transmission at SPI HAL layer and receive data from the SPI HAL layer. The API uses the event flag wait to synchronize to completion of data transfer .
.close	Disable the SPI device designated by the control handle and close the RTOS services used by the bus if no devices are connected to the bus. This function removes power to the SPI channel designated by the handle and disables the associated interrupts.
.lock	Lock the bus for a device. The locking allows devices to reserve a bus to themselves for a given period of time (i.e. between lock and unlock). This allows devices to complete several reads and writes on the bus without interrupt.
.unlock	Unlock the bus for a particular device and make the bus usable for other devices.
.version	Get the version information of the underlying driver.

6.29.3 Data structures

- sf_spi_bus_t
- sf_spi_cfg_t
- sf_spi_instance_t

6.29.4 Enumerations

- sf_spi_dev_state_t

6.29.5 Typedefs

- sf_spi_ctrl_t

6.29.6 Defines

- #define SF_SPI_API_VERSION_MAJOR

  Initial value: `(1U)`
- #define SF_SPI_API_VERSION_MINOR

  Initial value: `(3U)`

6.29.7 API Data

6.29.7.1 sf_spi_dev_state_t

`sf_spi_dev_state_t`

Detailed description

SF SPI device state

Enumerated values

Name	Description
SF_SPI_DEV_STATE_CLOSED	SPI device is closed.
SF_SPI_DEV_STATE_OPENED	SPI device is opened.

6.29.7.2 sf_spi_ctrl_t

`typedef void sf_spi_ctrl_t`

Detailed description

SPI framework control block. Allocate an instance specific control block to pass into the SPI framework API calls. Implemented as

- sf_spi_instance_ctrl_t

6.29.8 API Structures

6.29.8.1 sf_spi_bus_t

sf_spi_bus_t

Detailed description

Data structure defining an SPI bus.

Variables

- uint8_t channel

  Channel.
- uint32_t freq_hz_min

  Bus min frequency supported.
- TX_MUTEX * p_lock_mutex

  Lock mutex handle for this channel.
- TX_EVENT_FLAGS_GROUP * p_sync_eventflag

  Pointer to the event flag object for SPI data transfer.
- sf_spi_ctrl_t ** pp_curr_ctrl

  Current device using the bus.
- uint8_t * p_bus_name

  peripheral name SCI_SPI/RSPI
- spi_api_t const * p_lower_lvl_api

  Pointer to SPI HAL interface to be used in the framework.
- uint8_t device_count

  Number of devices on the bus, initialize to 0.

6.29.8.2 sf_spi_cfg_t

sf_spi_cfg_t

Detailed description

Configuration for Framework SPI driver.

Variables

- sf_spi_bus_t * p_bus

  Bus used by the device.
- ioport_port_pin_t chip_select

  Chip select for this device.
- ioport_level_t chip_select_level_active

  Polarity of CS, active High or Low.
- spi_cfg_t const * p_lower_lvl_cfg

  Pointer to SPI HAL configuration.

6.29.8.3 sf_spi_api_t

sf_spi_api_t

Detailed description

Definition of the SPI framework interface.

6.29.8.4 open

```
ssp_err_t(* sf_spi_api_t::open) (sf_spi_ctrl_t *p_ctrl, sf_spi_cfg_t const *const p_cfg)
```

Brief description

Open a designated SPI device on a bus.

Detailed description

Table 431:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to user-provided storage for the control block.
p_cfg	in	Pointer to SPI Framework configuration structure.

Parameter p_ctrl

Definition: `sf_spi_ctrl_t*p_ctrl`

SPI framework control block. Allocate an instance specific control block to pass into the SPI framework API calls. Implemented assf_spi_instance_ctrl_t

Parameter p_cfg

Definition: `sf_spi_cfg_t const *const p_cfg`

Configuration for Framework SPI driver.

- `sf_spi_cfg_t`::`sf_spi_bus_t`

  Bus used by the device.

- `sf_spi_cfg_t`::`ioport_port_pin_t`

  Chip select for this device.

- `sf_spi_cfg_t`::`ioport_level_t`

  Polarity of CS, active High or Low.

- `sf_spi_cfg_t`::`spi_cfg_t`

  Pointer to SPI HAL configuration.

6.29.8.5 read

```
ssp_err_t(* sf_spi_api_t::read) (sf_spi_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout)
```

Brief description

Receive data from SPI device.

Detailed description

NOTE: Call open to configure the SPI device before using this function.

Table 432:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the device.
p_dest	out	Pointer to destination buffer into which data will be copied that is received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count.
length	in	Indicates the number of units of data to be transferred (unit size specified by the bit_width).
bit_width	in	Indicates data bit width to be transferred.
timeout	in	Timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

Parameter p_ctrl

Definition: `sf_spi_ctrl_t*const p_ctrl`

SPI framework control block. Allocate an instance specific control block to pass into the SPI framework API calls. Implemented assf_spi_instance_ctrl_t

Parameter p_dest

`const`

Parameter length

`uint32_t`

Parameter bit_width

Parameter timeout

`uint32_t`

6.29.8.6 write

```
ssp_err_t(* sf_spi_api_t::write) (sf_spi_ctrl_t *const p_ctrl, void *const p_src,
uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout)
```

Brief description

Transmit data to SPI device.

Detailed description

NOTE: Call open to configure the SPI device before using this function.

Table 433:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the device.
p_src	in	Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL.
length	in	Indicates the number of units of data to be transferred (unit size specified by the bit_width).
bit_width	in	Indicates data bit width to be transferred.
timeout	in	Timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

Parameter p_ctrl

Definition: `sf_spi_ctrl_t*const p_ctrl`

SPI framework control block. Allocate an instance specific control block to pass into the SPI framework API calls. Implemented assf_spi_instance_ctrl_t

Parameter p_src

`const`

Parameter length

```
uint32_t
```

Parameter bit_width
Parameter timeout

```
uint32_t
```

6.29.8.7 writeRead

```
ssp_err_t(* sf_spi_api_t::writeRead) (sf_spi_ctrl_t *const p_ctrl, void *const
p_src, void *const p_dest, uint32_t const length, spi_bit_width_t const bit_width,
uint32_t const timeout)
```

Brief description

Simultaneously transmit data to an SPI device while receiving data from an SPI device (full duplex).

Detailed description

The writeread API gets mutex object, handles SPI data transmission at SPI HAL layer and receive data from the SPI HAL layer. The API uses the event flag wait to synchronize to completion of data transfer .

NOTE: Call open to configure the SPI before using this function.

Table 434:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the channel.
p_src	in	Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL.
p_dest	out	Pointer to destination buffer into which data will be copied that is received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count.
length	in	Indicates the number of units of data to be transferred (unit size specified by the bit_width).

Table 434:Parameters (Continued)

Name	Direction	Description
bit_width	in	Indicates data bit width to be transferred.
timeout	in	Timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

Parameter p_ctrl

Definition: `sf_spi_ctrl_t*const p_ctrl`

SPI framework control block. Allocate an instance specific control block to pass into the SPI framework API calls. Implemented assf_spi_instance_ctrl_t

Parameter p_src

`const`

Parameter p_dest

`const`

Parameter length

`uint32_t`

Parameter bit_width
Parameter timeout

`uint32_t`

6.29.8.8 close

```
ssp_err_t(* sf_spi_api_t::close) (sf_spi_ctrl_t *const p_ctrl)
```

Brief description

Disable the SPI device designated by the control handle and close the RTOS services used by the bus if no devices are connected to the bus. This function removes power to the SPI channel designated by the handle and disables the associated interrupts.

Detailed description

Table 435:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the device.

Parameter p_ctrl

Definition: `sf_spi_ctrl_t*const p_ctrl`

SPI framework control block. Allocate an instance specific control block to pass into the SPI framework API calls. Implemented assf_spi_instance_ctrl_t

6.29.8.9 lock

```
ssp_err_t(* sf_spi_api_t::lock) (sf_spi_ctrl_t *const p_ctrl)
```

Brief description

Lock the bus for a device. The locking allows devices to reserve a bus to themselves for a given period of time (i.e. between lock and unlock). This allows devices to complete several reads and writes on the bus without interrupt.

Detailed description

Table 436:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the device.

Parameter p_ctrl

Definition: `sf_spi_ctrl_t*const p_ctrl`

SPI framework control block. Allocate an instance specific control block to pass into the SPI framework API calls. Implemented assf_spi_instance_ctrl_t

6.29.8.10 unlock

```
ssp_err_t(* sf_spi_api_t::unlock) (sf_spi_ctrl_t *const p_ctrl)
```

Brief description

Unlock the bus for a particular device and make the bus usable for other devices.

Detailed description

Table 437:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the device.

Parameter p_ctrl

Definition: `sf_spi_ctrl_t*const p_ctrl`

SPI framework control block. Allocate an instance specific control block to pass into the SPI framework API calls. Implemented assf_spi_instance_ctrl_t

6.29.8.11 version

```
ssp_err_t(* sf_spi_api_t::version) (ssp_version_t *const p_version)
```

Brief description

Get the version information of the underlying driver.

Detailed description

Table 438:Parameters

Name	Direction	Description
p_version	out	pointer to memory location to return version number

Parameter p_version

6.29.8.12 sf_spi_instance_t

sf_spi_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_spi_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- sf_spi_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- sf_spi_api_t const * p_api

  Pointer to the API structure for this instance.

6.30 Thread Monitor Framework Interface

RTOS-integrated Framework Interface for monitoring of threads.

Any misbehaving threads cause a reset of the device. Both the WDT and IWDT HAL modules are supported by this framework module.

6.30.1 Summary

This is a ThreadX aware Watchdog Timer Thread Monitor Framework for monitoring threads in an application in which threads are executing at an expected rate. Threads to be monitored register themselves through SF_THREAD_MONITOR_ThreadRegister and increment a count by calling

SF_THREAD_MONITOR_CountIncrement each time they execute. Each monitored thread also provides expected maximum and minimum count values for normal execution.

The Thread Monitor runs periodically and checks the count value of each monitored thread. If the count value falls outside of the expected range of values, the Watchdog Timer is allowed to reset the device. If all thread counts are within their expected ranges, then the Watchdog Timer is refreshed.

The WDT and IWDT modules are supported by the Thread Monitor.

The Framework Layer can be used to protect the entire software project. This is achieved through a high priority thread (Framework Layer) which runs periodically within the refresh permitted window of the Watchdog Timer selected (IWDT is safest as has its own clock source and is started automatically after reset). This thread monitors the state of every other thread in the system. If any of these threads are not running as expected, then the Watchdog Timer is not refreshed and is not allowed to reset the system. If the threads are running as expected, then the Watchdog Timer is refreshed.

Monitoring the other threads is achieved as follows: Each monitored thread increments a count variable each time it runs. The Thread Monitor thread checks the count variable of each thread to make sure it is within an expected range. If any of the variables are out of range a reset is allowed. Otherwise all variables are cleared to zero and the watchdog is refreshed. A profiling mode is used to establish the expected ranges.

This approach is described in the following article:

Jack Ganssle, "Great Watchdog Timers for Embedded Systems," www.ganssle.com/watchdogs.htm

This method requires the instrumenting of each thread to increment its count variable, but this is little overhead for the massive gain in protection.

Interface used: WDT Interface

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Thread Monitor Interface description: Thread Monitor Framework

6.30.2 Interface API

sf_thread_monitor_api_t

Function name	Description
.open	Configures the WDT or IWDT module. From the configuration data the timeout period of the WDT/IWDT is determined and a thread created monitoring registered threads.
.close	Suspends the thread monitoring thread. Watchdog peripheral no longer refreshed.
.threadRegister	Registers a thread for monitoring.
.threadUnregister	Removes a thread from being monitored.

Function name	Description
.countIncrement	Safely increments a monitored thread's count value.
.versionGet	Get version and store it in provided pointer p_version.

6.30.3 Data structures

- sf_thread_monitor_cfg_t
- sf_thread_monitor_thread_counter_t
- sf_thread_monitor_counter_min_max_t
- sf_thread_monitor_instance_t

6.30.4 Typedefs

- sf_thread_monitor_ctrl_t

6.30.5 Defines

- #define THREAD_MONITOR_THREAD_STACK_SIZE

  Initial value:`400u`
- #define SF_THREAD_MONITOR_API_VERSION_MAJOR

  Initial value: `(1U)`

  Version of the API defined in this file
- #define SF_THREAD_MONITOR_API_VERSION_MINOR

  Initial value: `(3U)`

6.30.6 API Data

6.30.6.1 sf_thread_monitor_ctrl_t

```
typedef void sf_thread_monitor_ctrl_t
```

Detailed description

Thread monitor control block. Allocate an instance specific control block to pass into the thread monitor API calls. Implemented as

- sf_thread_monitor_instance_ctrl_t

6.30.7 API Structures

6.30.7.1 sf_thread_monitor_cfg_t

sf_thread_monitor_cfg_t

Detailed description

Configuration for RTOS Thread Monitor driver

Variables

- wdt_instance_t const * p_lower_lvl_wdt

  Pointer to lower level watchdog instance.

- bool profiling_mode_enabled

  Enables or disables profiling mode.

- UINT priority

  Priority of thread monitor thread.

6.30.7.2 sf_thread_monitor_thread_counter_t

sf_thread_monitor_thread_counter_t

Detailed description

Counter block for each monitored thread.

Variables

- uint32_t current_count

  Current count value for a thread.

- uint32_t minimum_count

  Minimum expected count value. If the current count is less than this value the watchdog will reset.

- uint32_t maximum_count

  Maximum expected count value. If the current count is more than this value the watchdog will reset

- bool active

  Indicates to the monitoring thread whether this count data is currently active. When a thread is registered this value will be set to false as the count is likely to be a partial count and so should not be monitored. This value will be set to true by the thread monitor thread when it clears all counts to zero.

- TX_THREAD * p_thread

  Pointer to thread for this counter data.

6.30.7.3 sf_thread_monitor_counter_min_max_t

sf_thread_monitor_counter_min_max_t

Detailed description

Counter block for each monitored thread.

Variables

- uint32_t minimum_count

  Minimum expected count value. If the current count is less than this value the watchdog will reset.

- uint32_t maximum_count

  Maximum expected count value. If the current count is more than this value the watchdog will reset

6.30.7.4 sf_thread_monitor_api_t

sf_thread_monitor_api_t

Detailed description

Thread monitor API structure. Thread Monitor implementations use the following API.

6.30.7.5 open

```
(* sf_thread_monitor_api_t::open) (sf_thread_monitor_ctrl_t *const p_ctrl,
sf_thread_monitor_cfg_t const *const p_cfg)
```

Brief description

Configures the WDT or IWDT module. From the configuration data the timeout period of the WDT/IWDT is determined and a thread created monitoring registered threads.

Detailed description

Implemented as

- SF_THREAD_MONITOR_Open

Table 439:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a structure allocated by user. Elements initialized here.
p_cfg	in	Pointer to configuration structure. All elements of the structure must be set by user.

Parameter p_ctrl

Definition: `sf_thread_monitor_ctrl_t*const p_ctrl`

Thread monitor control block. Allocate an instance specific control block to pass into the thread monitor API calls. Implemented assf_thread_monitor_instance_ctrl_t

Parameter p_cfg

Definition: sf_thread_monitor_cfg_t const *const p_cfg

Configuration for RTOS Thread Monitor driver

- sf_thread_monitor_cfg_t::wdt_instance_t

  Pointer to lower level watchdog instance.

- sf_thread_monitor_cfg_t::profiling_mode_enabled

  Enables or disables profiling mode.

- sf_thread_monitor_cfg_t::priority

  Priority of thread monitor thread.

6.30.7.6 close

```
(* sf_thread_monitor_api_t::close) (sf_thread_monitor_ctrl_t *const p_ctrl)
```

Brief description

Suspends the thread monitoring thread. Watchdog peripheral no longer refreshed.

Detailed description

Implemented as

- SF_THREAD_MONITOR_Close

Table 440:Parameters

Name	Direction	Description
p_ctrl	inout	Control structure set in SF_THREAD_MONITOR_Open.

Parameter p_ctrl

Definition: sf_thread_monitor_ctrl_t*const p_ctrl

Thread monitor control block. Allocate an instance specific control block to pass into the thread monitor API calls. Implemented assf_thread_monitor_instance_ctrl_t

6.30.7.7 threadRegister

```
(* sf_thread_monitor_api_t::threadRegister) (sf_thread_monitor_ctrl_t *const
p_ctrl, sf_thread_monitor_counter_min_max_t const *p_counter_min_max)
```

Brief description

Registers a thread for monitoring.

Detailed description

Implemented as

- SF_THREAD_MONITOR_ThreadRegister

Table 441:Parameters

Name	Direction	Description
p_ctrl	inout	Control structure set in SF_THREAD_MONITOR_Open.
p_counter_min_max	in	Pointer to structure containing min and max values for thread to be registered values.

Parameter p_ctrl

Definition: `sf_thread_monitor_ctrl_t*const p_ctrl`

Thread monitor control block. Allocate an instance specific control block to pass into the thread monitor API calls. Implemented assf_thread_monitor_instance_ctrl_t

Parameter p_counter_min_max

Definition: `sf_thread_monitor_counter_min_max_tconst *p_counter_min_max`

Counter block for each monitored thread.

- `sf_thread_monitor_counter_min_max_t::minimum_count`

  Minimum expected count value. If the current count is less than this value the watchdog will reset.

- `sf_thread_monitor_counter_min_max_t::maximum_count`

  Maximum expected count value. If the current count is more than this value the watchdog will reset

6.30.7.8 threadUnregister

```
(* sf_thread_monitor_api_t::threadUnregister) (sf_thread_monitor_ctrl_t *const
p_ctrl)
```

Brief description

Removes a thread from being monitored.

Detailed description

Implemented as

- SF_THREAD_MONITOR_ThreadUnregister

Table 442:Parameters

Name	Direction	Description
p_ctrl	inout	Control structure set in SF_THREAD_MONITOR_Open.

Parameter p_ctrl

Definition: `sf_thread_monitor_ctrl_t*const p_ctrl`

Thread monitor control block. Allocate an instance specific control block to pass into the thread monitor API calls. Implemented assf_thread_monitor_instance_ctrl_t

6.30.7.9 countIncrement

```
(* sf_thread_monitor_api_t::countIncrement) (sf_thread_monitor_ctrl_t *const
p_ctrl)
```

Brief description

Safely increments a monitored thread's count value.

Detailed description

Implemented as

- SF_THREAD_MONITOR_CountIncrement

Table 443:Parameters

Name	Direction	Description
p_ctrl	inout	Control structure set in SF_THREAD_MONITOR_Open.

Parameter p_ctrl

Definition: `sf_thread_monitor_ctrl_t*const p_ctrl`

Thread monitor control block. Allocate an instance specific control block to pass into the thread monitor API calls. Implemented assf_thread_monitor_instance_ctrl_t

6.30.7.10 versionGet

```
(* sf_thread_monitor_api_t::versionGet) ( *const p_version)
```

Brief description

Get version and store it in provided pointer p_version.

Detailed description

Implemented as

- SF_THREAD_MONITOR_VersionGet

Table 444:Parameters

Name	Direction	Description
p_version	inout	Pointer to structure storing API and code versions.

Parameter p_version

6.30.7.11 sf_thread_monitor_instance_t

sf_thread_monitor_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_thread_monitor_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- sf_thread_monitor_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- sf_thread_monitor_api_t const * p_api

  Pointer to the API structure for this instance.

6.31 CTSU Framework Interface

RTOS-integrated CTSU Framework Interface.

6.31.1 Summary

This is a ThreadX-aware CTSU interface which is used to drive the CTSU HAL driver. It can be used to run the CTSU hardware, and read back the results of the scans.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

CTSU Framework Interface description: Capacitive Touch Framework

6.31.2 Interface API

sf_touch_ctsu_api_t

Function name	Description
.open	Initialize the Touch CTSU Framework; configures the lower level hardware also and run scans at the configured Update Frequency.
.read	Read the results from the CTSU including raw data, binary data and other derived data according to the selected options.
.close	Close the Framework by ending any scans in progress and destroying internal threads.
.versionGet	Get version and store it in provided pointer p_version.

6.31.3 Data structures

- sf_touch_ctsu_callback_args_t
- sf_touch_ctsu_cfg_t
- sf_touch_ctsu_instance_t

6.31.4 Typedefs

- sf_touch_ctsu_ctrl_t

6.31.5 Defines

- #define SF_TOUCH_CTSU_VERSION_MAJOR

  Initial value: `(1U)`
- #define SF_TOUCH_CTSU_VERSION_MINOR

  Initial value: `(1U)`

6.31.6 API Data

6.31.6.1 sf_touch_ctsu_ctrl_t

```
typedef void sf_touch_ctsu_ctrl_t
```

Detailed description

Capacitive touch framework control block. Allocate an instance specific control block to pass into the capacitive touch framework API calls. Implemented as

- sf_touch_ctsu_instance_ctrl_t

6.31.7 API Structures

6.31.7.1 sf_touch_ctsu_callback_args_t

sf_touch_ctsu_callback_args_t

Detailed description

Callback argument structure

Variables

- void * p_context

6.31.7.2 sf_touch_ctsu_cfg_t

sf_touch_ctsu_cfg_t

Detailed description

Configuration for RTOS integrated CTSU Touch framework.

Variables

- UINT priority

  Priority of the touch panel thread.
- uint16_t update_hz

  The frequency to run the scans. This is set by the latest open() call.
- void(* p_callback)(sf_touch_ctsu_callback_args_t *p_args)

  Callback function;.
- void * p_context

  Arguments to the callback.
- ctsu_instance_t * p_ctsu_instance

  CTSU instance.

6.31.7.3 sf_touch_ctsu_api_t

sf_touch_ctsu_api_t

Detailed description

CTSU Framework API structure. Implementations will use the following API.

6.31.7.4 open

```
(* sf_touch_ctsu_api_t::open) (sf_touch_ctsu_ctrl_t *const p_ctrl,
sf_touch_ctsu_cfg_t const *const p_cfg)
```

Detailed description

Initialize the Touch CTSU Framework; configures the lower level hardware also and run scans at the configured Update Frequency. Implemented as

- SF_TOUCH_CTSU_Open

Table 445:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
p_cfg	in	Pointer to configuration structure

Parameter p_ctrl

Definition: `sf_touch_ctsu_ctrl_t*const p_ctrl`

Capacitive touch framework control block. Allocate an instance specific control block to pass into the capacitive touch framework API calls. Implemented assf_touch_ctsu_instance_ctrl_t

Parameter p_cfg

Definition: `sf_touch_ctsu_cfg_t const *const p_cfg`

Configuration for RTOS integrated CTSU Touch framework.

- `sf_touch_ctsu_cfg_t`::`priority`

  Priority of the touch panel thread.

- `sf_touch_ctsu_cfg_t`::`update_hz`

  The frequency to run the scans. This is set by the latest open() call.

- `sf_touch_ctsu_cfg_t`::`p_callback`

  Callback function;.

- `sf_touch_ctsu_cfg_t`::`p_context`

  Arguments to the callback.

- `sf_touch_ctsu_cfg_t`::`ctsu_instance_t`

  CTSU instance.

6.31.7.5 read

```
(* sf_touch_ctsu_api_t::read) (sf_touch_ctsu_ctrl_t *const p_ctrl, void *p_dest,
opts, const *channels, const uint16_t count)
```

Detailed description

Read the results from the CTSU including raw data, binary data and other derived data according to the selected options. Implemented as

- SF_TOUCH_CTSU_Read

Table 446:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
p_dest	out	Pointer to the destination location for the read data
opts	in	Read options
channels	in	Specify the channel/channel pairs to read data for
channels	in	Specify the number of channel/channel pairs to read data for

Parameter p_ctrl

Definition: `sf_touch_ctsu_ctrl_t*const p_ctrl`

Capacitive touch framework control block. Allocate an instance specific control block to pass into the capacitive touch framework API calls. Implemented assf_touch_ctsu_instance_ctrl_t

Parameter p_dest

`const`

Parameter opts
Parameter channels
Parameter channels

6.31.7.6 close

```
(* sf_touch_ctsu_api_t::close) (sf_touch_ctsu_ctrl_t *const p_ctrl)
```

Detailed description

Close the Framework by ending any scans in progress and destroying internal threads. Implemented as

- SF_TOUCH_CTSU_Close

Table 447:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure

Parameter p_ctrl

Definition: `sf_touch_ctsu_ctrl_t*const p_ctrl`

Capacitive touch framework control block. Allocate an instance specific control block to pass into the capacitive touch framework API calls. Implemented assf_touch_ctsu_instance_ctrl_t

6.31.7.7 versionGet

`(* sf_touch_ctsu_api_t::versionGet) ( *const p_version)`

Detailed description

Get version and store it in provided pointer p_version. Implemented as

- SF_TOUCH_CTSU_VersionGet

Table 448:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

6.31.7.8 sf_touch_ctsu_instance_t

sf_touch_ctsu_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_touch_ctsu_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- sf_touch_ctsu_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- sf_touch_ctsu_api_t const * p_api

  Pointer to the API structure for this instance.

6.32 CTSU Button Framework Interface

RTOS-integrated CTSU Button Framework Interface.

This framework layer uses the CTSU Framework Layer to implement a button interface. Using this Button Framework the user can configure and use multiple buttons using the configuration structure generated from WorkBench. An action on each button (press, release etc) will result in a callback with an argument indicating the button id and event type.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

CTSU Button Framework Interface description: Capacitive Touch Button Framework

6.32.1 Interface API

sf_touch_ctsu_button_api_t

Function name	Description
.open	Initialize the Touch CTSU Button Framework; configures the lower level hardware and registers callback functions for all the buttons.
.enable	Enable callback notification for a configured button.
.disable	Disable callback notification for a configured button.
.close	Close the Touch CTSU Button Framework; Closes the button framework and lower layers if no other modules are using it.
.versionGet	Get version and store it in provided pointer p_version.

6.32.2 Data structures

- sf_touch_ctsu_button_callback_args_t
- sf_touch_ctsu_button_individual_t
- sf_touch_ctsu_button_cfg_t
- sf_touch_ctsu_button_hdl
- sf_touch_ctsu_button_instance_t

6.32.3 Enumerations

- sf_touch_button_state_t

6.32.4 Typedefs

- sf_touch_ctsu_button_id

- sf_touch_ctsu_button_ctrl_t

6.32.5 Defines

- #define SF_TOUCH_CTSU_BUTTON_VERSION_MAJOR

  Initial value: `(1U)`

  Version Number of API.
- #define SF_TOUCH_CTSU_BUTTON_VERSION_MINOR

  Initial value: `(1U)`

6.32.6 API Data

6.32.6.1 sf_touch_button_state_t

`sf_touch_button_state_t`

Detailed description

Button States.

Enumerated values

Name	Description
TOUCH_BUTTON_STATE_RELEASED	Button is in the released state.
TOUCH_BUTTON_STATE_PRESSED	Button is in the pressed state.
TOUCH_BUTTON_STATE_LONG_HOLD	Button is pressed down for a long time (duration in sf_touch_ctsu_button_config.h)
TOUCH_BUTTON_STATE_SHORT_HOLD	Button is pressed down for a short time (duration in sf_touch_ctsu_button_config.h.
TOUCH_BUTTON_STATE_STUCK	Button is stuck.
TOUCH_BUTTON_STATE_INITIAL	Button has been initialized successfully.
TOUCH_BUTTON_STATE_CLOSING	Button has been disabled and will no longer generate events.
TOUCH_BUTTON_STATE_MULTI_TOUCH	More than one touch element is being touched.
TOUCH_BUTTON_STATE_DISABLED	Button is disabled from being updated.

6.32.6.2 sf_touch_ctsu_button_id

```
typedef uint32_t sf_touch_ctsu_button_id
```

Detailed description

Unique identifier used for each button

6.32.6.3 sf_touch_ctsu_button_ctrl_t

```
typedef void sf_touch_ctsu_button_ctrl_t
```

Detailed description

Capacitive touch button framework control block. Allocate an instance specific control block to pass into the capacitive touch button framework API calls. Implemented as

- sf_touch_ctsu_button_instance_ctrl_t

6.32.7 API Structures

6.32.7.1 sf_touch_ctsu_button_callback_args_t

sf_touch_ctsu_button_callback_args_t

Detailed description

Button callback arguments definitions

Variables

- sf_touch_ctsu_button_id id

  Unique id for button.

- sf_touch_button_state_t event

  Button callback event.

- void const * p_context

  Placeholder for user data.

6.32.7.2 sf_touch_ctsu_button_individual_t

sf_touch_ctsu_button_individual_t

Detailed description

Definition of a button and it's behavior in software.

Variables

- ctsu_channel_pair_t button_channel

  Define the channel/channel pairs which make up a button.
- uint8_t release_enable

  enable release events
- uint8_t press_enable

  enable press events
- uint8_t repeat_enable

  enable repeat events
- uint8_t shorthold_enable

  enable short hold events
- uint8_t longhold_enable

  enable long hold events
- uint8_t byte

  Byte representation of events enabled.
- union{} event_enable

  See source code for definition of this member.
- uint16_t debounce_threshold

  Number of consecutive times a button is determined as touched before state changes.
- sf_touch_ctsu_button_id id

  Unique id for button.

6.32.7.3 sf_touch_ctsu_button_cfg_t

sf_touch_ctsu_button_cfg_t

Detailed description

Button Configuration Structure

Variables

- sf_touch_ctsu_instance_t const *const p_lower_lvl_touch_framework

  Pointer to the Touch Framework instance.
- uint32_t button_count

  Number of buttons in configuration.
- sf_touch_ctsu_button_individual_t ** pp_button_cfgs

  Pointer to button configurations.

- void(* p_callback)(sf_touch_ctsu_button_callback_args_t *p_args)

  Callback function.
- void const * p_context

  Placeholder for user data.
- void const * p_extend

  Extension parameter for instance specific settings.

6.32.7.4 sf_touch_ctsu_button_hdl

sf_touch_ctsu_button_hdl

Detailed description

Handle for each button

Variables

- sf_touch_ctsu_button_individual_t button_cfg

  Individual button configuration.
- sf_touch_button_state_t state

  Represents the current state of the button.
- int16_t offset

  Offset in the results array and bit offset in the binary.
- sf_touch_button_state_t prev_state

  Holds previous state of the button.
- uint32_t debounce_counter

  Debounce counter.
- uint32_t active_event_counter

  Amount of time for which button is spending between states.
- uint32_t press_down_counter

  Time for which button is held down.
- uint32_t open

  Indicate that button has been opened.

6.32.7.5 sf_touch_ctsu_button_api_t

sf_touch_ctsu_button_api_t

Detailed description

Touch CTSU Button Framework API structure. Implementations will use the following API.

6.32.7.6 open

```
ssp_err_t(* sf_touch_ctsu_button_api_t::open) (sf_touch_ctsu_button_ctrl_t *const p_ctrl, sf_touch_ctsu_button_cfg_t const *const p_cfg)
```

Detailed description

Initialize the Touch CTSU Button Framework; configures the lower level hardware and registers callback functions for all the buttons. Implemented as

- SF_TOUCH_CTSU_Button_Open

Table 449:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
p_cfg	in	Pointer to configuration structure

Parameter p_ctrl

Definition: `sf_touch_ctsu_button_ctrl_t*const p_ctrl`

Capacitive touch button framework control block. Allocate an instance specific control block to pass into the capacitive touch button framework API calls. Implemented assf_touch_ctsu_button_instance_ctrl_t

Parameter p_cfg

Definition: `sf_touch_ctsu_button_cfg_t const *const p_cfg`

Button Configuration Structure

- `sf_touch_ctsu_button_cfg_t`::`sf_touch_ctsu_instance_t`

  Pointer to the Touch Framework instance.

- `sf_touch_ctsu_button_cfg_t`::`button_count`

  Number of buttons in configuration.

- `sf_touch_ctsu_button_cfg_t`::`sf_touch_ctsu_button_individual_t`

  Pointer to button configurations.

- `sf_touch_ctsu_button_cfg_t`::`p_callback`

  Callback function.

- `sf_touch_ctsu_button_cfg_t`::`p_context`

  Placeholder for user data.

- sf_touch_ctsu_button_cfg_t::p_extend

  Extension parameter for instance specific settings.

6.32.7.7 enable

```
ssp_err_t(* sf_touch_ctsu_button_api_t::enable) (sf_touch_ctsu_button_ctrl_t
*const p_ctrl, sf_touch_ctsu_button_id const button_id)
```

Detailed description

Enable callback notification for a configured button. Implemented as

- SF_TOUCH_CTSU_Button_Enable

Table 450:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
button_id	in	Unique identifier for a button

Parameter p_ctrl

Definition: `sf_touch_ctsu_button_ctrl_t*const p_ctrl`

Capacitive touch button framework control block. Allocate an instance specific control block to pass into the capacitive touch button framework API calls. Implemented assf_touch_ctsu_button_instance_ctrl_t

Parameter button_id

`const`

6.32.7.8 disable

```
ssp_err_t(* sf_touch_ctsu_button_api_t::disable) (sf_touch_ctsu_button_ctrl_t
*const p_ctrl, sf_touch_ctsu_button_id const button_id)
```

Detailed description

Disable callback notification for a configured button. Implemented as

- SF_TOUCH_CTSU_Button_Disable

Table 451:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure

Table 451:Parameters (Continued)

Name	Direction	Description
button_id	in	Unique identifier for a button

Parameter p_ctrl

Definition: `sf_touch_ctsu_button_ctrl_t*const p_ctrl`

Capacitive touch button framework control block. Allocate an instance specific control block to pass into the capacitive touch button framework API calls. Implemented assf_touch_ctsu_button_instance_ctrl_t

Parameter button_id

`const`

6.32.7.9 close

```
ssp_err_t(* sf_touch_ctsu_button_api_t::close) (sf_touch_ctsu_button_ctrl_t
*const p_ctrl)
```

Detailed description

Close the Touch CTSU Button Framework; Closes the button framework and lower layers if no other modules are using it. Implemented as

- SF_TOUCH_CTSU_Button_Close

Table 452:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure

Parameter p_ctrl

Definition: `sf_touch_ctsu_button_ctrl_t*const p_ctrl`

Capacitive touch button framework control block. Allocate an instance specific control block to pass into the capacitive touch button framework API calls. Implemented assf_touch_ctsu_button_instance_ctrl_t

6.32.7.10 versionGet

```
ssp_err_t(* sf_touch_ctsu_button_api_t::versionGet) (ssp_version_t *const
p_version)
```

Detailed description

Get version and store it in provided pointer p_version. Implemented as

- SF_TOUCH_CTSU_Button_VersionGet

Table 453:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

6.32.7.11 sf_touch_ctsu_button_instance_t

sf_touch_ctsu_button_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_touch_ctsu_button_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- sf_touch_ctsu_button_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- sf_touch_ctsu_button_api_t const * p_api

  Pointer to the API structure for this instance.

6.33 CTSU Slider Framework Interface

RTOS-integrated CTSU Slider Framework Interface.

This framework layer uses the CTSU Framework Layer to implement a slider or wheel interface. Using this Slider Framework the user can configure and use multiple sliders/wheels using the configuration structure generated from WorkBench. An action on each slider/wheel (press, release etc) will result in a callback for that slider with an argument indicating the event type.

Related SSP architecture topics:

- What is an SSP Interface? SSP Interfaces
- What is an SSP Layer? SSP Predefined Layers
- How to use SSP Interfaces and Modules? Using SSP Modules

CTSU Slider Framework Interface description: Capacitive Touch Slider Framework

6.33.1 Interface API

sf_touch_ctsu_slider_api_t

Function name	Description
.open	Initialize the Touch CTSU Slider Framework, configures the lower level hardware and registers callback functions for all the sliders.
.enable	Enable callback notification for a configured slider.
.disable	Disable callback notification for a configured slider.
.close	Close the Touch CTSU Slider Framework; Closes the slider framework and lower layers if no other modules are using it.
.versionGet	Get version and store it in provided pointer p_version.

6.33.2 Data structures

- sf_touch_ctsu_slider_callback_args_t
- sf_touch_ctsu_slider_cfg_t
- sf_touch_ctsu_slider_instance_t

6.33.3 Enumerations

- sf_touch_ctsu_slider_state_t
- sf_slider_type_t

6.33.4 Typedefs

- sf_touch_ctsu_slider_id_t
- sf_touch_ctsu_slider_ctrl_t

6.33.5 Defines

- #define SF_TOUCH_CTSU_SLIDER_VERSION_MAJOR

  Initial value: `(1U)`

  Version Number of API.

- #define SF_TOUCH_CTSU_SLIDER_VERSION_MINOR

  Initial value: `(1U)`

6.33.6 API Data

6.33.6.1 sf_touch_ctsu_slider_state_t

`sf_touch_ctsu_slider_state_t`

Detailed description

Slider States

Enumerated values

Name	Description
SF_TOUCH_CTSU_SLIDER_STATE_NO_CHANGE	Slider is in the released state
SF_TOUCH_CTSU_SLIDER_STATE_INITIALIZED	Slider is in the pressed state
SF_TOUCH_CTSU_SLIDER_STATE_TOUCHED	Slider is pressed
SF_TOUCH_CTSU_SLIDER_STATE_RELEASED	Slider is released
SF_TOUCH_CTSU_SLIDER_STATE_CLOSED	Slider has been disabled and will no longer generate events.
SF_TOUCH_CTSU_SLIDER_STATE_MULTI_TOUCH	More than one touch element is being touched
SF_TOUCH_CTSU_SLIDER_STATE_DISABLED	Slider is disabled from being updated.
SF_TOUCH_CTSU_SLIDER_STATE_HELD	Slider is held. (Continued press)

6.33.6.2 sf_slider_type_t

`sf_slider_type_t`

Detailed description

Slider type definition

Enumerated values

Name	Description
SF_SLIDER_TYPE_LINEAR	The slider is of a linear type

Name	Description
SF_SLIDER_TYPE_CIRCULAR	The slider is of a circular type (wheel)

6.33.6.3 sf_touch_ctsu_slider_id_t

```
typedef uint32_t sf_touch_ctsu_slider_id_t
```

Detailed description

Unique identifier used for each slider

6.33.6.4 sf_touch_ctsu_slider_ctrl_t

```
typedef void sf_touch_ctsu_slider_ctrl_t
```

Detailed description

Capacitive touch slider framework control block. Allocate an instance specific control block to pass into the capacitive touch slider framework API calls. Implemented as

- sf_touch_ctsu_slider_instance_ctrl_t

6.33.7 API Structures

6.33.7.1 sf_touch_ctsu_slider_callback_args_t

sf_touch_ctsu_slider_callback_args_t

Detailed description

Slider callback arguments definitions

Variables

- sf_touch_ctsu_slider_id_t id

  Unique slider identifier.
- uint32_t event

  Slider callback event.
- uint32_t current_position

  Current slider position.
- void const * p_context

  Placeholder for user data.

6.33.7.2 sf_touch_ctsu_slider_cfg_t

sf_touch_ctsu_slider_cfg_t

Detailed description

Slider Configuration Structure

Variables

- sf_touch_ctsu_instance_t * p_lower_lvl_touch_framework

  Pointer to the Touch Framework instance

- uint32_t slider_count

  Number of sliders in configuration

- void(* p_callback)(sf_touch_ctsu_slider_callback_args_t *p_args)

  Callback function

- void const * p_context

  Placeholder for user data

- void const * p_extend

  Extension parameter for instance specific settings.

6.33.7.3 sf_touch_ctsu_slider_api_t

sf_touch_ctsu_slider_api_t

Detailed description

Touch CTSU Slider Framework API structure. Implementations will use the following API.

6.33.7.4 open

```
ssp_err_t(* sf_touch_ctsu_slider_api_t::open) (sf_touch_ctsu_slider_ctrl_t *const p_ctrl, sf_touch_ctsu_slider_cfg_t const *const p_cfg)
```

Detailed description

Initialize the Touch CTSU Slider Framework, configures the lower level hardware and registers callback functions for all the sliders. Implemented as

- SF_TOUCH_CTSU_Slider_Open

Table 454:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
p_cfg	in	Pointer to configuration structure

Parameter p_ctrl

Definition: `sf_touch_ctsu_slider_ctrl_t*const p_ctrl`

Capacitive touch slider framework control block. Allocate an instance specific control block to pass into the capacitive touch slider framework API calls. Implemented assf_touch_ctsu_slider_instance_ctrl_t

Parameter p_cfg

Definition: sf_touch_ctsu_slider_cfg_t const *const p_cfg

Slider Configuration Structure

- sf_touch_ctsu_slider_cfg_t::sf_touch_ctsu_instance_t

  Pointer to the Touch Framework instance

- sf_touch_ctsu_slider_cfg_t::slider_count

  Number of sliders in configuration

- sf_touch_ctsu_slider_cfg_t::p_callback

  Callback function

- sf_touch_ctsu_slider_cfg_t::p_context

  Placeholder for user data

- sf_touch_ctsu_slider_cfg_t::p_extend

  Extension parameter for instance specific settings.

6.33.7.5 enable

```
ssp_err_t(* sf_touch_ctsu_slider_api_t::enable) (sf_touch_ctsu_slider_ctrl_t
*const p_ctrl, sf_touch_ctsu_slider_id_t const slider_id)
```

Detailed description

Enable callback notification for a configured slider. Implemented as

- SF_TOUCH_CTSU_Slider_Enable

Table 455:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
slider_id	in	Unique identifier for a slider

Parameter p_ctrl

Definition: sf_touch_ctsu_slider_ctrl_t*const p_ctrl

Capacitive touch slider framework control block. Allocate an instance specific control block to pass into the capacitive touch slider framework API calls. Implemented assf_touch_ctsu_slider_instance_ctrl_t

Parameter slider_id

6.33.7.6 disable

```
ssp_err_t(* sf_touch_ctsu_slider_api_t::disable) (sf_touch_ctsu_slider_ctrl_t
*const p_ctrl, sf_touch_ctsu_slider_id_t const slider_id)
```

Detailed description

Disable callback notification for a configured slider. Implemented as

- SF_TOUCH_CTSU_Slider_Disable

Table 456:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
slider_id	in	Unique identifier for a slider

Parameter p_ctrl

Definition: `sf_touch_ctsu_slider_ctrl_t*const p_ctrl`

Capacitive touch slider framework control block. Allocate an instance specific control block to pass into the capacitive touch slider framework API calls. Implemented assf_touch_ctsu_slider_instance_ctrl_t

Parameter slider_id

6.33.7.7 close

```
ssp_err_t(* sf_touch_ctsu_slider_api_t::close) (sf_touch_ctsu_slider_ctrl_t
*const p_ctrl)
```

Detailed description

Close the Touch CTSU Slider Framework; Closes the slider framework and lower layers if no other modules are using it. Implemented as

- SF_TOUCH_CTSU_Slider_Close

Table 457:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure

Parameter p_ctrl

Definition: `sf_touch_ctsu_slider_ctrl_t*const p_ctrl`

Capacitive touch slider framework control block. Allocate an instance specific control block to pass into the capacitive touch slider framework API calls. Implemented assf_touch_ctsu_slider_instance_ctrl_t

6.33.7.8 versionGet

```
ssp_err_t(* sf_touch_ctsu_slider_api_t::versionGet) (ssp_version_t *const
p_version)
```

Detailed description

Get version and store it in provided pointer p_version. Implemented as

- SF_TOUCH_CTSU_Slider_VersionGet

Table 458:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

6.33.7.9 sf_touch_ctsu_slider_instance_t

sf_touch_ctsu_slider_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_touch_ctsu_slider_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- sf_touch_ctsu_slider_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- sf_touch_ctsu_slider_api_t const * p_api

  Pointer to the API structure for this instance.

6.34 Touch Panel Framework Interface

RTOS-integrated Touch Panel Framework Interface.

6.34.1 Summary

This module is a ThreadX-aware Touch Panel Framework which scans for touch events and posts them to the Messaging Framework for distribution to touch event subscribers. This Interface is implemented by I2C Touch Panel Framework.

Interfaces used:

- External IRQ Framework Interface
- I2C Master Interface
- Messaging Framework Interface

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Touch Panel Framework Interface description: Touch Panel Framework

6.34.2 Interface API

sf_touch_panel_api_t

Function name	Description
.open	Create required RTOS objects, call lower level module for hardware specific initialization, and create a thread to post touch data to a message queue.
.calibrate	Begin calibration routine based on provided expected coordinates. Returns SSP_SUCCESS only if the tolerance is longer than the distance from the expected touch point to the actual touch point (using the formula below): p_calibrate->tolerance_pixels ^ 2 > (p_calibrate->x - x_measured) ^ 2 + (p_calibrate->y - y_measured) ^ 2
.start	Start scanning for touch events.
.stop	Stop scanning for touch events.
.reset	Reset touch controller if reset pin is provided, and resets the I2C bus.
.close	Terminate touch thread and close channel at HAL layer.
.versionGet	Gets version and stores it in provided pointer p_version.

6.34.3 Data structures

- sf_touch_panel_payload_t
- sf_touch_panel_cfg_t
- sf_touch_panel_calibrate_t
- sf_touch_panel_instance_t

6.34.4 Enumerations

- sf_touch_panel_event_t

6.34.5 Typedefs

- sf_touch_panel_ctrl_t

6.34.6 Defines

- #define SF_TOUCH_PANEL_API_VERSION_MAJOR

  Initial value: `(1U)`
- #define SF_TOUCH_PANEL_API_VERSION_MINOR

  Initial value: `(2U)`
- #define SF_TOUCH_PANEL_MESSAGE_WORDS

  Initial value: `((sizeof(sf_touch_panel_payload_t) + 3) / 4)`

  Touch panel message size in 4 byte words, rounded up.
- #define SF_TOUCH_PANEL_MAX_MESSAGES

  Initial value: `(4)`

  Maximum number of messages in touch panel message queue.
- #define SF_TOUCH_BYTES_PER_WORD

  Initial value: `(4)`

  Macro defining the number of bytes per word.
- #define SF_TOUCH_PANEL_MESSAGE_MEM_BYTES

  Initial value: `(#define SF_TOUCH_PANEL_MESSAGE_WORDS * #define SF_TOUCH_BYTES_PER_WORD \ * #define SF_TOUCH_PANEL_MAX_MESSAGES)`

  Touch message queue memory size.

6.34.7 API Data

6.34.7.1 sf_touch_panel_event_t

`sf_touch_panel_event_t`

Detailed description

Touch event list.

Enumerated values

Name	Description
SF_TOUCH_PANEL_EVENT_INVALID	Invalid touch data.
SF_TOUCH_PANEL_EVENT_HOLD	Touch has not moved since last touch event.
SF_TOUCH_PANEL_EVENT_MOVE	Touch has moved since last touch event.
SF_TOUCH_PANEL_EVENT_DOWN	New touch event reported.
SF_TOUCH_PANEL_EVENT_UP	Touch released.
SF_TOUCH_PANEL_EVENT_NONE	No valid touch event happened.

6.34.7.2 sf_touch_panel_ctrl_t

```
typedef void sf_touch_panel_ctrl_t
```

Detailed description

Touch panel framework control block. Allocate an instance specific control block to pass into the touch panel framework API calls. Implemented as

- sf_touch_panel_i2c_instance_ctrl_t

6.34.8 API Structures

6.34.8.1 sf_touch_panel_payload_t

sf_touch_panel_payload_t

Detailed description

Touch data payload posted to the message queue.

Variables

- sf_message_header_t header

  Required header for messaging framework.
- int16_t x

  X coordinate.
- int16_t y

  Y coordinate.
- sf_touch_panel_event_t event_type

  Touch event type.

6.34.8.2 sf_touch_panel_cfg_t

sf_touch_panel_cfg_t

Detailed description

Configuration for RTOS integrated touch panel framework.

Variables

- uint16_t hsize_pixels

  Horizontal size of screen in pixels.

- uint16_t vsize_pixels

  Vertical size of screen in pixels.

- UINT priority

  Priority of the touch panel thread.

- sf_message_instance_t const * p_message

  Pointer to messaging framework control block.

- uint8_t event_class_instance

  Event class instance number for posting touch event class messages.

- uint16_t update_hz

  The frequency to report repeat (SF_TOUCH_PANEL_EVENT_DOWN or SF_TOUCH_PANEL_EVENT_HOLD) touch events in Hertz. This will be converted to RTOS ticks in the driver and rounded up to the nearest integer value of RTOS ticks.

- uint16_t rotation_angle

  Touch coordinate rotation angle(0/90/180/270)

- void const * p_extend

  Pointer to hardware specific extension. See sf_touch_panel_<instance>.h.

6.34.8.3 sf_touch_panel_calibrate_t

sf_touch_panel_calibrate_t

Detailed description

Calibration data passed to SF_TOUCH_PANEL_Calibrate.

Variables

- uint16_t x

  Expected x coordinate.

- uint16_t y

  Expected y coordinate.

- uint16_t tolerance_pixels

  Acceptable linear deviation from the expected coordinates in pixels.

- void const * p_extend

  Pointer to hardware specific extension. See sf_touch_panel_<instance>_cfg_t in sf_touch_panel_<instance>.h.

6.34.8.4 sf_touch_panel_api_t

sf_touch_panel_api_t

Detailed description

Touch panel API structure. Touch panel implementations use the following API.

6.34.8.5 open

```
ssp_err_t(* sf_touch_panel_api_t::open) (sf_touch_panel_ctrl_t *const p_ctrl,
sf_touch_panel_cfg_t const *const p_cfg)
```

Detailed description

Create required RTOS objects, call lower level module for hardware specific initialization, and create a thread to post touch data to a message queue. Implemented as

- SF_TOUCH_PANEL_I2C_Open

Table 459:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to a structure allocated by user. This control structure is initialized in this function.
p_cfg	in	Pointer to configuration structure. All elements of the structure must be set by user.

Parameter p_ctrl

Definition: `sf_touch_panel_ctrl_t*const p_ctrl`

Touch panel framework control block. Allocate an instance specific control block to pass into the touch panel framework API calls. Implemented assf_touch_panel_i2c_instance_ctrl_t

Parameter p_cfg

Definition: `sf_touch_panel_cfg_t const *const p_cfg`

Configuration for RTOS integrated touch panel framework.

- `sf_touch_panel_cfg_t`::`hsize_pixels`

  Horizontal size of screen in pixels.

- sf_touch_panel_cfg_t::vsize_pixels

  Vertical size of screen in pixels.

- sf_touch_panel_cfg_t::priority

  Priority of the touch panel thread.

- sf_touch_panel_cfg_t::sf_message_instance_t

  Pointer to messaging framework control block.

- sf_touch_panel_cfg_t::event_class_instance

  Event class instance number for posting touch event class messages.

- sf_touch_panel_cfg_t::update_hz

  The frequency to report repeat (SF_TOUCH_PANEL_EVENT_DOWN or SF_TOUCH_PANEL_EVENT_HOLD) touch events in Hertz. This will be converted to RTOS ticks in the driver and rounded up to the nearest integer value of RTOS ticks.

- sf_touch_panel_cfg_t::rotation_angle

  Touch coordinate rotation angle(0/90/180/270)

- sf_touch_panel_cfg_t::p_extend

  Pointer to hardware specific extension. See sf_touch_panel_<instance>.h.

6.34.8.6 calibrate

```
ssp_err_t(* sf_touch_panel_api_t::calibrate) (sf_touch_panel_ctrl_t *const
p_ctrl, sf_touch_panel_calibrate_t const *const p_expected,
sf_touch_panel_payload_t const *const p_actual, ULONG timeout)
```

Detailed description

Begin calibration routine based on provided expected coordinates. Returns SSP_SUCCESS only if the tolerance is longer than the distance from the expected touch point to the actual touch point (using the formula below): p_calibrate->tolerance_pixels ^ 2 > (p_calibrate->x - x_measured) ^ 2 + (p_calibrate->y - y_measured) ^ 2 Implemented as

- SF_TOUCH_PANEL_I2C_Calibrate

Table 460:Parameters

Name	Direction	Description
p_ctrl	in	Handle set in touch_panel_api_t::open.
p_expected	in	Expected coordinates and tolerance for passing.

Table 460:Parameters (Continued)

Name	Direction	Description
p_actual	in	Pointer to message payload received from SF_MESSAGE_EVENT_CLASS_TOUCH event class.
timeout	in	ThreadX timeout. Select TX_NO_WAIT, a value in system clock counts between 1 and 0xFFFFFFFF, or TX_WAIT_FOREVER.

Parameter p_ctrl

Definition: `sf_touch_panel_ctrl_t*const p_ctrl`

Touch panel framework control block. Allocate an instance specific control block to pass into the touch panel framework API calls. Implemented assf_touch_panel_i2c_instance_ctrl_t

Parameter p_expected

Definition: `sf_touch_panel_calibrate_tconst *const p_expected`

Calibration data passed to SF_TOUCH_PANEL_Calibrate.

- `sf_touch_panel_calibrate_t::x`

  Expected x coordinate.

- `sf_touch_panel_calibrate_t::y`

  Expected y coordinate.

- `sf_touch_panel_calibrate_t::tolerance_pixels`

  Acceptable linear deviation from the expected coordinates in pixels.

- `sf_touch_panel_calibrate_t::p_extend`

  Pointer to hardware specific extension. See sf_touch_panel_<instance>_cfg_t in sf_touch_panel_<instance>.h.

Parameter p_actual

Definition: `sf_touch_panel_payload_tconst *const p_actual`

Touch data payload posted to the message queue.

- `sf_touch_panel_payload_t::header`

  Required header for messaging framework.

- `sf_touch_panel_payload_t::x`

  X coordinate.

- `sf_touch_panel_payload_t::y`

  Y coordinate.

- sf_touch_panel_payload_t::event_type

  Touch event type.

Parameter timeout

const

6.34.8.7 start

```
ssp_err_t(* sf_touch_panel_api_t::start) (sf_touch_panel_ctrl_t *const p_ctrl)
```

Brief description

Start scanning for touch events.

Detailed description

Implemented as

- SF_TOUCH_PANEL_I2C_Start

Table 461:Parameters

Name	Direction	Description
p_ctrl	in	Handle set in touch_panel_api_t::open.

Parameter p_ctrl

Definition: sf_touch_panel_ctrl_t*const p_ctrl

Touch panel framework control block. Allocate an instance specific control block to pass into the touch panel framework API calls. Implemented assf_touch_panel_i2c_instance_ctrl_t

6.34.8.8 stop

```
ssp_err_t(* sf_touch_panel_api_t::stop) (sf_touch_panel_ctrl_t *const p_ctrl)
```

Brief description

Stop scanning for touch events.

Detailed description

Implemented as

- SF_TOUCH_PANEL_I2C_Stop

Table 462:Parameters

Name	Direction	Description
p_ctrl	in	Handle set in touch_panel_api_t::open.

Parameter p_ctrl

Definition: `sf_touch_panel_ctrl_t*const p_ctrl`

Touch panel framework control block. Allocate an instance specific control block to pass into the touch panel framework API calls. Implemented assf_touch_panel_i2c_instance_ctrl_t

6.34.8.9 reset

```
ssp_err_t(* sf_touch_panel_api_t::reset) (sf_touch_panel_ctrl_t *const p_ctrl)
```

Brief description

Reset touch controller if reset pin is provided, and resets the I2C bus.

Detailed description

Implemented as

- SF_TOUCH_PANEL_I2C_Reset

NOTE: This does not include calibration. Use calibrate from the application after this function if calibration is required after reset.

Table 463:Parameters

Name	Direction	Description
p_ctrl	in	Handle set in touch_panel_api_t::open.

Parameter p_ctrl

Definition: `sf_touch_panel_ctrl_t*const p_ctrl`

Touch panel framework control block. Allocate an instance specific control block to pass into the touch panel framework API calls. Implemented assf_touch_panel_i2c_instance_ctrl_t

6.34.8.10 close

```
ssp_err_t(* sf_touch_panel_api_t::close) (sf_touch_panel_ctrl_t *const p_ctrl)
```

Brief description

Terminate touch thread and close channel at HAL layer.

Detailed description

Implemented as

- SF_TOUCH_PANEL_I2C_Close

Table 464:Parameters

Name	Direction	Description
p_ctrl	in	Handle set in touch_panel_api_t::open.

Parameter p_ctrl

Definition: `sf_touch_panel_ctrl_t*const p_ctrl`

Touch panel framework control block. Allocate an instance specific control block to pass into the touch panel framework API calls. Implemented assf_touch_panel_i2c_instance_ctrl_t

6.34.8.11 versionGet

```
ssp_err_t(* sf_touch_panel_api_t::versionGet) (ssp_version_t *const p_version)
```

Brief description

Gets version and stores it in provided pointer p_version.

Detailed description

Implemented as

- SF_TOUCH_PANEL_I2C_VersionGet

Table 465:Parameters

Name	Direction	Description
p_version	out	Code and API version used stored here.

Parameter p_version

6.34.8.12 sf_touch_panel_instance_t

sf_touch_panel_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_touch_panel_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_touch_panel_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_touch_panel_api_t const * p_api

  Pointer to the API structure for this instance.

6.35 SF WIFI Framework Interface

RTOS-integrated SF WIFI Framework Interface.

6.35.1 Summary

This SSP Interface provides access to the ThreadX-aware SF WIFI Framework.

6.35.2 Interface API

sf_wifi_api_t

Function name	Description
.open	Initializes the network interface for data transfers. Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.
.close	De-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.
.multicastListAdd	Add the given MAC address to the multicast filter list.
.multicastListDelete	Delete the given MAC address from the multicast filter list.
.statisticsGet	Get the interface statistics.
.transmit	Transmit data packet.
.provisioningSet	Set WiFi module provisioning which will configure the module in AP or Client mode.
.provisioningGet	Get the provisioning information of WiFi module.

Function name	Description
.infoGet	Get WiFi module information.
.scan	Scan for WiFi SSIDs.
.ACLAdd	Adds a MAC address to the Access Control List. Valid in AP mode only.
.ACLDelete	Deletes a MAC address from Access Control List. Valid in AP mode only.
.macAddressGet	Get WiFi module MAC address.
.macAddressSet	Set WiFi module MAC address.
.versionGet	Gets version and stores it in provided pointer p_version.

6.35.3 Data structures

- sf_wifi_ip_addr_t
- sf_wifi_info_t
- sf_wifi_callback_args_t
- sf_wifi_provisioning_t
- sf_wifi_cfg_t
- sf_wifi_stats_t
- sf_wifi_scan_t
- sf_wifi_ctrl_t
- sf_wifi_instance_t

6.35.4 Enumerations

- sf_wifi_ip_addr_version_t
- sf_wifi_interface_mode_t
- sf_wifi_wep_key_format_t
- sf_wifi_security_type_t
- sf_wifi_encryption_type_t
- sf_wifi_bss_type_t
- sf_wifi_interface_hw_mode_t
- sf_wifi_rts_t

- sf_wifi_preamble_t
- sf_wifi_wmm_t
- sf_wifi_high_throughput_t
- sf_wifi_ssid_broadcast_t
- sf_wifi_wds_t
- sf_wifi_mandatory_high_throughput_t
- sf_wifi_auto_negotiation_t
- sf_wifi_access_control_t
- sf_wifi_event_t

6.35.5 Defines

- #define SF_WIFI_API_VERSION_MAJOR

  Initial value: `(1U)`

  Major Version of the API defined in this file
- #define SF_WIFI_API_VERSION_MINOR

  Initial value: `(1U)`

  Minor Version of the API defined in this file
- #define SF_WIFI_SSID_LENGTH

  Initial value: `(32U)`

  WiFi SSID length.
- #define SF_WIFI_SECURITY_KEY_LENGTH

  Initial value: `(128U)`

  WiFi Security Key length.
- #define SF_WIFI_MAC_ADDR_LENGTH

  Initial value: `(6U)`

  WiFi MAC address length.
- #define SF_WIFI_TRUE

  Initial value: `(1U)`

  Boolean True condition.
- #define SF_WIFI_FALSE

  Initial value: `(0U)`

  Boolean False condition.

- #define SF_WIFI_IPV4_ADDRESS

  Initial value: `((((uint32_t) a) << (24U)) | (((uint32_t) b) << (16U)) | \ (((uint32_t) c) << (8U)) | ((uint32_t) d))`

  IP Address Generation Macro

6.35.6 API Data

6.35.6.1 sf_wifi_ip_addr_version_t

`sf_wifi_ip_addr_version_t`

Detailed description

IP address version

Enumerated values

Name	Description
SF_WIFI_IP_ADDR_VERSION_4	IPv4 address.
SF_WIFI_IP_ADDR_VERSION_6	IPv6 address.

6.35.6.2 sf_wifi_interface_mode_t

`sf_wifi_interface_mode_t`

Detailed description

WiFi Interface mode

Enumerated values

Name	Description
SF_WIFI_INTERFACE_MODE_AP	Access Point mode.
SF_WIFI_INTERFACE_MODE_CLIENT	Station Mode.

6.35.6.3 sf_wifi_wep_key_format_t

`sf_wifi_wep_key_format_t`

Detailed description

WiFi WEP Key Format

Enumerated values

Name	Description
SF_WIFI_WEP_KEY_FORMAT_ASCII	WEP Key in ASCII.
SF_WIFI_WEP_KEY_FORMAT_HEX	WEP Key in Hex.

6.35.6.4 sf_wifi_security_type_t

`sf_wifi_security_type_t`

Detailed description

WiFi Security type

Enumerated values

Name	Description
SF_WIFI_SECURITY_TYPE_OPEN	Open. No encryption used.
SF_WIFI_SECURITY_TYPE_WEP	128-bit WEP OPEN ASCII
SF_WIFI_SECURITY_TYPE_WPA	WiFi Protected Access.
SF_WIFI_SECURITY_TYPE_WPA2	WiFi Protected Access v2.

6.35.6.5 sf_wifi_encryption_type_t

`sf_wifi_encryption_type_t`

Detailed description

WiFi Encryption type

Enumerated values

Name	Description
SF_WIFI_ENCRYPTION_TYPE_AUTO	Automatic selection of encryption protocol.
SF_WIFI_ENCRYPTION_TYPE_TKIP	Temporal Key Integrity Protocol. Used by WPA.
SF_WIFI_ENCRYPTION_TYPE_CCMP	CTR mode with CBC-MAC Protocol. Used by WPA2.
SF_WIFI_ENCRYPTION_TYPE_WEP	WEP mode. Used by WEP.
SF_WIFI_ENCRYPTION_TYPE_NONE	No Encryption. Used by Open Security type.

6.35.6.6 sf_wifi_bss_type_t

`sf_wifi_bss_type_t`

Detailed description

WiFi BSS type

Enumerated values

Name	Description
SF_WIFI_BSS_TYPE_INFRASTRUCTURE	Infrastructure network.
SF_WIFI_BSS_TYPE_ADHOC	802.11 ad-hoc IBSS network
SF_WIFI_BSS_TYPE_ANY	Either infrastructure or ad-hoc network.
SF_WIFI_BSS_TYPE_UNKNOWN	BSS type is unknown.

6.35.6.7 sf_wifi_interface_hw_mode_t

`sf_wifi_interface_hw_mode_t`

Detailed description

WiFi Hardware mode

Enumerated values

Name	Description
SF_WIFI_INTERFACE_HW_MODE_11A	802.11a
SF_WIFI_INTERFACE_HW_MODE_11B	802.11b
SF_WIFI_INTERFACE_HW_MODE_11G	802.11g
SF_WIFI_INTERFACE_HW_MODE_11N	802.11n

6.35.6.8 sf_wifi_rts_t

`sf_wifi_rts_t`

Detailed description

WiFi RTS flag

Enumerated values

Name	Description
SF_WIFI_RTS_DISABLE	Disable RTS/CTS handshake.
SF_WIFI_RTS_ENABLE	Enable RTS/CTS handshake.

6.35.6.9 sf_wifi_preamble_t

`sf_wifi_preamble_t`

Detailed description

WiFi Preamble type

Enumerated values

Name	Description
SF_WIFI_PREAMBLE_SHORT	Use short preamble.
SF_WIFI_PREAMBLE_LONG	Use long preamble.

6.35.6.10 sf_wifi_wmm_t

`sf_wifi_wmm_t`

Detailed description

WiFi WMM flag

Enumerated values

Name	Description
SF_WIFI_WMM_DISABLE	Disable WMM.
SF_WIFI_WMM_ENABLE	Enable WMM.

6.35.6.11 sf_wifi_high_throughput_t

`sf_wifi_high_throughput_t`

Detailed description

WiFi High Throughput flag

Enumerated values

Name	Description
SF_WIFI_HIGH_THROUGHPUT_DISABLE	Disable high throughput mode.
SF_WIFI_HIGH_THROUGHPUT_ENABLE	Enable high throughput mode. Also requires WMM to be enabled.

6.35.6.12 sf_wifi_ssid_broadcast_t

`sf_wifi_ssid_broadcast_t`

Detailed description

WiFi SSID Broadcast flag

Enumerated values

Name	Description
SF_WIFI_SSID_BROADCAST_DISABLE	Disable SSID Broadcast.
SF_WIFI_SSID_BROADCAST_ENABLE	Enable SSID Broadcast.

6.35.6.13 sf_wifi_wds_t

`sf_wifi_wds_t`

Detailed description

WiFi WDS Flasg

Enumerated values

Name	Description
SF_WIFI_WDS_DISABLE	Disable WDS.
SF_WIFI_WDS_ENABLE	Enable WDS.

6.35.6.14 sf_wifi_mandatory_high_throughput_t

`sf_wifi_mandatory_high_throughput_t`

Detailed description

WiFi Mandatory High Throughput flag

Enumerated values

Name	Description
SF_WIFI_MANDATORY_HIGH_THROUGHPUT_DISABLE	Disable Mandatory HT requirement.
SF_WIFI_MANDATORY_HIGH_THROUGHPUT_ENABLE	Enable mandatory HT requirement.

6.35.6.15 sf_wifi_auto_negotiation_t

`sf_wifi_auto_negotiation_t`

Detailed description

WiFi Auto Negotiation flag

Enumerated values

Name	Description
SF_WIFI_AUTO_NEGOTIATION_DISABLE	Auto negotiation disable.
SF_WIFI_AUTO_NEGOTIATION_ENABLE	Auto negotiation enable.

6.35.6.16 sf_wifi_access_control_t

`sf_wifi_access_control_t`

Detailed description

WiFi Framework AccessContol mode

Enumerated values

Name	Description
SF_WIFI_ACCESS_CONTROL_DISABLE	Disable MAC address matching.
SF_WIFI_ACCESS_CONTROL_DENY	Deny association to stations on the MAC list.
SF_WIFI_ACCESS_CONTROL_ALLOW	Allow association to stations on the MAC list.

6.35.6.17 sf_wifi_event_t

`sf_wifi_event_t`

Detailed description

WiFi Framework event codes

Enumerated values

Name	Description
SF_WIFI_EVENT_RX	Packet received event.
SF_WIFI_EVENT_AP_CONNECT	Device Associated Successfully with AP.
SF_WIFI_EVENT_AP_DISCONNECT	Device Disconnected with AP.
SF_WIFI_EVENT_CLIENT_CONNECT	Client Associated Successfully with device AP.
SF_WIFI_EVENT_CLIENT_DISCONNECT	Client Disconnected from device AP.

6.35.7 API Structures

6.35.7.1 sf_wifi_ip_addr_t

sf_wifi_ip_addr_t

Detailed description

IP address information

Variables

- sf_wifi_ip_addr_version_t version

  IP Address Version : v4 or v6.
- uint32_t v4
- uint32_t v6[4]
- union{} addr

  IP address.

  See source code for definition of this member.

6.35.7.2 sf_wifi_info_t

sf_wifi_info_t

Detailed description

Configuration about underlying device driver.

Variables

- uint8_t * p_chipset

  Pointer to sting showing WiFi chipset/driver information.

- uint16_t rssi

  Received signal strength indication.
- uint16_t noise_level

  Signal to noise ratio.
- uint16_t link_quality

  Signal strength / quality.

6.35.7.3 sf_wifi_callback_args_t

sf_wifi_callback_args_t

Detailed description

WiFi framework callback parameter definition

Variables

- sf_wifi_event_t event

  Event code.
- uint8_t * p_data

  Packet data.
- uint32_t length

  Packet Data length.
- uint8_t mac_addr[SF_WIFI_MAC_ADDR_LENGTH]

  Client station MAC address.
- void const * p_context

  Context provided to user during callback.

6.35.7.4 sf_wifi_provisioning_t

sf_wifi_provisioning_t

Detailed description

WiFi Provisioning parameters

Variables

- sf_wifi_interface_mode_t mode

  Select AP or Client mode.
- uint8_t channel

  RF Channel number.
- uint8_t ssid[SF_WIFI_SSID_LENGTH+1]

  SSID.

- sf_wifi_security_type_t security

  Security type.
- sf_wifi_encryption_type_t encryption

  Encryption type.
- uint8_t key[SF_WIFI_SECURITY_KEY_LENGTH]

  Pre-shared key.
- void(* p_callback)(sf_wifi_callback_args_t *p_args)

  Pointer to Connection status notification callback function.

6.35.7.5 sf_wifi_cfg_t

sf_wifi_cfg_t

Detailed description

Define the WiFi configuration parameters

Variables

- uint8_t mac_addr[6]

  MAC address of WiFi Device.
- sf_wifi_interface_hw_mode_t hw_mode

  Modulation type: 11a/b/g/n.
- uint8_t tx_power

  Sets transmit power in dBm.
- sf_wifi_rts_t rts

  RTS/CTS handshake flag.
- uint16_t fragmentation

  Fragmentation threshold.
- uint8_t dtim

  Delivery traffic indication message interval.
- sf_wifi_high_throughput_t high_throughput

  High-throughput mode. Only valid for 802.11n.
- sf_wifi_preamble_t preamble

  Preamble type.
- sf_wifi_wmm_t wmm

  WiFi Multimedia Mode flag. If enabled, also requires.
- uint8_t max_stations

  Maximum permitted stations. Valid in AP mode only.

- sf_wifi_ssid_broadcast_t ssid_broadcast

  SSID broadcast flag. Valid in AP mode only.
- sf_wifi_access_control_t access_control

  Mode of access control MAC list.
- uint32_t beacon

  Beacon interval. Valid in AP mode only.
- uint32_t station_inactivity_timeout

  Station inactivity timeout value. Valid in AP mode only.
- sf_wifi_wds_t wds

  WDS flag. Valid in AP mode only.
- void * p_buffer_pool_rx

  Pointer to Network stack Rx buffer pool.
- sf_wifi_mandatory_high_throughput_t req_high_throughput

  Only allow HT mode. Valid in AP mode only.
- void(* p_callback)(sf_wifi_callback_args_t *p_args)

  Pointer to callback function.
- void const * p_context

  User defined context passed into callback function.
- void const * p_extend

  Instance specific configuration.

6.35.7.6 sf_wifi_stats_t

sf_wifi_stats_t

Detailed description

Define the statistic and error counters for this IP instance.

Variables

- uint32_t rx_pkts

  Packets received successfully.
- uint32_t tx_pkts

  Packets transmitted successfully.
- uint32_t tx_err

  Transmit errors.

6.35.7.7 sf_wifi_scan_t

sf_wifi_scan_t

Detailed description

Define the structure to store the SSID scan information

Variables

- sf_wifi_interface_hw_mode_t hw_mode

  Hardware mode 802.11a/b/g/n.

- uint8_t rssi

  Signal Strength.

- uint8_t ssid[SF_WIFI_SSID_LENGTH+1]

  SSID name.

- uint8_t bssid[SF_WIFI_MAC_ADDR_LENGTH]

  Basic Service Set Identification (i.e. MAC address of Access Point)

- uint8_t channel

  Radio channel that the AP beacon was received on.

- sf_wifi_security_type_t security

  Security type.

- sf_wifi_encryption_type_t encryption

  Encryption type.

- sf_wifi_bss_type_t bss_type

  Network type.

6.35.7.8 sf_wifi_ctrl_t

sf_wifi_ctrl_t

Detailed description

WiFi Framework control structure

Variables

- void * p_driver_handle

  Storage for information needed for each WiFi device driver in the system.

6.35.7.9 sf_wifi_api_t

sf_wifi_api_t

Detailed description

Framework API structure. Implementations will use the following API.

6.35.7.10 open

```
ssp_err_t(* sf_wifi_api_t::open) (sf_wifi_ctrl_t *p_ctrl, sf_wifi_cfg_t const
*const p_cfg)
```

Brief description

Initializes the network interface for data transfers.

Detailed description

Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.

Table 466:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to user-provided storage for the control block.
p_cfg	in	Pointer to WiFi configuration structure.

Parameter p_ctrl

Definition: `sf_wifi_ctrl_t`

WiFi Framework control structure

Parameter p_cfg

Definition: `sf_wifi_cfg_t const *const p_cfg`

Define the WiFi configuration parameters

- `sf_wifi_cfg_t`::`mac_addr`

  MAC address of WiFi Device.

- `sf_wifi_cfg_t`::`sf_wifi_interface_hw_mode_t`

  Modulation type: 11a/b/g/n.

  Enumerated as:

  – SF_WIFI_INTERFACE_HW_MODE_11A

  – SF_WIFI_INTERFACE_HW_MODE_11B

  – SF_WIFI_INTERFACE_HW_MODE_11G

  – SF_WIFI_INTERFACE_HW_MODE_11N

- `sf_wifi_cfg_t`::`tx_power`

  Sets transmit power in dBm.

- sf_wifi_cfg_t::sf_wifi_rts_t

  RTS/CTS handshake flag.

  Enumerated as:

  – SF_WIFI_RTS_DISABLE

  – SF_WIFI_RTS_ENABLE
- sf_wifi_cfg_t::fragmentation

  Fragmentation threshold.
- sf_wifi_cfg_t::dtim

  Delivery traffic indication message interval.
- sf_wifi_cfg_t::sf_wifi_high_throughput_t

  High-throughput mode. Only valid for 802.11n.

  Enumerated as:

  – SF_WIFI_HIGH_THROUGHPUT_DISABLE

  – SF_WIFI_HIGH_THROUGHPUT_ENABLE
- sf_wifi_cfg_t::sf_wifi_preamble_t

  Preamble type.

  Enumerated as:

  – SF_WIFI_PREAMBLE_SHORT

  – SF_WIFI_PREAMBLE_LONG
- sf_wifi_cfg_t::sf_wifi_wmm_t

  WiFi Multimedia Mode flag. If enabled, also requires.

  Enumerated as:

  – SF_WIFI_WMM_DISABLE

  – SF_WIFI_WMM_ENABLE
- sf_wifi_cfg_t::max_stations

  Maximum permitted stations. Valid in AP mode only.
- sf_wifi_cfg_t::sf_wifi_ssid_broadcast_t

  SSID broadcast flag. Valid in AP mode only.

  Enumerated as:

  – SF_WIFI_SSID_BROADCAST_DISABLE

  – SF_WIFI_SSID_BROADCAST_ENABLE

- sf_wifi_cfg_t::sf_wifi_access_control_t

  Mode of access control MAC list.

  Enumerated as:

  – SF_WIFI_ACCESS_CONTROL_DISABLE

  – SF_WIFI_ACCESS_CONTROL_DENY

  – SF_WIFI_ACCESS_CONTROL_ALLOW
- sf_wifi_cfg_t::beacon

  Beacon interval. Valid in AP mode only.
- sf_wifi_cfg_t::station_inactivity_timeout

  Station inactivity timeout value. Valid in AP mode only.
- sf_wifi_cfg_t::sf_wifi_wds_t

  WDS flag. Valid in AP mode only.

  Enumerated as:

  – SF_WIFI_WDS_DISABLE

  – SF_WIFI_WDS_ENABLE
- sf_wifi_cfg_t::p_buffer_pool_rx

  Pointer to Network stack Rx buffer pool.
- sf_wifi_cfg_t::sf_wifi_mandatory_high_throughput_t

  Only allow HT mode. Valid in AP mode only.

  Enumerated as:

  – SF_WIFI_MANDATORY_HIGH_THROUGHPUT_DISABLE

  – SF_WIFI_MANDATORY_HIGH_THROUGHPUT_ENABLE
- sf_wifi_cfg_t::p_callback

  Pointer to callback function.
- sf_wifi_cfg_t::p_context

  User defined context passed into callback function.
- sf_wifi_cfg_t::p_extend

  Instance specific configuration.

6.35.7.11 close

```
ssp_err_t(* sf_wifi_api_t::close) (sf_wifi_ctrl_t *const p_ctrl)
```

Brief description

De-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.

Detailed description

Table 467:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the control block for the WiFi module.

Parameter p_ctrl

Definition: `sf_wifi_ctrl_t`

WiFi Framework control structure

6.35.7.12 multicastListAdd

```
ssp_err_t(* sf_wifi_api_t::multicastListAdd) (sf_wifi_ctrl_t *const p_ctrl,
uint8_t const *const p_mac_addr)
```

Brief description

Add the given MAC address to the multicast filter list.

Detailed description

Table 468:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the WiFi module.
p_mac_addr	in	Pointer to the Mac address.

Parameter p_ctrl

Definition: `sf_wifi_ctrl_t`

WiFi Framework control structure

Parameter p_mac_addr

`uint8_t`

6.35.7.13 multicastListDelete

```
ssp_err_t(* sf_wifi_api_t::multicastListDelete) (sf_wifi_ctrl_t *const p_ctrl,
uint8_t const *const p_mac_addr)
```

Detailed description

Delete the given MAC address from the multicast filter list.

Table 469:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the WiFi module.
p_mac_addr	in	Pointer to the Mac address.

Parameter p_ctrl

Definition: `sf_wifi_ctrl_t`

WiFi Framework control structure

Parameter p_mac_addr

`uint8_t`

6.35.7.14 statisticsGet

```
ssp_err_t(* sf_wifi_api_t::statisticsGet) (sf_wifi_ctrl_t *const p_ctrl,
sf_wifi_stats_t *const p_wifi_device_stats)
```

Brief description

Get the interface statistics.

Detailed description

Table 470:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the WiFi module.
p_wifi_device_stats	out	Pointer to the WiFi module data statistics.

Parameter p_ctrl

Definition: `sf_wifi_ctrl_t`

WiFi Framework control structure

Parameter p_wifi_device_stats

Definition: `sf_wifi_stats_t*const p_wifi_device_stats`

Define the statistic and error counters for this IP instance.

- sf_wifi_stats_t::rx_pkts

  Packets received successfully.

- sf_wifi_stats_t::tx_pkts

  Packets transmitted successfully.

- sf_wifi_stats_t::tx_err

  Transmit errors.

6.35.7.15 transmit

```
ssp_err_t(* sf_wifi_api_t::transmit) (sf_wifi_ctrl_t *const p_ctrl, uint8_t
*const p_buf, uint32_t length)
```

Brief description

Transmit data packet.

Detailed description

Table 471:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the WiFi module.
p_buf	in	Pointer to transmit buffer
length	in	Transmit buffer length

Parameter p_ctrl

Definition: sf_wifi_ctrl_t

WiFi Framework control structure

Parameter p_buf

uint8_t

Parameter length

uint32_t

6.35.7.16 provisioningSet

```
ssp_err_t(* sf_wifi_api_t::provisioningSet) (sf_wifi_ctrl_t *const p_ctrl,
sf_wifi_provisioning_t const *const p_wifi_provisioning)
```

Brief description

Set WiFi module provisioning which will configure the module in AP or Client mode.

Detailed description

Table 472:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the WiFi module.
p_wifi_provisioning	in	Pointer to WiFi provisioning structure

Parameter p_ctrl

Definition: `sf_wifi_ctrl_t`

WiFi Framework control structure

Parameter p_wifi_provisioning

Definition: `sf_wifi_provisioning_tconst *const p_wifi_provisioning`

WiFi Provisioning parameters

- `sf_wifi_provisioning_t::mode`

  Select AP or Client mode.

- `sf_wifi_provisioning_t::channel`

  RF Channel number.

- `sf_wifi_provisioning_t::ssid`

  SSID.

- `sf_wifi_provisioning_t::security`

  Security type.

- `sf_wifi_provisioning_t::encryption`

  Encryption type.

- `sf_wifi_provisioning_t::key`

  Pre-shared key.

- `sf_wifi_provisioning_t::p_callback`

  Pointer to Connection status notification callback function.

6.35.7.17 provisioningGet

```
ssp_err_t(* sf_wifi_api_t::provisioningGet) (sf_wifi_ctrl_t *const p_ctrl,
sf_wifi_provisioning_t *const p_wifi_provisioning)
```

Brief description

Get the provisioning information of WiFi module.

Detailed description

Table 473:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the WiFi module.
p_wifi_provisioning	out	Pointer to WiFi provisioning structure

Parameter p_ctrl

Definition: `sf_wifi_ctrl_t`

WiFi Framework control structure

Parameter p_wifi_provisioning

Definition: `sf_wifi_provisioning_t*const p_wifi_provisioning`

WiFi Provisioning parameters

- `sf_wifi_provisioning_t::mode`

  Select AP or Client mode.

- `sf_wifi_provisioning_t::channel`

  RF Channel number.

- `sf_wifi_provisioning_t::ssid`

  SSID.

- `sf_wifi_provisioning_t::security`

  Security type.

- `sf_wifi_provisioning_t::encryption`

  Encryption type.

- `sf_wifi_provisioning_t::key`

  Pre-shared key.

- `sf_wifi_provisioning_t::p_callback`

  Pointer to Connection status notification callback function.

6.35.7.18 infoGet

```
ssp_err_t(* sf_wifi_api_t::infoGet) (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_info_t
*const p_wifi_info)
```

Brief description

Get WiFi module information.

Detailed description

Table 474:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the WiFi module.
p_wifi_info	out	Pointer to WiFi module information structure

Parameter p_ctrl

Definition: `sf_wifi_ctrl_t`

WiFi Framework control structure

Parameter p_wifi_info

Definition: `sf_wifi_info_t*const p_wifi_info`

Configuration about underlying device driver.

- `sf_wifi_info_t::p_chipset`

  Pointer to sting showing WiFi chipset/driver information.

- `sf_wifi_info_t::rssi`

  Received signal strength indication.

- `sf_wifi_info_t::noise_level`

  Signal to noise ratio.

- `sf_wifi_info_t::link_quality`

  Signal strength / quality.

6.35.7.19 scan

```
ssp_err_t(* sf_wifi_api_t::scan) (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_scan_t
*const p_scan, uint8_t *const p_cnt)
```

Brief description

Scan for WiFi SSIDs.

Detailed description

Table 475:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the WiFi module.

Table 475:Parameters (Continued)

Name	Direction	Description
p_scan	out	Pointer to structure which will hold scan result. It is the responsibility of the caller to ensure that adequate space is available to hold scan results.
p_cnt	inout	Pointer to variable, specifying maximum number of SSID's to scan and will be updated to number of actual SSIDs scanned by device

Parameter p_ctrl

Definition: `sf_wifi_ctrl_t`

WiFi Framework control structure

Parameter p_scan

Definition: `sf_wifi_scan_t*const p_scan`

Define the structure to store the SSID scan information

- `sf_wifi_scan_t::hw_mode`

  Hardware mode 802.11a/b/g/n.

- `sf_wifi_scan_t::rssi`

  Signal Strength.

- `sf_wifi_scan_t::ssid`

  SSID name.

- `sf_wifi_scan_t::bssid`

  Basic Service Set Identification (i.e. MAC address of Access Point)

- `sf_wifi_scan_t::channel`

  Radio channel that the AP beacon was received on.

- `sf_wifi_scan_t::security`

  Security type.

- `sf_wifi_scan_t::encryption`

  Encryption type.

- `sf_wifi_scan_t::bss_type`

  Network type.

Parameter p_cnt

`uint8_t`

6.35.7.20 ACLAdd

```
ssp_err_t(* sf_wifi_api_t::ACLAdd) (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac)
```

Brief description

Adds a MAC address to the Access Control List. Valid in AP mode only.

Detailed description

Table 476:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the WiFi module.
p_mac	in	Pointer to MAC address

Parameter p_ctrl

Definition: `sf_wifi_ctrl_t`

WiFi Framework control structure

Parameter p_mac

`uint8_t`

6.35.7.21 ACLDelete

```
ssp_err_t(* sf_wifi_api_t::ACLDelete) (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac)
```

Brief description

Deletes a MAC address from Access Control List. Valid in AP mode only.

Detailed description

Table 477:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the WiFi module.
p_mac	in	Pointer to MAC address

Parameter p_ctrl

Definition: `sf_wifi_ctrl_t`

WiFi Framework control structure

Parameter p_mac

`uint8_t`

6.35.7.22 macAddressGet

```
ssp_err_t(* sf_wifi_api_t::macAddressGet) (sf_wifi_ctrl_t *const p_ctrl, uint8_t *const p_mac)
```

Brief description

Get WiFi module MAC address.

Detailed description

Table 478:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the WiFi module.
p_mac	out	Pointer to MAC address

Parameter p_ctrl

Definition: `sf_wifi_ctrl_t`

WiFi Framework control structure

Parameter p_mac

`uint8_t`

6.35.7.23 macAddressSet

```
ssp_err_t(* sf_wifi_api_t::macAddressSet) (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac)
```

Brief description

Set WiFi module MAC address.

Detailed description

Table 479:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the WiFi module.
p_mac	in	Pointer to MAC address

Parameter p_ctrl

Definition: sf_wifi_ctrl_t

WiFi Framework control structure

Parameter p_mac

uint8_t

6.35.7.24 versionGet

```
ssp_err_t(* sf_wifi_api_t::versionGet) (ssp_version_t *const p_version)
```

Brief description

Gets version and stores it in provided pointer p_version.

Detailed description

Table 480:Parameters

Name	Direction	Description
p_version	out	pointer to memory location to return version number

Parameter p_version

6.35.7.25 sf_wifi_instance_t

sf_wifi_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sf_wifi_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_wifi_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_wifi_api_t const * p_api

  Pointer to the API structure for this instance.

6.36 SF WIFI NSAL Interface

RTOS-integrated SF WIFI NSAL Framework Interface.

6.36.1 Summary

This SSP Interface provides access to the ThreadX-aware SF WIFI NSAL Framework.

6.36.2 Data structures

- sf_wifi_nsal_cfg_t
- sf_wifi_nsal_callback_args_t

6.36.3 Enumerations

- sf_wifi_nsal_zero_copy_t

6.36.4 Defines

- #define SF_WIFI_NSAL_MAC_CHANGED

  Initial value: `(NX_LINK_USER_COMMAND + 1U)`

6.36.5 API Data

6.36.5.1 sf_wifi_nsal_zero_copy_t

`sf_wifi_nsal_zero_copy_t`

Detailed description

Zero Copy Configuration Enumeration

Enumerated values

Name	Description
SF_WIFI_NSAL_ZERO_COPY_DISABLE	Zero copy is disabled.
SF_WIFI_NSAL_ZERO_COPY_ENABLE	Zero copy is enabled.

6.36.6 API Structures

6.36.6.1 sf_wifi_nsal_cfg_t

sf_wifi_nsal_cfg_t

Detailed description

Define the NSAL configuration parameters

Variables

- sf_wifi_nsal_zero_copy_t tx_zero_copy

  Transmit path zero copy support.
- sf_wifi_nsal_zero_copy_t rx_zero_copy

  Receive path zero copy support.
- uint8_t * p_tx_packet_buffer

  Pointer to Tx buffer used to consolidate data from chained NetX packets.

6.36.6.2 sf_wifi_nsal_callback_args_t

sf_wifi_nsal_callback_args_t

Detailed description

Define the NSAL callback arguments

Variables

- NX_INTERFACE * p_interface

  Pointer to NetX interface.
- sf_wifi_nsal_cfg_t * p_wifi_nsal_cfg

  pointer to NSAL configuration

6.37 SF WIFI On-Chip Stack Interface

RTOS-integrated SF WIFI On-Chip Stack Interface.

6.37.1 Summary

This SSP Interface provides access to the ThreadX-aware SF WIFI Framework.

6.37.2 Interface API

sf_wifi_onchip_stack_api_t

Function name	Description
.open	Pointer to function which initializes the network interface for data transfers.Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.
.close	Pointer to function which un-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.
.ipAddressCfg	Configures IP address of the interface.
.dhcpServerStart	Starts DHCP server on the interface.
.dhcpServerStop	Stop DHCP server on the interface.
.versionGet	Gets version and stores it in provided pointer p_version.

6.37.3 Data structures

- sf_wifi_onchip_stack_ip_cfg_t
- sf_wifi_onchip_stack_cfg_t
- sf_wifi_onchip_stack_ctrl_t
- sf_wifi_onchip_stack_instance_t

6.37.4 Enumerations

- sf_wifi_onchip_stack_ip_addr_mode_t

6.37.5 Defines

- #define SF_WIFI_ONCHIP_STACK_API_VER_MAJOR

  Initial value: `(1U)`

  Major Version of the API defined in this file
- #define SF_WIFI_ONCHIP_STACK_API_VER_MINOR

  Initial value: `(0U)`

  Minor Version of the API defined in this file

6.37.6 API Data

6.37.6.1 sf_wifi_onchip_stack_ip_addr_mode_t

`sf_wifi_onchip_stack_ip_addr_mode_t`

Detailed description

IP addressing modes

Enumerated values

Name	Description
SF_WIFI_IP_ADDR_GET	Read the IP address assigned to interface.
SF_WIFI_IP_ADDR_STATIC	Statically configure the IP address.
SF_WIFI_IP_ADDR_DHCP	Get the IP address from DHCP server, dynamic assignment.

6.37.7 API Structures

6.37.7.1 sf_wifi_onchip_stack_ip_cfg_t

sf_wifi_onchip_stack_ip_cfg_t

Detailed description

Define IP Interface configuration information

Variables

- sf_wifi_onchip_stack_ip_addr_mode_t ip_addr_mode

  Addressing mode.
- sf_wifi_ip_addr_t ip_addr

  Interface IP address.
- sf_wifi_ip_addr_t netmask

  Interface netmask.
- sf_wifi_ip_addr_t gateway

  Interface Gateway.

6.37.7.2 sf_wifi_onchip_stack_cfg_t

sf_wifi_onchip_stack_cfg_t

Detailed description

Define the WiFi configuration parameters

Variables

- sf_wifi_instance_t const * p_lower_lvl_wifi

  Pointer to SF WiFi instance.

- void * p_extend

  Extended configuration.

6.37.7.3 sf_wifi_onchip_stack_ctrl_t

sf_wifi_onchip_stack_ctrl_t

Detailed description

WiFi Framework control structure

Variables

- sf_wifi_instance_t * p_lower_lvl_wifi

  Pointer to SF WiFi instance.

  Storage for information needed for each WiFi device driver in the system.

6.37.7.4 sf_wifi_onchip_stack_api_t

sf_wifi_onchip_stack_api_t

Detailed description

Framework API structure. Implementations will use the following API.

6.37.7.5 open

```
ssp_err_t(* sf_wifi_onchip_stack_api_t::open) (sf_wifi_onchip_stack_ctrl_t
*p_ctrl, sf_wifi_onchip_stack_cfg_t const *const p_cfg)
```

Detailed description

Pointer to function which initializes the network interface for data transfers.

Initial driver configuration, enable the driver link, enable interrupts and make device ready for data transfer.

Table 481:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to user-provided storage for the control block.
p_cfg	in	Pointer to configuration structure.

Parameter p_ctrl

Definition: sf_wifi_onchip_stack_ctrl_t

WiFi Framework control structure

Parameter p_cfg

Definition: sf_wifi_onchip_stack_cfg_t const *const p_cfg

Define the WiFi configuration parameters

- sf_wifi_onchip_stack_cfg_t::sf_wifi_instance_t

  Pointer to SF WiFi instance.

- sf_wifi_onchip_stack_cfg_t::p_extend

  Extended configuration.

6.37.7.6 close

```
ssp_err_t(* sf_wifi_onchip_stack_api_t::close) (sf_wifi_onchip_stack_ctrl_t
*const p_ctrl)
```

Detailed description

Pointer to function which un-initialize the network interface and may put it in low power mode or power it off. Close the driver, disable the driver link, disable interrupt.

Table 482:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the control block

Parameter p_ctrl

Definition: sf_wifi_onchip_stack_ctrl_t

WiFi Framework control structure

6.37.7.7 ipAddressCfg

```
ssp_err_t(* sf_wifi_onchip_stack_api_t::ipAddressCfg)
(sf_wifi_onchip_stack_ctrl_t *const p_ctrl, sf_wifi_onchip_stack_ip_cfg_t *const
p_cfg)
```

Detailed description

Configures IP address of the interface.

Table 483:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block

Table 483:Parameters (Continued)

Name	Direction	Description
p_cfg	inout	Pointer to IP configuration structure.

Parameter p_ctrl

Definition: sf_wifi_onchip_stack_ctrl_t

WiFi Framework control structure

Parameter p_cfg

6.37.7.8 dhcpServerStart

```
ssp_err_t(* sf_wifi_onchip_stack_api_t::dhcpServerStart)
(sf_wifi_onchip_stack_ctrl_t *const p_ctrl, sf_wifi_ip_addr_t const *const
p_start_ip, sf_wifi_ip_addr_t const *const p_end_ip)
```

Detailed description

Starts DHCP server on the interface.

Table 484:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block
p_start_ip	in	Pointer to start IP address
p_end_ip	in	Pointer to end IP address

Parameter p_ctrl

Definition: sf_wifi_onchip_stack_ctrl_t

WiFi Framework control structure

Parameter p_start_ip
Parameter p_end_ip

6.37.7.9 dhcpServerStop

```
ssp_err_t(* sf_wifi_onchip_stack_api_t::dhcpServerStop)
(sf_wifi_onchip_stack_ctrl_t *const p_ctrl)
```

Detailed description

Stop DHCP server on the interface.

Table 485:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block

Parameter p_ctrl

Definition: sf_wifi_onchip_stack_ctrl_t

WiFi Framework control structure

6.37.7.10 versionGet

```
ssp_err_t(* sf_wifi_onchip_stack_api_t::versionGet) (ssp_version_t *const
p_version)
```

Detailed description

Gets version and stores it in provided pointer p_version.

Table 486:Parameters

Name	Direction	Description
p_version	out	pointer to memory location to return version number

Parameter p_version

6.37.7.11 sf_wifi_onchip_stack_instance_t

sf_wifi_onchip_stack_instance_t

Detailed description

SF WiFi On Chip Stack Instance structure

Variables

- sf_wifi_onchip_stack_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- sf_wifi_onchip_stack_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- sf_wifi_onchip_stack_api_t const * p_api

  Pointer to the API structure for this instance.

6.38 SF WIFI NSAL on NetX

NetX NSAL interface implementation header file.

6.38.1 Summary

This SSP Interface provides access to the ThreadX-aware SF WIFI NSAL NX Framework.

6.38.2 Functions

- nsal_netx_driver

6.38.3 nsal_netx_driver

```
 nsal_netx_driver (  NX_IP_DRIVER * driver ,  sf_wifi_instance_t  const
* p_wifi_instance ,  sf_wifi_nsal_cfg_t  * p_wifi_nsal_cfg )
```

6.38.3.1 Detailed description

NSAL NetX Driver Entry function

6.38.3.2 Function steps

- Check if the link is up

Chapter 7 API Reference: Framework

7.1 API Reference: Framework Layer

The framework layer provides RTOS aware drivers for functional use cases.

- ADC Periodic Framework
- Audio Framework
- DAC Audio Playback Framework
- I2S Audio Playback Framework
- ADC Audio Recording Framework
- I2S Audio Recording Framework
- QSPI Block Media Framework
- RAM Block Media Framework
- SDMMC Block Media Framework
- Cellular NSAL Implementation on NetX
- Telnet Communication Framework
- Console Framework
- FX_IO Framework
- GUIX Synergy Port
- Telnet Communication Framework
- USB Communication Framework
- USB Communication Framework V2
- External IRQ Framework
- I2C Framework
- JPEG Decode Framework
- Messaging Framework
- Power Profiles Framework
- Power Profiles V2 Framework
- SPI Framework
- Thread Monitor Framework
- CTSU Framework
- CTSU Button Framework
- CTSU Slider Framework

- I2C Touch Panel Framework
- UART Framework Instance
- NetX Synergy Port
- NetX Synergy Port PHY Driver
- Express Logic USBX Synergy Port Framework

7.2 ADC Periodic Framework

RTOS-integrated ADC Framework.

7.2.1 Functions

- SF_ADC_PERIODIC_Open
- SF_ADC_PERIODIC_Start
- SF_ADC_PERIODIC_Stop
- SF_ADC_PERIODIC_Close
- SF_ADC_PERIODIC_VersionGet

7.2.2 Defines

- #define SF_ADC_PERIODIC_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file
- #define SF_ADC_PERIODIC_CODE_VERSION_MINOR

  Initial value: `(5U)`

7.2.3 SF_ADC_PERIODIC_Open

```
ssp_err_t SF_ADC_PERIODIC_Open ( sf_adc_periodic_ctrl_t  *const p_api_ctrl ,
 sf_adc_periodic_cfg_t  const *const p_cfg )
```

7.2.3.1 Brief description

Configures periodic ADC framework and optionally starts the timer.

7.2.3.2 Detailed description

The SF_ADC_PERIODIC_Open function acquires a mutex for the ADC Unit used, then calls the driver .open function in the p_api parameter. The mutex is released following the driver layer open function.

Table 487:Return values

Name	Description
SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_api_ctrl or p_cfg. See HAL driver for other possible causes.
SSP_ERR_INVALID_ARGUMENT	An invalid argument is used
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use a mutex or to create an internal thread.

See Common Error Codes or HAL driver for other possible return codes or causes.

7.2.3.3 Function steps

- Save driver structure pointer for use in other framework layer functions
- Create a mutex
- Get mutex before calling lower layer, which will access HW registers.
- Initialize the HAL layer
- Post the Mutex
- If any of the HAL layer initializations failed, then delete the mutex and exit the function with the error code
- Delete the mutex
- If there was a mutex error, return it
- Mark control block open so other tasks know it is valid

7.2.4 SF_ADC_PERIODIC_Start

```
ssp_err_t SF_ADC_PERIODIC_Start ( sf_adc_periodic_ctrl_t  *const p_api_ctrl )
```

7.2.4.1 Brief description

Gets mutex, starts the periodic ADC scan, and releases mutex.

7.2.4.2 Detailed description

ATTENTION: The driver will enable the ADC to be triggered via timer event; there will be a time delay from the time this function is called to the time the hardware timer count expires and triggers the scan.

Table 488:Return values

Name	Description
SSP_SUCCESS	ADC Periodic Scan started successfully.
SSP_ERR_ASSERTION	One or more pointers point to NULL.
SSP_ERR_NOT_OPEN	Driver control block not valid. Call SF_ADC_PERIODIC_Open to configure.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use a mutex or to create an internal thread.
SSP_ERR_IN_USE	The module is currently busy performing another operation

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls:

- scanStart
- start

7.2.4.3 Function steps

- Get mutex, start timer, then release mutex
- Enable the ADC to receive hardware triggers
- If the scan was successfully enabled in the ADC HAL,
- Start the timer to generate the ADC trigger events
- Return the mutex
- If there was a mutex error, return it

7.2.5 SF_ADC_PERIODIC_Stop

```
ssp_err_t SF_ADC_PERIODIC_Stop ( sf_adc_periodic_ctrl_t  *const p_api_ctrl )
```

7.2.5.1 Brief description

Gets mutex, stops the periodic ADC scan, and releases mutex.

7.2.5.2 Detailed description

Table 489:Return values

Name	Description
SSP_SUCCESS	Periodic ADC scan stopped successfully.
SSP_ERR_ASSERTION	One or more pointers point to NULL..
SSP_ERR_NOT_OPEN	Driver control block not valid. Call SF_ADC_PERIODIC_Open to configure.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use a mutex or to create an internal thread.
SSP_ERR_IN_USE	The module is currently busy performing another operation

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls:

- stop

7.2.5.3 Function steps

- Get mutex, stop timer, then release mutex
- If there was a mutex error, return it

7.2.6 SF_ADC_PERIODIC_Close

```
ssp_err_t SF_ADC_PERIODIC_Close ( sf_adc_periodic_ctrl_t  *const p_api_ctrl )
```

7.2.6.1 Brief description

The close function acquires the unit's mutex, closes all lower level drivers, releases and deletes the mutex.

7.2.6.2 Detailed description

Table 490:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	One or more pointers point to NULL.
SSP_ERR_NOT_OPEN	Driver control block not valid. Call SF_ADC_PERIODIC_Open to configure.
SSP_ERR_IN_USE	The module is currently busy performing another operation
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use a mutex or to create an internal thread.

See Common Error Codes or HAL driver for other possible return codes or causes.

7.2.6.3 Function steps

- Get mutex since this will access hardware registers
- Close the HAL layer modules
- Post the mutex
- If all the HAL layers closed successfully, then delete the mutex and mark module as un-initialized
- Delete RTOS services used
- Clear information from control block so other functions know this instance is closed
- If there was a mutex error, return it

7.2.7 SF_ADC_PERIODIC_VersionGet

```
ssp_err_t SF_ADC_PERIODIC_VersionGet ( ssp_version_t  *const p_version )
```

7.2.7.1 Brief description

Gets version and stores it in provided pointer p_version.

7.2.7.2 Detailed description

Table 491:Return values

Name	Description
SSP_SUCCESS	Version returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was null.

7.2.8 Extensions

7.2.8.1 sf_adc_periodic_instance_ctrl_t

sf_adc_periodic_instance_ctrl_t

Detailed description

Channel instance control block. DO NOT INITIALIZE. Initialization occurs when SF_ADC_PERIODIC_Open is called

Variables

- uint32_t open

  Used by driver to check if pointer to control block is valid.
- TX_MUTEX mutex

  Mutex used to protect access to lower level driver hardware registers.
- adc_instance_t const * p_lower_lvl_adc

  Pointer to the ADC instance.
- timer_instance_t const * p_lower_lvl_timer

  Pointer to the Timer instance.
- transfer_instance_t const * p_lower_lvl_transfer

  Pointer to the Transfer instance.
- void const *volatile p_src_transfer

  Source pointer for the low level transfer method.
- adc_data_size_t * p_data_buffer

  Pointer to the buffer that will store the samples.
- uint32_t data_buffer_length

  Length of the data buffer that will store the samples.
- uint32_t data_buffer_index

  Index of the data buffer where data is to be written to next.

- uint32_t sample_count

  Samples per channel to be buffered before notifying the app.
- uint32_t dtc_transfer_length

  Total Length of DTC transfer for requested number of samples.
- void(* p_callback)(*p_args)

  Callback function.
- void const * p_context

  Placeholder for user data.

7.3 Audio Framework

RTOS-integrated Audio Framework.

7.3.1 Summary

This module is a ThreadX-aware Audio Framework. The module implements Audio Framework Interface.
Name of module used by error logger macro

7.3.2 Functions

- SF_AUDIO_PLAYBACK_Open
- SF_AUDIO_PLAYBACK_Close
- SF_AUDIO_PLAYBACK_Start
- SF_AUDIO_PLAYBACK_Pause
- SF_AUDIO_PLAYBACK_Stop
- SF_AUDIO_PLAYBACK_Resume
- SF_AUDIO_PLAYBACK_VolumeSet
- SF_AUDIO_PLAYBACK_VersionGet

7.3.3 Defines

- #define SF_AUDIO_PLAYBACK_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file
- #define SF_AUDIO_PLAYBACK_CODE_VERSION_MINOR

  Initial value: `(6U)`

- #define SF_AUDIO_PLAYBACK_STACK_SIZE

  Initial value: (SF_AUDIO_PLAYBACK_CFG_THREAD_STACK_SIZE)

  Audio playback internal thread stack size. Varies by application, but rarely requires more than 256 bytes.

7.3.4 SF_AUDIO_PLAYBACK_Open

```
ssp_err_t SF_AUDIO_PLAYBACK_Open ( sf_audio_playback_ctrl_t  *const p_api_ctrl ,
 sf_audio_playback_cfg_t  const *const p_cfg )
```

7.3.4.1 Detailed description

Implements open.

Table 492:Return values

Name	Description
SSP_SUCCESS	Audio hardware successfully configured.
SSP_ERR_ASSERTION	A pointer is NULL or a parameter is invalid.
SSP_ERR_OUT_OF_MEMORY	The number of streams open at once is limited to SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS. If this number is exceeded, an out of memory error occurs.

See Common Error Codes or lower level drivers for other possible return codes.

NOTE: This function is not reentrant.

7.3.4.2 Function steps

- Open hardware if it is not already open.
- Create event flags to notify playback thread when playback of a buffer is complete.
- Store stream pointer in common control block.
- Initialize stream variables.
- Store queue pointers for audio data and clear stream owner pointers.
- Mark stream opened so it can be used by other API's.

7.3.5 SF_AUDIO_PLAYBACK_Close

```
ssp_err_t SF_AUDIO_PLAYBACK_Close ( sf_audio_playback_ctrl_t
*const p_api_ctrl )
```

7.3.5.1 Detailed description

Implements close.

Table 493:Return values

Name	Description
SSP_SUCCESS	Audio instance successfully closed
SSP_ERR_ASSERTION	p_ctrl is NULL
SSP_ERR_NOT_OPEN	The stream control block p_ctrl is not initialized.

See Common Error Codes or lower level drivers for other possible return codes.

NOTE: This function is not reentrant.

7.3.5.2 Function steps

- Mark stream as unused in hardware control block and determine if all streams are closed.
- Mark hardware control block as unused so it can be reconfigured.
- Close lower level drivers
- Delete RTOS services used
- Mark control block as unused so it can be reconfigured.

7.3.6 SF_AUDIO_PLAYBACK_Start

```
ssp_err_t SF_AUDIO_PLAYBACK_Start ( sf_audio_playback_ctrl_t
*const p_api_ctrl ,  sf_audio_playback_data_t  *const p_data ,   UINT
const timeout )
```

7.3.6.1 Detailed description

Implements start.

Table 494:Return values

Name	Description
SSP_SUCCESS	Buffer successfully sent to audio playback thread
SSP_ERR_ASSERTION	p_ctrl is NULL
SSP_ERR_NOT_OPEN	The stream control block p_ctrl is not initialized.

NOTE: This function is reentrant.

7.3.6.2 Function steps

- Ensure that audio data is only posted from the current stream owner.
- Store new stream owner if stream is unowned.
- Set message header to audio start event. Set instance to stream instance.
- Send message with audio data to audio thread.

7.3.7 SF_AUDIO_PLAYBACK_Pause

```
ssp_err_t SF_AUDIO_PLAYBACK_Pause ( sf_audio_playback_ctrl_t
*const p_api_ctrl )
```

7.3.7.1 Detailed description

Implements pause.

Table 495:Return values

Name	Description
SSP_SUCCESS	Audio playback pause message sent to audio playback thread for this stream.
SSP_ERR_ASSERTION	p_ctrl is NULL
SSP_ERR_NOT_OPEN	The stream control block p_ctrl is not initialized.

See Common Error Codes or lower level drivers for other possible return codes.

NOTE: This function is reentrant.

7.3.8 SF_AUDIO_PLAYBACK_Stop

```
ssp_err_t SF_AUDIO_PLAYBACK_Stop ( sf_audio_playback_ctrl_t  *const p_api_ctrl )
```

7.3.8.1 Detailed description

Implements stop.

Table 496:Return values

Name	Description
SSP_SUCCESS	Audio playback stop message sent to audio playback thread for this stream.
SSP_ERR_ASSERTION	p_ctrl is NULL
SSP_ERR_NOT_OPEN	The stream control block p_ctrl is not initialized.

See Common Error Codes or lower level drivers for other possible return codes.

NOTE: This function is reentrant.

7.3.9 SF_AUDIO_PLAYBACK_Resume

```
ssp_err_t SF_AUDIO_PLAYBACK_Resume ( sf_audio_playback_ctrl_t
*const p_api_ctrl )
```

7.3.9.1 Detailed description

Implements resume.

Table 497:Return values

Name	Description
SSP_SUCCESS	Audio playback resume message sent to audio playback thread for this stream.
SSP_ERR_ASSERTION	p_ctrl is NULL
SSP_ERR_NOT_OPEN	The stream control block p_ctrl is not initialized.

NOTE: This function is reentrant.

7.3.10 SF_AUDIO_PLAYBACK_VolumeSet

```
ssp_err_t SF_AUDIO_PLAYBACK_VolumeSet ( sf_audio_playback_ctrl_t
*const p_api_ctrl ,   uint8_t const volume )
```

7.3.10.1 Detailed description

Implements volumeSet.

Table 498:Return values

Name	Description
SSP_SUCCESS	Audio playback software volume level updated (applies to all streams).
SSP_ERR_ASSERTION	p_ctrl is NULL
SSP_ERR_NOT_OPEN	The stream control block p_ctrl is not initialized.

7.3.10.2 Function steps

- Update volume in control block.

7.3.11 SF_AUDIO_PLAYBACK_VersionGet

```
ssp_err_t SF_AUDIO_PLAYBACK_VersionGet ( ssp_version_t  *const p_version )
```

7.3.11.1 Detailed description

Implements versionGet.

Table 499:Return values

Name	Description
SSP_SUCCESS	Version returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was null.

7.3.12 Extensions

7.3.12.1 sf_audio_playback_common_instance_ctrl_t

sf_audio_playback_common_instance_ctrl_t

Detailed description

Audio common instance control block. DO NOT INITIALIZE. Initialization the first time open is called. Shared by all streams.

Variables

- uint32_t open

  Used to determine if driver is initialized.
- void const * p_next_buffer

  Pointer to next buffer (to be played when the current buffer completes).
- uint32_t next_length

  Length of next buffer (to be played when the current buffer completes).
- sf_message_instance_t const * p_message

  Pointer to message control block.
- TX_QUEUE * p_queue

  Queue subscribed to SF_MESSAGE_EVENT_CLASS_AUDIO events.
- sf_audio_playback_hw_instance_t const * p_lower_lvl_hw

  Hardware API's used.
- sf_audio_playback_instance_ctrl_t * p_stream[SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS]

  Stream specific data.
- TX_THREAD thread

  Main audio thread.

- TX_EVENT_FLAGS_GROUP flags

  Event flags used to end wait in audio thread.
- sf_audio_playback_data_type_t data_type

  Sample format required by the hardware.
- uint8_t volume

  Volume from 0 (muted) to 255 (maximum, default on open).
- uint8_t buffer_index

  Which ping pong buffer to use.
- uint8_t stack[SF_AUDIO_PLAYBACK_STACK_SIZE]

  Stack for audio thread.
- int16_t samples[2][SF_AUDIO_PLAYBACK_CFG_BUFFER_SIZE_BYTES/sizeof(int16_t)]

  Ping pong buffers, used to store converted data during transfer.
- bool playing

  State of audio instance (currently playing if true)

7.3.12.2 st_sf_audio_playback_instance_ctrl

st_sf_audio_playback_instance_ctrl

Detailed description

Audio stream instance control block. DO NOT INITIALIZE. Initialization occurs when open is called.

Variables

- uint32_t open

  Used to determine if driver is initialized.
- TX_THREAD * p_owner

  Pointer to thread that began the stream at this index. Used to ensure multiple threads don't interleave data on the same stream.
- void(* p_callback)(*p_args)

  Callback called when playback of a buffer passed to sf_audio_playback_api_t::start is complete.
- uint8_t class_instance

  Class instance used to identify the stream to the messaging framework.
- uint32_t samples_remaining

  Internal state of data samples remaining for this stream.
- uint32_t samples_total

  Total number of samples to play (independent of sample size like 8/12/16).

- uint32_t index

  Internal state of current data index for this stream.
- uint32_t end

  Used to track completion of looped playback.
- sf_audio_playback_data_t * p_data[2]

  Audio data read from queue.
- sf_audio_playback_status_t status

  Status of current stream.
- sf_audio_playback_common_instance_ctrl_t * p_common_ctrl

  Pointer to the hardware control block used by this stream.

7.4 DAC Audio Playback Framework

RTOS-integrated DAC implementation of Audio Playback Interface.

The Audio Playback Framework DAC implementation uses a timer to generate events at the sampling frequency, and uses these events to transfer PCM samples to the DAC.

7.4.1 Functions

- SF_AUDIO_PLAYBACK_HW_DAC_Open
- SF_AUDIO_PLAYBACK_HW_DAC_Start
- SF_AUDIO_PLAYBACK_HW_DAC_Stop
- SF_AUDIO_PLAYBACK_HW_DAC_Play
- SF_AUDIO_PLAYBACK_HW_DAC_DataTypeGet
- SF_AUDIO_PLAYBACK_HW_DAC_Close
- SF_AUDIO_PLAYBACK_HW_DAC_VersionGet
- sf_audio_playback_hw_callback_timer
- sf_audio_playback_hw_callback_common
- sf_audio_playback_hw_transfer_size_config

7.4.2 Variables

- g_sf_audio_playback_hw_on_sf_audio_playback_hw_dac

7.4.3 Defines

- #define SF_AUDIO_PLAYBACK_HW_DAC_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define SF_AUDIO_PLAYBACK_HW_DAC_CODE_VERSION_MINOR

  Initial value: `(6U)`

7.4.4 SF_AUDIO_PLAYBACK_HW_DAC_Open

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_Open ( sf_audio_playback_hw_ctrl_t
*const p_api_ctrl ,  sf_audio_playback_hw_cfg_t  const *const p_cfg )
```

7.4.4.1 Detailed description

Open the DAC audio driver, including the DAC HAL driver and helper timer and transfer HAL drivers.

Table 500:Return values

Name	Description
SSP_SUCCESS	Configuration of lower level drivers completed successfully.
SSP_ERR_ASSERTION	Null Pointer.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- open
- open
- open
- close
- close

NOTE: This function is reentrant if the lower level driver functions are reentrant.

7.4.4.2 Function steps

- Open Timer driver at selected frequency
- If DTC is selected, register the audio callback with the timer ISR. DTC calls activation source ISR when the transfer is complete.

- Open DAC module
- Open transfer module to transfer from buffer to DAC output register
- Configure transfer size of driver depending upon the underlying DAC resolution
- Open transfer driver
- Store driver data.
- Play linear ramp data to get DAC up to half the maximum output.

7.4.5 SF_AUDIO_PLAYBACK_HW_DAC_Start

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_Start ( sf_audio_playback_hw_ctrl_t
*const p_api_ctrl )
```

7.4.5.1 Detailed description

Start the DAC and timer HAL drivers.

Table 501:Return values

Name	Description
SSP_SUCCESS	Audio playback hardware started successfully.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- start
- start

NOTE: This function is reentrant if the lower level driver functions are reentrant.

7.4.5.2 Function steps

- Start DAC.
- Start timer.

7.4.6 SF_AUDIO_PLAYBACK_HW_DAC_Stop

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_Stop ( sf_audio_playback_hw_ctrl_t
*const p_api_ctrl )
```

7.4.6.1 Detailed description

Stop the DAC and timer HAL drivers.

Table 502:Return values

Name	Description
SSP_SUCCESS	Audio playback hardware stopped successfully.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- stop
- stop

NOTE: This function is reentrant if the lower level driver functions are reentrant.

7.4.6.2 Function steps

- Stop timer.
- Stop DAC.

7.4.7 SF_AUDIO_PLAYBACK_HW_DAC_Play

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_Play ( sf_audio_playback_hw_ctrl_t
*const p_api_ctrl ,   int16_t const *const p_buffer ,   uint32_t length )
```

7.4.7.1 Detailed description

Play a single audio buffer by input samples to the DAC at the sampling frequency configured by the timer.

Table 503:Return values

Name	Description
SSP_SUCCESS	Buffer playback began successfully.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_buffer is NULL or length is greater than 0x10000UL.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- reset

NOTE: This function is reentrant if the lower level driver functions are reentrant.

7.4.7.2 Function steps

- Reset transfer.

7.4.8 SF_AUDIO_PLAYBACK_HW_DAC_DataTypeGet

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_DataTypeGet ( sf_audio_playback_hw_ctrl_t
*const p_ctrl ,  sf_audio_playback_data_type_t  *const p_data_type )
```

7.4.8.1 Detailed description

Provides the expected data type in the pointer p_data_type.

Table 504:Return values

Name	Description
SSP_SUCCESS	Data type stored in p_data_type.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_data_type is NULL.

NOTE: This function is reentrant if the lower level driver functions are reentrant.

7.4.8.2 Function steps

- Store data type. The Synergy DAC supports only 12-bit unsigned data.

7.4.9 SF_AUDIO_PLAYBACK_HW_DAC_Close

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_Close ( sf_audio_playback_hw_ctrl_t
*const p_api_ctrl )
```

7.4.9.1 Detailed description

Close open audio driver.

Table 505:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- close
- close
- close

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

7.4.9.2 Function steps

- Close timer driver.
- Close DAC driver.
- Close transfer driver.

7.4.10 SF_AUDIO_PLAYBACK_HW_DAC_VersionGet

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_DAC_VersionGet ( ssp_version_t
*const p_version )
```

7.4.10.1 Detailed description

Stores the version of the firmware and API in provided pointer p_version.

Table 506:Return values

Name	Description
SSP_ERR_ASSERTION	The parameter p_version is NULL.
SSP_SUCCESS	Module version successfully stored in p_version.

NOTE: This function is reentrant.

7.4.11 sf_audio_playback_hw_callback_timer

```
sf_audio_playback_hw_callback_timer ( timer_callback_args_t  * p_args )
```

7.4.11.1 Detailed description

Callback function intercepted from timer driver.

Table 507:Parameters

Name	Direction	Description
p_args	in	Callback data can be used to identify what triggered the interrupt.

7.4.12 sf_audio_playback_hw_callback_common

```
sf_audio_playback_hw_callback_common (  void const * p_context )
```

7.4.12.1 Detailed description

Callback function intercepted from HAL layer.

Table 508:Parameters

Name	Direction	Description
p_context	in	Pointer to control block, used to identify what triggered the interrupt.

7.4.12.2 Function steps

- Recover context from ISR.
- Create callback arguments.
- Call user callback if not NULL.

7.4.13 sf_audio_playback_hw_transfer_size_config

```
 sf_audio_playback_hw_transfer_size_config ( transfer_cfg_t  const
* p_transfer ,  sf_audio_playback_hw_dac_cfg_t  const * p_cfg_extend )
```

7.4.13.1 Detailed description

Configures the transfer size of the transfer driver depending upon the underlying DAC resolution.

Table 509:Parameters

Name	Direction	Description
p_transfer	inout	Pointer to transfer driver configuration.
p_cfg_extend	inout	Pointer to hardware dependent configuration for DAC audio driver.

7.4.14 g_sf_audio_playback_hw_on_sf_audio_playback_hw_dac

sf_audio_playback_hw_api_t::g_sf_audio_playback_hw_on_sf_audio_playback_hw_dac

7.4.14.1 Detailed description

Function pointers for DAC implementation of audio playback API.

7.4.15 Extensions

7.4.15.1 sf_audio_playback_hw_dac_instance_ctrl_t

sf_audio_playback_hw_dac_instance_ctrl_t

Detailed description

Hardware dependent control block for DAC audio driver.

Variables

- void(* p_callback)(*p_args)

  Callback called when play is complete.
- void * p_context

  Placeholder for user data. Passed to the user callback in sf_audio_playback_hw_callback_args_t.
- dac_instance_t const * p_lower_lvl_dac

  DAC API used to access DAC hardware.

- timer_instance_t const * p_lower_lvl_timer

  Timer API used to generate sampling frequency.
- transfer_instance_t const * p_lower_lvl_transfer

  Transfer API used to transfer data each sampling frequency.

7.4.15.2 sf_audio_playback_hw_dac_cfg_t

sf_audio_playback_hw_dac_cfg_t

Detailed description

Hardware dependent configuration for DAC audio driver.

Variables

- dac_instance_t const * p_lower_lvl_dac

  DAC API used to access DAC hardware.
- timer_instance_t const * p_lower_lvl_timer

  Timer API used to generate sampling frequency.
- transfer_instance_t const * p_lower_lvl_transfer

  Transfer API used to transfer data each sampling frequency.

7.5 I2S Audio Playback Framework

RTOS-integrated I2S implementation of Audio Playback Interface.

The Audio Playback Framework I2S implementation uses the I2S interface for audio playback.

Name of module used by error logger macro

7.5.1 Functions

- SF_AUDIO_PLAYBACK_HW_I2S_Open
- SF_AUDIO_PLAYBACK_HW_I2S_Start
- SF_AUDIO_PLAYBACK_HW_I2S_Stop
- SF_AUDIO_PLAYBACK_HW_I2S_Play
- SF_AUDIO_PLAYBACK_HW_I2S_DataTypeGet
- SF_AUDIO_PLAYBACK_HW_I2S_Close
- SF_AUDIO_PLAYBACK_HW_I2S_VersionGet

7.5.2 Defines

- #define SF_AUDIO_PLAYBACK_HW_I2S_CODE_VERSION_MAJOR
  Initial value: `(1U)`
- #define SF_AUDIO_PLAYBACK_HW_I2S_CODE_VERSION_MINOR
  Initial value: `(5U)`

7.5.3 SF_AUDIO_PLAYBACK_HW_I2S_Open

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_I2S_Open ( sf_audio_playback_hw_ctrl_t
*const p_api_ctrl ,  sf_audio_playback_hw_cfg_t  const *const p_cfg )
```

7.5.3.1 Brief description

Open the I2S audio driver, including the I2S HAL driver and helper timer and transfer HAL drivers.

7.5.3.2 Detailed description

Table 510:Return values

Name	Description
SSP_SUCCESS	Configuration of lower level drivers completed successfully.
SSP_ERR_ASSERTION	One of the following parameter is null: p_ctrl or p_cfg or p_cfg_extend->p_lower_lvl_i2s or p_cfg_extend->p_lower_lvl_i2s->p_api.

See Common Error Codes and lower level driver function for other possible return codes. This function calls:

- open

NOTE: This function is reentrant if the lower level driver functions are reentrant.

7.5.3.3 Function steps

- Open I2S module
- Store driver data.

7.5.4 SF_AUDIO_PLAYBACK_HW_I2S_Start

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_I2S_Start ( sf_audio_playback_hw_ctrl_t
*const p_ctrl )
```

7.5.4.1 Brief description

Start the I2S and timer HAL drivers.

7.5.4.2 Detailed description

Table 511:Return values

Name	Description
SSP_SUCCESS	Audio playback hardware started successfully.

NOTE: This function is reentrant if the lower level driver functions are reentrant.

7.5.4.3 Function steps

- This API is not used - I2S is started when write is called from SF_AUDIO_PLAYBACK_HW_I2S_Play.

7.5.5 SF_AUDIO_PLAYBACK_HW_I2S_Stop

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_I2S_Stop ( sf_audio_playback_hw_ctrl_t
*const p_api_ctrl )
```

7.5.5.1 Brief description

Stop the I2S and timer HAL drivers.

7.5.5.2 Detailed description

Table 512:Return values

Name	Description
SSP_SUCCESS	Audio playback hardware stopped successfully.

Table 512:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.

See Common Error Codes and lower level driver function for other possible return codes. This function calls:

- stop

NOTE: This function is reentrant if the lower level driver functions are reentrant.

7.5.5.3 Function steps

- Stop I2S.

7.5.6 SF_AUDIO_PLAYBACK_HW_I2S_Play

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_I2S_Play ( sf_audio_playback_hw_ctrl_t
*const p_api_ctrl ,   int16_t const *const p_buffer ,   uint32_t length )
```

7.5.6.1 Brief description

Play a single audio buffer by input samples to the I2S at the sampling frequency configured by the timer.

7.5.6.2 Detailed description

Table 513:Return values

Name	Description
SSP_SUCCESS	Buffer playback began successfully.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_buffer is NULL or length is less than 0x10000U.

See Common Error Codes and lower level driver function for other possible return codes. This function calls:

- write

NOTE: This function is reentrant if the lower level driver functions are reentrant.

7.5.6.3 Function steps

- Reset transfer.

7.5.7 SF_AUDIO_PLAYBACK_HW_I2S_DataTypeGet

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_I2S_DataTypeGet ( sf_audio_playback_hw_ctrl_t
*const p_ctrl ,  sf_audio_playback_data_type_t  *const p_data_type )
```

7.5.7.1 Brief description

Provides the expected data type in the pointer p_data_type.

7.5.7.2 Detailed description

Table 514:Return values

Name	Description
SSP_SUCCESS	Data type stored in p_data_type.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_data_type is NULL.

NOTE: This function is reentrant if the lower level driver functions are reentrant.

7.5.7.3 Function steps

- Store data type. The audio framework supports only 16-bit signed data.

7.5.8 SF_AUDIO_PLAYBACK_HW_I2S_Close

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_I2S_Close ( sf_audio_playback_hw_ctrl_t
*const p_api_ctrl )
```

7.5.8.1 Brief description

Close open audio driver.

7.5.8.2 Detailed description

Table 515:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.

See Common Error Codes and lower level driver function for other possible return codes. This function calls:

- close

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

7.5.8.3 Function steps

- Close I2S driver.

7.5.9 SF_AUDIO_PLAYBACK_HW_I2S_VersionGet

```
ssp_err_t SF_AUDIO_PLAYBACK_HW_I2S_VersionGet ( ssp_version_t
*const p_version )
```

7.5.9.1 Brief description

Stores the version of the firmware and API in provided pointer p_version.

7.5.9.2 Detailed description

Table 516:Return values

Name	Description
SSP_ERR_ASSERTION	The parameter p_version is NULL.
SSP_SUCCESS	Module version successfully stored in p_version.

NOTE: This function is reentrant.

7.5.10 Extensions

7.5.10.1 sf_audio_playback_hw_i2s_instance_ctrl_t

sf_audio_playback_hw_i2s_instance_ctrl_t

Detailed description

Hardware dependent control block for I2S audio driver.

Variables

- void(* p_callback)(*p_args)

  Callback called when play is complete.
- void * p_context

  Placeholder for user data. Passed to the user callback in sf_audio_playback_hw_callback_args_t.
- i2s_instance_t const * p_lower_lvl_i2s

  I2S API used to access I2S hardware.

7.5.10.2 sf_audio_playback_hw_i2s_cfg_t

sf_audio_playback_hw_i2s_cfg_t

Detailed description

Hardware dependent configuration for I2S audio driver.

Variables

- i2s_instance_t const * p_lower_lvl_i2s

  I2S API used to access I2S hardware.

7.6 ADC Audio Recording Framework

RTOS-integrated ADC implementation of Audio Recording Interface.

The Audio Recording Framework implementation uses the ADC periodic interface for audio recording.

7.6.1 Functions

- SF_AUDIO_RECORD_ADC_Open
- SF_AUDIO_RECORD_ADC_Close
- SF_AUDIO_RECORD_ADC_Start

- SF_AUDIO_RECORD_ADC_Stop
- SF_AUDIO_RECORD_ADC_InfoGet
- SF_AUDIO_RECORD_ADC_VersionGet

7.6.2 Defines

- #define SF_AUDIO_RECORD_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file
- #define SF_AUDIO_RECORD_CODE_VERSION_MINOR

  Initial value: `(3U)`

7.6.3 SF_AUDIO_RECORD_ADC_Open

```
ssp_err_t SF_AUDIO_RECORD_ADC_Open ( sf_audio_record_ctrl_t  *const p_api_ctrl ,
 sf_audio_record_cfg_t  const *const p_cfg )
```

7.6.3.1 Brief description

Configure the ADC with user configurations.

7.6.3.2 Detailed description

The SF_AUDIO_RECORD_ADC_Open will initialize the configurations for the underlying ADC periodic framework. Implements

- open.

Table 517:Return values

Name	Description
SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_cfg is NULL. Or any one of the following p_cfg parameter is NULL/zero. p_cfg->p_capture_data_buffer, p_cfg->sample_count, p_cfg->capture_data_buffer_size, p_cfg->p_callback, p_cfg->sampling_rate_hz. Or resolution or time period is not matching. for other possible causes.
SSP_ERR_IN_USE	The channel specified has already been opened.

See Common Error Codes or sf_adc_periodic for other possible return codes or causes. This function calls

- open

NOTE: This function is reentrant for any unit.

7.6.3.3 Function steps

- Initialize the ADC periodic framework
- Initialize the configuration parameters for ADC periodic configuration structure
- Configure the ADC resolution depending on the data width in cfg
- Call the underlying ADC periodic open

7.6.4 SF_AUDIO_RECORD_ADC_Close

```
ssp_err_t SF_AUDIO_RECORD_ADC_Close ( sf_audio_record_ctrl_t
*const p_api_ctrl )
```

7.6.4.1 Brief description

Call the ADC periodic framework Close.

7.6.4.2 Detailed description

Implements

- close.

Table 518:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Control block p_ctrl is not initialized. Call SF_AUDIO_RECORD_ADC_Open to configure.

See Common Error Codes or sf_adc_periodic for other possible return codes or causes. This function calls

- close

7.6.4.3 Function steps

- Call the underlying ADC periodic close

7.6.5 SF_AUDIO_RECORD_ADC_Start

```
ssp_err_t SF_AUDIO_RECORD_ADC_Start ( sf_audio_record_ctrl_t
*const p_api_ctrl )
```

7.6.5.1 Brief description

Call the ADC periodic framework Start.

7.6.5.2 Detailed description

Implements

- start.

Table 519:Return values

Name	Description
SSP_SUCCESS	ADC Periodic Scan started successfully.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Control block p_ctrl is not initialized. Call SF_AUDIO_RECORD_ADC_Open to configure.

See Common Error Codes or sf_adc_periodic for other possible return codes or causes. This function calls

- start

7.6.5.3 Function steps

- Call the underlying ADC periodic start

7.6.6 SF_AUDIO_RECORD_ADC_Stop

```
ssp_err_t SF_AUDIO_RECORD_ADC_Stop ( sf_audio_record_ctrl_t  *const p_api_ctrl )
```

7.6.6.1 Brief description

Call the ADC periodic framework Stop.

7.6.6.2 Detailed description

Implements

- stop.

Table 520:Return values

Name	Description
SSP_SUCCESS	Periodic ADC scan stopped successfully.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Control block p_ctrl is not initialized. Call SF_AUDIO_RECORD_ADC_Open to configure.

See Common Error Codes or sf_adc_periodic for other possible return codes or causes. This function calls

- stop

7.6.6.3 Function steps

- Call the underlying ADC periodic start

7.6.7 SF_AUDIO_RECORD_ADC_InfoGet

```
ssp_err_t SF_AUDIO_RECORD_ADC_InfoGet ( sf_audio_record_ctrl_t
*const p_api_ctrl ,   sf_audio_record_info_t * p_info )
```

7.6.7.1 Brief description

Provide information about the channel supported by audio recording framework(MONO)

7.6.7.2 Detailed description

Implements

- infoGet.

Table 521:Return values

Name	Description
SSP_SUCCESS	InfoGet returns successfully.
SSP_ERR_ASSERTION	p_ctrl or p_info is null.

Table 521:Return values (Continued)

Name	Description
SSP_ERR_NOT_OPEN	Control block p_ctrl is not initialized. Call SF_AUDIO_RECORD_ADC_Open to configure.

See Common Error Codes or sf_adc_periodic for other possible return codes or causes.

7.6.8 SF_AUDIO_RECORD_ADC_VersionGet

```
ssp_err_t SF_AUDIO_RECORD_ADC_VersionGet ( ssp_version_t  *const p_version )
```

7.6.8.1 Brief description

Gets version and stores it in provided pointer p_version.

7.6.8.2 Detailed description

Implements

- versionGet.

Table 522:Return values

Name	Description
SSP_SUCCESS	VersionGet returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was null.

7.6.9 Extensions

7.6.9.1 sf_audio_record_adc_instance_ctrl_t

sf_audio_record_adc_instance_ctrl_t

Detailed description

Control block for audio recording Initialization occurs when open is called

Variables

- uint32_t open

  Used by driver to check if pointer to control block is valid

- TX_MUTEX mutex

  Mutex used to protect access to lower level driver hardware registers
- void * p_capture_data_buffer

  Pointer to the buffer that will store the samples
- uint32_t sample_count

  Samples per channel to be buffered before notifying the app
- void(* p_callback)(*p_args)

  Callback function.
- void const * p_context

  Placeholder for user data.
- sf_adc_periodic_instance_t const * p_lower_lvl_adc_periodic

  Lower level ADC periodic instance.

7.6.9.2 sf_audio_record_adc_hw_cfg_t

sf_audio_record_adc_hw_cfg_t

Detailed description

Variables

- sf_adc_periodic_instance_t const * p_lower_lvl_adc_periodic

7.7 I2S Audio Recording Framework

I2S implementation of Audio Recording Interface.

The Audio Recording Framework implementation uses the I2S interface for audio recording.

7.7.1 Functions

- SF_AUDIO_RECORD_I2S_Open
- SF_AUDIO_RECORD_I2S_Start
- SF_AUDIO_RECORD_I2S_Stop
- SF_AUDIO_RECORD_I2S_InfoGet
- SF_AUDIO_RECORD_I2S_Close
- SF_AUDIO_RECORD_I2S_VersionGet

7.7.2 Defines

- #define SF_AUDIO_RECORD_I2S_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file

- #define SF_AUDIO_RECORD_I2S_CODE_VERSION_MINOR

  Initial value: `(0U)`

7.7.3 SF_AUDIO_RECORD_I2S_Open

```
ssp_err_t SF_AUDIO_RECORD_I2S_Open ( sf_audio_record_ctrl_t  *const p_api_ctrl ,
 sf_audio_record_cfg_t  const *const p_cfg )
```

7.7.3.1 Brief description

Configures the Audio record I2S framework and I2S HAL driver The SF_AUDIO_RECORD_I2S_Open function acquires a mutex for the I2S Unit used, then calls the driver open function. The mutex is released following the driver layer open function.

7.7.3.2 Detailed description

Implements

- open.

Table 523:Return values

Name	Description
SSP_SUCCESS	Configuration of lower level drivers completed successfully.
SSP_ERR_ASSERTION	One of the following parameter is null: p_ctrl or p_cfg or p_cfg_extend->p_lower_lvl_i2s or p_cfg_extend->p_lower_lvl_i2s->p_api.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use a mutex.

See Common Error Codes and lower level driver function for other possible return codes. This function calls:

- open

7.7.3.3 Function steps

- Open I2S module

- Create a mutex
- Get mutex before calling lower layer, which will access HW registers.
- Initialize the configuration parameters
- Post the Mutex
- If any of the HAL layer initializations failed, then delete the mutex and exit the function with the error code
- Delete the mutex
- If there was a mutex error, return it
- Mark control block open so other tasks know it is valid

7.7.4 SF_AUDIO_RECORD_I2S_Start

```
ssp_err_t SF_AUDIO_RECORD_I2S_Start ( sf_audio_record_ctrl_t
*const p_api_ctrl )
```

7.7.4.1 Brief description

Gets mutex, starts recording data into the buffer.

7.7.4.2 Detailed description

Implements

- start.

Table 524:Return values

Name	Description
SSP_SUCCESS	I2S record started successfully.
SSP_ERR_ASSERTION	p_ctrl or p_ctrl->p_api is NULL.
SSP_ERR_NOT_OPEN	Driver control block not valid.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use a mutex.
SSP_ERR_IN_USE	The module is currently busy performing another operation

See Common Error Codes and lower level driver function for other possible return codes. This function calls:

- read

7.7.4.3 Function steps

- Get mutex, start i2s
- Call the underlying I2S driver read
- If any of the HAL layer initializations failed, then delete the mutex and exit the function with the error code
- Return the mutex
- If there was a mutex error, return it

7.7.5 SF_AUDIO_RECORD_I2S_Stop

```
ssp_err_t SF_AUDIO_RECORD_I2S_Stop ( sf_audio_record_ctrl_t  *const p_api_ctrl )
```

7.7.5.1 Brief description

Stops the audio recording and releases mutex.

7.7.5.2 Detailed description

Implements

- stop.

Table 525:Return values

Name	Description
SSP_SUCCESS	I2S Record stopped successfully.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Control block p_ctrl is not initialized.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use a mutex.

See Common Error Codes for other possible return codes or causes. This function calls: stop

7.7.5.3 Function steps

- Return the mutex
- If there was a mutex error, return it

7.7.6 SF_AUDIO_RECORD_I2S_InfoGet

```
ssp_err_t SF_AUDIO_RECORD_I2S_InfoGet ( sf_audio_record_ctrl_t
*const p_api_ctrl ,   sf_audio_record_info_t * p_info )
```

7.7.6.1 Brief description

Provide information about the channel supported by audio recording framework(stereo)

7.7.6.2 Detailed description

Implements

- infoGet.

Table 526:Return values

Name	Description
SSP_SUCCESS	InfoGet returns successfully.
SSP_ERR_ASSERTION	p_ctrl or p_info is null.
SSP_ERR_NOT_OPEN	Control block p_ctrl is not initialized.

See Common Error Codes for other possible return codes or causes.

7.7.7 SF_AUDIO_RECORD_I2S_Close

```
ssp_err_t SF_AUDIO_RECORD_I2S_Close ( sf_audio_record_ctrl_t
*const p_api_ctrl )
```

7.7.7.1 Brief description

Closes I2S driver, releases and deletes the mutex.

7.7.7.2 Detailed description

Implements

- close.

Table 527:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Control block p_ctrl is not initialized.

See Common Error Codes and lower level driver function for other possible return codes or causes. This function calls

- close

7.7.7.3 Function steps

- Call the underlying I2S close
- Post the mutex
- If all the HAL layers closed successfully, then delete the mutex and mark module as un-initialized
- Clear information from control block so other functions know this instance is closed

7.7.8 SF_AUDIO_RECORD_I2S_VersionGet

```
ssp_err_t SF_AUDIO_RECORD_I2S_VersionGet ( ssp_version_t  *const p_version )
```

7.7.8.1 Brief description

Gets version and stores it in provided pointer p_version.

7.7.8.2 Detailed description

Implements

- versionGet.

Table 528:Return values

Name	Description
SSP_SUCCESS	VersionGet returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was null.

7.7.9 Extensions

7.7.9.1 sf_audio_record_i2s_instance_ctrl_t

sf_audio_record_i2s_instance_ctrl_t

Detailed description

Control block for audio recording Initialization occurs when open is called

Variables

- uint32_t open

  Used by driver to check if pointer to control.
- TX_MUTEX mutex

  Mutex used to protect access to lower level driver hardware registers.
- void * p_capture_data_buffer

  Pointer to the buffer record buffer */.
- uint32_t capture_data_size

  capture data type
- uint32_t data_size

  Number of bytes captured for each iteration.
- uint32_t buffer_size

  size of the current record buffer */
- uint32_t current_buffer_index

  Index into current buffer */.
- void(* p_callback)(*p_args)

  Callback function.
- void const * p_context

  Placeholder for user data.
- i2s_instance_t const * p_lower_lvl_i2s

  Lower level I2S instance.

7.7.9.2 sf_audio_record_i2s_hw_cfg_t

sf_audio_record_i2s_hw_cfg_t

Detailed description

Variables

- i2s_instance_t const * p_lower_lvl_i2s

7.8 QSPI Block Media Framework

RTOS-integrated Block Media framework for QSPI driver.

7.8.1 Functions

- sf_block_media_qspi_initialize
- sf_block_media_qspi_erase
- sf_block_media_qspi_program
- SF_BLOCK_MEDIA_QSPI_Open
- SF_BLOCK_MEDIA_QSPI_Read
- SF_BLOCK_MEDIA_QSPI_Write
- SF_BLOCK_MEDIA_QSPI_Control
- SF_BLOCK_MEDIA_QSPI_Close
- SF_BLOCK_MEDIA_QSPI_VersionGet

7.8.2 Variables

- g_block_media_qspi_version
- g_sf_block_media_on_sf_block_media_qspi

7.8.3 Defines

- #define SF_BLOCK_MEDIA_QSPI_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define SF_BLOCK_MEDIA_QSPI_CODE_VERSION_MINOR

  Initial value: `(0U)`
- #define SF_QSPI_OPEN

  Initial value: `(0x51535049U)`

  "QSPI" in ASCII, used to identify general SF_BLOCK_MEDIA_QSPI control block
- #define QSPI_FLASH_BASE_ADDRESS

  Initial value:`BSP_PRV_QSPI_DEVICE_PHYSICAL_ADDRESS`
- #define IN_PROGRESS

  Initial value:`true`

- #define SF_BLOCK_MEDIA_QSPI_ERROR_RETURN

  Initial value:`#define SSP_ERROR_RETURN((a), (err), &g_module_name[0], &g_block_media_qspi_version)`

  Macro for error logger.

7.8.4 sf_block_media_qspi_initialize

```
ssp_err_t sf_block_media_qspi_initialize ( sf_block_media_qspi_instance_ctrl_t *const p_ctrl , sf_block_media_cfg_t const *const p_cfg )
```

7.8.4.1 Brief description

Acquires mutex, then handles driver initialization at the HAL layer. This function releases the mutex before returns to the caller.

7.8.4.2 Detailed description

Table 529:Return values

Name	Description
SSP_SUCCESS	Block Media driver for QSPI is successfully opened.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- open

7.8.4.3 Function steps

- Calling low level driver

7.8.5 sf_block_media_qspi_erase

```
ssp_err_t sf_block_media_qspi_erase ( sf_block_media_qspi_instance_ctrl_t * p_ctrl , uint8_t * p_device_address , uint32_t const block_count )
```

7.8.5.1 Brief description

This is the subroutine for the SF_BLOCK_MEDIA_QSPI_Write API.

7.8.5.2 Detailed description

Table 530:Return values

Name	Description
SSP_SUCCESS	No parameter error found

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- erase

7.8.5.3 Function steps

- Erase the block before writing
- Wait until operation is in progress
- Calculate next sub-sector address

7.8.6 sf_block_media_qspi_program

```
ssp_err_t sf_block_media_qspi_program ( sf_block_media_qspi_instance_ctrl_t
* p_ctrl ,   uint8_t * p_device_address ,   uint8_t * p_buffer ,
 qspi_info_t  qspi_info ,   uint32_t const block_count )
```

7.8.6.1 Brief description

This is the subroutine for the SF_BLOCK_MEDIA_QSPI_Write API.

7.8.6.2 Detailed description

Table 531:Return values

Name	Description
SSP_SUCCESS	No parameter error found

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- pageProgram

7.8.6.3 Function steps

- Write data into the block media QSPI flash

7.8.7 SF_BLOCK_MEDIA_QSPI_Open

```
ssp_err_t SF_BLOCK_MEDIA_QSPI_Open ( sf_block_media_ctrl_t  *const p_api_ctrl ,
 sf_block_media_cfg_t  const *const p_cfg )
```

7.8.7.1 Brief description

Open Block Media QSPI flash device for read/write and control. Parameter checking and Acquires mutex, then handles driver initialization at the HAL QSPI layer and marking the open flag in control block.

7.8.7.2 Detailed description

Name of module used by error logger macro

Table 532:Return values

Name	Description
SSP_SUCCESS	Block media for QSPI framework is successfully opened.
SSP_ERR_ASSERTION	One of the following parameters may be null: p_api_ctrl, p_cfg or configuration for qspi.
SSP_ERR_IN_USE	The channel specified has already been opened. or the mutex may be unavailable for the the device. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- open

7.8.7.3 Function steps

- On success populate the control structure
- Selecting smallest block erasable by underlying QSPI flash

7.8.8 SF_BLOCK_MEDIA_QSPI_Read

```
ssp_err_t SF_BLOCK_MEDIA_QSPI_Read ( sf_block_media_ctrl_t  *const p_api_ctrl ,
  uint8_t *const p_dest ,   uint32_t const start_block ,   uint32_t
const block_count )
```

7.8.8.1 Brief description

Read requested data from Block Media QSPI Flash through QSPI channel.

7.8.8.2 Detailed description

Table 533:Return values

Name	Description
SSP_SUCCESS	QSPI data read successfully.
SSP_ERR_ASSERTION	One of the following parameters may be null: p_api_ctrl or p_dest.
SSP_ERR_NOT_OPEN	Block media QSPI Framework module is not yet initialized.
SSP_ERR_IN_USE	The channel specified has already been opened. or the mutex may be unavailable for the the device. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- read

7.8.8.3 Function steps

- Obtain mutex before making HAL-level driver call.
- Read data from block media QSPI flash
- Release mutex

7.8.9 SF_BLOCK_MEDIA_QSPI_Write

```
ssp_err_t SF_BLOCK_MEDIA_QSPI_Write ( sf_block_media_ctrl_t  *const p_api_ctrl ,
  uint8_t const *const p_src ,   uint32_t const start_block ,   uint32_t
const block_count )
```

7.8.9.1 Brief description

Program requested data content to the Block Media QSPI flash memory.

7.8.9.2 Detailed description

Table 534:Return values

Name	Description
SSP_SUCCESS	Flash write finished successfully.
SSP_ERR_ASSERTION	One of the following parameters may be null: p_api_ctrl or p_src.
SSP_ERR_NOT_OPEN	Block media QSPI Framework module is not yet initialized.
SSP_ERR_IN_USE	The channel specified has already been opened. or the mutex may be unavailable for the the device. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- pageProgram

7.8.9.3 Function steps

- Obtain mutex before making HAL-level driver call.
- Get the information of Flash
- Calculate the address of flash
- Calculate the number of iteration required to complete write operation
- Write data into the block media QSPI flash
- Release mutex

7.8.10 SF_BLOCK_MEDIA_QSPI_Control

```
ssp_err_t SF_BLOCK_MEDIA_QSPI_Control ( sf_block_media_ctrl_t
*const p_api_ctrl ,  ssp_command_t  const command ,   void * p_data )
```

7.8.10.1 Brief description

Send control commands to and receive status of flash.

7.8.10.2 Detailed description

Table 535:Return values

Name	Description
SSP_SUCCESS	Command executed successfully.
SSP_ERR_ASSERTION	One of the following parameters may be null: p_api_ctrl.
SSP_ERR_NOT_OPEN	Block media QSPI Framework module is not yet initialized.
SSP_ERR_UNSUPPORTED	Command not support.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- infoGet

7.8.10.3 Function steps

- Get the information of Flash

7.8.11 SF_BLOCK_MEDIA_QSPI_Close

```
ssp_err_t SF_BLOCK_MEDIA_QSPI_Close ( sf_block_media_ctrl_t  *const p_api_ctrl )
```

7.8.11.1 Brief description

Close functionality will delete the resources which is initialized in open call.

7.8.11.2 Detailed description

Table 536:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	Parameters may be null: p_api_ctrl.
SSP_ERR_NOT_OPEN	Block media QSPI Framework module is not yet initialized.

Table 536:Return values (Continued)

Name	Description
SSP_ERR_IN_USE	The channel specified has already been opened. or the mutex may be unavailable for the the device. See HAL driver for other possible causes.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- close

7.8.11.3 Function steps

- Delete RTOS services allocated during the open call.

7.8.12 SF_BLOCK_MEDIA_QSPI_VersionGet

```
ssp_err_t SF_BLOCK_MEDIA_QSPI_VersionGet ( ssp_version_t  *const p_version )
```

7.8.12.1 Brief description

Get the firmware and API version of Block Media QSPI Framework.

7.8.12.2 Detailed description

Table 537:Return values

Name	Description
SSP_SUCCESS	Function executed successfully.
SSP_ERR_ASSERTION	Null Pointer.

NOTE: This function is reentrant.

7.8.13 g_block_media_qspi_version

```
ssp_version_t::g_block_media_qspi_version
```

7.8.13.1 Detailed description

Version data structure used by error logger macro.

7.8.14 g_sf_block_media_on_sf_block_media_qspi

sf_block_media_api_t::g_sf_block_media_on_sf_block_media_qspi

7.8.14.1 Detailed description

Block Media QSPI function pointers

7.8.14.2 Initialized as

```
g_sf_block_media_on_sf_block_media_qspi=
{
        .open       = SF_BLOCK_MEDIA_QSPI_Open,
        .read       = SF_BLOCK_MEDIA_QSPI_Read,
        .write      = SF_BLOCK_MEDIA_QSPI_Write,
        .ioctl      = SF_BLOCK_MEDIA_QSPI_Control,
        .close      = SF_BLOCK_MEDIA_QSPI_Close,
        .versionGet = SF_BLOCK_MEDIA_QSPI_VersionGet
}
```

7.8.15 Extensions

7.8.15.1 sf_block_media_on_qspi_cfg_t

sf_block_media_on_qspi_cfg_t

Detailed description

Variables

- qspi_instance_t const *const p_lower_lvl_qspi

  Pointer to QSPI instance structure.

7.8.15.2 sf_block_media_qspi_instance_ctrl_t

sf_block_media_qspi_instance_ctrl_t

Detailed description

QSPI block media instance control block.

Variables

- uint32_t open

  Used to determine if framework is initialized.

- uint32_t block_size

  Block size in bytes.
- qspi_instance_t * p_lower_lvl_qspi

  Pointer to QSPI instance structure.
- TX_MUTEX mutex

  Mutex used to protect access to lower level driver hardware registers.

7.9 RAM Block Media Framework

RTOS-integrated Block Media framework for RAM.

7.9.1 Functions

- SF_BLOCK_MEDIA_RAM_Open
- SF_BLOCK_MEDIA_RAM_Read
- SF_BLOCK_MEDIA_RAM_Write
- SF_BLOCK_MEDIA_RAM_Control
- SF_BLOCK_MEDIA_RAM_Close
- SF_BLOCK_MEDIA_RAM_VersionGet

7.9.2 Defines

- #define SF_BLOCK_MEDIA_RAM_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define SF_BLOCK_MEDIA_RAM_CODE_VERSION_MINOR

  Initial value: `(0U)`

7.9.3 SF_BLOCK_MEDIA_RAM_Open

```
ssp_err_t SF_BLOCK_MEDIA_RAM_Open ( sf_block_media_ctrl_t  *const p_api_ctrl ,
 sf_block_media_cfg_t  const *const p_cfg )
```

7.9.3.1 Brief description

Open device for read/write and control.

7.9.3.2 Detailed description

Table 538:Return values

Name	Description
SSP_SUCCESS	RAM buffer is available and is now open for read, write, and control access.
SSP_ERR_ASSERTION	p_api_ctrl, p_cfg, p_cfg->p_extend, or p_block_media_cfg->p_ram_buffer is NULL.
SSP_ERR_IN_USE	Framework is already open by someone else

7.9.3.3 Function steps

- Check if Framework is already in USE
- Copy the block size and RAM buffer memory address to control structure for further operation
- Mark device as a open

7.9.4 SF_BLOCK_MEDIA_RAM_Read

```
ssp_err_t SF_BLOCK_MEDIA_RAM_Read ( sf_block_media_ctrl_t  *const p_api_ctrl ,
  uint8_t *const p_dest ,   uint32_t const start_block ,   uint32_t
const block_count )
```

7.9.4.1 Brief description

Read data from RAM buffer.

7.9.4.2 Detailed description

Table 539:Return values

Name	Description
SSP_SUCCESS	Data read successfully.
SSP_ERR_ASSERTION	p_api_ctrl, p_dest or internal control block element is NULL.
SSP_ERR_NOT_OPEN	The Framework is not opened.
SSP_ERR_INVALID_BLOCKS	Invalid block passed to API

7.9.4.3 Function steps

- Check if Framework is NOT open
- Copy the content of p_ram_buffer into destination

7.9.5 SF_BLOCK_MEDIA_RAM_Write

```
ssp_err_t SF_BLOCK_MEDIA_RAM_Write ( sf_block_media_ctrl_t  *const p_api_ctrl ,
  uint8_t const *const p_src ,   uint32_t const start_block ,   uint32_t
const block_count )
```

7.9.5.1 Brief description

Write data to RAM buffer.

7.9.5.2 Detailed description

Table 540:Return values

Name	Description
SSP_SUCCESS	Data write successfully.
SSP_ERR_ASSERTION	p_api_ctrl, p_src or internal control block element is NULL.
SSP_ERR_NOT_OPEN	Framework is not opened.
SSP_ERR_INVALID_BLOCKS	Invalid block passed to API

7.9.5.3 Function steps

- Check if Framework is NOT open
- Copy the content of source into p_ram_buffer

7.9.6 SF_BLOCK_MEDIA_RAM_Control

```
ssp_err_t SF_BLOCK_MEDIA_RAM_Control ( sf_block_media_ctrl_t
*const p_api_ctrl ,  ssp_command_t  const command ,   void * p_data )
```

7.9.6.1 Brief description

Send control commands to and receive status of RAM buffer.

7.9.6.2 Detailed description

Table 541:Return values

Name	Description
SSP_SUCCESS	Command executed successfully.
SSP_ERR_ASSERTION	p_api_ctrl or p_data is NULL.
SSP_ERR_NOT_OPEN	Framework is not opened.
SSP_ERR_UNSUPPORTED	Command not supported.

7.9.6.3 Function steps

- Check if Framework is NOT open

7.9.7 SF_BLOCK_MEDIA_RAM_Close

```
ssp_err_t SF_BLOCK_MEDIA_RAM_Close ( sf_block_media_ctrl_t  *const p_api_ctrl )
```

7.9.7.1 Brief description

Close the Framework.

7.9.7.2 Detailed description

Table 542:Return values

Name	Description
SSP_SUCCESS	RAM buffer is available and is now open for read, write, and control access.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Framework is not opened.

7.9.7.3 Function steps

- Check if Framework is NOT open
- Mark Framework as close

7.9.8 SF_BLOCK_MEDIA_RAM_VersionGet

```
ssp_err_t SF_BLOCK_MEDIA_RAM_VersionGet ( ssp_version_t  *const p_version )
```

7.9.8.1 Brief description

Get version of Block Media RAM framework.

7.9.8.2 Detailed description

Return the version of the firmware and API.

Table 543:Return values

Name	Description
SSP_ERR_ASSERTION	p_version is NULL.
SSP_SUCCESS	version read successfully.

NOTE: This function is reentrant.

7.9.9 Extensions

7.9.9.1 sf_block_media_on_ram_cfg_t

sf_block_media_on_ram_cfg_t

Detailed description
Variables

- uint8_t * p_ram_buffer

  Pointer to RAM buffer address.
- uint32_t ram_buffer_size

  Size of RAM buffer.

7.9.9.2 sf_block_media_ram_instance_ctrl_t

sf_block_media_ram_instance_ctrl_t

Detailed description

RAM block media instance control block.

Variables

- uint8_t * p_ram_buffer

  pointer to RAM buffer address
- uint32_t block_size

  Block size in bytes.
- uint32_t open

  Used to determine if framework is initialized.
- uint32_t ram_buffer_size

  Size of RAM buffer.

7.10 SDMMC Block Media Framework

RTOS-integrated Block Media framework for SDMMC driver.

7.10.1 Functions

- SF_Block_Media_SDMMC_Open
- SF_Block_Media_SDMMC_Read
- SF_Block_Media_SDMMC_Write
- SF_Block_Media_SDMMC_Control
- SF_Block_Media_SDMMC_Close
- SF_Block_Media_SDMMC_VersionGet
- sf_block_media_sdmmc_callback

7.10.2 Defines

- #define BLOCK_MEDIA_SDMMC_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define BLOCK_MEDIA_SDMMC_CODE_VERSION_MINOR

  Initial value: `(5U)`

7.10.3 SF_Block_Media_SDMMC_Open

```
ssp_err_t SF_Block_Media_SDMMC_Open ( sf_block_media_ctrl_t  *const p_api_ctrl ,
 sf_block_media_cfg_t  const *const p_cfg )
```

7.10.3.1 Brief description

Open device for read/write and control.

7.10.3.2 Detailed description

Open an SD or MMC device port for read/write and control. This function initializes the SDMMC driver and hardware the first time it is called out of reset.

Table 544:Return values

Name	Description
SSP_SUCCESS	Port is available and is now open for read, write, and control access.
SSP_ERR_ASSERTION	p_ctrl, p_cfg, p_block_media_cfg or p_sdmmc is NULL.
SSP_ERR_INTERNAL	OS service call fails.
SSP_ERR_IN_USE	The channel specified has already been opened. No configurations were changed. Call the associated Close function or use associated Control commands to reconfigure the channel.

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- open

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

7.10.3.3 Function steps

- Create SDMMC event flag and put it into context
- Mark the stream as open by initializing "BMSO" in its ASCII equivalent.
- Cleanup before logging the error

7.10.4 SF_Block_Media_SDMMC_Read

```
ssp_err_t SF_Block_Media_SDMMC_Read ( sf_block_media_ctrl_t  *const p_api_ctrl ,
  uint8_t *const p_dest ,   uint32_t const start_block ,   uint32_t
const block_count )
```

7.10.4.1 Brief description

Read data from SD/MMC.

7.10.4.2 Detailed description

Read data from an SD or MMC device port.

Table 545:Return values

Name	Description
SSP_SUCCESS	Data read successfully.
SSP_ERR_ASSERTION	p_ctrl, p_sdmmc or p_dest is NULL.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_INTERNAL	OS service call fails.
SSP_ERR_READ_FAILED	Data read failed.

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- read

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

7.10.4.3 Function steps

- Wait until read operation is completed. Event is signaled in event flag object.

7.10.5 SF_Block_Media_SDMMC_Write

```
ssp_err_t SF_Block_Media_SDMMC_Write ( sf_block_media_ctrl_t
*const p_api_ctrl ,   uint8_t const *const p_src ,   uint32_t const start_block ,
  uint32_t const block_count )
```

7.10.5.1 Brief description

Write data to SDMMC channel.

7.10.5.2 Detailed description

Table 546:Return values

Name	Description
SSP_SUCCESS	Card write finished successfully.
SSP_ERR_ASSERTION	p_ctrl, p_sdmmc or p_src is NULL.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_INTERNAL	OS service call fails.
SSP_ERR_WRITE_FAILED	Data write failed.
SSP_ERR_WRITE_PROTECTED	SD or MMC card is Write Protected.

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- write

NOTE: This function is reentrant for different channels.

7.10.5.3 Function steps

- Wait until write operation is completed. Event is signaled in event flag object.

7.10.6 SF_Block_Media_SDMMC_Control

```
ssp_err_t SF_Block_Media_SDMMC_Control ( sf_block_media_ctrl_t
*const p_api_ctrl ,  ssp_command_t  const command ,   void * p_data )
```

7.10.6.1 Brief description

Send control commands to and receive status of SD/MMC port.

7.10.6.2 Detailed description

Send control commands to the SD/MMC port and receive the status of the SD/MMC port.

Table 547:Return values

Name	Description
SSP_SUCCESS	Command executed successfully.
SSP_ERR_ASSERTION	p_ctrl or p_sdmmc or p_data is Null.
SSP_ERR_NOT_OPEN	The channel is not opened.
SF_INFO_NOT_AVAILABLE	Information not available possibly because card has been removed or is defective.

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- control

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

7.10.7 SF_Block_Media_SDMMC_Close

```
ssp_err_t SF_Block_Media_SDMMC_Close ( sf_block_media_ctrl_t
*const p_api_ctrl )
```

7.10.7.1 Brief description

Close open device port.

7.10.7.2 Detailed description

Close an open SD/MMC device port.

Table 548:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_ctrl or p_sdmmc is NULL.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_INTERNAL	OS service call fails.

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- close

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

7.10.7.3 Function steps

- Mark control block as unused so it can be reconfigured.

7.10.8 SF_Block_Media_SDMMC_VersionGet

```
ssp_err_t SF_Block_Media_SDMMC_VersionGet ( ssp_version_t  *const p_version )
```

7.10.8.1 Brief description

Get version of Block Media SD/MMC driver.

7.10.8.2 Detailed description

Return the version of the firmware and API.

Table 549:Return values

Name	Description
SSP_ERR_ASSERTION	p_version is Pointer.
SSP_SUCCESS	version read successfully.

See Common Error Codes or lower level drivers for other possible return codes.

NOTE: This function is reentrant.

7.10.9 sf_block_media_sdmmc_callback

```
sf_block_media_sdmmc_callback ( sdmmc_callback_args_t  * pcb_arg )
```

7.10.9.1 Detailed description

SDMMC SSP framework level callback

Table 550:Parameters

Name	Direction	Description
pcb_arg	in	Pointer to callback parameters

Table 551:Return values

Name	Description
void	

7.10.9.2 Function steps

- Transfer complete event occurs wake up the suspended thread.

7.10.10 Extensions

7.10.10.1 sf_block_media_on_sdmmc_cfg_t

sf_block_media_on_sdmmc_cfg_t

Detailed description

Variables

- sdmmc_instance_t const *const p_lower_lvl_sdmmc

  Pointer to SDMMC instance structure.

7.10.10.2 sf_block_media_sdmmc_instance_ctrl_t

sf_block_media_sdmmc_instance_ctrl_t

Detailed description

SDMMC block media instance control block.

Variables

- uint32_t block_size

  Block size in bytes.

- sdmmc_instance_t * p_lower_lvl_sdmmc

  Pointer to SDMMC instance structure.
- TX_EVENT_FLAGS_GROUP eventflag

  Pointer to the event flag object for SDMMC data transfer.
- uint32_t open

  Used to determine if framework is initialized.

7.11 Cellular NSAL Implementation on NetX

Cellular NetX NSAL interface implementation header file.

7.11.1 Functions

- sf_cellular_nsal_netx_driver
- sf_cellular_nsal_ppp_send_byte
- sf_cellular_nsal_invalid_packet_handler

7.11.2 Defines

- #define SF_CELR_NSALNX_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file Major Version of code that implements the API defined in this file
- #define SF_CELR_NSALNX_CODE_VERSION_MINOR

  Initial value: `(1U)`

  Minor Version of code that implements the API defined in this file

7.11.3 sf_cellular_nsal_netx_driver

```
 sf_cellular_nsal_netx_driver (  NX_IP_DRIVER * driver_req_ptr ,
 sf_cellular_instance_t  const * p_cellular_instance ,  sf_cellular_nsal_cfg_t
* p_cellular_nsal_cfg )
```

7.11.3.1 Brief description

NetX IP Driver entry function.

7.11.3.2 Detailed description

Table 552:Parameters

Name	Direction	Description
driver_req_ptr	in	Pointer to NetX IP Driver
p_cellular_instance	in	Pointer to cellular framework instance
p_cellular_nsal_cfg	in	Pointer to cellular nsal configuration structure

NetX IP Driver entry function.

Table 553:Parameters

Name	Direction	Description
driver_req_ptr	in	NetX Driver Pointer
p_cellular_instance	in	Cellular Framework Instance
p_cellular_nsal_cfg	in	Cellular NSAL configuration

Table 554:Return values

Name	Description
void	

7.11.4 sf_cellular_nsal_ppp_send_byte

```
 sf_cellular_nsal_ppp_send_byte (  UCHAR byte ,  sf_cellular_instance_t  const
* p_celr_instance )
```

7.11.4.1 Brief description

PPP Send Byte Callback function call from PPP Stack.

7.11.4.2 Detailed description

NSAL PPP Send bytes callback API

Table 555:Parameters

Name	Direction	Description
byte	in	Byte to send
p_celr_instance	in	Pointer to cellular framework instance
byte	in	Byte to be send
p_celr_instance	in	Cellular instance pointer

7.11.4.3 Function steps

- Transmit byte

7.11.5 sf_cellular_nsal_invalid_packet_handler

```
sf_cellular_nsal_invalid_packet_handler (  NX_PACKET * p_packet_ptr ,
sf_cellular_instance_t  const * p_celr_instance )
```

7.11.5.1 Brief description

Cellular framework nsal invalid callback handler.

7.11.5.2 Detailed description

Table 556:Parameters

Name	Direction	Description
p_packet_ptr	in	Pointer to NetX Packet
p_celr_instance	in	Pointer to cellular framework instance

Cellular framework nsal invalid callback handler.

Table 557:Parameters

Name	Direction	Description
p_packet_ptr	in	Pointer to invalid PPP Packet received
p_celr_instance	in	Cellular instance pointer

7.11.5.3 Function steps

- variable to get Memory compare result
- Check whether packet contains "NO CARRIER" string, we are ignoring first and last 2 bytes of packet which are Carriage Return and Line Feed.
- check whether no carrier string present
- Disconnect Network
- Soft reset the module
- Provision the Module using callback
- Reestablish data connection
- Restart PPP
- Release the packet

7.12 Telnet Communication Framework

RTOS-integrated Communications Framework NetX telnet server implementation.

7.12.1 Functions

- SF_COMMS_TELNET_Open
- SF_COMMS_TELNET_Close
- SF_COMMS_TELNET_Read
- SF_COMMS_TELNET_Write
- SF_COMMS_TELNET_Lock
- SF_COMMS_TELNET_Unlock
- SF_COMMS_TELNET_VersionGet

7.12.2 Defines

- #define SF_COMMS_TELNET_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define SF_COMMS_TELNET_CODE_VERSION_MINOR

  Initial value: `(1U)`
- #define SF_COMMS_TELNET_INTERNAL_TIMEOUT

  Initial value: `(10)`
- #define SF_COMMS_TELNET_MAX_CONNECTION

  Initial value: `(4U)`
- #define SF_COMMS_TELNET_PACKET_SIZE_BYTES

  Initial value: `(1536U + 32U)`
- #define SF_COMMS_TELNET_PACKETS_IN_POOL_MEMORY

  Initial value: `(50U)`
- #define SF_COMMS_TELNET_PACKET_POOL_MEMORY_SIZE_BYTES

  Initial value: `((SF_COMMS_TELNET_PACKET_SIZE_BYTES + (uint32_t) sizeof(NX_PACKET)) * SF_COMMS_TELNET_PACKETS_IN_POOL_MEMORY)`
- #define SF_COMMS_TELNET_QUEUE_MEMORY_SIZE_BYTES

  Initial value: `(SF_COMMS_TELNET_PACKETS_IN_POOL_MEMORY * 4U )`
- #define SF_COMMS_TELNET_OPEN

  Initial value: `(0x434D544EU)`

  "CMTN" in ASCII, used to identify general timer handle

7.12.3 SF_COMMS_TELNET_Open

```
ssp_err_t SF_COMMS_TELNET_Open ( sf_comms_ctrl_t  *const p_api_ctrl ,
 sf_comms_cfg_t  const *const p_cfg )
```

7.12.3.1 Brief description

Initializes the Telnet server and other operating system resources.

7.12.3.2 Detailed description

Table 558:Return values

Name	Description
SSP_SUCCESS	comms Telnet instance opened successfully.
SSP_ERR_IN_USE	comms Telnet maximum connection limit reached.
SSP_ERR_ALREADY_OPEN	comms Telnet instance already open.
SSP_ERR_ASSERTION	Parameter check failed for one of the following: • Pointer p_api_ctrl is NULL. • Pointer p_cfg is NULL • Pointer p_cfg->p_extend is NULL • Pointer p_cfg_extend->p_telnet_server is NULL • Pointer p_cfg_extend->p_telnet_server is NULL • Telnet server field nx_telnet_server_id indicate this instance is not created already thus invlaid instance of Telnet server.
SSP_ERR_INTERNAL	An internal ThreadX Or NetX error has occurred. This is typically a failure to create/use thread mutex or failure create/enable an internal thread/feature for NetX service.

NOTE: - For Telnet server specific API details, refer Telnet server user manual.

NOTE: - For NetX specific API details, refer NetX user manual.

NOTE: - For ThreadX specific API details, refer ThreadX user manual.

NOTE: - This function is reentrant.

7.12.3.3 Function steps

- Initialize and start Telnet server and ThreadX resources for this connection
- Mark this connection initialized.

7.12.4 SF_COMMS_TELNET_Close

```
ssp_err_t SF_COMMS_TELNET_Close ( sf_comms_ctrl_t  *const p_api_ctrl )
```

7.12.4.1 Brief description

Disconnect Telnet server and clean up resources.

7.12.4.2 Detailed description

Table 559:Return values

Name	Description
SSP_SUCCESS	Connection successfully closed
SSP_ERR_ASSERTION	Parameter check failed for one of the following: • Pointer p_api_ctrl is NULL • p_ctrl->p_telnet_server is NULL • Telnet server field nx_telnet_server_id indicate this instance is not created already thus invlaid instance of Telnet server.
SSP_ERR_NOT_OPEN	Connection is not open

NOTE: - For Telnet server specific API details, refer Telnet server user manual.

NOTE: - For NetX specific API details, refer NetX user manual.

NOTE: - For ThreadX specific API details, refer ThreadX user manual.

NOTE: - This function is reentrant.

7.12.4.3 Function steps

- Check if connection is open
- Update connection record.
- Return connection not open if no record found
- Stop and release Telnet server and ThreadX resources for this connection
- Mark this connection uninitialized.

7.12.5 SF_COMMS_TELNET_Read

```
ssp_err_t SF_COMMS_TELNET_Read ( sf_comms_ctrl_t  *const p_api_ctrl ,   uint8_t
*const p_dest ,   uint32_t const bytes ,   UINT const timeout )
```

7.12.5.1 Brief description

Read data from the Telnet comms connection.

7.12.5.2 Detailed description

Table 560:Return values

Name	Description
SSP_SUCCESS	Data reception ends successfully.
SSP_ERR_ASSERTION	One of the following invalid parameter passed. • Pointer p_api_ctrl is NULL • Pointer p_dest is NULL • Invalid read length i.e. bytes value is zero
SSP_ERR_NOT_OPEN	Connection is not open
SSP_ERR_TIMEOUT	One of the following operation timed out. • 'Event flags get' timed out • 'Receive mutex get' timed out • 'Queue receive' timed out

Table 560:Return values (Continued)

Name	Description
SSP_ERR_INTERNAL	An internal ThreadX Or NetX error has occurred. This is typically a failure to create/use thread mutex or failure create/enable an internal thread/feature for NetX service.

NOTE: - For Telnet server specific API details, refer Telnet server user manual.

NOTE: - For NetX specific API details, refer NetX user manual.

NOTE: - For ThreadX specific API details, refer ThreadX user manual.

NOTE: - This API is reentrant.

7.12.5.3 Function steps

- Check if connection is open
- Check and wait for client to be connected with timeout.
- Get read lock.
- Get data from read queue.
- Release read lock.

7.12.6 SF_COMMS_TELNET_Write

```
ssp_err_t SF_COMMS_TELNET_Write ( sf_comms_ctrl_t  *const p_api_ctrl ,   uint8_t
const *const p_src ,   uint32_t const bytes ,   UINT const timeout )
```

7.12.6.1 Brief description

Write data to the Telnet comms connection.

7.12.6.2 Detailed description

Table 561:Return values

Name	Description
SSP_SUCCESS	Data transmission finished successfully.
SSP_ERR_ASSERTION	One of the following invalid parameter passed. • Pointer p_api_ctrl is NULL. • Pointer p_src is NULL. • Invalid write length i.e. bytes value is zero.
SSP_ERR_NOT_OPEN	Connection is not open
SSP_ERR_TIMEOUT	One of the following operation timed out. • 'Event flags get' timed out • 'Transmit mutex get' timed out
SSP_ERR_OUT_OF_MEMORY	Couldn't allocate pool memory for Telnet server
SSP_ERR_INTERNAL	An internal ThreadX Or NetX error has occurred. This is typically a failure to create/use thread mutex or failure create/enable an internal thread/feature for NetX service.

NOTE: - For Telnet server specific API details, refer Telnet server user manual.

NOTE: - For NetX specific API details, refer NetX user manual.

NOTE: - For ThreadX specific API details, refer ThreadX user manual.

NOTE: - This function is reentrant.

7.12.6.3 Function steps

- Check if connection is open
- Check and wait for client to be connected with timeout.
- Get write lock.
- Send data to the connected client.
- Release write lock.

7.12.7 SF_COMMS_TELNET_Lock

```
ssp_err_t SF_COMMS_TELNET_Lock ( sf_comms_ctrl_t  *const p_api_ctrl ,
 sf_comms_lock_t  lock_type ,   UINT timeout )
```

7.12.7.1 Brief description

Acquire lock type for the Telnet comms instance.

7.12.7.2 Detailed description

Table 562:Return values

Name	Description
SSP_SUCCESS	Acquired requested lock on given connection.
SSP_ERR_ASSERTION	Pointer p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Connection is not open.
SSP_ERR_TIMEOUT	Acquiring requested lock timed-out.

NOTE: - For Telnet server specific API details, refer Telnet server user manual.

NOTE: - For NetX specific API details, refer NetX user manual.

NOTE: - For ThreadX specific API details, refer ThreadX user manual.

7.12.7.3 Function steps

- Check if connection is open
- Get both lock if requested
- Else get the lock type requested

7.12.8 SF_COMMS_TELNET_Unlock

```
ssp_err_t SF_COMMS_TELNET_Unlock ( sf_comms_ctrl_t  *const p_api_ctrl ,
 sf_comms_lock_t  lock_type )
```

7.12.8.1 Brief description

Release lock type for the Telnet comms instance.

7.12.8.2 Detailed description

Table 563:Return values

Name	Description
SSP_SUCCESS	Released requested lock on given connection..
SSP_ERR_ASSERTION	Pointer p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Connection is not open.

NOTE: - For Telnet server specific API details, refer Telnet server user manual.

NOTE: - For NetX specific API details, refer NetX user manual.

NOTE: - For ThreadX specific API details, refer ThreadX user manual.

7.12.8.3 Function steps

- Check if connection is open

- Release Lock(s) on this connection.

7.12.9 SF_COMMS_TELNET_VersionGet

```
ssp_err_t SF_COMMS_TELNET_VersionGet ( ssp_version_t  *const p_version )
```

7.12.9.1 Brief description

Get driver version.

7.12.9.2 Detailed description

Table 564:Return values

Name	Description
SSP_SUCCESS	Operation successful
SSP_ERR_ASSERTION	p_version pointer is NULL

NOTE: - This function is reentrant.

7.12.10 Extensions

7.12.10.1 sf_comms_telnet_instance_ctrl_t

sf_comms_telnet_instance_ctrl_t

Detailed description

NetX Telnet server communications driver configuration

Variables

- uint32_t open
- NX_PACKET * p_current_packet
- uint32_t packet_index
- TX_MUTEX mutex[2]
- NX_PACKET_POOL pool
- uint8_t pool_memory[SF_COMMS_TELNET_PACKET_POOL_MEMORY_SIZE_BYTES]
- NX_TELNET_SERVER * p_telnet_server

- UINT logical_connection
- TX_EVENT_FLAGS_GROUP available

  Flag to tell if this connection is available or not.
- TX_QUEUE queue

  Queue of received bytes.
- uint8_t queue_memory[SF_COMMS_TELNET_QUEUE_MEMORY_SIZE_BYTES]

7.12.10.2 sf_comms_telnet_cfg_t

sf_comms_telnet_cfg_t

Detailed description

NetX Telnet server device communications driver configuration

Variables

- NX_TELNET_SERVER * p_telnet_server
- CHAR * p_telnet_server_name
- NX_IP * p_ip
- VOID * p_stack
- ULONG stack_size

7.13 Console Framework

RTOS-integrated Console Framework.

This is a ThreadX aware console framework implemented using the SSP communications framework.

7.13.1 Functions

- SF_CONSOLE_Open
- SF_CONSOLE_Close
- SF_CONSOLE_Parse
- SF_CONSOLE_Prompt
- SF_CONSOLE_Read
- SF_CONSOLE_Write
- SF_CONSOLE_ArgumentFind
- SF_CONSOLE_CallbackNextMenu
- SF_CONSOLE_VersionGet

7.13.2 Defines

- #define SF_CONSOLE_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file
- #define SF_CONSOLE_CODE_VERSION_MINOR

  Initial value: `(6U)`

7.13.3 SF_CONSOLE_Open

```
ssp_err_t SF_CONSOLE_Open ( sf_console_ctrl_t  *const p_api_ctrl ,
 sf_console_cfg_t  const *const p_cfg )
```

7.13.3.1 Detailed description

Table 565:Return values

Name	Description
SSP_SUCCESS	Console channel is successfully opened

See Common Error Codes or lower level drivers for other possible return codes.

NOTE: This function is reentrant for any channel.

7.13.3.2 Function steps

- Open UART driver
- Store echo configuration and initial menu in control block
- Prompt for input autostart is true

7.13.4 SF_CONSOLE_Close

```
ssp_err_t SF_CONSOLE_Close ( sf_console_ctrl_t  *const p_api_ctrl )
```

7.13.4.1 Detailed description

Table 566:Return values

Name	Description
SSP_SUCCESS	Console successfully closed

See Common Error Codes or lower level drivers for other possible return codes.

NOTE: This function is reentrant for any channel.

7.13.4.2 Function steps

- Close UART driver

7.13.5 SF_CONSOLE_Parse

```
ssp_err_t SF_CONSOLE_Parse ( sf_console_ctrl_t  *const p_api_ctrl ,
 sf_console_menu_t  const *const p_menu ,   uint8_t const *const p_input ,
  uint32_t const bytes )
```

7.13.5.1 Detailed description

Table 567:Return values

Name	Description
SSP_SUCCESS	Data parsed successfully, command found and callback called.
SSP_ERR_UNSUPPORTED	Command not found in the current menu.
SSP_ERR_ASSERTION	One or more input parameter pointers are invalid.

See Common Error Codes or lower level drivers for other possible return codes.

NOTE: This function is reentrant for any channel.

7.13.5.2 Function steps

- Print help menu if help command is entered
- Go back one menu if previous menu command is entered.
- Go back to root menu if root menu command is entered.
- Look for matching commands, call callback if command found.

7.13.6 SF_CONSOLE_Prompt

```
ssp_err_t SF_CONSOLE_Prompt ( sf_console_ctrl_t  *const p_api_ctrl ,
 sf_console_menu_t  const *const p_menu ,   UINT const timeout )
```

7.13.6.1 Detailed description

Table 568:Return values

Name	Description
SSP_SUCCESS	Received valid command and called callback
SSP_ERR_ASSERTION	p_ctrl is NULL

See Common Error Codes or lower level drivers for other possible return codes.

NOTE: This function is reentrant for any channel.

7.13.6.2 Function steps

- Update stored current menu pointer if a new pointer is specified.
- Print menu name followed by ">" to prompt for user input.
- Lock the console UART framework to reserve exclusive access until the command completes.
  NOTE: Transmission is only locked while the menu name is printed and while the input command is non-zero in length. This allow debug messages to print from other threads while echo is off or no input command has been entered.

  Wait for input
- Parse input and call associated user callback.
- Command is complete, so unlock comms reception.

7.13.7 SF_CONSOLE_Read

```
ssp_err_t SF_CONSOLE_Read ( sf_console_ctrl_t  *const p_api_ctrl ,   uint8_t
*const p_dest ,   uint32_t const bytes ,   uint32_t const timeout )
```

7.13.7.1 Detailed description

Table 569:Return values

Name	Description
SSP_SUCCESS	Data read completed successfully

See Common Error Codes or lower level drivers for other possible return codes.

NOTE: This function is reentrant for any channel.

7.13.7.2 Function steps

- Lock the communications framework reception until carriage return is received.
- Read one byte at a time, checking for carriage returns, backspace, delete, and escape codes.
- Unlock the communications framework reception

7.13.8 SF_CONSOLE_Write

```
ssp_err_t SF_CONSOLE_Write ( sf_console_ctrl_t  *const p_api_ctrl ,   uint8_t
const *const p_src ,   uint32_t const timeout )
```

7.13.8.1 Detailed description

Table 570:Return values

Name	Description
SSP_SUCCESS	Data write completed successfully

See Common Error Codes or lower level drivers for other possible return codes.

NOTE: This function is reentrant for any channel.

7.13.8.2 Function steps

- Write null terminated string. Calculate the length. If it isn't longer than the maximum, write the entire string to the console.

7.13.9 SF_CONSOLE_ArgumentFind

```
ssp_err_t SF_CONSOLE_ArgumentFind ( uint8_t const *const p_arg , uint8_t const
*const p_str , int32_t *const p_index , int32_t *const p_data )
```

7.13.9.1 Detailed description

Table 571:Return values

Name	Description
SSP_SUCCESS	Argument found successfully
SSP_ERR_ASSERTION	p_arg or p_str is NULL

See Common Error Codes or lower level drivers for other possible return codes.

NOTE: This function is reentrant for any channel.

7.13.9.2 Function steps

- Search for first letter match at beginning of word.
- If the input string matches the input argument, store the index of the character following the argument in p_index and the data at that index in p_data. Then return.

7.13.10 SF_CONSOLE_CallbackNextMenu

```
SF_CONSOLE_CallbackNextMenu ( sf_console_callback_args_t * p_args )
```

7.13.10.1 Detailed description

Callback provided to continue parsing the next menu down.

Table 572:Parameters

Name	Direction	Description
p_args	in	Pointer to callback arguments to use in the next menu.

7.13.10.2 Function steps

- Next level menu is passed in as the user context parameter
- Update current menu.
- Check to see if the next menu command was input in the remaining string.

7.13.11 SF_CONSOLE_VersionGet

```
ssp_err_t SF_CONSOLE_VersionGet ( ssp_version_t  *const p_version )
```

7.13.11.1 Detailed description

Console version get function.

Table 573:Parameters

Name	Direction	Description
p_version	in	Version information stored here.

Table 574:Return values

Name	Description
SSP_SUCCESS	Version stored in provided pointer.
SSP_ERR_ASSERTION	p_version was null.

7.13.11.2 Function steps

- Set version pointer

7.13.12 Extensions

7.13.12.1 sf_console_instance_ctrl_t

sf_console_instance_ctrl_t

Detailed description

Console instance control block. DO NOT INITIALIZE. Initialization occurs when open is called

Variables

- sf_console_menu_t const * p_current_menu

  Current menu is stored here.
- sf_comms_instance_t const * p_comms

  Pointer to communications driver instance.
- uint8_t new_line

  Whether to echo input commands to transmitter.
- bool echo

  Whether to echo input commands to transmitter.
- uint8_t input[SF_CONSOLE_MAX_INPUT_LENGTH]

  Input buffer used to store user input.

7.14 FX_IO Framework

FileX adaptation layer for block media device drivers.

SF_EL_FX FileX I/O is a single entry function which adapts FileX to Renesas Synergy block media device drivers.

7.14.1 Summary

SF_EL_FX Has no API file.

7.14.2 Functions

- SF_EL_FX_BlockDriver
- check_partition_offset
- check_fat_boot_record

7.14.3 Defines

- #define SF_EL_FX_API_VERSION_MAJOR

  Initial value: `(1)`

  Version of the API defined in this file
- #define SF_EL_FX_API_VERSION_MINOR

  Initial value: `(0)`
- #define SF_EL_FX_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file
- #define SF_EL_FX_CODE_VERSION_MINOR

  Initial value: `(5U)`
- #define SF_EL_FX_55AA_SIGNATURE_OFFSET

  Initial value: `(0x1FEU)`

  SSP FileX Support.
- #define SF_EL_FX_JUMP_INST_OFFSET

  Initial value: `(0U)`
- #define SF_EL_FX_FAT16_SECTORS_PER_FAT

  Initial value: `(0x16U)`
- #define SF_EL_FX_FAT32_SECTORS_PER_FAT

  Initial value: `(0x24U)`
- #define SF_EL_FX_FAT16_SECTOR_COUNT

  Initial value: `(0x13U)`
- #define SF_EL_FX_FAT32_SECTOR_COUNT

  Initial value: `(0x20U)`
- #define SF_EL_FX_PARTITION0_OFFSET

  Initial value: `(0x1C6U)`
- #define SF_EL_FX_PARTITION0_TYPE_OFFSET

  Initial value: `(0x1C2U)`

7.14.4 SF_EL_FX_BlockDriver

```
SF_EL_FX_BlockDriver (  FX_MEDIA * media_ptr )
```

7.14.4.1 Brief description

Access Block Media device functions open, close, read, write and control.

7.14.4.2 Detailed description

The file system relies on the media to be formatted prior to creating directories and files The sector size and sector count will change depending on the media type and size.

The File Allocation Table (FAT) starts after the reserved sectors in the media. The FAT area is basically an array of 12-bit, 16-bit, or 32-bit entries that determine if that cluster is allocated or part of a chain of clusters comprising a subdirectory or a file. The size of each FAT entry is determined by the number of clusters that need to be represented. If the number of clusters (derived from the total sectors divided by the sectors per cluster) is less than 4,086, 12-bit FAT entries are used. If the total number of clusters is greater than 4,086 and less than or equal to 65,525, 16-bit FAT entries are used. Otherwise, if the total number of clusters is greater than 65,525, 32-bit FAT entries are used. The SF_EL_FX_BlockDriver function is called from the FileX file system driver and issues requests to a Block Media device through the Synergy Block Media Interface.

Table 575:Parameters

Name	Direction	Description
p_fx_media	inout	FileX media control block. All information about each open media device are maintained in by the FX_MEDIA data type. The I/O driver communicates the success or failure of the request through the fx_media_driver_status member of FX_MEDIA (p_fx_media->fx_media_driver_status). Possible values are documented in the FileX User Guide.

Table 576:Return values

Name	Description
noneUses	block media driver for accesses.

7.14.4.3 Function steps

- Initialize FileX I/O status to error. It will change to FX_SUCCESS unless an operation fails.
- Process the driver request specified in the media control block.
- FileX reads one or more sectors into memory by issuing a read request to the I/O driver.
- FileX writes one or more sectors to the physical media by issuing a write request to the I/O driver.

- FileX flushes all sectors currently in the driver's sector cache to the physical media by issuing a flush request to the I/O driver. Synergy drivers do not currently cache sectors.
- Command not currently supported in available Synergy modules. Return driver success.
- FileX informs the driver to abort all further physical I/O activity with the physical media by issuing an abort request to the I/O driver. The driver should not perform any I/O again until it is re-initialized.
- Close/un-initialize device.
- Although the actual driver initialization processing is application specific, it usually consists of data structure initialization and possibly some preliminary hardware initialization. This request is the first made by FileX and is done from within the fx_media_open service. If media write protection is detected, the driver fx_media_driver_write_protect member of FX_MEDIA should be set to FX_TRUE.
- Open the block media.
- Get write protection status.
- Close the block media.
- FileX uses the uninit command to close the media.
- Close/un-initialize device.
- Instead of using a standard read request, FileX makes a specific request to read the media's boot sector.
- Read the boot record and return to the caller.
- Open the block media.
- Use already allocated buffer
- Read the first non hidden sector.
- Check the partition offset to determine if the current sector is the boot record or partition table.
- Read the sector at the offset indicated in the partition table.
- Check the partition offset to confirm it is the boot record.
- check_partition_offset failed or the current buffer is not the boot sector
- Instead of using a standard write request, FileX makes a specific request to write the media's boot sector.
- Open driver.
- Write the boot record and return to the caller.
- Invalid driver request.
- Successful driver request.

7.14.5 check_partition_offset

```
ssp_err_t check_partition_offset (  uint8_t * p_sector ,   uint32_t
* p_partition_offset )
```

7.14.5.1 Brief description

Checks if the sector passed is a valid boot record or master boot record. If not valid boot record returns the offset to the partitions boot record.

7.14.5.2 Detailed description

(end addtogroup SF_EL_FX)

Table 577:Parameters

Name	Direction	Description
p_sector	in	Pointer to sector.
p_partition_offset	out	Sector number of found partition table. 0 if boot sector.

Table 578:Return values

Name	Description
SSP_SUCCESS	Found boot record or partition offset.
SSP_ERR_MEDIA_OPEN_FAILED	Not a valid boot record or partition table.

7.14.5.3 Function steps

- Check for FAT boot record.
- Check for master boot record.
- Read the offset to partition 0.
- Check for partition type is set and the offset is not zero and boot record signature for valid MBR then report the partition offset.

7.14.6 check_fat_boot_record

```
ssp_err_t check_fat_boot_record (  uint8_t * p_sector ,   uint32_t
* p_partition_offset )
```

7.14.6.1 Brief description

Checks if the sector passed is a valid boot record. If not returns error.

7.14.6.2 Detailed description

Table 579:Parameters

Name	Direction	Description
p_sector	in	Pointer to sector
p_partition_offset	out	Sector number of found partition table. 0 if boot sector.

Table 580:Return values

Name	Description
SSP_SUCCESS	Found boot record or partition offset
SSP_ERR_MEDIA_OPEN_FAILED	Not a valid boot record or partition table

7.14.6.3 Function steps

- Check for a valid jump instruction.
- Check for a non zero sectors per FAT.
- Check for a non zero sector count.
- Check for jump instruction and sectors per fat and sector count are not 0 which indicate FAT Boot record.
- Get sector size.
- Check the sector size value.
- Check Boot record signature to make sure the buffer is valid then return valid boot record.

7.14.7 Extensions

7.14.7.1 sf_el_fx_t

sf_el_fx_t

Detailed description

Block Media Control Block Type

Variables

- sf_block_media_instance_t * p_lower_lvl_block_media

  Lower level block media pointer.
- void * p_extend

7.15 GUIX Synergy Port

GUIX adaptation layer.

7.15.1 Functions

- SF_EL_GX_Open
- SF_EL_GX_Close
- SF_EL_GX_VersionGet
- SF_EL_GX_Setup
- SF_EL_GX_CanvasInit

7.15.2 SF_EL_GX_Open

```
ssp_err_t SF_EL_GX_Open ( sf_el_gx_ctrl_t  *const p_api_ctrl ,  sf_el_gx_cfg_t
const *const p_cfg )
```

7.15.2.1 Brief description

GUIX adaptation framework driver for Synergy, open function to configure the framework module. The function initialize RTOS resources used by the module, initialize the control block based on user configuration, and transition the module state to SF_EL_GX_OPENED. This function calls following functions:

7.15.2.2 Detailed description

- sf_el_gx_open_param_check() Check configuration parameters if parameter check is enabled.
- tx_mutex_create() Creates the mutex for lock the driver during the context update.
- tx_mutex_delete() Deletes the mutex if kernel service calls failed in the process.
- tx_semaphore_create() Creates the semaphore for rendering and displaying synchronization.

Table 581:Return values

Name	Description
SSP_SUCCESS	Opened the module successfully.
SSP_ERR_ASSERTION	NULL pointer error happened.
SSP_ERR_IN_USE	SF_EL_GX is in-use.
SSP_ERR_INTERNAL	Error happened in kernel service calls.
SSP_ERR_INVALID_ARGUMENT	An invalid argument was passed to the driver.

NOTE: This function does not initialize the display or rendering hardware but setup function will do.

NOTE: This function registers a user callback function but it is optional. Set NULL to p_cfg::p_callback if user callback is not required.

NOTE: The configuration for the frame buffer B (p_cfg::p_framebuffer_b) is optional. Set NULL to p_framebuffer_b in case of a single-buffered system.

7.15.2.3 Function steps

- Creates global mutex for SF_EL_GX
- Locks the SF_EL_GX instance until driver setup is done by SF_EL_GX_Setup.
- Creates a semaphore for frame buffer flip
- Initializes the SF_EL_GX control block
- Saves the control block to the global pointer inside the module temporarily. Stored data will be used in sf_el_gx_driver_setup() which will be invoked by GUIX. This pointer will be valid at last but be protected until SF_EL_GX_Setup is done.
- Changes the driver state

7.15.3 SF_EL_GX_Close

```
ssp_err_t SF_EL_GX_Close ( sf_el_gx_ctrl_t  *const p_api_ctrl )
```

7.15.3.1 Brief description

GUIX adaptation framework for Synergy, Close function. This function calls following functions:

7.15.3.2 Detailed description

- tx_mutex_get() Gets the mutex to lock the driver while device access.
- tx_mutex_put() Puts the mutex to unlock the driver while device access.
- tx_mutex_delete() Deletes the mutex if kernel service calls failed in the process.
- tx_semaphore_delete() Deletes the semaphore for rendering and displaying synchronization.
- sf_el_gx_d2_close() Finalizes 2D Drawing Engine hardware.
- sf_el_gx_display_close() Finalizes display hardware.

Table 582:Return values

Name	Description
SSP_SUCCESS	Closed the module successfully.
SSP_ERR_ASSERTION	NULL pointer error happens.
SSP_ERR_NOT_OPEN	SF_EL_GX is not opened.
SSP_ERR_INTERNAL	Error happened in kernel service calls.
SSP_ERR_TIMEOUT	Error occurred in display driver.
SSP_ERR_D2D_ERROR_DEINIT	Error occurred in D/AVE 2D driver.

NOTE: This function is re-entrant.

7.15.3.3 Function steps

- Locks the driver to update the context.
- Finalizes display hardware
- Changes the driver state
- Deletes a semaphore for frame buffer flip
- Unlocks the SF_EL_GX instance
- Deletes driver global mutex

- Clears the temporary storage for the pointer to a control block. This procedure has done in SF_EL_GX_Setup in the expected function call sequence, but clear it here as well in case SF_EL_GX_Setup being not called.

7.15.4 SF_EL_GX_VersionGet

```
ssp_err_t SF_EL_GX_VersionGet ( ssp_version_t  * p_version )
```

7.15.4.1 Brief description

GUIX adaptation framework for Synergy, Version get function.

7.15.4.2 Detailed description

Table 583:Parameters

Name	Direction	Description
p_version	inout	The version number.

Table 584:Return values

Name	Description
SSP_SUCCESS	Successfully returned the module version.
SSP_ERR_ASSERTION	NULL pointer is passed to function.

NOTE: This function is re-entrant.

7.15.5 SF_EL_GX_Setup

```
SF_EL_GX_Setup (  GX_DISPLAY * p_display )
```

7.15.5.1 Brief description

GUIX adaptation framework for Synergy, Setup GUIX low level device drivers for Display and D/AVE 2D interface. This function has to be passed to the GUIX Studio display driver setup function gx_studio_display_configure() to let GUIX call this function and configure the GUIX low level device driver(s). This function calls following functions:

7.15.5.2 Detailed description

- tx_mutex_put() Puts the driver global mutex when the low level driver setup is done
- tx_mutex_delete() Deletes the mutex if kernel service calls failed in the process
- _gx_synergy_display_driver_565rgb_setup() Setups default GUIX callback functions for RGB565 in case of display format format is RGB565 format
- _gx_synergy_display_driver_24xrgb_setup() Setups default GUIX callback functions for RGB565 in case of display format format is RGB888, unpacked format
- _gx_display_driver_32argb_setup() Setups default GUIX callback functions for RGB565 in case of display format format is ARGB8888, unpacked format
- sf_el_gx_driver_setup() Setups low level device drivers and overrides the GUIX default callback functions with hardware accelerated functions.

Table 585:Return values

Name	Description
GX_SUCCESS	Device driver setup is successfully done.
GX_FAILURE	Device driver setup failed.

NOTE: Make sure SF_EL_GX_Open has been called when this function is called back by GUIX. The behavior is not defined if this function were not invoked by GUIX.

7.15.5.3 Function steps

- Copies the GX_DISPLAY context for later use.
- Setups GUIX low level device drivers
- Changes the driver state
- Clears the temporary storage for the pointer to a control block.
- Unlocks the SF_EL_GX instance since driver setup is done

7.15.6 SF_EL_GX_CanvasInit

```
ssp_err_t SF_EL_GX_CanvasInit ( sf_el_gx_ctrl_t  *const p_api_ctrl ,
  GX_WINDOW_ROOT * p_window_root )
```

7.15.6.1 Brief description

GUIX adaptation framework for Synergy, Canvas initialization, setup the memory address of first canvas to be rendered.

7.15.6.2 Detailed description

Table 586:Return values

Name	Description
SSP_SUCCESS	Memory address is successfully configured to a canvas.
SSP_ERR_ASSERTION	Invalid control block (NULL pointer) or window root (NULL pointer) passed to driver.
SSP_ERR_INVALID_CALL	Function call was made when the driver is not in SF_EL_GX_CONFIGURED state.
SSP_ERR_INTERNAL	Mutex operation had an error.

7.15.6.3 Function steps

- Locks the driver to update the context.
- Lets GUIX know the first canvas
- Unlocks the driver.

7.15.7 Extensions

7.15.7.1 sf_el_gx_instance_ctrl_t

sf_el_gx_instance_ctrl_t

Detailed description

GUIX adaptation layer for SSP. Instance control block for the SSP GUIX adaptation framework

Variables

- GX_DISPLAY * p_display

  Pointer to the GUIX display context.
- display_instance_t * p_display_instance

  Pointer to a display instance.
- display_runtime_cfg_t * p_display_runtime_cfg

  Pointer to a runtime display configuration.
- void * p_canvas

  Pointer to a canvas(reserved)

- void * p_framebuffer_read

  Pointer to a frame buffer (for displaying)
- void * p_framebuffer_write

  Pointer to a frame buffer (for rendering)
- void(* p_callback)(*p_args)

  Pointer to callback function.
- void * p_context

  Pointer to a context.
- TX_SEMAPHORE semaphore

  Semaphore for the frame buffer flip sync.
- bool rendering_enable

  Sync flag between Rendering and displaying.
- bool display_list_flushed

  Flag to show the display list is flushed.
- sf_el_gx_state_t state

  State of this module.
- void * p_jpegbuffer

  Pointer to a JPEG work buffer.
- uint32_t jpegbuffer_size

  Size of a JPEG work buffer.
- uint16_t rotation_angle

  Screen rotation angle(0/90/270)
- void * p_sf_jpeg_decode_instance

  Pointer to a JPEG framework instance.
- _Bool dave2d_buffer_cache_enabled

  D/AVE 2D buffer cache enabled/disabled.

7.16 Telnet Communication Framework

RTOS-integrated Communications Framework NetX telnet server implementation.

7.16.1 Functions

- SF_EL_NX_COMMS_Open

- SF_EL_NX_COMMS_Close
- SF_EL_NX_COMMS_Read
- SF_EL_NX_COMMS_Write
- SF_EL_NX_COMMS_Lock
- SF_EL_NX_COMMS_Unlock
- SF_EL_NX_COMMS_VersionGet
- netx_resource_setup
- netx_resource_release
- tx_resource_setup
- tx_resource_release
- receive_data
- telnet_new_connection
- telnet_connection_end
- telnet_receive_data
- sf_el_nx_comms_error

7.16.2 Variables

- gp_ctrls
- gp_ctrls_logical

7.16.3 Defines

- #define SF_EL_NX_COMMS_CODE_VERSION_MAJOR
  Initial value: `(1U)`
- #define SF_EL_NX_COMMS_CODE_VERSION_MINOR
  Initial value: `(5U)`
- #define SF_EL_NX_COMMS_PACKET_POOL_MEMORY_SIZE_BYTES
  Initial value: `((1536U + 32U + (uint32_t) sizeof(NX_PACKET)) * 50U)`
- #define SF_EL_NX_COMMS_IP_MEMORY_SIZE_BYTES
  Initial value: `(2048U)`
- #define SF_EL_NX_COMMS_ARP_MEMORY_SIZE_BYTES
  Initial value: `(1024U)`
- #define SF_EL_NX_COMMS_TELNET_SERVER_MEMORY_SIZE_BYTES
  Initial value: `(2048U)`

- #define SF_EL_NX_COMMS_QUEUE_MEMORY_SIZE_BYTES

  Initial value: `(20U)`
- #define OPEN

  Initial value: `(0x4E58434DU)`

  0x4E58434DU is "NXCM" in ASCII, used to identify NetX comms handle

7.16.4 SF_EL_NX_COMMS_Open

```
ssp_err_t SF_EL_NX_COMMS_Open ( sf_comms_ctrl_t  *const p_api_ctrl ,
 sf_comms_cfg_t  const *const p_cfg )
```

7.16.4.1 Detailed description

Initializes the Telnet server.

Table 587:Return values

Name	Description
SSP_SUCCESS	Channel opened successfully
SSP_ERR_ASSERTION	One of the following invalid parameter passed. • Pointer p_api_ctrl is NULL. • Pointer p_cfg is NULL • Pointer p_cfg->p_extend is NULL
SSP_ERR_INTERNAL	An internal ThreadX Or NetX error has occurred. This is typically a failure to create/use thread mutex or failure create/enable an internal thread/feature for NetX service.

NOTE: - For Telnet server specific API details, refer Telnet server user manual.

NOTE: - For NetX specific API details, refer NetX user manual.

NOTE: - For ThreadX specific API details, refer ThreadX user manual.

NOTE: - This function is reentrant.

7.16.4.2 Function steps

- Initialize and start NetX and NetX application resources
- Initialize and start ThreadX resources
- Mark control block open.

7.16.5 SF_EL_NX_COMMS_Close

```
ssp_err_t SF_EL_NX_COMMS_Close ( sf_comms_ctrl_t  *const p_api_ctrl )
```

7.16.5.1 Detailed description

Disconnect Telnet server and clean up variables.

Table 588:Return values

Name	Description
SSP_SUCCESS	Channel successfully closed
SSP_ERR_ASSERTION	Pointer p_api_ctrl is NULL
SSP_ERR_INTERNAL	An internal ThreadX Or NetX error has occurred. This is typically a failure to create/use thread mutex or failure create/enable an internal thread/feature for NetX service.

NOTE: - For Telnet server specific API details, refer Telnet server user manual.

NOTE: - For NetX specific API details, refer NetX user manual.

NOTE: - For ThreadX specific API details, refer ThreadX user manual.

NOTE: - This function is reentrant.

7.16.5.2 Function steps

- Stop and release NetX resources
- Release ThreadX resources
- Mark control block closed.

7.16.6 SF_EL_NX_COMMS_Read

```
ssp_err_t SF_EL_NX_COMMS_Read ( sf_comms_ctrl_t  *const p_api_ctrl ,   uint8_t
*const p_dest ,   uint32_t const bytes ,   UINT const timeout )
```

7.16.6.1 Detailed description

Read data from the Telnet server.

Table 589:Return values

Name	Description
SSP_SUCCESS	Data reception ends successfully.
SSP_ERR_ASSERTION	One of the following invalid parameter passed. • Pointer p_api_ctrl is NULL • Pointer p_dest is NULL • Invalid read length i.e. bytes value is zero
SSP_ERR_TIMEOUT	One of the following operation timed out. • 'Event flags get' timed out • 'Receive mutex get' timed out • 'Queue receive' timed out
SSP_ERR_INTERNAL	An internal ThreadX Or NetX error has occurred. This is typically a failure to create/use thread mutex or failure create/enable an internal thread/feature for NetX service.

NOTE: - For Telnet server specific API details, refer Telnet server user manual.

NOTE: - For NetX specific API details, refer NetX user manual.

NOTE: - For ThreadX specific API details, refer ThreadX user manual.

NOTE: - This API is reentrant.

7.16.6.2 Function steps

- Wait for a client to be connected.
- Get mutex.
- Receive data.
- Release mutex.

7.16.7 SF_EL_NX_COMMS_Write

```
ssp_err_t SF_EL_NX_COMMS_Write ( sf_comms_ctrl_t  *const p_api_ctrl ,   uint8_t
const *const p_src ,   uint32_t const bytes ,   UINT const timeout )
```

7.16.7.1 Detailed description

Write data to the Telnet server.

Table 590:Return values

Name	Description
SSP_SUCCESS	Data transmission finished successfully.
SSP_ERR_ASSERTION	One of the following invalid parameter passed. • Pointer p_api_ctrl is NULL. • Pointer p_src is NULL. • Invalid write length i.e. bytes value is zero.

Table 590:Return values (Continued)

Name	Description
SSP_ERR_TIMEOUT	One of the following operation timed out. • 'Event flags get' timed out • 'Transmit mutex get' timed out
SSP_ERR_OUT_OF_MEMORY	Couldn't allocate pool memory for Telnet server
SSP_ERR_INTERNAL	An internal ThreadX Or NetX error has occurred. This is typically a failure to create/use thread mutex or failure create/enable an internal thread/feature for NetX service.

NOTE: - For Telnet server specific API details, refer Telnet server user manual.

NOTE: - For NetX specific API details, refer NetX user manual.

NOTE: - For ThreadX specific API details, refer ThreadX user manual.

NOTE: - This function is reentrant.

7.16.7.2 Function steps

- Get Transmit mutex.
- Allocate a packet for data.
- Build the message.
- Send the packet to the client.
- Release Transmit mutex.

7.16.8 SF_EL_NX_COMMS_Lock

```
ssp_err_t SF_EL_NX_COMMS_Lock ( sf_comms_ctrl_t  *const p_api_ctrl ,
 sf_comms_lock_t  lock_type ,   UINT timeout )
```

7.16.8.1 Detailed description

Table 591:Return values

Name	Description
SSP_SUCCESS	Locking NX_COMMS resource successful.
SSP_ERR_ASSERTION	Pointer p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Channel is not open.
SSP_ERR_TIMEOUT	Mutex not available in timeout.

NOTE: - For Telnet server specific API details, refer Telnet server user manual.

NOTE: - For NetX specific API details, refer NetX user manual.

NOTE: - For ThreadX specific API details, refer ThreadX user manual.

7.16.8.2 Function steps

- Get transmit mutex if transmit or all mutexes are requested.
- Get receive mutex if receive or all mutexes are requested.

7.16.9 SF_EL_NX_COMMS_Unlock

```
ssp_err_t SF_EL_NX_COMMS_Unlock ( sf_comms_ctrl_t  *const p_api_ctrl ,
 sf_comms_lock_t  lock_type )
```

7.16.9.1 Detailed description

Table 592:Return values

Name	Description
SSP_SUCCESS	Unlocking NX_COMMS resource successful.
SSP_ERR_ASSERTION	Pointer p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Channel is not open.

NOTE: - For Telnet server specific API details, refer Telnet server user manual.

NOTE: - For NetX specific API details, refer NetX user manual.

NOTE: - For ThreadX specific API details, refer ThreadX user manual.

7.16.9.2 Function steps

- Release transmit mutex if transmit or all mutexes are released.
- Release receive mutex if receive or all mutexes are released.

7.16.10 SF_EL_NX_COMMS_VersionGet

```
ssp_err_t SF_EL_NX_COMMS_VersionGet ( ssp_version_t  *const p_version )
```

7.16.10.1 Detailed description

Get driver version

Table 593:Return values

Name	Description
SSP_SUCCESS	Operation successful

Table 593:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	p_version pointer is NULL

NOTE: - This function is reentrant.

7.16.11 netx_resource_setup

```
ssp_err_t netx_resource_setup ( sf_el_nx_comms_instance_ctrl_t  * p_ctrl ,
 sf_el_nx_comms_on_comms_cfg_t  * p_cfg_extend )
```

7.16.11.1 Brief description

This is a helper routine called by open API to initialize NetX and NetX application resources.

7.16.11.2 Detailed description

Table 594:Parameters

Name	Direction	Description
p_ctrl	in	- Pointer to a control structure allocated by user.
p_cfg_extend	in	- Pointer to lower level communications control structure

7.16.11.3 Function steps

- Initialize the NetX system.
- Create a packet pool.
- Create an IP instance.
- Enable ARP and supply ARP cache memory for the IP Instance.
- Enable TCP traffic.
- Enable ICMP.
- Create the NetX TELNET Server.
- Start the TELNET Server.

7.16.12 netx_resource_release

```
ssp_err_t netx_resource_release ( sf_el_nx_comms_instance_ctrl_t  * p_ctrl )
```

7.16.12.1 Brief description

This is a helper routine called by close API to release NetX and NetX application resources.

7.16.12.2 Detailed description

Table 595:Parameters

Name	Direction	Description
p_ctrl	in	- Pointer to a control structure allocated by user.

7.16.12.3 Function steps

- Initiate server disconnection.
- Delete Telnet server instance.
- Delete IP instance.

7.16.13 tx_resource_setup

```
ssp_err_t tx_resource_setup ( sf_el_nx_comms_instance_ctrl_t  * p_ctrl )
```

7.16.13.1 Brief description

This is a helper routine called by open API to initialize ThreadX resources.

7.16.13.2 Detailed description

Table 596:Parameters

Name	Direction	Description
p_ctrl	in	- Pointer to a control structure allocated by user.

7.16.13.3 Function steps

- Create the transmit mutex.
- Create the receive mutex.

7.16.14 tx_resource_release

```
ssp_err_t tx_resource_release ( sf_el_nx_comms_instance_ctrl_t  * p_ctrl )
```

7.16.14.1 Brief description

This is a helper routine called by Close API to release ThreadX resources.

7.16.14.2 Detailed description

Table 597:Parameters

Name	Direction	Description
p_ctrl	in	- Pointer to a control structure allocated by user.

7.16.14.3 Function steps

- Delete transmit mutex.
- Delete receive mutex.
- Delete event flags group.
- Delete queue instance.

7.16.15 receive_data

```
ssp_err_t receive_data ( sf_el_nx_comms_instance_ctrl_t  * p_ctrl ,   uint8_t
*const p_dest ,   uint32_t bytes ,   UINT const timeout )
```

7.16.15.1 Brief description

Called by the NetX Telnet Server on new Telnet client connection request.

7.16.15.2 Detailed description

Table 598:Parameters

Name	Direction	Description
p_ctrl	in	- Pointer to a control structure allocated by user.
p_dest	in	- Destination
bytes	in	- Size of data
timeout	in	- Timeout period

7.16.16 telnet_new_connection

```
telnet_new_connection (  NX_TELNET_SERVER * server_ptr ,
 UINT logical_connection )
```

7.16.16.1 Brief description

Called by the NetX Telnet Server on new Telnet client connection request.

7.16.16.2 Detailed description

Table 599:Parameters

Name	Direction	Description
server_ptr		- Telnet server instance pointer
logical_connection		- Logical connection for this request

7.16.16.3 Function steps

- Find which control block belongs to this server.
- Set event flag to show connection is complete.

7.16.17 telnet_connection_end

```
telnet_connection_end (  NX_TELNET_SERVER * server_ptr ,
 UINT logical_connection )
```

7.16.17.1 Brief description

Called by the NetX Telnet Server on telnet client disconnect.

7.16.17.2 Detailed description

Table 600:Parameters

Name	Direction	Description
server_ptr		- Telnet server instance pointer
logical_connection		- Logical connection for this request

7.16.17.3 Function steps

- Clear event flag.

7.16.18 telnet_receive_data

```
telnet_receive_data (  NX_TELNET_SERVER * server_ptr ,
 UINT logical_connection ,   NX_PACKET * packet_ptr )
```

7.16.18.1 Brief description

Called by the NetX Telnet Server on data receive from Telnet client.

7.16.18.2 Detailed description

Table 601:Parameters

Name	Direction	Description
server_ptr		- Telnet server instance pointer
logical_connection		- Logical connection for this request
packet_ptr		- received packet pointer

7.16.18.3 Function steps

- Send packet to read function.

7.16.19 sf_el_nx_comms_error

```
ssp_err_t sf_el_nx_comms_error ( UINT status , ssp_err_t err )
```

7.16.19.1 Brief description

Helper function log an error when callers return type is void to avoid compiler warnings.

7.16.19.2 Detailed description

Table 602:Parameters

Name	Direction	Description
status		- Actual returned status value
err		- Error type to log and return on error

7.16.20 Extensions

7.16.20.1 sf_el_nx_comms_instance_ctrl_t

sf_el_nx_comms_instance_ctrl_t

Detailed description

NetX Telnet server communications instance control structure. DO NOT INITIALIZE. Initialization occurs when open is called

Variables

- uint32_t open
- NX_PACKET * p_current_packet
- uint32_t packet_index
- TX_MUTEX mutex[2]
- NX_PACKET_POOL pool
- uint8_t pool_memory[SF_EL_NX_COMMS_PACKET_POOL_MEMORY_SIZE_BYTES]
- NX_IP ip
- uint8_t ip_memory[SF_EL_NX_COMMS_IP_MEMORY_SIZE_BYTES]
- uint8_t arp_memory[SF_EL_NX_COMMS_ARP_MEMORY_SIZE_BYTES]
- NX_TELNET_SERVER telnet_server
- uint8_t telnet_server_memory[SF_EL_NX_COMMS_TELNET_SERVER_MEMORY_SIZE_BYTES]
- UINT logical_connection

- TX_EVENT_FLAGS_GROUP available

  Flag to tell if this connection is available or not.
- TX_QUEUE queue

  Queue of received bytes.
- uint8_t queue_memory[SF_EL_NX_COMMS_QUEUE_MEMORY_SIZE_BYTES]

7.16.20.2 sf_el_nx_comms_on_comms_cfg_t

sf_el_nx_comms_on_comms_cfg_t

Detailed description

NetX Telnet server device communications driver configuration

Variables

- uint32_t ip_address
- uint32_t subnet_mask
- VOID(* driver)(NX_IP_DRIVER *driver_req_ptr)

7.17 Express Logic USBX Synergy Port Framework

USBX Component SYNERGY Controller Driver

7.17.1 Functions

- usbhs_usb_int_resume_isr
- usbfs_int_isr
- usbfs_resume_isr
- ux_dcd_synergy_buffer_empty_interrupt
- ux_dcd_synergy_buffer_notready_interrupt
- ux_dcd_synergy_buffer_read
- ux_dcd_synergy_buffer_ready_interrupt
- ux_dcd_synergy_buffer_write
- ux_dcd_synergy_current_endpoint_change
- ux_dcd_synergy_data_buffer_size
- ux_dcd_synergy_endpoint_create
- ux_dcd_synergy_endpoint_destroy
- ux_dcd_synergy_endpoint_nak_set

- ux_dcd_synergy_endpoint_reset
- ux_dcd_synergy_endpoint_stall
- ux_dcd_synergy_endpoint_status
- ux_dcd_synergy_fifo_port_change
- ux_dcd_synergy_fifo_read
- ux_dcd_synergy_fifo_read_dma
- ux_dcd_synergy_read_dma_configure
- ux_dcd_synergy_read_dma_set_16bit
- ux_dcd_synergy_fifo_dma_start_read
- ux_dcd_synergy_fifo_read_software_copy
- ux_dcd_synergy_fifo_read_software_copy_16bit
- ux_dcd_synergy_fifo_read_last_bytes
- ux_dcd_synergy_fifoc_write
- ux_dcd_synergy_fifo_write_software_copy
- ux_dcd_synergy_fifo_write_software_copy_16bit
- ux_dcd_synergy_fifo_write_last_bytes
- ux_dcd_synergy_fifod_write
- ux_dcd_synergy_fifod_write_dma
- ux_dcd_synergy_write_dma_configure
- ux_dcd_synergy_write_dma_set_16bit
- ux_dcd_synergy_fifo_dma_start_write
- ux_dcd_synergy_frame_number_get
- ux_dcd_synergy_function
- ux_dcd_synergy_initialize
- ux_dcd_synergy_initialize_transfer_support
- ux_dcd_synergy_initialize_common
- ux_dcd_synergy_initialize_common_complete
- ux_dcd_dma_complete_tx
- ux_dcd_dma_complete_rx
- ux_dcd_synergy_initialize_complete
- ux_dcd_synergy_interrupt_handler
- ux_dcd_synergy_register_clear
- ux_dcd_synergy_register_read
- ux_dcd_synergy_register_set

- ux_dcd_synergy_register_write
- ux_dcd_synergy_transfer_callback
- ux_dcd_synergy_transfer_request
- ux_hcd_synergy_asynch_queue_process
- ux_hcd_synergy_asynch_queue_process_bemp
- ux_hcd_synergy_asynch_queue_process_brdy
- ux_hcd_synergy_asynch_queue_process_nrdy
- ux_hcd_synergy_asynch_queue_process_sign
- ux_hcd_synergy_asynch_schedule
- ux_hcd_synergy_asynchronous_endpoint_create
- ux_hcd_synergy_asynchronous_endpoint_destroy
- ux_hcd_synergy_buffer_empty_interrupt
- ux_hcd_synergy_buffer_notready_interrupt
- ux_hcd_synergy_buffer_read
- ux_hcd_synergy_buffer_ready_interrupt
- ux_hcd_synergy_buffer_write
- ux_hcd_synergy_bulk_endpoint_create
- ux_hcd_synergy_bulk_int_td_add
- ux_hcd_synergy_control_endpoint_create
- ux_hcd_synergy_control_td_add
- ux_hcd_synergy_controller_disable
- ux_hcd_synergy_current_endpoint_change
- ux_hcd_synergy_data_buffer_size
- ux_hcd_synergy_ed_obtain
- ux_hcd_synergy_ed_td_clean
- ux_hcd_synergy_endpoint_nak_set
- ux_hcd_synergy_endpoint_reset
- ux_hcd_synergy_entry
- ux_hcd_synergy_fifo_port_change
- ux_hcd_synergy_fifo_read
- ux_hcd_synergy_fifo_read_software_copy
- ux_hcd_synergy_fifo_read_software_copy_16bit
- ux_hcd_synergy_fifo_read_software_copy_8bit
- ux_hcd_synergy_fifo_read_dma_start

- ux_hcd_synergy_fifo_read_dma_set_16bit_or_8bit
- ux_hcd_synergy_fifoc_write
- ux_hcd_synergy_fifo_write_software_copy
- ux_hcd_synergy_fifo_write_software_copy_16bit
- ux_hcd_synergy_fifo_write_software_copy_8bit
- ux_hcd_synergy_fifo_write_software_copy_remaining_bytes
- ux_hcd_synergy_fifod_write
- ux_hcd_synergy_fifod_write_dma_start
- ux_hcd_synergy_fifod_write_dma_set_16bit_or_8bit
- ux_hcd_synergy_frame_number_get
- ux_hcd_synergy_frame_number_set
- ux_hcd_synergy_initialize
- ux_hcd_synergy_initialize_common
- ux_hcd_synergy_initialize_transfer_support
- ux_hcd_synergy_initialize_common_complete
- ux_hcd_dma_complete_tx
- ux_hcd_dma_complete_rx
- ux_hcd_synergy_interrupt_endpoint_create
- ux_hcd_synergy_interrupt_handler
- ux_hcd_synergy_iso_queue_process
- ux_hcd_synergy_iso_queue_process_bemp
- ux_hcd_synergy_iso_queue_bemp_transfer_end_check
- ux_hcd_synergy_iso_queue_process_brdy
- ux_hcd_synergy_iso_queue_brdy_out_transaction
- ux_hcd_synergy_iso_queue_brdy_in_transaction
- ux_hcd_synergy_iso_queue_process_nrdy
- ux_hcd_synergy_iso_schedule
- ux_hcd_synergy_isochronous_endpoint_create
- ux_hcd_synergy_isochronous_td_obtain
- ux_hcd_synergy_least_traffic_list_get
- ux_hcd_synergy_periodic_endpoint_destroy
- ux_hcd_synergy_periodic_schedule
- ux_hcd_synergy_periodic_tree_create
- ux_hcd_synergy_port_disable

- ux_hcd_synergy_port_enable
- ux_hcd_synergy_port_reset
- ux_hcd_synergy_port_resume
- ux_hcd_synergy_port_status_get
- ux_hcd_synergy_port_suspend
- ux_hcd_synergy_power_down_port
- ux_hcd_synergy_power_on_port
- ux_hcd_synergy_power_root_hubs
- ux_hcd_synergy_register_clear
- ux_hcd_synergy_register_read
- ux_hcd_synergy_register_set
- ux_hcd_synergy_register_write
- ux_hcd_synergy_regular_td_obtain
- ux_hcd_synergy_request_bulk_transfer
- ux_hcd_synergy_request_control_transfer
- ux_hcd_synergy_request_interrupt_transfer
- ux_hcd_synergy_request_isochronous_transfer
- ux_hcd_synergy_request_transfer
- ux_hcd_synergy_td_add
- ux_hcd_synergy_transfer_abort

7.17.2 Defines

- #define UX_DCD_SYNERGY_SLAVE_CONTROLLER

  Initial value: `(0x80U)`
- #define UX_DCD_SYNERGY_MAX_ED

  Initial value: `(7U)`
- #define UX_DCD_SYNERGY_ENABLE

  Initial value: `(1U)`
- #define UX_DCD_SYNERGY_DISABLE

  Initial value: `(0U)`
- #define UX_DCD_SYNERGY_MAX_BULK_PAYLOAD

  Initial value: `(512U)`
- #define UX_DCD_SYNERGY_MAX_CONTROL_PAYLOAD

  Initial value: `(512U)`

- #define UX_DCD_SYNERGY_MAX_BUF_SIZE
  Initial value: `(64U)`
- #define UX_DCD_SYNERGY_MAX_BUF_NUM
  Initial value: `(127U)`
- #define UX_DCD_SYNERGY_MIN_PIPE
  Initial value: `(4UL)`
- #define UX_DCD_SYNERGY_MAX_PIPE
  Initial value: `(7UL)`
- #define UX_SYNERGY_DCD_SYSCFG
  Initial value: `(0x00UL)`
- #define UX_SYNERGY_DCD_BUSWAIT
  Initial value: `(0x02UL)`
- #define UX_SYNERGY_DCD_SYSSTS
  Initial value: `(0x04UL)`
- #define UX_SYNERGY_DCD_PLLSTA
  Initial value: `(0x06UL)`
- #define UX_SYNERGY_DCD_DVSTCTR
  Initial value: `(0x08UL)`
- #define UX_SYNERGY_DCD_CFIFO
  Initial value: `(0x14UL)`
- #define UX_SYNERGY_DCD_CFIFOH
  Initial value: `(0x16UL)`
- #define UX_SYNERGY_DCD_CFIFOHH
  Initial value: `(0x17UL)`
- #define UX_SYNERGY_DCD_D0FIFO
  Initial value: `(0x18UL)`
- #define UX_SYNERGY_DCD_D0FIFOH
  Initial value: `(0x1AUL)`
- #define UX_SYNERGY_DCD_D0FIFOHH
  Initial value: `(0x1BUL)`
- #define UX_SYNERGY_DCD_D1FIFO
  Initial value: `(0x1CUL)`
- #define UX_SYNERGY_DCD_D1FIFOH
  Initial value: `(0x1EUL)`

- #define UX_SYNERGY_DCD_D1FIFOHH
  Initial value: (0x1FUL)
- #define UX_SYNERGY_DCD_CFIFOSEL
  Initial value: (0x20UL)
- #define UX_SYNERGY_DCD_CFIFOCTR
  Initial value: (0x22UL)
- #define UX_SYNERGY_DCD_D0FIFOSEL
  Initial value: (0x28UL)
- #define UX_SYNERGY_DCD_D0FIFOCTR
  Initial value: (0x2AUL)
- #define UX_SYNERGY_DCD_D1FIFOSEL
  Initial value: (0x2CUL)
- #define UX_SYNERGY_DCD_D1FIFOCTR
  Initial value: (0x2EUL)
- #define UX_SYNERGY_DCD_INTENB0
  Initial value: (0x30UL)
- #define UX_SYNERGY_DCD_INTENB1
  Initial value: (0x32UL)
- #define UX_SYNERGY_DCD_BRDYENB
  Initial value: (0x36UL)
- #define UX_SYNERGY_DCD_NRDYENB
  Initial value: (0x38UL)
- #define UX_SYNERGY_DCD_BEMPENB
  Initial value: (0x3AUL)
- #define UX_SYNERGY_DCD_SOFCFG
  Initial value: (0x3CUL)
- #define UX_SYNERGY_DCD_PHYSET
  Initial value: (0x3EUL)
- #define UX_SYNERGY_DCD_INTSTS0
  Initial value: (0x40UL)
- #define UX_SYNERGY_DCD_INTSTS1
  Initial value: (0x42UL)
- #define UX_SYNERGY_DCD_BRDYSTS
  Initial value: (0x46UL)

- #define UX_SYNERGY_DCD_NRDYSTS
  Initial value: (0x48UL)
- #define UX_SYNERGY_DCD_BEMPSTS
  Initial value: (0x4AUL)
- #define UX_SYNERGY_DCD_FRMNUM
  Initial value: (0x4CUL)
- #define UX_SYNERGY_DCD_DVCHGR
  Initial value: (0x4EUL)
- #define UX_SYNERGY_DCD_USBADDR
  Initial value: (0x50UL)
- #define UX_SYNERGY_DCD_USBREQ
  Initial value: (0x54UL)
- #define UX_SYNERGY_DCD_USBVAL
  Initial value: (0x56UL)
- #define UX_SYNERGY_DCD_USBINDX
  Initial value: (0x58UL)
- #define UX_SYNERGY_DCD_USBLENG
  Initial value: (0x5AUL)
- #define UX_SYNERGY_DCD_DCPCFG
  Initial value: (0x5CUL)
- #define UX_SYNERGY_DCD_DCPMAXP
  Initial value: (0x5EUL)
- #define UX_SYNERGY_DCD_DCPCTR
  Initial value: (0x60UL)
- #define UX_SYNERGY_DCD_PIPESEL
  Initial value: (0x64UL)
- #define UX_SYNERGY_DCD_PIPECFG
  Initial value: (0x68UL)
- #define UX_SYNERGY_DCD_PIPEMAXP
  Initial value: (0x6CUL)
- #define UX_SYNERGY_DCD_PIPEPERI
  Initial value: (0x6EUL)
- #define UX_SYNERGY_DCD_PIPE1CTR
  Initial value: (0x70UL)

- #define UX_SYNERGY_DCD_PIPE2CTR
  Initial value: `(0x72UL)`
- #define UX_SYNERGY_DCD_PIPE3CTR
  Initial value: `(0x74UL)`
- #define UX_SYNERGY_DCD_PIPE4CTR
  Initial value: `(0x76UL)`
- #define UX_SYNERGY_DCD_PIPE5CTR
  Initial value: `(0x78UL)`
- #define UX_SYNERGY_DCD_PIPE6CTR
  Initial value: `(0x7AUL)`
- #define UX_SYNERGY_DCD_PIPE7CTR
  Initial value: `(0x7CUL)`
- #define UX_SYNERGY_DCD_PIPE8CTR
  Initial value: `(0x7EUL)`
- #define UX_SYNERGY_DCD_PIPE9CTR
  Initial value: `(0x80UL)`
- #define UX_SYNERGY_DCD_PIPE1TRE
  Initial value: `(0x90UL)`
- #define UX_SYNERGY_DCD_PIPE1TRN
  Initial value: `(0x92UL)`
- #define UX_SYNERGY_DCD_PIPE2TRE
  Initial value: `(0x94UL)`
- #define UX_SYNERGY_DCD_PIPE2TRN
  Initial value: `(0x96UL)`
- #define UX_SYNERGY_DCD_PIPE3TRE
  Initial value: `(0x98UL)`
- #define UX_SYNERGY_DCD_PIPE3TRN
  Initial value: `(0x9AUL)`
- #define UX_SYNERGY_DCD_PIPE4TRE
  Initial value: `(0x9CUL)`
- #define UX_SYNERGY_DCD_PIPE4TRN
  Initial value: `(0x9EUL)`
- #define UX_SYNERGY_DCD_PIPE5TRE
  Initial value: `(0xA0UL)`

- #define UX_SYNERGY_DCD_PIPE5TRN
  Initial value: `(0xA2UL)`
- #define UX_SYNERGY_DCD_USBMC
  Initial value: `(0xCCUL)`
- #define UX_SYNERGY_DCD_DEVADD0
  Initial value: `(0xD0UL)`
- #define UX_SYNERGY_DCD_DEVADD1
  Initial value: `(0xD2UL)`
- #define UX_SYNERGY_DCD_DEVADD2
  Initial value: `(0xD4UL)`
- #define UX_SYNERGY_DCD_DEVADD3
  Initial value: `(0xD6UL)`
- #define UX_SYNERGY_DCD_DEVADD4
  Initial value: `(0xD8UL)`
- #define UX_SYNERGY_DCD_DEVADD5
  Initial value: `(0xDAUL)`
- #define UX_SYNERGY_DCD_PHYSLEW
  Initial value: `(0xF0UL)`
- #define UX_SYNERGY_DCD_LPSTS
  Initial value: `(0x102UL)`
- #define UX_SYNERGY_DCD_UCKSEL
  Initial value: `(0xC4U)`
- #define UX_SYNERGY_DCD_BUSWAIT_DEFAULT_VAL
  Initial value: `(0xFU)`
- #define UX_SYNERGY_DCD_BUSWAIT_MASK
  Initial value: `(0xFU)`
- #define UX_SYNERGY_DCD_BUSWAIT_CALC_FREQ_PCLK_CYC
  Initial value: `(17142857U)`
- #define UX_SYNERGY_DCD_BUSWAIT_CALC_FREQ_PCLK_CYCx10
  Initial value: `(1714288U)`
- #define UX_SYNERGY_DCD_SYSCFG_SCKE
  Initial value: `(1U<<10)`
- #define UX_SYNERGY_DCD_SYSCFG_CNEN
  Initial value: `(1U<<8)`

- #define UX_SYNERGY_DCD_SYSCFG_HSE
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_DCD_SYSCFG_DCFM
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_DCD_SYSCFG_DRPD
  Initial value: `(1U<<5)`
- #define UX_SYNERGY_DCD_SYSCFG_DPRPU
  Initial value: `(1U<<4)`
- #define UX_SYNERGY_DCD_SYSCFG_DMRPU
  Initial value: `(1U<<3)`
- #define UX_SYNERGY_DCD_SYSCFG_USBE
  Initial value: `(1)`
- #define UX_SYNERGY_DCD_SYSSTS_LNST
  Initial value: `(3)`
- #define UX_SYNERGY_DCD_SYSSTS_SOFEN
  Initial value: `(0x20U)`
- #define UX_SYNERGY_DCD_DVSTCTR_UACKEY0
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_DCD_DVSTCTR_UACKEY1
  Initial value: `(1U<<12)`
- #define UX_SYNERGY_DCD_DVSTCTR_WKUP
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_DCD_DVSTCTR_RWUPE
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_DCD_DVSTCTR_USBRST
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_DCD_DVSTCTR_RESUME
  Initial value: `(1U<<5)`
- #define UX_SYNERGY_DCD_DVSTCTR_UACT
  Initial value: `(1U<<4)`
- #define UX_SYNERGY_DCD_DVSTCTR_RHST
  Initial value: `(0x7U)`
- #define UX_SYNERGY_DCD_DVSTCTR_SPEED_LOW
  Initial value: `(1)`

- #define UX_SYNERGY_DCD_DVSTCTR_SPEED_FULL
  Initial value: `(2)`
- #define UX_SYNERGY_DCD_DVSTCTR_SPEED_HIGH
  Initial value: `(3)`
- #define UX_SYNERGY_DCD_DVSTCTR_RESET_IN_PROGRESS
  Initial value: `(4)`
- #define UX_SYNERGY_DCD_TESTMODE_HOSTPCC
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_DCD_DXFBCFG_DFACC
  Initial value: `(0U<<12)`
- #define UX_SYNERGY_DCD_CFIFOSEL_RCNT
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_DCD_CFIFOSEL_REW
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_DCD_CFIFOSEL_MBW_8
  Initial value: `(0U<<10)`
- #define UX_SYNERGY_DCD_CFIFOSEL_MBW_16
  Initial value: `(1U<<10)`
- #define UX_SYNERGY_DCD_CFIFOSEL_MBW_32
  Initial value: `(2U<<10)`
- #define UX_SYNERGY_DCD_CFIFOSEL_MBW_MASK
  Initial value: `(3U<<10)`
- #define UX_SYNERGY_DCD_CFIFOSEL_BIGEND
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_DCD_CFIFOSEL_ISEL
  Initial value: `(1U<<5)`
- #define UX_SYNERGY_DCD_CFIFOSEL_CURPIPE_MASK
  Initial value: `(0xFU)`
- #define UX_SYNERGY_DCD_DFIFOSEL_RCNT
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_DCD_DFIFOSEL_REW
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_DCD_DFIFOSEL_DCLRM
  Initial value: `(1U<<13)`

- #define UX_SYNERGY_DCD_DFIFOSEL_DREQE
  Initial value: `(1U<<12)`
- #define UX_SYNERGY_DCD_DFIFOSEL_MBW_8
  Initial value: `(0U<<10)`
- #define UX_SYNERGY_DCD_DFIFOSEL_MBW_16
  Initial value: `(1U<<10)`
- #define UX_SYNERGY_DCD_DFIFOSEL_MBW_32
  Initial value: `(2U<<10)`
- #define UX_SYNERGY_DCD_DFIFOSEL_MBW_MASK
  Initial value: `(3U<<10)`
- #define UX_SYNERGY_DCD_DFIFOSEL_BIGEND
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_DCD_DFIFOSEL_CURPIPE_MASK
  Initial value: `(0xFU)`
- #define UX_SYNERGY_DCD_FIFOCTR_BVAL
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_DCD_FIFOCTR_BCLR
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_DCD_FIFOCTR_FRDY
  Initial value: `(1U<<13)`
- #define UX_SYNERGY_DCD_FIFOCTR_DTLN
  Initial value: `(0xFFFU)`
- #define UX_SYNERGY_DCD_INTENB0_VBSE
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_DCD_INTENB0_RSME
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_DCD_INTENB0_SOFE
  Initial value: `(1U<<13)`
- #define UX_SYNERGY_DCD_INTENB0_DVSE
  Initial value: `(1U<<12)`
- #define UX_SYNERGY_DCD_INTENB0_CTRE
  Initial value: `(1U<<11)`
- #define UX_SYNERGY_DCD_INTENB0_BEMPE
  Initial value: `(1U<<10)`

- #define UX_SYNERGY_DCD_INTENB0_NRDYE
  Initial value: `(1U<<9)`
- #define UX_SYNERGY_DCD_INTENB0_BRDYE
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_DCD_INTENB1_BCHGE
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_DCD_INTENB1_DTCHE
  Initial value: `(1U<<12)`
- #define UX_SYNERGY_DCD_INTENB1_ATTCHE
  Initial value: `(1U<<11)`
- #define UX_SYNERGY_DCD_INTENB1_EOFERRE
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_DCD_INTENB1_SIGNE
  Initial value: `(1U<<5)`
- #define UX_SYNERGY_DCD_INTENB1_SACKE
  Initial value: `(1U<<4)`
- #define UX_SYNERGY_DCD_PIPE0
  Initial value: `(1U<<0)`
- #define UX_SYNERGY_DCD_PIPE_ALL
  Initial value: `(0x3FFU)`
- #define UX_SYNERGY_DCD_SOFCFG_TRNENSEL
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_DCD_SOFCFG_BRDYM
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_DCD_SOFCFG_INIT
  Initial value: `(1U<<5)`
- #define UX_SYNERGY_DCD_PHYSET_HSEB
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_DCD_PHYSET_REPSTART
  Initial value: `(1U<<11)`
- #define UX_SYNERGY_DCD_PHYSET_REPSEL
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_DCD_PHYSET_CLKSEL_1
  Initial value: `(1U<<5)`

- #define UX_SYNERGY_DCD_PHYSET_CLKSEL_0
  Initial value: `(1U<<4)`
- #define UX_SYNERGY_DCD_PHYSET_CDPEN
  Initial value: `(1U<<3)`
- #define UX_SYNERGY_DCD_PHYSET_PLLRESET
  Initial value: `(1U<<1)`
- #define UX_SYNERGY_DCD_PHYSET_DIRPD
  Initial value: `(1U<<0)`
- #define UX_SYNERGY_DCD_INTSTS0_VBINT
  Initial value: `(USHORT)(1U<<15)`
- #define UX_SYNERGY_DCD_INTSTS0_RESM
  Initial value: `(USHORT)(1U<<14)`
- #define UX_SYNERGY_DCD_INTSTS0_SOFR
  Initial value: `(1U<<13)`
- #define UX_SYNERGY_DCD_INTSTS0_DVST
  Initial value: `(USHORT)(1U<<12)`
- #define UX_SYNERGY_DCD_INTSTS0_CTRT
  Initial value: `(USHORT)(1U<<11)`
- #define UX_SYNERGY_DCD_INTSTS0_BEMP
  Initial value: `(1U<<10)`
- #define UX_SYNERGY_DCD_INTSTS0_NRDY
  Initial value: `(1U<<9)`
- #define UX_SYNERGY_DCD_INTSTS0_BRDY
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_DCD_INTSTS0_VBSTS
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_DCD_INTSTS0_DVSQ_MASK
  Initial value: `(7U<<4)`
- #define UX_SYNERGY_DCD_INTSTS0_VALID
  Initial value: `(USHORT)(1U<<3)`
- #define UX_SYNERGY_DCD_INTSTS0_CTSQ_MASK
  Initial value: `(7U)`
- #define UX_SYNERGY_DCD_INTSTS0_DVSQ_POWERED
  Initial value: `(0x0000U)`

- #define UX_SYNERGY_DCD_INTSTS0_DVSQ_DEFAULT
  Initial value: `(0x0010U)`
- #define UX_SYNERGY_DCD_INTSTS0_DVSQ_ADDRESS
  Initial value: `(0x0020U)`
- #define UX_SYNERGY_DCD_INTSTS0_DVSQ_CONFIGURED
  Initial value: `(0x0030U)`
- #define UX_SYNERGY_DCD_INTSTS0_DVSQ_SUSPENDED_POWERED
  Initial value: `(0x0040U)`
- #define UX_SYNERGY_DCD_INTSTS0_DVSQ_SUSPENDED_DEFAULT
  Initial value: `(0x0050U)`
- #define UX_SYNERGY_DCD_INTSTS0_DVSQ_SUSPENDED_ADDRESS
  Initial value: `(0x0060U)`
- #define UX_SYNERGY_DCD_INTSTS0_DVSQ_SUSPENDED_CONFIGURED
  Initial value: `(0x0070U)`
- #define UX_SYNERGY_DCD_INTSTS0_CTSQ_SETUP
  Initial value: `(0x0000U)`
- #define UX_SYNERGY_DCD_INTSTS0_CTSQ_CRDS
  Initial value: `(0x0001U)`
- #define UX_SYNERGY_DCD_INTSTS0_CTSQ_CRSS
  Initial value: `(0x0002U)`
- #define UX_SYNERGY_DCD_INTSTS0_CTSQ_CWDS
  Initial value: `(0x0003U)`
- #define UX_SYNERGY_DCD_INTSTS0_CTSQ_CWSS
  Initial value: `(0x0004U)`
- #define UX_SYNERGY_DCD_INTSTS0_CTSQ_CWNDSS
  Initial value: `(0x0005U)`
- #define UX_SYNERGY_DCD_INTSTS0_CTSQ_CTSE
  Initial value: `(0x0006U)`
- #define UX_SYNERGY_DCD_INTSTS1_BCHG
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_DCD_INTSTS1_DTCH
  Initial value: `(1U<<12)`
- #define UX_SYNERGY_DCD_INTSTS1_ATTCH
  Initial value: `(1U<<11)`

- #define UX_SYNERGY_DCD_INTSTS1_EOFERR
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_DCD_INTSTS1_SIGN
  Initial value: `(1U<<5)`
- #define UX_SYNERGY_DCD_INTSTS1_SACK
  Initial value: `(1U<<4)`
- #define UX_SYNERGY_DCD_FRMNUM_OVRN
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_DCD_FRMNUM_CRCE
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_DCD_FRMNUM_FRNM_MASK
  Initial value: `(0x7FF)`
- #define UX_SYNERGY_DCD_DCPCFG_DIR
  Initial value: `(1U<<4)`
- #define UX_SYNERGY_DCD_DCPCFG_SHTNAK
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_DCD_DCPCFG_CNTMD
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_DCD_DCPMAXP_DEVADDR_SHIFT
  Initial value: `(12)`
- #define UX_SYNERGY_DCD_DCPMAXP_DEVADDR_MASK
  Initial value: `(0xF000U)`
- #define UX_SYNERGY_DCD_DCPCTR_BSTS
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_DCD_DCPCTR_INBUFM
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_DCD_DCPCTR_CSCLR
  Initial value: `(1U<<13)`
- #define UX_SYNERGY_DCD_DCPCTR_CSSTS
  Initial value: `(1U<<12)`
- #define UX_SYNERGY_DCD_DCPCTR_SUREQCLR
  Initial value: `(1U<<11)`
- #define UX_SYNERGY_DCD_DCPCTR_SQCLR
  Initial value: `(1U<<8)`

- #define UX_SYNERGY_DCD_DCPCTR_SQSET
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_DCD_DCPCTR_SQMON
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_DCD_DCPCTR_PBUSY
  Initial value: `(1U<<5)`
- #define UX_SYNERGY_DCD_DCPCTR_PINGE
  Initial value: `(1U<<4)`
- #define UX_SYNERGY_DCD_DCPCTR_CCPL
  Initial value: `(1U<<2)`
- #define UX_SYNERGY_DCD_DCPCTR_PID_MASK
  Initial value: `(3U)`
- #define UX_SYNERGY_DCD_DCPCTR_PIDNAK
  Initial value: `(0)`
- #define UX_SYNERGY_DCD_DCPCTR_PIDBUF
  Initial value: `(1U)`
- #define UX_SYNERGY_DCD_DCPCTR_PIDSTALL
  Initial value: `(2U)`
- #define UX_SYNERGY_DCD_DCPCTR_PIDSTALL2
  Initial value: `(3U)`
- #define UX_SYNERGY_DCD_PIPECFG_TYPE_MASK
  Initial value: `(0xC000U)`
- #define UX_SYNERGY_DCD_PIPECFG_TYPE_BIT_USED
  Initial value: `(0)`
- #define UX_SYNERGY_DCD_PIPECFG_TYPE_BULK
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_DCD_PIPECFG_TYPE_INTERRUPT
  Initial value: `(2U<<14)`
- #define UX_SYNERGY_DCD_PIPECFG_TYPE_ISOCHRONOUS
  Initial value: `(3U<<14)`
- #define UX_SYNERGY_DCD_PIPECFG_BFRE
  Initial value: `(1U<<10)`
- #define UX_SYNERGY_DCD_PIPECFG_DBLB
  Initial value: `(1U<<9)`

- #define UX_SYNERGY_DCD_PIPECFG_CNTMD
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_DCD_PIPECFG_SHTNAK
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_DCD_PIPECFG_DIR
  Initial value: `(1U<<4)`
- #define UX_SYNERGY_DCD_PIPECFG_EPNUM_MASK
  Initial value: `(0xFU)`
- #define UX_SYNERGY_DCD_PIPEBUF_SIZEMASK
  Initial value: `(0x1FU<<10)`
- #define UX_SYNERGY_DCD_PIPEBUF_BUFNMBMASK
  Initial value: `(0xFFU<<10)`
- #define UX_SYNERGY_DCD_PIPEBUF_SHIFT
  Initial value: `(10)`
- #define UX_SYNERGY_DCD_PIPEMAXP_DEVSELMASK
  Initial value: `(0xFU<<12)`
- #define UX_SYNERGY_DCD_PIPEMAXP_DEVSEL_SHIFT
  Initial value: `(12)`
- #define UX_SYNERGY_DCD_PIPEMAXP_MXPSMASK
  Initial value: `(0x7FF)`
- #define UX_SYNERGY_DCD_PIPE1TRE_TRCLR
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_DCD_PIPE1TRE_TRENB
  Initial value: `(1U<<9)`
- #define UX_SYNERGY_DCD_FIFO_D0
  Initial value: `(0UL)`
- #define UX_SYNERGY_DCD_FIFO_D1
  Initial value: `(1UL)`
- #define UX_SYNERGY_DCD_FIFO_C
  Initial value: `(2UL)`
- #define UX_SYNERGY_DCD_DEVADDX_SPEED_LOW
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_DCD_DEVADDX_SPEED_FULL
  Initial value: `(2U<<6)`

- #define UX_SYNERGY_DCD_DEVADDX_SPEED_HIGH
  Initial value: `(3U<<6)`
- #define UX_SYNERGY_DCD_DEVADDX_UPPHUB_SHIFT
  Initial value: `(11)`
- #define UX_SYNERGY_DCD_DEVADDX_HUBPORT_SHIFT
  Initial value: `(8)`
- #define UX_SYNERGY_DCD_USBMC_VDCEN
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_DCD_DCPCTR_DATA1
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_DCD_DCPCTR_DATA0
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_DCD_PIPESEL_NO_PIPE
  Initial value: `(0x000FU)`
- #define UX_SYNERGY_DCD_PIPESEL_NO_PIPE_USHORT
  Initial value: `(0x0FU)`
- #define UX_SYNERGY_DCD_UCKSEL_UCKSELC
  Initial value: `(1)`
- #define UX_SYNERGY_DCD_PIPE0_SIZE
  Initial value: `(256U)`
- #define UX_SYNERGY_DCD_PIPE_NB_BUFFERS
  Initial value: `(64)`
- #define UX_SYNERGY_DCD_PIPECTR_INBUFM
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_DCD_PIPECTR_BSTS
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_DCD_PIPECTR_CSCLR
  Initial value: `(1U<<13)`
- #define UX_SYNERGY_DCD_PIPECTR_CSSTS
  Initial value: `(1U<<12)`
- #define UX_SYNERGY_DCD_PIPECTR_ATREPM
  Initial value: `(1U<<11)`
- #define UX_SYNERGY_DCD_PIPECTR_SQCLR
  Initial value: `(1U<<8)`

- #define UX_SYNERGY_DCD_PIPECTR_SQSET
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_DCD_PIPECTR_SQMON
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_DCD_PIPECTR_PBUSY
  Initial value: `(1U<<5)`
- #define UX_SYNERGY_DCD_PIPECTR_PID_MASK
  Initial value: `(3U)`
- #define UX_SYNERGY_DCD_PIPECTR_PIDNAK
  Initial value: `(0U)`
- #define UX_SYNERGY_DCD_PIPECTR_PIDBUF
  Initial value: `(1U)`
- #define UX_SYNERGY_DCD_PIPECTR_PIDSTALL
  Initial value: `(2U)`
- #define UX_SYNERGY_DCD_PIPECTR_PIDSTALL2
  Initial value: `(3U)`
- #define UX_SYNERGY_DCD_PIPECTR_DATA1
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_DCD_PIPECTR_DATA0
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_DCD_COMMAND_STATUS_RESET
  Initial value: `(0)`
- #define UX_SYNERGY_DCD_INIT_RESET_DELAY
  Initial value: `(10)`
- #define UX_SYNERGY_DCD_MAX_BUF_SIZE
  Initial value: `(64)`
- #define UX_SYNERGY_DCD_MAX_BUF_NUM
  Initial value: `(135UL)`
- #define UX_SYNERGY_DCD_PIPE1_BUF_START_NUM
  Initial value: `(8UL)`
- #define UX_SYNERGY_DCD_PIPE6_BUF_START_NUM
  Initial value: `(4UL)`
- #define UX_SYNERGY_DCD_FIFO_WRITING
  Initial value: `(2U)`

- #define UX_SYNERGY_DCD_FIFO_WRITE_END
  Initial value: `(3U)`
- #define UX_SYNERGY_DCD_FIFO_WRITE_SHORT
  Initial value: `(4U)`
- #define UX_SYNERGY_DCD_FIFO_WRITE_DMA
  Initial value: `(5U)`
- #define UX_SYNERGY_DCD_FIFO_WRITE_ERROR
  Initial value: `(6U)`
- #define UX_SYNERGY_DCD_FIFO_READING
  Initial value: `(2U)`
- #define UX_SYNERGY_DCD_FIFO_READ_END
  Initial value: `(3U)`
- #define UX_SYNERGY_DCD_FIFO_READ_SHORT
  Initial value: `(4U)`
- #define UX_SYNERGY_DCD_FIFO_READ_DMA
  Initial value: `(5U)`
- #define UX_SYNERGY_DCD_FIFO_READ_ERROR
  Initial value: `(6U)`
- #define UX_SYNERGY_DCD_FIFO_READ_OVER
  Initial value: `(7U)`
- #define UX_SYNERGY_DCD_ED_BRDY
  Initial value: `(0x00000001U)`
- #define UX_SYNERGY_DCD_ED_NRDY
  Initial value: `(0x00000002U)`
- #define UX_SYNERGY_DCD_ED_BEMP
  Initial value: `(0x00000004U)`
- #define UX_SYNERGY_DCD_ED_EOFERR
  Initial value: `(0x00000010U)`
- #define UX_SYNERGY_DCD_ED_SIGN
  Initial value: `(0x00000020U)`
- #define UX_SYNERGY_DCD_ED_SACK
  Initial value: `(0x00000040U)`
- #define UX_SYNERGY_DCD_PHYSLEW_SLEW_SLEWR00
  Initial value: `(1U<<0)`

- #define UX_SYNERGY_DCD_PHYSLEW_SLEW_SLEWR01
  Initial value: `(1U<<1)`
- #define UX_SYNERGY_DCD_PHYSLEW_SLEW_SLEWF00
  Initial value: `(1U<<2)`
- #define UX_SYNERGY_DCD_PHYSLEW_SLEW_SLEWF01
  Initial value: `(1U<<3)`
- #define UX_SYNERGY_DCD_LPSTS_SUSPENDM
  Initial value: `(1U<<14)`
- #define UX_DCD_SYNERGY_ED_STATUS_UNUSED
  Initial value: `(0U)`
- #define UX_DCD_SYNERGY_ED_STATUS_USED
  Initial value: `(1U)`
- #define UX_DCD_SYNERGY_ED_STATUS_TRANSFER
  Initial value: `(2U)`
- #define UX_DCD_SYNERGY_ED_STATUS_STALLED
  Initial value: `(4U)`
- #define UX_DCD_SYNERGY_ED_STATE_IDLE
  Initial value: `(0U)`
- #define UX_DCD_SYNERGY_ED_STATE_DATA_TX
  Initial value: `(1U)`
- #define UX_DCD_SYNERGY_ED_STATE_DATA_RX
  Initial value: `(2U)`
- #define UX_DCD_SYNERGY_ED_STATE_STATUS_TX
  Initial value: `(3U)`
- #define UX_DCD_SYNERGY_ED_STATE_STATUS_RX
  Initial value: `(4U)`
- #define UX_SYNERGY_CONTROLLER
  Initial value: `(0)`
- #define UX_SYNERGY_CONTROLLER_S7G2
  Initial value: `(1)`
- #define UX_SYNERGY_CONTROLLER_S3A7
  Initial value: `(2)`
- #define UX_SYNERGY_MAX_BULK_PAYLOAD
  Initial value: `(512)`

- #define UX_SYNERGY_MAX_ISO_PAYLOAD

  Initial value: `(1024)`
- #define UX_SYNERGY_MAX_CONTROL_PAYLOAD

  Initial value: `(512)`
- #define UX_SYNERGY_FRAME_DELAY

  Initial value: `(4UL)`
- #define UX_SYNERGY_PERIODIC_ENTRY_NB

  Initial value: `(32)`
- #define UX_SYNERGY_PERIODIC_ENTRY_MASK

  Initial value: `(0x1FU)`
- #define UX_SYNERGY_ENABLE

  Initial value: `(1UL)`
- #define UX_SYNERGY_DISABLE

  Initial value: `(0UL)`
- #define UX_SYNERGY_HC_SYSCFG

  Initial value: `(0x00UL)`
- #define UX_SYNERGY_HC_BUSWAIT

  Initial value: `(0x02UL)`
- #define UX_SYNERGY_HC_SYSSTS0

  Initial value: `(0x04UL)`
- #define UX_SYNERGY_HC_PLLSTA

  Initial value: `(0x06UL)`
- #define UX_SYNERGY_HC_DVSTCTR0

  Initial value: `(0x08UL)`
- #define UX_SYNERGY_HC_CFIFO

  Initial value: `(0x14UL)`
- #define UX_SYNERGY_HC_CFIFOH

  Initial value: `(0x16UL)`
- #define UX_SYNERGY_HC_CFIFOHH

  Initial value: `(0x17UL)`
- #define UX_SYNERGY_HC_D0FIFO

  Initial value: `(0x18UL)`
- #define UX_SYNERGY_HC_D1FIFO

  Initial value: `(0x1CUL)`

- #define UX_SYNERGY_HC_CFIFOSEL
  Initial value: (0x20UL)
- #define UX_SYNERGY_HC_CFIFOCTR
  Initial value: (0x22UL)
- #define UX_SYNERGY_HC_D0FIFOSEL
  Initial value: (0x28UL)
- #define UX_SYNERGY_HC_D0FIFOCTR
  Initial value: (0x2AUL)
- #define UX_SYNERGY_HC_D1FIFOSEL
  Initial value: (0x2CUL)
- #define UX_SYNERGY_HC_D1FIFOCTR
  Initial value: (0x2EUL)
- #define UX_SYNERGY_HC_INTENB0
  Initial value: (0x30UL)
- #define UX_SYNERGY_HC_INTENB1
  Initial value: (0x32UL)
- #define UX_SYNERGY_HC_BRDYENB
  Initial value: (0x36UL)
- #define UX_SYNERGY_HC_NRDYENB
  Initial value: (0x38UL)
- #define UX_SYNERGY_HC_BEMPENB
  Initial value: (0x3AUL)
- #define UX_SYNERGY_HC_SOFCFG
  Initial value: (0x3CUL)
- #define UX_SYNERGY_HC_PHYSET
  Initial value: (0x3EUL)
- #define UX_SYNERGY_HC_INTSTS0
  Initial value: (0x40UL)
- #define UX_SYNERGY_HC_INTSTS1
  Initial value: (0x42UL)
- #define UX_SYNERGY_HC_BRDYSTS
  Initial value: (0x46UL)
- #define UX_SYNERGY_HC_NRDYSTS
  Initial value: (0x48UL)

- #define UX_SYNERGY_HC_BEMPSTS
  Initial value: (0x4AUL)
- #define UX_SYNERGY_HC_FRMNUM
  Initial value: (0x4CUL)
- #define UX_SYNERGY_HC_DVCHGR
  Initial value: (0x4EUL)
- #define UX_SYNERGY_HC_USBADDR
  Initial value: (0x50UL)
- #define UX_SYNERGY_HC_USBREQ
  Initial value: (0x54UL)
- #define UX_SYNERGY_HC_USBVAL
  Initial value: (0x56UL)
- #define UX_SYNERGY_HC_USBINDX
  Initial value: (0x58UL)
- #define UX_SYNERGY_HC_USBLENG
  Initial value: (0x5AUL)
- #define UX_SYNERGY_HC_DCPCFG
  Initial value: (0x5CUL)
- #define UX_SYNERGY_HC_DCPMAXP
  Initial value: (0x5EUL)
- #define UX_SYNERGY_HC_DCPCTR
  Initial value: (0x60UL)
- #define UX_SYNERGY_HC_PIPESEL
  Initial value: (0x64UL)
- #define UX_SYNERGY_HC_PIPECFG
  Initial value: (0x68UL)
- #define UX_SYNERGY_HC_PIPEBUF
  Initial value: (0x6AUL)
- #define UX_SYNERGY_HC_PIPEMAXP
  Initial value: (0x6CUL)
- #define UX_SYNERGY_HC_PIPEPERI
  Initial value: (0x6EUL)
- #define UX_SYNERGY_HC_PIPE1CTR
  Initial value: (0x70UL)

- #define UX_SYNERGY_HC_PIPE2CTR
  Initial value: `(0x72UL)`
- #define UX_SYNERGY_HC_PIPE3CTR
  Initial value: `(0x74UL)`
- #define UX_SYNERGY_HC_PIPE4CTR
  Initial value: `(0x76UL)`
- #define UX_SYNERGY_HC_PIPE5CTR
  Initial value: `(0x78UL)`
- #define UX_SYNERGY_HC_PIPE6CTR
  Initial value: `(0x7AUL)`
- #define UX_SYNERGY_HC_PIPE7CTR
  Initial value: `(0x7CUL)`
- #define UX_SYNERGY_HC_PIPE8CTR
  Initial value: `(0x7EUL)`
- #define UX_SYNERGY_HC_PIPE9CTR
  Initial value: `(0x80UL)`
- #define UX_SYNERGY_HC_PIPE1TRE
  Initial value: `(0x90UL)`
- #define UX_SYNERGY_HC_PIPE1TRN
  Initial value: `(0x92UL)`
- #define UX_SYNERGY_HC_PIPE2TRE
  Initial value: `(0x94UL)`
- #define UX_SYNERGY_HC_PIPE2TRN
  Initial value: `(0x96UL)`
- #define UX_SYNERGY_HC_PIPE3TRE
  Initial value: `(0x98UL)`
- #define UX_SYNERGY_HC_PIPE3TRN
  Initial value: `(0x9AUL)`
- #define UX_SYNERGY_HC_PIPE4TRE
  Initial value: `(0x9CUL)`
- #define UX_SYNERGY_HC_PIPE4TRN
  Initial value: `(0x9EUL)`
- #define UX_SYNERGY_HC_PIPE5TRE
  Initial value: `(0xA0UL)`

- #define UX_SYNERGY_HC_PIPE5TRN

  Initial value: `(0xA2UL)`
- #define UX_SYNERGY_HC_USBMC

  Initial value: `(0xCCUL)`
- #define UX_SYNERGY_HC_DEVADD0

  Initial value: `(0xD0UL)`
- #define UX_SYNERGY_HC_DEVADD1

  Initial value: `(0xD2UL)`
- #define UX_SYNERGY_HC_DEVADD2

  Initial value: `(0xD4UL)`
- #define UX_SYNERGY_HC_DEVADD3

  Initial value: `(0xD6UL)`
- #define UX_SYNERGY_HC_DEVADD4

  Initial value: `(0xD8UL)`
- #define UX_SYNERGY_HC_DEVADD5

  Initial value: `(0xDAUL)`
- #define UX_SYNERGY_HC_PHYSLEW

  Initial value: `(0xF0UL)`
- #define UX_SYNERGY_HC_LPSTS

  Initial value: `(0x102UL)`
- #define UX_SYNERGY_HC_PFKUSB_ENABLED

  Initial value: `(1U<<4)`
- #define UX_SYNERGY_HC_PFKUSB_MODE_HOST

  Initial value: `(1)`
- #define UX_SYNERGY_HC_BUSWAIT_MASK

  Initial value: `(0xFU)`
- #define UX_SYNERGY_HC_BUSWAIT_CALC_FREQ_PCLK_CYC

  Initial value: `(17142857U)`
- #define UX_SYNERGY_HC_BUSWAIT_CALC_FREQ_PCLK_CYCX10

  Initial value: `(1714288U)`
- #define UX_SYNERGY_HC_SYSCFG_SCKE

  Initial value: `(1U<<10)`
- #define UX_SYNERGY_HC_SYSCFG_CNEN

  Initial value: `(1U<<8)`

- #define UX_SYNERGY_HC_SYSCFG_HSE
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_HC_SYSCFG_DCFM
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_HC_SYSCFG_DRPD
  Initial value: `(1U<<5)`
- #define UX_SYNERGY_HC_SYSCFG_DPRPU
  Initial value: `(1U<<4)`
- #define UX_SYNERGY_HC_SYSCFG_USBE
  Initial value: `(1U<<0)`
- #define UX_SYNERGY_HC_SYSSTS0_LNST
  Initial value: `(3)`
- #define UX_SYNERGY_HC_SYSSTS0_IDMON
  Initial value: `(1U<<2)`
- #define UX_SYNERGY_HC_SYSSTS0_SOFEA
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_HC_SYSSTS0_HTACT
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_HC_SYSSTS0_OVCMON
  Initial value: `(0xC0000U)`
- #define UX_SYNERGY_HC_SYSSTS0_LNST_J_STATE_FS
  Initial value: `(1U)`
- #define UX_SYNERGY_HC_SYSSTS0_LNST_J_STATE_LS
  Initial value: `(2U)`
- #define UX_SYNERGY_HC_PLLSTA_PLLLOCK
  Initial value: `(1U<<0)`
- #define UX_SYNERGY_HC_DVSTCTR0_HNPBTOA
  Initial value: `(1U<<11)`
- #define UX_SYNERGY_HC_DVSTCTR0_EXICEN
  Initial value: `(1U<<10)`
- #define UX_SYNERGY_HC_DVSTCTR0_VBUSEN
  Initial value: `(1U<<9)`
- #define UX_SYNERGY_HC_DVSTCTR0_WKUP
  Initial value: `(1U<<8)`

- #define UX_SYNERGY_HC_DVSTCTR0_RWUPE
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_HC_DVSTCTR0_USBRST
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_HC_DVSTCTR0_RESUME
  Initial value: `(1U<<5)`
- #define UX_SYNERGY_HC_DVSTCTR0_UACT
  Initial value: `(1U<<4)`
- #define UX_SYNERGY_HC_DVSTCTR0_RHST
  Initial value: `(0x7U)`
- #define UX_SYNERGY_HC_DVSTCTR0_SPEED_LOW
  Initial value: `(1)`
- #define UX_SYNERGY_HC_DVSTCTR0_SPEED_FULL
  Initial value: `(2)`
- #define UX_SYNERGY_HC_DVSTCTR0_SPEED_HIGH
  Initial value: `(3)`
- #define UX_SYNERGY_HC_DVSTCTR0_RESET_IN_PROGRESS
  Initial value: `(4)`
- #define UX_SYNERGY_HC_CFIFOSEL_RCNT
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_HC_CFIFOSEL_REW
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_HC_CFIFOSEL_MBW_8
  Initial value: `(0U<<10)`
- #define UX_SYNERGY_HC_CFIFOSEL_MBW_16
  Initial value: `(1U<<10)`
- #define UX_SYNERGY_HC_CFIFOSEL_MBW_32
  Initial value: `(2U<<10)`
- #define UX_SYNERGY_HC_CFIFOSEL_MBW_MASK
  Initial value: `(3U<<10)`
- #define UX_SYNERGY_HC_CFIFOSEL_BIGEND
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_HC_CFIFOSEL_ISEL
  Initial value: `(1U<<5)`

- #define UX_SYNERGY_HC_CFIFOSEL_CURPIPE_MASK
  Initial value: `(0xFU)`
- #define UX_SYNERGY_HC_DFIFOSEL_RCNT
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_HC_DFIFOSEL_REW
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_HC_DFIFOSEL_DCLRM
  Initial value: `(1U<<13)`
- #define UX_SYNERGY_HC_DFIFOSEL_DREQE
  Initial value: `(1U<<12)`
- #define UX_SYNERGY_HC_DFIFOSEL_MBW_8
  Initial value: `(0U<<10)`
- #define UX_SYNERGY_HC_DFIFOSEL_MBW_16
  Initial value: `(1U<<10)`
- #define UX_SYNERGY_HC_DFIFOSEL_MBW_32
  Initial value: `(2U<<10)`
- #define UX_SYNERGY_HC_DFIFOSEL_MBW_MASK
  Initial value: `(3U<<10)`
- #define UX_SYNERGY_HC_DFIFOSEL_BIGEND
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_HC_DFIFOSEL_CURPIPE_MASK
  Initial value: `(0xF)`
- #define UX_SYNERGY_HC_FIFOCTR_BVAL
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_HC_FIFOCTR_BCLR
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_HC_FIFOCTR_FRDY
  Initial value: `(1U<<13)`
- #define UX_SYNERGY_HC_FIFOCTR_DTLN
  Initial value: `(0xFFF)`
- #define UX_SYNERGY_HC_INTENB0_VBSE
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_HC_INTENB0_RSME
  Initial value: `(1U<<14)`

- #define UX_SYNERGY_HC_INTENB0_SOFE
  Initial value: `(1U<<13)`
- #define UX_SYNERGY_HC_INTENB0_DVSE
  Initial value: `(1U<<12)`
- #define UX_SYNERGY_HC_INTENB0_CTRE
  Initial value: `(1U<<11)`
- #define UX_SYNERGY_HC_INTENB0_BEMPE
  Initial value: `(1U<<10)`
- #define UX_SYNERGY_HC_INTENB0_NRDYE
  Initial value: `(1U<<9)`
- #define UX_SYNERGY_HC_INTENB0_BRDYE
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_HC_INTENB1_OVRCRE
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_HC_INTENB1_BCHGE
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_HC_INTENB1_DTCHE
  Initial value: `(1U<<12)`
- #define UX_SYNERGY_HC_INTENB1_ATTCHE
  Initial value: `(1U<<11)`
- #define UX_SYNERGY_HC_INTENB1_L1RSMENDE
  Initial value: `(1U<<9)`
- #define UX_SYNERGY_HC_INTENB1_LPSMENDE
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_HC_INTENB1_EOFERRE
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_HC_INTENB1_SIGNE
  Initial value: `(1U<<5)`
- #define UX_SYNERGY_HC_INTENB1_SACKE
  Initial value: `(1U<<4)`
- #define UX_SYNERGY_HC_INTENB1_PDDETINTE
  Initial value: `(1U<<0)`
- #define UX_SYNERGY_HC_PIPE0
  Initial value: `(1U<<0)`

- #define UX_SYNERGY_HC_PIPE_ALL
  Initial value: `(0x3FF)`
- #define UX_SYNERGY_HC_SOFCFG_TRNENSEL
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_HC_SOFCFG_BRDYM
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_HC_SOFCFG_INTL
  Initial value: `(1U<<5)`
- #define UX_SYNERGY_HC_SOFCFG_EDGESTS
  Initial value: `(1U<<4)`
- #define UX_SYNERGY_HC_PHYSET_HSEB
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_HC_PHYSET_REPSTART
  Initial value: `(1U<<11)`
- #define UX_SYNERGY_HC_PHYSET_REPSEL
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_HC_PHYSET_CLKSEL_1
  Initial value: `(1U<<5)`
- #define UX_SYNERGY_HC_PHYSET_CLKSEL_0
  Initial value: `(1U<<4)`
- #define UX_SYNERGY_HC_PHYSET_CDPEN
  Initial value: `(1U<<3)`
- #define UX_SYNERGY_HC_PHYSET_PLLRESET
  Initial value: `(1U<<1)`
- #define UX_SYNERGY_HC_PHYSET_DIRPD
  Initial value: `(1U<<0)`
- #define UX_SYNERGY_HC_INTSTS0_VBINT
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_HC_INTSTS0_RESM
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_HC_INTSTS0_SOFR
  Initial value: `(1U<<13)`
- #define UX_SYNERGY_HC_INTSTS0_DVST
  Initial value: `(1U<<12)`

- #define UX_SYNERGY_HC_INTSTS0_CTRT
  Initial value: `(1U<<11)`
- #define UX_SYNERGY_HC_INTSTS0_BEMP
  Initial value: `(1U<<10)`
- #define UX_SYNERGY_HC_INTSTS0_NRDY
  Initial value: `(1U<<9)`
- #define UX_SYNERGY_HC_INTSTS0_BRDY
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_HC_INTSTS0_VBSTS
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_HC_INTSTS0_VALID
  Initial value: `(1U<<3)`
- #define UX_SYNERGY_HC_INTSTS0_CTSQ_MASK
  Initial value: `(7)`
- #define UX_SYNERGY_HC_INTSTS0_ALL_CLEAR
  Initial value: `(0U)`
- #define UX_SYNERGY_HC_INTSTS1_OVRCRE
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_HC_INTSTS1_BCHG
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_HC_INTSTS1_DTCH
  Initial value: `(1U<<12)`
- #define UX_SYNERGY_HC_INTSTS1_ATTCH
  Initial value: `(1U<<11)`
- #define UX_SYNERGY_HC_INTSTS1_EOFERR
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_HC_INTSTS1_SIGN
  Initial value: `(1U<<5)`
- #define UX_SYNERGY_HC_INTSTS1_SACK
  Initial value: `(1U<<4)`
- #define UX_SYNERGY_HC_INTSTS1_ALL_CLEAR
  Initial value: `(0U)`
- #define UX_SYNERGY_HC_FRMNUM_OVRN
  Initial value: `(1U<<15)`

- #define UX_SYNERGY_HC_FRMNUM_CRCE
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_HC_FRMNUM_FRNM_MASK
  Initial value: `(0x7FFU)`
- #define UX_SYNERGY_HC_DCPCFG_DIR
  Initial value: `(1U<<4)`
- #define UX_SYNERGY_HC_DCPMAXP_DEVADDR_SHIFT
  Initial value: `(12U)`
- #define UX_SYNERGY_HC_DCPMAXP_DEVADDR_MASK
  Initial value: `(0xf000U)`
- #define UX_SYNERGY_HC_DCPCTR_BSTS
  Initial value: `(1U<<15)`
- #define UX_SYNERGY_HC_DCPCTR_SUREQ
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_HC_DCPCTR_SUREQCLR
  Initial value: `(1U<<11)`
- #define UX_SYNERGY_HC_DCPCTR_SQCLR
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_HC_DCPCTR_SQSET
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_HC_DCPCTR_SQMON
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_HC_DCPCTR_PBUSY
  Initial value: `(1U<<5)`
- #define UX_SYNERGY_HC_DCPCTR_CCPL
  Initial value: `(1U<<2)`
- #define UX_SYNERGY_HC_DCPCTR_PID_MASK
  Initial value: `(3UL)`
- #define UX_SYNERGY_HC_DCPCTR_PIDNAK
  Initial value: `(0UL)`
- #define UX_SYNERGY_HC_DCPCTR_PIDBUF
  Initial value: `(1UL)`
- #define UX_SYNERGY_HC_DCPCTR_PIDSTALL
  Initial value: `(2UL)`

- #define UX_SYNERGY_HC_DCPCTR_PIDSTALL2
  Initial value: `(3UL)`
- #define UX_SYNERGY_HC_PIPECFG_TYPE_MASK
  Initial value: `(0xC000)`
- #define UX_SYNERGY_HC_PIPECFG_TYPE_BIT_USED
  Initial value: `(0)`
- #define UX_SYNERGY_HC_PIPECFG_TYPE_BULK
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_HC_PIPECFG_TYPE_INTERRUPT
  Initial value: `(2U<<14)`
- #define UX_SYNERGY_HC_PIPECFG_TYPE_ISOCHRONOUS
  Initial value: `(3U<<14)`
- #define UX_SYNERGY_HC_PIPECFG_BFRE
  Initial value: `(1U<<10)`
- #define UX_SYNERGY_HC_PIPECFG_DBLB
  Initial value: `(1U<<9)`
- #define UX_SYNERGY_HC_PIPECFG_CNTMD
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_HC_PIPECFG_SHTNAK
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_HC_PIPECFG_DIR
  Initial value: `(1U<<4)`
- #define UX_SYNERGY_HC_PIPECFG_EPNUM_MASK
  Initial value: `(0xFU)`
- #define UX_SYNERGY_HC_PIPEBUF_SIZEMASK
  Initial value: `(0x1FU<<10)`
- #define UX_SYNERGY_HC_PIPEBUF_BUFNMBMASK
  Initial value: `(0xFFU<<10)`
- #define UX_SYNERGY_HC_PIPEBUF_SHIFT
  Initial value: `(10U)`
- #define UX_SYNERGY_HC_PIPEMAXP_DEVSELMASK
  Initial value: `(0xFU<<12)`
- #define UX_SYNERGY_HC_PIPEMAXP_DEVSEL_SHIFT
  Initial value: `(12U)`

- #define UX_SYNERGY_HC_PIPEMAXP_MXPSMASK
  Initial value: `(0x7FF)`
- #define UX_SYNERGY_HC_PIPETRE_TRENB
  Initial value: `(1U<<9)`
- #define UX_SYNERGY_HC_FIFO_D0
  Initial value: `(0UL)`
- #define UX_SYNERGY_HC_FIFO_D1
  Initial value: `(1UL)`
- #define UX_SYNERGY_HC_FIFO_C
  Initial value: `(2UL)`
- #define UX_SYNERGY_HC_DEVADD_SPEED_LOW
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_HC_DEVADD_SPEED_FULL
  Initial value: `(2U<<6)`
- #define UX_SYNERGY_HC_DEVADD_SPEED_HIGH
  Initial value: `(3U<<6)`
- #define UX_SYNERGY_HC_DEVADD_UPPHUB_SHIFT
  Initial value: `(11U)`
- #define UX_SYNERGY_HC_DEVADD_HUBPORT_SHIFT
  Initial value: `(8U)`
- #define UX_SYNERGY_HC_USBMC_VDCEN
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_HC_DCPCTR_DATA1
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_HC_DCPCTR_DATA0
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_HC_PIPESEL_NO_PIPE
  Initial value:`0x000f`
- #define UX_SYNERGY_HC_PIPE0_SIZE
  Initial value: `(256)`
- #define UX_SYNERGY_HC_PIPE_NB_BUFFERS
  Initial value: `(64)`
- #define UX_SYNERGY_HC_PIPECTR_BSTS
  Initial value: `(1U<<15)`

- #define UX_SYNERGY_HC_PIPECTR_INBUFM
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_HC_PIPECTR_ATREPM
  Initial value: `(1U<<10)`
- #define UX_SYNERGY_HC_PIPECTR_ACLRM
  Initial value: `(1U<<9)`
- #define UX_SYNERGY_HC_PIPECTR_SQCLR
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_HC_PIPECTR_SQSET
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_HC_PIPECTR_SQMON
  Initial value: `(1U<<6)`
- #define UX_SYNERGY_HC_PIPECTR_PBUSY
  Initial value: `(1U<<5)`
- #define UX_SYNERGY_HC_PIPECTR_PID_MASK
  Initial value: `(3)`
- #define UX_SYNERGY_HC_PIPECTR_PIDNAK
  Initial value: `(0)`
- #define UX_SYNERGY_HC_PIPECTR_PIDBUF
  Initial value: `(1)`
- #define UX_SYNERGY_HC_PIPECTR_PIDSTALL
  Initial value: `(2)`
- #define UX_SYNERGY_HC_PIPECTR_PIDSTALL2
  Initial value: `(3)`
- #define UX_SYNERGY_HC_PIPECTR_DATA1
  Initial value: `(1U<<7)`
- #define UX_SYNERGY_HC_PIPECTR_DATA0
  Initial value: `(1U<<8)`
- #define UX_SYNERGY_HC_AVAILABLE_BANDWIDTH
  Initial value: `(2304UL)`
- #define UX_SYNERGY_HC_INIT_DELAY
  Initial value: `(1000)`
- #define UX_SYNERGY_HC_RESET_RETRY
  Initial value: `(1000)`

- #define UX_SYNERGY_HC_RESET_DELAY
  Initial value: `(10)`
- #define UX_SYNERGY_HC_PORT_RESET_RETRY
  Initial value: `(50)`
- #define UX_SYNERGY_HC_FORCE_PORT_RESET_RETRY
  Initial value: `(50)`
- #define UX_SYNERGY_HC_FORCE_PORT_RESET_DELAY
  Initial value: `(1)`
- #define UX_SYNERGY_HC_CHECK_PORT_RESET_RETRY
  Initial value: `(500)`
- #define UX_SYNERGY_HC_PORT_RESET_DELAY
  Initial value: `(300)`
- #define UX_SYNERGY_HC_PORT_RESET_RECOVERY_DELAY
  Initial value: `(100)`
- #define UX_SYNERGY_HC_COMMAND_STATUS_RESET
  Initial value: `(0)`
- #define UX_SYNERGY_HC_INIT_RESET_DELAY
  Initial value: `(10)`
- #define UX_SYNERGY_HC_MAX_BUF_SIZE
  Initial value: `(64)`
- #define UX_SYNERGY_HC_BUF_BLOCK_SIZE
  Initial value: `(64)`
- #define UX_SYNERGY_HC_MAX_BUF_SIZE_PIPE1_to_2_FS
  Initial value: `(256)`
- #define UX_SYNERGY_HC_MAX_BUF_SIZE_PIPE3_to_9_FS
  Initial value: `(64)`
- #define UX_SYNERGY_HC_MAX_BUF_SIZE_PIPE1_to_2_HS
  Initial value: `(1024)`
- #define UX_SYNERGY_HC_MAX_BUF_SIZE_PIPE3_to_5_HS
  Initial value: `(512)`
- #define UX_SYNERGY_HC_MAX_BUF_SIZE_PIPE6_to_9_HS
  Initial value: `(64)`
- #define UX_SYNERGY_HC_MAX_BUF_NUM
  Initial value: `(135)`

- #define UX_SYNERGY_HC_PIPE1_BUF_START_NUM
  Initial value: `(8)`
- #define UX_SYNERGY_HC_FIFO_WRITING
  Initial value: `(2)`
- #define UX_SYNERGY_HC_FIFO_WRITE_END
  Initial value: `(3)`
- #define UX_SYNERGY_HC_FIFO_WRITE_SHORT
  Initial value: `(4)`
- #define UX_SYNERGY_HC_FIFO_WRITE_DMA
  Initial value: `(5)`
- #define UX_SYNERGY_HC_FIFO_WRITE_ERROR
  Initial value: `(6)`
- #define UX_SYNERGY_HC_FIFO_READING
  Initial value: `(2)`
- #define UX_SYNERGY_HC_FIFO_READ_END
  Initial value: `(3)`
- #define UX_SYNERGY_HC_FIFO_READ_SHORT
  Initial value: `(4)`
- #define UX_SYNERGY_HC_FIFO_READ_DMA
  Initial value: `(5)`
- #define UX_SYNERGY_HC_FIFO_READ_ERROR
  Initial value: `(6)`
- #define UX_SYNERGY_HC_FIFO_READ_OVER
  Initial value: `(7)`
- #define UX_SYNERGY_HC_ED_BRDY
  Initial value: `(0x00000001U)`
- #define UX_SYNERGY_HC_ED_NRDY
  Initial value: `(0x00000002U)`
- #define UX_SYNERGY_HC_ED_BEMP
  Initial value: `(0x00000004U)`
- #define UX_SYNERGY_HC_ED_EOFERR
  Initial value: `(0x00000010U)`
- #define UX_SYNERGY_HC_ED_SIGN
  Initial value: `(0x00000020U)`

- #define UX_SYNERGY_HC_ED_SACK
  Initial value: `(0x00000040U)`
- #define UX_SYNERGY_HC_ED_TIMEOUT
  Initial value: `(0x00000080U)`
- #define UX_SYNERGY_HC_PHYSLEW_SLEW_SLEWR00
  Initial value: `(1U<<0)`
- #define UX_SYNERGY_HC_PHYSLEW_SLEW_SLEWR01
  Initial value: `(1U<<1)`
- #define UX_SYNERGY_HC_PHYSLEW_SLEW_SLEWF00
  Initial value: `(1U<<2)`
- #define UX_SYNERGY_HC_PHYSLEW_SLEW_SLEWF01
  Initial value: `(1U<<3)`
- #define UX_SYNERGY_HC_LPSTS_SUSPENDM
  Initial value: `(1U<<14)`
- #define UX_SYNERGY_HC_PORT_ENABLED
  Initial value: `(1)`
- #define UX_SYNERGY_HC_PORT_DISABLED
  Initial value: `(0)`
- #define UX_SYNERGY_ED_STATIC
  Initial value: `(0x80000000)`
- #define UX_SYNERGY_ED_SKIP
  Initial value: `(0x40000000UL)`
- #define UX_SYNERGY_TD_SETUP_PHASE
  Initial value: `(0x00010000)`
- #define UX_SYNERGY_TD_DATA_PHASE
  Initial value: `(0x00020000)`
- #define UX_SYNERGY_TD_STATUS_PHASE
  Initial value: `(0x00040000)`
- #define UX_SYNERGY_TD_OUT
  Initial value: `(0x00000800)`
- #define UX_SYNERGY_TD_IN
  Initial value: `(0x00001000)`
- #define UX_SYNERGY_TD_TOGGLE_FROM_ED
  Initial value: `(0x80000000)`

- #define UX_SYNERGY_TD_ACK_PENDING
  Initial value: `(0x00080000)`
- #define UX_SYNERGY_TD_SETUP_TYPE
  Initial value: `(1)`
- #define UX_SYNERGY_TD_DATA_TYPE
  Initial value: `(2)`
- #define UX_SYNERGY_TD_STATUS_TYPE
  Initial value: `(3)`
- #define UX_SYNERGY_TD_MAX_ERROR
  Initial value: `(3)`

7.17.3 usbhs_usb_int_resume_isr

```
usbhs_usb_int_resume_isr ( void )
```

7.17.3.1 Brief description

This function calls the host interrupt handler or the device interrupt handler.

7.17.4 usbfs_int_isr

```
usbfs_int_isr ( void )
```

7.17.4.1 Brief description

All the USBFS interrupts are handled by this ISR.

7.17.5 usbfs_resume_isr

```
usbfs_resume_isr ( void )
```

7.17.5.1 Brief description

VBINT(VBUS interrupt), RESM(Resume interrupt) OVRCR(Over current input), BCHG(Change interrupt), and PDDETINT0(Bus change interrupt) fire this ISR. This is only used for canceling following standby modes.

7.17.5.2 Detailed description

- Canceling the software standby mode
- Canceling deep standby mode

7.17.6 ux_dcd_synergy_buffer_empty_interrupt

```
ux_dcd_synergy_buffer_empty_interrupt (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_ED * ed ,   ULONG flag )
```

7.17.6.1 Brief description

This function processes the BEMP interrupt for the specific endpoint.

7.17.6.2 Detailed description

Table 603:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to a DCD control block
ed	in	Pointer to physical Endpoint(ED) control block
flag	in	Flag to enable or disable the buffer empty interrupt.

7.17.7 ux_dcd_synergy_buffer_notready_interrupt

```
ux_dcd_synergy_buffer_notready_interrupt (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_ED * ed ,   ULONG flag )
```

7.17.7.1 Brief description

This function processes the NRDY(Not Ready) interrupt for the specific endpoint.

7.17.7.2 Detailed description

Table 604:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to a DCD control block
ed	in	Pointer to physical Endpoint(ED) control block

Table 604:Parameters (Continued)

Name	Direction	Description
flag	in	Flag for DCD synergy enable or disable.

7.17.8 ux_dcd_synergy_buffer_read

```
ux_dcd_synergy_buffer_read (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_ED * ed )
```

7.17.8.1 Brief description

This function reads from a specified pipe into a buffer.

7.17.8.2 Detailed description

Table 605:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to a DCD control block
ed	in	Pointer to a physical Endpoint(ED) control block

Table 606:Return values

Name	Description
UX_SUCCESS	Read a data from buffer successfully.
UX_ERROR	Unable to read a data from buffer.

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_dcd_synergy_fifo_read

7.17.9 ux_dcd_synergy_buffer_ready_interrupt

```
ux_dcd_synergy_buffer_ready_interrupt (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_ED * ed ,   ULONG flag )
```

7.17.9.1 Brief description

This function enable or disable the BRDY(Ready) interrupt for the pipe.

7.17.9.2 Detailed description

Table 607:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to a DCD control block
ed	in	Pointer to a physical Endpoint(ED) control block
flag	in	Check whether DCD synergy is enable or disable.

7.17.10 ux_dcd_synergy_buffer_write

```
ux_dcd_synergy_buffer_write (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_ED * ed )
```

7.17.10.1 Brief description

This function writes a data from input buffer to the specified PIPE.

7.17.10.2 Detailed description

Table 608:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to a DCD control block
ed	in	Pointer to a physical Endpoint(ED) control block

Table 609:Return values

Name	Description
UX_SUCCESS	Write a data to FIFO(D0, D1 and C) successfully.

Table 609:Return values (Continued)

Name	Description
UX_ERROR	Unable to write a data to FIFO(D0, D1 and C).

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_dcd_synergy_fifod_write
- ux_dcd_synergy_fifoc_write

7.17.11 ux_dcd_synergy_current_endpoint_change

```
ux_dcd_synergy_current_endpoint_change (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_ED * ed ,   ULONG direction )
```

7.17.11.1 Brief description

This function configures the FIFO as per the request specified in endpoint descriptor.

7.17.11.2 Detailed description

Table 610:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to a DCD control block
ed	in	Pointer to a physical Endpoint(ED) control block
direction	in	Endpoint direction

7.17.12 ux_dcd_synergy_data_buffer_size

```
ux_dcd_synergy_data_buffer_size (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_ED * ed )
```

7.17.12.1 Brief description

This function returns the size of the data buffer and selects the specified pipe.

7.17.12.2 Detailed description

Table 611:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to a DCD control block
ed	in	Pointer to a physical Endpoint(ED) control block

Table 612:Return values

Name	Description
buffer_size	Maximum packet size.

7.17.13 ux_dcd_synergy_endpoint_create

```
ux_dcd_synergy_endpoint_create (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_SLAVE_ENDPOINT * endpoint )
```

7.17.13.1 Brief description

This function creates a physical endpoint.

7.17.13.2 Detailed description

Table 613:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to a DCD control block
endpoint	in	Pointer to a Device Controller Endpoint structure.

Table 614:Return values

Name	Description
UX_SUCCESS	Endpoint is created successfully.
UX_ERROR	Buffer is not free or endpoint creation is unsuccessful.
UX_NO_ED_AVAILABLE	Endpoint is already in use.

7.17.14 ux_dcd_synergy_endpoint_destroy

```
ux_dcd_synergy_endpoint_destroy (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_SLAVE_ENDPOINT * endpoint )
```

7.17.14.1 Brief description

This function will destroy a physical endpoint.

7.17.14.2 Detailed description

Table 615:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to a DCD control block
endpoint	in	Pointer to a Device Controller Endpoint structure.

Table 616:Return values

Name	Description
UX_SUCCESS	Endpoint is destroyed successfully.

7.17.15 ux_dcd_synergy_endpoint_nak_set

```
ux_dcd_synergy_endpoint_nak_set (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_ED * ed )
```

7.17.15.1 Brief description

This function sets a NAK(Not Acknowledged) to an endpoint.

7.17.15.2 Detailed description

Table 617:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to a DCD control block
ed	in	Pointer to a physical Endpoint(ED) control block

7.17.16 ux_dcd_synergy_endpoint_reset

```
ux_dcd_synergy_endpoint_reset (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_SLAVE_ENDPOINT * endpoint )
```

7.17.16.1 Brief description

This function will reset a physical endpoint.

7.17.16.2 Detailed description

Table 618:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to a DCD control block
endpoint	in	Pointer to a Device Controller Endpoint structure.

Table 619:Return values

Name	Description
UX_SUCCESS	Endpoint is reset successfully.
UX_NO_ED_AVAILABLE	Device Controller Endpoint structure pointer is NULL

7.17.17 ux_dcd_synergy_endpoint_stall

```
ux_dcd_synergy_endpoint_stall (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_SLAVE_ENDPOINT * endpoint )
```

7.17.17.1 Brief description

This function will stall a physical endpoint.

7.17.17.2 Detailed description

Table 620:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to a DCD control block
endpoint	in	Pointer to a Device Controller Endpoint structure.

Table 621:Return values

Name	Description
UX_SUCCESS	Endpoint is stalled successfully.
UX_NO_ED_AVAILABLE	Device Controller Endpoint control pointer is Null.

7.17.18 ux_dcd_synergy_endpoint_status

```
ux_dcd_synergy_endpoint_status (  UX_DCD_SYNERGY * dcd_synergy ,
 ULONG endpoint_index )
```

7.17.18.1 Brief description

This function will retrieve the status of the endpoint.

7.17.18.2 Detailed description

Table 622:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to a DCD control block
endpoint_index	in	Endpoint number who's status is to be known.

Table 623:Return values

Name	Description
UX_ERROR	Endpoint already in use.
UX_FALSE	Endpoint is stalled.
UX_TRUE	Endpoint is not stalled.

7.17.19 ux_dcd_synergy_fifo_port_change

```
ux_dcd_synergy_fifo_port_change (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_ED * ed ,   ULONG direction )
```

7.17.19.1 Brief description

This function configures the FIFO port.

7.17.19.2 Detailed description

Table 624:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to a DCD control block
ed	in	Pointer to a physical Endpoint(ED) control block
direction	in	Direction to switch

Table 625:Return values

Name	Description
UX_ERROR	Unable to change fifo port.

synergy_register Current endpoint index(pipe)

7.17.20 ux_dcd_synergy_fifo_read

```
 ux_dcd_synergy_fifo_read (  UX_DCD_SYNERGY * dcd_synergy ,   UX_DCD_SYNERGY_ED
* ed )
```

7.17.20.1 Brief description

This function reads from the hardware FIFO C, D0 or D1 and stores in the destination buffer.

7.17.20.2 Detailed description

Table 626:Parameters

Name	Direction	Description
dcd_synergy	inout	: Pointer to a DCD control block
ed	inout	: Pointer to a physical Endpoint(ED) control block

Table 627:Return values

Name	Description
UX_ERROR	FIFO is not accessible.
UX_SYNERGY_DCD_FIFO_READ_OVER	FIFO read overflow.
UX_SYNERGY_DCD_FIFO_READ_SHORT	Short packet is received.
UX_SYNERGY_DCD_FIFO_READING	Continue reading FIFO.

7.17.21 ux_dcd_synergy_fifo_read_dma

```
ux_dcd_synergy_fifo_read_dma (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_ED * ed ,   UINT dma_bytes_to_transfer )
```

7.17.21.1 Brief description

This function reads the data from HW D0/ D1 FIFO using DMA.

7.17.21.2 Detailed description

Table 628:Parameters

Name	Direction	Description
dcd_synergy	inout	: Pointer to a DCD control block
ed	inout	: Pointer to a physical Endpoint(ED) control block
dma_bytes_to_transfer	inout	: No of bytes to be transferred using DMA

Table 629:Return values

Name	Description
UX_ERROR	FIFO is not accessible.
UX_SYNERGY_DCD_FIFO_READ_OVER	FIFO read overflow.
UX_SYNERGY_DCD_FIFO_READ_SHORT	Short packet is received.
UX_SYNERGY_DCD_FIFO_READING	Continue reading FIFO.

7.17.22 ux_dcd_synergy_read_dma_configure

```
ux_dcd_synergy_read_dma_configure (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_PAYLOAD_TRANSFER * p_payload ,   ULONG fifo_sel ,
 ULONG ep_buff_size )
```

7.17.22.1 Brief description

USBX DCD DMA read setup function. Call a subroutine for selected USB controller hardware.

7.17.22.2 Detailed description

Table 630:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to the DCD control block
p_payload	inout	Pointer to a payload transfer structure
fifo_sel	in	FIFO select register

7.17.23 ux_dcd_synergy_read_dma_set_16bit

```
ux_dcd_synergy_read_dma_set_16bit (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_PAYLOAD_TRANSFER * p_payload ,   ULONG fifo_sel ,
 ULONG ep_buff_size )
```

7.17.23.1 Brief description

USBX DCD DMA read setup function for USB hardwares with 16-bit FIFO.

7.17.23.2 Detailed description

Table 631:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to the DCD control block
p_payload	inout	Pointer to a payload transfer structure
fifo_sel	in	FIFO select register

7.17.24 ux_dcd_synergy_fifo_dma_start_read

```
 ux_dcd_synergy_fifo_dma_start_read (  UX_DCD_SYNERGY * dcd_synergy ,   UCHAR
* p_payload_buffer ,   VOID * p_fifo ,   VOID * p_fifo_ctrl ,   VOID
* p_fifo_sel )
```

7.17.24.1 Brief description

USBX DCD FIFO read - DMA start function.

7.17.24.2 Detailed description

Table 632:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to the DCD control block
p_payload_buffer	inout	Pointer to a payload buffer
p_fifo	in	FIFO register address

7.17.25 ux_dcd_synergy_fifo_read_software_copy

```
ux_dcd_synergy_fifo_read_software_copy (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_PAYLOAD_TRANSFER * p_payload ,   VOID * p_fifo ,
 ULONG fifo_sel )
```

7.17.25.1 Brief description

USBX DCD FIFO read by software copy. Call a subroutine for selected USB controller hardware.

7.17.25.2 Detailed description

Table 633:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to the DCD control block
p_payload	inout	Pointer to a payload transfer structure
p_fifo	in	FIFO register address
fifo_sel	in	FIFO select register

7.17.26 ux_dcd_synergy_fifo_read_software_copy_16bit

```
ux_dcd_synergy_fifo_read_software_copy_16bit (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_PAYLOAD_TRANSFER * p_payload ,   VOID * p_fifo ,
 ULONG fifo_sel )
```

7.17.26.1 Brief description

USBX DCD FIFO read - Software copy for USB hardwares with 16-bit FIFO.

7.17.26.2 Detailed description

Table 634:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to the DCD control block
p_payload	inout	Pointer to a payload transfer structure
p_fifo	in	FIFO register address
fifo_sel	in	FIFO select register

7.17.27 ux_dcd_synergy_fifo_read_last_bytes

```
 ux_dcd_synergy_fifo_read_last_bytes (  UX_DCD_SYNERGY_PAYLOAD_TRANSFER
* p_payload ,   VOID * p_fifo )
```

7.17.27.1 Brief description

USBX DCD FIFO read - Copy last bytes from FIFO by software if the rest bytes are less than FIFO access width.

7.17.27.2 Detailed description

Table 635:Parameters

Name	Direction	Description
p_payload	inout	Pointer to a payload transfer structure
p_fifo	in	FIFO register address

7.17.28 ux_dcd_synergy_fifoc_write

```
ux_dcd_synergy_fifoc_write (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_ED * ed )
```

7.17.28.1 Brief description

This function writes a data from a buffer into USB FIFO using CPU.

7.17.28.2 Detailed description

Table 636:Parameters

Name	Direction	Description
dcd_synergy	inout	: Pointer to a DCD control block
ed	inout	: Pointer to a physical Endpoint(ED) control block

Table 637:Return values

Name	Description
UX_ERROR	FIFO is not accessible.
UX_SYNERGY_DCD_FIFO_WRITE_END	Status for fifo write ends.
UX_SYNERGY_DCD_FIFO_WRITE_SHORT	Status for fifo short packet.
UX_SYNERGY_DCD_FIFO_WRITING	Status for fifo multiple writes.

7.17.29 ux_dcd_synergy_fifo_write_software_copy

```
ux_dcd_synergy_fifo_write_software_copy (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_PAYLOAD_TRANSFER * p_payload ,   VOID * p_fifo ,
 ULONG fifo_sel )
```

7.17.29.1 Brief description

USBX DCD FIFO write by software copy. Call a subroutine for selected USB controller hardware.

7.17.29.2 Detailed description

Table 638:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to the DCD control block
p_payload	inout	Pointer to a payload transfer structure
p_fifo	in	FIFO register address
fifo_sel	in	FIFO select register

7.17.30 ux_dcd_synergy_fifo_write_software_copy_16bit

```
ux_dcd_synergy_fifo_write_software_copy_16bit (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_PAYLOAD_TRANSFER * p_payload ,   VOID * p_fifo ,
 ULONG fifo_sel )
```

7.17.30.1 Brief description

USBX DCD FIFO write - Software copy for USB hardwares with 16-bit FIFO.

7.17.30.2 Detailed description

Table 639:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to the DCD control block
p_payload	inout	Pointer to a payload transfer structure
p_fifo	in	FIFO register address
fifo_sel	in	FIFO select register

7.17.31 ux_dcd_synergy_fifo_write_last_bytes

```
 ux_dcd_synergy_fifo_write_last_bytes (  UX_DCD_SYNERGY_PAYLOAD_TRANSFER
* p_payload ,   VOID * p_fifo ,   ULONG usb_base )
```

7.17.31.1 Brief description

USBX DCD FIFO write - Copy last bytes to FIFO by software if the rest bytes are less than FIFO access width.

7.17.31.2 Detailed description

Table 640:Parameters

Name	Direction	Description
p_payload	inout	Pointer to a payload transfer structure
p_fifo	in	FIFO register address
usb_base	in	USB controller hardware base address

7.17.32 ux_dcd_synergy_fifod_write

```
ux_dcd_synergy_fifod_write (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_ED * ed )
```

7.17.32.1 Brief description

This function writes a buffer to FIFOD0 or FIFOD1.

7.17.32.2 Detailed description

Table 641:Parameters

Name	Direction	Description
dcd_synergy	inout	: Pointer to a DCD control block
ed	inout	: Pointer to a physical Endpoint(ED) control block

Table 642:Return values

Name	Description
UX_ERROR	FIFO is not accessible.

Table 642:Return values (Continued)

Name	Description
UX_SYNERGY_DCD_FIFO_WRITE_END	Write ends of FIFO.
UX_SYNERGY_DCD_FIFO_WRITE_SHORT	Write short packet.
UX_SYNERGY_DCD_FIFO_WRITING	Continue multiple write to FIFO.

7.17.33 ux_dcd_synergy_fifod_write_dma

```
ux_dcd_synergy_fifod_write_dma (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_ED * ed ,   UINT dma_bytes_to_transfer )
```

7.17.33.1 Brief description

This function writes a buffer to FIFOD0 or FIFOD1 using DMA.

7.17.33.2 Detailed description

Table 643:Parameters

Name	Direction	Description
dcd_synergy	inout	: Pointer to a DCD control block
ed	inout	: Pointer to a physical Endpoint(ED) control block
dma_bytes_to_transfer	inout	: No of bytes to be transferred using DMA

Table 644:Return values

Name	Description
UX_ERROR	FIFO is not accessible.
UX_SYNERGY_DCD_FIFO_WRITE_END	Write ends of FIFO.
UX_SYNERGY_DCD_FIFO_WRITE_SHORT	Write short packet.
UX_SYNERGY_DCD_FIFO_WRITING	Continue multiple write to FIFO.

7.17.34 ux_dcd_synergy_write_dma_configure

```
ux_dcd_synergy_write_dma_configure (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_PAYLOAD_TRANSFER * p_payload ,   ULONG fifo_sel ,
 ULONG endpoint_size )
```

7.17.34.1 Brief description

USBX DCD DMA write setup function. Call a subroutine for selected USB controller hardware.

7.17.34.2 Detailed description

Table 645:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to the DCD control block
p_payload	inout	Pointer to a payload transfer structure
fifo_sel	in	FIFO select register
endpoint_size	in	Endpoint size

7.17.35 ux_dcd_synergy_write_dma_set_16bit

```
ux_dcd_synergy_write_dma_set_16bit (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_DCD_SYNERGY_PAYLOAD_TRANSFER * p_payload ,   ULONG fifo_sel ,
 ULONG endpoint_size )
```

7.17.35.1 Brief description

USBX DCD DMA write setup function for USB hardwares with 16-bit FIFO.

7.17.35.2 Detailed description

Table 646:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to the DCD control block

Table 646:Parameters (Continued)

Name	Direction	Description
p_payload	inout	Pointer to a payload transfer structure
fifo_sel	in	FIFO select register
endpoint_size	in	Endpoint size

7.17.36 ux_dcd_synergy_fifo_dma_start_write

```
 ux_dcd_synergy_fifo_dma_start_write (  UX_DCD_SYNERGY * dcd_synergy ,   UCHAR
* p_payload_buffer ,   VOID * p_fifo_add ,   VOID * p_fifo_ctrl ,   VOID
* p_fifo_sel )
```

7.17.36.1 Brief description

USBX DCD DMA FIFO write - DMA start function.

7.17.36.2 Detailed description

Table 647:Parameters

Name	Direction	Description
dcd_synergy	in	Pointer to the DCD control block
p_payload_buffer	inout	Pointer to a payload buffer
p_fifo_add	in	FIFO register address
p_fifo_ctrl	in	FIFO register control
p_fifo_sel	in	FIFO register select

7.17.37 ux_dcd_synergy_frame_number_get

```
 ux_dcd_synergy_frame_number_get (  UX_DCD_SYNERGY * dcd_synergy ,   ULONG
* frame_number )
```

7.17.37.1 Brief description

This function will return the frame number currently used by the controller. This function is mostly used for isochronous purposes.

7.17.37.2 Detailed description

Table 648:Parameters

Name	Direction	Description
dcd_synergy	inout	: Pointer to a DCD control block
frame_number	inout	: Pointer to a frame number in use

Table 649:Return values

Name	Description
UX_SUCCESS	In use frame number is returned successfully.

7.17.38 ux_dcd_synergy_function

```
 ux_dcd_synergy_function (  UX_SLAVE_DCD * dcd ,   UINT function ,   VOID
* parameter )
```

7.17.38.1 Brief description

This function act as interface between upper layer USBX device stack and synergy controller.

7.17.38.2 Detailed description

Table 650:Parameters

Name	Direction	Description
dcd	inout	: Pointer to a USBX control block.
function	inout	: Function requested to be despatched.
parameter	inout	: Pointer to requested function parameters.

Table 651:Return values

Name	Description
UX_CONTROLLER_UNKNOWN	Desired controller is not specified.
UX_FUNCTION_NOT_SUPPORTED	DCD function is not supported by the controller.
UX_SUCCESS	DCD function despatched successfully.

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_dcd_synergy_frame_number_get
- ux_dcd_synergy_transfer_request
- ux_dcd_synergy_endpoint_create
- ux_dcd_synergy_endpoint_destroy
- ux_dcd_synergy_endpoint_reset
- ux_dcd_synergy_endpoint_stall
- ux_dcd_synergy_endpoint_status

7.17.39 ux_dcd_synergy_initialize

```
ux_dcd_synergy_initialize (  ULONG dcd_io )
```

7.17.39.1 Brief description

This function initializes the USB slave controller for Renesas Synergy MCUs.

7.17.39.2 Detailed description

Table 652:Parameters

Name	Direction	Description
dcd_io	inout	: Address of DCD

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_dcd_synergy_initialize_common()

7.17.40 ux_dcd_synergy_initialize_transfer_support

```
ux_dcd_synergy_initialize_transfer_support (  ULONG dcd_io ,
 UX_DCD_SYNERGY_TRANSFER * p_transfer_instance )
```

7.17.40.1 Brief description

The function initializes the USB slave controller of the Renesas Synergy MCUs with associated DMA transfer modules.

7.17.40.2 Detailed description

Table 653:Parameters

Name	Direction	Description
dcd_io	in	Address of the USB controller.
p_transfer_instance	in	Pointer to Synergy Transfer module instances.

Table 654:Return values

Name	Description
UX_SUCCESS	Completed the USB controller initialization successfully.
UX_CONTROLLER_INIT_FAILED	Failed to initialize the USB controller.
UX_MEMORY_INSUFFICIENT	Memory was not allocated properly for the Synergy DCD instance.

7.17.41 ux_dcd_synergy_initialize_common

```
ux_dcd_synergy_initialize_common (  ULONG dcd_io )
```

7.17.41.1 Brief description

USBX DCD first half common part of the USB controller initialization.

7.17.41.2 Detailed description

Table 655:Parameters

Name	Direction	Description
dcd_io	in	Address of the USB controller.

Table 656:Return values

Name	Description
UX_SUCCESS	Completed the USB controller initialization successfully.
UX_CONTROLLER_INIT_FAILED	Failed to initialize the USB controller.
UX_MEMORY_INSUFFICIENT	Memory was not allocated properly for the Synergy DCD instance.

7.17.42 ux_dcd_synergy_initialize_common_complete

```
ux_dcd_synergy_initialize_common_complete (  void  )
```

7.17.42.1 Brief description

USBX DCD bottom-half common part of the USB controller initialization.

7.17.43 ux_dcd_dma_complete_tx

```
ux_dcd_dma_complete_tx ( transfer_callback_args_t  * p_args )
```

7.17.43.1 Brief description

USBX DCD DMA write callback function.

7.17.43.2 Detailed description

Table 657:Parameters

Name	Direction	Description
p_args	in	Pointer to the argument of a Transfer module callback function.

7.17.44 ux_dcd_dma_complete_rx

```
ux_dcd_dma_complete_rx ( transfer_callback_args_t  * p_args )
```

7.17.44.1 Brief description

USBX DCD DMA read callback function.

7.17.44.2 Detailed description

Table 658:Parameters

Name	Direction	Description
p_args	in	Pointer to the argument of a Transfer module callback function.

7.17.45 ux_dcd_synergy_initialize_complete

```
ux_dcd_synergy_initialize_complete (  VOID  )
```

7.17.45.1 Brief description

This function completes the initialization of the USB slave controller for the Renesas Synergy MCUs.

7.17.45.2 Detailed description

Table 659:Return values

Name	Description
UX_SUCCESS	USB slave is initialized successfully.

7.17.46 ux_dcd_synergy_interrupt_handler

```
ux_dcd_synergy_interrupt_handler (  VOID  )
```

7.17.46.1 Brief description

This function is the interrupt handler for the synergy controller. The controller will trigger an interrupt when something happens on an endpoint whose mask has been set in the interrupt enable register.

7.17.47 ux_dcd_synergy_register_clear

```
ux_dcd_synergy_register_clear (  UX_DCD_SYNERGY * dcd_synergy ,
 ULONG synergy_register ,   USHORT value )
```

7.17.47.1 Brief description

This function clears a bit in a register of the synergy.

7.17.47.2 Detailed description

Table 660:Parameters

Name	Direction	Description
dcd_synergy	inout	: Pointer to a DCD control block
synergy_register	inout	: Register to clear
value	inout	: Value to clear

7.17.48 ux_dcd_synergy_register_read

```
ux_dcd_synergy_register_read (  UX_DCD_SYNERGY * dcd_synergy ,
 ULONG synergy_register )
```

7.17.48.1 Brief description

This function reads a synergy USB register.

7.17.48.2 Detailed description

Table 661:Parameters

Name	Direction	Description
dcd_synergy	inout	: Pointer to a DCD control block
synergy_register	inout	: Register to read

Table 662:Return values

Name	Description
dcd_reg	Value read from USB register.

7.17.49 ux_dcd_synergy_register_set

```
ux_dcd_synergy_register_set (  UX_DCD_SYNERGY * dcd_synergy ,
 ULONG synergy_register ,   USHORT value )
```

7.17.49.1 Brief description

This function set a bit in synergy USB register.

7.17.49.2 Detailed description

Table 663:Parameters

Name	Direction	Description
dcd_synergy	inout	: Pointer to a DCD control block
synergy_register	inout	: Register to set
value	inout	: Value to set

7.17.50 ux_dcd_synergy_register_write

```
ux_dcd_synergy_register_write (  UX_DCD_SYNERGY * dcd_synergy ,
 ULONG synergy_register ,   USHORT value )
```

7.17.50.1 Brief description

This function writes a bit in synergy USB register.

7.17.50.2 Detailed description

Table 664:Parameters

Name	Direction	Description
dcd_synergy	inout	: Pointer to a DCD control block
synergy_register	inout	: Register to write
value	inout	: Value to write

7.17.51 ux_dcd_synergy_transfer_callback

```
ux_dcd_synergy_transfer_callback (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_SLAVE_TRANSFER * transfer_request ,   ULONG interrupt_status ,
 ULONG ctsq_mask )
```

7.17.51.1 Brief description

This function is invoked under ISR when an event happens on a specific endpoint.

7.17.51.2 Detailed description

Table 665:Parameters

Name	Direction	Description
dcd_synergy	inout	: Pointer to a DCD control block
transfer_request	inout	: Pointer to USBX Device Transfer Request structure
interrupt_status	inout	: Check if we have SETUP condition or BRDY or BEMP interrupt.
ctsq_mask	inout	: Mask to isolate the CTSQ field.

Table 666:Return values

Name	Description
UX_SUCCESS	Function is invoked under ISR successfully.

7.17.52 ux_dcd_synergy_transfer_request

```
ux_dcd_synergy_transfer_request (  UX_DCD_SYNERGY * dcd_synergy ,
 UX_SLAVE_TRANSFER * transfer_request )
```

7.17.52.1 Brief description

This function will initiate a transfer to a specific endpoint. If the endpoint is IN, the endpoint register will be set to accept the request. If the endpoint is IN, the endpoint FIFO will be filled with the buffer and the endpoint register set.

7.17.52.2 Detailed description

Table 667:Parameters

Name	Direction	Description
dcd_synergy	inout	: Pointer to a DCD control block
transfer_request	inout	: Pointer to transfer request

Table 668:Return values

Name	Description
UX_SUCCESS	Transfer to a specific endpoint is initiated successfully.
ux_slave_transfer_request_completion_code	Pointer to structure UX_SLAVE_TRANSFER(transfer request completion code).
UX_TRANSFER_ERROR	Transfer is completed with error.

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_dcd_synergy_buffer_write

7.17.53 ux_hcd_synergy_asynch_queue_process

```
ux_hcd_synergy_asynch_queue_process (  UX_HCD_SYNERGY * hcd_synergy )
```

7.17.53.1 Brief description

This function process the asynchronous transactions. The function will identify the USB interrupts occurred associated with an endpoint and will process the interrupts.

7.17.53.2 Detailed description

Table 669:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block

7.17.54 ux_hcd_synergy_asynch_queue_process_bemp

```
ux_hcd_synergy_asynch_queue_process_bemp (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_SYNERGY_ED * ed )
```

7.17.54.1 Brief description

This function process the BEMP(Buffer Empty) interrupt that occurred on a specific ED.

7.17.54.2 Detailed description

Table 670:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
ed	inout	: Pointer to an Endpoint control block

7.17.55 ux_hcd_synergy_asynch_queue_process_brdy

```
ux_hcd_synergy_asynch_queue_process_brdy (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_SYNERGY_ED * ed )
```

7.17.55.1 Brief description

This function process the BRDY(Buffer Ready)interrupt that occurred on a specific ED.

7.17.55.2 Detailed description

Table 671:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
ed	inout	: Pointer to an Endpoint control block

7.17.56 ux_hcd_synergy_asynch_queue_process_nrdy

```
ux_hcd_synergy_asynch_queue_process_nrdy (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_SYNERGY_ED * ed )
```

7.17.56.1 Brief description

This function process the NRDY(Not Ready) Interrupt that occurred on a specific ED.

7.17.56.2 Detailed description

Table 672:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
ed	inout	: Pointer to an Endpoint control block

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- UX_TRANSFER::ux_transfer_request_completion_function

7.17.57 ux_hcd_synergy_asynch_queue_process_sign

```
ux_hcd_synergy_asynch_queue_process_sign (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_SYNERGY_ED * ed )
```

7.17.57.1 Brief description

This function process the Setup transaction Error Interrupt.

7.17.57.2 Detailed description

Table 673:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
ed	inout	: Pointer to an Endpoint control block

7.17.58 ux_hcd_synergy_asynch_schedule

```
ux_hcd_synergy_asynch_schedule (  UX_HCD_SYNERGY * hcd_synergy )
```

7.17.58.1 Brief description

This function schedules new transfers from the control or bulk lists.

7.17.58.2 Detailed description

Table 674:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block

7.17.59 ux_hcd_synergy_asynchronous_endpoint_create

```
ux_hcd_synergy_asynchronous_endpoint_create (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_ENDPOINT * endpoint )
```

7.17.59.1 Brief description

This function will create an asynchronous endpoint. The control and bulk endpoints fall into this category.

7.17.59.2 Detailed description

Table 675:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
endpoint	inout	: Pointer to endpoint

Table 676:Return values

Name	Description
UX_NO_ED_AVAILABLE	ED for new endpoint not available.
UX_NO_TD_AVAILABLE	Dummy TD not available for terminating the ED transfer chain.
UX_SUCCESS	Asynchronous endpoint created successfully.

7.17.60 ux_hcd_synergy_asynchronous_endpoint_destroy

```
ux_hcd_synergy_asynchronous_endpoint_destroy (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_ENDPOINT * endpoint )
```

7.17.60.1 Brief description

This function will destroy an asynchronous endpoint. The control and bulk endpoints fall into this category.

7.17.60.2 Detailed description

Table 677:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
endpoint	inout	: Pointer to endpoint

Table 678:Return values

Name	Description
UX_ENDPOINT_HANDLE_UNKNOWN	Physical endpoint has not been initialized properly.
UX_SUCCESS	Asynchronous endpoint destroyed successfully.

7.17.61 ux_hcd_synergy_buffer_empty_interrupt

```
ux_hcd_synergy_buffer_empty_interrupt (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_SYNERGY_ED * ed ,   ULONG flag )
```

7.17.61.1 Brief description

This function enable or disable the BEMP(Buffer Empty) interrupt for the pipe.

7.17.61.2 Detailed description

Table 679:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
ed	in	: Pointer to physical Endpoint(ED) control block
flag	in	: Check whether DCD synergy is enable or disable.

7.17.62 ux_hcd_synergy_buffer_notready_interrupt

```
ux_hcd_synergy_buffer_notready_interrupt (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_SYNERGY_ED * ed ,   ULONG flag )
```

7.17.62.1 Brief description

This function enable or disable the NRDY(Not Ready) interrupt for the pipe.

7.17.62.2 Detailed description

Table 680:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
ed	in	: Pointer to physical Endpoint(ED) control block
flag	in	: Check whether DCD synergy is enable or disable.

7.17.63 ux_hcd_synergy_buffer_read

```
ux_hcd_synergy_buffer_read ( UX_HCD_SYNERGY * hcd_synergy , UX_SYNERGY_ED
* ed )
```

7.17.63.1 Brief description

This function reads from a specified pipe into a buffer.

7.17.63.2 Detailed description

Table 681:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
ed	in	: Pointer to a physical Endpoint(ED) control block

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_hcd_synergy_fifo_read

7.17.64 ux_hcd_synergy_buffer_ready_interrupt

```
ux_hcd_synergy_buffer_ready_interrupt ( UX_HCD_SYNERGY * hcd_synergy ,
 UX_SYNERGY_ED * ed , ULONG flag )
```

7.17.64.1 Brief description

This function enable or disable the BRDY(Ready) interrupt for the pipe.

7.17.64.2 Detailed description

Table 682:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
ed	in	: Pointer to physical Endpoint(ED) control block
flag	in	: Check whether DCD synergy is enable or disable.

7.17.65 ux_hcd_synergy_buffer_write

```
 ux_hcd_synergy_buffer_write (  UX_HCD_SYNERGY * hcd_synergy ,   UX_SYNERGY_ED
* ed )
```

7.17.65.1 Brief description

This function writes data to the selected FIFO of the endpoint.

7.17.65.2 Detailed description

Table 683:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
ed	in	: Pointer to a physical Endpoint(ED) control block

Table 684:Return values

Name	Description
UX_SUCCESS	Buffer written to the specified PIPE successfully.

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_hcd_synergy_fifod_write
- ux_hcd_synergy_fifoc_write

7.17.66 ux_hcd_synergy_bulk_endpoint_create

```
ux_hcd_synergy_bulk_endpoint_create (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_ENDPOINT * endpoint )
```

7.17.66.1 Brief description

This function will create a bulk endpoint.

7.17.66.2 Detailed description

Table 685:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
endpoint	inout	: Pointer to a USBX Endpoint Container structure

Table 686:Return values

Name	Description
UX_ERROR	PIPE is not available for bulk endpoint creation .
UX_SUCCESS	Bulk endpoints created successfully.
UX_NO_ED_AVAILABLE	ED for bulk endpoint is not available.
UX_NO_TD_AVAILABLE	Dummy TD for for terminating the ED transfer chain is not available.

7.17.67 ux_hcd_synergy_bulk_int_td_add

```
ux_hcd_synergy_bulk_int_td_add (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_SYNERGY_ED * ed )
```

7.17.67.1 Brief description

This function adds a transfer descriptor to an Bulk or INT ED.

7.17.67.2 Detailed description

Table 687:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
ed	inout	: Pointer to a Synergy ED structure

Table 688:Return values

Name	Description
UX_SUCCESS	Transfer descriptor added successfully.

7.17.68 ux_hcd_synergy_control_endpoint_create

```
ux_hcd_synergy_control_endpoint_create (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_ENDPOINT * endpoint )
```

7.17.68.1 Brief description

This function will create a control endpoint.

7.17.68.2 Detailed description

Table 689:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
endpoint	inout	: Pointer to an Endpoint control block

Table 690:Return values

Name	Description
UX_SUCCESS	Control endpoint created successfully.
UX_NO_ED_AVAILABLE	Failed to obtain an ED for control endpoint.
UX_NO_TD_AVAILABLE	Failed to obtain a TD for control endpoint.

7.17.69 ux_hcd_synergy_control_td_add

```
ux_hcd_synergy_control_td_add ( UX_HCD_SYNERGY * hcd_synergy , UX_SYNERGY_ED
* ed )
```

7.17.69.1 Brief description

This function adds a transfer descriptor to an ED.

7.17.69.2 Detailed description

Table 691:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
ed	inout	: Pointer to Synergy ED structure

Table 692:Return values

Name	Description
UX_SUCCESS	Transfer descriptor added to an ED successfully.

7.17.70 ux_hcd_synergy_controller_disable

```
ux_hcd_synergy_controller_disable ( UX_HCD_SYNERGY * hcd_synergy )
```

7.17.70.1 Brief description

This function will disable the Synergy controller. The controller will release all its resources (memory, IO ...). After this, the controller will not send SOF any longer. All transactions should have been completed, all classes should have been closed.

7.17.70.2 Detailed description

Table 693:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block

Table 694:Return values

Name	Description
UX_SUCCESS	Synergy controller disabled successfully.

7.17.71 ux_hcd_synergy_current_endpoint_change

```
ux_hcd_synergy_current_endpoint_change (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_SYNERGY_ED * ed ,   ULONG direction )
```

7.17.71.1 Brief description

This function change the endpoint in the FIFO.

7.17.71.2 Detailed description

Table 695:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
ed	in	: Pointer to Synergy ED structure
direction	in	: Direction to transfer

7.17.72 ux_hcd_synergy_data_buffer_size

```
ux_hcd_synergy_data_buffer_size (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_SYNERGY_ED * ed )
```

7.17.72.1 Brief description

This function returns the size of the buffer data.

7.17.72.2 Detailed description

Table 696:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
ed	in	: Pointer to Synergy ED structure

Table 697:Return values

Name	Description
buffer_size	buffer size

7.17.73 ux_hcd_synergy_ed_obtain

```
ux_hcd_synergy_ed_obtain (  UX_HCD_SYNERGY * hcd_synergy )
```

7.17.73.1 Brief description

This function obtains a free ED from the ED list.

7.17.73.2 Detailed description

Table 698:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block

Table 699:Return values

Name	Description
UX_NULL	No available ED in the ED list.

ed Endpoint descriptor pointer address.

7.17.74 ux_hcd_synergy_ed_td_clean

```
ux_hcd_synergy_ed_td_clean (  UX_SYNERGY_ED * ed )
```

7.17.74.1 Brief description

This function process cleans the ED of all tds except the last dummy TD.

7.17.74.2 Detailed description

Table 700:Parameters

Name	Direction	Description
ed	inout	: Pointer to Synergy ED structure

7.17.75 ux_hcd_synergy_endpoint_nak_set

```
ux_hcd_synergy_endpoint_nak_set (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_SYNERGY_ED * ed )
```

7.17.75.1 Brief description

This function sets a NAK(Not Acknowledged) to an endpoint.

7.17.75.2 Detailed description

Table 701:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block

Table 701:Parameters (Continued)

Name	Direction	Description
ed	in	: Pointer to Synergy ED structure

7.17.76 ux_hcd_synergy_endpoint_reset

```
ux_hcd_synergy_endpoint_reset ( UX_HCD_SYNERGY * hcd_synergy , UX_ENDPOINT
* endpoint )
```

7.17.76.1 Brief description

This function will reset an endpoint.

7.17.76.2 Detailed description

Table 702:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
endpoint	inout	: Pointer to an Endpoint control block

Table 703:Return values

Name	Description
UX_SUCCESS	Endpoint reset successfully.

7.17.77 ux_hcd_synergy_entry

```
ux_hcd_synergy_entry ( UX_HCD * hcd , UINT function , VOID * parameter )
```

7.17.77.1 Brief description

This function is the entry function to the USB driver from the USB stack.

7.17.77.2 Detailed description

Table 704:Parameters

Name	Direction	Description
hcd	in	: Pointer to USBX Host Controller structure.
function	in	: Function for driver to perform
parameter	in	: Pointer to function parameter(s)

Table 705:Return values

Name	Description
UX_SUCCESS	HCD function is dispatched successfully.
UX_CONTROLLER_UNKNOWN	Synergy controller is not known.
UX_FUNCTION_NOT_SUPPORTED	Function not supported.

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_hcd_synergy_controller_disable
- ux_hcd_synergy_port_status_get
- ux_hcd_synergy_port_enable
- ux_hcd_synergy_port_disable
- ux_hcd_synergy_power_on_port
- ux_hcd_synergy_power_down_port
- ux_hcd_synergy_port_suspend
- ux_hcd_synergy_port_resume
- ux_hcd_synergy_port_reset
- ux_hcd_synergy_frame_number_get
- ux_hcd_synergy_request_transfer
- ux_hcd_synergy_transfer_abort
- ux_hcd_synergy_control_endpoint_create
- ux_hcd_synergy_bulk_endpoint_create
- ux_hcd_synergy_interrupt_endpoint_create

- ux_hcd_synergy_isochronous_endpoint_create
- ux_hcd_synergy_asynchronous_endpoint_destroy
- ux_hcd_synergy_periodic_endpoint_destroy
- ux_hcd_synergy_endpoint_reset

7.17.78 ux_hcd_synergy_fifo_port_change

```
ux_hcd_synergy_fifo_port_change (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_SYNERGY_ED * ed ,   ULONG direction )
```

7.17.78.1 Brief description

This function change the port of the FIFO.

7.17.78.2 Detailed description

Table 706:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
ed	in	: Pointer to Synergy ED structure
direction	in	: Direction to transfer

Table 707:Return values

Name	Description
synergy_register	Content of FIFO control register.
UX_ERROR	Port not changed successfully or unable to access FIFO.

7.17.79 ux_hcd_synergy_fifo_read

```
 ux_hcd_synergy_fifo_read (  UX_HCD_SYNERGY * hcd_synergy ,   UX_SYNERGY_ED
* ed )
```

7.17.79.1 Brief description

This function read data from the FIFO configured for the PIPE(FIFO C, D0 or D1).

7.17.79.2 Detailed description

Table 708:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
ed	in	: Pointer to Synergy ED structure

Table 709:Return values

Name	Description
UX_ERROR	Unable to access FIFO successfully.
UX_SYNERGY_HC_FIFO_READ_OVER	Status set to read overflow.
UX_SYNERGY_HC_FIFO_READ_SHORT	Short packet to read.
UX_SYNERGY_HC_FIFO_READING	Continue reading buffer.

7.17.80 ux_hcd_synergy_fifo_read_software_copy

```
ux_hcd_synergy_fifo_read_software_copy (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_HCD_SYNERGY_PAYLOAD_TRANSFER * payload ,   UX_HCD_SYNERGY_FIFO * fifo )
```

7.17.80.1 Brief description

USBX HCD CPU FIFO read by software copy. Call a suitable subroutine for selected USB controller hardware.

7.17.80.2 Detailed description

Table 710:Parameters

Name	Direction	Description
hcd_synergy	in	Pointer to the HCD control block
payload	in	Pointer to the Payload block
fifo	in	Pointer to the fifo block

7.17.81 ux_hcd_synergy_fifo_read_software_copy_16bit

```
ux_hcd_synergy_fifo_read_software_copy_16bit (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_HCD_SYNERGY_PAYLOAD_TRANSFER * payload ,   UX_HCD_SYNERGY_FIFO * fifo )
```

7.17.81.1 Brief description

USBX HCD CPU FIFO read - Software copy with 16-bit FIFO access.

7.17.81.2 Detailed description

Table 711:Parameters

Name	Direction	Description
hcd_synergy	in	Pointer to the HCD control block
payload	inout	Pointer to the Payload block
fifo	in	Pointer to the fifo block

7.17.82 ux_hcd_synergy_fifo_read_software_copy_8bit

```
ux_hcd_synergy_fifo_read_software_copy_8bit (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_HCD_SYNERGY_PAYLOAD_TRANSFER * payload ,   UX_HCD_SYNERGY_FIFO * fifo )
```

7.17.82.1 Brief description

USBX HCD CPU FIFO read - Software copy with 8-bit FIFO access.

7.17.82.2 Detailed description

Table 712:Parameters

Name	Direction	Description
hcd_synergy	in	Pointer to the HCD control block
payload	inout	Pointer to the Payload block
fifo	in	Pointer to the fifo block

7.17.83 ux_hcd_synergy_fifo_read_dma_start

```
ux_hcd_synergy_fifo_read_dma_start (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_HCD_SYNERGY_PAYLOAD_TRANSFER * payload ,   UX_HCD_SYNERGY_FIFO * fifo )
```

7.17.83.1 Brief description

USBX HCD DMA read setup function.

7.17.83.2 Detailed description

Table 713:Parameters

Name	Direction	Description
hcd_synergy	inout	Pointer to the HCD control block
payload	inout	Pointer to the Payload block
fifo	in	Pointer to the fifo block

7.17.84 ux_hcd_synergy_fifo_read_dma_set_16bit_or_8bit

```
 ux_hcd_synergy_fifo_read_dma_set_16bit_or_8bit (  UX_HCD_SYNERGY
* hcd_synergy ,   UX_HCD_SYNERGY_PAYLOAD_TRANSFER * payload ,
  UX_HCD_SYNERGY_FIFO * fifo )
```

7.17.84.1 Brief description

USBX HCD DMA FIFO read - DMA transfer with 16 or 8 bit FIFO access.

7.17.84.2 Detailed description

Table 714:Parameters

Name	Direction	Description
hcd_synergy	inout	Pointer to the HCD control block
payload	inout	Pointer to the Payload block
fifo	in	Pointer to the fifo block

7.17.85 ux_hcd_synergy_fifoc_write

```
ux_hcd_synergy_fifoc_write (  UX_HCD_SYNERGY * hcd_synergy ,   UX_SYNERGY_ED
* ed )
```

7.17.85.1 Brief description

This function writes a buffer to FIFOC.

7.17.85.2 Detailed description

Table 715:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
ed	inout	: Pointer to Synergy ED structure

Table 716:Return values

Name	Description
UX_ERROR	Unable to access FIFO successfully.
UX_SYNERGY_HC_FIFO_WRITE_END	Writing at ends.
UX_SYNERGY_HC_FIFO_WRITE_SHORT	Writing short data.
UX_SYNERGY_HC_FIFO_WRITING	Doing multiple writes.

7.17.86 ux_hcd_synergy_fifo_write_software_copy

```
ux_hcd_synergy_fifo_write_software_copy (  UX_HCD_SYNERGY * hcd_synergy ,
 ULONG payload_length ,   UCHAR * payload_buffer ,   VOID * fifo_addr ,
 ULONG fifo_sel )
```

7.17.86.1 Brief description

USBX HCD CPU FIFO write by software copy. Call a suitable subroutine for selected USB controller hardware.

7.17.86.2 Detailed description

Table 717:Parameters

Name	Direction	Description
hcd_synergy	in	Pointer to the HCD control block
payload_length	in	Payload length
payload_buffer	in	Payload buffer address
fifo_addr	in	FIFO register address
fifo_sel	in	FIFO select register

7.17.87 ux_hcd_synergy_fifo_write_software_copy_16bit

```
ux_hcd_synergy_fifo_write_software_copy_16bit (  UX_HCD_SYNERGY * hcd_synergy ,
 ULONG payload_length ,   UCHAR * payload_buffer ,   VOID * fifo_addr ,
 ULONG fifo_sel )
```

7.17.87.1 Brief description

USBX HCD CPU FIFO write - Software copy with 16-bit FIFO access.

7.17.87.2 Detailed description

Table 718:Parameters

Name	Direction	Description
hcd_synergy	in	Pointer to the HCD control block
payload_length	inout	Payload length
payload_buffer	inout	Payload buffer address
fifo_addr	inout	FIFO register address
fifo_sel	in	FIFO select register

7.17.88 ux_hcd_synergy_fifo_write_software_copy_8bit

```
ux_hcd_synergy_fifo_write_software_copy_8bit (  UX_HCD_SYNERGY * hcd_synergy ,
 ULONG payload_length ,   UCHAR * payload_buffer ,   VOID * fifo_addr ,
 ULONG fifo_sel )
```

7.17.88.1 Brief description

USBX HCD CPU FIFO write - Software copy with 8-bit FIFO access.

7.17.88.2 Detailed description

Table 719:Parameters

Name	Direction	Description
hcd_synergy	in	Pointer to the HCD control block
payload_length	inout	Payload length
payload_buffer	inout	Payload buffer address
fifo_addr	inout	FIFO register address
fifo_sel	in	FIFO select register

7.17.89 ux_hcd_synergy_fifo_write_software_copy_remaining_bytes

```
 ux_hcd_synergy_fifo_write_software_copy_remaining_bytes (  UX_HCD_SYNERGY
* hcd_synergy ,   ULONG payload_length ,   UCHAR * payload_buffer ,   VOID
* fifo_addr )
```

7.17.89.1 Brief description

USBX HCD CPU FIFO write - Copy remaining bytes to FIFO by software if the rest bytes are less than FIFO access width.

7.17.89.2 Detailed description

Table 720:Parameters

Name	Direction	Description
hcd_synergy	in	Pointer to the HCD control block

Table 720:Parameters (Continued)

Name	Direction	Description
payload_length	inout	Payload length
payload_buffer	inout	Payload buffer address
fifo_addr	inout	FIFO register address

7.17.90 ux_hcd_synergy_fifod_write

```
ux_hcd_synergy_fifod_write (  UX_HCD_SYNERGY * hcd_synergy ,   UX_SYNERGY_ED
* ed )
```

7.17.90.1 Brief description

This function writes a buffer data to FIFOD0 or FIFOD1.

7.17.90.2 Detailed description

Table 721:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
ed	inout	: Pointer to Synergy ED structure

Table 722:Return values

Name	Description
UX_ERROR	Unable to access FIFO successfully.
UX_SYNERGY_HC_FIFO_WRITE_END	Writing at ends.
UX_SYNERGY_HC_FIFO_WRITE_SHORT	Writing short data.
UX_SYNERGY_HC_FIFO_WRITING	Doing multiple writes.

7.17.91 ux_hcd_synergy_fifod_write_dma_start

```
ux_hcd_synergy_fifod_write_dma_start (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_HCD_SYNERGY_PAYLOAD_TRANSFER * payload ,   UX_HCD_SYNERGY_FIFO * fifo )
```

7.17.91.1 Brief description

USBX HCD DMA write setup function. Call a suitable subroutine for selected USB controller hardware.

7.17.91.2 Detailed description

Table 723:Parameters

Name	Direction	Description
hcd_synergy	inout	Pointer to the HCD control block
payload	inout	Pointer to the Payload block
fifo	in	Pointer to the fifo block

7.17.92 ux_hcd_synergy_fifod_write_dma_set_16bit_or_8bit

```
 ux_hcd_synergy_fifod_write_dma_set_16bit_or_8bit (  UX_HCD_SYNERGY
* hcd_synergy ,   UX_HCD_SYNERGY_PAYLOAD_TRANSFER * payload ,
  UX_HCD_SYNERGY_FIFO * fifo )
```

7.17.92.1 Brief description

USBX HCD DMA FIFO write - DMA transfer with 16 or 8 bit FIFO access.

7.17.92.2 Detailed description

Table 724:Parameters

Name	Direction	Description
hcd_synergy	inout	Pointer to the HCD control block
payload	inout	Pointer to the Payload block
fifo	inout	Pointer to the fifo block

7.17.93 ux_hcd_synergy_frame_number_get

```
 ux_hcd_synergy_frame_number_get (  UX_HCD_SYNERGY * hcd_synergy ,   ULONG
* frame_number )
```

7.17.93.1 Brief description

This function will return the frame number currently used by the controller. This function is mostly used for isochronous purposes and for timing.

7.17.93.2 Detailed description

Table 725:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
frame_number	inout	: Frame number to set

Table 726:Return values

Name	Description
UX_SUCCESS	Frame number returned successfully.

7.17.94 ux_hcd_synergy_frame_number_set

```
ux_hcd_synergy_frame_number_set (  UX_HCD_SYNERGY * hcd_synergy ,
 ULONG frame_number )
```

7.17.94.1 Brief description

This function will set the current frame number to the one specified. This function is mostly used for isochronous purpos.es.

7.17.94.2 Detailed description

Table 727:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
frame_number	inout	: Frame number to set

7.17.95 ux_hcd_synergy_initialize

```
ux_hcd_synergy_initialize ( UX_HCD * hcd )
```

7.17.95.1 Brief description

This function initializes the Synergy controller.

7.17.95.2 Detailed description

Table 728:Parameters

Name	Direction	Description
hcd	inout	: Pointer to USBX host controller structure.

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_hcd_synergy_initialize_common()

7.17.96 ux_hcd_synergy_initialize_common

```
ux_hcd_synergy_initialize_common ( UX_HCD * hcd )
```

7.17.96.1 Brief description

This function Initialize USB peripheral.

7.17.96.2 Detailed description

Table 729:Parameters

Name	Direction	Description
hcd	inout	: Pointer to USBX host controller structure.

Table 730:Return values

Name	Description
UX_SUCCESS	Initialize hcd transfer support successfully.
UX_CONTROLLER_INIT_FAILED	Failed in Transfer module setup, or Unsupported USB controller was specified.
UX_MEMORY_INSUFFICIENT	Failed in allocation memory.

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_hcd_synergy_periodic_tree_create

7.17.97 ux_hcd_synergy_initialize_transfer_support

```
 ux_hcd_synergy_initialize_transfer_support (  UX_HCD * hcd ,   const
UX_HCD_SYNERGY_TRANSFER * p_transfer_instance )
```

7.17.97.1 Brief description

USBX HCD Transfer Support with DMA support.

7.17.97.2 Detailed description

Table 731:Parameters

Name	Direction	Description
hcd	inout	Pointer to the USBX HCD control block.
p_transfer_instance	in	Pointer to Transfer module instances.

Table 732:Return values

Name	Description
UX_SUCCESS	Initialize hcd transfer support successfully.
UX_CONTROLLER_INIT_FAILED	Failed in Transfer module setup, or Unsupported USB controller was specified.
UX_SEMAPHORE_ERROR	Failed in creating a semaphore used for DMA transfer.
UX_MEMORY_INSUFFICIENT	Failed in allocation memory.

7.17.98 ux_hcd_synergy_initialize_common_complete

```
ux_hcd_synergy_initialize_common_complete (  UX_HCD * hcd )
```

7.17.98.1 Brief description

USBX HCD final procedure for the initialization.

7.17.98.2 Detailed description

Table 733:Parameters

Name	Direction	Description
hcd	in	Pointer to the USBX HCD control block.

7.17.99 ux_hcd_dma_complete_tx

```
ux_hcd_dma_complete_tx ( transfer_callback_args_t  * p_args )
```

7.17.99.1 Brief description

USBX HCD DMA write callback function.

7.17.99.2 Detailed description

Table 734:Parameters

Name	Direction	Description
p_args	in	Pointer to the argument of a Transfer module callback function.

7.17.100 ux_hcd_dma_complete_rx

```
ux_hcd_dma_complete_rx ( transfer_callback_args_t  * p_args )
```

7.17.100.1 Brief description

USBX HCD DMA read callback function.

7.17.100.2 Detailed description

Table 735:Parameters

Name	Direction	Description
p_args	in	Pointer to the argument of a Transfer module callback function.

7.17.101 ux_hcd_synergy_interrupt_endpoint_create

```
ux_hcd_synergy_interrupt_endpoint_create (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_ENDPOINT * endpoint )
```

7.17.101.1 Brief description

This function will create an interrupt endpoint. The interrupt endpoint has an interval of operation from 1 to 255. The Synergy has no hardware scheduler but we still build an interrupt tree similar to the OHCI controller.

7.17.101.2 Detailed description

This routine will match the best interval for the Synergy hardware. It will also determine the best node to hook the endpoint based on the load that already exists on the horizontal ED chain.

The tricky part is to understand how the interrupt matrix is constructed. We have used eds with the skip bit on to build a frame of anchor eds. Each ED creates a node for an appropriate combination of interval frequency in the list.

After obtaining a pointer to the list with the lowest traffic, we traverse the list from the highest interval until we reach the interval required. At that node, we anchor our real ED to the node and link the ED that was attached to the node to our ED.

Table 736:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
endpoint	inout	: Pointer to an Endpoint control block

Table 737:Return values

Name	Description
UX_SUCCESS	Interrupt endpoint created successfully.
UX_ERROR	Available pipe is not found for interrupt endpoint.
UX_NO_ED_AVAILABLE	Failed to obtain an ED for new endpoint.
UX_NO_TD_AVAILABLE	Failed to obtain a TD for terminating the ED transfer chain.

7.17.102 ux_hcd_synergy_interrupt_handler

```
ux_hcd_synergy_interrupt_handler (  UINT hcd_index )
```

7.17.102.1 Brief description

This function is the interrupt handler for the Synergy USB HS controller. Normally an interrupt occurs from the controller when there is either a EOF signal and there has been transfers within the frame or when there is a change on one of the downstream ports.

7.17.102.2 Detailed description

All we need to do in the ISR is scan the controllers to find out which one has issued a IRQ. If there is work to do for this controller we need to wake up the corresponding thread to take care of the job.

Table 738:Parameters

Name	Direction	Description
hcd_index	in	: HCD number

7.17.103 ux_hcd_synergy_iso_queue_process

```
ux_hcd_synergy_iso_queue_process (  UX_HCD_SYNERGY * hcd_synergy )
```

7.17.103.1 Brief description

This function process the isochronous transactions that happened in the last frame.

7.17.103.2 Detailed description

Table 739:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block

7.17.104 ux_hcd_synergy_iso_queue_process_bemp

```
ux_hcd_synergy_iso_queue_process_bemp (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_SYNERGY_ED * ed )
```

7.17.104.1 Brief description

This function process the BEMP(Buffer Empty) Interrupt that occurred on a specific ED used for Isochronous transfer.

7.17.104.2 Detailed description

Table 740:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
ed	inout	: Pointer to an Endpoint control block

7.17.105 ux_hcd_synergy_iso_queue_bemp_transfer_end_check

```
 ux_hcd_synergy_iso_queue_bemp_transfer_end_check (  UX_TRANSFER
* transfer_request ,   ULONG actual_length ,   ULONG max_packet_size )
```

7.17.105.1 Brief description

This is the subroutine of ux_hcd_synergy_iso_queue_process_bemp to check a transfer request is completed or not. The function identify the completion by checking following conditions.

7.17.105.2 Detailed description

- PID was set to STALL.
- Actual transferred data length matches to the requested length.
- ZLP was sent.
- Short packet was sent.

Table 741:Parameters

Name	Direction	Description
transfer_request	in	: Pointer to a Transfer Request
actual_length	in	: Data length transmitted
max_packet_size	in	: Max packet size of PIPE used for a data transfer.

7.17.106 ux_hcd_synergy_iso_queue_process_brdy

```
ux_hcd_synergy_iso_queue_process_brdy (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_SYNERGY_ED * ed )
```

7.17.106.1 Brief description

This function process the BRDY(Buffer Ready)interrupt that occurred on a specific ED used for isochronous transfer.

7.17.106.2 Detailed description

Table 742:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
ed	inout	: Pointer to an Endpoint control block

7.17.107 ux_hcd_synergy_iso_queue_brdy_out_transaction

```
ux_hcd_synergy_iso_queue_brdy_out_transaction (  UX_SYNERGY_ED * ed )
```

7.17.107.1 Brief description

This is the subroutine of ux_hcd_synergy_iso_queue_process_brdy to perform BRDY interrupt processing for OUT direction PIPE used for isochrnous transfer.

7.17.107.2 Detailed description

Table 743:Parameters

Name	Direction	Description
ed	inout	: Pointer to an Endpoint control block

7.17.108 ux_hcd_synergy_iso_queue_brdy_in_transaction

```
ux_hcd_synergy_iso_queue_brdy_in_transaction (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_SYNERGY_ED * ed )
```

7.17.108.1 Brief description

This is the subroutine of ux_hcd_synergy_iso_queue_process_brdy to perform BRDY interrupt processing for IN direction PIPE used for isochrnous transfer.

7.17.108.2 Detailed description

Table 744:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
ed	inout	: Pointer to an Endpoint control block

7.17.109 ux_hcd_synergy_iso_queue_process_nrdy

```
ux_hcd_synergy_iso_queue_process_nrdy (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_SYNERGY_ED * ed )
```

7.17.109.1 Brief description

This function process the NRDY(Not Ready) Interrupt that occurred on a specific ED used for Isochronous transfer.

7.17.109.2 Detailed description

Table 745:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
ed	inout	: Pointer to an Endpoint control block

7.17.110 ux_hcd_synergy_iso_schedule

```
ux_hcd_synergy_iso_schedule (  UX_HCD_SYNERGY * hcd_synergy )
```

7.17.110.1 Brief description

This function schedules new transfers from isochronous list.

7.17.110.2 Detailed description

Table 746:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block

7.17.111 ux_hcd_synergy_isochronous_endpoint_create

```
ux_hcd_synergy_isochronous_endpoint_create (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_ENDPOINT * endpoint )
```

7.17.111.1 Brief description

This function creates an isochronous endpoint.

7.17.111.2 Detailed description

Table 747:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
endpoint	inout	: Pointer to an Endpoint control block

Table 748:Return values

Name	Description
UX_SUCCESS	Isochronous endpoint is created successfully.
UX_NO_ED_AVAILABLE	Failed to obtain an ED terminating the ED transfer chain.
UX_NO_TD_AVAILABLE	Failed to obtain a TD for new endpoint.

7.17.112 ux_hcd_synergy_isochronous_td_obtain

```
ux_hcd_synergy_isochronous_td_obtain (  UX_HCD_SYNERGY * hcd_synergy )
```

7.17.112.1 Brief description

This function obtains a free TD from the isochronous TD list.

7.17.112.2 Detailed description

Table 749:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block

Table 750:Return values

Name	Description
td	Pointer to Synergy ISO Trasfer Descriptor.

Table 750:Return values (Continued)

Name	Description
UX_NULL	TD not available in the TD list.

7.17.113 ux_hcd_synergy_least_traffic_list_get

```
ux_hcd_synergy_least_traffic_list_get (  UX_HCD_SYNERGY * hcd_synergy )
```

7.17.113.1 Brief description

This function return a pointer to the first ED in the periodic tree that has the least traffic registered.

7.17.113.2 Detailed description

Table 751:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block

Table 752:Return values

Name	Description
min_bandwidth_ed	End descriptor interrupt Number(ed)

7.17.114 ux_hcd_synergy_periodic_endpoint_destroy

```
ux_hcd_synergy_periodic_endpoint_destroy (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_ENDPOINT * endpoint )
```

7.17.114.1 Brief description

This function will destroy an isochronous endpoint.

7.17.114.2 Detailed description

Table 753:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
endpoint	inout	: Pointer to an Endpoint control block

Table 754:Return values

Name	Description
UX_SUCCESS	Isochronous endpoint is destroyed successfully.
UX_ENDPOINT_HANDLE_UNKNOWN	Physical endpoint has not been initialized properly.

7.17.115 ux_hcd_synergy_periodic_schedule

```
ux_hcd_synergy_periodic_schedule (  UX_HCD_SYNERGY * hcd_synergy )
```

7.17.115.1 Brief description

This function schedules new transfers from the periodic interrupt list.

7.17.115.2 Detailed description

Table 755:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block

7.17.116 ux_hcd_synergy_periodic_tree_create

```
ux_hcd_synergy_periodic_tree_create (  UX_HCD_SYNERGY * hcd_synergy )
```

7.17.116.1 Brief description

This function creates the periodic static tree for the interrupt and isochronous eds.

7.17.116.2 Detailed description

Table 756:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block

Table 757:Return values

Name	Description
UX_SUCCESS	Periodic tree created successfully.
UX_NO_ED_AVAILABLE	Failed to obtain an ED.

7.17.117 ux_hcd_synergy_port_disable

```
ux_hcd_synergy_port_disable (  UX_HCD_SYNERGY * hcd_synergy ,
 ULONG port_index )
```

7.17.117.1 Brief description

This function will disable a specific port attached to the root HUB.

7.17.117.2 Detailed description

Table 758:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
port_index	in	: Port Index

Table 759:Return values

Name	Description
UX_SUCCESS	Port disabled successfully.

Table 759:Return values (Continued)

Name	Description
UX_PORT_INDEX_UNKNOWN	Invalid port .

7.17.118 ux_hcd_synergy_port_enable

```
ux_hcd_synergy_port_enable (  UX_HCD_SYNERGY * hcd_synergy ,
 ULONG port_index )
```

7.17.118.1 Brief description

This function will enable a specific port attached to the root HUB.

7.17.118.2 Detailed description

Table 760:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
port_index	in	: Port Index

Table 761:Return values

Name	Description
UX_SUCCESS	Port enabled successfully.
UX_PORT_INDEX_UNKNOWN	Invalid port.
UX_NO_DEVICE_CONNECTED	Device not connected properly.

7.17.119 ux_hcd_synergy_port_reset

```
ux_hcd_synergy_port_reset (  UX_HCD_SYNERGY * hcd_synergy ,
 ULONG port_index )
```

7.17.119.1 Brief description

This function will reset a specific port attached to the root HUB.

7.17.119.2 Detailed description

Table 762:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
port_index	in	: Port Index

Table 763:Return values

Name	Description
UX_SUCCESS	Port reset successfully.
UX_PORT_INDEX_UNKNOWN	Invalid port.
UX_NO_DEVICE_CONNECTED	Device not connected properly.

7.17.120 ux_hcd_synergy_port_resume

```
ux_hcd_synergy_port_resume (  UX_HCD_SYNERGY * hcd_synergy ,
 UINT port_index )
```

7.17.120.1 Brief description

This function will resume a specific port attached to the root HUB. Present, this function is not supported for resume port.

7.17.120.2 Detailed description

Table 764:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
port_index	in	: Port Index

Table 765:Return values

Name	Description
UX_FUNCTION_NOT_SUPPORTED	Unsupported function.

7.17.121 ux_hcd_synergy_port_status_get

```
ux_hcd_synergy_port_status_get (  UX_HCD_SYNERGY * hcd_synergy ,
 ULONG port_index )
```

7.17.121.1 Brief description

This function will return the status for each port attached to the root HUB.

7.17.121.2 Detailed description

Table 766:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
port_index	in	: Port Index

Table 767:Return values

Name	Description
UX_PORT_INDEX_UNKNOWN	Invalid port.
port_status	Synergy Port status

7.17.122 ux_hcd_synergy_port_suspend

```
ux_hcd_synergy_port_suspend (  UX_HCD_SYNERGY * hcd_synergy ,
 ULONG port_index )
```

7.17.122.1 Brief description

This function will suspend a specific port attached to the root HUB. Present, this function is does not supported.

7.17.122.2 Detailed description

Table 768:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
port_index	in	: Port Index

Table 769:Return values

Name	Description
UX_FUNCTION_NOT_SUPPORTED	Unsupported function.

7.17.123 ux_hcd_synergy_power_down_port

```
ux_hcd_synergy_power_down_port (  UX_HCD_SYNERGY * hcd_synergy ,
 ULONG port_index )
```

7.17.123.1 Brief description

This function will power down a specific port attached to the root HUB. Present, this function is does not supported.

7.17.123.2 Detailed description

Table 770:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
port_index	in	: Port Index

Table 771:Return values

Name	Description
UX_FUNCTION_NOT_SUPPORTED	Unsupported function.

7.17.124 ux_hcd_synergy_power_on_port

```
ux_hcd_synergy_power_on_port (  UX_HCD_SYNERGY * hcd_synergy ,
 ULONG port_index )
```

7.17.124.1 Brief description

This function will power a specific port attached to the root HUB. Present, this function is does not supported.

7.17.124.2 Detailed description

Table 772:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
port_index	in	: Port Index

Table 773:Return values

Name	Description
UX_FUNCTION_NOT_SUPPORTED	Unsupported function.

7.17.125 ux_hcd_synergy_power_root_hubs

```
ux_hcd_synergy_power_root_hubs (  UX_HCD_SYNERGY * hcd_synergy )
```

7.17.125.1 Brief description

This function will power the root HUB. Present, this function is does not supported.

7.17.125.2 Detailed description

Table 774:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block

7.17.126 ux_hcd_synergy_register_clear

```
ux_hcd_synergy_register_clear (  UX_HCD_SYNERGY * hcd_synergy ,
 ULONG synergy_register ,   USHORT value )
```

7.17.126.1 Brief description

This function clears flags in a synergy USB register.

7.17.126.2 Detailed description

Table 775:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
synergy_register	in	: Register to write
value	in	: Value to clear

7.17.127 ux_hcd_synergy_register_read

```
ux_hcd_synergy_register_read (  UX_HCD_SYNERGY * hcd_synergy ,
 ULONG synergy_register )
```

7.17.127.1 Brief description

This function reads a data from synergy USB register.

7.17.127.2 Detailed description

Table 776:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
synergy_register	in	: Register to read

Table 777:Return values

Name	Description
hcd_reg	Value read from the specified register(ULONG value).

7.17.128 ux_hcd_synergy_register_set

```
ux_hcd_synergy_register_set (  UX_HCD_SYNERGY * hcd_synergy ,
 ULONG synergy_register ,   USHORT value )
```

7.17.128.1 Brief description

This function sets flags in a synergy USB register.

7.17.128.2 Detailed description

Table 778:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
synergy_register	in	: Register to read
value	in	: Value to be set

7.17.129 ux_hcd_synergy_register_write

```
ux_hcd_synergy_register_write (  UX_HCD_SYNERGY * hcd_synergy ,
 ULONG synergy_register ,   USHORT value )
```

7.17.129.1 Brief description

This function writes a data to a Synergy USB register.

7.17.129.2 Detailed description

Table 779:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block
synergy_register	in	: Register to write
value	in	: Value to write

7.17.130 ux_hcd_synergy_regular_td_obtain

```
ux_hcd_synergy_regular_td_obtain (  UX_HCD_SYNERGY * hcd_synergy )
```

7.17.130.1 Brief description

This function obtains a free TD from the regular TD list.

7.17.130.2 Detailed description

Table 780:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block

Table 781:Return values

Name	Description
td	A pointer to Synergy TD.
UX_NULL	Null pointer.

7.17.131 ux_hcd_synergy_request_bulk_transfer

```
ux_hcd_synergy_request_bulk_transfer (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_TRANSFER * transfer_request )
```

7.17.131.1 Brief description

This function performs a bulk transfer request. A bulk transfer can be larger than the size of the Synergy buffer so it may be required to chain multiple tds to accommodate this transfer request. A bulk transfer is non blocking, so we return before the transfer request is completed.

7.17.131.2 Detailed description

Table 782:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
transfer_request	inout	: Pointer to transfer request

Table 783:Return values

Name	Description
UX_SUCCESS	Bulk transfer happened successfully.
UX_NO_TD_AVAILABLE	Unavailable New TD.

7.17.132 ux_hcd_synergy_request_control_transfer

```
ux_hcd_synergy_request_control_transfer (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_TRANSFER * transfer_request )
```

7.17.132.1 Brief description

This function performs a control transfer from a transfer request. The USB control transfer is in 3 phases (setup, data, status). This function will chain all phases of the control sequence before setting the Synergy endpoint as a candidate for transfer.

7.17.132.2 Detailed description

Table 784:Parameters

Name	Direction	Description
hcd_synergy	inout	: Pointer to a HCD control block

Table 784:Parameters (Continued)

Name	Direction	Description
transfer_request	inout	: Pointer to transfer request

Table 785:Return values

Name	Description
UX_MEMORY_INSUFFICIENT	Insufficient memory to build the SETUP request.
UX_NO_TD_AVAILABLE	Unavailable New TD.
ux_transfer_request_completion_code	Pointer to USBX transfer request structure(request completion code).

7.17.133 ux_hcd_synergy_request_interrupt_transfer

```
ux_hcd_synergy_request_interrupt_transfer (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_TRANSFER * transfer_request )
```

7.17.133.1 Brief description

This function performs an interrupt transfer request. An interrupt transfer can only be as large as the MaxpacketField in the endpoint descriptor. This was verified at the USB layer and does not need to be reverified here.

7.17.133.2 Detailed description

Table 786:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
transfer_request	inout	: Pointer to transfer request

Table 787:Return values

Name	Description
UX_SUCCESS	Interrupt transfer request processed successfully.

Table 787:Return values (Continued)

Name	Description
UX_NO_TD_AVAILABLE	Unavailable new TD.

7.17.134 ux_hcd_synergy_request_isochronous_transfer

```
ux_hcd_synergy_request_isochronous_transfer (  UX_HCD_SYNERGY * hcd_synergy ,
 UX_TRANSFER * transfer_request )
```

7.17.134.1 Brief description

This function performs an isochronous transfer request.

7.17.134.2 Detailed description

Table 788:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
transfer_request	inout	: Pointer to transfer request

Table 789:Return values

Name	Description
UX_SUCCESS	Isochronous transfer request processed successfully.
UX_NO_TD_AVAILABLE	Unavailable new TD.

7.17.135 ux_hcd_synergy_request_transfer

```
 ux_hcd_synergy_request_transfer (  UX_HCD_SYNERGY * hcd_synergy ,   UX_TRANSFER
* transfer_request )
```

7.17.135.1 Brief description

This function is the handler for all the transactions on the USB. The transfer request passed as parameter contains the endpoint and the device descriptors in addition to the type of transaction de be executed. This function routes the transfer request to according to the type of transfer to be executed.

7.17.135.2 Detailed description

Table 790:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
transfer_request	inout	: Pointer to transfer request

Table 791:Return values

Name	Description
UX_ERROR	Error while Isolating the endpoint type and routing the transfer request.
UX_NO_DEVICE_CONNECTED	No device attached.

See Common Error Codes for other possible return codes or causes. This function calls:

- ux_hcd_synergy_request_control_transfer
- ux_hcd_synergy_request_bulk_transfer
- ux_hcd_synergy_request_interrupt_transfer
- ux_hcd_synergy_request_isochronous_transfer

7.17.136 ux_hcd_synergy_td_add

```
ux_hcd_synergy_td_add (  UX_HCD_SYNERGY * hcd_synergy ,   UX_SYNERGY_ED * ed )
```

7.17.136.1 Brief description

This function add new TD for control, Bulk or Interrupt endpoint.

7.17.136.2 Detailed description

Table 792:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block

Table 792:Parameters (Continued)

Name	Direction	Description
ed	in	: Pointer to Synergy ED structure

Table 793:Return values

Name	Description
UX_SUCCESS	Transfer done successfully.

7.17.137 ux_hcd_synergy_transfer_abort

```
ux_hcd_synergy_transfer_abort ( UX_HCD_SYNERGY * hcd_synergy , UX_TRANSFER
* transfer_request )
```

7.17.137.1 Brief description

This function will abort transactions attached to a transfer request.

7.17.137.2 Detailed description

Table 794:Parameters

Name	Direction	Description
hcd_synergy	in	: Pointer to a HCD control block
transfer_request	inout	: Pointer to transfer request

Table 795:Return values

Name	Description
UX_SUCCESS	Transactions attached to transfer request are aborted successfully.
UX_ENDPOINT_HANDLE_UNKNOWN	Endpoint is not initialized properly.

7.17.138 Extensions

7.18 USB Communication Framework

RTOS-integrated USBX CDC ACM device implementation.

7.18.1 Functions

- SF_EL_UX_COMMS_Open
- SF_EL_UX_COMMS_Close
- SF_EL_UX_COMMS_Read
- SF_EL_UX_COMMS_Write
- SF_EL_UX_COMMS_Lock
- SF_EL_UX_COMMS_Unlock
- SF_EL_UX_COMMS_VersionGet
- ux_cdc_instance_activate
- ux_cdc_instance_deactivate
- sf_el_ux_comms_read_leftover

7.18.2 Defines

- #define SF_EL_UX_COMMS_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define SF_EL_UX_COMMS_CODE_VERSION_MINOR

  Initial value: `(6U)`

7.18.3 SF_EL_UX_COMMS_Open

```
ssp_err_t SF_EL_UX_COMMS_Open ( sf_comms_ctrl_t  *const p_api_ctrl ,
 sf_comms_cfg_t  const *const p_cfg )
```

7.18.3.1 Brief description

Initializes a USB channel for CDC ACM mode.

7.18.3.2 Detailed description

Table 796:Return values

Name	Description
SSP_SUCCESS	Channel opened successfully
SSP_ERR_ASSERTION	Pointer to sf_el_ux_comms control block is NULL
SSP_ERR_INTERNAL	USBX initialize call returned an error.

NOTE: This function is reentrant.

Table 797:Parameters

Name	Direction	Description
p_api_ctrl	in	Pointer to control structure block.
p_cfg	in	Pointer to configuration structure block. This parameter is not used in the framework hence the NULL parameter check not implemented.

Table 798:Return values

Name	Description
SSP_SUCCESS	Channel opened successfully
SSP_ERR_ASSERTION	p_api_ctrl pointer parameter to control block is NULL
SSP_ERR_TIMEOUT	Mutex not available in timeout.
SSP_ERR_IN_USE	Channel/peripheral is running/busy.

NOTE: This function is reentrant.

7.18.3.3 Function steps

- Create the mutexes.
- Mark control block open by intializing it to "UXCM" in its ASCII equivalent .
- Suspend here until a USBX CDC instance is created by the USBX CDC for this module.
- Create the mutexes.
- Mark control block open.

7.18.4 SF_EL_UX_COMMS_Close

```
ssp_err_t SF_EL_UX_COMMS_Close ( sf_comms_ctrl_t  *const p_api_ctrl )
```

7.18.4.1 Brief description

Releases all the ThreadX Resources.

7.18.4.2 Detailed description

Finalize CDC ACM operation for a USB channel. and clear SCI device control block members.

Table 799:Return values

Name	Description
SSP_SUCCESS	Channel successfully closed
SSP_ERR_ASSERTION	Pointer to control block is NULL
SSP_ERR_INTERNAL	Deletion of mutex failed

NOTE: This function is reentrant.

Table 800:Return values

Name	Description
SSP_SUCCESS	Channel successfully closed
SSP_ERR_ASSERTION	Pointer to control block is NULL
SSP_ERR_INTERNAL	Deletion of mutex failed

NOTE: This function is reentrant. No actual USB COM port close operation being held.

7.18.4.3 Function steps

- Delete transmit mutex.
- Delete receive mutex.
- Delete transmit mutex.
- Delete receive mutex.

7.18.5 SF_EL_UX_COMMS_Read

```
ssp_err_t SF_EL_UX_COMMS_Read ( sf_comms_ctrl_t  *const p_api_ctrl ,   uint8_t
*const p_dest ,   uint32_t const bytes ,   UINT const timeout )
```

7.18.5.1 Brief description

Read data from the USBX CDC ACM driver.

7.18.5.2 Detailed description

Read data from the USBX CDC-ACM driver.

Table 801:Return values

Name	Description
SSP_SUCCESS	Data reception ends successfully.
SSP_ERR_TIMEOUT	Wait for the USBX instance to be created until the timeout occurs.
SSP_ERR_INTERNAL	USB device not enumerated or USBX read call returned an error.

NOTE: This API is reentrant.

Table 802:Return values

Name	Description
SSP_SUCCESS	Data reception ends successfully.
SSP_ERR_INTERNAL	USB device not enumerated or USBX read call returned an error.
SSP_ERR_ASSERTION	Either of pointer type argument is NULL.
SSP_ERR_NOT_OPEN	Module is not opened.

NOTE: This API is reentrant.

7.18.5.3 Function steps

- Read data leftover from the last packet.
- Read from the CDC class.
- Buffer overflow occurred.
- Check if module has been opened.
- Check for the USB to be plugged in.
- If there is data leftover from the last packet, use it.
- Read from the CDC class.
- Buffer overflow occurred.

7.18.6 SF_EL_UX_COMMS_Write

```
ssp_err_t SF_EL_UX_COMMS_Write ( sf_comms_ctrl_t  *const p_api_ctrl ,   uint8_t
const *const p_src ,   uint32_t const bytes ,   UINT const timeout )
```

7.18.6.1 Brief description

Write data to the USBX CDC ACM driver.

7.18.6.2 Detailed description

Write data to the USBX CDC-ACM framework.

Table 803:Return values

Name	Description
SSP_SUCCESS	Data transmission finished successfully.
SSP_ERR_TIMEOUT	Wait for the USBX instance to be created until the timeout occurs.
SSP_ERR_ASSERTION	Pointer to sf_el_ux_comms control block is NULL.
SSP_ERR_INTERNAL	USB device not enumerated or USBX write call returned an error.

NOTE: This function is reentrant.

Table 804:Return values

Name	Description
SSP_SUCCESS	Data transmission finished successfully.
SSP_ERR_ASSERTION	Pointer to control block is NULL
SSP_ERR_NOT_OPEN	Module is not opened.
SSP_ERR_INTERNAL	USB device not enumerated or USBX write call returned an error.

NOTE: This function is reentrant.

7.18.6.3 Function steps

- Wait for the USB to be plugged in.
- Check if module has been opened.
- Check for the USB to be plugged in.

7.18.7 SF_EL_UX_COMMS_Lock

```
ssp_err_t SF_EL_UX_COMMS_Lock ( sf_comms_ctrl_t  *const p_api_ctrl ,
 sf_comms_lock_t  lock_type ,   UINT timeout )
```

7.18.7.1 Brief description

Locks the communication to a thread by acquiring a mutex before the timeout occurs.

7.18.7.2 Detailed description

Lock the USB COM resource.

Table 805:Return values

Name	Description
SSP_SUCCESS	Locking sf_el_ux_comms resource successful.
SSP_ERR_ASSERTION	Pointer to sf_el_ux_comms control block is NULL.
SSP_ERR_TIMEOUT	Mutex not available in timeout.
SSP_ERR_NOT_OPEN	The framework is not opened. Open the framework first before calling this API.
SSP_SUCCESS	Locking a USB COM resource successful.
SSP_ERR_ASSERTION	Pointer to control block is NULL.
SSP_ERR_NOT_OPEN	Module is not opened.
SSP_ERR_TIMEOUT	Mutex not available in timeout.

7.18.8 SF_EL_UX_COMMS_Unlock

```
ssp_err_t SF_EL_UX_COMMS_Unlock ( sf_comms_ctrl_t  *const p_api_ctrl ,
 sf_comms_lock_t  lock_type )
```

7.18.8.1 Brief description

Unlocks the communication to a thread by releasing the mutex for write or read or both.

7.18.8.2 Detailed description

Unlock the USB COM resource.

Table 806:Return values

Name	Description
SSP_SUCCESS	Unlocks the sf_el_ux_comms communication resource successful.
SSP_ERR_ASSERTION	Pointer to sf_el_ux_comms control block is NULL.
SSP_ERR_NOT_OPEN	The framework is not opened. Open the framework first before calling this API.
SSP_SUCCESS	Unlocking a USB COM resource successful.
SSP_ERR_ASSERTION	Pointer to control block is NULL.
SSP_ERR_NOT_OPEN	Module is not opened.
SSP_ERR_TIMEOUT	An internal ThreadX error has occurred. This is typically a failure to use a mutex.

7.18.9 SF_EL_UX_COMMS_VersionGet

```
ssp_err_t SF_EL_UX_COMMS_VersionGet ( ssp_version_t  *const p_version )
```

7.18.9.1 Brief description

Get driver version.

7.18.9.2 Detailed description

Get driver version

Table 807:Parameters

Name	Direction	Description
p_version	out	Version will be stored here.

Table 808:Return values

Name	Description
SSP_SUCCESS	Version stored in provided pointer.
SSP_ERR_ASSERTION	p_version was null.

NOTE: This function is reentrant.

Table 809:Parameters

Name	Direction	Description
p_version	out	Version will be stored here.

NOTE: This function is reentrant.

7.18.10 ux_cdc_instance_activate

```
ux_cdc_instance_activate (  VOID * cdc_instance )
```

7.18.11 ux_cdc_instance_deactivate

```
ux_cdc_instance_deactivate (  VOID * cdc_instance )
```

7.18.12 sf_el_ux_comms_read_leftover

```
ssp_err_t sf_el_ux_comms_read_leftover ( sf_el_ux_comms_instance_ctrl_t
* p_ux_comms_ctrl ,   uint8_t *const p_destination ,   uint32_t
* p_remaining_bytes ,   UINT time )
```

7.18.12.1 Brief description

Subroutine for SF_EL_UX_COMMS_Read API which will be called to read the leftover data from last packet.

7.18.12.2 Detailed description

Table 810:Parameters

Name	Direction	Description
p_ux_comms_ctrl	in	Pointer to the sf_el_ux_comms control block.
p_destination	out	Pointer to the destination buffer.
p_remaining_bytes	inout	Pointer to remaining bytes.
time	in	User specified timeout value.

Table 811:Return values

Name	Description
SSP_ERR_TIMEOUT	Wait for the USBX instance to be created until the timeout occurs.
SSP_SUCCESS	Read the leftover data from previous packet successful.

7.18.12.3 Function steps

- Wait for the USB to be plugged in.
- If there is data leftover from the last packet, use it.

7.18.13 Extensions

7.18.13.1 sf_el_ux_comms_instance_ctrl_t

sf_el_ux_comms_instance_ctrl_t

Detailed description

USBX CDC ACM device communications instance control structure. DO NOT INITIALIZE. Initialization occurs when open is called

Variables

- uint32_t open
- TX_MUTEX mutex[2]

- UX_SLAVE_CLASS_CDC_ACM * p_cdc
- uint32_t leftover_length
- uint32_t index
- uint8_t memory[SF_EL_UX_COMMS_CFG_USB_MEMORY_SIZE_BYTES]
- uint8_t rx_memory[SF_EL_UX_COMMS_CFG_BUFFER_MAX_LENGTH]
- TX_SEMAPHORE semaphore

7.18.14 Modules

- Build Time Configurations

7.18.14.1 Build Time Configurations

Defines

- #define SF_EL_UX_COMMS_CFG_PARAM_CHECKING_ENABLE

  Initial value: `(BSP_CFG_PARAM_CHECKING_ENABLE)`

  Specify whether to include code for API parameter checking. Valid settings include:BSP_CFG_PARAM_-CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h1 : Includes parameter checking0 : Compiles out parameter checking
- #define SF_EL_UX_COMMS_CFG_USB_MEMORY_SIZE_BYTES

  Initial value: `(65536)`

  Specify the size of memory to allocate for USBX. Size of 65536 is recommended for device applications.
- #define SF_EL_UX_COMMS_CFG_BUFFER_MAX_LENGTH

  Initial value: `(128)`

  Specify the size of memory to allocate for a read input buffer.

7.19 USB Communication Framework V2

RTOS-integrated USBX CDC ACM device implementation.

7.19.1 Defines

- #define SF_EL_UX_COMMS_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define SF_EL_UX_COMMS_CODE_VERSION_MINOR

  Initial value: `(5U)`

7.19.2 Extensions

7.19.2.1 sf_el_ux_comms_instance_ctrl_t

sf_el_ux_comms_instance_ctrl_t

Detailed description

USBX CDC ACM device communications instance control structure. DO NOT INITIALIZE. Initialization occurs when open is called

Variables

- uint32_t open
- TX_MUTEX mutex[2]
- UX_SLAVE_CLASS_CDC_ACM * p_cdc
- uint32_t leftover_length
- uint32_t index
- uint8_t memory[SF_EL_UX_COMMS_CFG_USB_MEMORY_SIZE_BYTES]
- uint8_t rx_memory[SF_EL_UX_COMMS_CFG_BUFFER_MAX_LENGTH]
- TX_SEMAPHORE semaphore

7.20 External IRQ Framework

RTOS-integrated external IRQ Framework.

7.20.1 Summary

This module is a ThreadX-aware external IRQ Framework for external inputs such as switches or other binary signals.

7.20.2 Functions

- SF_EXTERNAL_IRQ_Open
- SF_EXTERNAL_IRQ_Wait
- SF_EXTERNAL_IRQ_VersionGet
- SF_EXTERNAL_IRQ_Close

7.20.3 Defines

- #define SF_EXTERNAL_IRQ_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define SF_EXTERNAL_IRQ_CODE_VERSION_MINOR

  Initial value: `(5U)`

7.20.4 SF_EXTERNAL_IRQ_Open

```
ssp_err_t SF_EXTERNAL_IRQ_Open ( sf_external_irq_ctrl_t  *const p_api_ctrl ,
 sf_external_irq_cfg_t  const *const p_cfg )
```

7.20.4.1 Brief description

Configure external IRQ and optionally enable external IRQ callbacks. Implements open.

7.20.4.2 Detailed description

The SF_EXTERNAL_IRQ_Open function acquires a mutex for the external IRQ channel used, then calls the HAL driver open function. After successful initialization, the external IRQ is ready for use.

Table 812:Return values

Name	Description
SSP_SUCCESS	Initialization was successful and external interrupt has started.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_ctrl, p_api, or p_cfg, p_api, or p_api->open. See HAL driver for other possible causes.
SSP_ERR_IN_USE	This channel is already open.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls:

- open

NOTE: This function is reentrant for any channel.

7.20.4.3 Function steps

- Save driver structure for use in other framework layer functions
- Create semaphore for use with wait function
- Get mutex before calling lower layer, which will access HW registers.
- Prepare configuration for lower layer
- Open lower layer
- Post the mutex
- If low level initialization failed, delete the mutex and exit the function with the error code
- Delete RTOS services used and log the error
- log the error and return the error
- If there was any mutex error, return it
- Mark control block open so other tasks know it is valid

7.20.5 SF_EXTERNAL_IRQ_Wait

```
ssp_err_t SF_EXTERNAL_IRQ_Wait ( sf_external_irq_ctrl_t  *const p_api_ctrl ,
  ULONG const timeout )
```

7.20.5.1 Brief description

Get mutex, wait for external interrupt to expire, and release mutex. Implements wait.

7.20.5.2 Detailed description

Table 813:Return values

Name	Description
SSP_SUCCESS	External interrupt stopped successfully.
SSP_ERR_NOT_OPEN	Driver control block not valid. Call SF_EXTERNAL_IRQ_Open to configure.
SSP_ERR_TIMEOUT	Time out happens while waiting a semaphore.
SSP_ERR_WAIT_ABORTED	Suspension was aborted by another thread.

See Common Error Codes or HAL driver for other possible return codes or causes.

7.20.5.3 Function steps

- Wait for semaphore post from ISR

7.20.6 SF_EXTERNAL_IRQ_VersionGet

```
ssp_err_t SF_EXTERNAL_IRQ_VersionGet ( ssp_version_t  *const p_version )
```

7.20.6.1 Brief description

Get version and store it in provided pointer p_version. Implements versionGet.

7.20.6.2 Detailed description

Table 814:Return values

Name	Description
SSP_SUCCESS	Version returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was null.

7.20.7 SF_EXTERNAL_IRQ_Close

```
ssp_err_t SF_EXTERNAL_IRQ_Close ( sf_external_irq_ctrl_t  *const p_api_ctrl )
```

7.20.7.1 Brief description

Release channel mutex and close channel at HAL layer. Implements close.

7.20.7.2 Detailed description

Table 815:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter ctrl is NULL.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_IN_USE	This channel is already open.

Table 815:Return values (Continued)

Name	Description
SSP_ERR_UNSUPPORTED	Unsupported operation.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls:

- close

7.20.7.3 Function steps

- Get mutex since this will access hardware registers
- Close channel in HAL layer
- Post the mutex
- Clear information from control block so other functions know this block is closed
- Clear information from control block so other functions know this instance is closed
- Delete RTOS services used

7.20.8 Extensions

7.20.8.1 sf_external_irq_instance_ctrl_t

sf_external_irq_instance_ctrl_t

Detailed description

Instance control block. DO NOT INITIALIZE. Initialization occurs when open is called

Variables

- uint32_t open

  Used by driver to check if control block is valid.
- TX_MUTEX mutex

  Mutex used to protect access to lower level driver hardware registers.
- TX_SEMAPHORE semaphore

  Semaphore used for SF_EXTERNAL_IRQ_Wait.
- external_irq_instance_t const * p_lower_lvl_irq

  Pointer to lower level driver instance.
- bool callback_used

  Used by driver to check if wait can be used.

7.21 I2C Framework

RTOS-integrated I2C Framework.

SSP I2C framework driver API .

7.21.1 Summary

This is a ThreadX-aware I2C driver API. The API implements the I2C Framework interface and can access several hardware peripherals at the HAL layer. The connection to the HAL layer is established by passing in a driver structure in SF_I2C_Open.

7.21.2 Interface Used

I2C Framework

7.21.3 Functions

- sf_i2c_callback
- sf_i2c_common_start
- sf_i2c_common_wait
- sf_i2c_common_finish
- sf_i2c_check_lower_lvl_driver_parameters
- sf_i2c_check_common_parameters
- sf_i2c_reconfigure_device
- SF_I2C_Open
- SF_I2C_Read
- SF_I2C_Write
- SF_I2C_Reset
- SF_I2C_Close
- SF_I2C_Lock
- SF_I2C_Unlock
- SF_I2C_VersionGet

7.21.4 Variables

- g_sf_i2c_version
- g_sf_i2c_on_sf_i2c

7.21.5 Defines

- #define SF_I2C_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define SF_I2C_CODE_VERSION_MINOR

  Initial value: `(5U)`
- #define SF_I2C_ERROR_RETURN

  Initial value:`#define SSP_ERROR_RETURN((expression), (error), &g_module_name[0], &g_sf_i2c_version)`

7.21.6 sf_i2c_callback

```
sf_i2c_callback ( i2c_callback_args_t  * parg )
```

7.21.6.1 Brief description

I2C SSP framework level callback.

7.21.6.2 Detailed description

Table 816:Parameters

Name	Direction	Description
p_args	in	Pointer to callback parameters

Table 817:Return values

Name	Description
void	

7.21.7 sf_i2c_common_start

```
ssp_err_t sf_i2c_common_start ( sf_i2c_instance_ctrl_t  *const p_ctrl ,
  uint32_t const timeout )
```

7.21.7.1 Brief description

Common I2C transfer start function. Used in all framework read write calls. This function checks is there any need for reconfiguration and gets the mutex and reconfigures if required.

7.21.7.2 Detailed description

(end addtogroup SF_I2C)

Table 818:Parameters

Name	Direction	Description
p_ctrl	in	Control handle for I2C framework driver context for a device.
timeout	in	ThreadX timeout.

Table 819:Return values

Name	Description
SSP_SUCCESS	Transfer started successfully.
SSP_ERR_INTERNAL	Internal error occurred.

NOTE: This function is reentrant for any device.

7.21.7.3 Function steps

- Reconfigure the device address, if necessary

7.21.8 sf_i2c_common_wait

```
ssp_err_t sf_i2c_common_wait ( sf_i2c_instance_ctrl_t  *const p_ctrl ,
 i2c_event_t  event ,   uint32_t const timeout )
```

7.21.8.1 Brief description

Common I2C wait. Waits for an operation to finish.

7.21.8.2 Detailed description

Table 820:Parameters

Name	Direction	Description
p_ctrl	in	Control handle for I2C framework driver context for a device
event	in	Event
timeout	in	ThreadX timeout

Table 821:Return values

Name	Description
SSP_SUCCESS	Data transmitted successfully.
SSP_ERR_TRANSFER_ABORTED	Transfer aborted.
SSP_ERR_INTERNAL	Internal error occurred.

NOTE: This function is reentrant for any device.

7.21.9 sf_i2c_common_finish

```
ssp_err_t sf_i2c_common_finish ( sf_i2c_instance_ctrl_t  *const p_ctrl )
```

7.21.9.1 Brief description

Common I2C finish. Release mutex if the bus is not locked and restart not issued..

7.21.9.2 Detailed description

Table 822:Parameters

Name	Direction	Description
p_ctrl	in	Control handle for I2C framework driver context for a device

Table 823:Return values

Name	Description
SSP_SUCCESS	Transfer finished successfully.
SSP_ERR_INTERNAL	Internal error occurred.

NOTE: This function is reentrant for any device.

7.21.10 sf_i2c_check_lower_lvl_driver_parameters

```
sf_i2c_check_lower_lvl_driver_parameters ( sf_i2c_cfg_t  const *const p_cfg )
```

7.21.10.1 Brief description

Checks whether lower level I2C module and bus are configured.

7.21.10.2 Detailed description

Table 824:Parameters

Name	Direction	Description
p_cfg	in	Pointer to I2C framework Configuration Structure

Table 825:Return values

Name	Description
true	Lower level I2C driver, and I2C bus are configured.
false	Lower level I2C driver, and I2C bus are not configured.

7.21.11 sf_i2c_check_common_parameters

```
 sf_i2c_check_common_parameters ( sf_i2c_ctrl_t  *const p_ctrl ,   uint32_t
const bytes )
```

7.21.11.1 Brief description

Check if I2C framework control block address and number of bytes are not NULL.

7.21.11.2 Detailed description

Table 826:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to I2C framework control block
bytes	in	Number of bytes of data to be transferred

Table 827:Return values

Name	Description
true	I2C framework control block address, number of bytes are not NULL.
false	I2C framework control block address, number of bytes are NULL.

7.21.12 sf_i2c_reconfigure_device

```
ssp_err_t sf_i2c_reconfigure_device ( sf_i2c_instance_ctrl_t  *const p_ctrl )
```

7.21.12.1 Brief description

Assign a slave address of new device to current device on the bus.

7.21.12.2 Detailed description

Table 828:Parameters

Name	Direction	Description
p_ctrl	in	Control handle for I2C framework context for a device

Table 829:Return values

Name	Description
SSP_SUCCESS	New device slave address assigned to current device

See Common Error Codes and lower level drivers for other possible return codes. These driver functions are:

- slaveAddressSet

7.21.12.3 Function steps

- Reconfigure the device by changing the slave address.
- Change the slave address and addressing mode.
- Assign this device to current.

7.21.13 SF_I2C_Open

```
ssp_err_t SF_I2C_Open ( sf_i2c_ctrl_t  *const p_api_ctrl ,  sf_i2c_cfg_t  const
*const p_cfg )
```

7.21.13.1 Brief description

Initialize a I2C bus and open low level I2C driver.

7.21.13.2 Detailed description

Table 830:Return values

Name	Description
SSP_SUCCESS	I2C device is successfully opened.
SSP_ERR_ASSERTION	One of the following parameters is NULL: p_api_ctrl, p_cfg, Pointer to Open, Close, Read, Write, or reset API interfaces,p_cfg->p_bus.
SSP_ERR_INTERNAL	Internal error occurred.

See Common Error Codes and lower level driver function for other possible return codes. This driver function is

- open

NOTE: This function is reentrant for any channel.

NOTE: Control handle must be cleared by caller before calling this function.

7.21.13.3 Function steps

- Copy bus to control
- Initialize bus lock to false
- Set framework level callback function.
- Save context for use in ISRs.
- Create mutex for this bus.
- Create Event flag.
- Open low level.
- Save device configuration for reconfiguration.
- Set device state as Opened.
- Initialize restarted flag to false.
- Increment device count.

7.21.14 SF_I2C_Read

```
ssp_err_t SF_I2C_Read ( sf_i2c_ctrl_t  *const p_api_ctrl ,   uint8_t
*const p_dest ,   uint32_t const bytes ,   bool const restart ,   uint32_t
const timeout )
```

7.21.14.1 Brief description

Start the transfer process and receive data from I2C device.

7.21.14.2 Detailed description

Table 831:Return values

Name	Description
SSP_SUCCESS	Data received successfully.
SSP_ERR_NOT_OPEN	Device instance not opened.
SSP_ERR_ASSERTION	One of the following parameters is NULL: p_api_ctrl, p_dest, bytes, timeout.

See Common Error Codes and lower level driver function for other possible return codes. This driver function is:

- read

7.21.14.3 Function steps

- Check whether device is opened or not.
- Start transfer process - check lock, check reconfiguration, get Mutex.
- Get the low level control in use.
- Perform read.
- Wait for callback to set event flag.
- Finish transfer.
- Finish transfer.

7.21.15 SF_I2C_Write

```
ssp_err_t SF_I2C_Write ( sf_i2c_ctrl_t  *const p_api_ctrl ,   uint8_t
*const p_src ,   uint32_t const bytes ,   bool const restart ,   uint32_t
const timeout )
```

7.21.15.1 Brief description

Start the transfer process and send data on I2C bus.

7.21.15.2 Detailed description

Table 832:Return values

Name	Description
SSP_SUCCESS	Data written successfully.
SSP_ERR_NOT_OPEN	Device instance not opened.
SSP_ERR_ASSERTION	One of the following parameters is NULL: p_api_ctrl, p_src, bytes.

See Common Error Codes and lower level driver function for other possible return codes. This driver function is:

- write

7.21.15.3 Function steps

- Check whether device is opened or not.
- Start transfer process - check lock, check reconfiguration, get Mutex.
- Get the low level control in use.
- Wait for callback to set event flag.

7.21.16 SF_I2C_Reset

```
ssp_err_t SF_I2C_Reset ( sf_i2c_ctrl_t  *const p_api_ctrl ,   uint32_t
const timeout )
```

7.21.16.1 Brief description

Abort any in-progress transfer.

7.21.16.2 Detailed description

Table 833:Return values

Name	Description
SSP_SUCCESS	Channel was reseted without issue.
SSP_ERR_NOT_OPEN	Device was not even opened.
SSP_ERR_IN_USE	In-use error.
SSP_ERR_INTERNAL	Internal error occurred.
SSP_ERR_ASSERTION	Following parameters is NULL: p_api_ctrl.

See Common Error Codes and lower level driver function for other possible return codes. This driver function is:

- reset

7.21.16.3 Function steps

- Check whether device is opened or not.
- Get the low level control in use.
- Get mutex since this will access hardware registers.

7.21.17 SF_I2C_Close

```
ssp_err_t SF_I2C_Close ( sf_i2c_ctrl_t  *const p_api_ctrl )
```

7.21.17.1 Brief description

Close the I2C device designated by the control handle and close the RTOS services used by the bus if no device is connected to the bus.

7.21.17.2 Detailed description

Table 834:Return values

Name	Description
SSP_SUCCESS	Device is successfully closed.
SSP_ERR_IN_USE	In-use error.

Table 834:Return values (Continued)

Name	Description
SSP_ERR_NOT_OPEN	Device was not even opened.*
SSP_ERR_INTERNAL	Internal error occurred.
SSP_ERR_ASSERTION	Following parameters is NULL: p_api_ctrl.

See Common Error Codes and lower level driver function for other possible return codes. This driver function is:

- close

NOTE: This function is reentrant for any device.

7.21.17.3 Function steps

- Check whether device is opened or not.
- See if the device need to close through close. close only if this is the current device, else it might have closed during reconfiguration, or not opened through the open().
- Get mutex since this will access hardware registers.
- Close low level driver.
- Release mutex.
- Decrement device count.
- Check whether any device is still using the bus.
- Delete RTOS services used by bus.
- Set device to closed state.

7.21.18 SF_I2C_Lock

```
ssp_err_t SF_I2C_Lock ( sf_i2c_ctrl_t  *const p_api_ctrl )
```

7.21.18.1 Brief description

Lock the bus for a device. Once bus is locked by a device it can not be used by other devices.

7.21.18.2 Detailed description

Table 835:Return values

Name	Description
SSP_SUCCESS	I2C channel is successfully locked.
SSP_ERR_NOT_OPEN	Device not opened.
SSP_ERR_IN_USE	In-use error.
SSP_ERR_ASSERTION	Following parameters is NULL: p_api_ctrl.

NOTE: This function is reentrant for any device.

7.21.18.3 Function steps

- Check whether device is opened or not.
- Check whether lock is already taken.
- Get the mutex for this device.
- Reconfigure the device address, if necessary
- Set lock flag.

7.21.19 SF_I2C_Unlock

```
ssp_err_t SF_I2C_Unlock ( sf_i2c_ctrl_t  *const p_api_ctrl )
```

7.21.19.1 Brief description

Unlock the locked bus and make the bus usable for other devices.

7.21.19.2 Detailed description

Table 836:Return values

Name	Description
SSP_SUCCESS	I2C bus is successfully unlocked.

Table 836:Return values (Continued)

Name	Description
SSP_ERR_NOT_OPEN	Device not opened.
SSP_ERR_IN_USE	In-use error.
SSP_ERR_ASSERTION	Following parameters is NULL: p_api_ctrl.

NOTE: This function is reentrant for any device.

7.21.19.3 Function steps

- Check whether device is opened or not.
- Clear lock flag.
- Release the mutex so that others can use the bus.

7.21.20 SF_I2C_VersionGet

```
ssp_err_t SF_I2C_VersionGet ( ssp_version_t  *const p_version )
```

7.21.20.1 Brief description

Get the version information of the framework.

7.21.20.2 Detailed description

Table 837:Return values

Name	Description
SSP_SUCCESS	Got version number successfully.
SSP_ERR_ASSERTION	Following parameters is NULL: p_version.

7.21.20.3 Function steps

- Checks error. Further parameter checking can be done at the driver layer.

7.21.21 g_sf_i2c_version

ssp_version_t::g_sf_i2c_version

7.21.21.1 Detailed description

Version data structure used by error logger macro.

7.21.21.2 Initialized as

```
g_sf_i2c_version=
{
.api_version_minor  = SF_I2C_API_VERSION_MINOR,
    .api_version_major  = #define  SF_I2C_API_VERSION_MAJOR,
.code_version_minor = SF_I2C_CODE_VERSION_MINOR,
    .code_version_major = SF_I2C_CODE_VERSION_MAJOR,
}
```

7.21.22 g_sf_i2c_on_sf_i2c

sf_i2c_api_t::g_sf_i2c_on_sf_i2c

7.21.22.1 Initialized as

```
g_sf_i2c_on_sf_i2c=
{
    .open    = SF_I2C_Open,
    .read    = SF_I2C_Read,
    .write   = SF_I2C_Write,
    .reset   = SF_I2C_Reset,
    .close   = SF_I2C_Close,
    .lock    = SF_I2C_Lock,
    .unlock  = SF_I2C_Unlock,
    .version = SF_I2C_VersionGet
}
```

7.21.23 Extensions

7.21.23.1 sf_i2c_instance_ctrl_t

sf_i2c_instance_ctrl_t

Detailed description

I2C instance control block. DO NOT INITIALIZE. Initialization occurs when open is called.

Variables

- sf_i2c_bus_t * p_bus

  Bus using this device. Copy from configuration structure.
- i2c_master_instance_t const * p_lower_lvl_i2c

  I2C instance.
- i2c_cfg_t lower_lvl_cfg

  Used to reconfigure I2C driver.
- i2c_ctrl_t * p_lower_lvl_ctrl

  I2C peripheral control block.
- sf_i2c_dev_state_t dev_state

  Device status.
- bool locked

  Lock and unlock bus for a device.
- bool restarted

  Indicates whether device issued a restart.

7.22 JPEG Decode Framework

RTOS-integrated JPEG Framework.

7.22.1 Functions

- sf_jpeg_initialize
- sf_jpeg_callback_function
- SF_JPEG_Decode_Open
- SF_JPEG_Decode_InputBufferSet
- SF_JPEG_Decode_LinesDecodedGet
- SF_JPEG_Decode_HorizontalStrideSet
- SF_JPEG_Decode_ImageSubsampleSet
- SF_JPEG_Decode_OutputBufferSet
- SF_JPEG_Decode_Wait
- SF_JPEG_Decode_StatusGet
- SF_JPEG_Decode_PixelFormatGet
- SF_JPEG_Decode_ImageSizeGet
- SF_JPEG_Decode_Close

- SF_JPEG_Decode_VersionGet

7.22.2 Variables

- s_sf_jpeg_version
- p_event_flag
- g_sf_jpeg_decode_on_sf_jpeg_decode

7.22.3 Defines

- #define SF_JPEG_DECODE_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file
- #define SF_JPEG_DECODE_CODE_VERSION_MINOR

  Initial value: `(5U)`
- #define SF_JPEG_DECODE_OPEN

  Initial value: `(0x4A504547U)`

  "JPEG" in ASCII, used to identify general JPEG control block
- #define SF_JPEG_ERROR_RETURN

  Initial value:
  ```
  #define  SSP_ERROR_RETURN((a), (err), &g_module_name[0], &s_sf_jpeg_version)
  ```

  Macro for error logger.
- #define JPEG_ALL_EVENTS

  Initial value: `((ULONG) 0xFFFFFFFF)`

7.22.4 sf_jpeg_initialize

```
ssp_err_t sf_jpeg_initialize ( sf_jpeg_decode_instance_ctrl_t  *const p_ctrl ,
 sf_jpeg_decode_cfg_t  const *const p_cfg )
```

7.22.4.1 Brief description

Acquires mutex, then handles driver initialization at the HAL layer. This function releases the mutex before returns to the caller.

7.22.4.2 Detailed description

Table 838:Return values

Name	Description
SSP_SUCCESS	JPEG Decode driver is successfully opened.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- open

7.22.5 sf_jpeg_callback_function

```
sf_jpeg_callback_function ( jpeg_decode_callback_args_t  * p_args )
```

7.22.6 SF_JPEG_Decode_Open

```
ssp_err_t SF_JPEG_Decode_Open ( sf_jpeg_decode_ctrl_t  *const p_api_ctrl ,
 sf_jpeg_decode_cfg_t  const *const p_cfg )
```

7.22.6.1 Brief description

Parameter checking and initialize JPEG decode with sf_jpeg_initialize helper function and marking the open flag in control block.

7.22.6.2 Detailed description

Table 839:Return values

Name	Description
SSP_SUCCESS	JPEG Decode framework is successfully opened.
SSP_ERR_ASSERTION	One of the following parameters may be null: p_ctrl or p_cfg.
SSP_ERR_ALREADY_OPEN	JPEG Decode framework is already open.

Table 839:Return values (Continued)

Name	Description
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes.

7.22.7 SF_JPEG_Decode_InputBufferSet

```
ssp_err_t SF_JPEG_Decode_InputBufferSet ( sf_jpeg_decode_ctrl_t
*const p_api_ctrl ,   void *const p_buffer ,   uint32_t const buffer_size )
```

7.22.7.1 Brief description

Configures JPEG coded input data.

7.22.7.2 Detailed description

This API configures the decode input buffer address register. After the input buffer address is set, the driver checks whether the output buffer address is set, and verifies that the output buffer size is large enough to hold at least eight output lines of data. If both the input buffer and output buffer are set properly, the driver automatically starts the decode process.

NOTE: Call SF_JPEG_Decode_Open to configure the JPEG codec block before using this function.

Table 840:Return values

Name	Description
SSP_SUCCESS	Decode input buffer is successfully configured.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- inputBufferSet

7.22.7.3 Function steps

- Obtain mutex before making HAL-level driver call.
- Call the HAL driver layer inputBufferSet routine.
- Release mutex

7.22.8 SF_JPEG_Decode_LinesDecodedGet

```
ssp_err_t SF_JPEG_Decode_LinesDecodedGet ( sf_jpeg_decode_ctrl_t
*const p_api_ctrl ,   uint32_t *const p_lines )
```

7.22.8.1 Brief description

Obtain number of lines decoded by the codec.

7.22.8.2 Detailed description

NOTE: Call SF_JPEG_Decode_Open to configure the JPEG codec block before using this function.

Table 841:Return values

Name	Description
SSP_SUCCESS	Lines decoded value is successfully obtained.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- linesDecodedGet

7.22.8.3 Function steps

- Obtain mutex before making HAL-level driver call.
- Release mutex

7.22.9 SF_JPEG_Decode_HorizontalStrideSet

```
ssp_err_t SF_JPEG_Decode_HorizontalStrideSet ( sf_jpeg_decode_ctrl_t
*const p_api_ctrl ,   uint32_t horizontal_stride )
```

7.22.9.1 Brief description

Configure the horizontal stride value.

7.22.9.2 Detailed description

NOTE: Call SF_JPEG_Decode_Open to configure the JPEG codec block before using this function.

Table 842:Return values

Name	Description
SSP_SUCCESS	Horizontal Stride value is successfully configured.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- horizontalStrideSet

7.22.9.3 Function steps

- Obtain mutex before making HAL-level driver call.
- Release mutex

7.22.10 SF_JPEG_Decode_ImageSubsampleSet

```
ssp_err_t SF_JPEG_Decode_ImageSubsampleSet ( sf_jpeg_decode_ctrl_t
*const p_api_ctrl ,  jpeg_decode_subsample_t  horizontal_subsample ,
 jpeg_decode_subsample_t  vertical_subsample )
```

7.22.10.1 Brief description

Configure the horizontal and vertical subsample values. This allows an application to reduce the size of the decoded image at runtime.

7.22.10.2 Detailed description

NOTE: Call SF_JPEG_Decode_Open to configure the JPEG codec block before using this function.

Table 843:Return values

Name	Description
SSP_SUCCESS	Image subsample values are successfully configured.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- imageSubsampleSet

7.22.10.3 Function steps

- Obtain mutex before making HAL-level driver call.
- Release mutex

7.22.11 SF_JPEG_Decode_OutputBufferSet

```
ssp_err_t SF_JPEG_Decode_OutputBufferSet ( sf_jpeg_decode_ctrl_t
*const p_api_ctrl ,   void * p_buffer ,   uint32_t buffer_size )
```

7.22.11.1 Brief description

Configure the decode output buffer.

7.22.11.2 Detailed description

This API configures the decode output buffer address register. After the output buffer address is set, the driver computers the number of output lines the buffer is able to hold. The hardware requires the number out output lines to decode at a time is multiple of eight. If both the input buffer and output buffer are set properly, the driver automatically starts the decode process.

NOTE: Call SF_JPEG_Decode_Open to configure the JPEG codec block before using this function.

Table 844:Return values

Name	Description
SSP_SUCCESS	Output buffer is successfully configured.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- outputBufferSet

7.22.11.3 Function steps

- Obtain mutex before making HAL-level driver call.
- Release mutex

7.22.12 SF_JPEG_Decode_Wait

```
ssp_err_t SF_JPEG_Decode_Wait ( sf_jpeg_decode_ctrl_t  *const p_api_ctrl ,
 jpeg_decode_status_t  *const p_status ,   uint32_t timeout )
```

7.22.12.1 Brief description

Wait for current JPEG codec operation to finish.

7.22.12.2 Detailed description

NOTE: Call SF_JPEG_Decode_Open to configure the JPEG codec block before using this function.

Table 845:Return values

Name	Description
SSP_SUCCESS	The wait function returns successfully.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_TIMEOUT	The wait operation timed out, the underlying driver did not response in time.
SSP_ERR_WAIT_ABORTED	System internal error occurred.

7.22.12.3 Function steps

- Obtain mutex before making HAL-level driver call.

7.22.13 SF_JPEG_Decode_StatusGet

```
ssp_err_t SF_JPEG_Decode_StatusGet ( sf_jpeg_decode_ctrl_t  *const p_api_ctrl ,
 jpeg_decode_status_t  *const p_status )
```

7.22.13.1 Brief description

Obtain JPEG codec status. This function can be used to poll the device instead of using SF_JPEG_Decode_Wait to block on JPEG operations.

7.22.13.2 Detailed description

NOTE: Call SF_JPEG_Decode_Open to configure the JPEG codec block before using this function.

Table 846:Return values

Name	Description
SSP_SUCCESS	The JPEG status information is obtained.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- statusGet

7.22.13.3 Function steps

- Obtain mutex before making HAL-level driver call.
- Release mutex

7.22.14 SF_JPEG_Decode_PixelFormatGet

```
ssp_err_t SF_JPEG_Decode_PixelFormatGet ( sf_jpeg_decode_ctrl_t
*const p_api_ctrl ,  jpeg_decode_color_space_t  *const p_color_space )
```

7.22.14.1 Brief description

Obtain the format of the image. This function is only useful for decoding a JPEG image.

7.22.14.2 Detailed description

NOTE: Call SF_JPEG_Decode_Open to configure the JPEG codec block before using this function.

Table 847:Return values

Name	Description
SSP_SUCCESS	The JPEG image size is obtained.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- pixelFormatGet

7.22.14.3 Function steps

- Obtain mutex before making HAL-level driver call.
- Release mutex

7.22.15 SF_JPEG_Decode_ImageSizeGet

```
ssp_err_t SF_JPEG_Decode_ImageSizeGet ( sf_jpeg_decode_ctrl_t
*const p_api_ctrl ,   uint16_t * p_horizontal_size ,   uint16_t
* p_vertical_size )
```

7.22.15.1 Brief description

Obtain the size of the image. This function is only useful for decoding a JPEG image.

7.22.15.2 Detailed description

NOTE: Call SF_JPEG_Decode_Open to configure the JPEG codec block before using this function.

Table 848:Return values

Name	Description
SSP_SUCCESS	The JPEG image size is obtained.
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- imageSizeGet

7.22.15.3 Function steps

- Obtain mutex before making HAL-level driver call.
- Release mutex

7.22.16 SF_JPEG_Decode_Close

```
ssp_err_t SF_JPEG_Decode_Close ( sf_jpeg_decode_ctrl_t  *const p_api_ctrl )
```

7.22.16.1 Brief description

Close JPEG codec device. Un-finished codec operation is interrupted, and output data are discarded.

7.22.16.2 Detailed description

NOTE: Call SF_JPEG_Decode_Open to configure the timer before using this function.

Table 849:Return values

Name	Description
SSP_SUCCESS	The JPEG decode device is successfully closed.

Table 849:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	p_ctrl is null.
SSP_ERR_NOT_OPEN	JPEG Decode Framework module is not yet initialized.
SSP_ERR_IN_USE	The mutex may be unavailable for the the device. See HAL driver for other possible causes.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- close

7.22.16.3 Function steps

- Obtain mutex before making HAL-level driver call.
- Release mutex
- Clear information from control block so other functions know this block is closed
- Delete RTOS services allocated during the open call.

7.22.17 SF_JPEG_Decode_VersionGet

```
ssp_err_t SF_JPEG_Decode_VersionGet ( ssp_version_t  *const p_version )
```

7.22.17.1 Brief description

Get version and store it in provided pointer p_version.

7.22.17.2 Detailed description

Table 850:Return values

Name	Description
SSP_SUCCESS	Version returned successfully.
SSP_ERR_ASSERTION	Parameter p_version is null.

7.22.18 g_sf_jpeg_decode_on_sf_jpeg_decode

sf_jpeg_decode_api_t::g_sf_jpeg_decode_on_sf_jpeg_decode

7.22.18.1 Initialized as

```
g_sf_jpeg_decode_on_sf_jpeg_decode=
{
    .open                   = SF_JPEG_Decode_Open,
    .inputBufferSet         = SF_JPEG_Decode_InputBufferSet,
    .outputBufferSet        = SF_JPEG_Decode_OutputBufferSet,
    .linesDecodedGet        = SF_JPEG_Decode_LinesDecodedGet,
    .horizontalStrideSet    = SF_JPEG_Decode_HorizontalStrideSet,
    .imageSubsampleSet      = SF_JPEG_Decode_ImageSubsampleSet,
    .wait                   = SF_JPEG_Decode_Wait,
    .statusGet              = SF_JPEG_Decode_StatusGet,
    .imageSizeGet           = SF_JPEG_Decode_ImageSizeGet,
    .pixelFormatGet         = SF_JPEG_Decode_PixelFormatGet,
    .close                  = SF_JPEG_Decode_Close,
    .versionGet             = SF_JPEG_Decode_VersionGet
}
```

7.22.19 Extensions

7.22.19.1 sf_jpeg_decode_instance_ctrl_t

sf_jpeg_decode_instance_ctrl_t

Detailed description

JPEG framework instance control block. DO NOT INITIALIZE. Initialization occurs when open is called.

Variables

- uint32_t open

  Indicate whether the driver is open.

- uint32_t state

  Used by driver to check if pointer to control block is valid.

- TX_MUTEX mutex

  Mutex used to protect access to lower level driver hardware.

- TX_EVENT_FLAGS_GROUP events

  Event flags used by the HAL driver to notify the framework driver of.

- jpeg_decode_instance_t const * p_lower_lvl_jpeg_decode

  Pointer to lower level instance.

7.23 Messaging Framework

RTOS-integrated Messaging Framework implementation.

7.23.1 Functions

- SF_MESSAGE_Open
- SF_MESSAGE_Close
- SF_MESSAGE_BufferAcquire
- SF_MESSAGE_BufferRelease
- SF_MESSAGE_Post
- SF_MESSAGE_Pend
- SF_MESSAGE_VersionGet

7.23.2 Defines

- #define SF_MESSAGE_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file
- #define SF_MESSAGE_CODE_VERSION_MINOR

  Initial value: `(5U)`
- #define SF_MESSAGE_QUEUE_MESSAGE_WORDS

  Initial value: `(1)`

  The size of a message queue in words

7.23.3 SF_MESSAGE_Open

```
ssp_err_t SF_MESSAGE_Open ( sf_message_ctrl_t  *const p_api_ctrl ,
 sf_message_cfg_t  const *const p_cfg )
```

7.23.3.1 Brief description

Initialize message framework. This function initiates the messaging framework control block, configures the work memory corresponding to the configuration parameters.

7.23.3.2 Detailed description

Table 851:Return values

Name	Description
SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	p_ctrl, p_cfg or p_cfg->p_work_memory_start is NULL.
SSP_ERR_INTERNAL	OS service call fails.
SSP_ERR_IN_USE	The Messaging Framework is in use.
SSP_ERR_INVALID_WORKBUFFER_SIZE	Invalid work buffer size.
SSP_ERR_INVALID_MSG_BUFFER_SIZE	Message buffer size is invalid.
SSP_ERR_ILLEGAL_SUBSCRIBER_LISTS	Message subscriber lists is illegal.

NOTE: This API function is allowed to be called once per instance. The behavior if called twice is undefined.

NOTE: This API function only allows to be called from thread context.

7.23.3.3 Function steps

- Creates the memory pools in the work memory area
- Registers subscriber lists
- Changes the messaging framework status from CLOSED to OPENED

7.23.4 SF_MESSAGE_Close

```
ssp_err_t SF_MESSAGE_Close ( sf_message_ctrl_t  *const p_api_ctrl )
```

7.23.4.1 Brief description

Closes message framework.

7.23.4.2 Detailed description

Table 852:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Message framework module has yet to be opened.
SSP_ERR_ILLEGAL_SUBSCRIBER_LISTS	Message subscriber lists is illegal.
SSP_ERR_INTERNAL	OS service call fails

NOTE: This API function only allows to be called from thread context.

7.23.4.3 Function steps

- Finds subscribers and flushes their queues
- Deletes memory pools allocated in the work memory

7.23.5 SF_MESSAGE_BufferAcquire

```
ssp_err_t SF_MESSAGE_BufferAcquire ( sf_message_ctrl_t  const
*const p_api_ctrl ,  sf_message_header_t  ** pp_buffer ,
 sf_message_acquire_cfg_t  const *const p_acquire_cfg ,   uint32_t
const wait_option )
```

7.23.5.1 Brief description

Acquire buffer for message passing from the block.

7.23.5.2 Detailed description

Table 853:Return values

Name	Description
SSP_SUCCESS	Buffer acquisition was successful.

Table 853:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	p_ctrl, p_acquire_cfg or pp_buffer is NULL.
SSP_ERR_NOT_OPEN	Message framework module has yet to be opened.
SSP_ERR_NO_MORE_BUFFER	No more buffer found in the memory block pool.
SSP_ERR_TIMEOUT	OS service call returns timeout.
SSP_ERR_INTERNAL	OS service call fails.

NOTE: This API function allows to be called from not only thread but also ISR.

7.23.5.3 Function steps

- Allocates buffer in the block memory pool.
- Clears buffer control block
- Sets the buffer in-use flag
- Sets the address of the allocated buffer to 'pp_buffer'
- Clears the event class and event code in the buffer. This is because the initial value in the buffer control block is unknown and it would not be safe.
- Sets the 'buffer_keep' flag in the buffer control block if SF_MESSAGE_ACQUIRE_OPTION_KEEP is set to the 'option' argument

7.23.6 SF_MESSAGE_BufferRelease

```
ssp_err_t SF_MESSAGE_BufferRelease ( sf_message_ctrl_t  *const p_api_ctrl ,
 sf_message_header_t  *const p_buffer ,  sf_message_release_option_t
const option )
```

7.23.6.1 Brief description

Release buffer obtained by SF_MESSAGE_BufferAcquire.

7.23.6.2 Detailed description

Table 854:Return values

Name	Description
SSP_SUCCESS	Buffer release was successful.
SSP_ERR_NOT_OPEN	Message framework module has yet to be opened.
SSP_ERR_ASSERTION	p_ctrl or p_buffer is NULL.
SSP_ERR_ILLEGAL_BUFFER_ADDRESS	If buffer address is not aligned or p_buffer is not in the block pool range.
SSP_ERR_BUFFER_RELEASED	Buffer has been released.

NOTE: This API function allows to be called from thread but also from ISR.

7.23.6.3 Function steps

- Calculates the address of the buffer control block
- Release buffer in the condition below. (1) The counting semaphore is zero and the buffer keep option is not specified. (2) 'option' is set to SF_MESSAGE_RELEASE_OPTION_FORCED_RELEASE.
- Clears the flags in the buffer control block.
- Set back the backed up interrupt mask level.
- Invokes an user callback function if it is registered in the condition below. (1) The counting semaphore is zero. (2) 'option' is set to SF_MESSAGE_RELEASE_OPTION_FORCED_RELEASE.
- Sets SF_MESSAGE_CALLBACK_EVENT_NAK if any subscribers for the message have responded NAK
- Sets SF_MESSAGE_CALLBACK_EVENT_ACK if all subscribers for the message have responded ACK
- Sets the pointer to the context to kept in the buffer control block
- Invokes the registered user callback function.

7.23.7 SF_MESSAGE_Post

```
ssp_err_t SF_MESSAGE_Post ( sf_message_ctrl_t  *const p_api_ctrl ,
 sf_message_header_t  const *const p_buffer ,  sf_message_post_cfg_t  const
*const p_post_cfg ,  sf_message_post_err_t  *const p_post_err ,   uint32_t
const wait_option )
```

7.23.7.1 Brief description

Post a message to the subscribers.

7.23.7.2 Detailed description

Table 855:Return values

Name	Description
SSP_SUCCESS	Message posting was successful.
SSP_ERR_ASSERTION	p_ctrl or p_buffer is NULL.
SSP_ERR_NOT_OPEN	Message framework module has yet to be opened.
SSP_ERR_NO_SUBSCRIBER_FOUND	No subscriber found.
SSP_ERR_BUFFER_RELEASED	Buffer has been released.
SSP_ERR_MESSAGE_QUEUE_FULL	Queue is full (Timeout occurs before sending a message if timeout option is specified)
SSP_ERR_ILLEGAL_BUFFER_ADDRESS	If buffer address is not aligned or p_buffer is not in the block pool range.
SSP_ERR_INTERNAL	OS service call fails

NOTE: This API function allows to be called from not only thread but also ISR(if wait_option is TX_NO_WAIT).

NOTE: Another buffer writing to the buffer before the message read by message consumers results data overwriting.

7.23.7.3 Function steps

- Checks the number of the subscribers of specified event class
- Calculates the address of the buffer control block
- Counts up the counting semaphore in the buffer control block
- Registers user callback function and context passed from user

7.23.8 SF_MESSAGE_Pend

```
ssp_err_t SF_MESSAGE_Pend ( sf_message_ctrl_t  const *const p_api_ctrl ,
  TX_QUEUE const *const p_queue ,  sf_message_header_t  ** pp_buffer ,  uint32_t
const wait_option )
```

7.23.8.1 Brief description

Pend on a message.

7.23.8.2 Detailed description

Table 856:Return values

Name	Description
SSP_SUCCESS	Message pending was successful.
SSP_ERR_ASSERTION	p_ctrl, pp_buffer or p_queue is NULL.
SSP_ERR_NOT_OPEN	Message framework module has yet to be opened.
SSP_ERR_MESSAGE_QUEUE_EMPTY	Queue is empty.
SSP_ERR_TIMEOUT	OS service call returns timeout.
SSP_ERR_INTERNAL	OS service call fails.

NOTE: This API function allows to be called from not only thread but also ISR(if wait_option is TX_NO_WAIT).

7.23.8.3 Function steps

- Receiving message here. Receiving data is not message itself but the pointer to the buffer
- If there is no data in the message queue and TX_NO_WAIT is specified to wait_option, return immediately with SSP_ERR_MESSAGE_QUEUE_EMPTY error code

7.23.9 SF_MESSAGE_VersionGet

```
ssp_err_t SF_MESSAGE_VersionGet ( ssp_version_t  *const p_version )
```

7.23.9.1 Brief description

Get the version of the messaging framework. Stores version information in provided pointer.

7.23.9.2 Detailed description

Table 857:Return values

Name	Description
SSP_SUCCESS	Got version number successfully.
SSP_ERR_ASSERTION	p_version is NULL.

7.23.10 Extensions

7.23.10.1 sf_message_instance_ctrl_t

sf_message_instance_ctrl_t

Detailed description

Messaging framework instance control block structure

Variables

- TX_BLOCK_POOL block_pool

  Pointer to the memory block pool control.
- sf_message_subscriber_list_t ** pp_subscriber_lists

  Pointer array to the subscriber lists.
- uint32_t buffer_size

  Bytes of the message buffer.
- uint32_t number_of_buffers

  The number of allocated buffers.
- uint16_t number_of_subscriber_groups

  The number of subscriber groups.
- sf_message_state_t state

  Status of the message framework.

7.24 Power Profiles Framework

Power Profiles Framework.

7.24.1 Functions

- SF_POWER_PROFILES_Open
- SF_POWER_PROFILES_Sleep
- SF_POWER_PROFILES_Close
- SF_POWER_PROFILES_VersionGet

7.24.2 Defines

- #define SF_POWER_PROFILES_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file
- #define SF_POWER_PROFILES_CODE_VERSION_MINOR

  Initial value: `(4U)`

7.24.3 SF_POWER_PROFILES_Open

```
ssp_err_t SF_POWER_PROFILES_Open ( sf_power_profiles_ctrl_t  *const p_ctrl ,
 sf_power_profiles_cfg_t  const *const p_cfg )
```

7.24.3.1 Brief description

Configures the Power Profiles framework and opens any required HAL layer drivers that will be used.

7.24.3.2 Detailed description

The SF_POWER_PROFILES_Open function acquires a mutex for the Power Profile module, it then calls the open/init functions for the necessary lower level drivers (RTC, LPM) and finally calls the driver .open function for the Power Profiles module. The mutex is released following the driver layer open functions.

Table 858:Return values

Name	Description
SSP_SUCCESS	Initialization was successful.

Table 858:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_ctrl, p_api, or p_cfg. See HAL driver for other possible causes.
SSP_ERR_IN_USE	The mutex may be unavailable for the unit requested. See HAL driver for other possible causes.
SSP_ERR_INVALID_HW_CONDITION	Incompatible system clock configuration.
SSP_ERR_INTERNAL	Creation of mutex failed

See Common Error Codes or HAL driver for other possible return codes or causes.

NOTE: - For ThreadX specific API details, refer ThreadX user manual.

7.24.3.3 Function steps

- Save driver structure pointer for use in other framework layer functions
- Keep the control pointer that is associated with the already open RTC module
- Keep the pointer to the callback function
- Keep the pointer to the sleep and awake ioport tables
- Make a local copy of the LPM configuration structure to modify it
- Initialize the LPM HAL driver
- Mark driver as open by initializing it to "PPMG" in its ASCII equivalent

7.24.4 SF_POWER_PROFILES_Sleep

```
ssp_err_t SF_POWER_PROFILES_Sleep ( sf_power_profiles_ctrl_t  *const p_ctrl )
```

7.24.4.1 Brief description

Places the MCU in Software Standby mode.

7.24.4.2 Detailed description

Table 859:Return values

Name	Description
SSP_SUCCESS	Entered Sleep and then Awoke, or called with mode == SF_POWER_PROFILES_MODE_RUN.
SSP_ERR_ASSERTION	p_ctrl or p_ctrl->p_api is NULL.
SSP_ERR_NOT_OPEN	Driver control block not valid. Call SF_POWER_PROFILES_Open to configure.
SSP_ERR_UNSUPPORTED	This function is not supported by the HAL driver (p_ctrl->p_api->start is NULL).

See Common Error Codes or HAL driver for other possible return codes or causes.

NOTE: - For ThreadX specific API details, refer ThreadX user manual.

7.24.4.3 Function steps

- PREPARE FOR SLEEP MODE
- Set the ioport pins per the supplied sleep configuration
- Check to see if LOCO and/or Subclk are currently running
- Get the current WUPEN bits for restore later
- Get some info about the RTC
- Set the WUPEN bits such the RTC Periodic and Alarm are capable of interrupting and waking them MCU
- Stop the LOCO
- Stop the Sub-Clock
- Set the WUPEN bits such the RTC Periodic and Alarm are capable of interrupting and waking them MCU
- These clocks are allowed to be on during Software Standby mode. If the user is not using RTC mode then we will turn these off.
- Stop the LOCO
- Stop the Sub-Clock
- S7G2

- Some peripherals can be set to run even in Software Standby mode. To insure lowest power usage we will turn those off before going to sleep, if they are on and restore them on Wakeup These include: SCI0 (MSTPCRB b31) IIC0 (MSTPCRB b09)
- MSTPCRB 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1
- Above are our desired bit settings for MSTPCRB, our MSTBCRB mask is therefore 0x7FFFFDFF
- D/A Convertor 0 (MSTPCRD b20), AGT1 (MSTPCRD b1) AGT0 (MSTPCRD b3) Comparator-OC5 (MSTPCRD b23) Comparator-OC4 (MSTPCRD b24) Comparator-OC3 (MSTPCRD b25) Comparator-OC2 (MSTPCRD b26) Comparator-OC1 (MSTPCRD b27) Comparator-OC0 (MSTPCRD b28)
- MSTPCRD 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 1 1 1 0 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1
- Above are our desired bit settings for MSTPCRD, our MSTBCRD mask is therefore 0xE06FFFF3
- S3A7
- Some peripherals can be set to run even in Software Standby mode. To insure lowest power usage we will turn those off before going to sleep, if they are on and restore them on Wakeup These include: SCI0 (MSTPCRB b31) IIC0 (MSTPCRB b09)
- MSTPCRB 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1
- Above are our desired bit settings for MSTPCRB, our MSTBCRB mask is therefore 0x7FFFFDFF
- D/A Convertor 0 (MSTPCRD b20), AGT1 (MSTPCRD b2) AGT0 (MSTPCRD b3) Comparator-OC0 (MSTPCRD b28) Low Power Comparator (MSTPCRD b29)
- MSTPCRD 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1
- Above are our desired bit settings for MSTPCRD, our MSTBCRD mask is therefore 0xCFEFFFF3
- < Disable the LCD
- issue the Pre-Sleep Callback
- Populate the callback argument fields
- Call the user callback
- Setup to enter the low power mode
- ENTER STANDBY MODE
- AWAKE
- Restore any clocks we stopped
- Restore the original mstp mode
- < Reenable the LCD
- Set the ioport pins per the supplied wake configuration
- issue the Post-Sleep Callback
- Populate the callback argument fields
- Call the user callback

- Restore the WUPEN contents to what they were before we entered Software Standby mode

7.24.5 SF_POWER_PROFILES_Close

```
ssp_err_t SF_POWER_PROFILES_Close ( sf_power_profiles_ctrl_t  *const p_ctrl )
```

7.24.5.1 Brief description

The close function acquires the unit's mutex, calls the driver close function, and releases the mutex.

7.24.5.2 Detailed description

Table 860:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_ctrl or p_ctrl->p_api is NULL.
SSP_ERR_NOT_OPEN	Driver control block not valid. Call SF_POWER_PROFILES_Open to configure.
SSP_ERR_UNSUPPORTED	This function is not supported by the HAL driver (p_ctrl->p_api->close is NULL).
SSP_ERR_INTERNAL	Deletion of mutex failed.

See Common Error Codes or HAL driver for other possible return codes or causes.

NOTE: - For ThreadX specific API details, refer ThreadX user manual.

7.24.5.3 Function steps

- Clear information from control block so other functions know this block is closed

7.24.6 SF_POWER_PROFILES_VersionGet

```
ssp_err_t SF_POWER_PROFILES_VersionGet ( ssp_version_t  *const p_version )
```

7.24.6.1 Brief description

Gets version and stores it in provided pointer p_version.

7.24.6.2 Detailed description

Table 861:Return values

Name	Description
SSP_SUCCESS	Version returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was null.

7.25 Power Profiles V2 Framework

Power Profiles Framework.

7.25.1 Functions

- SF_POWER_PROFILES_V2_Open
- SF_POWER_PROFILES_V2_RunApply
- SF_POWER_PROFILES_V2_LowPowerApply
- SF_POWER_PROFILES_V2_Close
- SF_POWER_PROFILES_V2_VersionGet
- clock_config_apply
- low_power_config_apply
- ioport_cfg_apply
- pre_low_power_callback_handle
- post_low_power_callback_handle

7.25.2 Defines

- #define SF_POWER_PROFILES_V2_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file

- #define SF_POWER_PROFILES_V2_CODE_VERSION_MINOR

  Initial value: (0U)

7.25.3 SF_POWER_PROFILES_V2_Open

```
ssp_err_t SF_POWER_PROFILES_V2_Open ( sf_power_profiles_v2_ctrl_t
*const p_ctrl ,  sf_power_profiles_v2_cfg_t  const *const p_cfg )
```

7.25.3.1 Brief description

Configures the Power Profiles framework and opens any required HAL layer drivers that will be used.

7.25.3.2 Detailed description

The SF_POWER_PROFILES_V2_Open function initializes the critical data structures and variables.

Table 862:Return values

Name	Description
SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_ctrl, p_api, or p_cfg. See HAL driver for other possible causes.
SSP_ERR_IN_USE	Power profiles framework is already open.
SSP_ERR_INTERNAL	Unable to obtain mutex. Unable to release mutex.
See	Common Error Codes for other possible return codes or causes.

7.25.4 SF_POWER_PROFILES_V2_RunApply

```
ssp_err_t SF_POWER_PROFILES_V2_RunApply ( sf_power_profiles_v2_ctrl_t
*const p_ctrl ,  sf_power_profiles_v2_run_cfg_t  const *const p_cfg )
```

7.25.4.1 Brief description

Applies a Run profile.

7.25.4.2 Detailed description

The SF_POWER_PROFILES_V2_RunApply function will:

- Apply an IO port configuration, if supplied

- Apply a clock configuration

Table 863:Return values

Name	Description
SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_ctrl, p_api, or p_cfg. See HAL driver for other possible causes.
SSP_ERR_INVALID_ARGUMENT	Clock configuration is invalid.
SSP_ERR_NOT_OPEN	Power profiles framework is not open.
SSP_ERR_IN_USE	Unable to obtain mutex.
SSP_ERR_INTERNAL	Unable to release mutex.
SSP_ERR_INVALID_HW_CONDITION	Incompatible system clock configuration.
See	Common Error Codes, r_ioport, or r_cgc driver for other possible return codes or causes.

7.25.5 SF_POWER_PROFILES_V2_LowPowerApply

```
ssp_err_t SF_POWER_PROFILES_V2_LowPowerApply ( sf_power_profiles_v2_ctrl_t
*const p_ctrl ,  sf_power_profiles_v2_low_power_cfg_t  const *const p_cfg )
```

7.25.5.1 Brief description

Applies a Low Power profile.

7.25.5.2 Detailed description

The SF_POWER_PROFILES_V2_LowPowerApply function will:

- Apply a LPMv2 configuration to prepare for low power
- Apply an IO port configuration to prepare for low power, if supplied
- Notify application that low power is about to be entered
- Enter low power mode
- When low power mode is exited, apply an IO port configuration for wake up, if supplied
- Notify application that wake up has occurred

Table 864:Return values

Name	Description
SSP_SUCCESS	Entered and exited low power mode successfully.
SSP_ERR_ASSERTION	p_ctrl or p_ctrl->p_api is NULL.
SSP_ERR_NOT_OPEN	Power profiles framework is not open.
SSP_ERR_UNSUPPORTED	This function is not supported by one of the HAL drivers, r_lpmv2, r_ioport.
SSP_ERR_INVALID_MODE	r_lpmv2 mode is not LPMV2_LOW_POWER_MODE_SLEEP but r_lmv2 p_extend is NULL.
SSP_ERR_IN_USE	Unable to obtain mutex.
SSP_ERR_INTERNAL	Unable to release mutex.
See	Common Error Codes, r_ioport, or r_lpmv2 drivers for other possible return codes or causes.

7.25.6 SF_POWER_PROFILES_V2_Close

```
ssp_err_t SF_POWER_PROFILES_V2_Close ( sf_power_profiles_v2_ctrl_t
*const p_ctrl )
```

7.25.6.1 Brief description

Closes the framework.

7.25.6.2 Detailed description

Table 865:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Power profiles framework is not open.

Table 865:Return values (Continued)

Name	Description
SSP_ERR_IN_USE	Unable to obtain mutex.
SSP_ERR_INTERNAL	Unable to release mutex.

7.25.7 SF_POWER_PROFILES_V2_VersionGet

```
ssp_err_t SF_POWER_PROFILES_V2_VersionGet ( ssp_version_t  *const p_version )
```

7.25.7.1 Brief description

Gets version and stores it in provided pointer p_version.

7.25.7.2 Detailed description

Table 866:Return values

Name	Description
SSP_SUCCESS	Version returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was null.

7.25.8 clock_config_apply

```
ssp_err_t clock_config_apply ( cgc_clocks_cfg_t  const *const p_clock_cfg )
```

7.25.8.1 Brief description

Internal function that checks and applies the clock configuration.

7.25.8.2 Detailed description

Table 867:Return values

Name	Description
SSP_SUCCESS	Clock configuration applied successfully.

Table 867:Return values (Continued)

Name	Description
See	Common Error Codes or r_cgc driver for other possible return codes or causes.

7.25.9 low_power_config_apply

```
ssp_err_t low_power_config_apply ( sf_power_profiles_v2_ctrl_t  *const p_ctrl ,
 sf_power_profiles_v2_low_power_cfg_t  const *const p_cfg )
```

7.25.9.1 Brief description

Apply the low power configuration using the lpm driver. Internal function, do not use directly. Returns mutex if an error occurs.

7.25.9.2 Detailed description

Table 868:Parameters

Name	Direction	Description
p_ctrl	in	PPM V2 control structure
p_cfg	in	PPM V2 config structure

Table 869:Return values

Name	Description
SSP_SUCCESS	LPM configuration was successful.
See	Common Error Codes or r_lpmv2 driver for other possible return codes or causes.

7.25.10 ioport_cfg_apply

```
ssp_err_t ioport_cfg_apply ( sf_power_profiles_v2_ctrl_t  *const p_ctrl ,
 ioport_cfg_t  const * p_ioport_pin_tbl )
```

7.25.10.1 Brief description

Apply the IO Port configuration using the ioport driver. Internal function, do not use directly. Returns mutex if an error occurs.

7.25.10.2 Detailed description

Table 870:Parameters

Name	Direction	Description
p_ctrl	in	PPM V2 control structure
p_ioport_pin_tbl	in	Pointer to ioport settings

Table 871:Return values

Name	Description
SSP_SUCCESS	IO Port configuration was successful.
See	Common Error Codes or r_ioport driver for other possible return codes or causes.

7.25.11 pre_low_power_callback_handle

```
 pre_low_power_callback_handle ( sf_power_profiles_v2_low_power_cfg_t  const
*const p_cfg )
```

7.25.11.1 Brief description

Notify application using pre-low power callback. Internal function, do not use directly.

7.25.11.2 Detailed description

Table 872:Parameters

Name	Direction	Description
p_cfg	in	PPM V2 config structure

Table 873:Return values

Name	Description
none	

7.25.12 post_low_power_callback_handle

```
 post_low_power_callback_handle ( sf_power_profiles_v2_low_power_cfg_t  const
*const p_cfg )
```

7.25.12.1 Brief description

Notify application using post-low power callback. Internal function, do not use directly.

7.25.12.2 Detailed description

Table 874:Parameters

Name	Direction	Description
p_cfg	in	PPM V2 config structure

Table 875:Return values

Name	Description
none	

7.26 SPI Framework

RTOS-integrated SPI Framework.

7.26.1 Functions

- sf_spi_callback
- sf_spi_common_start
- sf_spi_common_wait

- sf_spi_common_finish
- sf_spi_bus_device_check
- sf_spi_chipselect_assert
- sf_spi_chipselect_deassert
- sf_spi_check_lower_lvl_driver_parameters
- sf_spi_reconfigure_device
- sf_spi_check_common_parameters
- SF_SPI_Open
- SF_SPI_Read
- SF_SPI_Write
- SF_SPI_WriteRead
- SF_SPI_Close
- SF_SPI_Lock
- SF_SPI_Unlock
- SF_SPI_VersionGet

7.26.2 Variables

- g_sf_spi_version
- g_sf_spi_on_sf_spi

7.26.3 Defines

- #define SF_SPI_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define SF_SPI_CODE_VERSION_MINOR

  Initial value: `(5U)`
- #define SF_SPI_ERROR_RETURN

  Initial value:`#define SSP_ERROR_RETURN((expression), (error), &g_module_name[0], &g_sf_spi_version)`

7.26.4 sf_spi_callback

```
sf_spi_callback ( spi_callback_args_t * parg )
```

7.26.4.1 Brief description

SPI SSP framework level callback.

7.26.4.2 Detailed description

Table 876:Parameters

Name	Direction	Description
pcb_arg	in	Pointer to callback parameters

Table 877:Return values

Name	Description
void	

7.26.4.3 Function steps

- Event occurs wake up the suspended thread.
- < Points location for event flag for reception
- Set flag to trigger waiting thread.

7.26.5 sf_spi_common_start

```
ssp_err_t sf_spi_common_start ( sf_spi_instance_ctrl_t  *const p_ctrl ,
  uint32_t const timeout )
```

7.26.5.1 Brief description

Common SPI transfer start function. Used in all framework read write calls. This function checks whether there is any need for reconfiguration and if yes then it checks whether device is supported by the bus. If bus supports the device it acquires the mutex, reconfigures the bus and then enables the slave.

7.26.5.2 Detailed description

Table 878:Parameters

Name	Direction	Description
p_ctrl	in	Control handle for SPI framework driver context for a device
timeout	in	ThreadX timeout

Table 879:Return values

Name	Description
SSP_SUCCESS	SPI channel is successfully closed
SSP_ERR_INTERNAL	Internal error

See Common Error Codes and lower level driver function for other possible return codes. These driver functions are:

- close
- open

NOTE: This function is reentrant for any device.

7.26.5.3 Function steps

- See if SPI bus needs to be reconfigured.
- Different device is using bus than last time. Check that bus supports device.
- Get mutex for this bus.
- Need to reconfigure.
- Enable slave.

7.26.6 sf_spi_common_wait

```
ssp_err_t sf_spi_common_wait ( sf_spi_instance_ctrl_t  *const p_ctrl ,
  uint32_t const timeout )
```

7.26.6.1 Brief description

Common SPI wait. Waits for an operation to finish.

7.26.6.2 Detailed description

Table 880:Parameters

Name	Direction	Description
p_ctrl	in	Control handle for SPI framework driver context for a device
timeout	in	ThreadX timeout.

Table 881:Return values

Name	Description
SSP_SUCCESS	SPI channel is successfully closed.
SSP_ERR_INTERNAL	Internal error occurs.
SSP_ERR_TRANSFER_ABORTED	The data transfer was aborted.
SSP_ERR_UNDERFLOW	Read underflow occurs.
SSP_ERR_MODE_FAULT	Mode fault error occurs.
SSP_ERR_READ_OVERFLOW	Read overflow occurs.
SSP_ERR_PARITY	Parity error occurs.
SSP_ERR_OVERRUN	Overrun error occurs.

NOTE: This function is reentrant for any device.

7.26.7 sf_spi_common_finish

```
ssp_err_t sf_spi_common_finish ( sf_spi_instance_ctrl_t  *const p_ctrl )
```

7.26.7.1 Brief description

Common SPI finish. Release mutex and deassert chip select.

7.26.7.2 Detailed description

Table 882:Parameters

Name	Direction	Description
p_ctrl	in	Control handle for SPI framework driver context for a device

Table 883:Return values

Name	Description
SSP_SUCCESS	Chip select deactivated and mutex released.
SSP_ERR_INTERNAL	Internal error.

NOTE: This function is reentrant for any device.

7.26.8 sf_spi_bus_device_check

```
ssp_err_t sf_spi_bus_device_check ( sf_spi_instance_ctrl_t  *const p_ctrl )
```

7.26.8.1 Brief description

SPI device check, checks device compatibility on the bus.

7.26.8.2 Detailed description

Table 884:Parameters

Name	Direction	Description
p_ctrl	in	Control handle for SPI framework driver context for a device

Table 885:Return values

Name	Description
SSP_SUCCESS	Selected device supported by the SPI bus
SSP_ERR_UNSUPPORTED	Selected device not supported by the SPI bus

NOTE: This function is reentrant for any device.

7.26.9 sf_spi_chipselect_assert

```
sf_spi_chipselect_assert ( ioport_port_pin_t  chip_select ,
ioport_level_t  active_level )
```

7.26.9.1 Brief description

SPI SSP framework level chip select utility function.

7.26.9.2 Detailed description

Table 886:Parameters

Name	Direction	Description
chip_select	in	Chip select pin
active_level	in	Active high or active low

Table 887:Return values

Name	Description
void	

7.26.10 sf_spi_chipselect_deassert

```
sf_spi_chipselect_deassert ( ioport_port_pin_t  chip_select ,
ioport_level_t  active_level )
```

7.26.10.1 Brief description

SPI SSP framework level chip select utility function.

7.26.10.2 Detailed description

Table 888:Parameters

Name	Direction	Description
chip_select	in	Chip select pin
active_level	in	Active high or active low

Table 889:Return values

Name	Description
void	

7.26.11 sf_spi_check_lower_lvl_driver_parameters

```
sf_spi_check_lower_lvl_driver_parameters ( sf_spi_cfg_t  const *const p_cfg )
```

7.26.11.1 Brief description

Checks whether lower level SPI module and bus are defined.

7.26.11.2 Detailed description

Table 890:Parameters

Name	Direction	Description
p_cfg	in	Pointer to SPI framework Configuration Structure

Table 891:Return values

Name	Description
true	Lower level SPI driver, and SPI bus are configured
false	Lower level SPI driver, and SPI bus are not configured

7.26.12 sf_spi_reconfigure_device

```
ssp_err_t sf_spi_reconfigure_device ( sf_spi_instance_ctrl_t  *const p_ctrl )
```

7.26.12.1 Brief description

Assign a new device address to current device.

7.26.12.2 Detailed description

Table 892:Parameters

Name	Direction	Description
p_ctrl	in	Control handle for SPI framework context for a device

Table 893:Return values

Name	Description
SSP_SUCCESS	New device address assigned to current device

See Common Error Codes and lower level drivers for other possible return codes. These driver functions are:

- close
- open

7.26.12.3 Function steps

- < Get the current device.
- Close the device currently using the bus.
- Assign bus for the new device to use.

- Assign this device to current.

7.26.13 sf_spi_check_common_parameters

```
 sf_spi_check_common_parameters ( sf_spi_ctrl_t  *const p_ctrl ,   uint32_t
const length ,   uint32_t const timeout )
```

7.26.13.1 Brief description

Checks if SPI framework control block address, length and timeout are NULL.

7.26.13.2 Detailed description

Table 894:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to SPI framework control block
length	in	Number of bytes of data to be transferred
timeout	in	Timeout in ThreadX tick counts

Table 895:Return values

Name	Description
true	SPI framework control block address, length and timeout are not NULL.
false	SPI framework control block address, length and timeout are NULL.

7.26.14 SF_SPI_Open

```
ssp_err_t SF_SPI_Open ( sf_spi_ctrl_t  *const p_api_ctrl ,  sf_spi_cfg_t  const
*const p_cfg )
```

7.26.14.1 Brief description

Initialize a SPI bus and open low level SPI driver.

7.26.14.2 Detailed description

Table 896:Return values

Name	Description
SSP_SUCCESS	SPI channel is successfully opened.
SSP_ERR_ASSERTION	One of the following parameters is NULL: p_api_ctrl, p_cfg, Pointer to Open, Close, Read, Write, or Writeread API interfaces, p_cfg->p_bus or p_cfg->p_lower_lvl_cfg.
SSP_ERR_INTERNAL	Internal error occurred.

See Common Error Codes and lower level driver function for other possible return codes. This driver function is

- open

NOTE: This function is reentrant for any channel.

NOTE: Control block must be cleared by caller before calling this function.

7.26.14.3 Function steps

- Copy bus to control
- Copy chip_select to control
- Copy chip_select level to control
- Initialize bus lock to false
- Set framework level callback function.
- Save context for use in ISRs.
- Use bus channel in device open.
- Open only for the first device on the bus. If the device requires a bus reconfiguration then that will happen when a later read/write occurs.
- Create mutex for this bus.
- Create Event flag.
- Open the low level SPI module.
- Assign last used device ctrl on this bus.

- Save device configuration for reconfiguration.
- Set device state as Opened.
- Increment device count.
- Initialize chip select.

7.26.15 SF_SPI_Read

```
ssp_err_t SF_SPI_Read ( sf_spi_ctrl_t  *const p_api_ctrl ,  void *const p_dest ,
  uint32_t const length ,  spi_bit_width_t  const bit_width ,  uint32_t
const timeout )
```

7.26.15.1 Brief description

Starts the transfer process and receives data from SPI device.

7.26.15.2 Detailed description

Table 897:Return values

Name	Description
SSP_SUCCESS	Data read completed successfully.
SSP_ERR_ASSERTION	One of the following parameters is NULL: p_api_ctrl, p_dest, length, timeout.
SSP_ERR_NOT_OPEN	Device not opened.

See Common Error Codes and lower level driver function for other possible return codes. This driver function is:

- read

7.26.15.3 Function steps

- Check whether device is open.
- Start transfer process - check lock, check reconfiguration, check bus compatibility, enable chip select.
- Perform write read.
- Finish transfer.

7.26.16 SF_SPI_Write

```
ssp_err_t SF_SPI_Write ( sf_spi_ctrl_t  *const p_api_ctrl ,  void *const p_src ,
  uint32_t const length ,  spi_bit_width_t  const bit_width ,  uint32_t
const timeout )
```

7.26.16.1 Brief description

Starts the transfer process and writes data to SPI device.

7.26.16.2 Detailed description

Table 898:Return values

Name	Description
SSP_SUCCESS	Data write completed successfully.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_api_ctrl, p_src, length, timeout.
SSP_ERR_NOT_OPEN	Device not opened.

See Common Error Codes and lower level driver function for other possible return codes. This driver function is:

- write

7.26.16.3 Function steps

- Check that device is open.
- Start transfer process - check lock, check reconfiguration, check bus compatibility, enable chip select.
- Perform write read.
- Finish transfer.

7.26.17 SF_SPI_WriteRead

```
ssp_err_t SF_SPI_WriteRead ( sf_spi_ctrl_t  *const p_api_ctrl ,  void
*const p_src ,  void *const p_dest ,  uint32_t const length ,  spi_bit_width_t
const bit_width ,  uint32_t const timeout )
```

7.26.17.1 Brief description

Simultaneously transmit data to SPI device while receiving data from SPI device(full duplex).

7.26.17.2 Detailed description

Table 899:Return values

Name	Description
SSP_SUCCESS	Data write completed successfully.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_api_ctrl, p_src, p_dest, length, timeout.
SSP_ERR_NOT_OPEN	Device not opened.

See Common Error Codes and lower level driver function for other possible return codes. This driver function is:

- writeRead

7.26.17.3 Function steps

- Check whether device is open.
- Start transfer process - check lock, check reconfiguration, check bus compatibility, enable chip select.
- Perform write read.
- Finish transfer.

7.26.18 SF_SPI_Close

```
ssp_err_t SF_SPI_Close ( sf_spi_ctrl_t  *const p_api_ctrl )
```

7.26.18.1 Brief description

Disable the SPI device designated by the control handle and close the RTOS services used by the bus if no devices are connected to the bus.

7.26.18.2 Detailed description

Table 900:Return values

Name	Description
SSP_SUCCESS	SPI channel is successfully closed.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.

Table 900:Return values (Continued)

Name	Description
SSP_ERR_NOT_OPEN	Device not opened.
SSP_ERR_IN_USE	In-use error.
SSP_ERR_INTERNAL	Internal error occurred.

See Common Error Codes and lower level driver function for other possible return codes. This driver function is:

- close

NOTE: This function is reentrant for any device.

7.26.18.3 Function steps

- Check whether device is open.
- Check whether the device must be closed through close. Close only if this is the current device. Otherwise the device might have been closed during reconfiguration or might not have been opened through open().
- Get mutex since this will access hardware registers.
- Close low level driver.
- Release mutex.
- Decrement device count.
- < Check is there any device still using the bus.
- Delete RTOS services used by bus.
- Set device to closed state

7.26.19 SF_SPI_Lock

```
ssp_err_t SF_SPI_Lock ( sf_spi_ctrl_t  *const p_api_ctrl )
```

7.26.19.1 Brief description

Lock the bus for a device.

7.26.19.2 Detailed description

Table 901:Return values

Name	Description
SSP_SUCCESS	SPI channel is successfully closed.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Device not opened.
SSP_ERR_IN_USE	In-use error.

NOTE: This function is reentrant for any device.

7.26.19.3 Function steps

- Check whether device is open.
- Check whether device is already locked.
- Get the mutex for this device.
- Set lock flag.
- Enable slave.

7.26.20 SF_SPI_Unlock

```
ssp_err_t SF_SPI_Unlock ( sf_spi_ctrl_t  *const p_api_ctrl )
```

7.26.20.1 Brief description

Unlock the bus for a particular device and make the bus usable for other devices.

7.26.20.2 Detailed description

Table 902:Return values

Name	Description
SSP_SUCCESS	SPI channel is successfully closed.

Table 902:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Device not opened.
SSP_ERR_IN_USE	In-use error.

NOTE: This function is reentrant for any device.

7.26.20.3 Function steps

- Check whether device is open.
- Clear lock flag.
- Release the mutex so that others can use the bus.
- Disable slave.

7.26.21 SF_SPI_VersionGet

```
ssp_err_t SF_SPI_VersionGet ( ssp_version_t  *const p_version )
```

7.26.21.1 Brief description

Get the version information of the framework.

7.26.21.2 Detailed description

Table 903:Return values

Name	Description
SSP_ERR_ASSERTION	p_version is NULL.
SSP_SUCCESS	Successful return.

7.26.21.3 Function steps

- Checks error. Further parameter checking can be done at the driver layer.

7.26.22 g_sf_spi_version

ssp_version_t::g_sf_spi_version

7.26.22.1 Detailed description

Version data structure used by error logger macro.

7.26.23 g_sf_spi_on_sf_spi

sf_spi_api_t::g_sf_spi_on_sf_spi

7.26.23.1 Initialized as

```
g_sf_spi_on_sf_spi=
{
    .open      = SF_SPI_Open,
    .read      = SF_SPI_Read,
    .write     = SF_SPI_Write,
    .writeRead = SF_SPI_WriteRead,
    .close     = SF_SPI_Close,
    .lock      = SF_SPI_Lock,
    .unlock    = SF_SPI_Unlock,
    .version   = SF_SPI_VersionGet
}
```

7.26.24 Extensions

7.26.24.1 sf_spi_instance_ctrl_t

sf_spi_instance_ctrl_t

Detailed description

SPI device context. DO NOT INITIALIZE. Initialization occurs when open is called.

Variables

- sf_spi_bus_t * p_bus

  Bus using this device (copy from cfg)
- ioport_port_pin_t chip_select

  Chip select for this device (copy from cfg)
- ioport_level_t chip_select_level_active

  Polarity of CS, active High or Low (copy from cfg)

- spi_cfg_t lower_lvl_cfg

  SPI peripheral configuration, use for bus reconfiguration.
- spi_ctrl_t * p_lower_lvl_ctrl

  SPI peripheral control block.
- sf_spi_dev_state_t dev_state

  Device status.
- bool locked

  Lock and unlock bus for a device.

7.27 Thread Monitor Framework

Framework module providing monitoring of threads.

Any misbehaving threads result in the device being reset. Both the WDT and IWDT HAL modules are supported by this framework module.

7.27.1 Interface Used

WDT Interface

7.27.2 Functions

- SF_THREAD_MONITOR_Thread
- SF_THREAD_MONITOR_Open
- SF_THREAD_MONITOR_Close
- SF_THREAD_MONITOR_ThreadRegister
- SF_THREAD_MONITOR_ThreadUnregister
- SF_THREAD_MONITOR_CountIncrement
- SF_THREAD_MONITOR_VersionGet

7.27.3 Defines

- #define SF_THREAD_MONITOR_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file
- #define SF_THREAD_MONITOR_CODE_VERSION_MINOR

  Initial value: `(5U)`

7.27.4 SF_THREAD_MONITOR_Thread

```
SF_THREAD_MONITOR_Thread (  ULONG thread_input )
```

7.27.4.1 Brief description

The SF_THREAD_MONITOR_thread() is the main thread monitor thread which runs periodically.

7.27.4.2 Detailed description

The period is determined from the timeout period of the Watchdog hardware. This thread's period should be half of that of the watchdog hardware so the watchdog is refreshed at 50% of the count value.

Table 904:Parameters

Name	Direction	Description
thread_input	in	This is the address of the control block for the thread monitor module. This allows this thread to be created multiple times with the data used and hardware interfaced with governed by the control structure.

As this is a ThreadX thread this function does not return.

NOTE: This function is not reentrant.

NOTE: If there are no threads registered to be monitored the monitoring thread will refresh/tickle/kick the watchdog and so prevent the watchdog from reseting the device.

7.27.5 SF_THREAD_MONITOR_Open

```
ssp_err_t SF_THREAD_MONITOR_Open ( sf_thread_monitor_ctrl_t  *const p_api_ctrl ,
 sf_thread_monitor_cfg_t  const *const p_cfg )
```

7.27.5.1 Brief description

Calls the driver .open function in the p_lower_lvl_wdt parameter.

7.27.5.2 Detailed description

After successively opening and configuring the HAL driver, the SF_THREAD_MONITOR_Open function starts the thread monitoring thread. This thread is responsible for checking thread execution through thread count variables and for refreshing the implemented watchdog timer peripheral.

Table 905:Return values

Name	Description
SSP_SUCCESS	Initialization was successful and watchdog monitoring thread has started.
SSP_ERR_ASSERTION	p_ctrl, p_cfg, p_cfg->p_lower_lvl_wdt pointers and/or p_cfg->p_lower_lvl_wdt are NULL.
SSP_ERR_IN_USE	Thread monitor has already been opened.
SSP_ERR_INVALID_MODE	Low level watchdog peripheral returned an error when opened.
SSP_ERR_UNSUPPORTED	Data structure could not be allocated.
SSP_ERR_INVALID_ARGUMENT	One or more configuration options is invalid for the low level driver.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes.

NOTE: This function is not reentrant.

NOTE: If there are no threads registered to be monitored the monitoring thread refreshes the watchdog and prevents the watchdog from reseting the device.

NOTE: The Thread Monitor can be closed with SF_THREAD_MONITOR_Close and then SF_THREAD_MONITOR_Open can be called again. However, if the underlying watchdog timer hardware such as WDT or IWDT cannot be closed or stopped then there are likely to be undesirable results.

7.27.6 SF_THREAD_MONITOR_Close

```
ssp_err_t SF_THREAD_MONITOR_Close ( sf_thread_monitor_ctrl_t
*const p_api_ctrl )
```

7.27.6.1 Brief description

Stop the thread monitoring thread. All threads are unregistered and no longer monitored.

7.27.6.2 Detailed description

Table 906:Return values

Name	Description
SSP_SUCCESS	Close was successful and watchdog monitoring thread has stopped.
SSP_ERR_ASSERTION	p_ctrl pointer is NULL.
SSP_ERR_NOT_OPEN	SF_THREAD_MONITOR_Open has either not been called or it was not called successfully.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes.

NOTE: This function is not reentrant.

NOTE: This function DOES NOT stop the watchdog timer (e.g. WDT or IWDT). It does however stop the thread which refreshes the these timers. To prevent the hardware watchdog from reseting the device the watchdog must be refreshed elsewhere.

NOTE: The Thread Monitor can be closed with SF_THREAD_MONITOR_Close. However, if the underlying watchdog timer hardware such as WDT or IWDT cannot be closed or stopped then there are likely to be undesirable results.

7.27.7 SF_THREAD_MONITOR_ThreadRegister

```
ssp_err_t SF_THREAD_MONITOR_ThreadRegister ( sf_thread_monitor_ctrl_t
*const p_api_ctrl ,  sf_thread_monitor_counter_min_max_t  const
* p_counter_min_max )
```

7.27.7.1 Brief description

Register a thread to be monitored by the watchdog monitoring thread.

7.27.7.2 Detailed description

This thread must supply the minimum and maximum expected values for the thread. The mininimum and maximum values can be determined by using monitoring mode.

Table 907:Return values

Name	Description
SSP_SUCCESS	Thread successfully registered.
SSP_ERR_ASSERTION	p_ctrl or p_counter_min_max pointers are NULL.
SSP_ERR_INSUFFICIENT_SPACE	Not enough entries in the threads to be monitored array to add this thread. Increase the value of THREAD_MONITOR_CFG_MAX_NUMBER_OF_THREADS in sf_thread_moinitor_cfg.h
SSP_ERR_NOT_OPEN	SF_THREAD_MONITOR_Open has either not been called or it was not called successfully.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes.

NOTE: This function is reentrant.

7.27.8 SF_THREAD_MONITOR_ThreadUnregister

```
ssp_err_t SF_THREAD_MONITOR_ThreadUnregister ( sf_thread_monitor_ctrl_t
*const p_api_ctrl )
```

7.27.8.1 Brief description

Remove a thread from being monitored by the Watchdog monitoring thread.

7.27.8.2 Detailed description

Table 908:Return values

Name	Description
SSP_SUCCESS	Thread successfully unregistered.
SSP_ERR_ASSERTION	p_ctrl pointer is NULL.
SSP_ERR_NOT_OPEN	SF_THREAD_MONITOR_Open has either not been called or it was not called successfully.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes.

NOTE: This function is reentrant.

7.27.9 SF_THREAD_MONITOR_CountIncrement

```
ssp_err_t SF_THREAD_MONITOR_CountIncrement ( sf_thread_monitor_ctrl_t
*const p_api_ctrl )
```

7.27.9.1 Brief description

Safely increment the counter of a thread. A mutex is used to ensure this increment occurs without corruption.

7.27.9.2 Detailed description

Table 909:Return values

Name	Description
SSP_SUCCESS	Thread counter successfully incremented.
SSP_ERR_ASSERTION	p_ctrl pointer is NULL.
SSP_ERR_INSUFFICIENT_SPACE	Not enough entries in the threads to be monitored array to add this thread.

Table 909:Return values (Continued)

Name	Description
SSP_ERR_NOT_OPEN	SF_THREAD_MONITOR_Open has either not been called or it was not called successfully.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or HAL driver for other possible return codes or causes.

NOTE: This function is reentrant.

7.27.10 SF_THREAD_MONITOR_VersionGet

```
ssp_err_t SF_THREAD_MONITOR_VersionGet ( ssp_version_t  *const p_version )
```

7.27.10.1 Brief description

Get version and store it in provided pointer p_version. Implements versionGet.

7.27.10.2 Detailed description

Table 910:Return values

Name	Description
SSP_SUCCESS	Version returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was null.

7.27.11 Extensions

7.27.11.1 sf_thread_monitor_instance_ctrl_t

sf_thread_monitor_instance_ctrl_t

Detailed description

Thread monitor control block. DO NOT INITIALIZE. Initialization occurs when open is called.

Variables

- uint32_t open

  Used by driver to check if the control structure is valid
- wdt_instance_t const * p_lower_lvl_wdt

  Pointer to interface structure for the watchdog peripheral
- uint32_t timeout_period_msec

  Time in milliseconds of the watchdog timeout period. Used to calculate the period of the monitoring thread.
- uint32_t timeout_period_watchdog_clocks

  Maximum count value of the watchdog. Used to synchronise to the count.
- bool profiling_mode_enabled

  Used by the driver to check if profiling mode is enabled.
- TX_MUTEX mutex

  Mutex to protect access to the thread counters.
- uint32_t profiling_mode_check

  Value used to verify profiling mode is enabled when prfiling_mode_enabled == true.
- sf_thread_monitor_thread_counter_t thread_counters[THREAD_MONITOR_CFG_MAX_NUMBER_OF_THREADS]

  Data storage for the thread counter information.
- TX_THREAD thread

  Thread monitor thread.
- void const * p_extend

  Extended configuration data.
- uint8_t stack[THREAD_MONITOR_THREAD_STACK_SIZE]

  Stack for thread monitor thread.

7.28 CTSU Framework

RTOS-integrated CTSU Framework.

7.28.1 Functions

- SF_TOUCH_CTSU_Open
- SF_TOUCH_CTSU_Read
- SF_TOUCH_CTSU_Close
- SF_TOUCH_CTSU_VersionGet

7.28.2 Defines

- #define SF_TOUCH_CTSU_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define SF_TOUCH_CTSU_CODE_VERSION_MINOR

  Initial value: `(3U)`

7.28.3 SF_TOUCH_CTSU_Open

```
ssp_err_t SF_TOUCH_CTSU_Open ( sf_touch_ctsu_ctrl_t  *const p_api_ctrl ,
 sf_touch_ctsu_cfg_t  const *const p_cfg )
```

7.28.3.1 Brief description

Configures Touch CTSU framework.

7.28.3.2 Detailed description

The SF_TOUCH_CTSU_Open function checks to see if it was already initialized and if not, then initializes the CTSU HAL layer, saves the callback for the calling layer, and creates the internal thread. If the framework was already initialized by the same widget layer, then it will return an error. If the calling widget is a first time caller, then the callback and context are stored internally.

Table 911:Return values

Name	Description
SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_ctrl, p_api, or p_cfg. See HAL driver for other possible causes.
SSP_ERR_IN_USE	The framework has already been initialized by this particular widget.
SSP_ERR_INTERNAL	The hardware initialization or thread/semaphore failed.
SSP_ERR_INVALID_ARGUMENT	An input parameter was outside the supported bounds.

NOTE: Refer to the lower level driver for additional return value descriptions.

See Common Error Codes or HAL driver for other possible return codes or causes.

7.28.3.3 Function steps

- Initialize the elements to NULL if this is the first time the framework is being opened
- If the framework has already been opened by at least one widget layer; this is verified by checking to see if the callback from the widget layer is the same as any of the already stored ones
- Check if the table already contains the callback and the passed callback is not NULL. If true, then the same widget layer is calling the initialization function again in error. If the passed callback is NULL, this means that the widget layer does not want a callback.
- Save the widget callback and context into the framework callback array
- Verify there was room to store the callback and/or context
- If the framework has not been opened by at least one widget layer then initialize the lower layer and create the internal thread
- Create semaphore to control access to the lower level driver
- Delete the semaphore_ctsu_scan_complete semaphore
- Return error
- Assign the instance for the lower level into the control structure
- Create a local version of the CTSU HAL driver so that we can modify the HAL callback to use the callback defined in this file.
- Change the callback function to private function in this file.
- Open the CTSU for operation
- Save configuration and lower level API info
- Close the lower level driver to clean up hardware locks and memory
- Clear the callback
- Clear the callback
- Delete the semaphores
- Return the error
- Create internal CTSU thread to drive scans.
- Delete the semaphore
- If this framework has already been initialized by a previous call, then update the p_ctrl for the current calling layer
- Save the locally saved pointer to the lower level control to the current calling frameworks lower level control so that the current framework can close the lower level
- Save the callback index
- Mark control block open so other tasks know it is valid
- Mark the local flag as initialized

7.28.4 SF_TOUCH_CTSU_Read

```
ssp_err_t SF_TOUCH_CTSU_Read ( sf_touch_ctsu_ctrl_t  *const p_api_ctrl ,  void
* p_dest ,  ctsu_read_t  read_options ,  ctsu_channel_pair_t const * channels ,
  const uint16_t count )
```

7.28.4.1 Brief description

Reads data from Touch CTSU framework.

7.28.4.2 Detailed description

The SF_TOUCH_CTSU_Read function allows the user to read back data from the lower level layer. Access to the lower level data is controlled via semaphore to prevent data reading when a scan or processing is ongoing.

Table 912:Return values

Name	Description
SSP_SUCCESS	Read completed successfully.
SSP_ERR_ASSERTION	p_ctrl or p_ctrl->p_api is NULL.
SSP_ERR_NOT_OPEN	Driver control block not valid. Call SF_TOUCH_CTSU_Open to configure.
SSP_ERR_UNSUPPORTED	This function is not supported by the HAL driver (p_ctrl->p_lower_lvl_api->read is NULL).

See Common Error Codes or HAL driver for other possible return codes or causes.

7.28.4.3 Function steps

- Hold the TX status
- Wait for a scan to complete if the CTSU is scanning.
- Read the data from the lower layer driver
- Return the semaphore
- Read the converted data from the lower layer driver. Reading this data does not require a semaphore since this data is not actively updated by an ongoing scan.
- If there is a ThreadX error, return it, else return the lower level driver return code

7.28.5 SF_TOUCH_CTSU_Close

```
ssp_err_t SF_TOUCH_CTSU_Close ( sf_touch_ctsu_ctrl_t  *const p_api_ctrl )
```

7.28.5.1 Brief description

The close function checks to see if there are any other registered callbacks for other widgets. If there are then this function simply sets the callback for this widget to NULL and returns. If there are no other registered callbacks then this function terminates the internal scanning thread and calls the lower level close function.

7.28.5.2 Detailed description

Table 913:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_ctrl or p_ctrl->p_api is NULL.
SSP_ERR_NOT_OPEN	Driver control block not valid. Call SF_TOUCH_CTSU_Open to configure.
SSP_ERR_INTERNAL	ThreadX call failed.
SSP_ERR_UNSUPPORTED	This function is not supported by the HAL driver (p_ctrl->p_lower_lvl_api->close is NULL).

See Common Error Codes or HAL driver for other possible return codes or causes.

7.28.5.3 Function steps

- Hold the status
- Set the callback for this widget to NULL
- Clear information from control block so other functions know this block is closed
- Check if the table contains any non-NULL callbacks which would indicate that some other widget layer is also using this framework. If no non-NULL callbacks are in the table then the internal thread can be terminated and the HAL closed
- Return success
- Grab the semaphore first before deleting the Thread. May be unnecessary depending on TX Implementation
- Terminate the internal scanning thread first
- Delete the internal scanning thread once it has been terminated
- Close the lower layer driver
- Delete the semaphore
- Clear the local flag to indicate that the framework module is closed
- If there is a ThreadX error, return it, else return the lower level driver return code

7.28.6 SF_TOUCH_CTSU_VersionGet

```
ssp_err_t SF_TOUCH_CTSU_VersionGet ( ssp_version_t  *const p_version )
```

7.28.6.1 Brief description

Gets version and stores it in provided pointer p_version.

7.28.6.2 Detailed description

Table 914:Return values

Name	Description
SSP_SUCCESS	Version returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was null.

7.28.7 Extensions

7.28.7.1 sf_touch_ctsu_instance_ctrl_t

sf_touch_ctsu_instance_ctrl_t

Detailed description

Instance control block. DO NOT INITIALIZE. Initialization occurs when open is called

Variables

- uint32_t open

  Used by driver to check if control block is valid.
- uint16_t update_hz

  The frequency to run the scans at.
- ctsu_instance_t * p_lower_lvl_instance

  Pointer to CTSU instance.
- uint32_t cb_index

  Indicates the index in callback registry table.

7.29 CTSU Button Framework

RTOS-integrated CTSU Button Framework.

7.29.1 Functions

- SF_TOUCH_CTSU_Button_Open
- SF_TOUCH_CTSU_Button_Close
- SF_TOUCH_CTSU_Button_Enable
- SF_TOUCH_CTSU_Button_Disable
- SF_TOUCH_CTSU_Button_VersionGet

7.29.2 Defines

- #define SF_TOUCH_CTSU_BUTTON_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file
- #define SF_TOUCH_CTSU_BUTTON_CODE_VERSION_MINOR

  Initial value: `(2U)`
- #define SF_TOUCH_CTSU_BUTTON_MAX_CHANNELS

  Initial value:`CTSU_MAX_CHANNELS`

  Maximum supported button channel
- #define SF_TOUCH_CTSU_BUTTON_BIT_MASK_ARRAY_SIZE

  Initial value:`6`

7.29.3 SF_TOUCH_CTSU_Button_Open

```
ssp_err_t SF_TOUCH_CTSU_Button_Open ( sf_touch_ctsu_button_ctrl_t
*const p_api_ctrl ,  sf_touch_ctsu_button_cfg_t  const *const p_cfg )
```

7.29.3.1 Brief description

Configures Touch CTSU Button framework.

7.29.3.2 Detailed description

The SF_TOUCH_CTSU_Button_Open function initializes all the buttons in the configuration structure and also initializes the lower level framework module.

Table 915:Return values

Name	Description
SSP_SUCCESS	Initialization was successful.

Table 915:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_ctrl, p_api, or p_cfg. See lower driver for other possible causes.
SSP_ERR_IN_USE	The framework has already been initialized by this particular widget.

See HAL driver for other possible return codes or causes.

7.29.4 SF_TOUCH_CTSU_Button_Close

```
ssp_err_t SF_TOUCH_CTSU_Button_Close ( sf_touch_ctsu_button_ctrl_t
* p_api_ctrl )
```

7.29.4.1 Brief description

Closes the Touch CTSU Button framework.

7.29.4.2 Detailed description

The SF_TOUCH_CTSU_Button_Close closes the button framework.

Table 916:Return values

Name	Description
SSP_SUCCESS	Close was successful.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_ctrl, p_api, or p_cfg.
SSP_ERR_NOT_OPEN	Driver control block not valid. Call SF_TOUCH_CTSU_Button_Open to configure.
SSP_ERR_INTERNAL	The hardware initialization or thread/mutex/semaphore failed.

See HAL driver for other possible return codes or causes.

7.29.4.3 Function steps

- Close the lower level module

- Mark the module as uninitialized

7.29.5 SF_TOUCH_CTSU_Button_Enable

```
ssp_err_t SF_TOUCH_CTSU_Button_Enable ( sf_touch_ctsu_button_ctrl_t
* p_api_ctrl ,  sf_touch_ctsu_button_id  id )
```

7.29.5.1 Brief description

Enables events for a specific button.

7.29.5.2 Detailed description

The SF_TOUCH_CTSU_Button_Enable function resets a buttons state and enables events to be generated from actions on that button.

Table 917:Return values

Name	Description
SSP_SUCCESS	Button Enable was successful.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_ctrl.
SSP_ERR_NOT_OPEN	Driver control block not valid. Call SF_TOUCH_CTSU_Button_Open to configure.
SSP_ERR_INTERNAL	The hardware initialization or thread/mutex/semaphore failed.

See HAL driver for other possible return codes or causes.

7.29.5.3 Function steps

- Iterate through the list of button identifiers to find the button and then enable the button

7.29.6 SF_TOUCH_CTSU_Button_Disable

```
ssp_err_t SF_TOUCH_CTSU_Button_Disable ( sf_touch_ctsu_button_ctrl_t
* p_api_ctrl ,  sf_touch_ctsu_button_id  id )
```

7.29.6.1 Brief description

Disables a specific button.

7.29.6.2 Detailed description

The SF_TOUCH_CTSU_Button_Disable function disables events from a particular button.

Table 918:Return values

Name	Description
SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_ctrl.
SSP_ERR_NOT_OPEN	Driver control block not valid. Call SF_TOUCH_CTSU_Button_Open to configure.

See HAL driver for other possible return codes or causes.

7.29.6.3 Function steps

- Iterate through the list of button identifiers to find the button and then disable the button

7.29.7 SF_TOUCH_CTSU_Button_VersionGet

```
ssp_err_t SF_TOUCH_CTSU_Button_VersionGet ( ssp_version_t  *const p_version )
```

7.29.7.1 Brief description

Gets version and stores it in provided pointer p_version.

7.29.7.2 Detailed description

Table 919:Return values

Name	Description
SSP_SUCCESS	Version returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was null.

7.29.8 Extensions

7.29.8.1 sf_touch_ctsu_button_instance_ctrl_t

sf_touch_ctsu_button_instance_ctrl_t

Detailed description

Control structure for this Interface

Variables

- uint32_t opened

  Save the initialization state.
- sf_touch_ctsu_button_hdl * p_button_hdl

  Pointer to the button handle.
- uint32_t button_count

  Button Count.
- sf_touch_ctsu_instance_t const * p_lower_lvl_ctsu

  Pointer to the lower level instance.
- void const * p_context

  Placeholder for user data.
- void(* p_callback)(*p_args)

  Callback function.

7.30 CTSU Slider Framework

RTOS-integrated CTSU Slider Framework.

7.30.1 Functions

- SF_TOUCH_CTSU_Slider_VersionGet
- SF_TOUCH_CTSU_Slider_Open
- SF_TOUCH_CTSU_Slider_Close
- SF_TOUCH_CTSU_Slider_Enable
- SF_TOUCH_CTSU_Slider_Disable

7.30.2 Defines

- #define SF_TOUCH_CTSU_SLIDER_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file
- #define SF_TOUCH_CTSU_SLIDER_CODE_VERSION_MINOR

  Initial value: `(3U)`
- #define SF_TOUCH_CTSU_SLIDER_MAX_CHANNELS

  Initial value:`CTSU_MAX_CHANNELS`

  Maximum supported CTSU channels
- #define SF_TOUCH_CTSU_SLIDER_MAX_CHANNELS_PER_SLIDER

  Initial value:`10`

  Maximum supported channels per slider based on Workbench6 as of February 2016
- #define SF_TOUCH_CTSU_SLIDER_BIT_MASK_ARRAY_SIZE

  Initial value:`6`

7.30.3 SF_TOUCH_CTSU_Slider_VersionGet

```
ssp_err_t SF_TOUCH_CTSU_Slider_VersionGet ( ssp_version_t  *const p_version )
```

7.30.3.1 Brief description

Gets version and stores it in provided pointer p_version.

7.30.3.2 Detailed description

Table 920:Return values

Name	Description
SSP_SUCCESS	Version returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was null.

7.30.4 SF_TOUCH_CTSU_Slider_Open

```
ssp_err_t SF_TOUCH_CTSU_Slider_Open ( sf_touch_ctsu_slider_ctrl_t
*const p_api_ctrl ,  sf_touch_ctsu_slider_cfg_t  const *const p_cfg )
```

7.30.4.1 Brief description

Configures Touch CTSU Slider framework.

7.30.4.2 Detailed description

The SF_TOUCH_CTSU_Slider_Open function initializes all the sliders/wheels in the configuration structure and initializes the lower level framework module.

Table 921:Return values

Name	Description
SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_ctrl, p_api, or p_cfg. See lower driver for other possible causes.
SSP_ERR_IN_USE	The framework has already been initialized by this particular widget.

See Common Error Codes or HAL driver for other possible return codes or causes.

7.30.4.3 Function steps

- Discover active channels

7.30.5 SF_TOUCH_CTSU_Slider_Close

```
ssp_err_t SF_TOUCH_CTSU_Slider_Close ( sf_touch_ctsu_slider_ctrl_t
*const p_api_ctrl )
```

7.30.5.1 Brief description

Closes the Touch CTSU Slider framework.

7.30.5.2 Detailed description

The SF_TOUCH_CTSU_Slider_Close

Table 922:Return values

Name	Description
SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_ctrl, p_api, or p_cfg. See lower driver for other possible causes.
SSP_ERR_NOT_OPEN	The framework has not been initialized by this particular widget.
SSP_ERR_UNSUPPORTED	p_ctrl->p_lower_lvl_ctsu->p_api or p_ctrl->p_lower_lvl_ctsu->p_api->close was NULL

See Common Error Codes or HAL driver for other possible return codes or causes.

7.30.5.3 Function steps

- Close the lower level module
- Mark the module as uninitialized

7.30.6 SF_TOUCH_CTSU_Slider_Enable

```
ssp_err_t SF_TOUCH_CTSU_Slider_Enable ( sf_touch_ctsu_slider_ctrl_t
*const p_api_ctrl ,  sf_touch_ctsu_slider_id_t  id )
```

7.30.6.1 Brief description

The SF_TOUCH_CTSU_Slider_Enable.

7.30.6.2 Detailed description

Table 923:Return values

Name	Description
SSP_SUCCESS	Enable was successful.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_ctrl, p_api, or p_cfg. See lower driver for other possible causes.

Table 923:Return values (Continued)

Name	Description
SSP_ERR_NOT_OPEN	The framework has not been initialized by this particular widget.
SSP_ERR_INVALID_ARGUMENT	Invalid slider identifier.

See Common Error Codes or HAL driver for other possible return codes or causes.

7.30.6.3 Function steps

- Iterate through the list of slider identifiers to find the slider and then enable the slider

7.30.7 SF_TOUCH_CTSU_Slider_Disable

```
ssp_err_t SF_TOUCH_CTSU_Slider_Disable ( sf_touch_ctsu_slider_ctrl_t
*const p_api_ctrl ,  sf_touch_ctsu_slider_id_t  id )
```

7.30.7.1 Brief description

The SF_TOUCH_CTSU_Slider_Disable.

7.30.7.2 Detailed description

Table 924:Return values

Name	Description
SSP_SUCCESS	Disable was successful.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_ctrl, p_api, or p_cfg. See lower driver for other possible causes.
SSP_ERR_NOT_OPEN	The framework has not been initialized by this particular widget.
SSP_ERR_IN_USE	The framework has already been initialized by this particular widget.

See Common Error Codes or HAL driver for other possible return codes or causes.

7.30.7.3 Function steps

- Iterate through the list of slider identifiers to find the slider and then disable the slider

7.30.8 Extensions

7.30.8.1 sf_touch_ctsu_slider_instance_ctrl_t

sf_touch_ctsu_slider_instance_ctrl_t

Detailed description

Instance control structure for this module

Variables

- uint32_t opened

  Save the initialization state of the framework.
- uint32_t slider_count

  Slider Count.
- sf_touch_ctsu_instance_t const * p_lower_lvl_ctsu

  Pointer to the lower level instance.
- void const * p_context

  Placeholder for user data.
- void(* p_callback)(*p_args)

  Function to call when an event occurs

7.30.8.2 sf_slider_on_ctsu_cfg_t

sf_slider_on_ctsu_cfg_t

Detailed description

Individual slider structure to be populated by Workbench

Variables

- sf_slider_type_tconst type

  Linear or circular (wheel)
- uint32_t num_slider_channels
- ctsu_channel_pair_t const * p_slider_channels

  Define the channel/channel pairs which make up a slider.
- int32_t const * p_normalization

- int32_t * p_channel_average
- uint32_t * p_offset
- uint16_t *const p_slider_scount
- uint16_t *const p_slider_baseline
- int32_t *const p_slider_delta
- sf_touch_ctsu_slider_id_t id

  Unique identifier for slider
- int32_t max_slider_value

  Maximum slider value, must be greater than 0; Minimum is always 0
- const uint16_t slider_norm_max

  Individual slider settings that can be modified by the user. Should be the same as in st_sf_touch_ctsu_slider_h-dlTOUCH_SLIDER_CFG_NORM_MAX
- const int32_t slider_threshold

  Touch threshold. Ensure that value is greater than 0
- const int32_t channel_average_weight

  Weight of running average of counts for each channel
- const int32_t prev_sum_weight

  Weight of running average of previous sum in position calculation, must be greater than 0
- const int32_t cutoff

  Defines how far below prev_sum running average to get before "SF_TOUCH_SLIDER_STATE_RELEASED" detected

7.30.8.3 st_sf_touch_ctsu_slider_hdl

st_sf_touch_ctsu_slider_hdl

Detailed description

Handle for each slider

Needed for forward reference to sf_touch_ctsu_slider_hdl_t

Variables

- uint32_t open

  Indicate that slider has been opened
- sf_touch_ctsu_slider_id_t id

  Unique identifier for slider.
- sf_touch_ctsu_slider_state_t state

  Represents the current state of the slider.

- sf_slider_type_t type

  Linear or circular (wheel)
- uint32_t num_slider_channels
- ctsu_channel_pair_t const * p_slider_channels

  Define the channel/channel pairs which make up a slider.
- int32_t const * p_normalization
- int32_t * p_channel_average
- uint32_t * p_offset
- uint16_t * p_slider_scount
- uint16_t * p_slider_baseline
- int32_t * p_slider_delta
- uint32_t position

  Calculated position.
- int32_t prev_sum

  Used to store the running average of previous sum in position calculation
- int32_t max_slider_value

  Maximum slider value, must be greater than 0; Minimum is always 0
- ssp_err_t(* p_update)(*const hdl, const *const p_lower_lvl_touch_framework, uint32_t *p_pos, *const p_state)

  Function to calculate position of touch.
- uint64_t bit_mask[SF_TOUCH_CTSU_SLIDER_BIT_MASK_ARRAY_SIZE]

  Bit mask to be used in update function
- uint16_t slider_norm_max

  Individual slider settings that can be modified by the user. Should be the same as in sf_slider_on_ctsu_cfg_t-TOUCH_SLIDER_CFG_NORM_MAX
- int32_t slider_threshold

  Touch threshold. Ensure that value is greater than 0
- int32_t channel_average_weight

  Weight of running average of counts for each channel
- int32_t prev_sum_weight

  Weight of running average of previous sum in position calculation, must be greater than 0
- int32_t cutoff

  Defines how far below prev_sum running average to get before "SF_TOUCH_SLIDER_STATE_RELEASED" detected

7.31 I2C Touch Panel Framework

RTOS-integrated touch panel Framework implementation for external I2C touch chips.

7.31.1 Summary

This is a ThreadX touch panel framework implemented for external I2C touch controllers with IRQ pins used to notify the application when new data is available.

7.31.2 Functions

- SF_TOUCH_PANEL_I2C_Open
- SF_TOUCH_PANEL_I2C_Calibrate
- SF_TOUCH_PANEL_I2C_Start
- SF_TOUCH_PANEL_I2C_Stop
- SF_TOUCH_PANEL_I2C_Reset
- SF_TOUCH_PANEL_I2C_Close
- SF_TOUCH_PANEL_I2C_VersionGet

7.31.3 Defines

- #define SF_TOUCH_PANEL_I2C_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define SF_TOUCH_PANEL_I2C_CODE_VERSION_MINOR

  Initial value: `(5U)`
- #define SF_TOUCH_PANEL_I2C_RESET_PIN_UNUSED

  Initial value: `(0xFFFF)`
- #define SF_TOUCH_PANEL_I2C_STACK_SIZE

  Initial value: `(1024)`

  Stack size for I2C touch panel thread.

7.31.4 SF_TOUCH_PANEL_I2C_Open

```
ssp_err_t SF_TOUCH_PANEL_I2C_Open ( sf_touch_panel_ctrl_t  *const p_api_ctrl ,
 sf_touch_panel_cfg_t  const *const p_cfg )
```

7.31.4.1 Brief description

Implements open.

7.31.4.2 Detailed description

Table 925:Return values

Name	Description
SSP_SUCCESS	Touch panel thread created and lower level drivers opened successfully.
SSP_ERR_ASSERTION	A pointer parameter was NULL, or a lower level driver reported this error.
SSP_ERR_INTERNAL	The touch panel thread or event flags could not be created, or a lower level driver reported this error.
SSP_ERR_IN_USE	Mutex was not available, or a lower level driver reported this error.

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- open
- reset
- open

NOTE: This function is reentrant for any panel.

7.31.4.3 Function steps

- Store API's.
- Store user configurations in control block.
- Initialize previous event.
- Get mutex before calling lower layer, which will access HW registers.
- Open the lower level I2C driver.
- Reset the touch chip.
- Open the lower level external IRQ driver.
- Return mutex
- Create event flags for internal communication.
- Create main touch panel thread.
- Mark control block open so other tasks know it is valid

7.31.5 SF_TOUCH_PANEL_I2C_Calibrate

```
ssp_err_t SF_TOUCH_PANEL_I2C_Calibrate ( sf_touch_panel_ctrl_t
*const p_api_ctrl ,  sf_touch_panel_calibrate_t  const *const p_expected ,
 sf_touch_panel_payload_t  const *const p_actual ,   ULONG timeout )
```

7.31.5.1 Brief description

Implements calibrate.

7.31.5.2 Detailed description

Table 926:Return values

Name	Description
SSP_SUCCESS	Touch panel calibrated successfully.
SSP_ERR_ASSERTION	A pointer parameter was NULL, or a lower level driver reported this error.
SSP_ERR_CALIBRATE_FAILED	Actual touch value was not in expected range.
SSP_ERR_NOT_OPEN	Touch panel is not configured. Call SF_TOUCH_PANEL_I2C_Open.

See Common Error Codes or lower level drivers for other possible return codes.

NOTE: This function is reentrant for any panel.

7.31.5.3 Function steps

- Timeout not used in this implementation.
- Determine if touch message data matches expected value.

7.31.6 SF_TOUCH_PANEL_I2C_Start

```
ssp_err_t SF_TOUCH_PANEL_I2C_Start ( sf_touch_panel_ctrl_t  *const p_api_ctrl )
```

7.31.6.1 Brief description

Implements start.

7.31.6.2 Detailed description

Table 927:Return values

Name	Description
SSP_SUCCESS	Touch panel start flag posted to touch thread.
SSP_ERR_NOT_OPEN	Touch panel is not configured. Call SF_TOUCH_PANEL_I2C_Open.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or lower level drivers for other possible return codes.

NOTE: This function is reentrant for any panel.

7.31.6.3 Function steps

- Set start flag.

7.31.7 SF_TOUCH_PANEL_I2C_Stop

```
ssp_err_t SF_TOUCH_PANEL_I2C_Stop ( sf_touch_panel_ctrl_t  *const p_api_ctrl )
```

7.31.7.1 Brief description

Implements stop.

7.31.7.2 Detailed description

Table 928:Return values

Name	Description
SSP_SUCCESS	Touch panel will not respond to touch events.
SSP_ERR_ASSERTION	Parameter p_ctrl was NULL, or a lower level driver reported this error.
SSP_ERR_NOT_OPEN	Touch panel is not configured. Call SF_TOUCH_PANEL_I2C_Open.

Table 928:Return values (Continued)

Name	Description
SSP_ERR_INTERNAL	An internal ThreadX error has occurred.

See Common Error Codes or lower level drivers for other possible return codes.

NOTE: This function is reentrant for any panel.

7.31.7.3 Function steps

- Set stop flag.

7.31.8 SF_TOUCH_PANEL_I2C_Reset

```
ssp_err_t SF_TOUCH_PANEL_I2C_Reset ( sf_touch_panel_ctrl_t  *const p_api_ctrl )
```

7.31.8.1 Brief description

Implements reset.

7.31.8.2 Detailed description

Table 929:Return values

Name	Description
SSP_SUCCESS	Touch panel and lower level I2C driver were reset successfully.
SSP_ERR_ASSERTION	Parameter p_ctrl was NULL, or a lower level driver reported this error.
SSP_ERR_IN_USE	Mutex was not available, or a lower level driver reported this error.
SSP_ERR_NOT_OPEN	Touch panel is not configured. Use Open API to configure.

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- reset

NOTE: This function is reentrant for any panel.

7.31.8.3 Function steps

- Obtain mutex since this accesses shared resources.
- Call hardware specific reset function.
- Release mutex.

7.31.9 SF_TOUCH_PANEL_I2C_Close

```
ssp_err_t SF_TOUCH_PANEL_I2C_Close ( sf_touch_panel_ctrl_t  *const p_api_ctrl )
```

7.31.9.1 Brief description

Implements close.

7.31.9.2 Detailed description

Table 930:Return values

Name	Description
SSP_SUCCESS	Touch panel instance successfully closed.
SSP_ERR_ASSERTION	Parameter p_ctrl was NULL, or a lower level driver reported this error.
SSP_ERR_NOT_OPEN	Touch panel is not configured. Call SF_TOUCH_PANEL_I2C_Open.

See Common Error Codes or lower level drivers for other possible return codes. This function calls:

- close
- close
- bufferRelease

NOTE: This function is reentrant.

7.31.9.3 Function steps

- Close the lower level external IRQ driver.
- Close the lower level I2C driver.
- If I2C Abort returned from previous close, one more try to close I2C driver.
- Suspend internal thread.
- If a messaging framework buffer is taken but a touch message is not yet posted to the subscribers, release it.
- Delete RTOS services used
- Mark control block close

7.31.10 SF_TOUCH_PANEL_I2C_VersionGet

```
ssp_err_t SF_TOUCH_PANEL_I2C_VersionGet ( ssp_version_t  *const p_version )
```

7.31.10.1 Brief description

Implements versionGet.

7.31.10.2 Detailed description

Table 931:Return values

Name	Description
SSP_SUCCESS	Version returned successfully.
SSP_ERR_ASSERTION	Parameter p_version was null.

7.31.11 Extensions

7.31.11.1 sf_touch_panel_i2c_chip_t

sf_touch_panel_i2c_chip_t

Detailed description

Chip definition.

Variables

- ssp_err_t(* payloadGet)(*const p_ctrl, *const p_payload)

  Reads the touch chip and fills in the touch payload data. p_ctrlPointer to a structure allocated by user. This control structure is initialized in this function. p_payloadPointer to the payload to data structure. Touch data provided should be processed to logical pixel values.

- ssp_err_t(* reset)(*const p_ctrl)

  Resets the touch chip. p_ctrlPointer to a structure allocated by user. This control structure is initialized in this function.

7.31.11.2 sf_touch_panel_i2c_instance_ctrl_t

sf_touch_panel_i2c_instance_ctrl_t

Detailed description

Instance control block. DO NOT INITIALIZE. Initialization occurs when open is called

Variables

- uint32_t open

  Used by driver to check if control block is valid.

- uint16_t hsize_pixels

  Horizontal size of screen in pixels.

- uint16_t vsize_pixels

  Vertical size of screen in pixels.

- sf_message_instance_t const * p_message

  Pointer to messaging framework control block.

- uint8_t event_class_instance

  Event class instance number for posting touch event class messages.

- sf_touch_panel_payload_t * p_payload

  Pointer to buffer used to store payload.

- sf_touch_panel_payload_t last_payload

  Stores most recent payload put on queue for comparison.

- TX_MUTEX mutex

  Mutex used to protect access to shared resources.

- TX_EVENT_FLAGS_GROUP flags

  Event flags for internal communication.

- TX_THREAD thread

  Main touch panel thread.

- ioport_port_pin_t pin

  Reset pin.

- i2c_master_instance_t const * p_lower_lvl_i2c

  Lower level I2C.

- sf_external_irq_instance_t const * p_lower_lvl_irq

  Lower level external IRQ.

- uint8_t stack[SF_TOUCH_PANEL_I2C_STACK_SIZE]

  Stack for touch panel thread.

- sf_touch_panel_i2c_chip_t const * p_chip

  Chip specific functions and definitions.

- uint16_t update_hz

  The frequency to report repeat (SF_TOUCH_PANEL_EVENT_DOWN or SF_TOUCH_PANEL_EVENT_HOLD) touch events in Hertz. This will be converted to RTOS ticks in the driver and rounded up to the nearest integer value of RTOS ticks.

- uint16_t rotation_angle

  Touch coordinate rotation angle(0/90/180/270)

7.31.11.3 sf_touch_panel_i2c_cfg_t

sf_touch_panel_i2c_cfg_t

Detailed description

Configuration for RTOS touch panel driver.

Variables

- i2c_master_instance_t const * p_lower_lvl_i2c

  Pointer to lower level I2C.

- sf_external_irq_instance_t const * p_lower_lvl_irq

  Pointer to lower level external IRQ.

- ioport_port_pin_t pin

  Port pin connected to reset line on touch controller chip. Set to SF_TOUCH_PANEL_I2C_RESET_PIN_UNUSED if unused.

- sf_touch_panel_i2c_chip_t const *const p_chip

  Selected touch controller chip.

7.32 UART Framework Instance

RTOS-integrated Communications Framework UART implementation.

7.32.1 Functions

- SF_UART_COMMS_Open

- SF_UART_COMMS_Close
- SF_UART_COMMS_Read
- SF_UART_COMMS_Write
- SF_UART_COMMS_Lock
- SF_UART_COMMS_Unlock
- SF_UART_COMMS_VersionGet

7.32.2 Defines

- #define SF_UART_COMMS_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file
- #define SF_UART_COMMS_CODE_VERSION_MINOR

  Initial value: `(7U)`

7.32.3 SF_UART_COMMS_Open

```
ssp_err_t SF_UART_COMMS_Open ( sf_comms_ctrl_t  *const p_api_ctrl ,
 sf_comms_cfg_t  const *const p_cfg )
```

7.32.3.1 Brief description

Open the UART for communication.

7.32.3.2 Detailed description

Table 932:Parameters

Name	Direction	Description
p_api_ctrl	inout	Pointer to the UART control block
p_cfg	in	Pointer to the configuration structure

Table 933:Return values

Name	Description
SSP_SUCCESS	Channel opened successfully.

Table 933:Return values (Continued)

Name	Description
SSP_ERR_IN_USE	Channel already in use.
SSP_ERR_ASSERTION	Parameter check failed for one of the following: • Pointer p_api_ctrl is NULL. • Pointer p_cfg is NULL • Pointer p_cfg->p_extend is NULL • Pointer p_cfg_extend->p_lower_lvl_uart is NULL • Pointer p_cfg_extend->p_lower_lvl_uart->p_api->open is NULL • Pointer p_cfg_extend->p_lower_lvl_uart->p_cfg is NULL
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use thread mutex or failure create/use event flags or queue.

See Common Error Codes or functions called by this function for other possible codes. This function calls:

- open

NOTE: This function is reentrant for any channel. Handle must be cleared by caller before calling this function.

7.32.3.3 Function steps

- Checks error. Further parameter checking can be done at the driver layer.
- Initialize and start ThreadX resources
- Calls open function of UART HAL driver

7.32.4 SF_UART_COMMS_Close

```
ssp_err_t SF_UART_COMMS_Close ( sf_comms_ctrl_t  *const p_api_ctrl )
```

7.32.4.1 Brief description

Close the UART Channel and clean up the resources.

7.32.4.2 Detailed description

Table 934:Parameters

Name	Direction	Description
p_api_ctrl	in	Pointer to the UART control block

Table 935:Return values

Name	Description
SSP_SUCCESS	UART channel is successfully closed.
SSP_ERR_ASSERTION	Parameter check failed for one of the following: • Pointer p_api_ctrl is NULL. • Pointer p_ctrl->p_lower_lvl_uart is NULL • Pointer p_ctrl->p_lower_lvl_uart->p_api is NULL • Pointer p_ctrl->p_lower_lvl_uart->p_api->close is NULL
SSP_ERR_NOT_OPEN	Channel is not opened.
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to delete thread mutex , event flags or queue.

See Common Error Codes or functions called by this function for other possible codes. This function calls:

- close

NOTE: This function is reentrant for any channel.

7.32.4.3 Function steps

- Checks error. Further parameter checking can be done at the driver layer.
- Calls close function of UART HAL driver
- Release ThreadX resources

7.32.5 SF_UART_COMMS_Read

```
ssp_err_t SF_UART_COMMS_Read ( sf_comms_ctrl_t  *const p_api_ctrl ,   uint8_t
*const p_dest ,   uint32_t const bytes ,   UINT const timeout )
```

7.32.5.1 Brief description

Read user specified number of bytes into destination buffer pointer.

7.32.5.2 Detailed description

Table 936:Parameters

Name	Direction	Description
p_api_ctrl	in	Pointer to the UART control block
bytes	in	No.of bytes to be read
timeout	in	timeout for read
p_dest	out	Destination buffer

Table 937:Return values

Name	Description
SSP_SUCCESS	Data reception ends successfully.
SSP_ERR_NOT_OPEN	Channel is not opened.
SSP_ERR_HW_LOCKED	Channel is locked.
SSP_ERR_ASSERTION	Pointer to UART control block/pointer to destination address is NULL.
SSP_ERR_INVALID_MODE	Channel is used for non-UART mode.
SSP_ERR_INSUFFICIENT_DATA	Not enough data in receive circular buffer.
SSP_ERR_OVERFLOW	Hardware overflow.
SSP_ERR_FRAMING	Framing error.
SSP_ERR_PARITY	Parity error.

Table 937:Return values (Continued)

Name	Description
SSP_ERR_BREAK_DETECT	Break signal detected.
SSP_ERR_TIMEOUT	One of the following operation timed out. • 'Event flags get' timed out • 'Receive mutex get' timed out
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use thread mutex or failure create/use event flags or queue.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- read
- communicationAbort

7.32.5.3 Function steps

- Locks the UART reception hardware resource
- Unlock the UART reception hardware resource

7.32.6 SF_UART_COMMS_Write

```
ssp_err_t SF_UART_COMMS_Write ( sf_comms_ctrl_t  *const p_api_ctrl ,   uint8_t
const *const p_src ,   uint32_t const bytes ,   UINT const timeout )
```

7.32.6.1 Brief description

Write user specified number of bytes from the source buffer.

7.32.6.2 Detailed description

Table 938:Parameters

Name	Direction	Description
p_api_ctrl	in	Pointer to the UART control block
bytes	in	Number of bytes to to be written.

Table 938:Parameters (Continued)

Name	Direction	Description
timeout	in	Timeout for write. This timeout must be long enough for the write to complete. A timeout of 0 or TX_NO_WAIT results in a return value of SSP_ERR_TIMEOUT.
p_src	out	Source buffer

Table 939:Return values

Name	Description
SSP_SUCCESS	Data transmission finished successfully.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL. Pointer to source buffer is NULL.
SSP_ERR_NOT_OPEN	Channel is not opened.
SSP_ERR_INVALID_MODE	Channel is used for non-UART mode or illegal mode is set in handle.
SSP_ERR_INSUFFICIENT_SPACE	Not enough space in transmission circular buffer.
SSP_ERR_HW_LOCKED	Could not lock hardware.
SSP_ERR_TIMEOUT	'Transmit mutex get' timed out
SSP_ERR_INTERNAL	An internal ThreadX error has occurred. This is typically a failure to create/use thread mutex or failure create/use event flags or queue.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- write
- communicationAbort

7.32.6.3 Function steps

- Locks the UART transmission hardware resource
- Clear the event flag.
- Calls write function of UART HAL driver

- Wait until write operation is completed. Event is signaled in event flag object
- Calls communicationAbort function of UART HAL driver to abort write operation.
- Unlock the UART transmission hardware resource

7.32.7 SF_UART_COMMS_Lock

```
ssp_err_t SF_UART_COMMS_Lock ( sf_comms_ctrl_t  *const p_api_ctrl ,
 sf_comms_lock_t  lock_type ,   UINT timeout )
```

7.32.7.1 Brief description

Lock the UART resource.

7.32.7.2 Detailed description

Table 940:Parameters

Name	Direction	Description
p_api_ctrl	in	Pointer to the UART control block
lock_type	in	Type of Lock.
timeout	in	timeout for lock.

Table 941:Return values

Name	Description
SSP_SUCCESS	Locking UART resource successful.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL.
SSP_ERR_NOT_OPEN	Channel is not opened.
SSP_ERR_TIMEOUT	Timeout Error. 'Receive mutex get' timed out 'Transmit mutex get' timed out

7.32.8 SF_UART_COMMS_Unlock

```
ssp_err_t SF_UART_COMMS_Unlock ( sf_comms_ctrl_t  *const p_api_ctrl ,
 sf_comms_lock_t  lock_type )
```

7.32.8.1 Brief description

UnLock the UART resource.

7.32.8.2 Detailed description

Table 942:Parameters

Name	Direction	Description
p_api_ctrl	in	Pointer to the UART control block
lock_type	in	Type of Lock.

Table 943:Return values

Name	Description
SSP_SUCCESS	Unlocking UART resource successful.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL.
SSP_ERR_NOT_OPEN	Channel is not opened.
SSP_ERR_TIMEOUT	Timeout Error. 'Receive mutex put' timed out 'Transmit mutex put' timed out

7.32.9 SF_UART_COMMS_VersionGet

```
ssp_err_t SF_UART_COMMS_VersionGet ( ssp_version_t  *const p_version )
```

7.32.9.1 Detailed description

Table 944:Return values

Name	Description
SSP_SUCCESS	Version number obtained successfully.
SSP_ERR_ASSERTION	

7.32.10 API Data

7.32.10.1 sf_uart_comms_state_t

sf_uart_comms_state_t

Detailed description

Framework UART state

Enumerated values

Name	Description
SF_UART_COMMS_STATE_CLOSED	UART port is closed.
SF_UART_COMMS_STATE_OPENED	UART port is opened.
SF_UART_COMMS_STATE_READING	UART port is on data reception.
SF_UART_COMMS_STATE_WRITING	UART port is on data transmission.

7.32.11 Extensions

7.32.11.1 sf_uart_comms_instance_ctrl_t

sf_uart_comms_instance_ctrl_t

Detailed description

UART communications instance control structure. DO NOT INITIALIZE. Initialization occurs when open is called

Variables

- uint32_t state

  UART status.
- uart_instance_t const * p_lower_lvl_uart

  Pointer to UART interface (copied from cfg)
- TX_MUTEX mutex[2]

  Pointer to the mutex object for UART resource mutual exclusion.
- TX_EVENT_FLAGS_GROUP eventflag[2]

  Pointer to the event flag object for UART data transfer.
- TX_QUEUE queue

  Queue for reading.

- uint32_t queue_mem[SF_UART_COMMS_CFG_QUEUE_SIZE_WORDS]
  Queue memory.

7.32.11.2 sf_uart_comms_cfg_t

sf_uart_comms_cfg_t

Detailed description

Configuration for RTOS integrated UART driver

Variables

- uart_instance_t const * p_lower_lvl_uart
  Pointer to UART Driver instance

7.33 NetX Synergy Port

Name of module used by error logger macro

7.33.1 Functions

- edmac_eint_isr
- nx_ms_to_ticks
- nx_synergy_ethernet_init
- nx_driver_timer_handler
- nx_driver_event_handler
- enet_hw_enable_interrupt
- nx_ether_custom_packet_send
- nx_ether_driver
- nx_ether_interrupt
- nx_ether_send
- nx_ether_write
- nx_storage_init
- nx_tx_interrupt
- nx_rx_interrupt
- nx_receive
- nx_driver_deferred_processing
- nx_ethernet_version_get

7.33.2 edmac_eint_isr

```
edmac_eint_isr (  void  )
```

7.33.2.1 Brief description

edmac_eint_isr

7.33.2.2 Detailed description

This is the Ethernet interrupt service routine. It clears the interrupt flag and calls nx_ether_interrupt to process the interrupt.

7.33.2.3 Function steps

- Save context if RTOS is used
- Process the interrupt.
- Clear pending interrupt flag to make sure it doesn't fire again after exiting.
- Restore context if RTOS is used

7.33.3 nx_ms_to_ticks

```
nx_ms_to_ticks (  ULONG time_ms )
```

7.33.4 nx_synergy_ethernet_init

```
nx_synergy_ethernet_init ( NX_REC * nx_rec_ptr ,  nx_cfg_t  * sf_el_nx_cfg_ptr )
```

7.33.4.1 Brief description

nx_synergy_ethernet_init

7.33.4.2 Detailed description

This function initializes the specified Ethernet port.

Table 945:Parameters

Name	Direction	Description
nx_rec_ptr	in	: Pointer to Ethernet record structure

Table 945:Parameters (Continued)

Name	Direction	Description
sf_el_nx_cfg_ptr	in	: Pointer to NetX configuration structure

Table 946:Return values

Name	Description
SSP_SUCCESS	: Call successful.
NX_NOT_SUCCESSFUL	: Call not successful.

7.33.4.3 Function steps

- Configure the Ethernet interrupt.
- Enable clock
- Reset PHY
- Initialize & reset EDMAC and ETHERC
- Wait at least 64 cycles of PCLKA to reset the EDMAC and ETHERC. PCLKA must be at least 12.5 MHz to use Ethernet, so wait at least 5.12 us.
- Set ETHERC default modes 100 Mbps, Full duplex
- Set to little Endian.
- Initialize controller
- Set up descriptor addresses
- Configure FIFO
- Enable EDMAC receive
- Enable receive, transmit, ECI interrupts.
- Initialize PHY.
- Start PHY auto negotiation
- Create a timer to poll for completion of autonegotiation.

7.33.5 nx_driver_timer_handler

```
nx_driver_timer_handler (  ULONG data )
```

7.33.5.1 Brief description

nx_driver_timer_handler

7.33.5.2 Detailed description

This function is called from the timer expiration event. On every time event, this routine register a deferred event in IP helper thread

Table 947:Parameters

Name	Direction	Description
data	in	: Data.

7.33.5.3 Function steps

- Mark the polling flag.
- Call NetX to register a deferred event.

7.33.6 nx_driver_event_handler

```
nx_driver_event_handler ( NX_REC  * nx_rec_ptr )
```

7.33.6.1 Brief description

nx_driver_event_handler

7.33.6.2 Detailed description

This function is called from the IP thread deferred event. On every deferred event, this routine checks for Phy link status and handles link up/down and link change. During initialization this routine is responsible for checking for autonegotiation.

Table 948:Parameters

Name	Direction	Description
nx_rec_ptr	in	: Pointer to Ethernet record structure.

7.33.6.3 Function steps

- Determine previous link status.
- Save link status and changed polling interval.

- Check PHY link status.
- Save link status and changed polling interval.

7.33.7 enet_hw_enable_interrupt

```
enet_hw_enable_interrupt ( NX_REC  * nx_rec_ptr )
```

7.33.7.1 Brief description

enet_hw_enable_interrupt

7.33.7.2 Detailed description

This function enables interrupts for the given Ethernet port.

Table 949:Parameters

Name	Direction	Description
nx_rec_ptr	in	: Pointer to Ethernet record structure.

NOTE: The pointer parameter passed to this function is already validated at the higher level.

7.33.7.3 Function steps

- Enable interrupts at the NVIC.
- Enable interrupts at the Ethernet controller

7.33.8 nx_ether_custom_packet_send

```
ssp_err_t nx_ether_custom_packet_send (  NX_PACKET_POOL * pool_ptr ,  NX_REC
* nx_record_ptr ,   UCHAR * data ,   UINT length ,   USHORT ether_type ,
 nx_mac_address_t  dest_mac_address )
```

7.33.8.1 Brief description

nx_ether_custom_packet_send

7.33.8.2 Detailed description

Ethernet driver routine to send a user data in raw Ethernet packet frame from a Source HW address to destination HW address with the requested EtherType through a Ethernet channel

Table 950:Parameters

Name	Direction	Description
pool_ptr	in	: Pointer to packet pool to use
nx_record_ptr	in	: Pointer to Ethernet record structure
data	in	: Pointer to data to be send
length	in	: length of data to send
ether_type	in	: Type of Ethernet packet
dest_mac_address	in	: Destination hardware address to send the user data

Table 951:Return values

Name	Description
SSP_SUCCESS	: Call successful
SSP_ERR_ASSERTION	: - A parameter pointers point to NULL • Invalid Ethernet record structure pointer • Invalid data length
SSP_ERR_OUT_OF_MEMORY	: No memory is free to allocate the packet in the pool buffer
SSP_ERR_NOT_OPEN	: Link not enabled
SSP_ERR_INTERNAL	: - Maximum transmit queue depth has been reached • No packet is available to append the data

7.33.8.3 Function steps

- Free the packet that we will not send.
- Setup the prepend pointer in order to build the Ethernet frame. Backup another 2 bytes to get 32-bit word alignment.
- Build the actual Ethernet frame.

- Build the sender's MAC access.
- Set the Ethernet frame type.
- Endian swapping if necessary.
- Put the frame on the wire.

7.33.9 nx_ether_driver

```
ssp_err_t nx_ether_driver (  NX_IP_DRIVER * driver_req_ptr ,  NX_REC
* nx_rec_ptr ,  nx_cfg_y  * sf_el_nx_cfg_ptr )
```

7.33.9.1 Brief description

nx_ether_driver

7.33.9.2 Detailed description

Ethernet driver function for Renesas Synergy.

Table 952:Parameters

Name	Direction	Description
driver_req_ptr	in	: Pointer to driver request structure
nx_rec_ptr	in	: Pointer to Ethernet record structure
sf_el_nx_cfg_ptr	in	: sf_el_nx configuration structure pointer. This is similar to SSP configuration structure

Table 953:Return values

Name	Description
SSP_SUCCESS	: Call successful.
SSP_ERR_ASSERTION	: One or more pointers point to NULL.

See NX user manual for all possible return values.

7.33.9.3 Function steps

- Perform parameter checking

- Setup the IP pointer from the driver request.
- Default to successful return.
- Extract driver command.
- Process the driver request.
- Save the callback record pointer in the record structure.
- Record the interface structure in the driver record.
- This command is used in multi homed devices - return NX_SUCCESS by default.
- Process driver initialization.
- Initialize BDs.
- Record the interface structure in the driver record.
- Update associated channel information in driver record
- Configure mac address.
- Get MAC address from user.
- Initialize the ETHERC and E-DMAC hardware.
- Setup the link maximum IP layer transfer unit.
- See if we can honor the NX_LINK_ENABLE request. NX_ALREADY_ENABLED: Device has already been enabled
- Enable the interrupts, at the interrupt controller and Ethernet controller.
- Make sure we are in the right state to do the NX_LINK_DISABLE.
- Disable receive and transmit.
- Clear link enabled flag.
- Clear the enabled flag since there is no-one else.
- Free the packet that we will not send.
- Process driver send packet. Place the Ethernet frame at the front of the packet.
- Adjust packet length.
- Setup the prepend pointer in order to build the Ethernet frame. Backup another 2 bytes to get 32-bit word alignment.
- Build the actual Ethernet frame.
- Use MAC broadcast for this frame.
- Build the sender's MAC access.
- Set the Ethernet frame type.
- Endian swapping if necessary.
- Put the frame on the wire.
- Enable multicast.
- The device automatically receives multicast packets. Nothing needs to be done.
- Process driver deferred requests. Process a device driver function on behalf of the IP thread.

- Return the unhandled command status.
- Return NULL in the supplied return pointer.

7.33.10 nx_ether_interrupt

```
nx_ether_interrupt ( NX_REC  * nx_rec_ptr )
```

7.33.10.1 Brief description

nx_ether_interrupt

7.33.10.2 Detailed description

This function is the main Renesas Ethernet interrupt handler.

Table 954:Parameters

Name	Direction	Description
nx_rec_ptr	in	: Pointer to Ethernet record structure

7.33.10.3 Function steps

- Read EDMAC and EtherC status register.
- Clear all interrupts.
- Frame transmit completed interrupt The MAC sets Transmit Complete flag only if all frames are transmitted. This creates a delay in processing transmitted frames. The work around: this driver process transmitted packets when frame receive interrupt occurs. This way the transmitted packets can be released before processing received packets, reducing the delay.
- Frame received.
- Special case for ECI interrupt + link change.
- To report link status changes to the application, add a semaphore put or event flag set to the statement below.
- Link present.

7.33.11 nx_ether_send

```
nx_ether_send (  NX_PACKET * packet ,  NX_REC  * nx_rec_ptr )
```

7.33.11.1 Brief description

nx_ether_send

7.33.11.2 Detailed description

This function decides whether to queue or send the packet

Table 955:Parameters

Name	Direction	Description
packet	in	: Pointer to the data packet
nx_rec_ptr	in	: Pointer to Ethernet record structure

NX_SUCCESS : Success NX_TX_QUEUE_DEPTH : Queue depth NX_NOT_ENABLED : Link is not enabled See NX user manual for all possible return values

7.33.11.3 Function steps

- Check if the link is enabled.
- Call write function to send the packet, then check the result.
- Transmit failed, drop the packet.

7.33.12 nx_ether_write

```
nx_ether_write (  NX_PACKET * pkt ,  NX_REC  * nx_rec_ptr )
```

7.33.12.1 Brief description

nx_ether_write

7.33.12.2 Detailed description

This function writes the data to the TX FIFO buffer descriptor. This function can only be called when there is no active transmission.

Table 956:Parameters

Name	Direction	Description
packet	in	: Pointer to the data packet
nx_rec_ptr	in	: Pointer to Ethernet record structure

NX_SUCCESS : Success NX_NO_MORE_ENTRIES : No more entries(BD's) allowed See NX user manual for all possible return values

NOTE: The pointer parameters passed to this function are already validated at higher level.

7.33.12.3 Function steps

- Check if the BD is free.
- Set up a transfer descriptor for each NX_PACKET in the packet chain.
- Special case: alignment requirement to 5 bits for data 16 bytes or less. To accommodate this special case, NX_PACKETs need to be minimum 32+16 in size.
- Check alignment for packets 16 bytes or less.
- Loop from back to front to copy data in the NX_PACKET.
- Set new prepend and append pointers.
- Set up the transmit BD.
- This was the last packet in the chain. Store the original packet pointer in the last BD.
- Get next NetX packet in the chain.
- Move to next BD.
- Disable interrupts.
- Restore interrupts.
- Set active flag to indicate BDs are ready.
- Set active flag in reverse order, move to previous BD.
- Set active flag.
- If the transmission is suspended, resume transmission.
- Start transmission.

7.33.13 nx_storage_init

```
nx_storage_init (  NX_IP * ip_ptr ,  NX_REC  * nx_rec_ptr )
```

7.33.13.1 Brief description

nx_storage_init

7.33.13.2 Detailed description

This function initializes byte pool, packet pool, list of descriptors and Tx & Rx FIFO.

Table 957:Parameters

Name	Direction	Description
ip_ptr	in	: Pointer to the data packet
nx_rec_ptr	in	: Pointer to Ethernet record structure

NX_SUCCESS : Successful send NX_NO_PACKET : No packet available NX_WAIT_ABORTED : Requested suspension was aborted by a call to tx_thread_wait_abort. See NX user manual for all possible return values

NOTE: The pointer parameters passed to this function are already validated at higher level.

NOTE: The number of receive buffer descriptors NUM_RX_DESC is a user configurable value which is present in sf_el_nx_cfg header file and it defaults to 8.

7.33.13.3 Function steps

- All memory is assumed to be allocated at link time in the .bss section to avoid memory allocation errors at runtime.
- 16 byte alignment for BDs.
- Initialize receive buffer descriptors.
- Allocate packets from the pool to be used for reception.
- Check if the allocation was successful.
- Release allocated packets.
- Set buffer length, payload address (32byte align), initial status.
- Length has to be multiple of 32.
- Set DL bit in last descriptor.

7.33.14 nx_tx_interrupt

```
nx_tx_interrupt ( NX_REC  * nx_rec_ptr )
```

7.33.14.1 Brief description

nx_tx_interrupt

7.33.14.2 Detailed description

This function handles the Tx done interrupt.

Table 958:Parameters

Name	Direction	Description
nx_rec_ptr	in	: Pointer to Ethernet record structure

NOTE: The pointer parameter passed to this function is already validated at the higher level.

7.33.14.3 Function steps

- Loop through buffers in use.
- Check if the BD is in non active state.
- Check if this is the last BD in a chain.
- Release packet transmitted.
- Move to next BD.

7.33.15 nx_rx_interrupt

```
nx_rx_interrupt ( NX_REC  * nx_rec_ptr )
```

7.33.15.1 Brief description

nx_rx_interrupt

7.33.15.2 Detailed description

This function handles the receive interrupt.

Table 959:Parameters

Name	Direction	Description
nx_rec_ptr	in	: Pointer to Ethernet record structure

NOTE: The pointer parameter passed to this function is already validated at the higher level.

7.33.15.3 Function steps

- Do for every complete frame in the receive buffer.
- Each frame consists of 1 or more BDs.
- The driver should be able to process at least 1 completed frame when it gets here.
- Get the number of BDs in this receive chain: Last BD has FRAME_POSITION_END set, BDs preceding should have DESCRIPTOR_FLAG_ACTIVE bit cleared. The value of FRAME_POSITION_END is 0x10000000U The value of TX/RX DESCRIPTOR_FLAG_ACTIVE is 0x80000000U
- This frame has a BD that is still active and is not fully received, break out here.
- Move to next BD.
- Get first packet from first BD.
- Store the total frame length from the last BD + the padding in the NetX packet.
- Get the last packet in the chain, we need this to build NetX packet chain.
- For each packet in the chain, except the last one:
- Adjust the append pointer according to the size in BD.
- Update remaining_bytes.
- Chain packet (set _next pointer).
- Set nx_packet_last pointer to point to last packet in chain.
- Insert a new NetX packet into the BD.
- Set buffer length and address and align the address to 32 bytes boundary.
- Make the size to be multiple of 32.
- Check if we had some successful allocation.
- Release the packet.
- Set the BDs to active.
- Set buffer length and address. Align to 32 byte address
- Make the size to be multiple of 32.
- Check if we had allocation failures.
- Advance the index for next reception.
- Last BD or packet.
- Set append pointer for the last packet.
- Clear _next pointer.
- Insert a new NetX packet into the BD.
- Send the packet up the stack.

- Allocation failed. Assign last packet to the BD.
- Prepare the BD for receiving data.
- Set buffer length and address align to 32 byte address.
- Make the size to be multiple of 32.
- advance the index for next reception
- If reception is in suspended state, resume it.
- Resume reception.

7.33.16 nx_receive

```
nx_receive ( NX_REC  * nx_rec_ptr ,   NX_PACKET * packet_ptr )
```

7.33.16.1 Brief description

nx_receive

7.33.16.2 Detailed description

nx_rx_interrupt() This function processing incoming packets. This routine would be called from the receive packet ISR.

Table 960:Parameters

Name	Direction	Description
nx_rec_ptr	in	: Pointer to Ethernet record structure
packet_ptr	in	: Packet pointer

NOTE: The pointer parameters passed to this function are already validated at higher level.

7.33.16.3 Function steps

- Pickup the packet header to determine where the packet needs to be sent.
- Clean off the Ethernet header.
- Adjust the packet length.
- Route the incoming packet according to its Ethernet type.
- Call the callback for unsupported packet type, if defined.
- If Ethernet header id invalid, release the packet.

7.33.17 nx_driver_deferred_processing

```
 nx_driver_deferred_processing (  NX_IP_DRIVER * driver_req_ptr ,  NX_REC
* nx_rec_ptr )
```

7.33.17.1 Brief description

nx_driver_deferred_processing

7.33.17.2 Detailed description

This function processing the deferred action within the context of the IP thread.

Table 961:Parameters

Name	Direction	Description
driver_req_ptr	in	: Driver command from the IP thread
nx_rec_ptr	in	: Pointer to Ethernet record structure

7.33.17.3 Function steps

- Check if we need to poll PHY status.
- If so, clear the polling flag.
- Mark request as successful.

7.33.18 nx_ethernet_version_get

```
ssp_err_t nx_ethernet_version_get ( ssp_version_t  *const p_version )
```

7.33.18.1 Brief description

Retrieve the API version number.

7.33.18.2 Detailed description

Table 962:Return values

Name	Description
SSP_SUCCESS	Successful return.

Table 962:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	The parameter p_version is NULL.

7.33.18.3 Function steps

- Return the version number

7.34 NetX Synergy Port PHY Driver

7.34.1 Functions

- Phy_Init
- Phy_Start_Autonegotiate
- Phy_Get_Autonegotiate
- Phy_GetLinkStatus

7.34.2 Phy_Init

```
Phy_Init (  uint32_t channel )
```

7.34.2.1 Brief description

Phy_Init - Initialize Ethernet PHY device.

7.34.2.2 Detailed description

Table 963:Parameters

Name	Direction	Description
channel	in	Ethernet channel number

Table 964:Return values

Name	Description
R_PHY_OK	PHY device is initialized successfully
R_PHY_ERROR	PHY device is not initialized successfully

NOTE: Refer HW manual for valid channels for a target MCU

7.34.2.3 Function steps

- Resets PHY device
- Waits the reset completion, PHY_CONTROL_RESET bit is self-cleared after 1 is written to it.
- Sets Duplex Mode as Full-duplex

7.34.3 Phy_Start_Autonegotiate

```
Phy_Start_Autonegotiate (  uint32_t channel ,   uint8_t pause )
```

7.34.3.1 Brief description

Sets Auto-Negotiation advertisement and starts auto-negotiation.

7.34.3.2 Detailed description

Table 965:Parameters

Name	Direction	Description
channel	in	Ethernet channel number
pause	in	Using state of pause frames

Table 966:Return values

Name	Description
void	

7.34.4 Phy_Get_Autonegotiate

```
Phy_Get_Autonegotiate (  uint32_t channel ,   uint16_t * p_line_speed_duplex ,
 uint16_t * p_local_pause ,   uint16_t * p_partner_pause )
```

7.34.4.1 Brief description

Gets capabilities of an Ethernet PHY device.

7.34.4.2 Detailed description

Table 967:Parameters

Name	Direction	Description
channel	in	Ethernet channel number
p_line_speed_duplex	in	A pointer to the location of both the line speed and the duplex
p_local_pause	in	A pointer to the location to store the local pause bits
p_partner_pause	in	A pointer to the location to store the partner pause bits

Table 968:Return values

Name	Description
R_PHY_OK	Got information successfully
R_PHY_ERROR	PHY device is yet to be initialized

NOTE: The value returned to local_pause and patner_pause is used as it is as an argument of ether_pause_resolution function.

NOTE: Validating parameters is not required by this function as it by design is only called by sf_el_nx framework with valid parameters.

7.34.4.3 Function steps

- Reads the status register. Because reading the first time shows the previous state, the Link status bit should be read twice.
- Checks the link status
- Check the auto-negotiation status
- Gets local pause capability

7.34.5 Phy_GetLinkStatus

```
Phy_GetLinkStatus (  uint32_t channel )
```

7.34.5.1 Brief description

Phy_GetLinkStatus - Returns the status of the physical link.

7.34.5.2 Detailed description

Table 969:Parameters

Name	Direction	Description
channel	in	Ethernet channel number

Table 970:Return values

Name	Description
R_PHY_OK	PHY device is initialized successfully
R_PHY_ERROR	PHY device is not initialized successfully

7.35 TES D/AVE 2D Port Driver

7.35.1 Functions

7.35.2 d1_opendevice

```
d1_opendevice (  d1_long_t flags )
```

7.35.2.1 Brief description

Creates a device handle to access hardware. This function initializes the D1 device handle, supply module clock to D/AVE 2D hardware and enables the D/AVE 2D interrupt. It is called by the D/AVE 2D driver function d2_inithw() to initialize the D/AVE 2D hardware.

7.35.2.2 Detailed description

Table 971:Parameters

Name	Direction	Description
flags	in	Reserved. Not used in this function.

Table 972:Return values

Name	Description
Non-NULL	The function returns the pointer to a d1_device object if the D1 device handle was successfully initialized.
NULL	The function returns NULL if failed to create the display list synchronization semaphore.

7.35.2.3 Function steps

- Get new device handle.
- Initialize device data.
- Access to FMI through the productFeatureGet interface and get the D/AVE 2D hardware information.
- Initialize the D/AVE 2D hardware bass address in the device context.
- Supply clock to the D/AVE 2D hardware.
- Create the semaphore to notify the completion of display list execution.
- Enable the D/AVE 2D interrupt if semaphore creation was successful.
- If failed to enable the D/AVE 2D interrupt, set NULL to handle.
- If failed to create the semaphore, set NULL to handle.
- Returns the pointer to the d1_device object.

7.35.3 d1_closedevice

```
d1_closedevice (  d1_device * handle )
```

7.35.3.1 Brief description

Close a device handle. It is called by the D/AVE 2D driver function d2_deinithw to de-initialize the D/AVE 2D hardware. Disables the D/AVE 2D interrupt and stop the module clock supply.

7.35.3.2 Detailed description

Table 973:Parameters

Name	Direction	Description
handle	in	Pointer to the d1_device object.

Table 974:Return values

Name	Description
0	The function returns 0 if error occurred, NULL is passed to the argument handle, or failed to release the semaphore.
1	The function returns 1 if successfully device handle was closed.

7.35.3.3 Function steps

- Disable the D/AVE 2D interrupt.
- Stop clock supply to the D/AVE 2D hardware.
- Delete used semaphore

7.35.4 d1_setregister

```
d1_setregister (  d1_device * handle ,   d1_int_t deviceid ,   d1_int_t index ,
 d1_long_t value )
```

7.35.4.1 Brief description

Write data to the D/AVE 2D hardware register.

7.35.4.2 Detailed description

Table 975:Parameters

Name	Direction	Description
handle	in	Pointer to a device handle.

Table 975:Parameters (Continued)

Name	Direction	Description
deviceid	in	D1_DAVE2D(Rendering core) or D1_DLISTINDIRECT(Lists of dlist support). The others are ignored.
index	in	Register index (word offset from the D/AVE 2D base address).
value	in	32-bit value to write.

7.35.4.3 Function steps

- Write data to specified D/AVE 2D register.

7.35.5 d1_getregister

```
d1_getregister (  d1_device * handle ,   d1_int_t deviceid ,   d1_int_t index )
```

7.35.5.1 Brief description

Read data from hardware register. Reading a register from an invalid or unsupported device ID will always return 0.

7.35.5.2 Detailed description

Table 976:Parameters

Name	Direction	Description
handle	in	Pointer to a device handle.
deviceid	in	D1_DAVE2D(Rendering core) or D1_DLISTINDIRECT(Lists of dlist support). The others are ignored.
index	in	Register index (starts with 0).

Table 977:Return values

Name	Description
Value	The function returns 32-bit value of the register.

7.35.6 d1_devicesupported

```
d1_devicesupported (  d1_device * handle ,   d1_int_t deviceid )
```

7.35.6.1 Brief description

Check if the specified device ID is valid for the D/AVE 2D implemented for Synergy. Use this function to verify that a specific hardware interface is available on the current host system.

7.35.6.2 Detailed description

Table 978:Parameters

Name	Direction	Description
handle	in	Pointer to a device handle.
deviceid	in	D1_DAVE2D(Rendering core). The others are ignored.

Table 979:Return values

Name	Description
0	The function returns 0 if specified device ID not supported.
1	The function returns 1 if specified device ID supported.

7.35.7 d1_queryirq

```
d1_queryirq (  d1_device * handle ,   d1_int_t irqmask ,   d1_int_t timeout )
```

7.35.7.1 Brief description

Wait for next IRQ being happened.

7.35.7.2 Detailed description

Table 980:Parameters

Name	Direction	Description
handle	in	Pointer to the d1_device object (Not used).
irqmask	in	Interrupt ID (Not used. Synergy only uses Display list IRQ).
timeout	in	Timeout value.

Table 981:Return values

Name	Description
0	Always return 0.

7.35.7.3 Function steps

- Wait for dlist processing to complete.

7.35.8 d1_allocmem

```
d1_allocmem (  d1_uint_t size )
```

7.35.8.1 Brief description

Allocates memory in the driver heap.

7.35.8.2 Detailed description

Table 982:Parameters

Name	Direction	Description
size	in	Size of the memory to be allocated.

Table 983:Return values

Name	Description
Non-NULL	The function returns the pointer to memory chunk if memory allocation was successful.
NULL	The function returns NULL if memory allocation was failed.

7.35.8.3 Function steps

- Create a byte memory pool in the driver heap if this function call is the first time.
- Allocate memory from a byte memory pool.

7.35.9 d1_freemem

```
d1_freemem (  void * ptr )
```

7.35.9.1 Brief description

Free the specified memory area in the driver heap.

7.35.9.2 Detailed description

Table 984:Parameters

Name	Direction	Description
ptr	in	Pointer to the memory area to be freed.

7.35.9.3 Function steps

- Free specified memory allocated by d1_allocmem.

7.35.10 d1_memsize

```
d1_memsize (  void * ptr )
```

7.35.10.1 Brief description

This function intends to return the size of the given memory block but we don't return valid value. This function returns always 1.

7.35.10.2 Detailed description

Table 985:Parameters

Name	Direction	Description
ptr	in	Pointer to a memory block in Heap.

Table 986:Return values

Name	Description
1	The function always return 1.

7.35.10.3 Function steps

- Always return 1.

7.35.11 d1_allocvidmem

```
d1_allocvidmem (  d1_device * handle ,   d1_int_t memtype ,   d1_uint_t size )
```

7.35.11.1 Brief description

Allocate video memory. Synergy does not use the virtual memory space so this function exactly works same as d1_allocmem.

7.35.11.2 Detailed description

Table 987:Parameters

Name	Direction	Description
handle	in	Pointer to the d1_device object.
memtype	in	Type of memory.

Table 987:Parameters (Continued)

Name	Direction	Description
size	in	Number of bytes.

Table 988:Return values

Name	Description
Non-NULL	The function returns the pointer to memory chunk if memory allocation was successful.
NULL	The function returns Null if memory allocation was failed.

7.35.12 d1_freevidmem

```
d1_freevidmem (  d1_device * handle ,   d1_int_t memtype ,   void * ptr )
```

7.35.12.1 Brief description

Free video memory. Synergy does not use the virtual memory space so this function exactly works same as d1_freemem.

7.35.12.2 Detailed description

Table 989:Parameters

Name	Direction	Description
handle	in	Pointer to the d1_device object.
memtype	in	Type of memory.
ptr	in	Address returned by d1_allocvidmem.

7.35.13 d1_queryvidmem

```
d1_queryvidmem (  d1_device * handle ,   d1_int_t memtype ,   d1_int_t query )
```

7.35.13.1 Brief description

Get current memory status. This function does not do anything special.

7.35.13.2 Detailed description

Table 990:Parameters

Name	Direction	Description
handle	in	Pointer to the d1_device object.
memtype	in	Type of memory.
query	in	Type of requested information.

Table 991:Return values

Name	Description
0	The function always return 0.

7.35.14 d1_queryarchitecture

```
d1_queryarchitecture (  d1_device * handle )
```

7.35.14.1 Brief description

Return hints about systems memory architecture.

7.35.14.2 Detailed description

Table 992:Parameters

Name	Direction	Description
handle	in	Pointer to the d1_device object.

Table 993:Return values

Name	Description
d1_ma_unified	The function always return d1_ma_unified (Unified memory architecture).

7.35.15 d1_mapvidmem

```
d1_mapvidmem (  d1_device * handle ,   void * ptr ,   d1_int_t flags )
```

7.35.15.1 Brief description

Map video memory for direct CPU access. Synergy does not use the virtual memory space so this function does not do anything special.

7.35.15.2 Detailed description

Table 994:Parameters

Name	Direction	Description
handle	in	Pointer to the d1_device object.
ptr	in	Video memory address returned by d1_allocvidmem.
flags	in	Memory mapping flags.

Table 995:Return values

Name	Description
ptr	The function just returns ptr back since no mapping required for Synergy.

7.35.16 d1_unmapvidmem

```
d1_unmapvidmem (  d1_device * handle ,   void * ptr )
```

7.35.16.1 Brief description

Release memory mapping. Synergy does not use the virtual memory space so this function does not do anything special.

7.35.16.2 Detailed description

Table 996:Parameters

Name	Direction	Description
handle	in	Pointer to the d1_device object.
ptr	in	Mapped video memory address returned by d1_mapvidmem.

Table 997:Return values

Name	Description
1	The function always return 1.

7.35.17 d1_maptovidmem

```
d1_maptovidmem (  d1_device * handle ,   void * ptr )
```

7.35.17.1 Brief description

Map CPU accessible address of a video memory block back to video memory address. Synergy does not use the virtual memory space so this function does not do anything special.

7.35.17.2 Detailed description

Table 998:Parameters

Name	Direction	Description
handle	in	Pointer to the d1_device object.
ptr	in	CPU accessible address pointing to a video memory block originally allocated using d1_allocvidmem.

Table 999:Return values

Name	Description
ptr	The function just returns ptr back since no mapping required for Synergy.

7.35.18 d1_mapfromvidmem

```
d1_mapfromvidmem (  d1_device * handle ,   void * ptr )
```

7.35.18.1 Brief description

Map already allocated video memory address to an address for direct CPU access.

7.35.18.2 Detailed description

Table 1000:Parameters

Name	Direction	Description
handle	in	Pointer to the d1_device object.
ptr	in	Video memory address returned by d1_allocvidmem.

Table 1001:Return values

Name	Description
ptr	The function just returns ptr back since no mapping required for Synergy.

7.35.19 d1_copytovidmem

```
d1_copytovidmem (  d1_device * handle ,   void * dst ,   const void * src ,
 d1_uint_t size ,   d1_int_t flags )
```

7.35.19.1 Brief description

Copy data to video memory. Destination (video) memory area has to be allocated by d1_allocvidmem.

7.35.19.2 Detailed description

Table 1002:Parameters

Name	Direction	Description
handle	in	Pointer to the d1_device object.
dst	in	pointer into video memory (destination).
src	in	Pointer into system memory (source).
size	in	Number of bytes to copy.
flags	in	Bitfield containing additional information on data to be copied.

Table 1003:Return values

Name	Description
1	The function always return 1.

7.35.19.3 Function steps

- Simply use C standard memcpy.

7.35.20 d1_copyfromvidmem

```
d1_copyfromvidmem (  d1_device * handle ,   void * dst ,   const void * src ,
 d1_uint_t size ,   d1_int_t flags )
```

7.35.20.1 Brief description

Copy data from video memory. Source (video) memory area has to be allocated by d1_allocvidmem.

7.35.20.2 Detailed description

Table 1004:Parameters

Name	Direction	Description
handle	in	Pointer to the d1_device object.
dst	in	pointer into system memory (destination).
src	in	Pointer into video memory (source).
size	in	Number of bytes to copy.
flags	in	Reserved for future use.

Table 1005:Return values

Name	Description
1	The function always return 1.

7.35.20.3 Function steps

- Simply use C standard memcpy.

7.35.21 d1_cacheflush

```
d1_cacheflush (  d1_device * handle ,   d1_int_t memtype )
```

7.35.21.1 Brief description

Flush CPU data caches. Synergy does not have a cache memory so does not do anything special.

7.35.21.2 Detailed description

Table 1006:Parameters

Name	Direction	Description
handle	in	Pointer to the d1_device object.

Table 1006:Parameters (Continued)

Name	Direction	Description
memtype	in	Memory pools to flush (can be ored together).

Table 1007:Return values

Name	Description
1	The function always return 1.

7.35.22 d1_cacheblockflush

```
d1_cacheblockflush ( d1_device * handle , d1_int_t memtype , const void
* ptr , d1_uint_t size )
```

7.35.22.1 Brief description

Flush part of CPU data caches. Synergy does not have a cache memory so does not do anything special.

7.35.22.2 Detailed description

Table 1008:Parameters

Name	Direction	Description
handle	in	Pointer to the d1_device object.
memtype	in	Memory pools to flush (can be ored together).
ptr	in	Start address of memory to be flushed.
size	in	Size of memory to be flushed.

Table 1009:Return values

Name	Description
1	The function always return 1.

Chapter 8 API Reference: HAL Driver Interfaces

8.1 API Reference: HAL Interfaces

The HAL Interfaces offer common APIs for functional use cases. They can be implemented by one or more HAL layer drivers. Framework Layer drivers connect to these HAL drivers through the Interface Layer.

- ADC Interface
- AES Interface
- ARC4 Interface
- CAC Interface
- CAN Interface
- CGC Interface
- COMPARATOR Interface
- CRC Interface
- CTSU Interface
- DAC Interface
- Display Interface
- DOC Interface
- ELC Interface
- External IRQ Interface
- Flash Interface
- FMI Interface
- HASH Algorithm Interface
- I2C Master Interface
- I2S Interface
- Input Capture Interface
- I/O Port Interface
- JPEG Decode Interface
- JPEG Encode Interface
- Key Matrix Interface
- Low Power Modes Interface

- Low Power Modes V2 Interface
- Low Voltage Detection Interface
- OPAMP Interface
- PDC Interface
- Quad SPI Flash Interface
- RTC Interface
- SD/MMC Interface
- SLCDC Interface
- SPI Interface
- Timer Interface
- Transfer Interface
- Random number generation
- UART Interface
- WDT Interface

8.2 ADC Interface

Interface for A/D Converters.

8.2.1 Summary

The ADC interface provides standard ADC functionality including one-shot mode (single scan), continuous scan and group scan. It also allows configuration of hardware and software triggers for starting scans. After each conversion an interrupt can be triggered, and if a callback function is provided, the call back is invoked with the appropriate event information.

Implemented by: ADC

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

ADC Interface description: ADC Driver

8.2.2 Interface API

adc_api_t

Function name	Description
.open	Initialize ADC Unit; apply power, set the operational mode, trigger sources, interrupt priority, and configurations common to all channels and sensors.
.scanCfg	Configure the scan including the channels, groups and scan triggers to be used for the unit that was initialized in the open call. Some configurations are not supported for all implementations. See implementation for details.
.scanStart	Start the scan (in case of a software trigger), or enable the hardware trigger.
.scanStop	Stop the ADC scan (in case of a software trigger), or disable the hardware trigger.
.scanStatusGet	Check scan status.
.read	Read ADC conversion result.
.read32	Read ADC conversion result into a 32-bit word.
.sampleStateCountSet	Set the sample state count for the specified channel. Not supported for all implementations. See implementation for details.
.calibrate	Calibrate ADC or associated PGA (programmable gain amplifier). The driver may require implementation specific arguments to the p_extend input. Not supported for all implementations. See implementation for details.
.offsetSet	Set offset for input PGA configured for differential input. Not supported for all implementations. See implementation for details.
.close	Close the specified ADC unit by ending any scan in progress, disabling interrupts, and removing power to the specified A/D unit.
.infoGet	Return the ADC data register address of the first (lowest number) channel and the total number of bytes to be read in order for the DTC/DMAC to read the conversion results of all configured channels. Return the temperature sensor calibration and slope data.
.versionGet	Retrieve the API version.

8.2.3 Data structures

- adc_sample_state_t
- adc_callback_args_t
- adc_info_t
- adc_channel_cfg_t
- adc_cfg_t
- adc_instance_t

8.2.4 Enumerations

- adc_mode_t
- adc_resolution_t
- adc_alignment_t
- adc_add_t
- adc_clear_t
- adc_trigger_t
- adc_sample_state_reg_t
- adc_cb_event_t
- adc_group_a_t
- adc_register_t

8.2.5 Typedefs

- adc_data_size_t
- adc_ctrl_t

8.2.6 Defines

- #define ADC_API_VERSION_MAJOR

  Initial value: `(1U)`

  Includes board and MCU related header files. Version Number of API.
- #define ADC_API_VERSION_MINOR

  Initial value: `(6U)`

8.2.7 API Data

8.2.7.1 adc_mode_t

adc_mode_t

Detailed description

ADC operation mode definitions

Enumerated values

Name	Description
ADC_MODE_SINGLE_SCAN	Single scan - one or more channels.
ADC_MODE_GROUP_SCAN	Two trigger sources to trigger scan for two groups which contain one or more channels.
ADC_MODE_CONTINUOUS_SCAN	Continuous scan - one or more channels.

8.2.7.2 adc_resolution_t

adc_resolution_t

Detailed description

ADC data resolution definitions

Enumerated values

Name	Description
ADC_RESOLUTION_12_BIT	12 bit resolution
ADC_RESOLUTION_10_BIT	10 bit resolution
ADC_RESOLUTION_8_BIT	8 bit resolution
ADC_RESOLUTION_14_BIT	14 bit resolution
ADC_RESOLUTION_16_BIT	16 bit resolution
ADC_RESOLUTION_24_BIT	24 bit resolution

8.2.7.3 adc_alignment_t

adc_alignment_t

Detailed description

ADC data alignment definitions

Enumerated values

Name	Description
ADC_ALIGNMENT_RIGHT	Data alignment right.
ADC_ALIGNMENT_LEFT	Data alignment left.

8.2.7.4 adc_add_t

`adc_add_t`

Detailed description

ADC data sample addition and averaging options

Enumerated values

Name	Description
ADC_ADD_OFF	Addition turned off for channels/sensors.
ADC_ADD_TWO	Add two samples.
ADC_ADD_THREE	Add three samples.
ADC_ADD_FOUR	Add four samples.
ADC_ADD_SIXTEEN	Add sixteen samples.
ADC_ADD_AVERAGE_TWO	Average two samples.
ADC_ADD_AVERAGE_FOUR	Average four samples.
ADC_ADD_AVERAGE_SIXTEEN	Add sixteen samples.

8.2.7.5 adc_clear_t

`adc_clear_t`

Detailed description

ADC clear after read definitions

Enumerated values

Name	Description
ADC_CLEAR_AFTER_READ_OFF	Clear after read off.
ADC_CLEAR_AFTER_READ_ON	Clear after read on.

8.2.7.6 adc_trigger_t

`adc_trigger_t`

Detailed description

ADC trigger mode definitions

Enumerated values

Name	Description
ADC_TRIGGER_ASYNC_EXT_TRG0	External asynchronous trigger; not for group modes.
ADC_TRIGGER_SYNC_ELC	Synchronous trigger via ELC.
ADC_TRIGGER_SOFTWARE	Software trigger; not for group modes.

8.2.7.7 adc_sample_state_reg_t

`adc_sample_state_reg_t`

Detailed description

ADC sample state registers

Enumerated values

Name	Description
ADC_SAMPLE_STATE_CHANNEL_0	Sample state register channel 0.
ADC_SAMPLE_STATE_CHANNEL_1	Sample state register channel 1.
ADC_SAMPLE_STATE_CHANNEL_2	Sample state register channel 2.
ADC_SAMPLE_STATE_CHANNEL_3	Sample state register channel 3.
ADC_SAMPLE_STATE_CHANNEL_4	Sample state register channel 4.
ADC_SAMPLE_STATE_CHANNEL_5	Sample state register channel 5.

Name	Description
ADC_SAMPLE_STATE_CHANNEL_6	Sample state register channel 6.
ADC_SAMPLE_STATE_CHANNEL_7	Sample state register channel 7.
ADC_SAMPLE_STATE_CHANNEL_8	Sample state register channel 8.
ADC_SAMPLE_STATE_CHANNEL_9	Sample state register channel 9.
ADC_SAMPLE_STATE_CHANNEL_10	Sample state register channel 10.
ADC_SAMPLE_STATE_CHANNEL_11	Sample state register channel 11.
ADC_SAMPLE_STATE_CHANNEL_12	Sample state register channel 12.
ADC_SAMPLE_STATE_CHANNEL_13	Sample state register channel 13.
ADC_SAMPLE_STATE_CHANNEL_14	Sample state register channel 14.
ADC_SAMPLE_STATE_CHANNEL_15	Sample state register channel 15.
ADC_SAMPLE_STATE_CHANNEL_16_TO_21	Sample state register channel 16 to 21 for unit 0 on S7G2.
ADC_SAMPLE_STATE_CHANNEL_16_TO_20	Sample state register channel 16 to 20 for unit 1 on S7G2.
ADC_SAMPLE_STATE_CHANNEL_16_TO_27	Sample state register channel 16 to 27 for unit 0 on S3A7.
ADC_SAMPLE_STATE_TEMPERATURE	Sample state register channel temperature.
ADC_SAMPLE_STATE_VOLTAGE	Sample state register channel voltage.

8.2.7.8 adc_cb_event_t

`adc_cb_event_t`

Detailed description

ADC callback event definitions

Enumerated values

Name	Description
ADC_EVENT_SCAN_COMPLETE	Normal/Group A scan complete.
ADC_EVENT_SCAN_COMPLETE_GROUP_B	Group B scan complete.
ADC_EVENT_CALIBRATION_COMPLETE	Calibration complete.

Name	Description
ADC_EVENT_CONVERSION_COMPLETE	Conversion complete.

8.2.7.9 adc_group_a_t

`adc_group_a_t`

Detailed description

ADC action for group A interrupts group B scan. This enumeration is used to specify the priority between Group A and B in group mode.

Enumerated values

Name	Description
ADC_GROUP_A_PRIORITY_OFF	Group A ignored and does not interrupt ongoing group B scan.
ADC_GROUP_A_GROUP_B_WAIT_FOR_TRIGGER	Group A interrupts Group B(single scan) which restarts at next Group B trigger.
ADC_GROUP_A_GROUP_B_RESTART_SCAN	Group A interrupts Group B(single scan) which restarts immediately after Group A scan is complete.
ADC_GROUP_A_GROUP_B_CONTINUOUS_SCAN	Group A interrupts Group B(continuous scan) which continues scanning without a new Group B trigger.

8.2.7.10 adc_register_t

`adc_register_t`

Detailed description

ADC registers used for the Read() argument

Enumerated values

Name	Description
ADC_REG_CHANNEL_0	ADC channel 0.
ADC_REG_CHANNEL_1	ADC channel 1.
ADC_REG_CHANNEL_2	ADC channel 2.
ADC_REG_CHANNEL_3	ADC channel 3.

Name	Description
ADC_REG_CHANNEL_4	ADC channel 4.
ADC_REG_CHANNEL_5	ADC channel 5.
ADC_REG_CHANNEL_6	ADC channel 6.
ADC_REG_CHANNEL_7	ADC channel 7.
ADC_REG_CHANNEL_8	ADC channel 8.
ADC_REG_CHANNEL_9	ADC channel 9.
ADC_REG_CHANNEL_10	ADC channel 10.
ADC_REG_CHANNEL_11	ADC channel 11.
ADC_REG_CHANNEL_12	ADC channel 12.
ADC_REG_CHANNEL_13	ADC channel 13.
ADC_REG_CHANNEL_14	ADC channel 14.
ADC_REG_CHANNEL_15	ADC channel 15.
ADC_REG_CHANNEL_16	ADC channel 16.
ADC_REG_CHANNEL_17	ADC channel 17.
ADC_REG_CHANNEL_18	ADC channel 18.
ADC_REG_CHANNEL_19	ADC channel 19.
ADC_REG_CHANNEL_20	ADC channel 20.
ADC_REG_CHANNEL_21	ADC channel 21.
ADC_REG_CHANNEL_22	ADC channel 22.
ADC_REG_CHANNEL_23	ADC channel 23.
ADC_REG_CHANNEL_24	ADC channel 24.
ADC_REG_CHANNEL_25	ADC channel 25.
ADC_REG_CHANNEL_26	ADC channel 26.
ADC_REG_CHANNEL_27	ADC channel 27.

Name	Description
ADC_REG_TEMPERATURE	ADC channel temperature.
ADC_REG_VOLT	ADC channel volt.

8.2.7.11 adc_data_size_t

```
typedef uint16_t adc_data_size_t
```

8.2.7.12 adc_ctrl_t

```
typedef void adc_ctrl_t
```

Detailed description

ADC control block. Allocate using driver instance control structure from driver instance header file.

8.2.8 API Structures

8.2.8.1 adc_sample_state_t

adc_sample_state_t

Detailed description

ADC sample state configuration

Variables

- adc_sample_state_reg_t reg_id

  Sample state register ID.

- uint8_t num_states

  Number of sampling states for conversion. Ch16-20/21 use the same value.

8.2.8.2 adc_callback_args_t

adc_callback_args_t

Detailed description

ADC callback arguments definitions

Variables

- uint16_t unit

  ADC device in use.

- adc_cb_event_t event

  ADC callback event.

- void const * p_context

  Placeholder for user data.

- adc_register_t channel

  Channel of conversion result. Only valid for ADC_EVENT_CONVERSION_COMPLETE.

8.2.8.3 adc_info_t

adc_info_t

Detailed description

ADC Information Structure for Transfer Interface

Variables

- __I uint16_t * p_address

  The address to start reading the data from.

- uint32_t length

  The total number of transfers to read.

- transfer_size_t transfer_size

  The size of each transfer.

- elc_peripheral_t elc_peripheral

  Name of the peripheral in the ELC list.

- elc_event_t elc_event

  Name of the ELC event for the peripheral.

- uint32_t calibration_data

  Temperature sensor calibration data (0xFFFFFFFF if unsupported). Refer to hardware manual for steps on using slope with calibration data to determine temperature

- int16_t slope_microvolts

  Temperature sensor slope in microvolts/°C.

- bool calibration_ongoing

  Calibration is in progress.

8.2.8.4 adc_channel_cfg_t

adc_channel_cfg_t

Detailed description

ADC channel(s) configuration

Variables

- uint32_t scan_mask

  Channels/bits: bit 0 is ch0; bit 15 is ch15. Use #define ADC_MASK_xxx from r_adc.h.
- uint32_t scan_mask_group_b

  Valid for group modes. Use #define ADC_MASK_xxx from r_adc.h.
- adc_group_a_t priority_group_a

  Valid for group modes.
- uint32_t add_mask

  Valid if add enabled in Open(). Use #define ADC_MASK_xxx from r_adc.h.
- uint8_t sample_hold_mask

  Channels/bits 0-2. Use #define ADC_MASK_xxx from r_adc.h.
- uint8_t sample_hold_states

  Number of states to be used for sample and hold. Affects channels 0-2.

8.2.8.5 adc_cfg_t

adc_cfg_t

Detailed description

ADC general configuration

Variables

- uint16_t unit

  ADC Unit to be used.
- adc_mode_t mode

  ADC operation mode.
- adc_resolution_t resolution

  ADC resolution 8, 10, or 12-bit.
- adc_alignment_t alignment

  Specify left or right alignment; ignored if addition used.
- adc_add_t add_average_count

  Add or average samples.
- adc_clear_t clearing

  Clear after read.
- adc_trigger_t trigger

  Default and Group A trigger source.
- adc_trigger_t trigger_group_b

  Group B trigger source; valid only for group mode.

- uint8_t scan_end_ipl

  Scan end interrupt priority.
- uint8_t scan_end_b_ipl

  Scan end group B interrupt priority.
- bool calib_adc_skip

  Option to perform calibration when channels are configured.
- void(* p_callback)(adc_callback_args_t *p_args)

  Callback function; set to NULL for none.
- void const * p_context

  Placeholder for user data. Passed to the user callback in adc_api_t::adc_callback_args_t.
- void const * p_extend

  Extension parameter for hardware specific settings.

8.2.8.6 adc_api_t

adc_api_t

Detailed description

ADC functions implemented at the HAL layer will follow this API.

8.2.8.7 open

```
ssp_err_t(* adc_api_t::open) (adc_ctrl_t *const p_ctrl, adc_cfg_t const *const
p_cfg)
```

Detailed description

Initialize ADC Unit; apply power, set the operational mode, trigger sources, interrupt priority, and configurations common to all channels and sensors. Implemented as

- R_ADC_Open
- R_SDADC_Open

NOTE: Configure peripheral clocks, ADC pins and IRQs prior to calling this function.

Table 1010:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure

Table 1010:Parameters (Continued)

Name	Direction	Description
p_cfg	in	Pointer to configuration structure

Parameter p_ctrl

Definition: `adc_ctrl_t*const p_ctrl`

ADC control block. Allocate using driver instance control structure from driver instance header file.

Parameter p_cfg

Definition: `adc_cfg_t const *const p_cfg`

ADC general configuration

- `adc_cfg_t`::`unit`

  ADC Unit to be used.

- `adc_cfg_t`::`adc_mode_t`

  ADC operation mode.

  Enumerated as:

  – ADC_MODE_SINGLE_SCAN

  – ADC_MODE_GROUP_SCAN

  – ADC_MODE_CONTINUOUS_SCAN

- `adc_cfg_t`::`adc_resolution_t`

  ADC resolution 8, 10, or 12-bit.

  Enumerated as:

  – ADC_RESOLUTION_12_BIT

  – ADC_RESOLUTION_10_BIT

  – ADC_RESOLUTION_8_BIT

  – ADC_RESOLUTION_14_BIT

  – ADC_RESOLUTION_16_BIT

  – ADC_RESOLUTION_24_BIT

- `adc_cfg_t`::`adc_alignment_t`

  Specify left or right alignment; ignored if addition used.

  Enumerated as:

  – ADC_ALIGNMENT_RIGHT

  – ADC_ALIGNMENT_LEFT

- adc_cfg_t::adc_add_t

  Add or average samples.

  Enumerated as:

  – ADC_ADD_OFF
  – ADC_ADD_TWO
  – ADC_ADD_THREE
  – ADC_ADD_FOUR
  – ADC_ADD_SIXTEEN
  – ADC_ADD_AVERAGE_TWO
  – ADC_ADD_AVERAGE_FOUR
  – ADC_ADD_AVERAGE_SIXTEEN

- adc_cfg_t::adc_clear_t

  Clear after read.

  Enumerated as:

  – ADC_CLEAR_AFTER_READ_OFF
  – ADC_CLEAR_AFTER_READ_ON

- adc_cfg_t::adc_trigger_t

  Default and Group A trigger source.

  Enumerated as:

  – ADC_TRIGGER_ASYNC_EXT_TRG0
  – ADC_TRIGGER_SYNC_ELC
  – ADC_TRIGGER_SOFTWARE

- adc_cfg_t::adc_trigger_t

  Group B trigger source; valid only for group mode.

  Enumerated as:

  – ADC_TRIGGER_ASYNC_EXT_TRG0
  – ADC_TRIGGER_SYNC_ELC
  – ADC_TRIGGER_SOFTWARE

- adc_cfg_t::scan_end_ipl

  Scan end interrupt priority.

- adc_cfg_t::scan_end_b_ipl

  Scan end group B interrupt priority.

- adc_cfg_t::calib_adc_skip

  Option to perform calibration when channels are configured.

- adc_cfg_t::p_callback

  Callback function; set to NULL for none.

- adc_cfg_t::p_context

  Placeholder for user data. Passed to the user callback in adc_api_t::adc_callback_args_t.

- adc_cfg_t::p_extend

  Extension parameter for hardware specific settings.

8.2.8.8 scanCfg

```
ssp_err_t(* adc_api_t::scanCfg) (adc_ctrl_t *const p_ctrl, adc_channel_cfg_t
const *const p_channel_cfg)
```

Detailed description

Configure the scan including the channels, groups and scan triggers to be used for the unit that was initialized in the open call. Some configurations are not supported for all implementations. See implementation for details. Implemented as

- R_ADC_ScanConfigure
- R_SDADC_ScanConfigure

Table 1011:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
p_channel_cfg	in	Pointer to scan configuration structure

Parameter p_ctrl

Definition: adc_ctrl_t*const p_ctrl

ADC control block. Allocate using driver instance control structure from driver instance header file.

Parameter p_channel_cfg

Definition: adc_channel_cfg_t const *const p_channel_cfg

ADC channel(s) configuration

- adc_channel_cfg_t::scan_mask

  Channels/bits: bit 0 is ch0; bit 15 is ch15. Use #define ADC_MASK_xxx from r_adc.h.

- adc_channel_cfg_t::scan_mask_group_b

  Valid for group modes. Use #define ADC_MASK_xxx from r_adc.h.
- adc_channel_cfg_t::adc_group_a_t

  Valid for group modes.

  Enumerated as:

  – ADC_GROUP_A_PRIORITY_OFF

  – ADC_GROUP_A_GROUP_B_WAIT_FOR_TRIGGER

  – ADC_GROUP_A_GROUP_B_RESTART_SCAN

  – ADC_GROUP_A_GROUP_B_CONTINUOUS_SCAN
- adc_channel_cfg_t::add_mask

  Valid if add enabled in Open(). Use #define ADC_MASK_xxx from r_adc.h.
- adc_channel_cfg_t::sample_hold_mask

  Channels/bits 0-2. Use #define ADC_MASK_xxx from r_adc.h.
- adc_channel_cfg_t::sample_hold_states

  Number of states to be used for sample and hold. Affects channels 0-2.

8.2.8.9 scanStart

```
ssp_err_t(* adc_api_t::scanStart) (adc_ctrl_t *const p_ctrl)
```

Detailed description

Start the scan (in case of a software trigger), or enable the hardware trigger. Implemented as

- R_ADC_ScanStart
- R_SDADC_ScanStart

Table 1012:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure

Parameter p_ctrl

Definition: adc_ctrl_t*const p_ctrl

ADC control block. Allocate using driver instance control structure from driver instance header file.

8.2.8.10 scanStop

```
ssp_err_t(* adc_api_t::scanStop) (adc_ctrl_t *const p_ctrl)
```

Detailed description

Stop the ADC scan (in case of a software trigger), or disable the hardware trigger. Implemented as

- R_ADC_ScanStop
- R_SDADC_ScanStop

Table 1013:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure

Parameter p_ctrl

Definition: `adc_ctrl_t*const p_ctrl`

ADC control block. Allocate using driver instance control structure from driver instance header file.

8.2.8.11 scanStatusGet

```
ssp_err_t(* adc_api_t::scanStatusGet) (adc_ctrl_t *const p_ctrl)
```

Detailed description

Check scan status. Implemented as

- R_ADC_CheckScanDone
- R_SDADC_CheckScanDone

Table 1014:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure

Parameter p_ctrl

Definition: `adc_ctrl_t*const p_ctrl`

ADC control block. Allocate using driver instance control structure from driver instance header file.

8.2.8.12 read

```
ssp_err_t(* adc_api_t::read) (adc_ctrl_t *const p_ctrl, adc_register_t const
reg_id, uint16_t *const p_data)
```

Detailed description

Read ADC conversion result. Implemented as

- R_ADC_Read

- R_SDADC_Read

Table 1015:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
reg_id	in	ADC channel to read (see enumeration adc_register_t)
p_data	in	Pointer to variable to load value into.

Parameter p_ctrl

Definition: `adc_ctrl_t*const p_ctrl`

ADC control block. Allocate using driver instance control structure from driver instance header file.

Parameter reg_id

Definition: `adc_register_tconst reg_id`

ADC registers used for the Read() argument

Parameter p_data

`uint16_t`

8.2.8.13 read32

```
ssp_err_t(* adc_api_t::read32) (adc_ctrl_t *const p_ctrl, adc_register_t const
reg_id, uint32_t *const p_data)
```

Detailed description

Read ADC conversion result into a 32-bit word. Implemented as

- R_SDADC_Read32

Table 1016:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
reg_id	in	ADC channel to read (see enumeration adc_register_t)
p_data	in	Pointer to variable to load value into.

Parameter p_ctrl

Definition: `adc_ctrl_t*const p_ctrl`

ADC control block. Allocate using driver instance control structure from driver instance header file.

Parameter reg_id

Definition: adc_register_tconst reg_id

ADC registers used for the Read() argument

Parameter p_data

uint32_t

8.2.8.14 sampleStateCountSet

```
ssp_err_t(* adc_api_t::sampleStateCountSet) (adc_ctrl_t *const p_ctrl,
adc_sample_state_t *p_sample)
```

Detailed description

Set the sample state count for the specified channel. Not supported for all implementations. See implementation for details. Implemented as

- R_ADC_SetSampleStateCount

Table 1017:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
p_sample	in	Pointer to the ADC channels and corresponding sample states to be set

Parameter p_ctrl

Definition: adc_ctrl_t*const p_ctrl

ADC control block. Allocate using driver instance control structure from driver instance header file.

Parameter p_sample

Definition: adc_sample_state_t*p_sample

ADC sample state configuration

- adc_sample_state_t::reg_id

  Sample state register ID.

- adc_sample_state_t::num_states

  Number of sampling states for conversion. Ch16-20/21 use the same value.

8.2.8.15 calibrate

```
ssp_err_t(* adc_api_t::calibrate) (adc_ctrl_t *const p_ctrl, void *const p_extend)
```

Detailed description

Calibrate ADC or associated PGA (programmable gain amplifier). The driver may require implementation specific arguments to the p_extend input. Not supported for all implementations. See implementation for details. Implemented as

- R_SDADC_Calibrate

Table 1018:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
p_extend	in	Pointer to implementation specific arguments

Parameter p_ctrl

Definition: `adc_ctrl_t*const p_ctrl`

ADC control block. Allocate using driver instance control structure from driver instance header file.

Parameter p_extend

`const`

8.2.8.16 offsetSet

```
ssp_err_t(* adc_api_t::offsetSet) (adc_ctrl_t *const p_ctrl, adc_register_t const reg_id, int32_t const offset)
```

Detailed description

Set offset for input PGA configured for differential input. Not supported for all implementations. See implementation for details. Implemented as

- R_SDADC_OffsetSet

Table 1019:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
reg_id	in	ADC channel to read (see enumeration adc_register_t)
offset	in	See implementation for details.

Parameter p_ctrl

Definition: `adc_ctrl_t*const p_ctrl`

ADC control block. Allocate using driver instance control structure from driver instance header file.

Parameter reg_id

Definition: adc_register_tconst reg_id

ADC registers used for the Read() argument

Parameter offset

8.2.8.17 close

```
ssp_err_t(* adc_api_t::close) (adc_ctrl_t *const p_ctrl)
```

Detailed description

Close the specified ADC unit by ending any scan in progress, disabling interrupts, and removing power to the specified A/D unit. Implemented as

- R_ADC_Close
- R_SDADC_Close

Table 1020:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure

Parameter p_ctrl

Definition: adc_ctrl_t*const p_ctrl

ADC control block. Allocate using driver instance control structure from driver instance header file.

8.2.8.18 infoGet

```
ssp_err_t(* adc_api_t::infoGet) (adc_ctrl_t *const p_ctrl, adc_info_t *const
p_adc_info)
```

Detailed description

Return the ADC data register address of the first (lowest number) channel and the total number of bytes to be read in order for the DTC/DMAC to read the conversion results of all configured channels. Return the temperature sensor calibration and slope data. Implemented as

- R_ADC_InfoGet
- R_SDADC_InfoGet

Table 1021:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
p_adc_info	out	Pointer to ADC information structure

Parameter p_ctrl

Definition: `adc_ctrl_t*const p_ctrl`

ADC control block. Allocate using driver instance control structure from driver instance header file.

Parameter p_adc_info

Definition: `adc_info_t*const p_adc_info`

ADC Information Structure for Transfer Interface

- `adc_info_t::p_address`

  The address to start reading the data from.

- `adc_info_t::length`

  The total number of transfers to read.

- `adc_info_t::transfer_size`

  The size of each transfer.

- `adc_info_t::elc_peripheral`

  Name of the peripheral in the ELC list.

- `adc_info_t::elc_event`

  Name of the ELC event for the peripheral.

- `adc_info_t::calibration_data`

  Temperature sensor calibration data (0xFFFFFFFF if unsupported). Refer to hardware manual for steps on using slope with calibration data to determine temperature

- `adc_info_t::slope_microvolts`

  Temperature sensor slope in microvolts/°C.

- `adc_info_t::calibration_ongoing`

  Calibration is in progress.

8.2.8.19 versionGet

```
ssp_err_t(* adc_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Retrieve the API version. Implemented as

- R_ADC_VersionGet
- R_SDADC_VersionGet

NOTE: This function retrieves the API version.

Table 1022:Parameters

Name	Direction	Description
p_version	in	Pointer to version structure

Parameter p_version

8.2.8.20 adc_instance_t

adc_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- adc_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- adc_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- adc_channel_cfg_t const * p_channel_cfg

  Pointer to the channel configuration structure for this instance.
- adc_api_t const * p_api

  Pointer to the API structure for this instance.

8.3 AES Interface

AES encryption and decryption APIs.

8.3.1 Interface API

aes_api_t

Function name	Description
.open	AES module open function. Must be called before performing any encrypt/decrypt operations.
.createKey	Generate an AES key for encrypt / decrypt operations.

Function name	Description
.encrypt	AES encryption. Encrypt input data with ECB mode using a 128-bit AES key
.addAdditionalAuthenticationData	Add additional authentication data (called before starting an encryption or decryption operation)
.encryptFinal	AES final encryption using the chaining mode and padding mode specified in the aes.open() function call.
.decrypt	AES Decryption. Decrypt input data with ECB mode using a 128-bit AES key
.setGcmTag	set parameter specific to the mode
.getGcmTag	Get authentication tag data.
.close	Close the AES module.
.zeroPaddingEncrypt	AES zero padding encryption using the chaining mode and padding mode specified. Implementation for GCM mode only API usage -.
.zeroPaddingDecrypt	AES zero padding decryption using the chaining mode and padding mode specified. Implementation for GCM mode only API usage -.
.versionGet	Gets version and stores it in provided pointer p_version.

8.3.2 Data structures

- aes_ctrl_t
- aes_cfg_t
- aes_instance_t

8.3.3 Variables

- g_sce_crypto_api
- g_aes128ecb_on_sce
- g_aes128cbc_on_sce
- g_aes128ctr_on_sce
- g_aes256ecb_on_sce
- g_aes256cbc_on_sce

- g_aes256ctr_on_sce
- g_aes128gcm_on_sce
- g_aes128xts_on_sce
- g_aes256gcm_on_sce
- g_aes256xts_on_sce
- g_aes128gcm_on_sceHrk
- g_aes256gcm_on_sceHrk
- g_aes192ecb_on_sce
- g_aes192cbc_on_sce
- g_aes192ctr_on_sce
- g_aes192gcm_on_sce
- g_aes128ecb_on_sceHrk
- g_aes128cbc_on_sceHrk
- g_aes128ctr_on_sceHrk
- g_aes128xts_on_sceHrk
- g_aes256ecb_on_sceHrk
- g_aes256cbc_on_sceHrk
- g_aes256ctr_on_sceHrk
- g_aes256xts_on_sceHrk
- g_aes192ecb_on_sceHrk
- g_aes192cbc_on_sceHrk
- g_aes192ctr_on_sceHrk
- g_aes192gcm_on_sceHrk

8.3.4 Defines

- #define AES_API_VERSION_MAJOR

  Initial value: `(1U)`

  Register definitions, common services and error codes.
- #define AES_API_VERSION_MINOR

  Initial value: `(0U)`
- #define DRV_AES_CONTEXT_BUFFER_SIZE

  Initial value: `(64)`

- #define AES_XTS_128_WRAPPPED_SECRET_KEY_SIZE_BYTES

  Initial value: `(52)`

  Return Wrapped AES-XTS secret key size in bytes for a 128-bit AES XTS Mode Key
- #define AES_XTS_256_WRAPPPED_SECRET_KEY_SIZE_BYTES

  Initial value: `(84)`

  Return AES-XTS secret key size in bytes for a 256-bit AES XTS Mode Key
- #define AES128_WRAPPPED_SECRET_KEY_SIZE_BYTES

  Initial value: `(36)`

  Return Wrapped AES secret key size in bytes for a 128-bit AES Key
- #define AES192_WRAPPPED_SECRET_KEY_SIZE_BYTES

  Initial value: `(52)`

  Return Wrapped AES secret key size in bytes for a 192-bit AES Key
- #define AES256_WRAPPPED_SECRET_KEY_SIZE_BYTES

  Initial value: `(52)`

  Return Wrapped AES secret key size in bytes for a 256-bit AES Key
- #define AES128_SECRET_KEY_SIZE_BYTES

  Initial value: `(16)`

  Return AES secret key size in bytes for a 128-bit AES Key
- #define AES192_SECRET_KEY_SIZE_BYTES

  Initial value: `(24)`

  Return AES secret key size in bytes for a 192-bit AES Key
- #define AES256_SECRET_KEY_SIZE_BYTES

  Initial value: `(32)`

  Return AES secret key size in bytes for a 256-bit AES Key

8.3.5 g_sce_crypto_api

```
const g_sce_crypto_api
```

8.3.6 g_aes128ecb_on_sce

```
aes_api_t::g_aes128ecb_on_sce
```

8.3.6.1 Detailed description

AES interface available boards - S7G2, S5D9, S5D5, S3A7, S3A3 and S3A6 - Chaining modes CBC, GCM, CTR, ECB, XTS for 128 & 256-bit S7G2, S5D9, and S5D5 - Chaining modes CBC, GCM, CTR, ECB for 192-bit

AES 128-bit ECB mode implementation

8.3.7 g_aes128cbc_on_sce

aes_api_t::g_aes128cbc_on_sce

8.3.7.1 Detailed description

AES 128-bit CBC mode implementation

8.3.8 g_aes128ctr_on_sce

aes_api_t::g_aes128ctr_on_sce

8.3.8.1 Detailed description

AES 128-bit CTR mode implementation

8.3.9 g_aes256ecb_on_sce

aes_api_t::g_aes256ecb_on_sce

8.3.9.1 Detailed description

AES 256-bit ECB mode implementation

8.3.10 g_aes256cbc_on_sce

aes_api_t::g_aes256cbc_on_sce

8.3.10.1 Detailed description

AES 256-bit CBC mode implementation

8.3.11 g_aes256ctr_on_sce

aes_api_t::g_aes256ctr_on_sce

8.3.11.1 Detailed description

AES 256-bit CTR mode implementation

8.3.12 g_aes128gcm_on_sce

aes_api_t::g_aes128gcm_on_sce

8.3.12.1 Detailed description

AES 128-bit GCM mode implementation

8.3.13 g_aes128xts_on_sce

aes_api_t::g_aes128xts_on_sce

8.3.13.1 Detailed description

AES 128-bit CCM mode implementation

8.3.14 g_aes256gcm_on_sce

aes_api_t::g_aes256gcm_on_sce

8.3.14.1 Detailed description

AES 256-bit GCM mode implementation

8.3.15 g_aes256xts_on_sce

aes_api_t::g_aes256xts_on_sce

8.3.15.1 Detailed description

AES 256-bit XTS mode implementation

8.3.16 g_aes128gcm_on_sceHrk

aes_api_t::g_aes128gcm_on_sceHrk

8.3.17 g_aes256gcm_on_sceHrk

aes_api_t::g_aes256gcm_on_sceHrk

8.3.18 g_aes192ecb_on_sce

aes_api_t::g_aes192ecb_on_sce

8.3.18.1 Detailed description

AES 192-bit ECB mode implementation

8.3.19 g_aes192cbc_on_sce

aes_api_t::g_aes192cbc_on_sce

8.3.19.1 Detailed description

AES 192-bit CBC mode implementation

8.3.20 g_aes192ctr_on_sce

aes_api_t::g_aes192ctr_on_sce

8.3.20.1 Detailed description

AES 192-bit CTR mode implementation

8.3.21 g_aes192gcm_on_sce

aes_api_t::g_aes192gcm_on_sce

8.3.21.1 Detailed description

AES 192-bit GCM mode implementation

8.3.22 g_aes128ecb_on_sceHrk

aes_api_t::g_aes128ecb_on_sceHrk

8.3.22.1 Detailed description

HRK Supported global structure definitions

8.3.23 g_aes128cbc_on_sceHrk

aes_api_t::g_aes128cbc_on_sceHrk

8.3.24 g_aes128ctr_on_sceHrk

aes_api_t::g_aes128ctr_on_sceHrk

8.3.25 g_aes128xts_on_sceHrk

aes_api_t::g_aes128xts_on_sceHrk

8.3.26 g_aes256ecb_on_sceHrk

aes_api_t::g_aes256ecb_on_sceHrk

8.3.27 g_aes256cbc_on_sceHrk

aes_api_t::g_aes256cbc_on_sceHrk

8.3.28 g_aes256ctr_on_sceHrk

aes_api_t::g_aes256ctr_on_sceHrk

8.3.29 g_aes256xts_on_sceHrk

aes_api_t::g_aes256xts_on_sceHrk

8.3.30 g_aes192ecb_on_sceHrk

aes_api_t::g_aes192ecb_on_sceHrk

8.3.31 g_aes192cbc_on_sceHrk

aes_api_t::g_aes192cbc_on_sceHrk

8.3.32 g_aes192ctr_on_sceHrk

aes_api_t::g_aes192ctr_on_sceHrk

8.3.33 g_aes192gcm_on_sceHrk

aes_api_t::g_aes192gcm_on_sceHrk

8.3.33.1 Detailed description

AES 192-bit GCM HRK mode implementation

8.3.34 API Structures

8.3.34.1 aes_ctrl_t

aes_ctrl_t

Detailed description

AES Interface control structure

Variables

- crypto_ctrl_t * p_crypto_ctrl

  pointer to crypto engine control structure
- crypto_api_t const * p_crypto_api

  pointer to crypto engine API
- uint32_t work_buffer[DRV_AES_CONTEXT_BUFFER_SIZE]

  Examples: AES-GCM mode uses this for storing authentication tag etc.

  used for storing context/state of the Cipher

8.3.34.2 aes_cfg_t

aes_cfg_t

Detailed description

AES Interface configuration structure. User must fill in these values before invoking the open() function

Variables

- crypto_api_t const * p_crypto_api

  pointer to crypto engine api

8.3.34.3 aes_api_t

aes_api_t

Detailed description

AES_Interface SCE functions implemented at the HAL layer will follow this API.

8.3.34.4 open

```
uint32_t(* aes_api_t::open) (aes_ctrl_t *const p_ctrl, aes_cfg_t const *const
p_cfg)
```

Detailed description

AES module open function. Must be called before performing any encrypt/decrypt operations.

Table 1023:Parameters

Name	Direction	Description
p_ctrl	inout	pointer to control structure for the AES interface. Must be declared by user. Elements are set here.
p_cfg	in	pointer to control structure for the AES configuration. All elemenets of this structure must be set by user.

Parameter p_ctrl

Definition: `aes_ctrl_t`

AES Interface control structure

Parameter p_cfg

Definition: `aes_cfg_t const *const p_cfg`

AES Interface configuration structure. User must fill in these values before invoking the open() function

- `aes_cfg_t`::`p_crypto_api`

  pointer to crypto engine api

8.3.34.5 createKey

```
uint32_t(* aes_api_t::createKey) (aes_ctrl_t *const p_ctrl, uint32_t num_words,
uint32_t *p_key)
```

Brief description

Generate an AES key for encrypt / decrypt operations.

Detailed description

Table 1024:Parameters

Name	Direction	Description
p_ctrl	inout	pointer to control structure for the AES interface.
num_words	in	number of words in buffer p_key
p_key	out	pointer to key buffer. Generated key will be stored at this location.

Parameter p_ctrl

Definition: aes_ctrl_t

AES Interface control structure

Parameter num_words

uint32_t

Parameter p_key

uint32_t

8.3.34.6 encrypt

```
uint32_t(* aes_api_t::encrypt) (aes_ctrl_t *const p_ctrl, const uint32_t *p_key,
uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)
```

Brief description

AES encryption.

Detailed description

Encrypt input data with ECB mode using a 128-bit AES key

Table 1025:Parameters

Name	Direction	Description
*p_key	in	pointer to the AES plain-text key
*p_iv	in	is a pointer to initialization vector. For ECB mode this parameter is unused. NULL value is acceptable.
num_words	in	data buffer size in words. Each word is 4-bytes. multiples of 4
*p_source	in	input data buffer
*p_dest	out	output data buffer

Parameter *p_key

uint32_t

Parameter *p_iv

uint32_t

Parameter num_words

uint32_t

Parameter *p_source

uint32_t

Parameter *p_dest

uint32_t

8.3.34.7 addAdditionalAuthenticationData

```
uint32_t(* aes_api_t::addAdditionalAuthenticationData) (aes_ctrl_t *const p_ctrl,
const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source)
```

Brief description

Add additional authentication data (called before starting an encryption or decryption operation)

Detailed description

Table 1026:Parameters

Name	Direction	Description
*p_key	in	pointer to the AES plain-text key
*p_iv	in	unused for ECB mode
num_words	in	data buffer size in words. Each word is 4-bytes. multiples of 4
*p_source	in	input data buffer

Parameter *p_key

uint32_t

Parameter *p_iv

uint32_t

Parameter num_words

uint32_t

Parameter *p_source

uint32_t

8.3.34.8 encryptFinal

```
uint32_t(* aes_api_t::encryptFinal) (aes_ctrl_t *const p_ctrl, const uint32_t
*p_key, uint32_t *p_iv, uint32_t input_num_words, uint32_t *p_source, uint32_t
output_num_words, uint32_t *p_dest)
```

Brief description

AES final encryption using the chaining mode and padding mode specified in the aes.open() function call.

8.3.34.9 decrypt

```
uint32_t(* aes_api_t::decrypt) (aes_ctrl_t *const p_ctrl, const uint32_t *p_key,
uint32_t *p_iv, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest)
```

Brief description

AES Decryption.

Detailed description

Decrypt input data with ECB mode using a 128-bit AES key

Table 1027:Parameters

Name	Direction	Description
*p_key	in	128-bit plain key
*p_iv	in	is a pointer to initialization vector. For ECB mode this parameter is unused. NULL value is acceptable.
num_words	in	Size in words of p_source and p_dest data buffers. Each word is 4-bytes. Must be multiples of 4 words.
*p_source	in	input data buffer
*p_dest	out	output data buffer

Parameter *p_key

`uint32_t`

Parameter *p_iv

`uint32_t`

Parameter num_words
Parameter *p_source

`uint32_t`

Parameter *p_dest

`uint32_t`

8.3.34.10 setGcmTag

```
uint32_t(* aes_api_t::setGcmTag) (aes_ctrl_t *const p_ctrl, uint32_t num_words,
uint32_t *p_source)
```

Brief description

set parameter specific to the mode

Detailed description

Table 1028:Parameters

Name	Direction	Description
p_ctrl	in	pointer to the control structure
num_words	in	number of words in p_source buffer. This must be atleast 4 words
p_source	in	pointer to authentication tag data buffer, must be of size 4 words.

Parameter p_ctrl

Definition: `aes_ctrl_t`

AES Interface control structure

Parameter num_words

`uint32_t`

Parameter p_source

`uint32_t`

8.3.34.11 getGcmTag

```
uint32_t(* aes_api_t::getGcmTag) (aes_ctrl_t *const p_ctrl, uint32_t num_words,
uint32_t *p_dest)
```

Brief description

Get authentication tag data.

Detailed description

Table 1029:Parameters

Name	Direction	Description
p_ctrl	in	pointer to the control structure
num_words	in	number of words in p_dest buffer. This must be atleast 4 words
p_dest	in	pointer to data buffer, must be of size 4 words.

Parameter p_ctrl

Definition: `aes_ctrl_t`

AES Interface control structure

Parameter num_words

`uint32_t`

Parameter p_dest

`uint32_t`

8.3.34.12 close

```
uint32_t(* aes_api_t::close) (aes_ctrl_t *const p_ctrl)
```

Detailed description

Close the AES module.

Table 1030:Parameters

Name	Direction	Description
p_ctrl	in	pointer to the control structure

Parameter p_ctrl

Definition: `aes_ctrl_t`

AES Interface control structure

8.3.34.13 zeroPaddingEncrypt

```
uint32_t(* aes_api_t::zeroPaddingEncrypt) (aes_ctrl_t *const p_ctrl, const
uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t
*p_dest)
```

Brief description

AES zero padding encryption using the chaining mode and padding mode specified. Implementation for GCM mode only API usage -.

Detailed description

1) Provide any Add Authentication Data (AAD): set p_dest = NULL
2) Encryption: set p_source to input data and p_dest will return encrypted data
3) Get/Compute Tag: set p_source = NULL

Table 1031:Parameters

Name	Direction	Description
p_ctrl	in	pointer to the control structure

Table 1031:Parameters (Continued)

Name	Direction	Description
*p_key	in	pointer to the AES plain-text key, the buffer size should be equal to the keylength
*p_iv	in	the buffer size must be 16 bytes
num_bytes	in	data buffer size in bytes.
*p_source	in	input data buffer, computes tag when set to NULL.
*p_dest	out	output data buffer, adds authentication data when set to NULL.

NOTE: this function is not thread safe.

Parameter p_ctrl

Definition: `aes_ctrl_t`

AES Interface control structure

Parameter *p_key

`uint32_t`

Parameter *p_iv

`uint32_t`

Parameter num_bytes

`uint32_t`

Parameter *p_source

`uint32_t`

Parameter *p_dest

`uint32_t`

8.3.34.14 zeroPaddingDecrypt

```
uint32_t(* aes_api_t::zeroPaddingDecrypt) (aes_ctrl_t *const p_ctrl, const
uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t
*p_dest)
```

Brief description

AES zero padding decryption using the chaining mode and padding mode specified. Implementation for GCM mode only API usage -.

Detailed description

1) Set expected tag value using the setGcmTag function
2) Provide any Add Authentication Data (AAD), invoke this API using p_dest = NULL
1) Decryption: set p_source to input encrypted data, decrypted data will be returned in p_dest
2) Verify the tag, invoke this API using p_source = NULL and p_dest = NULL, the return value indicates authentication tag verification status.

Table 1032:Parameters

Name	Direction	Description
p_ctrl	in	pointer to the control structure
*p_key	in	pointer to the AES plain-text key, the buffer size should be equal to the keylength
*p_iv	in	the buffer size must be 16 bytes
num_bytes	in	data buffer size in bytes.
*p_source	in	input data buffer, computes tag when set to NULL.
*p_dest	out	output data buffer, adds authentication data when set to NULL.

NOTE: this function is not thread safe.

Parameter p_ctrl

Definition: `aes_ctrl_t`

AES Interface control structure

Parameter *p_key

`uint32_t`

Parameter *p_iv

`uint32_t`

Parameter num_bytes

`uint32_t`

Parameter *p_source

`uint32_t`

Parameter *p_dest

`uint32_t`

8.3.34.15 versionGet

```
uint32_t(* aes_api_t::versionGet) ( *const p_version)
```

Detailed description

Gets version and stores it in provided pointer p_version.

Table 1033:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.3.34.16 aes_instance_t

aes_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- aes_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- aes_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- aes_api_t const * p_api

  Pointer to the API structure for this instance.

8.4 ARC4 Interface

ARC4 encryption and decryption APIs.

8.4.1 Interface API

arc4_api_t

Function name	Description
.open	ARC4 module open function. Must be called before performing any encrypt/decrypt operations. Initializes the context for the encrypt or decrypt operations using the chosen Cipher interface.
.keySet	ARC4 module key set function. Resets the state of the ARC4 encryption block.
.arc4Process	Encrypt or decrypt source data p_source of length num_bytes and write the results to destination buffer p_dest
.close	Close the ARC4 module.
.versionGet	Gets version and stores it in provided pointer p_version.

8.4.2 Data structures

- arc4_ctrl_t
- arc4_cfg_t
- arc4_instance_t

8.4.3 Variables

- g_sce_crypto_api
- g_arc4_on_sce

8.4.4 Defines

- #define ARC4_API_VERSION_MAJOR

  Initial value: (01)

  Register definitions, common services and error codes.
- #define ARC4_API_VERSION_MINOR

  Initial value: (00)

- #define DRV_ARC4_CONTEXT_BUFFER_SIZE

  Initial value: (66)

8.4.5 g_sce_crypto_api

const g_sce_crypto_api

8.4.6 g_arc4_on_sce

arc4_api_t::g_arc4_on_sce

8.4.6.1 Detailed description

ARC4 interface is only available on S7G2, S5D9 and S5D5.

SCE/ARC4 implementation of ARC4 API.

8.4.7 API Structures

8.4.7.1 arc4_ctrl_t

arc4_ctrl_t

Detailed description

ARC4 Interface control structure

Variables

- crypto_ctrl_t * p_crypto_ctrl

  pointer to crypto engine control structure

- crypto_api_t const * p_crypto_api

  pointer to crypto engine API

- uint32_t state

  used to identify state of the ARC4 control block

- uint32_t work_buffer[DRV_ARC4_CONTEXT_BUFFER_SIZE]

- bsp_lock_t open

  indicates whether driver is opened with this control block

  used for storing context of the cipher < ARC4 uses this for storing the sbox results for the next encrypt/ decrypt operations

8.4.7.2 arc4_cfg_t

arc4_cfg_t

Detailed description

ARC4 Interface configuration structure. User must fill in these values before invoking the open() function

Variables

- crypto_api_t const * p_crypto_api

  pointer to crypto engine api

- uint32_t length

  Length of p_key in bytes.

- uint8_t const * p_key

  ARC4 key to use for encrypt or decrypto operations.

8.4.7.3 arc4_api_t

arc4_api_t

Detailed description

ARC4_Interface SCE functions implemented at the HAL layer will follow this API.

8.4.7.4 open

```
uint32_t(* arc4_api_t::open) (arc4_ctrl_t *const p_ctrl, arc4_cfg_t const *const p_cfg)
```

Detailed description

ARC4 module open function. Must be called before performing any encrypt/decrypt operations. Initializes the context for the encrypt or decrypt operations using the chosen Cipher interface.

Table 1034:Parameters

Name	Direction	Description
p_ctrl	inout	pointer to control structure for the ARC4 interface. Must be declared by user. Elements are set here.
p_cfg	in	pointer to control structure for the ARC4 configuration. All elemenets of this structure must be set by user.

Parameter p_ctrl

Definition: `arc4_ctrl_t`

ARC4 Interface control structure

Parameter p_cfg

Definition: `arc4_cfg_t const *const p_cfg`

ARC4 Interface configuration structure. User must fill in these values before invoking the open() function

- arc4_cfg_t::p_crypto_api

  pointer to crypto engine api

- arc4_cfg_t::length

  Length of p_key in bytes.

- arc4_cfg_t::p_key

  ARC4 key to use for encrypt or decrypto operations.

8.4.7.5 keySet

```
uint32_t(* arc4_api_t::keySet) (arc4_ctrl_t *const p_ctrl, uint32_t length,
uint8_t const *p_key)
```

Detailed description

ARC4 module key set function. Resets the state of the ARC4 encryption block.

Table 1035:Parameters

Name	Direction	Description
p_ctrl	inout	pointer to control structure for the ARC4 interface.
length	in	length of the p_key key material in bytes
p_key	in	pointer to the key material to use for encryption operations.

Parameter p_ctrl

Definition: arc4_ctrl_t

ARC4 Interface control structure

Parameter length

uint32_t

Parameter p_key

uint8_t

8.4.7.6 arc4Process

```
uint32_t(* arc4_api_t::arc4Process) (arc4_ctrl_t *const p_ctrl, uint32_t
num_bytes, uint8_t *p_source, uint8_t *p_dest)
```

Brief description

Encrypt or decrypt source data p_source of length num_bytes and write the results to destination buffer p_dest

Detailed description

Table 1036:Parameters

Name	Direction	Description
p_ctrl	inout	pointer to control structure for ARC4 interface.
num_bytes	in	number of bytes to encrypt or decrypt, the value must be a multiple of 16
p_source	in	pointer to source data buffer
p_dest	out	pointer to destination data buffer

Parameter p_ctrl

Definition: `arc4_ctrl_t`

ARC4 Interface control structure

Parameter num_bytes

`uint32_t`

Parameter p_source

`uint8_t`

Parameter p_dest

`uint8_t`

8.4.7.7 close

```
uint32_t(* arc4_api_t::close) (arc4_ctrl_t *const p_ctrl)
```

Detailed description

Close the ARC4 module.

Table 1037:Parameters

Name	Direction	Description
p_ctrl	inout	pointer to the control structure

Parameter p_ctrl

Definition: `arc4_ctrl_t`

ARC4 Interface control structure

8.4.7.8 versionGet

```
uint32_t(* arc4_api_t::versionGet) ( *const p_version)
```

Detailed description

Gets version and stores it in provided pointer p_version.

Table 1038:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.4.7.9 arc4_instance_t

arc4_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- arc4_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- arc4_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- arc4_api_t const * p_api

  Pointer to the API structure for this instance.

8.5 CAC Interface

Interface for clock frequency accuracy measurements.

Register definitions, common services and error codes.

8.5.1 Summary

The interface for the clock frequency accuracy measurement circuit (CAC) peripheral is used to check a system clock frequency with a reference clock signal by counting the number of pulses of the clock (system clock) to be measured.

Related SSP architecture topics:

- SSP Interfaces

- SSP Predefined Layers
- Using SSP Modules

CAC Interface description: CAC Driver

8.5.2 Interface API

cac_api_t

Function name	Description
.open	Open function for CAC device.
.read	Read function for CAC peripheral.
.close	Close function for CAC device.
.stopMeasurement	End a measurement for the CAC peripheral.
.startMeasurement	Begin a measurement for the CAC peripheral.
.reset	Reset function for CAC device.
.versionGet	Get the CAC API and code version information.

8.5.3 Data structures

- cac_ref_clock_config_t
- cac_meas_clock_config_t
- cac_callback_args_t
- cac_cfg_t
- cac_instance_t

8.5.4 Enumerations

- cac_event_t
- cac_clock_type_t
- cac_clock_source_t
- cac_ref_divider_t
- cac_ref_digfilter_t
- cac_ref_edge_t

- cac_meas_divider_t

8.5.5 Typedefs

- cac_ctrl_t

8.5.6 Defines

- #define CAC_API_VERSION_MAJOR
  Initial value: `(1U)`
- #define CAC_API_VERSION_MINOR
  Initial value: `(2U)`

8.5.7 API Data

8.5.7.1 cac_event_t

`cac_event_t`

Detailed description

Event types returned by the ISR callback when used in CAC interrupt mode

Enumerated values

Name	Description
CAC_EVENT_FREQUENCY_ERROR	Frequency error.
CAC_EVENT_MEASUREMENT_COMPLETE	Measurement complete.
CAC_EVENT_COUNTER_OVERFLOW	Counter overflow.

8.5.7.2 cac_clock_type_t

`cac_clock_type_t`

Detailed description

Enumeration of the two possible clocks.

Enumerated values

Name	Description
CAC_CLOCK_MEASURED	Measurement clock.
CAC_CLOCK_REFERENCE	Reference clock.

8.5.7.3 cac_clock_source_t

`cac_clock_source_t`

Detailed description

Enumeration of the possible clock sources for both the reference and measurement clocks.

Enumerated values

Name	Description
CAC_CLOCK_SOURCE_MAIN_OSC	Main clock oscillator.
CAC_CLOCK_SOURCE_SUBCLOCK	Sub-clock.
CAC_CLOCK_SOURCE_HOCO	HOCO (High speed on chip oscillator)
CAC_CLOCK_SOURCE_MOCO	MOCO (Middle speed on chip oscillator)
CAC_CLOCK_SOURCE_LOCO	LOCO (Middle speed on chip oscillator)
CAC_CLOCK_SOURCE_PCLKB	PCLKB (Peripheral Clock B)
CAC_CLOCK_SOURCE_IWDT	IWDT- Dedicated on-chip oscillator.
CAC_CLOCK_SOURCE_MEAS_MAX	Maximum measured clock.
CAC_CLOCK_SOURCE_EXTERNAL	Externally supplied measurement clock on CACREF pin.
CAC_CLOCK_SOURCE_REF_MAX	Maximum reference clock.

8.5.7.4 cac_ref_divider_t

`cac_ref_divider_t`

Detailed description

Enumeration of available dividers for the reference clock.

Enumerated values

Name	Description
CAC_REF_DIV_32	Reference clock divided by 32.
CAC_REF_DIV_128	Reference clock divided by 128.
CAC_REF_DIV_1024	Reference clock divided by 1024.
CAC_REF_DIV_8192	Reference clock divided by 8192.

8.5.7.5 cac_ref_digfilter_t

`cac_ref_digfilter_t`

Detailed description

Enumeration of available digital filter settings for an external reference clock.

Enumerated values

Name	Description
CAC_REF_DIGITAL_FILTER_OFF	No digital filter on the CACREF pin for reference clock.
CAC_REF_DIGITAL_FILTER_1	Sampling clock for digital filter = measuring frequency.
CAC_REF_DIGITAL_FILTER_4	Sampling clock for digital filter = measuring frequency/4.
CAC_REF_DIGITAL_FILTER_16	Sampling clock for digital filter = measuring frequency/16.

8.5.7.6 cac_ref_edge_t

`cac_ref_edge_t`

Detailed description

Enumeration of available edge detect settings for the reference clock.

Enumerated values

Name	Description
CAC_REF_EDGE_RISE	Rising edge detect for the Reference clock.
CAC_REF_EDGE_FALL	Falling edge detect for the Reference clock.

Name	Description
CAC_REF_EDGE_BOTH	Both Rising and Falling edges detect for the Reference clock.

8.5.7.7 cac_meas_divider_t

```
cac_meas_divider_t
```

Detailed description

Enumeration of available dividers for the measurement clock

Enumerated values

Name	Description
CAC_MEAS_DIV_1	Measurement clock divided by 1.
CAC_MEAS_DIV_4	Measurement clock divided by 4.
CAC_MEAS_DIV_8	Measurement clock divided by 8.
CAC_MEAS_DIV_32	Measurement clock divided by 32.

8.5.7.8 cac_ctrl_t

```
typedef void cac_ctrl_t
```

Detailed description

CAC control block. Allocate an instance specific control block to pass into the CAC API calls. Implemented as

- cac_instance_ctrl_t

8.5.8 API Structures

8.5.8.1 cac_ref_clock_config_t

cac_ref_clock_config_t

Detailed description

Structure defining the settings that apply to reference clock configuration.

Variables

- cac_ref_divider_t divider

  Divider specification for the Reference clock.

- cac_clock_source_t clock

  Clock source for the Reference clock.
- cac_ref_digfilter_t digfilter

  Digital filter selection for the CACREF ext clock.
- cac_ref_edge_t edge

  Edge detection for the Reference clock.

8.5.8.2 cac_meas_clock_config_t

cac_meas_clock_config_t

Detailed description

Structure defining the settings that apply to measurement clock configuration.

Variables

- cac_meas_divider_t divider

  Divider specification for the Measurement clock.
- cac_clock_source_t clock

  Clock source for the Measurement clock.

8.5.8.3 cac_callback_args_t

cac_callback_args_t

Detailed description

Callback function parameter data

Variables

- cac_event_t event

  The event can be used to identify what caused the callback (cac ready or error).
- void const * p_context

  Placeholder for user data. Set in cac_api_t::open function in cac_cfg_t.

8.5.8.4 cac_cfg_t

cac_cfg_t

Detailed description

CAC Configuration

Variables

- cac_ref_clock_config_t cac_ref_clock

  reference clock specific settings

- cac_meas_clock_config_t cac_meas_clock
  measurement clock specific settings
- uint16_t cac_upper_limit
  the upper limit counter threshold
- uint16_t cac_lower_limit
  the lower limit counter threshold
- bool mei_interrupt_enabled
  True if Measurement Complete interrupt is enabled.
- bool ovf_interrupt_enabled
  True if Overflow interrupt is enabled.
- bool ferr_interrupt_enabled
  True if Frequency Error interrupt is enabled.
- bool continuous_mode
  True if measurement continuously restarts after completing.
- uint8_t frequency_error_ipl
  Frequency error interrupt priority.
- uint8_t measurement_end_ipl
  Measurement end interrupt priority.
- uint8_t overflow_ipl
  Overflow interrupt priority.
- void(* p_callback)(cac_callback_args_t *p_args)
  Callback provided when a CAC interrupt ISR occurs.
- void const * p_extend
  CAC hardware dependent configuration */.
- void const * p_context
  Placeholder for user data. Passed to user callback in cac_callback_args_t.

8.5.8.5 cac_api_t

cac_api_t

Detailed description

CAC functions implemented at the HAL layer API

8.5.8.6 open

```
ssp_err_t(* cac_api_t::open) (cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)
```

Detailed description

Open function for CAC device.

Table 1039:Parameters

Name	Direction	Description
p_ctrl	out	Pointer to CAC device control. Must be declared by user. Value set here.
cac_cfg_t	in	Pointer to CAC configuration structure. All elements of this structure must be set by user.

Parameter p_ctrl

Definition: `cac_ctrl_t*const p_ctrl`

CAC control block. Allocate an instance specific control block to pass into the CAC API calls. Implemented ascac_instance_ctrl_t

Parameter cac_cfg_t

Definition: `cac_cfg_t const *const p_cfg`

CAC Configuration

- `cac_cfg_t::cac_ref_clock_config_t`

  reference clock specific settings
- `cac_cfg_t::cac_meas_clock_config_t`

  measurement clock specific settings
- `cac_cfg_t::cac_upper_limit`

  the upper limit counter threshold
- `cac_cfg_t::cac_lower_limit`

  the lower limit counter threshold
- `cac_cfg_t::mei_interrupt_enabled`

  True if Measurement Complete interrupt is enabled.
- `cac_cfg_t::ovf_interrupt_enabled`

  True if Overflow interrupt is enabled.
- `cac_cfg_t::ferr_interrupt_enabled`

  True if Frequency Error interrupt is enabled.

- cac_cfg_t::continuous_mode

  True if measurement continuously restarts after completing.

- cac_cfg_t::frequency_error_ipl

  Frequency error interrupt priority.

- cac_cfg_t::measurement_end_ipl

  Measurement end interrupt priority.

- cac_cfg_t::overflow_ipl

  Overflow interrupt priority.

- cac_cfg_t::p_callback

  Callback provided when a CAC interrupt ISR occurs.

- cac_cfg_t::p_extend

  CAC hardware dependent configuration */.

- cac_cfg_t::p_context

  Placeholder for user data. Passed to user callback in cac_callback_args_t.

8.5.8.7 read

```
ssp_err_t(* cac_api_t::read) (cac_ctrl_t *const p_ctrl, uint8_t *const p_status,
uint16_t *const p_counter)
```

Detailed description

Read function for CAC peripheral.

Table 1040:Parameters

Name	Direction	Description
p_ctrl	in	Control for the CAC device context.
p_status	in	Pointer to variable in which to store the current CASTR register contents.
p_counter	in	Pointer to variable in which to store the current CACNTBR register contents.

Parameter p_ctrl

Definition: cac_ctrl_t*const p_ctrl

CAC control block. Allocate an instance specific control block to pass into the CAC API calls. Implemented ascac_instance_ctrl_t

Parameter p_status

uint8_t

Parameter p_counter

uint16_t

8.5.8.8 close

```
ssp_err_t(* cac_api_t::close) (cac_ctrl_t *const p_ctrl)
```

Detailed description

Close function for CAC device.

Table 1041:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to CAC device control.

Parameter p_ctrl

Definition: cac_ctrl_t*const p_ctrl

CAC control block. Allocate an instance specific control block to pass into the CAC API calls. Implemented ascac_instance_ctrl_t

8.5.8.9 stopMeasurement

```
ssp_err_t(* cac_api_t::stopMeasurement) (cac_ctrl_t *const p_ctrl)
```

Detailed description

End a measurement for the CAC peripheral.

Table 1042:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to CAC device control.

Parameter p_ctrl

Definition: cac_ctrl_t*const p_ctrl

CAC control block. Allocate an instance specific control block to pass into the CAC API calls. Implemented ascac_instance_ctrl_t

8.5.8.10 startMeasurement

```
ssp_err_t(* cac_api_t::startMeasurement) (cac_ctrl_t *const p_ctrl)
```

Detailed description

Begin a measurement for the CAC peripheral.

Table 1043:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to CAC device control.

Parameter p_ctrl

Definition: `cac_ctrl_t*const p_ctrl`

CAC control block. Allocate an instance specific control block to pass into the CAC API calls. Implemented ascac_instance_ctrl_t

8.5.8.11 reset

```
ssp_err_t(* cac_api_t::reset) (cac_ctrl_t *const p_ctrl)
```

Detailed description

Reset function for CAC device.

Table 1044:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to CAC device control.

Parameter p_ctrl

Definition: `cac_ctrl_t*const p_ctrl`

CAC control block. Allocate an instance specific control block to pass into the CAC API calls. Implemented ascac_instance_ctrl_t

8.5.8.12 versionGet

```
ssp_err_t(* cac_api_t::versionGet) (ssp_version_t *p_version)
```

Detailed description

Get the CAC API and code version information.

Table 1045:Parameters

Name	Direction	Description
p_version	out	is value returned.

Parameter p_version

8.5.8.13 cac_instance_t

cac_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- cac_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- cac_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- cac_api_t const * p_api

  Pointer to the API structure for this instance.

8.6 CAN Interface

Interface for CAN peripheral.

8.6.1 Summary

The CAN interface provides common APIs for CAN HAL drivers. CAN interface supports following features.

- Full-duplex CAN communication
- Generic CAN parameter setting
- Interrupt driven transmit/receive processing
- Callback function support with returning event code
- Hardware resource locking during a transaction

8.6.2 Interface API

can_api_t

Function name	Description
.open	Open function for CAN device
.read	Read function for CAN device, non-Blocking.
.write	Write function for CAN device

Function name	Description
.close	Close function for CAN device
.control	Control function for CAN device
.infoGet	Get CAN channel info.
.versionGet	Version get function for CAN device

8.6.3 Data structures

- can_status_t
- can_error_t
- can_info_t
- can_callback_args_t
- can_bit_timing_cfg_t
- can_frame_t
- can_mailbox_t
- can_cfg_t
- can_instance_t

8.6.4 Enumerations

- can_event_t
- can_mode_t
- can_id_mode_t
- can_frame_type_t
- can_message_mode_t
- can_clock_source_t
- can_time_segment1_t
- can_time_segment2_t
- can_sync_jump_width_t
- can_mailbox_send_receive_t
- can_command_t

8.6.5 Typedefs

- can_id_t
- can_ctrl_t

8.6.6 Defines

- #define CAN_API_VERSION_MAJOR

  Initial value: `(1U)`
- #define CAN_API_VERSION_MINOR

  Initial value: `(2U)`

8.6.7 API Data

8.6.7.1 can_event_t

`can_event_t`

Detailed description

CAN event codes

Enumerated values

Name	Description
CAN_EVENT_RX_COMPLETE	Receive complete event.
CAN_EVENT_TX_COMPLETE	Transmit complete event.
CAN_EVENT_ERR_BUS_OFF	Bus Off event.
CAN_EVENT_BUS_RECOVERY	Bus Off Recovery event.
CAN_EVENT_ERR_PASSIVE	Error Passive event.
CAN_EVENT_ERR_WARNING	Error Warning event.
CAN_EVENT_MAILBOX_OVERWRITE_OVERRUN	DEPRECATED, Mailbox has been overrun. This event is not used when the mailbox is overwritten.
CAN_EVENT_MAILBOX_OVERRUN	Mailbox has been overrun.

8.6.7.2 can_mode_t

`can_mode_t`

Detailed description

CAN Operation modes

Enumerated values

Name	Description
CAN_MODE_NORMAL	CAN Normal Mode.
CAN_MODE_HALT	CAN Halt Mode.
CAN_MODE_SLEEP	CAN SLEEP Mode.
CAN_MODE_EXIT_SLEEP	CAN Exit SLEEP Mode.
CAN_MODE_RESET	CAN Reset Mode.
CAN_MODE_LISTEN	CAN Listen Mode.
CAN_MODE_LOOPBACK_INTERNAL	CAN Internal Loopback Mode.
CAN_MODE_LOOPBACK_EXTERNAL	CAN External Loopback Mode.

8.6.7.3 can_id_mode_t

`can_id_mode_t`

Detailed description

CAN ID modes

Enumerated values

Name	Description
CAN_ID_MODE_STANDARD	Standard IDs of 11 bits used.
CAN_ID_MODE_EXTENDED	Extended IDs of 29 bits used.

8.6.7.4 can_frame_type_t

`can_frame_type_t`

Detailed description

CAN frame types

Enumerated values

Name	Description
CAN_FRAME_TYPE_DATA	Data frame type.
CAN_FRAME_TYPE_REMOTE	Remote frame type.

8.6.7.5 can_message_mode_t

`can_message_mode_t`

Detailed description

CAN Message Modes

Enumerated values

Name	Description
CAN_MESSAGE_MODE_OVERWRITE	Receive data will be overwritten if not read before the next frame.
CAN_MESSAGE_MODE_OVERRUN	Receive data will be retained until it is read.

8.6.7.6 can_clock_source_t

`can_clock_source_t`

Detailed description

CAN Source Clock

Enumerated values

Name	Description
CAN_CLOCK_SOURCE_PCLKB	PCLB is the source of the CAN Clock.
CAN_CLOCK_SOURCE_CANMCLK	CANMCLK is the source of the CAN Clock.

8.6.7.7 can_time_segment1_t

`can_time_segment1_t`

Detailed description

CAN Time Segment 1 Time Quanta

Enumerated values

Name	Description
CAN_TIME_SEGMENT1_TQ4	Time Segment 1 setting for 4 Time Quanta.
CAN_TIME_SEGMENT1_TQ5	Time Segment 1 setting for 5 Time Quanta.
CAN_TIME_SEGMENT1_TQ6	Time Segment 1 setting for 6 Time Quanta.
CAN_TIME_SEGMENT1_TQ7	Time Segment 1 setting for 7 Time Quanta.
CAN_TIME_SEGMENT1_TQ8	Time Segment 1 setting for 8 Time Quanta.
CAN_TIME_SEGMENT1_TQ9	Time Segment 1 setting for 9 Time Quanta.
CAN_TIME_SEGMENT1_TQ10	Time Segment 1 setting for 10 Time Quanta.
CAN_TIME_SEGMENT1_TQ11	Time Segment 1 setting for 11 Time Quanta.
CAN_TIME_SEGMENT1_TQ12	Time Segment 1 setting for 12 Time Quanta.
CAN_TIME_SEGMENT1_TQ13	Time Segment 1 setting for 13 Time Quanta.
CAN_TIME_SEGMENT1_TQ14	Time Segment 1 setting for 14 Time Quanta.
CAN_TIME_SEGMENT1_TQ15	Time Segment 1 setting for 15 Time Quanta.
CAN_TIME_SEGMENT1_TQ16	Time Segment 1 setting for 16 Time Quanta.

8.6.7.8 can_time_segment2_t

```
can_time_segment2_t
```

Detailed description

CAN Time Segment 2 Time Quanta

Enumerated values

Name	Description
CAN_TIME_SEGMENT2_TQ2	Time Segment 2 setting for 2 Time Quanta.
CAN_TIME_SEGMENT2_TQ3	Time Segment 2 setting for 3 Time Quanta.
CAN_TIME_SEGMENT2_TQ4	Time Segment 2 setting for 4 Time Quanta.

Name	Description
CAN_TIME_SEGMENT2_TQ5	Time Segment 2 setting for 5 Time Quanta.
CAN_TIME_SEGMENT2_TQ6	Time Segment 2 setting for 6 Time Quanta.
CAN_TIME_SEGMENT2_TQ7	Time Segment 2 setting for 7 Time Quanta.
CAN_TIME_SEGMENT2_TQ8	Time Segment 2 setting for 8 Time Quanta.

8.6.7.9 can_sync_jump_width_t

`can_sync_jump_width_t`

Detailed description

CAN Synchronization Jump Width Time Quanta

Enumerated values

Name	Description
CAN_SYNC_JUMP_WIDTH_TQ1	Synchronization Jump Width setting for 1 Time Quanta.
CAN_SYNC_JUMP_WIDTH_TQ2	Synchronization Jump Width setting for 2 Time Quanta.
CAN_SYNC_JUMP_WIDTH_TQ3	Synchronization Jump Width setting for 3 Time Quanta.
CAN_SYNC_JUMP_WIDTH_TQ4	Synchronization Jump Width setting for 4 Time Quanta.

8.6.7.10 can_mailbox_send_receive_t

`can_mailbox_send_receive_t`

Detailed description

CAN Mailbox type

Enumerated values

Name	Description
CAN_MAILBOX_RECEIVE	Mailbox is for receiving.
CAN_MAILBOX_TRANSMIT	Mailbox is for sending.

8.6.7.11 can_command_t

```
can_command_t
```

Detailed description

CAN control commands.

Enumerated values

Name	Description
CAN_COMMAND_MODE_SWITCH	Switch CAN operating mode..

8.6.7.12 can_id_t

```
typedef uint32_t can_id_t
```

Detailed description

CAN Id

8.6.7.13 can_ctrl_t

```
typedef void can_ctrl_t
```

Detailed description

CAN control block. Allocate an instance specific control block to pass into the CAN API calls. Implemented as

- can_instance_ctrl_t

8.6.8 API Structures

8.6.8.1 can_status_t

Detailed description

CAN Status

Variables

status

status_b

8.6.8.2 status

```
uint32_t can_status_t::status
```

8.6.8.3 status_b

```
struct can_status_t::st_status_b can_status_t::status_b
```

8.6.8.4 can_error_t

Detailed description

CAN Error Code

Variables

error

error_b

8.6.8.5 error

```
uint32_t can_error_t::error
```

8.6.8.6 error_b

```
struct can_error_t::st_error_b can_error_t::error_b
```

8.6.8.7 can_info_t

can_info_t

Detailed description

Variables

- can_mode_t operation_mode

  Can operation mode.
- can_status_t status

  CAN status.
- uint32_t bit_rate

  CAN bit rate.
- uint8_t error_count_transmit

  Transmit error count.
- uint8_t error_count_receive

  Receive error count.
- can_error_t error_code

  Error code, cleared after reading.

8.6.8.8 can_callback_args_t

can_callback_args_t

Detailed description

CAN callback parameter definition

Variables

- uint32_t channel

  Device channel number.

- can_event_t event

  Event code.

- uint32_t mailbox

  Mailbox number of interrupt source.

- void const * p_context

  Context provided to user during callback.

8.6.8.9 can_bit_timing_cfg_t

can_bit_timing_cfg_t

Detailed description

CAN bit rate configuration.

Variables

- uint32_t baud_rate_prescaler

  Baud rate prescaler. Valid values: 1 - 1024.

- can_time_segment1_t time_segment_1

  Time segment 1 control.

- can_time_segment2_t time_segment_2

  Time segment 2 control.

- can_sync_jump_width_t synchronization_jump_width

  Synchronization jump width.

8.6.8.10 can_frame_t

can_frame_t

Detailed description

CAN data Frame

Variables

- can_id_t id

  CAN id.
- uint8_t data_length_code

  CAN Data Length code, number of bytes in the message.
- uint8_t data[8]

  CAN data, up to 8 bytes.
- can_frame_type_t type

  Frame type, data or remote frame.

8.6.8.11 can_mailbox_t

can_mailbox_t

Detailed description

CAN Mailbox

Variables

- can_id_t mailbox_id

  Mailbox ID.
- can_mailbox_send_receive_t mailbox_type

  Receive or Transmit mailbox type.
- can_frame_type_t frame_type

  Frame type for receive mailbox.

8.6.8.12 can_cfg_t

can_cfg_t

Detailed description

CAN Configuration

Variables

- uint32_t channel

  CAN channel.
- can_bit_timing_cfg_t * p_bit_timing

  CAN bit timing.
- can_id_mode_t id_mode

  Standard or Extended ID mode.
- uint32_t mailbox_count

  Number of mailboxes.

- can_mailbox_t * p_mailbox

  Pointer to mailboxes.
- can_message_mode_t message_mode

  Overwrite message or overrun.
- void(* p_callback)(can_callback_args_t *p_args)

  Pointer to callback function.
- void const * p_context

  User defined callback context.
- void const * p_extend

  CAN hardware dependent configuration.
- uint8_t error_ipl

  Error interrupt priority.
- uint8_t mailbox_rx_ipl

  Receive interrupt priority.
- uint8_t mailbox_tx_ipl

  Transmit interrupt priority.

8.6.8.13 can_api_t

can_api_t

Detailed description

Shared Interface definition for CAN

8.6.8.14 open

```
ssp_err_t(* can_api_t::open) (can_ctrl_t *const p_ctrl, can_cfg_t const *const
p_cfg)
```

Detailed description

Open function for CAN device Implemented as

- R_CAN_Open

Table 1046:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the CAN control block Must be declared by user. Value set here.

Table 1046:Parameters (Continued)

Name	Direction	Description
can_cfg_t	in	Pointer to CAN configuration structure. All elements of this structure must be set by user.

Parameter p_ctrl

Definition: `can_ctrl_t*const p_ctrl`

CAN control block. Allocate an instance specific control block to pass into the CAN API calls. Implemented ascan_instance_ctrl_t

Parameter can_cfg_t

Definition: `can_cfg_t const *const p_cfg`

CAN Configuration

- `can_cfg_t`::`channel`

  CAN channel.

- `can_cfg_t`::`can_bit_timing_cfg_t`

  CAN bit timing.

- `can_cfg_t`::`can_id_mode_t`

  Standard or Extended ID mode.

  Enumerated as:

  – CAN_ID_MODE_STANDARD

  – CAN_ID_MODE_EXTENDED

- `can_cfg_t`::`mailbox_count`

  Number of mailboxes.

- `can_cfg_t`::`can_mailbox_t`

  Pointer to mailboxes.

- `can_cfg_t`::`can_message_mode_t`

  Overwrite message or overrun.

  Enumerated as:

  – CAN_MESSAGE_MODE_OVERWRITE

  – CAN_MESSAGE_MODE_OVERRUN

- `can_cfg_t`::`p_callback`

  Pointer to callback function.

- can_cfg_t::p_context

  User defined callback context.

- can_cfg_t::p_extend

  CAN hardware dependent configuration.

- can_cfg_t::error_ipl

  Error interrupt priority.

- can_cfg_t::mailbox_rx_ipl

  Receive interrupt priority.

- can_cfg_t::mailbox_tx_ipl

  Transmit interrupt priority.

8.6.8.15 read

```
ssp_err_t(* can_api_t::read) (can_ctrl_t *const p_ctrl, uint32_t mailbox,
can_frame_t *const p_frame)
```

Detailed description

Read function for CAN device, non-Blocking. Implemented as

- R_CAN_Read

Table 1047:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the CAN control block for the channel.
mailbox	in	Mailbox to read from.
p_frame	out	Pointer for frame of CAN ID, DLC, data and frame type.

Parameter p_ctrl

Definition: can_ctrl_t*const p_ctrl

CAN control block. Allocate an instance specific control block to pass into the CAN API calls. Implemented ascan_instance_ctrl_t

Parameter mailbox

uint32_t

Parameter p_frame

Definition: can_frame_t*const p_frame

CAN data Frame

- can_frame_t::id

  CAN id.

- can_frame_t::data_length_code

  CAN Data Length code, number of bytes in the message.

- can_frame_t::data

  CAN data, up to 8 bytes.

- can_frame_t::type

  Frame type, data or remote frame.

8.6.8.16 write

```
ssp_err_t(* can_api_t::write) (can_ctrl_t *const p_ctrl, uint32_t mailbox,
can_frame_t *const p_frame)
```

Detailed description

Write function for CAN device Implemented as

- R_CAN_Write

Table 1048:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the CAN control block.
mailbox	in	Mailbox to write to.
p_frame	in	Pointer for frame of CAN ID, DLC, data and frame type to write.

Parameter p_ctrl

Definition: can_ctrl_t*const p_ctrl

CAN control block. Allocate an instance specific control block to pass into the CAN API calls. Implemented ascan_instance_ctrl_t

Parameter mailbox

uint32_t

Parameter p_frame

Definition: can_frame_t*const p_frame

CAN data Frame

- can_frame_t::id

  CAN id.

- can_frame_t::data_length_code

  CAN Data Length code, number of bytes in the message.

- can_frame_t::data

  CAN data, up to 8 bytes.

- can_frame_t::type

  Frame type, data or remote frame.

8.6.8.17 close

```
ssp_err_t(* can_api_t::close) (can_ctrl_t *const p_ctrl)
```

Detailed description

Close function for CAN device Implemented as

- R_CAN_Close

Table 1049:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the CAN control block.

Parameter p_ctrl

Definition: can_ctrl_t*const p_ctrl

CAN control block. Allocate an instance specific control block to pass into the CAN API calls. Implemented ascan_instance_ctrl_t

8.6.8.18 control

```
ssp_err_t(* can_api_t::control) (can_ctrl_t *const p_ctrl, can_command_t const
command, void *p_data)
```

Detailed description

Control function for CAN device Implemented as

- R_CAN_Control

Table 1050:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the CAN control block.
command	in	Command type.

Table 1050:Parameters (Continued)

Name	Direction	Description
p_data	in	Command data.

Parameter p_ctrl

Definition: `can_ctrl_t*const p_ctrl`

CAN control block. Allocate an instance specific control block to pass into the CAN API calls. Implemented ascan_instance_ctrl_t

Parameter command
Parameter p_data

`const`

8.6.8.19 infoGet

```
ssp_err_t(* can_api_t::infoGet) (can_ctrl_t *const p_ctrl, can_info_t *const
p_info)
```

Detailed description

Get CAN channel info. Implemented as

- R_CAN_InfoGet

Table 1051:Parameters

Name	Direction	Description
p_ctrl	in	Handle for channel (pointer to channel control block)
p_info	out	Memory address to return channel specific data to.

Parameter p_ctrl

Definition: `can_ctrl_t*const p_ctrl`

CAN control block. Allocate an instance specific control block to pass into the CAN API calls. Implemented ascan_instance_ctrl_t

Parameter p_info

Definition: `can_info_t*const p_info`

- `can_info_t::operation_mode`

  Can operation mode.

- `can_info_t::status`

  CAN status.

- can_info_t::bit_rate

  CAN bit rate.
- can_info_t::error_count_transmit

  Transmit error count.
- can_info_t::error_count_receive

  Receive error count.
- can_info_t::error_code

  Error code, cleared after reading.

8.6.8.20 versionGet

```
ssp_err_t(* can_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Version get function for CAN device Implemented as

- R_CAN_VersionGet

Table 1052:Parameters

Name	Direction	Description
p_version	in	Pointer to the memory to store the version information

Parameter p_version

8.6.8.21 can_instance_t

can_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- can_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- can_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- can_api_t const * p_api

  Pointer to the API structure for this instance.

8.7 CGC Interface

Interface for clock generation.

8.7.1 Summary

The CGC interface provides the ability to configure and use all of the CGC module's capabilities. Among the capabilities is the selection of several clock sources to use as the system clock source. Additionally, the system clocks can be divided down to provide a wide range of frequencies for various system and peripheral needs.

Clock stability can be checked and clocks may also be stopped to save power when not needed. The API has a function to return the frequency of the system and system peripheral clocks at run time. There is also a feature to detect when the main oscillator has stopped, with the option of calling a user provided callback function.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

CGC Interface description: CGC Driver

8.7.2 Interface API

cgc_api_t

Function name	Description
.init	Initial configuration
.clocksCfg	Configure all system clocks.
.clockStart	Start a clock.
.clockStop	Stop a clock.
.systemClockSet	Set the system clock.
.systemClockGet	Get the system clock information.
.systemClockFreqGet	Return the frequency of the selected clock.
.clockCheck	Check the stability of the selected clock.
.oscStopDetect	Configure the Main Oscillator stop detection.
.oscStopStatusClear	Clear the oscillator stop detection flag.

Function name	Description
.busClockOutCfg	Configure the bus clock output secondary divider. The primary divider is set using the bsp clock configuration and the systemClockSet function (S7G2 and S3A7 only).
.busClockOutEnable	Enable the bus clock output (S7G2 and S3A7 only).
.busClockOutDisable	Disable the bus clock output (S7G2 and S3A7 only).
.clockOutCfg	Configure clockOut.
.clockOutEnable	Enable clock output on the CLKOUT pin. The source of the clock is controlled by clockOutCfg.
.clockOutDisable	Disable clock output on the CLKOUT pin. The source of the clock is controlled by clockOutCfg.
.lcdClockCfg	Configure the segment LCD Clock (S3A7 and S124 only).
.lcdClockEnable	Enable the LCD clock (S3A7 and S124 only).
.lcdClockDisable	Disables the LCD clock (S3A7 and S124 only).
.sdadcClockCfg	Configure the 24-bit Sigma-Delta A/D Converter Clock (S1JA only).
.sdadcClockEnable	Enable the SDADC clock (S1JA only).
.sdadcClockDisable	Disables the SDADC clock (S1JA only).
.sdramClockOutEnable	Enables the SDRAM clock output (S7G2 only).
.sdramClockOutDisable	Disables the SDRAM clock (S7G2 only).
.usbClockCfg	Configures the USB clock (S7G2 only).
.systickUpdate	Update the Systick timer.
.versionGet	Gets the CGC driver version.

8.7.3 Data structures

- cgc_callback_args_t
- cgc_clock_cfg_t
- cgc_system_clock_cfg_t
- cgc_clocks_cfg_t

- cgc_instance_t

8.7.4 Enumerations

- cgc_event_t
- cgc_clock_t
- cgc_pll_div_t
- cgc_sys_clock_div_t
- cgc_system_clocks_t
- cgc_clockout_dividers_t
- cgc_bclockout_dividers_t
- cgc_usb_clock_div_t
- cgc_systick_period_units_t
- cgc_clock_change_t

8.7.5 Defines

- #define CGC_API_VERSION_MAJOR
  Initial value: `(1U)`
- #define CGC_API_VERSION_MINOR
  Initial value: `(3U)`

8.7.6 API Data

8.7.6.1 cgc_event_t

`cgc_event_t`

Detailed description

Events that can trigger a callback function

Enumerated values

Name	Description
CGC_EVENT_OSC_STOP_DETECT	Oscillator stop detection has caused the event.

8.7.6.2 cgc_clock_t

```
cgc_clock_t
```

Detailed description

System clock source identifiers - The source of ICLK, BCLK, FCLK, PCLKS A-D and UCLK prior to the system clock divider

Enumerated values

Name	Description
CGC_CLOCK_HOCO	The high speed on chip oscillator.
CGC_CLOCK_MOCO	The middle speed on chip oscillator.
CGC_CLOCK_LOCO	The low speed on chip oscillator.
CGC_CLOCK_MAIN_OSC	The main oscillator.
CGC_CLOCK_SUBCLOCK	The subclock oscillator.
CGC_CLOCK_PLL	The PLL oscillator.

8.7.6.3 cgc_pll_div_t

```
cgc_pll_div_t
```

Detailed description

PLL divider values

Enumerated values

Name	Description
CGC_PLL_DIV_1	PLL divider of 1.
CGC_PLL_DIV_2	PLL divider of 2.
CGC_PLL_DIV_3	PLL divider of 3 (S7G2 only).
CGC_PLL_DIV_4	PLL divider of 4 (S3A7 only).

8.7.6.4 cgc_sys_clock_div_t

`cgc_sys_clock_div_t`

Detailed description

System clock divider values - The individually selectable divider of each of the system clocks, ICLK, BCLK, FCLK, PCLKS A-D

Enumerated values

Name	Description
CGC_SYS_CLOCK_DIV_1	System clock divided by 1.
CGC_SYS_CLOCK_DIV_2	System clock divided by 2.
CGC_SYS_CLOCK_DIV_4	System clock divided by 4.
CGC_SYS_CLOCK_DIV_8	System clock divided by 8.
CGC_SYS_CLOCK_DIV_16	System clock divided by 16.
CGC_SYS_CLOCK_DIV_32	System clock divided by 32.
CGC_SYS_CLOCK_DIV_64	System clock divided by 64.

8.7.6.5 cgc_system_clocks_t

`cgc_system_clocks_t`

Detailed description

System clock identifiers - Used as an input parameter to the systemClockFreqGet function.

Enumerated values

Name	Description
CGC_SYSTEM_CLOCKS_PCLKA	PCLKA - Peripheral module clock A.
CGC_SYSTEM_CLOCKS_PCLKB	PCLKB - Peripheral module clock B.
CGC_SYSTEM_CLOCKS_PCLKC	PCLKC - Peripheral module clock C.
CGC_SYSTEM_CLOCKS_PCLKD	PCLKD - Peripheral module clock D.
CGC_SYSTEM_CLOCKS_BCLK	BCLK - External bus Clock.

Name	Description
CGC_SYSTEM_CLOCKS_FCLK	FCLK - FlashIF clock.
CGC_SYSTEM_CLOCKS_ICLK	ICLK - System clock.

8.7.6.6 cgc_clockout_dividers_t

`cgc_clockout_dividers_t`

Detailed description

Divider values for the CLKOUT output.

Enumerated values

Name	Description
CGC_CLOCKOUT_DIV_1	Clockout source is divided by 1.
CGC_CLOCKOUT_DIV_2	Clockout source is divided by 2.
CGC_CLOCKOUT_DIV_4	Clockout source is divided by 4.
CGC_CLOCKOUT_DIV_8	Clockout source is divided by 8.
CGC_CLOCKOUT_DIV_16	Clockout source is divided by 16.
CGC_CLOCKOUT_DIV_32	Clockout source is divided by 32.
CGC_CLOCKOUT_DIV_64	Clockout source is divided by 64.
CGC_CLOCKOUT_DIV_128	Clockout source is divided by 128.

8.7.6.7 cgc_bclockout_dividers_t

`cgc_bclockout_dividers_t`

Detailed description

Divider values for the external bus clock output.

Enumerated values

Name	Description
CGC_BCLOCKOUT_DIV_1	External bus clock source is divided by 1.
CGC_BCLOCKOUT_DIV_2	External bus clock source is divided by 2.

8.7.6.8 cgc_usb_clock_div_t

`cgc_usb_clock_div_t`

Detailed description

USB clock divider values

Enumerated values

Name	Description
CGC_USB_CLOCK_DIV_3	Divide USB source clock by 3.
CGC_USB_CLOCK_DIV_4	Divide USB source clock by 4.
CGC_USB_CLOCK_DIV_5	Divide USB source clock by 5.

8.7.6.9 cgc_systick_period_units_t

`cgc_systick_period_units_t`

Detailed description

Available period units for R_CGC_SystickUpdate

Enumerated values

Name	Description
CGC_SYSTICK_PERIOD_UNITS_MILLISECONDS	Requested period in milliseconds.
CGC_SYSTICK_PERIOD_UNITS_MICROSECONDS	Requested period in microseconds.

8.7.6.10 cgc_clock_change_t

`cgc_clock_change_t`

Detailed description

Clock options

Enumerated values

Name	Description
CGC_CLOCK_CHANGE_NONE	No change to the clock.
CGC_CLOCK_CHANGE_STOP	Stop the clock.
CGC_CLOCK_CHANGE_START	Start the clock.

8.7.7 API Structures

8.7.7.1 cgc_callback_args_t

cgc_callback_args_t

Detailed description

Callback function parameter data

Variables

- cgc_event_t event

  The event can be used to identify what caused the callback.
- void const * p_context

  Placeholder for user data.

8.7.7.2 cgc_clock_cfg_t

cgc_clock_cfg_t

Detailed description

Clock configuration structure - Used as an input parameter to the clockStart function for the PLL clock.

Variables

- cgc_clock_t source_clock

  PLL source clock (S7G2 only).

- cgc_pll_div_t divider

  PLL divider.
- float multiplier

  PLL multiplier.

8.7.7.3 cgc_system_clock_cfg_t

cgc_system_clock_cfg_t

Detailed description

Clock configuration structure - Used as an input parameter to the systemClockSet and systemClockGet functions.

Variables

- cgc_sys_clock_div_t pclka_div

  Divider value for PCLKA.
- cgc_sys_clock_div_t pclkb_div

  Divider value for PCLKB.
- cgc_sys_clock_div_t pclkc_div

  Divider value for PCLKC.
- cgc_sys_clock_div_t pclkd_div

  Divider value for PCLKD.
- cgc_sys_clock_div_t bclk_div

  Divider value for BCLK.
- cgc_sys_clock_div_t fclk_div

  Divider value for FCLK.
- cgc_sys_clock_div_t iclk_div

  Divider value for ICLK.

8.7.7.4 cgc_clocks_cfg_t

cgc_clocks_cfg_t

Detailed description

Clock configuration

Variables

- cgc_clock_t system_clock

  System clock source enumeration.

- cgc_clock_cfg_t pll_cfg
  PLL configuration structure.
- cgc_system_clock_cfg_t sys_cfg
  Clock dividers structure.
- cgc_clock_change_t loco_state
  State of LOCO.
- cgc_clock_change_t moco_state
  State of MOCO.
- cgc_clock_change_t hoco_state
  State of HOCO.
- cgc_clock_change_t subosc_state
  State of Sub-oscillator.
- cgc_clock_change_t mainosc_state
  State of Main oscillator.
- cgc_clock_change_t pll_state
  State of PLL.

8.7.7.5 cgc_api_t

cgc_api_t

Detailed description

CGC functions implemented at the HAL layer follow this API.

8.7.7.6 init

```
ssp_err_t(* cgc_api_t::init) (void)
```

Detailed description

Initial configuration Implemented as

- R_CGC_Init

NOTE: The BSP module calls this function at startup. No further initialization is necessary.

8.7.7.7 clocksCfg

```
ssp_err_t(* cgc_api_t::clocksCfg) (cgc_clocks_cfg_t const *const p_clock_cfg)
```

Detailed description

Configure all system clocks. Implemented as

- R_CGC_ClocksCfg

NOTE: The BSP module calls this function at startup, but it can also be called from the application to change clocks at runtime.

8.7.7.8 clockStart

```
ssp_err_t(* cgc_api_t::clockStart) (cgc_clock_t clock_source, cgc_clock_cfg_t
*p_clock_cfg)
```

Detailed description

Start a clock. Implemented as

- R_CGC_ClockStart

NOTE: Clock to be started must not be running prior to calling this function or an error will be returned.

Table 1053:Parameters

Name	Direction	Description
clock_source	in	Clock source to initialize.
p_clock_cfg	in	Pointer to a structure that contains the dividers or multipliers to be used when configuring the PLL.

Parameter clock_source

Definition: `cgc_clock_tclock_source`

System clock source identifiers - The source of ICLK, BCLK, FCLK, PCLKS A-D and UCLK prior to the system clock divider

Parameter p_clock_cfg

Definition: `cgc_clock_cfg_t *p_clock_cfg`

Clock configuration structure - Used as an input parameter to the clockStart function for the PLL clock.

- `cgc_clock_cfg_t`::`cgc_clock_t`

  PLL source clock (S7G2 only).

  Enumerated as:

  - CGC_CLOCK_HOCO
  - CGC_CLOCK_MOCO
  - CGC_CLOCK_LOCO
  - CGC_CLOCK_MAIN_OSC
  - CGC_CLOCK_SUBCLOCK
  - CGC_CLOCK_PLL
- cgc_clock_cfg_t::cgc_pll_div_t

  PLL divider.

  Enumerated as:

  - CGC_PLL_DIV_1
  - CGC_PLL_DIV_2
  - CGC_PLL_DIV_3
  - CGC_PLL_DIV_4
- cgc_clock_cfg_t::multiplier

  PLL multiplier.

8.7.7.9 clockStop

```
ssp_err_t(* cgc_api_t::clockStop) (cgc_clock_t clock_source)
```

Detailed description

Stop a clock. Implemented as

- R_CGC_ClockStop

NOTE: Clock to be stopped must not be stopped prior to calling this function or an error will be returned.

Table 1054:Parameters

Name	Direction	Description
clock_source	in	The clock source to stop.

Parameter clock_source

Definition: cgc_clock_tclock_source

System clock source identifiers - The source of ICLK, BCLK, FCLK, PCLKS A-D and UCLK prior to the system clock divider

8.7.7.10 systemClockSet

```
ssp_err_t(* cgc_api_t::systemClockSet) (cgc_clock_t clock_source,
cgc_system_clock_cfg_t const *const p_clock_cfg)
```

Detailed description

Set the system clock. Implemented as

- R_CGC_SystemClockSet

NOTE: The clock to be set as the system clock must be running prior to calling this function.

Table 1055:Parameters

Name	Direction	Description
clock_source	in	Clock source to set as system clock
p_clock_cfg	in	Pointer to the clock dividers configuration passed by the caller.

Parameter clock_source

Definition: `cgc_clock_t`clock_source

System clock source identifiers - The source of ICLK, BCLK, FCLK, PCLKS A-D and UCLK prior to the system clock divider

Parameter p_clock_cfg

Definition: `cgc_system_clock_cfg_t const *const p_clock_cfg`

Clock configuration structure - Used as an input parameter to the systemClockSet and systemClockGet functions.

- `cgc_system_clock_cfg_t`::`cgc_sys_clock_div_t`

  Divider value for PCLKA.

  Enumerated as:

  – CGC_SYS_CLOCK_DIV_1

  – CGC_SYS_CLOCK_DIV_2

  – CGC_SYS_CLOCK_DIV_4

  – CGC_SYS_CLOCK_DIV_8

- CGC_SYS_CLOCK_DIV_16
- CGC_SYS_CLOCK_DIV_32
- CGC_SYS_CLOCK_DIV_64

- cgc_system_clock_cfg_t::cgc_sys_clock_div_t

  Divider value for PCLKB.

  Enumerated as:

  - CGC_SYS_CLOCK_DIV_1
  - CGC_SYS_CLOCK_DIV_2
  - CGC_SYS_CLOCK_DIV_4
  - CGC_SYS_CLOCK_DIV_8
  - CGC_SYS_CLOCK_DIV_16
  - CGC_SYS_CLOCK_DIV_32
  - CGC_SYS_CLOCK_DIV_64

- cgc_system_clock_cfg_t::cgc_sys_clock_div_t

  Divider value for PCLKC.

  Enumerated as:

  - CGC_SYS_CLOCK_DIV_1
  - CGC_SYS_CLOCK_DIV_2
  - CGC_SYS_CLOCK_DIV_4
  - CGC_SYS_CLOCK_DIV_8
  - CGC_SYS_CLOCK_DIV_16
  - CGC_SYS_CLOCK_DIV_32
  - CGC_SYS_CLOCK_DIV_64

- cgc_system_clock_cfg_t::cgc_sys_clock_div_t

  Divider value for PCLKD.

  Enumerated as:

  - CGC_SYS_CLOCK_DIV_1
  - CGC_SYS_CLOCK_DIV_2
  - CGC_SYS_CLOCK_DIV_4
  - CGC_SYS_CLOCK_DIV_8

- CGC_SYS_CLOCK_DIV_16
- CGC_SYS_CLOCK_DIV_32
- CGC_SYS_CLOCK_DIV_64

- cgc_system_clock_cfg_t::cgc_sys_clock_div_t

  Divider value for BCLK.

  Enumerated as:

  - CGC_SYS_CLOCK_DIV_1
  - CGC_SYS_CLOCK_DIV_2
  - CGC_SYS_CLOCK_DIV_4
  - CGC_SYS_CLOCK_DIV_8
  - CGC_SYS_CLOCK_DIV_16
  - CGC_SYS_CLOCK_DIV_32
  - CGC_SYS_CLOCK_DIV_64

- cgc_system_clock_cfg_t::cgc_sys_clock_div_t

  Divider value for FCLK.

  Enumerated as:

  - CGC_SYS_CLOCK_DIV_1
  - CGC_SYS_CLOCK_DIV_2
  - CGC_SYS_CLOCK_DIV_4
  - CGC_SYS_CLOCK_DIV_8
  - CGC_SYS_CLOCK_DIV_16
  - CGC_SYS_CLOCK_DIV_32
  - CGC_SYS_CLOCK_DIV_64

- cgc_system_clock_cfg_t::cgc_sys_clock_div_t

  Divider value for ICLK.

  Enumerated as:

  - CGC_SYS_CLOCK_DIV_1
  - CGC_SYS_CLOCK_DIV_2
  - CGC_SYS_CLOCK_DIV_4
  - CGC_SYS_CLOCK_DIV_8

- CGC_SYS_CLOCK_DIV_16
- CGC_SYS_CLOCK_DIV_32
- CGC_SYS_CLOCK_DIV_64

8.7.7.11 systemClockGet

```
ssp_err_t(* cgc_api_t::systemClockGet) (cgc_clock_t *p_clock_source,
cgc_system_clock_cfg_t *p_set_clock_cfg)
```

Detailed description

Get the system clock information. Implemented as

- R_CGC_SystemClockGet

Table 1056:Parameters

Name	Direction	Description
clock_source	out	Returns the current system clock.
p_clock_cfg	out	Returns the current system clock dividers.

Parameter clock_source

Parameter p_clock_cfg

8.7.7.12 systemClockFreqGet

```
ssp_err_t(* cgc_api_t::systemClockFreqGet) (cgc_system_clocks_t clock, uint32_t
*p_freq_hz)
```

Detailed description

Return the frequency of the selected clock. Implemented as

- R_CGC_SystemClockFreqGet

Table 1057:Parameters

Name	Direction	Description
clock	in	Specifies the internal clock whose frequency is returned.

Table 1057:Parameters (Continued)

Name	Direction	Description
p_freq_hz	out	Returns the frequency in Hz referenced by this pointer.

Parameter clock

Definition: `cgc_system_clocks_tclock`

System clock identifiers - Used as an input parameter to the cgc_api_t::systemClockFreqGet function.

Parameter p_freq_hz

`uint32_t`

8.7.7.13 clockCheck

```
ssp_err_t(* cgc_api_t::clockCheck) (cgc_clock_t clock_source)
```

Detailed description

Check the stability of the selected clock. Implemented as

- R_CGC_ClockCheck

Table 1058:Parameters

Name	Direction	Description
clock_source	in	Which clock source to check for stability.

Parameter clock_source

Definition: `cgc_clock_tclock_source`

System clock source identifiers - The source of ICLK, BCLK, FCLK, PCLKS A-D and UCLK prior to the system clock divider

8.7.7.14 oscStopDetect

```
ssp_err_t(* cgc_api_t::oscStopDetect) (void(*p_callback)(cgc_callback_args_t
*p_args), bool enable)
```

Detailed description

Configure the Main Oscillator stop detection. Implemented as

- R_CGC_OscStopDetect

Table 1059:Parameters

Name	Direction	Description
p_callback	in	Callback function that will be called by the NMI interrupt when an oscillation stop is detected. If the second argument is "false", then this argument can be NULL.
enable	in	Enable/disable Oscillation Stop Detection.

Parameter p_callback

Parameter enable

`const`

8.7.7.15 oscStopStatusClear

```
ssp_err_t(* cgc_api_t::oscStopStatusClear) (void)
```

Detailed description

Clear the oscillator stop detection flag. Implemented as

- R_CGC_OscStopStatusClear

8.7.7.16 busClockOutCfg

```
ssp_err_t(* cgc_api_t::busClockOutCfg) (cgc_bclockout_dividers_t divider)
```

Detailed description

Configure the bus clock output secondary divider. The primary divider is set using the bsp clock configuration and the systemClockSet function (S7G2 and S3A7 only).

Implemented as

- R_CGC_BusClockOutCfg

Table 1060:Parameters

Name	Direction	Description
divider	in	The divider of 1 or 2 of the clock source.

Parameter divider

Definition: cgc_bclockout_dividers_tdivider

Divider values for the external bus clock output.

8.7.7.17 busClockOutEnable

```
ssp_err_t(* cgc_api_t::busClockOutEnable) (void)
```

Detailed description

Enable the bus clock output (S7G2 and S3A7 only). Implemented as

- R_CGC_BusClockOutEnable

8.7.7.18 busClockOutDisable

```
ssp_err_t(* cgc_api_t::busClockOutDisable) (void)
```

Detailed description

Disable the bus clock output (S7G2 and S3A7 only). Implemented as

- R_CGC_BusClockOutDisable

8.7.7.19 clockOutCfg

```
ssp_err_t(* cgc_api_t::clockOutCfg) (cgc_clock_t clock, cgc_clockout_dividers_t
divider)
```

Detailed description

Configure clockOut. Implemented as

- R_CGC_ClockOutCfg

Table 1061:Parameters

Name	Direction	Description
clock	in	Clock source.
divider	in	Divider of between 1 and 128 of the clock source.

Parameter clock

Parameter divider

Definition: cgc_clockout_dividers_tdivider

Divider values for the CLKOUT output.

8.7.7.20 clockOutEnable

```
ssp_err_t(* cgc_api_t::clockOutEnable) (void)
```

Detailed description

Enable clock output on the CLKOUT pin. The source of the clock is controlled by clockOutCfg. Implemented as

- R_CGC_ClockOutEnable

8.7.7.21 clockOutDisable

```
ssp_err_t(* cgc_api_t::clockOutDisable) (void)
```

Detailed description

Disable clock output on the CLKOUT pin. The source of the clock is controlled by clockOutCfg. Implemented as

- R_CGC_ClockOutDisable

8.7.7.22 lcdClockCfg

```
ssp_err_t(* cgc_api_t::lcdClockCfg) (cgc_clock_t clock)
```

Detailed description

Configure the segment LCD Clock (S3A7 and S124 only). Implemented as

- R_CGC_LCDClockCfg

Table 1062:Parameters

Name	Direction	Description
clock	in	Segment LCD clock source.

Parameter clock

8.7.7.23 lcdClockEnable

```
ssp_err_t(* cgc_api_t::lcdClockEnable) (void)
```

Detailed description

Enable the LCD clock (S3A7 and S124 only). Implemented as

- R_CGC_LCDClockEnable

8.7.7.24 lcdClockDisable

```
ssp_err_t(* cgc_api_t::lcdClockDisable) (void)
```

Detailed description

Disables the LCD clock (S3A7 and S124 only). Implemented as

- R_CGC_LCDClockDisable

8.7.7.25 sdadcClockCfg

```
ssp_err_t(* cgc_api_t::sdadcClockCfg) (cgc_clock_t clock)
```

Detailed description

Configure the 24-bit Sigma-Delta A/D Converter Clock (S1JA only). Implemented as

- R_CGC_SDADCClockCfg

Table 1063:Parameters

Name	Direction	Description
clock	in	SDADC clock source.

Parameter clock

8.7.7.26 sdadcClockEnable

```
ssp_err_t(* cgc_api_t::sdadcClockEnable) (void)
```

Detailed description

Enable the SDADC clock (S1JA only). Implemented as

- R_CGC_SDADCClockEnable

8.7.7.27 sdadcClockDisable

```
ssp_err_t(* cgc_api_t::sdadcClockDisable) (void)
```

Detailed description

Disables the SDADC clock (S1JA only). Implemented as

- R_CGC_SDADCClockDisable

8.7.7.28 sdramClockOutEnable

```
ssp_err_t(* cgc_api_t::sdramClockOutEnable) (void)
```

Detailed description

Enables the SDRAM clock output (S7G2 only). Implemented as

- R_CGC_SDRAMClockOutEnable

8.7.7.29 sdramClockOutDisable

```
ssp_err_t(* cgc_api_t::sdramClockOutDisable) (void)
```

Detailed description

Disables the SDRAM clock (S7G2 only). Implemented as

- R_CGC_SDRAMClockOutDisable

8.7.7.30 usbClockCfg

```
ssp_err_t(* cgc_api_t::usbClockCfg) (cgc_usb_clock_div_t divider)
```

Detailed description

Configures the USB clock (S7G2 only). Implemented as

- R_CGC_USBClockCfg

Table 1064:Parameters

Name	Direction	Description
divider	in	The divider of 3, 4 or 5, of the clock source.

Parameter divider

Definition: `cgc_usb_clock_div_t`divider

USB clock divider values

8.7.7.31 systickUpdate

```
ssp_err_t(* cgc_api_t::systickUpdate) (uint32_t period_count,
cgc_systick_period_units_t units)
```

Detailed description

Update the Systick timer. Implemented as

- R_CGC_SystickUpdate

Table 1065:Parameters

Name	Direction	Description
period_count	in	The duration for the systick period.
units	in	The units for the provided period.

Parameter period_count

`uint32_t`

Parameter units

8.7.7.32 versionGet

```
ssp_err_t(* cgc_api_t::versionGet) (ssp_version_t *p_version)
```

Detailed description

Gets the CGC driver version. Implemented as

- R_CGC_VersionGet

Table 1066:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.7.7.33 cgc_instance_t

cgc_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- cgc_clock_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- cgc_api_t const * p_api

  Pointer to the API structure for this instance.

8.8 COMPARATOR Interface

Interface for Comparators.

8.8.1 Summary

The comparator interface provides standard comparator functionality, including generating an event when the comparator result changes.

Implemented by: High-Speed Analog ComparatorLow Power Analog Comparator

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

8.8.2 Interface API

comparator_api_t

Function name	Description
.open	Initialize the comparator.
.outputEnable	Start the comparator.
.infoGet	Provide information such as the recommended minimum stabilization wait time.
.statusGet	Provide current comparator status.
.close	Stop the comparator.
.versionGet	Retrieve the API version.

8.8.3 Data structures

- comparator_info_t
- comparator_status_t

- comparator_callback_args_t
- comparator_cfg_t
- comparator_instance_t

8.8.4 Enumerations

- comparator_mode_t
- comparator_trigger_t
- comparator_polarity_invert_t
- comparator_pin_output_t
- comparator_filter_t
- comparator_state_t

8.8.5 Typedefs

- comparator_ctrl_t

8.8.6 Defines

- #define COMPARATOR_API_VERSION_MAJOR

  Initial value: `(1U)`

  Includes board and MCU related header files. Version Number of API.
- #define COMPARATOR_API_VERSION_MINOR

  Initial value: `(0U)`

8.8.7 API Data

8.8.7.1 comparator_mode_t

`comparator_mode_t`

Detailed description

Select whether to invert the polarity of the comparator output.

Enumerated values

Name	Description
COMPARATOR_MODE_NORMAL	Normal mode.

Name	Description
COMPARATOR_MODE_WINDOW	Window mode, not supported by all implementations.

8.8.7.2 comparator_trigger_t

`comparator_trigger_t`

Detailed description

Trigger type: rising edge, falling edge, both edges, low level.

Enumerated values

Name	Description
COMPARATOR_TRIGGER_RISING	Rising edge trigger.
COMPARATOR_TRIGGER_FALLING	Falling edge trigger.
COMPARATOR_TRIGGER_BOTH_EDGE	Both edges trigger.

8.8.7.3 comparator_polarity_invert_t

`comparator_polarity_invert_t`

Detailed description

Select whether to invert the polarity of the comparator output.

Enumerated values

Name	Description
COMPARATOR_POLARITY_INVERT_OFF	Do not invert polarity.
COMPARATOR_POLARITY_INVERT_ON	Invert polarity.

8.8.7.4 comparator_pin_output_t

`comparator_pin_output_t`

Detailed description

Select whether to include the comparator output on the output pin.

Enumerated values

Name	Description
COMPARATOR_PIN_OUTPUT_OFF	Do not include comparator output on output pin.
COMPARATOR_PIN_OUTPUT_ON	Include comparator output on output pin.

8.8.7.5 comparator_filter_t

`comparator_filter_t`

Detailed description

Comparator digital filtering sample clock divisor settings.

Enumerated values

Name	Description
COMPARATOR_FILTER_OFF	Disable debounce filter.
COMPARATOR_FILTER_1	Filter using PCLK divided by 1, not supported by all implementations.
COMPARATOR_FILTER_8	Filter using PCLK divided by 8.
COMPARATOR_FILTER_16	Filter using PCLK divided by 16, not supported by all implementations.
COMPARATOR_FILTER_32	Filter using PCLK divided by 32.

8.8.7.6 comparator_state_t

`comparator_state_t`

Detailed description

Current comparator state.

Enumerated values

Name	Description
COMPARATOR_STATE_OUTPUT_DISABLED	comparator_api_t::outputEnable() has not been called
COMPARATOR_STATE_OUTPUT_LOW	VCMP < VREF if polarity is not inverted, VCMP > VREF if inverted.

Name	Description
COMPARATOR_STATE_OUTPUT_HIGH	VCMP > VREF if polarity is not inverted, VCMP < VREF if inverted.

8.8.7.7 comparator_ctrl_t

```
typedef void comparator_ctrl_t
```

Detailed description

Comparator control block. Allocate an instance specific control block to pass into the comparator API calls. Implemented as

- acmphs_instance_ctrl_t
- acmplp_instance_ctrl_t

8.8.8 API Structures

8.8.8.1 comparator_info_t

comparator_info_t

Detailed description

Comparator information.

Variables

- uint32_t min_stabilization_wait_us

  Minimum stabilization wait time in microseconds.

8.8.8.2 comparator_status_t

comparator_status_t

Detailed description

Comparator status.

Variables

- comparator_state_t state

  Current comparator state.

8.8.8.3 comparator_callback_args_t

comparator_callback_args_t

Detailed description

Callback function parameter data

Variables

- void const * p_context

  Placeholder for user data. Set in comparator_api_t::open function in comparator_cfg_t.
- uint32_t channel

  The physical hardware channel that caused the interrupt.

8.8.8.4 comparator_cfg_t

comparator_cfg_t

Detailed description

User configuration structure, used in open function

Variables

- uint8_t channel

  Hardware channel used.
- uint8_t irq_ipl

  Interrupt priority.
- comparator_mode_t mode

  Normal or window mode.
- comparator_trigger_t trigger

  Trigger setting.
- comparator_filter_t filter

  Digital filter clock divisor setting.
- comparator_polarity_invert_t invert

  Whether to invert output.
- comparator_pin_output_t pin_output

  Whether to include output on output pin.
- void(* p_callback)(comparator_callback_args_t *p_args)

  Callback called when comparator event occurs.
- void const * p_context

  Placeholder for user data. Passed to the user callback in comparator_callback_args_t.
- void const * p_extend

  Comparator hardware dependent configuration.

8.8.8.5 comparator_api_t

comparator_api_t

Detailed description

Comparator functions implemented at the HAL layer will follow this API.

8.8.8.6 open

```
ssp_err_t(* comparator_api_t::open) (comparator_ctrl_t *const p_ctrl,
comparator_cfg_t const *const p_cfg)
```

Detailed description

Initialize the comparator. Implemented as

- R_ACMPHS_Open
- R_ACMPLP_Open

Table 1067:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to instance control block
p_cfg	in	Pointer to configuration

Parameter p_ctrl

Definition: `comparator_ctrl_t*const p_ctrl`

Comparator control block. Allocate an instance specific control block to pass into the comparator API calls. Implemented asacmphs_instance_ctrl_tacmplp_instance_ctrl_t

Parameter p_cfg

Definition: `comparator_cfg_t const *const p_cfg`

User configuration structure, used in open function

- `comparator_cfg_t`::`channel`

  Hardware channel used.

- `comparator_cfg_t`::`irq_ipl`

  Interrupt priority.

- `comparator_cfg_t`::`comparator_mode_t`

  Normal or window mode.

  Enumerated as:

  – COMPARATOR_MODE_NORMAL

  – COMPARATOR_MODE_WINDOW

- comparator_cfg_t::comparator_trigger_t

  Trigger setting.

  Enumerated as:

  – COMPARATOR_TRIGGER_RISING

  – COMPARATOR_TRIGGER_FALLING

  – COMPARATOR_TRIGGER_BOTH_EDGE

- comparator_cfg_t::comparator_filter_t

  Digital filter clock divisor setting.

  Enumerated as:

  – COMPARATOR_FILTER_OFF

  – COMPARATOR_FILTER_1

  – COMPARATOR_FILTER_8

  – COMPARATOR_FILTER_16

  – COMPARATOR_FILTER_32

- comparator_cfg_t::comparator_polarity_invert_t

  Whether to invert output.

  Enumerated as:

  – COMPARATOR_POLARITY_INVERT_OFF

  – COMPARATOR_POLARITY_INVERT_ON

- comparator_cfg_t::comparator_pin_output_t

  Whether to include output on output pin.

  Enumerated as:

  – COMPARATOR_PIN_OUTPUT_OFF

  – COMPARATOR_PIN_OUTPUT_ON

- comparator_cfg_t::p_callback

  Callback called when comparator event occurs.

- comparator_cfg_t::p_context

  Placeholder for user data. Passed to the user callback in comparator_callback_args_t.

- comparator_cfg_t::p_extend

  Comparator hardware dependent configuration.

8.8.8.7 outputEnable

```
ssp_err_t(* comparator_api_t::outputEnable) (comparator_ctrl_t *const p_ctrl)
```

Detailed description

Start the comparator. Implemented as

- R_ACMPHS_OutputEnable
- R_ACMPLP_OutputEnable

Table 1068:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to instance control block

Parameter p_ctrl

Definition: `comparator_ctrl_t*const p_ctrl`

Comparator control block. Allocate an instance specific control block to pass into the comparator API calls. Implemented asacmphs_instance_ctrl_tacmplp_instance_ctrl_t

8.8.8.8 infoGet

```
ssp_err_t(* comparator_api_t::infoGet) (comparator_ctrl_t *const p_ctrl,
comparator_info_t *const p_info)
```

Detailed description

Provide information such as the recommended minimum stabilization wait time. Implemented as

- R_ACMPHS_InfoGet
- R_ACMPLP_InfoGet

Table 1069:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to instance control block
p_info	out	Comparator information stored here

Parameter p_ctrl

Definition: `comparator_ctrl_t*const p_ctrl`

Comparator control block. Allocate an instance specific control block to pass into the comparator API calls. Implemented asacmphs_instance_ctrl_tacmplp_instance_ctrl_t

Parameter p_info

Definition: `comparator_info_t*const p_info`

Comparator information.

- comparator_info_t::min_stabilization_wait_us

  Minimum stabilization wait time in microseconds.

8.8.8.9 statusGet

```
ssp_err_t(* comparator_api_t::statusGet) (comparator_ctrl_t *const p_ctrl,
comparator_status_t *const p_status)
```

Detailed description

Provide current comparator status. Implemented as

- R_ACMPHS_StatusGet
- R_ACMPLP_StatusGet

Table 1070:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to instance control block
p_status	out	Status stored here

Parameter p_ctrl

Definition: comparator_ctrl_t*const p_ctrl

Comparator control block. Allocate an instance specific control block to pass into the comparator API calls. Implemented asacmphs_instance_ctrl_tacmplp_instance_ctrl_t

Parameter p_status

Definition: comparator_status_t*const p_status

Comparator status.

- comparator_status_t::state

  Current comparator state.

8.8.8.10 close

```
ssp_err_t(* comparator_api_t::close) (comparator_ctrl_t *const p_ctrl)
```

Detailed description

Stop the comparator. Implemented as

- R_ACMPHS_Close
- R_ACMPLP_Close

Table 1071:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to instance control block

Parameter p_ctrl

Definition: `comparator_ctrl_t*const p_ctrl`

Comparator control block. Allocate an instance specific control block to pass into the comparator API calls. Implemented asacmphs_instance_ctrl_tacmplp_instance_ctrl_t

8.8.8.11 versionGet

```
ssp_err_t(* comparator_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Retrieve the API version. Implemented as

- R_ACMPHS_VersionGet
- R_ACMPLP_VersionGet

NOTE: This function retrieves the API version.

Table 1072:Parameters

Name	Direction	Description
p_version	in	Pointer to version structure

Parameter p_version

8.8.8.12 comparator_instance_t

comparator_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- comparator_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- comparator_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- comparator_api_t const * p_api

  Pointer to the API structure for this instance.

8.9 CRC Interface

Interface for cyclic redundancy checking.

8.9.1 Summary

The CRC (Cyclic Redundancy Check) calculator generates CRC codes using five different polynomials including 8 bit, 16 bit, and 32 bit variations. Calculation can be performed by sending data to the block using the CPU or by snooping on read or write activity on one of 10 SCI channels.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

CRC Interface description: CRC Driver

8.9.2 Interface API

crc_api_t

Function name	Description
.open	Open the CRC driver module.
.close	Close the CRC module driver
.crcResultGet	Return the current calculated value.
.snoopEnable	Enable snooping.
.snoopDisable	Disable snooping.
.snoopCfg	Configure the snoop channel and direction.
.calculate	Perform a CRC calculation on a block of data.

Function name	Description
.versionGet	Get the driver version based on compile time macros.

8.9.3 Data structures

- crc_cfg_t
- crc_snoop_cfg_t
- crc_instance_t

8.9.4 Enumerations

- crc_polynomial_t
- crc_bit_order_t
- crc_snoop_direction_t

8.9.5 Typedefs

- crc_ctrl_t

8.9.6 Defines

- #define CRC_API_VERSION_MAJOR
  Initial value: `(1U)`
- #define CRC_API_VERSION_MINOR
  Initial value: `(2U)`

8.9.7 API Data

8.9.7.1 crc_polynomial_t

`crc_polynomial_t`

Detailed description

CRC Generating Polynomial Switching (GPS).

Enumerated values

Name	Description
CRC_POLYNOMIAL_CRC_8	8-bit CRC-8 (X8 + X2 + X + 1)
CRC_POLYNOMIAL_CRC_16	16-bit CRC-16 (X16 + X15 + X2 + 1)
CRC_POLYNOMIAL_CRC_CCITT	16-bit CRC-CCITT (X16 + X12 + X5 + 1)
CRC_POLYNOMIAL_CRC_32	32-bit CRC-32 (X32 + X26 + X23 + X22 + X16 + X12 + X11 + X10 + X8 + X7 + X5 + X4 + X2 + X + 1)
CRC_POLYNOMIAL_CRC_32C	32-bit CRC-32C (X32 + X28 + X27 + X26 + X25 + X23 + X22 + X20 + X19 + X18 + X14 + X13 + X11 + X10 + X9 + X8 + X6 + 1)

8.9.7.2 crc_bit_order_t

`crc_bit_order_t`

Detailed description

CRC Calculation Switching (LMS)

Enumerated values

Name	Description
CRC_BIT_ORDER_LMS_LSB	Generates CRC for LSB first communication.
CRC_BIT_ORDER_LMS_MSB	Generates CRC for MSB first communication.

8.9.7.3 crc_snoop_direction_t

`crc_snoop_direction_t`

Detailed description

Snoop-On-Write/Read Switch (CRCSWR)

Enumerated values

Name	Description
CRC_SNOOP_DIRECTION_RECEIVE	Snoop-on-read.
CRC_SNOOP_DIRECTION_TRANSMIT	Snoop-on-write.

8.9.7.4 crc_ctrl_t

```
typedef void crc_ctrl_t
```

Detailed description

CRC control block. Allocate an instance specific control block to pass into the CRC API calls. Implemented as

- crc_instance_ctrl_t

8.9.8 API Structures

8.9.8.1 crc_cfg_t

crc_cfg_t

Detailed description

User configuration structure, used in open function

Variables

- crc_polynomial_t polynomial

  CRC Generating Polynomial Switching. (GPS)
- crc_bit_order_t bit_order

  CRC Calculation Switching (LMS)
- void const * p_extend

  CRC Hardware Dependent Configuration.

8.9.8.2 crc_snoop_cfg_t

crc_snoop_cfg_t

Detailed description

Snoop configuration

Variables

- uint32_t snoop_channel

  Register Snoop Address (CRCSA)
- crc_snoop_direction_t snoop_direction

  Snoop-On-Write/Read Switch (CRCSWR)

8.9.8.3 crc_api_t

crc_api_t

Detailed description

CRC driver structure. General CRC functions implemented at the HAL layer will follow this API.

8.9.8.4 open

```
ssp_err_t(* crc_api_t::open) (crc_ctrl_t *const p_ctrl, crc_cfg_t const *const
p_cfg)
```

Detailed description

Open the CRC driver module. Implemented as

- R_CRC_Open

Table 1073:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to CRC device handle.
p_cfg	in	Pointer to a configuration structure.

Parameter p_ctrl

Definition: `crc_ctrl_t*const p_ctrl`

CRC control block. Allocate an instance specific control block to pass into the CRC API calls. Implemented ascrc_instance_ctrl_t

Parameter p_cfg

Definition: `crc_cfg_t const *const p_cfg`

User configuration structure, used in open function

- `crc_cfg_t::crc_polynomial_t`

  CRC Generating Polynomial Switching. (GPS)

  Enumerated as:

  – CRC_POLYNOMIAL_CRC_8

  – CRC_POLYNOMIAL_CRC_16

  – CRC_POLYNOMIAL_CRC_CCITT

  – CRC_POLYNOMIAL_CRC_32

  – CRC_POLYNOMIAL_CRC_32C

- `crc_cfg_t::crc_bit_order_t`

  CRC Calculation Switching (LMS)

  Enumerated as:

  – CRC_BIT_ORDER_LMS_LSB

  – CRC_BIT_ORDER_LMS_MSB

- crc_cfg_t::p_extend

  CRC Hardware Dependent Configuration.

8.9.8.5 close

```
ssp_err_t(* crc_api_t::close) (crc_ctrl_t *const p_ctrl)
```

Detailed description

Close the CRC module driver Implemented as

- R_CRC_Close

Table 1074:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to crc device handle

Table 1075:Return values

Name	Description
SSP_SUCCESS	Configuration was successful.

Parameter p_ctrl

Definition: crc_ctrl_t*const p_ctrl

CRC control block. Allocate an instance specific control block to pass into the CRC API calls. Implemented ascrc_instance_ctrl_t

Parameter SSP_SUCCESS

8.9.8.6 crcResultGet

```
ssp_err_t(* crc_api_t::crcResultGet) (crc_ctrl_t *const p_ctrl, uint32_t
*crc_result)
```

Detailed description

Return the current calculated value. Implemented as

- R_CRC_CalculatedValueGet

Table 1076:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to CRC device handle.
crc_result	out	The calculated value from the last CRC calculation.

Parameter p_ctrl

Definition: `crc_ctrl_t*const p_ctrl`

CRC control block. Allocate an instance specific control block to pass into the CRC API calls. Implemented ascrc_instance_ctrl_t

Parameter crc_result

`uint32_t`

8.9.8.7 snoopEnable

```
ssp_err_t(* crc_api_t::snoopEnable) (crc_ctrl_t *const p_ctrl, uint32_t crc_seed)
```

Detailed description

Enable snooping. Implemented as

- R_CRC_SnoopEnable

Table 1077:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to CRC device handle.
crc_seed	in	CRC seed.

Parameter p_ctrl

Definition: `crc_ctrl_t*const p_ctrl`

CRC control block. Allocate an instance specific control block to pass into the CRC API calls. Implemented ascrc_instance_ctrl_t

Parameter crc_seed

`uint32_t`

8.9.8.8 snoopDisable

```
ssp_err_t(* crc_api_t::snoopDisable) (crc_ctrl_t *const p_ctrl)
```

Detailed description

Disable snooping. Implemented as

- R_CRC_SnoopDisable

Table 1078:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to crc device handle.

Parameter p_ctrl

Definition: `crc_ctrl_t*const p_ctrl`

CRC control block. Allocate an instance specific control block to pass into the CRC API calls. Implemented ascrc_instance_ctrl_t

8.9.8.9 snoopCfg

```
ssp_err_t(* crc_api_t::snoopCfg) (crc_ctrl_t *const p_ctrl, crc_snoop_cfg_t
*const p_snoop_cfg)
```

Detailed description

Configure the snoop channel and direction. Implemented as

- R_CRC_SnoopCfg

Table 1079:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to crc device handle.
p_snoopCfg	in	Snoop configuration.

Parameter p_ctrl

Definition: `crc_ctrl_t*const p_ctrl`

CRC control block. Allocate an instance specific control block to pass into the CRC API calls. Implemented ascrc_instance_ctrl_t

Parameter p_snoopCfg

8.9.8.10 calculate

```
ssp_err_t(* crc_api_t::calculate) (crc_ctrl_t *const p_ctrl, void
*p_input_buffer, uint32_t num_bytes, uint32_t crc_seed, uint32_t *p_crc_result)
```

Detailed description

Perform a CRC calculation on a block of data. Implemented as

- R_CRC_Calculate

Table 1080:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to crc device handle.
input_buffer	in	A pointer to an array of data values.
num_bytes	in	The number of bytes (not elements) in the array.
crc_seed	in	The seeded value for crc calculations.
crc_result	out	The calculated value of the CRC calculation.

Parameter p_ctrl

Definition: `crc_ctrl_t*const p_ctrl`

CRC control block. Allocate an instance specific control block to pass into the CRC API calls. Implemented ascrc_instance_ctrl_t

Parameter input_buffer
Parameter num_bytes

`uint32_t`

Parameter crc_seed

`uint32_t`

Parameter crc_result

8.9.8.11 versionGet

```
ssp_err_t(* crc_api_t::versionGet) (ssp_version_t *version)
```

Detailed description

Get the driver version based on compile time macros. Implemented as

- R_CRC_VersionGet

8.9.8.12 crc_instance_t

crc_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- crc_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- crc_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- crc_api_t const * p_api

  Pointer to the API structure for this instance.

8.10 CTSU Interface

Interface for Capacitive Touch Controllers.

8.10.1 Summary

The CTSU interface provides the functionality necessary to open, close, run and control the CTSU depending upon the configuration passed as arguments.

Implemented by: CTSU

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

CTSU Interface description: CTSU Driver

8.10.2 Interface API

ctsu_api_t

Function name	Description
.open	Initialize the CTSU; enable power and clock and set the register configuration.
.close	Close the CTSU by ending any scan in progress, disabling interrupts, and removing power to the peripheral and saving configurations according to options.
.scan	Start off a single CTSU scan.

Function name	Description
.update	Update the CTSU internal data including the touch decision and other derived data according to the results of the scan.
.read	Read the results from the CTSU including raw data, binary data and other derived data according to the selected options.
.versionGet	Retrieve the API version.

8.10.3 Data structures

- ctsu_callback_args_t
- ctsu_channel_pair_t
- ctsu_channel_setting_t
- ctsu_channel_data_self_t
- ctsu_channel_data_mutual_t
- ctsu_hw_cfg_t
- ctsu_functions_t
- ctsu_cfg_t
- ctsu_instance_t

8.10.4 Enumerations

- ctsu_event_t
- ctsu_read_t
- ctsu_process_option_t
- ctsu_close_option_t
- ctsu_action_t

8.10.5 Typedefs

- ctsu_ctrl_t

8.10.6 Defines

- #define CTSU_API_VERSION_MAJOR

  Initial value: `(1U)`

  Includes board and MCU related header files. Version of the API defined in this file
- #define CTSU_API_VERSION_MINOR

  Initial value: `(2U)`

8.10.7 API Data

8.10.7.1 ctsu_event_t

`ctsu_event_t`

Detailed description

CTSU callback event definitions

Enumerated values

Name	Description
CTSU_EVENT_SCAN_COMPLETE	Scan cycle is complete. This callback is occurring from an ISR.
CTSU_EVENT_PARAMETERS_UPDATED	All derived parameters from raw CTSU data are updated. Callback is not from ISR.

8.10.7.2 ctsu_read_t

`ctsu_read_t`

Detailed description

Different options with which a user can query the results of a CTSU scan. User cannot logic 'or' these together.

Enumerated values

Name	Description
CTSU_READ_BINARY_DATA_ALL	Read binary string of touch determination.
CTSU_READ_BINARY_DATA_SEL	Read binary string of touch determination for selected channels only.
CTSU_READ_RAW_SENSOR_OUTPUT_ALL	Read raw CTSU output for all active channels.

Name	Description
CTSU_READ_RAW_SENSOR_OUTPUT_SEL	Read raw CTSU output for selected channels.
CTSU_READ_FILTERED_SENSOR_VALUES_ALL	Read scount values output from filter for all active channels.
CTSU_READ_FILTERED_SENSOR_VALUES_SEL	Read scount values output from filter for all active channels.
CTSU_READ_DELTA_COUNT_ALL	Write out the difference between sensor and baseline count to p_dest array for all active sensors.
CTSU_READ_DELTA_COUNT_SEL	Write out the difference between sensor and baseline count to p_dest array for selected channels.
CTSU_READ_BASELINE_COUNT_ALL	Read the current sensor baseline used to determine if a channel is being touched for all active channels.
CTSU_READ_BASELINE_COUNT_SEL	Read the current sensor baseline used to determine if a channel is being touched for selected channels.
CTSU_READ_FILTERED_REF_ICO_VALUES_ALL	Read the rcount values output from filter for all active channels.
CTSU_READ_FILTERED_REF_ICO_VALUES_SEL	Read the rcount values output from filter for selected channels only.

8.10.7.3 ctsu_process_option_t

`ctsu_process_option_t`

Detailed description

Different options that can be provided to the Process function.

Enumerated values

Name	Description
CTSU_PROCESS_OPTION_DEFAULT_SETTING	Recommended option for most use cases.
CTSU_PROCESS_OPTION_AUTO_SCAN	Automatically start a scan after all internal variables have been updated.
CTSU_PROCESS_OPTION_ENABLE_DRIFT_COMP	Perform drift compensation and start a new scan when internal variables are updated.
CTSU_PROCESS_OPTION_ENABLE_AUTO_CALIB	Perform auto-calibration and start a new scan when internal variables are updated.

Name	Description
CTSU_PROCESS_OPTION_NONE	Skip all optional actions.

8.10.7.4 ctsu_close_option_t

`ctsu_close_option_t`

Detailed description

Options which can be passed to the Close function. User can logically 'OR' these together.

Enumerated values

Name	Description
CTSU_CLOSE_OPTION_SUSPEND	Suspend the CTSU by setting the CTSUSNZ bit.
CTSU_CLOSE_OPTION_SAVE_CONFIG	Save the configuration in the second argument provided to the close function.
CTSU_CLOSE_OPTION_RESET_SFRS	Bring the SFRs back to their P-O-RS values.
CTSU_CLOSE_OPTION_POWER_DOWN	Disable clock supply to the CTSU and set it into low power mode.

8.10.7.5 ctsu_action_t

`ctsu_action_t`

Detailed description

State of the Process/Update CTSU sensor data function

Enumerated values

Name	Description
CTSU_ACTION_START_NEW_SCAN	Start new scan.
CTSU_ACTION_WAITING_FOR_SCAN_COMPLETE	Wait for scan complete.
CTSU_ACTION_CHECK_FOR_ERRORS	Check for errors.
CTSU_ACTION_FILTER_DATA	Filter data.
CTSU_ACTION_UPDATE_TOUCH_INFO	Update touch info.
CTSU_ACTION_UPDATE_BUTTONS	Update buttons.

Name	Description
CTSU_ACTION_UPDATE_SLIDERS	Update sliders.
CTSU_ACTION_RUN_DRIFT_COMP	Run drift compensation.
CTSU_ACTION_RUN_AUTO_TUNE	Run auto tuning.
CTSU_ACTION_RESTART_STATE_MACHINE	Restart state machine.

8.10.7.6 ctsu_ctrl_t

```
typedef void ctsu_ctrl_t
```

Detailed description

CTSU control block. Allocate an instance specific control block to pass into the CTSU API calls. Implemented as

- ctsu_instance_ctrl_t

8.10.8 API Structures

8.10.8.1 ctsu_callback_args_t

ctsu_callback_args_t

Detailed description

CTSU callback arguments definitions

Variables

- ctsu_event_t event

  CTSU callback event.

- void const * p_context

  Placeholder for user data.

8.10.8.2 ctsu_channel_pair_t

ctsu_channel_pair_t

Detailed description

Structure storing information about a touch sensor element

Variables

- int8_t rx

  Denotes the primary channel.

- int8_t tx

  Denotes the secondary channel (used only for mutual capacitance mode)

8.10.8.3 ctsu_channel_setting_t

ctsu_channel_setting_t

Detailed description

Structure to be holding values to be written to the SFRs in the WR ISR

Variables

- uint16_t ctsussc

  Holds value for the CTSUSSC register.

- uint16_t ctsuso0

  Holds value for the CTSUSO0 register.

- uint16_t ctsuso1

  Holds value for the CTSUSO1 register.

8.10.8.4 ctsu_channel_data_self_t

ctsu_channel_data_self_t

Detailed description

Immediate raw data readings from the CTSU registers CTSUSC(sensor count) and CTSURC(Reference count) in self-capacitance mode.

Variables

- uint16_t sensor_count

  Raw sensor count.

- uint16_t reference_count

  Raw reference count.

8.10.8.5 ctsu_channel_data_mutual_t

ctsu_channel_data_mutual_t

Detailed description

Immediate raw data readings from the CTSU registers CTSUSC(sensor count) and CTSURC(Reference count) in mutual-capacitance mode.

Variables

- uint16_t sen_cnt_1

  Raw Sensor count primary reading.

- uint16_t ref_cnt_1

  Raw reference ICO count primary reading.

- uint16_t sen_cnt_2

  Raw sensor ICO count secondary reading.

- uint16_t ref_cnt_2

  Raw reference ICO count secondary reading.

8.10.8.6 ctsu_hw_cfg_t

ctsu_hw_cfg_t

Detailed description

Structure defining a hardware configuration to be passed to the Open function

Variables

- R_CTSU_Type ctsu_settings

  User defined SFR settings for CR0, CR1, SDPRS, SST, CHACn, CHTRCn, DCLKC.

- ctsu_channel_setting_t * write_settings

  User defined initial settings for thresholds for each active channel . Threshold is difference between runtime baseline and filtered output of sensor count.

  User defined settings for SSC, SO0, SO1 for each active channel.

- uint16_t * threshold
- uint16_t * hysteresis

  User defined settings for tolerance in count values.

- uint16_t * baseline

  A baseline of the expected count for each active channel when not touched.

- void * raw_result

  A pointer to a buffer which will hold raw results of CTSU measurement.

- void * filter_output

  A pointer to a buffer which will hold output after filtering raw results.

- void * binary_result

  A pointer to a location where binary data can be stored.

- ctsu_channel_pair_t * excluded

  A pointer to an array which contains a list of channel pairs which need to be ignored in ascending order of rx and then tx.

- int8_t num_excluded

  Number of elements in the array excluded.

- const uint16_t * series_resistance

  Resistance of the channel (to determine RC constant when tuning).

8.10.8.7 ctsu_functions_t

ctsu_functions_t

Detailed description

User defined functions that can be used to override the default processing functions used internally by the driver. For advanced usage only.

Variables

- int32_t(* preFilter)(void *p_args)

  Used for calculating raw data SNR before data gets filtered.

- int32_t(* filter)(volatile uint16_t *output, volatile uint16_t *input)

  Weighted averaging using CTSU_CFG_FILTER_DEPTH.

- int32_t(* postFilter)(void *p_args)

  Processing the filter results.

- int32_t(* ctsuDecode)(void *p_args)

  Algorithm to decide if channel is touched or not.

- int32_t(* otDriftComp)(void *p_args)

  Algorithm to perform manipulations to baseline, envelope, and thresholds on initialization.

- int32_t(* rtDriftComp)(void *p_args)

  Algorithm to perform run time manipulations to baseline, envelope, and thresholds.

- int32_t(* otAutoTune)(void *p_args)

  Function called once on initialization of system.

- int32_t(* rtAutoTune)(void *p_args)

  Function called to auto tune sensor when system is running.

8.10.8.8 ctsu_cfg_t

ctsu_cfg_t

Detailed description

Structure defining a configuration structure for the CTSU driver

Variables

- transfer_instance_t const *const p_lower_lvl_transfer_read

  Pointer to the Transfer instance to read results.

- transfer_instance_t const *const p_lower_lvl_transfer_write

  Pointer to the Transfer instance to write cfg.
- ctsu_hw_cfg_t * p_ctsu_hw_cfg

  Pointer to a CTSU configuration.
- ctsu_functions_t * p_ctsu_functions

  Pointer to a place holder for custom data functions.
- void(* p_callback)(ctsu_callback_args_t *p_args)

  Callback to the function to use when scan is complete.
- void * p_context

  Pointer to data that should be passed to update_complete notification.
- ctsu_process_option_t ctsu_soft_option

  Software options to use when performing Open and Process.
- ctsu_close_option_t ctsu_close_option

  Software options to use when closing touch operation.
- uint8_t write_ipl

  Write interrupt priority.
- uint8_t read_ipl

  Read interrupt priority.
- uint8_t end_ipl

  End interrupt priority.

8.10.8.9 ctsu_api_t

ctsu_api_t

Detailed description

CTSU HAL driver API structure. Functions implemented at the HAL layer will follow this API.

8.10.8.10 open

```
ssp_err_t(* ctsu_api_t::open) (ctsu_ctrl_t *p_ctrl, ctsu_cfg_t *p_cfg)
```

Detailed description

Initialize the CTSU; enable power and clock and set the register configuration. Implemented as

- R_CTSU_Open

Table 1081:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
p_cfg	in	Pointer to configuration structure

Parameter p_ctrl

Definition: `ctsu_ctrl_t*p_ctrl`

CTSU control block. Allocate an instance specific control block to pass into the CTSU API calls. Implemented asctsu_instance_ctrl_t

Parameter p_cfg

Definition: `ctsu_cfg_t *p_cfg`

Structure defining a configuration structure for the CTSU driver

- `ctsu_cfg_t`::`transfer_instance_t`

  Pointer to the Transfer instance to read results.

- `ctsu_cfg_t`::`transfer_instance_t`

  Pointer to the Transfer instance to write cfg.

- `ctsu_cfg_t`::`ctsu_hw_cfg_t`

  Pointer to a CTSU configuration.

- `ctsu_cfg_t`::`ctsu_functions_t`

  Pointer to a place holder for custom data functions.

- `ctsu_cfg_t`::`p_callback`

  Callback to the function to use when scan is complete.

- `ctsu_cfg_t`::`p_context`

  Pointer to data that should be passed to update_complete notification.

- `ctsu_cfg_t`::`ctsu_process_option_t`

  Software options to use when performing Open and Process.

  Enumerated as:

  – CTSU_PROCESS_OPTION_DEFAULT_SETTING

  – CTSU_PROCESS_OPTION_AUTO_SCAN

  – CTSU_PROCESS_OPTION_ENABLE_DRIFT_COMP

  – CTSU_PROCESS_OPTION_ENABLE_AUTO_CALIB

  – CTSU_PROCESS_OPTION_NONE

- ctsu_cfg_t::ctsu_close_option_t

  Software options to use when closing touch operation.

  Enumerated as:

  – CTSU_CLOSE_OPTION_SUSPEND

  – CTSU_CLOSE_OPTION_SAVE_CONFIG

  – CTSU_CLOSE_OPTION_RESET_SFRS

  – CTSU_CLOSE_OPTION_POWER_DOWN

- ctsu_cfg_t::write_ipl

  Write interrupt priority.

- ctsu_cfg_t::read_ipl

  Read interrupt priority.

- ctsu_cfg_t::end_ipl

  End interrupt priority.

8.10.8.11 close

```
ssp_err_t(* ctsu_api_t::close) (ctsu_ctrl_t *p_ctrl, ctsu_close_option_t opts)
```

Detailed description

Close the CTSU by ending any scan in progress, disabling interrupts, and removing power to the peripheral and saving configurations according to options. Implemented as

- R_CTSU_Close

Table 1082:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
opts	in	Closing options

Parameter p_ctrl

Definition: ctsu_ctrl_t*p_ctrl

CTSU control block. Allocate an instance specific control block to pass into the CTSU API calls. Implemented asctsu_instance_ctrl_t

Parameter opts

Definition: ctsu_close_option_topts

Options which can be passed to the Close function. User can logically 'OR' these together.

8.10.8.12 scan

```
ssp_err_t(* ctsu_api_t::scan) (ctsu_ctrl_t *p_ctrl)
```

Detailed description

Start off a single CTSU scan. Implemented as

- R_CTSU_Start_Scan

Table 1083:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure

Parameter p_ctrl

Definition: `ctsu_ctrl_t*p_ctrl`

CTSU control block. Allocate an instance specific control block to pass into the CTSU API calls. Implemented asctsu_instance_ctrl_t

8.10.8.13 update

```
ssp_err_t(* ctsu_api_t::update) (ctsu_ctrl_t *p_ctrl)
```

Detailed description

Update the CTSU internal data including the touch decision and other derived data according to the results of the scan. Implemented as

- R_CTSU_Update_Parameters

Table 1084:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure

Parameter p_ctrl

Definition: `ctsu_ctrl_t*p_ctrl`

CTSU control block. Allocate an instance specific control block to pass into the CTSU API calls. Implemented asctsu_instance_ctrl_t

8.10.8.14 read

```
ssp_err_t(* ctsu_api_t::read) (ctsu_ctrl_t *p_ctrl, void *p_dest, ctsu_read_t
opts, const ctsu_channel_pair_t *channels, const uint16_t count)
```

Detailed description

Read the results from the CTSU including raw data, binary data and other derived data according to the selected options. Implemented as

- R_CTSU_Read_Results

Table 1085:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
p_dest	out	Pointer to the destination location for the read data
opts	in	Read options, use only one option per call to read, do not logically OR values
channels	in	Specify the channel/channel pairs to read data for
count	in	Specify the number of channel/channel pairs to read data for

Parameter p_ctrl

Definition: `ctsu_ctrl_t*p_ctrl`

CTSU control block. Allocate an instance specific control block to pass into the CTSU API calls. Implemented asctsu_instance_ctrl_t

Parameter p_dest

`const`

Parameter opts

Definition: `ctsu_read_topts`

Different options with which a user can query the results of a CTSU scan. User cannot logic 'or' these together.

Parameter channels

Definition: `ctsu_channel_pair_tctsu_channel_pair_t *channels`

Structure storing information about a touch sensor element

- `ctsu_channel_pair_t::rx`

  Denotes the primary channel.

- `ctsu_channel_pair_t::tx`

  Denotes the secondary channel (used only for mutual capacitance mode)

Parameter count

`uint16_t`

8.10.8.15 versionGet

```
ssp_err_t(* ctsu_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Retrieve the API version. Implemented as

- R_CTSU_VersionGet()

NOTE: This function retrieves the API version.

Table 1086:Parameters

Name	Direction	Description
p_version	in	Pointer to version structure

Parameter p_version

8.10.8.16 ctsu_instance_t

ctsu_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- ctsu_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- ctsu_cfg_t * p_cfg

  Pointer to the configuration structure for this instance.
- ctsu_api_t const * p_api

  Pointer to the API structure for this instance.

8.11 Display Interface

Interface for LCD panel displays.

8.11.1 Summary

The display interface provides standard display functionality:

- Signal timing configuration for LCD panels with RGB interface.
- Dot clock source selection (internal or external) and frequency divider.
- Blending of multiple graphics layers on the background screen.
- Color correction (brightness/configuration/gamma correction).
- Interrupts and callback function.

Implemented by: GLCDC

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Display Interface description: GLCDC Display Driver

8.11.2 Interface API

display_api_t

Function name	Description
.open	Open display device.
.close	Close display device.
.start	Display start.
.stop	Display stop.
.layerChange	Change layer parameters at runtime.
.correction	Color correction.
.clut	Set CLUT for display device.
.statusGet	Get status for display device.
.versionGet	Get version.

8.11.3 Data structures

- display_timing_t
- display_color_t
- display_coordinate_t

- display_brightness_t
- display_contrast_t
- display_correction_t
- gamma_correction_t
- display_gamma_correction_t
- display_clut_t
- display_input_cfg_t
- display_output_cfg_t
- display_layer_t
- display_callback_args_t
- display_cfg_t
- display_runtime_cfg_t
- display_clut_cfg_t
- display_status_t
- display_instance_t

8.11.4 Enumerations

- display_frame_layer_t
- display_state_t
- display_event_t
- display_in_format_t
- display_out_format_t
- display_endian_t
- display_color_order_t
- display_signal_polarity_t
- display_sync_edge_t
- display_fade_control_t
- display_fade_status_t

8.11.5 Typedefs

- display_ctrl_t

8.11.6 Defines

- #define DISPLAY_API_VERSION_MAJOR

  Initial value: `(1U)`
- #define DISPLAY_API_VERSION_MINOR

  Initial value: `(2U)`
- #define DISPLAY_GAMMA_CURVE_ELEMENT_NUM

  Initial value: `(16)`

8.11.7 API Data

8.11.7.1 display_frame_layer_t

`display_frame_layer_t`

Detailed description

Display frame number

Enumerated values

Name	Description
DISPLAY_FRAME_LAYER_1	Frame layer 1.
DISPLAY_FRAME_LAYER_2	Frame layer 2.

8.11.7.2 display_state_t

`display_state_t`

Detailed description

Display interface operation state

Enumerated values

Name	Description
DISPLAY_STATE_CLOSED	Display closed.
DISPLAY_STATE_OPENED	Display opened.
DISPLAY_STATE_DISPLAYING	Displaying.

8.11.7.3 display_event_t

`display_event_t`

Detailed description

Display event codes

Enumerated values

Name	Description
DISPLAY_EVENT_GR1_UNDERFLOW	Graphics frame1 underflow occurs.
DISPLAY_EVENT_GR2_UNDERFLOW	Graphics frame2 underflow occurs.
DISPLAY_EVENT_LINE_DETECTION	Designated line is processed.

8.11.7.4 display_in_format_t

`display_in_format_t`

Detailed description

Input format setting

Enumerated values

Name	Description
DISPLAY_IN_FORMAT_32BITS_ARGB8888	ARGB8888, 32 bits.
DISPLAY_IN_FORMAT_32BITS_RGB888	RGB888, 32 bits.
DISPLAY_IN_FORMAT_16BITS_RGB565	RGB565, 16 bits.
DISPLAY_IN_FORMAT_16BITS_ARGB1555	ARGB1555, 16 bits.
DISPLAY_IN_FORMAT_16BITS_ARGB4444	ARGB4444, 16 bits.
DISPLAY_IN_FORMAT_CLUT8	CLUT8.
DISPLAY_IN_FORMAT_CLUT4	CLUT4.
DISPLAY_IN_FORMAT_CLUT1	CLUT1.

8.11.7.5 display_out_format_t

`display_out_format_t`

Detailed description

Output format setting

Enumerated values

Name	Description
DISPLAY_OUT_FORMAT_24BITS_RGB888	RGB888, 24 bits.
DISPLAY_OUT_FORMAT_18BITS_RGB666	RGB666, 18 bits.
DISPLAY_OUT_FORMAT_16BITS_RGB565	RGB565, 16 bits.
DISPLAY_OUT_FORMAT_8BITS_SERIAL	SERIAL, 8 bits.

8.11.7.6 display_endian_t

`display_endian_t`

Detailed description

Data endian select

Enumerated values

Name	Description
DISPLAY_ENDIAN_LITTLE	Little-endian.
DISPLAY_ENDIAN_BIG	Big-endian.

8.11.7.7 display_color_order_t

`display_color_order_t`

Detailed description

RGB color order select

Enumerated values

Name	Description
DISPLAY_COLOR_ORDER_RGB	Color order RGB.
DISPLAY_COLOR_ORDER_BGR	Color order BGR.

8.11.7.8 display_signal_polarity_t

`display_signal_polarity_t`

Detailed description

Polarity of a signal select

Enumerated values

Name	Description
DISPLAY_SIGNAL_POLARITY_LOACTIVE	Low active signal.
DISPLAY_SIGNAL_POLARITY_HIACTIVE	High active signal.

8.11.7.9 display_sync_edge_t

`display_sync_edge_t`

Detailed description

Signal synchronization edge select

Enumerated values

Name	Description
DISPLAY_SIGNAL_SYNC_EDGE_RISING	Signal is synchronized to rising edge.
DISPLAY_SIGNAL_SYNC_EDGE_FALLING	Signal is synchronized to falling edge.

8.11.7.10 display_fade_control_t

`display_fade_control_t`

Detailed description

Fading control

Enumerated values

Name	Description
DISPLAY_FADE_CONTROL_NONE	Applying no fading control.
DISPLAY_FADE_CONTROL_FADEIN	Applying fade-in control.
DISPLAY_FADE_CONTROL_FADEOUT	Applying fade-out control.

8.11.7.11 display_fade_status_t

`display_fade_status_t`

Detailed description

Fading status

Enumerated values

Name	Description
DISPLAY_FADE_STATUS_NOT_UNDERWAY	Fade-in/fade-out is not in progress.
DISPLAY_FADE_STATUS_FADING_UNDERWAY	Fade-in or fade-out is in progress.
DISPLAY_FADE_STATUS_UNCERTAIN	Fade-in/fade-out status is uncertain just before hardware working.

8.11.7.12 display_ctrl_t

`typedef void display_ctrl_t`

Detailed description

Display control block. Allocate an instance specific control block to pass into the display API calls. Implemented as

- glcd_instance_ctrl_tDisplay control block

8.11.8 API Structures

8.11.8.1 display_timing_t

display_timing_t

Detailed description

Display signal timing setting

Variables

- uint16_t total_cyc

  Total cycles in one line or total lines in one frame.
- uint16_t display_cyc

  Active video cycles or lines.
- uint16_t back_porch

  Back poach cycles or lines.
- uint16_t sync_width

  Sync signal asserting width.

- display_signal_polarity_t sync_polarity

  Sync signal polarity.

8.11.8.2 display_color_t

display_color_t

Detailed description

RGB Color setting

Variables

- uint32_t argb
- uint8_t b

  blue
- uint8_t g

  green
- uint8_t r

  red
- uint8_t a

  a
- struct{} byte

  See source code for definition of this member.
- union{} union{}

  See source code for definition of this member.

8.11.8.3 display_coordinate_t

display_coordinate_t

Detailed description

Contrast (gain) correction setting

Variables

- int16_t x

  Coordinate X, this allows to set signed value.
- int16_t y

  Coordinate Y, this allows to set signed value.

8.11.8.4 display_brightness_t

display_brightness_t

Detailed description

Brightness (DC) correction setting

Variables

- bool enable

  Brightness Correction On/Off.

- uint16_t r

  Brightness (DC) adjustment for R channel.

- uint16_t g

  Brightness (DC) adjustment for G channel.

- uint16_t b

  Brightness (DC) adjustment for B channel.

8.11.8.5 display_contrast_t

display_contrast_t

Detailed description

Contrast (gain) correction setting

Variables

- bool enable

  Contrast Correction On/Off.

- uint8_t r

  Contrast (gain) adjustment for R channel.

- uint8_t g

  Contrast (gain) adjustment for G channel.

- uint8_t b

  Contrast (gain) adjustment for B channel.

8.11.8.6 display_correction_t

display_correction_t

Detailed description

Color correction setting

Variables

- display_brightness_t brightness

  Brightness.

- display_contrast_t contrast

  Contrast.

8.11.8.7 gamma_correction_t

gamma_correction_t

Detailed description

Gamma correction setting for each color

Variables

- bool enable

  Gamma Correction On/Off.

- uint16_t gain[DISPLAY_GAMMA_CURVE_ELEMENT_NUM]

  Gain adjustment.

- uint16_t threshold[DISPLAY_GAMMA_CURVE_ELEMENT_NUM]

  Start threshold.

8.11.8.8 display_gamma_correction_t

display_gamma_correction_t

Detailed description

Gamma correction setting

Variables

- gamma_correction_t r

  Gamma correction for R channel.

- gamma_correction_t g

  Gamma correction for G channel.

- gamma_correction_t b

  Gamma correction for B channel.

8.11.8.9 display_clut_t

display_clut_t

Detailed description

CLUT setting

Variables

- uint32_t color_num

  The number of colors in CLUT.

- const uint32_t * p_clut

  Address of the area storing the CLUT data (in ARGB8888 format)

8.11.8.10 display_input_cfg_t

display_input_cfg_t

Detailed description

Graphics plane input configuration structure

Variables

- uint32_t * p_base

  Base address to the frame buffer.
- uint16_t hsize

  Horizontal pixel size in a line.
- uint16_t vsize

  Vertical pixel size in a frame.
- uint32_t hstride

  Memory stride (bytes) in a line.
- display_in_format_t format

  Input format setting.
- bool line_descending_enable

  Line descending enable.
- bool lines_repeat_enable

  Line repeat enable.
- uint16_t lines_repeat_times

  Expected number of line repeating.

8.11.8.11 display_output_cfg_t

display_output_cfg_t

Detailed description

Display output configuration structure

Variables

- display_timing_t htiming

  Horizontal display cycle setting.
- display_timing_t vtiming

  Vertical display cycle setting.

- display_out_format_t format

  Output format setting.
- display_endian_t endian

  Bit order of output data.
- display_color_order_t color_order

  Color order in pixel.
- display_signal_polarity_t data_enable_polarity

  Data Enable signal polarity.
- display_sync_edge_t sync_edge

  Signal sync edge selection.
- display_color_t bg_color

  Background color.
- display_brightness_t brightness

  Brightness setting.
- display_contrast_t contrast

  Contrast setting.
- display_gamma_correction_t * p_gamma_correction

  Pointer to gamma correction setting.
- bool dithering_on

  Dithering on/off.

8.11.8.12 display_layer_t

display_layer_t

Detailed description

Graphics layer blend setup parameter structure

Variables

- display_coordinate_t coordinate

  Blending location (starting point of image)
- display_color_t bg_color

  Color outside region.
- display_fade_control_t fade_control

  Layer fade-in/out control on/off.
- uint8_t fade_speed

  Layer fade-in/out frame rate.

8.11.8.13 display_callback_args_t

display_callback_args_t

Detailed description

Display callback parameter definition

Variables

- display_event_t event

  Event code.

- void const * p_context

  Context provided to user during callback.

8.11.8.14 display_cfg_t

display_cfg_t

Detailed description

Display main configuration structure

Variables

- display_input_cfg_t input[DISPLAY_FRAME_LAYER_2+1]

  Graphics input frame setting.

  Generic configuration for display devices

- display_output_cfg_t output

  Graphics output frame setting.

- display_layer_t layer[DISPLAY_FRAME_LAYER_2+1]

  Graphics layer blend setting.

- uint8_t line_detect_ipl

  Line detect interrupt priority.

- uint8_t underflow_1_ipl

  Underflow 1 interrupt priority.

- uint8_t underflow_2_ipl

  Underflow 1 interrupt priority.

- void(* p_callback)(display_callback_args_t *p_args)

  Pointer to callback function.

  Configuration for display event processing

- void const * p_context

  User defined context passed into callback function.

- void const * p_extend

  Display hardware dependent configuration.

  Pointer to display peripheral specific configuration

8.11.8.15 display_runtime_cfg_t

display_runtime_cfg_t

Detailed description

Display main configuration structure

Variables

- display_input_cfg_t input

  Graphics input frame setting.

  Generic configuration for display devices

- display_layer_t layer

  Graphics layer alpha blending setting.

8.11.8.16 display_clut_cfg_t

display_clut_cfg_t

Detailed description

Display CLUT configuration structure

Variables

- uint32_t * p_base

  Pointer to CLUT source data.

- uint16_t start

  Beginning of CLUT entry to be updated.

- uint16_t size

  Size of CLUT entry to be updated.

8.11.8.17 display_status_t

display_status_t

Detailed description

Display Status

Variables

- display_state_t state

  Status of GLCD module.

- display_fade_status_t fade_status[DISPLAY_FRAME_LAYER_2+1]

  Status of fade-in/fade-out status.

8.11.8.18 display_api_t

display_api_t

Detailed description

Shared Interface definition for display peripheral

8.11.8.19 open

```
ssp_err_t(* display_api_t::open) (display_ctrl_t *const p_ctrl, display_cfg_t
const *const p_cfg)
```

Detailed description

Open display device. Implemented as

- R_GLCD_Open

Table 1087:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to display interface control block. Must be declared by user. Value set here.
p_cfg	in	Pointer to display configuration structure. All elements of this structure must be set by user.

Parameter p_ctrl

Definition: `display_ctrl_t*const p_ctrl`

Display control block. Allocate an instance specific control block to pass into the display API calls. Implemented asglcd_instance_ctrl_tDisplay control block

Parameter p_cfg

Definition: `display_cfg_t const *const p_cfg`

Display main configuration structure

- display_cfg_t::display_input_cfg_t

  Graphics input frame setting.

  Generic configuration for display devices

- display_cfg_t::display_output_cfg_t

  Graphics output frame setting.

- display_cfg_t::display_layer_t

  Graphics layer blend setting.
- display_cfg_t::line_detect_ipl

  Line detect interrupt priority.
- display_cfg_t::underflow_1_ipl

  Underflow 1 interrupt priority.
- display_cfg_t::underflow_2_ipl

  Underflow 1 interrupt priority.
- display_cfg_t::p_callback

  Pointer to callback function.

  Configuration for display event processing
- display_cfg_t::p_context

  User defined context passed into callback function.
- display_cfg_t::p_extend

  Display hardware dependent configuration.

  Pointer to display peripheral specific configuration

8.11.8.20 close

```
ssp_err_t(* display_api_t::close) (display_ctrl_t *const p_ctrl)
```

Detailed description

Close display device. Implemented as

- R_GLCD_Close

Table 1088:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to display interface control block.

Parameter p_ctrl

Definition: display_ctrl_t*const p_ctrl

Display control block. Allocate an instance specific control block to pass into the display API calls. Implemented asglcd_instance_ctrl_tDisplay control block

8.11.8.21 start

```
ssp_err_t(* display_api_t::start) (display_ctrl_t *const p_ctrl)
```

Detailed description

Display start. Implemented as

- R_GLCD_Start

Table 1089:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to display interface control block.

Parameter p_ctrl

Definition: `display_ctrl_t*const p_ctrl`

Display control block. Allocate an instance specific control block to pass into the display API calls. Implemented asglcd_instance_ctrl_tDisplay control block

8.11.8.22 stop

```
ssp_err_t(* display_api_t::stop) (display_ctrl_t *const p_ctrl)
```

Detailed description

Display stop. Implemented as

- R_GLCD_Stop

Table 1090:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to display interface control block.

Parameter p_ctrl

Definition: `display_ctrl_t*const p_ctrl`

Display control block. Allocate an instance specific control block to pass into the display API calls. Implemented asglcd_instance_ctrl_tDisplay control block

8.11.8.23 layerChange

```
ssp_err_t(* display_api_t::layerChange) (display_ctrl_t const *const p_ctrl,
display_runtime_cfg_t const *const p_cfg, display_frame_layer_t frame)
```

Detailed description

Change layer parameters at runtime. Implemented as

- R_GLCD_LayerChange

Table 1091:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to display interface control block.
p_cfg	in	Pointer to run-time layer configuration structure.
frame	in	Number of graphic frames.

Parameter p_ctrl

Definition: `display_ctrl_tconst *const p_ctrl`

Display control block. Allocate an instance specific control block to pass into the display API calls. Implemented asglcd_instance_ctrl_tDisplay control block

Parameter p_cfg

Definition: `display_runtime_cfg_t const *const p_cfg`

Display main configuration structure

- `display_runtime_cfg_t`::`display_input_cfg_t`

  Graphics input frame setting.

  Generic configuration for display devices

- `display_runtime_cfg_t`::`display_layer_t`

  Graphics layer alpha blending setting.

Parameter frame

8.11.8.24 correction

```
ssp_err_t(* display_api_t::correction) (display_ctrl_t const *const p_ctrl,
display_correction_t const *const p_param)
```

Detailed description

Color correction. Implemented as

- R_GLCD_ColorCorrection

Table 1092:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to display interface control block.
param	in	Pointer to color correction configuration structure.

Parameter p_ctrl

Definition: `display_ctrl_tconst *const p_ctrl`

Display control block. Allocate an instance specific control block to pass into the display API calls. Implemented asglcd_instance_ctrl_tDisplay control block

Parameter param

8.11.8.25 clut

```
ssp_err_t(* display_api_t::clut) (display_ctrl_t const *const p_ctrl,
display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t frame)
```

Detailed description

Set CLUT for display device. Implemented as

- R_GLCD_ClutUpdate

Table 1093:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to display interface control block.
p_clut_cfg	in	Pointer to CLUT configuration structure.
frame	in	Number of frame buffer corresponding to the CLUT.

Parameter p_ctrl

Definition: `display_ctrl_tconst *const p_ctrl`

Display control block. Allocate an instance specific control block to pass into the display API calls. Implemented asglcd_instance_ctrl_tDisplay control block

Parameter p_clut_cfg

Definition: `display_clut_cfg_t const *const p_clut_cfg`

Display CLUT configuration structure

- display_clut_cfg_t::p_base

  Pointer to CLUT source data.

- display_clut_cfg_t::start

  Beginning of CLUT entry to be updated.

- display_clut_cfg_t::size

  Size of CLUT entry to be updated.

Parameter frame

8.11.8.26 statusGet

```
ssp_err_t(* display_api_t::statusGet) (display_ctrl_t const *const p_ctrl,
display_status_t *const p_status)
```

Detailed description

Get status for display device. Implemented as

- R_GLCD_StatusGet

Table 1094:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to display interface control block.
status	in	Pointer to display interface status structure.

Parameter p_ctrl

Definition: display_ctrl_tconst *const p_ctrl

Display control block. Allocate an instance specific control block to pass into the display API calls. Implemented asglcd_instance_ctrl_tDisplay control block

Parameter status

8.11.8.27 versionGet

```
ssp_err_t(* display_api_t::versionGet) (ssp_version_t *p_version)
```

Detailed description

Get version. Implemented as

- R_GLCD_VersionGet

Table 1095:Parameters

Name	Direction	Description
p_version	in	Pointer to the memory to store the version information.

Parameter p_version

8.11.8.28 display_instance_t

display_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- display_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- display_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- display_api_t const * p_api

  Pointer to the API structure for this instance.

8.12 Digital-to-Analog Converter

8.12.1 DAC Interface

Interface for D/A converters.

8.12.1.1 Summary

The DAC interface provides standard Digital/Analog Converter functionality. A DAC application writes digital sample data to the device and generates analog output on the DAC output pin.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

DAC Interface description: DAC Driver

8.12.1.2 Interface API

dac_api_t

Function name	Description
.open	Initial configuration.
.close	Close the D/A Converter.
.write	Write sample value to the D/A Converter.
.start	Start the D/A Converter if it has not been started yet.
.stop	Stop the D/A Converter if the converter is running.
.versionGet	Get version and store it in provided pointer p_version.
.infoGet	Get information about DAC Resolution and store it in provided pointer p_info.

8.12.1.3 Data structures

- dac_info_t
- dac_cfg_t
- dac_instance_t

8.12.1.4 Enumerations

- dac_data_format_t

8.12.1.5 Typedefs

- dac_size_t
- dac_ctrl_t

8.12.1.6 Defines

- #define DAC_API_VERSION_MAJOR

  Initial value: `(1U)`

- #define DAC_API_VERSION_MINOR

  Initial value: (2U)

8.12.1.7 API Data

dac_data_format_t

dac_data_format_t

Detailed description

DAC Open API AD/DA data format settings.

Enumerated values

Name	Description
DAC_DATA_FORMAT_FLUSH_RIGHT	LSB of data is flush to the right leaving the top 4 bits unused.
DAC_DATA_FORMAT_FLUSH_LEFT	MSB of data is flush to the left leaving the bottom 4 bits unused.

dac_size_t

typedef uint16_t dac_size_t

Detailed description

Data type to store DAC output value.

dac_ctrl_t

typedef void dac_ctrl_t

Detailed description

DAC control block. Allocate an instance specific control block to pass into the DAC API calls. Implemented as

- dac_instance_ctrl_t

8.12.1.8 API Structures

dac_info_t

dac_info_t

Detailed description

DAC information structure to store various information for a DAC

Variables

- uint8_t bit_width

  Resolution of the DAC.

dac_cfg_t

dac_cfg_t

Detailed description

DAC Open API configuration parameter

Variables

- uint8_t channel

  ID associated with this DAC channel.

- bool ad_da_synchronized

  AD/DA synchronization.

- dac_data_format_t data_format

  Data format.

- bool output_amplifier_enabled

  Output amplifier enable.

- void const * p_extend

dac_api_t

dac_api_t

Detailed description

DAC driver structure. General DAC functions implemented at the HAL layer follow this API.

open

```
ssp_err_t(* dac_api_t::open) (dac_ctrl_t *p_ctrl, dac_cfg_t const *const p_cfg)
```

Detailed description

Initial configuration. Implemented as

- R_DAC_Open
- R_DAC8_Open

Table 1096:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block. Must be declared by user. Elements set here.
p_cfg	in	Pointer to configuration structure. All elements of this structure must be set by user.

Parameter p_ctrl

Definition: dac_ctrl_t*p_ctrl

DAC control block. Allocate an instance specific control block to pass into the DAC API calls. Implemented asdac_instance_ctrl_t

Parameter p_cfg

Definition: dac_cfg_t const *const p_cfg

DAC Open API configuration parameter

- dac_cfg_t::channel

  ID associated with this DAC channel.

- dac_cfg_t::ad_da_synchronized

  AD/DA synchronization.

- dac_cfg_t::dac_data_format_t

  Data format.

  Enumerated as:

  – DAC_DATA_FORMAT_FLUSH_RIGHT

  – DAC_DATA_FORMAT_FLUSH_LEFT

- dac_cfg_t::output_amplifier_enabled

  Output amplifier enable.

- dac_cfg_t::p_extend

close

ssp_err_t(* dac_api_t::close) (dac_ctrl_t *p_ctrl)

Detailed description

Close the D/A Converter. Implemented as

- R_DAC_Close
- R_DAC8_Close

Table 1097:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this timer.

Parameter p_ctrl

Definition: dac_ctrl_t*p_ctrl

DAC control block. Allocate an instance specific control block to pass into the DAC API calls. Implemented asdac_instance_ctrl_t

write

```
ssp_err_t(* dac_api_t::write) (dac_ctrl_t *p_ctrl, dac_size_t value)
```

Detailed description

Write sample value to the D/A Converter. Implemented as

- R_DAC_Write
- R_DAC8_Write

Table 1098:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this timer.
value	in	Sample value to be written to the D/A Converter.

Parameter p_ctrl

Definition: `dac_ctrl_t*p_ctrl`

DAC control block. Allocate an instance specific control block to pass into the DAC API calls. Implemented asdac_instance_ctrl_t

Parameter value

Definition: `dac_size_tvalue`

Data type to store DAC output value.

start

```
ssp_err_t(* dac_api_t::start) (dac_ctrl_t *p_ctrl)
```

Detailed description

Start the D/A Converter if it has not been started yet. Implemented as

- R_DAC_Start
- R_DAC8_Start

Table 1099:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this timer.

Parameter p_ctrl

Definition: `dac_ctrl_t*p_ctrl`

DAC control block. Allocate an instance specific control block to pass into the DAC API calls. Implemented asdac_instance_ctrl_t

stop

```
ssp_err_t(* dac_api_t::stop) (dac_ctrl_t *p_ctrl)
```

Detailed description

Stop the D/A Converter if the converter is running. Implemented as

- R_DAC_Stop
- R_DAC8_Stop

Table 1100:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this timer.

Parameter p_ctrl

Definition: `dac_ctrl_t*p_ctrl`

DAC control block. Allocate an instance specific control block to pass into the DAC API calls. Implemented asdac_instance_ctrl_t

versionGet

```
ssp_err_t(* dac_api_t::versionGet) (ssp_version_t *p_version)
```

Detailed description

Get version and store it in provided pointer p_version. Implemented as

- R_DAC_VersionGet
- R_DAC8_VersionGet

Table 1101:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

infoGet

```
ssp_err_t(* dac_api_t::infoGet) (dac_info_t *const p_info)
```

Detailed description

Get information about DAC Resolution and store it in provided pointer p_info. Implemented as

- R_DAC_InfoGet
- R_DAC8_InfoGet

Table 1102:Parameters

Name	Direction	Description
p_info	out	Collection of information for this DAC.

Parameter p_info

Definition: `dac_info_t*const p_info`

DAC information structure to store various information for a DAC

- `dac_info_t::bit_width`
  Resolution of the DAC.

dac_instance_t

dac_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- dac_ctrl_t * p_ctrl
  Pointer to the control structure for this instance.
- dac_cfg_t const * p_cfg
  Pointer to the configuration structure for this instance.
- dac_api_t const * p_api
  Pointer to the API structure for this instance.

8.12.2 dac_api_t

dac_api_t

8.12.2.1 Detailed description

DAC driver structure. General DAC functions implemented at the HAL layer follow this API.

8.12.3 open

```
ssp_err_t(* dac_api_t::open) (dac_ctrl_t *p_ctrl, dac_cfg_t const *const p_cfg)
```

8.12.3.1 Detailed description

Initial configuration. Implemented as

- R_DAC_Open
- R_DAC8_Open

Table 1103:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block. Must be declared by user. Elements set here.
p_cfg	in	Pointer to configuration structure. All elements of this structure must be set by user.

8.12.3.2 Parameter p_ctrl

Definition: `dac_ctrl_t*p_ctrl`

DAC control block. Allocate an instance specific control block to pass into the DAC API calls. Implemented asdac_instance_ctrl_t

8.12.3.3 Parameter p_cfg

Definition: `dac_cfg_t const *const p_cfg`

DAC Open API configuration parameter

- `dac_cfg_t`::`channel`

  ID associated with this DAC channel.

- `dac_cfg_t`::`ad_da_synchronized`

  AD/DA synchronization.

- `dac_cfg_t`::`dac_data_format_t`

  Data format.

  Enumerated as:

  – DAC_DATA_FORMAT_FLUSH_RIGHT

  – DAC_DATA_FORMAT_FLUSH_LEFT

- `dac_cfg_t`::`output_amplifier_enabled`

  Output amplifier enable.

- `dac_cfg_t`::`p_extend`

8.12.4 close

```
ssp_err_t(* dac_api_t::close) (dac_ctrl_t *p_ctrl)
```

8.12.4.1 Detailed description

Close the D/A Converter. Implemented as

- R_DAC_Close
- R_DAC8_Close

Table 1104:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this timer.

8.12.4.2 Parameter p_ctrl

Definition: `dac_ctrl_t*p_ctrl`

DAC control block. Allocate an instance specific control block to pass into the DAC API calls. Implemented asdac_instance_ctrl_t

8.12.5 write

```
ssp_err_t(* dac_api_t::write) (dac_ctrl_t *p_ctrl, dac_size_t value)
```

8.12.5.1 Detailed description

Write sample value to the D/A Converter. Implemented as

- R_DAC_Write
- R_DAC8_Write

Table 1105:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this timer.
value	in	Sample value to be written to the D/A Converter.

8.12.5.2 Parameter p_ctrl

Definition: dac_ctrl_t*p_ctrl

DAC control block. Allocate an instance specific control block to pass into the DAC API calls. Implemented asdac_instance_ctrl_t

8.12.5.3 Parameter value

Definition: dac_size_tvalue

Data type to store DAC output value.

8.12.6 start

```
ssp_err_t(* dac_api_t::start) (dac_ctrl_t *p_ctrl)
```

8.12.6.1 Detailed description

Start the D/A Converter if it has not been started yet. Implemented as

- R_DAC_Start
- R_DAC8_Start

Table 1106:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this timer.

8.12.6.2 Parameter p_ctrl

Definition: dac_ctrl_t*p_ctrl

DAC control block. Allocate an instance specific control block to pass into the DAC API calls. Implemented asdac_instance_ctrl_t

8.12.7 stop

```
ssp_err_t(* dac_api_t::stop) (dac_ctrl_t *p_ctrl)
```

8.12.7.1 Detailed description

Stop the D/A Converter if the converter is running. Implemented as

- R_DAC_Stop

- R_DAC8_Stop

Table 1107:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this timer.

8.12.7.2 Parameter p_ctrl

Definition: `dac_ctrl_t*p_ctrl`

DAC control block. Allocate an instance specific control block to pass into the DAC API calls. Implemented asdac_instance_ctrl_t

8.12.8 versionGet

```
ssp_err_t(* dac_api_t::versionGet) (ssp_version_t *p_version)
```

8.12.8.1 Detailed description

Get version and store it in provided pointer p_version. Implemented as

- R_DAC_VersionGet
- R_DAC8_VersionGet

Table 1108:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

8.12.8.2 Parameter p_version

8.12.9 infoGet

```
ssp_err_t(* dac_api_t::infoGet) (dac_info_t *const p_info)
```

8.12.9.1 Detailed description

Get information about DAC Resolution and store it in provided pointer p_info. Implemented as

- R_DAC_InfoGet

- R_DAC8_InfoGet

Table 1109:Parameters

Name	Direction	Description
p_info	out	Collection of information for this DAC.

8.12.9.2 Parameter p_info

Definition: `dac_info_t*const p_info`

DAC information structure to store various information for a DAC

- `dac_info_t::bit_width`
  Resolution of the DAC.

8.12.10 dac_cfg_t

dac_cfg_t

8.12.10.1 Detailed description

DAC Open API configuration parameter

8.12.10.2 Variables

- uint8_t channel
  ID associated with this DAC channel.
- bool ad_da_synchronized
  AD/DA synchronization.
- dac_data_format_t data_format
  Data format.
- bool output_amplifier_enabled
  Output amplifier enable.
- void const * p_extend

8.13 DOC Interface

Interface for the Data Operation Circuit.

Defines the API and data structures for the DOC implementation of the Data Operation Circuit (DOC) interface.

8.13.1 Summary

This module implements the DOC_API using the Data Operation Circuit (DOC).

Implemented by: DOC

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

DOC Interface description: Data Operation Circuit Driver

8.13.2 Interface API

doc_api_t

Function name	Description
.open	Initial configuration.
.close	Allow the driver to be reconfigured. Will reduce power consumption.
.statusGet	Get the DOC status and stores it in the provided pointer p_status.
.statusClear	Clear DOPCF status flag.
.write	Write to the DODIR and DODSR registers.
.inputRegisterWrite	Write to the DODIR register.
.versionGet	Get version and stores it in provided pointer p_version.

8.13.3 Data structures

- doc_callback_args_t
- doc_data_t
- doc_cfg_t
- doc_instance_t

8.13.4 Enumerations

- doc_event_t
- doc_status_t

8.13.5 Typedefs

- doc_size_t
- doc_ctrl_t

8.13.6 Defines

- #define DOC_API_VERSION_MAJOR

  Initial value: `(1U)`

  Register definitions, common services and error codes.
- #define DOC_API_VERSION_MINOR

  Initial value: `(2U)`

8.13.7 API Data

8.13.7.1 doc_event_t

`doc_event_t`

Detailed description

Event that can trigger a callback function.

Enumerated values

Name	Description
DOC_EVENT_COMPARISON_MISMATCH	Comparison of data has resulted in a mismatch.
DOC_EVENT_ADDITION	Addition of data has resulted in a value greater than H'FFFF.
DOC_EVENT_SUBTRACTION	Subtraction of data has resulted in a value less than H'0000.
DOC_EVENT_COMPARISON_MATCH	Comparison of data has resulted in a match.

8.13.7.2 doc_status_t

```
doc_status_t
```

Detailed description

Status of the data comparison operation.

Enumerated values

Name	Description
DOC_STATUS_CONDITION_FALSE	Data comparison condition NOT met (match or mismatch), addition result NOT > H'FFFF, subtraction result NOT < H'0000.
DOC_STATUS_CONDITION_TRUE	Data comparison condition met (match or mismatch), addition result > H'FFFF, subtraction result < H'0000.

8.13.7.3 doc_size_t

```
typedef uint16_t doc_size_t
```

Detailed description

Size of the comparison data supported by the Data Operation Circuit (DOC)

8.13.7.4 doc_ctrl_t

```
typedef void doc_ctrl_t
```

Detailed description

DOC control block. Allocate an instance specific control block to pass into the DOC API calls. Implemented as

- doc_instance_ctrl_t

8.13.8 API Structures

8.13.8.1 doc_callback_args_t

doc_callback_args_t

Detailed description

Callback function parameter data.

Variables

- doc_event_t event

  The event is used to identify what caused the callback.

- void const * p_context

  Placeholder for user data. Set in doc_api_t::open function in doc_cfg_t.

8.13.8.2 doc_data_t

doc_data_t

Detailed description

Data to be written to DOC register for comparison/addition/subtraction.

Variables

- doc_size_t dodir

  Value to be written to the DOC DODIR.

- doc_size_t dodsr

  Value to be written to the DOC DODSR.

8.13.8.3 doc_cfg_t

doc_cfg_t

Detailed description

User configuration structure, used in the open function.

Variables

- doc_event_t event

  Select enumerated value from doc_event_t.

- uint8_t irq_ipl

  DOC interrupt priority.

- void(* p_callback)(doc_callback_args_t *p_args)

  Callback provided when a DOC ISR occurs.

- void const * p_context

  Placeholder for user data. Passed to the user callback in doc_callback_args_t.

8.13.8.4 doc_api_t

doc_api_t

Detailed description

Data Operation Circuit (DOC) API structure. DOC functions implemented at the HAL layer will follow this API.

8.13.8.5 open

```
ssp_err_t(* doc_api_t::open) (doc_ctrl_t *const p_ctrl, doc_cfg_t const *const
p_cfg)
```

Detailed description

Initial configuration. Implemented as

- R_DOC_Open

NOTE: Peripheral clocks should be configured prior to calling this function.

Table 1110:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block. Must be declared by user. Elements set here.
p_cfg	in	Pointer to configuration structure. All elements of this structure must be set by user.

Parameter p_ctrl

Definition: `doc_ctrl_t*const p_ctrl`

DOC control block. Allocate an instance specific control block to pass into the DOC API calls. Implemented asdoc_instance_ctrl_t

Parameter p_cfg

Definition: `doc_cfg_t const *const p_cfg`

User configuration structure, used in the open function.

- `doc_cfg_t::doc_event_t`

  Select enumerated value from doc_event_t.

  Enumerated as:

  – DOC_EVENT_COMPARISON_MISMATCH

  – DOC_EVENT_ADDITION

  – DOC_EVENT_SUBTRACTION

– DOC_EVENT_COMPARISON_MATCH

- doc_cfg_t::irq_ipl

  DOC interrupt priority.

- doc_cfg_t::p_callback

  Callback provided when a DOC ISR occurs.

- doc_cfg_t::p_context

  Placeholder for user data. Passed to the user callback in doc_callback_args_t.

8.13.8.6 close

```
ssp_err_t(* doc_api_t::close) (doc_ctrl_t *const p_ctrl)
```

Detailed description

Allow the driver to be reconfigured. Will reduce power consumption. Implemented as

- R_DOC_Close

Table 1111:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call.

Parameter p_ctrl

Definition: doc_ctrl_t*const p_ctrl

DOC control block. Allocate an instance specific control block to pass into the DOC API calls. Implemented asdoc_instance_ctrl_t

8.13.8.7 statusGet

```
ssp_err_t(* doc_api_t::statusGet) (doc_ctrl_t *const p_ctrl, doc_status_t
*p_status)
```

Detailed description

Get the DOC status and stores it in the provided pointer p_status. Implemented as

- R_DOC_StatusGet

NOTE: Call open to configure the DOC before using this function.

Table 1112:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call.
p_status	out	Indicates the status of the comparison/addition/subtraction operation. Result will be one of doc_status_t.

Parameter p_ctrl

Definition: `doc_ctrl_t*const p_ctrl`

DOC control block. Allocate an instance specific control block to pass into the DOC API calls. Implemented asdoc_instance_ctrl_t

Parameter p_status

Definition: `doc_status_t*p_status`

Status of the data comparison operation.

8.13.8.8 statusClear

```
ssp_err_t(* doc_api_t::statusClear) (doc_ctrl_t *const p_ctrl)
```

Detailed description

Clear DOPCF status flag. Implemented as

- R_DOC_StatusClear

NOTE: Call open to configure the DOC before using this function.

Table 1113:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call.

Parameter p_ctrl

Definition: `doc_ctrl_t*const p_ctrl`

DOC control block. Allocate an instance specific control block to pass into the DOC API calls. Implemented asdoc_instance_ctrl_t

8.13.8.9 write

```
ssp_err_t(* doc_api_t::write) (doc_ctrl_t *const p_ctrl, doc_data_t *const p_data)
```

Detailed description

Write to the DODIR and DODSR registers. Implemented as

- R_DOC_Write

NOTE: Call open to configure the DOC before using this function.

Table 1114:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call.
p_data	in	Pointer to data to be written to DOC DODIR and DODSR registers.

Parameter p_ctrl

Definition: `doc_ctrl_t*const p_ctrl`

DOC control block. Allocate an instance specific control block to pass into the DOC API calls. Implemented asdoc_instance_ctrl_t

Parameter p_data

Definition: `doc_data_t*const p_data`

Data to be written to DOC register for comparison/addition/subtraction.

- `doc_data_t::dodir`

  Value to be written to the DOC DODIR.

- `doc_data_t::dodsr`

  Value to be written to the DOC DODSR.

8.13.8.10 inputRegisterWrite

```
ssp_err_t(* doc_api_t::inputRegisterWrite) (doc_ctrl_t *const p_ctrl, doc_size_t
data)
```

Detailed description

Write to the DODIR register. Implemented as

- R_DOC_InputRegisterWrite

NOTE: Call open to configure the DOC before using this function.

Table 1115:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call.
data	in	Data to be written to DOC DODIR register.

Parameter p_ctrl

Definition: `doc_ctrl_t*const p_ctrl`

DOC control block. Allocate an instance specific control block to pass into the DOC API calls. Implemented asdoc_instance_ctrl_t

Parameter data

Definition: `doc_size_tdata`

Size of the comparison data supported by the Data Operation Circuit (DOC)

8.13.8.11 versionGet

```
ssp_err_t(* doc_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Get version and stores it in provided pointer p_version. Implemented as

- R_DOC_VersionGet

Table 1116:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.13.8.12 doc_instance_t

doc_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- doc_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- doc_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- doc_api_t const * p_api

  Pointer to the API structure for this instance.

8.14 ELC Interface

Interface for the Event Link Controller.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Event Link Controller Interface description: ELC Driver

Related SSP architecture topics:

- What is an SSP Interface? SSP Interfaces
- What is a SSP Layer? SSP Predefined Layers
- How to use SSP Interfaces and Modules? Using SSP Modules

Event Link Controller Interface description: ELC Driver

8.14.1 Interface API

elc_api_t

Function name	Description
.init	Initialize all links in the Event Link Controller.
.softwareEventGenerate	Generate a software event in the Event Link Controller.
.linkSet	Create a single event link.

Function name	Description
.linkBreak	Break an event link.
.enable	Enable the operation of the Event Link Controller.
.disable	Disable the operation of the Event Link Controller.
.versionGet	Get the driver version based on compile time macros.

8.14.2 Data structures

- elc_link_t
- elc_cfg_t
- elc_instance_t

8.14.3 Enumerations

- elc_software_event_t

8.14.4 Defines

- #define ELC_API_VERSION_MAJOR
  Initial value: `(1U)`
- #define ELC_API_VERSION_MINOR
  Initial value: `(2U)`

8.14.5 API Data

8.14.5.1 elc_software_event_t

`elc_software_event_t`

Detailed description

Software event number

Enumerated values

Name	Description
ELC_SOFTWARE_EVENT_0	Software event 0.

Name	Description
ELC_SOFTWARE_EVENT_1	Software event 1.

8.14.6 API Structures

8.14.6.1 elc_link_t

elc_link_t

Detailed description

Individual event link. The actual peripheral definitions can be found in the MCU specific (ie. /mcu/S124/bsp_elc.h) bsp_elc.h files.

Variables

- elc_peripheral_t peripheral

  Peripheral to receive the signal.

- elc_event_t event

  Signal that gets sent to the Peripheral.

8.14.6.2 elc_cfg_t

elc_cfg_t

Detailed description

Main configuration structure for the Event Link Controller

Variables

- bool autostart

  Start operation and enable interrupts during open().

- uint32_t link_count

  Number of event links.

- elc_link_t const * link_list

  Event links.

8.14.6.3 elc_api_t

elc_api_t

Detailed description

ELC driver structure. General ELC functions implemented at the HAL layer follow this API.

8.14.6.4 init

```
ssp_err_t(* elc_api_t::init) (elc_cfg_t const *const p_cfg)
```

Detailed description

Initialize all links in the Event Link Controller. Implemented as

- R_ELC_Init

Table 1117:Parameters

Name	Direction	Description
p_cfg	in	Pointer to configuration structure.

Parameter p_cfg

Definition: `elc_cfg_t const *const p_cfg`

Main configuration structure for the Event Link Controller

- `elc_cfg_t::autostart`

  Start operation and enable interrupts during open().

- `elc_cfg_t::link_count`

  Number of event links.

- `elc_cfg_t::elc_link_t`

  Event links.

8.14.6.5 softwareEventGenerate

```
ssp_err_t(* elc_api_t::softwareEventGenerate) (elc_software_event_t event_num)
```

Detailed description

Generate a software event in the Event Link Controller. Implemented as

- R_ELC_SoftwareEventGenerate

Table 1118:Parameters

Name	Direction	Description
eventNum	in	Software event number to be generated.

Parameter eventNum

8.14.6.6 linkSet

```
ssp_err_t(* elc_api_t::linkSet) (elc_peripheral_t peripheral, elc_event_t signal)
```

Detailed description

Create a single event link. Implemented as

- R_ELC_LinkSet

Table 1119:Parameters

Name	Direction	Description
peripheral	in	The peripheral block that will receive the event signal.
signal	in	The event signal.

Parameter peripheral
Parameter signal

8.14.6.7 linkBreak

```
ssp_err_t(* elc_api_t::linkBreak) (elc_peripheral_t peripheral)
```

Detailed description

Break an event link. Implemented as

- R_ELC_LinkBreak

Table 1120:Parameters

Name	Direction	Description
peripheral	in	The peripheral that should no longer be linked.

Parameter peripheral

8.14.6.8 enable

```
ssp_err_t(* elc_api_t::enable) (void)
```

Detailed description

Enable the operation of the Event Link Controller. Implemented as

- R_ELC_Enable

8.14.6.9 disable

```
ssp_err_t(* elc_api_t::disable) (void)
```

Detailed description

Disable the operation of the Event Link Controller. Implemented as

- R_ELC_Disable

8.14.6.10 versionGet

```
ssp_err_t(* elc_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Get the driver version based on compile time macros. Implemented as

- R_ELC_VersionGet

Table 1121:Parameters

Name	Direction	Description
p_version	out	is value returned.

Parameter p_version

8.14.6.11 elc_instance_t

elc_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- elc_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- elc_api_t const * p_api

  Pointer to the API structure for this instance.

8.15 External IRQ Interface

Interface for detecting external interrupts.

8.15.1 Summary

The external IRQ interface supports external inputs, for example input from pins or capacitive touch buttons. When an input trigger is detected, a user provided callback function will be called.

Implemented by: ICU

Related interfaces: Key Matrix Interface

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

External IRQ Interface description: External IRQ Driver

8.15.2 Interface API

external_irq_api_t

Function name	Description
.open	Initial configuration.
.enable	Enable callback when external IRQ occurs.
.disable	Disable callback when external IRQ occurs.
.triggerSet	Set trigger.
.filterEnable	Enables noise filter.
.filterDisable	Disable noise filter.
.close	Allow driver to be reconfigured. May reduce power consumption.
.versionGet	Get version and store it in provided pointer p_version.

8.15.3 Data structures

- external_irq_callback_args_t
- external_irq_cfg_t
- external_irq_instance_t

8.15.4 Enumerations

- external_irq_trigger_t
- external_irq_pclk_div_t

8.15.5 Typedefs

- external_irq_ctrl_t

8.15.6 Defines

- #define EXTERNAL_IRQ_API_VERSION_MAJOR

  Initial value: `(1U)`

  EXTERNAL IRQ API version number (Major)
- #define EXTERNAL_IRQ_API_VERSION_MINOR

  Initial value: `(2U)`

  EXTERNAL IRQ API version number (Minor)

8.15.7 API Data

8.15.7.1 external_irq_trigger_t

`external_irq_trigger_t`

Detailed description

Trigger type: rising edge, falling edge, both edges, low level.

Enumerated values

Name	Description
EXTERNAL_IRQ_TRIG_FALLING	Falling edge trigger.
EXTERNAL_IRQ_TRIG_RISING	Rising edge trigger.
EXTERNAL_IRQ_TRIG_BOTH_EDGE	Both edges trigger.
EXTERNAL_IRQ_TRIG_LEVEL_LOW	Low level trigger.

8.15.7.2 external_irq_pclk_div_t

`external_irq_pclk_div_t`

Detailed description

External IRQ input pin digital filtering sample clock divisor settings.

Enumerated values

Name	Description
EXTERNAL_IRQ_PCLK_DIV_BY_1	Filter using PCLK divided by 1.
EXTERNAL_IRQ_PCLK_DIV_BY_8	Filter using PCLK divided by 8.
EXTERNAL_IRQ_PCLK_DIV_BY_32	Filter using PCLK divided by 32.
EXTERNAL_IRQ_PCLK_DIV_BY_64	Filter using PCLK divided by 64.

8.15.7.3 external_irq_ctrl_t

```
typedef void external_irq_ctrl_t
```

Detailed description

External IRQ control block. Allocate an instance specific control block to pass into the external IRQ API calls. Implemented as

- icu_instance_ctrl_t

8.15.8 API Structures

8.15.8.1 external_irq_callback_args_t

external_irq_callback_args_t

Detailed description

Callback function parameter data

Variables

- void const * p_context

  Placeholder for user data. Set in external_irq_api_t::open function in external_irq_cfg_t.

- uint32_t channel

  The physical hardware channel that caused the interrupt.

8.15.8.2 external_irq_cfg_t

external_irq_cfg_t

Detailed description

User configuration structure, used in open function

Variables

- uint8_t channel

  Hardware channel used.
- uint8_t irq_ipl

  Interrupt priority.
- external_irq_trigger_t trigger

  Trigger setting.
- external_irq_pclk_div_t pclk_div

  Digital filter clock divisor setting.
- bool autostart

  Start operation and enable interrupts during open().
- bool filter_enable

  Digital filter enable/disable setting.
- void(* p_callback)(external_irq_callback_args_t *p_args)

  Callback provided external input trigger occurs.
- void const * p_context

  Placeholder for user data. Passed to the user callback in external_irq_callback_args_t.
- void const * p_extend

  External IRQ hardware dependent configuration.

8.15.8.3 external_irq_api_t

external_irq_api_t

Detailed description

External interrupt driver structure. External interrupt functions implemented at the HAL layer will follow this API.

8.15.8.4 open

```
ssp_err_t(* external_irq_api_t::open) (external_irq_ctrl_t *const p_ctrl,
external_irq_cfg_t const *const p_cfg)
```

Detailed description

Initial configuration. Implemented as

- R_ICU_ExternalIrqOpen

Table 1122:Parameters

Name	Direction	Description
p_ctrl	out	Pointer to control block. Must be declared by user. Value set here.
p_cfg	in	Pointer to configuration structure. All elements of the structure must be set by user.

Parameter p_ctrl

Definition: `external_irq_ctrl_t*const p_ctrl`

External IRQ control block. Allocate an instance specific control block to pass into the external IRQ API calls. Implemented asicu_instance_ctrl_t

Parameter p_cfg

Definition: `external_irq_cfg_t const *const p_cfg`

User configuration structure, used in open function

- `external_irq_cfg_t`::`channel`

  Hardware channel used.

- `external_irq_cfg_t`::`irq_ipl`

  Interrupt priority.

- `external_irq_cfg_t`::`external_irq_trigger_t`

  Trigger setting.

  Enumerated as:

  – EXTERNAL_IRQ_TRIG_FALLING

  – EXTERNAL_IRQ_TRIG_RISING

  – EXTERNAL_IRQ_TRIG_BOTH_EDGE

  – EXTERNAL_IRQ_TRIG_LEVEL_LOW

- `external_irq_cfg_t`::`external_irq_pclk_div_t`

  Digital filter clock divisor setting.

  Enumerated as:

  – EXTERNAL_IRQ_PCLK_DIV_BY_1

  – EXTERNAL_IRQ_PCLK_DIV_BY_8

  – EXTERNAL_IRQ_PCLK_DIV_BY_32

  – EXTERNAL_IRQ_PCLK_DIV_BY_64

- external_irq_cfg_t::autostart

  Start operation and enable interrupts during open().

- external_irq_cfg_t::filter_enable

  Digital filter enable/disable setting.

- external_irq_cfg_t::p_callback

  Callback provided external input trigger occurs.

- external_irq_cfg_t::p_context

  Placeholder for user data. Passed to the user callback in external_irq_callback_args_t.

- external_irq_cfg_t::p_extend

  External IRQ hardware dependent configuration.

8.15.8.5 enable

```
ssp_err_t(* external_irq_api_t::enable) (external_irq_ctrl_t *const p_ctrl)
```

Detailed description

Enable callback when external IRQ occurs. Implemented as

- R_ICU_ExternalIrqEnable

Table 1123:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in Open call for this external interrupt.

Parameter p_ctrl

Definition: external_irq_ctrl_t*const p_ctrl

External IRQ control block. Allocate an instance specific control block to pass into the external IRQ API calls. Implemented asicu_instance_ctrl_t

8.15.8.6 disable

```
ssp_err_t(* external_irq_api_t::disable) (external_irq_ctrl_t *const p_ctrl)
```

Detailed description

Disable callback when external IRQ occurs. Implemented as

- R_ICU_ExternalIrqDisable

Table 1124:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in Open call for this external interrupt.

Parameter p_ctrl

Definition: `external_irq_ctrl_t*const p_ctrl`

External IRQ control block. Allocate an instance specific control block to pass into the external IRQ API calls. Implemented asicu_instance_ctrl_t

8.15.8.7 triggerSet

```
ssp_err_t(* external_irq_api_t::triggerSet) (external_irq_ctrl_t *const p_ctrl,
external_irq_trigger_t const trigger)
```

Detailed description

Set trigger. Implemented as

- R_ICU_ExternalIrqTriggerSet

Table 1125:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in Open call for this external interrupt.
trigger	in	Trigger type

Parameter p_ctrl

Definition: `external_irq_ctrl_t*const p_ctrl`

External IRQ control block. Allocate an instance specific control block to pass into the external IRQ API calls. Implemented asicu_instance_ctrl_t

Parameter trigger

8.15.8.8 filterEnable

```
ssp_err_t(* external_irq_api_t::filterEnable) (external_irq_ctrl_t *const p_ctrl)
```

Detailed description

Enables noise filter. Implemented as

- R_ICU_ExternalIrqFilterEnable

Table 1126:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in Open call for this external interrupt.

Parameter p_ctrl

Definition: `external_irq_ctrl_t*const p_ctrl`

External IRQ control block. Allocate an instance specific control block to pass into the external IRQ API calls. Implemented asicu_instance_ctrl_t

8.15.8.9 filterDisable

```
ssp_err_t(* external_irq_api_t::filterDisable) (external_irq_ctrl_t *const
p_ctrl)
```

Detailed description

Disable noise filter. Implemented as

- R_ICU_ExternalIrqFilterDisable

Table 1127:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in Open call for this external interrupt.

Parameter p_ctrl

Definition: `external_irq_ctrl_t*const p_ctrl`

External IRQ control block. Allocate an instance specific control block to pass into the external IRQ API calls. Implemented asicu_instance_ctrl_t

8.15.8.10 close

```
ssp_err_t(* external_irq_api_t::close) (external_irq_ctrl_t *const p_ctrl)
```

Detailed description

Allow driver to be reconfigured. May reduce power consumption. Implemented as

- R_ICU_ExternalIrqClose

Table 1128:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in Open call for this external interrupt.

Parameter p_ctrl

Definition: `external_irq_ctrl_t*const p_ctrl`

External IRQ control block. Allocate an instance specific control block to pass into the external IRQ API calls. Implemented asicu_instance_ctrl_t

8.15.8.11 versionGet

```
ssp_err_t(* external_irq_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Get version and store it in provided pointer p_version. Implemented as

- R_ICU_ExternalIrqVersionGet

Table 1129:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.15.8.12 external_irq_instance_t

external_irq_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- external_irq_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- external_irq_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- external_irq_api_t const * p_api

  Pointer to the API structure for this instance.

8.16 Flash Interface

Interface for the flash controller.

8.16.1 Summary

The Flash interface provides the functionality necessary to read, write, erase and blank check the Flash memory. Additionally functions are provided to configure the access window and swap areas of the flash memory.

Implemented by:

- High-performance Flash
- Low Power Flash

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Flash Interface description: Flash Driver

8.16.2 Interface API

flash_api_t

Function name	Description
.open	Open FLASH device.
.write	Write FLASH device.
.read	Read FLASH device.
.erase	Erase FLASH device.
.blankCheck	Blank check FLASH device.
.infoGet	Close FLASH device.
.close	Close FLASH device.
.statusGet	Get Status for FLASH device.
.accessWindowSet	Set Access Window for FLASH device.

Function name	Description
.accessWindowClear	Clear any existing Code Flash access window for FLASH device.
.reset	Reset function for FLASH device.
.updateFlashClockFreq	Update Flash clock frequency (FCLK) and recalculate timeout values
.startupAreaSelect	Select which block - Default (Block 0) or Alternate (Block 1) is used as the start-up area block. swap_type | is_temporary | Operation FLASH_STARTUP_AREA_BLOCK0: false On next reset Startup area will be Block 0.FLASH_STARTUP_AREA_BLOCK0 | false | On next reset Startup area will be Block 0. Block 0.FLASH_STARTUP_AREA_BLOCK1: false On next reset Startup area will be Block 1.FLASH_STARTUP_AREA_BLOCK1 | true | Startup area is immediately, but temporarily switched to Block 1. Block 1.FLASH_STARTUP_AREA_BTFLG | true | Startup area is immediately, but temporarily switched to... taken.the Block determined by the Configuration BTFLG.
.versionGet	Get Flash driver version.

8.16.3 Data structures

- flash_fmi_block_info_t
- flash_fmi_regions_t
- flash_info_t
- flash_callback_args_t
- flash_cfg_t
- flash_instance_t

8.16.4 Enumerations

- flash_result_t
- flash_startup_area_swap_t
- flash_event_t

8.16.5 Typedefs

- flash_ctrl_t

8.16.6 Defines

- #define FLASH_API_VERSION_MAJOR

  Initial value: `(1U)`

  FLASH HAL API version number (Major)
- #define FLASH_API_VERSION_MINOR

  Initial value: `(2U)`

  FLASH HAL API version number (Minor)

8.16.7 API Data

8.16.7.1 flash_result_t

`flash_result_t`

Detailed description

Result type for certain operations

Enumerated values

Name	Description
FLASH_RESULT_BLANK	Return status for Blank Check Function.
FLASH_RESULT_NOT_BLANK	Return status for Blank Check Function.
FLASH_RESULT_BGO_ACTIVE	Flash is configured for BGO mode. Result is returned in callback.

8.16.7.2 flash_startup_area_swap_t

`flash_startup_area_swap_t`

Detailed description

Parameter for specifying the startup area swap being requested by startupAreaSelect()

Enumerated values

Name	Description
FLASH_STARTUP_AREA_BLOCK1	Startup area will be set to Block 1.
FLASH_STARTUP_AREA_BLOCK0	Startup area will be set to Block 0.
FLASH_STARTUP_AREA_BTFLG	Startup area will be set based on the value of the BTFLG.

8.16.7.3 flash_event_t

```
flash_event_t
```

Detailed description

Event types returned by the ISR callback when used in Data Flash BGO mode

Enumerated values

Name	Description
FLASH_EVENT_ERASE_COMPLETE	Erase operation successfully completed.
FLASH_EVENT_WRITE_COMPLETE	Write operation successfully completed.
FLASH_EVENT_BLANK	Blank check operation successfully completed. Specified area is blank.
FLASH_EVENT_NOT_BLANK	Blank check operation successfully completed. Specified area is NOT blank.
FLASH_EVENT_ERR_DF_ACCESS	Data Flash operation failed. Can occur when writing an unerased section.
FLASH_EVENT_ERR_CF_ACCESS	Code Flash operation failed. Can occur when writing an unerased section.
FLASH_EVENT_ERR_CMD_LOCKED	Operation failed, FCU is in Locked state (often result of an illegal command)
FLASH_EVENT_ERR_FAILURE	Erase or Program Operation failed.

8.16.7.4 flash_ctrl_t

```
typedef void flash_ctrl_t
```

Detailed description

Flash control block. Allocate an instance specific control block to pass into the flash API calls. Implemented as

- flash_lp_instance_ctrl_t
- flash_hp_instance_ctrl_t

8.16.8 API Structures

8.16.8.1 flash_fmi_block_info_t

flash_fmi_block_info_t

Detailed description

Flash block details stored in factory flash.

Variables

- uint32_t block_section_st_addr

  starting address for this block section (blocks of this size)
- uint32_t block_section_end_addr

  ending address for this block section (blocks of this size)
- uint32_t block_size

  Flash erase block size.
- uint32_t block_size_write

  Flash write block size.

8.16.8.2 flash_fmi_regions_t

flash_fmi_regions_t

Detailed description

Flash block details

Variables

- uint32_t num_regions

  Length of block info array.
- flash_fmi_block_info_t const * p_block_array

  Block info array base address.

8.16.8.3 flash_info_t

flash_info_t

Detailed description

Information about the flash blocks

Variables

- flash_fmi_regions_t code_flash

  Information about the code flash regions.
- flash_fmi_regions_t data_flash

  Information about the code flash regions.

8.16.8.4 flash_callback_args_t

flash_callback_args_t

Detailed description

Callback function parameter data

Variables

- flash_event_t event

  Event can be used to identify what caused the callback (flash ready or error).
- void const * p_context

  Placeholder for user data. Set in flash_api_t::open function in::flash_cfg_t.

8.16.8.5 flash_cfg_t

flash_cfg_t

Detailed description

FLASH Configuration

Variables

- bool data_flash_bgo

  True if BGO (Background Operation) is enabled for Data Flash.
- void(* p_callback)(flash_callback_args_t *p_args)

  Callback provided when a Flash interrupt ISR occurs.
- void const * p_extend

  FLASH hardware dependent configuration.
- void const * p_context

  Placeholder for user data. Passed to user callback in flash_callback_args_t.
- uint8_t irq_ipl

  Flash ready interrupt priority.
- uint8_t err_irq_ipl

  Flash error interrupt priority (unused in r_flash_lp)

8.16.8.6 flash_api_t

flash_api_t

Detailed description

Shared Interface definition for FLASH

8.16.8.7 open

```
ssp_err_t(* flash_api_t::open) (flash_ctrl_t *const p_ctrl, flash_cfg_t const
*const p_cfg)
```

Detailed description

Open FLASH device. Implemented as

- R_FLASH_LP_Open
- R_FLASH_HP_Open

Table 1130:Parameters

Name	Direction	Description
p_ctrl	out	Pointer to FLASH device control. Must be declared by user. Value set here.
flash_cfg_t	in	Pointer to FLASH configuration structure. All elements of this structure must be set by the user.

Parameter p_ctrl

Definition: `flash_ctrl_t*const p_ctrl`

Flash control block. Allocate an instance specific control block to pass into the flash API calls. Implemented asflash_lp_instance_ctrl_tflash_hp_instance_ctrl_t

Parameter flash_cfg_t

Definition: `flash_cfg_t const *const p_cfg`

FLASH Configuration

- `flash_cfg_t`::`data_flash_bgo`

  True if BGO (Background Operation) is enabled for Data Flash.

- `flash_cfg_t`::`p_callback`

  Callback provided when a Flash interrupt ISR occurs.

- `flash_cfg_t`::`p_extend`

  FLASH hardware dependent configuration.

- flash_cfg_t::p_context

  Placeholder for user data. Passed to user callback in flash_callback_args_t.

- flash_cfg_t::irq_ipl

  Flash ready interrupt priority.

- flash_cfg_t::err_irq_ipl

  Flash error interrupt priority (unused in r_flash_lp)

8.16.8.8 write

```
ssp_err_t(* flash_api_t::write) (flash_ctrl_t *const p_ctrl, uint32_t const
src_address, uint32_t const flash_address, uint32_t const num_bytes)
```

Detailed description

Write FLASH device. Implemented as

- R_FLASH_LP_Write
- R_FLASH_HP_Write

Table 1131:Parameters

Name	Direction	Description
p_ctrl	in	Control for the FLASH device context.
src_address	in	Address of the buffer containing the data to write to Flash.
flash_address	in	Code Flash or Data Flash address to write. The address must be on a programming line boundary.
num_bytes	in	The number of bytes to write. This number must be a multiple of the programming size. For Code Flash this is FLASH_MIN_PGM_SIZE_CF. For Data Flash this is FLASH_MIN_PGM_SIZE_DF.

ATTENTION: Specifying a number that is not a multiple of the programming size will result in SF_FLASH_ERR_-BYTES being returned and no data written.

Parameter p_ctrl

Definition: flash_ctrl_t*const p_ctrl

Flash control block. Allocate an instance specific control block to pass into the flash API calls. Implemented asflash_lp_instance_ctrl_tflash_hp_instance_ctrl_t

Parameter src_address

`uint32_t`

Parameter flash_address

`uint32_t`

Parameter num_bytes

`uint32_t`

8.16.8.9 read

```
ssp_err_t(* flash_api_t::read) (flash_ctrl_t *const p_ctrl, uint8_t *const
p_dest_address, uint32_t const flash_address, uint32_t const num_bytes)
```

Detailed description

Read FLASH device. Implemented as

- R_FLASH_LP_Read
- R_FLASH_HP_Read

Table 1132:Parameters

Name	Direction	Description
p_ctrl	in	Control for the FLASH device context.
p_dest_address	in	Pointer to caller's destination buffer used to hold the data read from Flash.
flash_address	in	Code Flash or Data Flash starting address to read from.
num_bytes	in	The number of bytes to read.

Parameter p_ctrl

Definition: `flash_ctrl_t*const p_ctrl`

Flash control block. Allocate an instance specific control block to pass into the flash API calls. Implemented asflash_lp_instance_ctrl_tflash_hp_instance_ctrl_t

Parameter p_dest_address

`uint8_t`

Parameter flash_address

`uint32_t`

Parameter num_bytes

uint32_t

8.16.8.10 erase

```
ssp_err_t(* flash_api_t::erase) (flash_ctrl_t *const p_ctrl, uint32_t const
address, uint32_t const num_blocks)
```

Detailed description

Erase FLASH device. Implemented asR_FLASH_LP_EraseR_FLASH_HP_Erase

Table 1133:Parameters

Name	Direction	Description
p_ctrl	in	Control for the FLASH device.
address	in	The block containing this address is the first block erased.
num_blocks	in	Specifies the number of blocks to be erased, the starting block determined by the block_erase_address.

Parameter p_ctrl

Definition: flash_ctrl_t*const p_ctrl

Flash control block. Allocate an instance specific control block to pass into the flash API calls. Implemented asflash_lp_instance_ctrl_tflash_hp_instance_ctrl_t

Parameter address

uint32_t

Parameter num_blocks

uint32_t

8.16.8.11 blankCheck

```
ssp_err_t(* flash_api_t::blankCheck) (flash_ctrl_t *const p_ctrl, uint32_t const
address, uint32_t const num_bytes, flash_result_t *const p_blank_check_result)
```

Detailed description

Blank check FLASH device. Implemented as

- R_FLASH_LP_BlankCheck
- R_FLASH_HP_BlankCheck

Table 1134:Parameters

Name	Direction	Description
p_ctrl	in	Control for the FLASH device context.
address	in	The starting address of the Flash area to blank check.
num_bytes	in	Specifies the number of bytes that need to be checked. See the specific handler for details.
p_blank_check_result	out	Pointer that will be populated by the API with the results of the blank check operation in non-BGO (blocking) mode. In this case the blank check operation completes here and the result is returned. In Data Flash BGO mode the blank check operation is only started here and the result obtained later when the supplied callback routine is called. In this case FLASH_RESULT_BGO_ACTIVE will be returned in p_blank_check_result.

Parameter p_ctrl

Definition: `flash_ctrl_t*const p_ctrl`

Flash control block. Allocate an instance specific control block to pass into the flash API calls. Implemented asflash_lp_instance_ctrl_tflash_hp_instance_ctrl_t

Parameter address

`uint32_t`

Parameter num_bytes

`uint32_t`

Parameter p_blank_check_result

Definition: `flash_result_t*const p_blank_check_result`

Result type for certain operations

8.16.8.12 infoGet

```
ssp_err_t(* flash_api_t::infoGet) (flash_ctrl_t *const p_ctrl, flash_info_t
*const p_info)
```

Detailed description

Close FLASH device. Implemented as

- R_FLASH_LP_InfoGet
- R_FLASH_HP_InfoGet

Table 1135:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to FLASH device control.
p_info	out	Pointer to FLASH info structure.

Parameter p_ctrl

Definition: `flash_ctrl_t*const p_ctrl`

Flash control block. Allocate an instance specific control block to pass into the flash API calls. Implemented asflash_lp_instance_ctrl_tflash_hp_instance_ctrl_t

Parameter p_info

Definition: `flash_info_t*const p_info`

Information about the flash blocks

- `flash_info_t::code_flash`

  Information about the code flash regions.

- `flash_info_t::data_flash`

  Information about the code flash regions.

8.16.8.13 close

```
ssp_err_t(* flash_api_t::close) (flash_ctrl_t *const p_ctrl)
```

Detailed description

Close FLASH device. Implemented as

- R_FLASH_LP_Close
- R_FLASH_HP_Close

Table 1136:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to FLASH device control.

Parameter p_ctrl

Definition: `flash_ctrl_t*const p_ctrl`

Flash control block. Allocate an instance specific control block to pass into the flash API calls. Implemented asflash_lp_instance_ctrl_tflash_hp_instance_ctrl_t

8.16.8.14 statusGet

```
ssp_err_t(* flash_api_t::statusGet) (flash_ctrl_t *const p_ctrl)
```

Detailed description

Get Status for FLASH device. Implemented as

- R_FLASH_LP_StatusGet
- R_FLASH_HP_StatusGet

Table 1137:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to FLASH device control.

Parameter p_ctrl

Definition: `flash_ctrl_t*const p_ctrl`

Flash control block. Allocate an instance specific control block to pass into the flash API calls. Implemented asflash_lp_instance_ctrl_tflash_hp_instance_ctrl_t

8.16.8.15 accessWindowSet

```
ssp_err_t(* flash_api_t::accessWindowSet) (flash_ctrl_t *const p_ctrl, uint32_t
const start_addr, uint32_t const end_addr)
```

Detailed description

Set Access Window for FLASH device. Implemented as

- R_FLASH_LP_AccessWindowSet
- R_FLASH_HP_AccessWindowSet

Table 1138:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to FLASH device control.
start_addr	in	Determines the Starting block for the Code Flash access window.
end_addr	in	Determines the Ending block for the Code Flash access window.

Parameter p_ctrl

Definition: `flash_ctrl_t*const p_ctrl`

Flash control block. Allocate an instance specific control block to pass into the flash API calls. Implemented asflash_lp_instance_ctrl_tflash_hp_instance_ctrl_t

Parameter start_addr

`uint32_t`

Parameter end_addr

`uint32_t`

8.16.8.16 accessWindowClear

```
ssp_err_t(* flash_api_t::accessWindowClear) (flash_ctrl_t *const p_ctrl)
```

Detailed description

Clear any existing Code Flash access window for FLASH device. Implemented as

- R_FLASH_LP_AccessWindowClear
- R_FLASH_HP_AccessWindowClear

Table 1139:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to FLASH device control.
start_addr	in	Determines the Starting block for the Code Flash access window.
end_addr	in	Determines the Ending block for the Code Flash access window.

Parameter p_ctrl

Definition: `flash_ctrl_t*const p_ctrl`

Flash control block. Allocate an instance specific control block to pass into the flash API calls. Implemented asflash_lp_instance_ctrl_tflash_hp_instance_ctrl_t

Parameter start_addr
Parameter end_addr

8.16.8.17 reset

```
ssp_err_t(* flash_api_t::reset) (flash_ctrl_t *const p_ctrl)
```

Detailed description

Reset function for FLASH device. Implemented as

- R_FLASH_LP_Reset

- R_FLASH_HP_Reset

Table 1140:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to FLASH device control.

Parameter p_ctrl

Definition: `flash_ctrl_t*const p_ctrl`

Flash control block. Allocate an instance specific control block to pass into the flash API calls. Implemented asflash_lp_instance_ctrl_tflash_hp_instance_ctrl_t

8.16.8.18 updateFlashClockFreq

```
ssp_err_t(* flash_api_t::updateFlashClockFreq) (flash_ctrl_t *const p_ctrl)
```

Detailed description

Update Flash clock frequency (FCLK) and recalculate timeout values Implemented as

- R_FLASH_LP_UpdateFlashClockFreq
- R_FLASH_HP_UpdateFlashClockFreq

Table 1141:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to FLASH device control.

Parameter p_ctrl

Definition: `flash_ctrl_t*const p_ctrl`

Flash control block. Allocate an instance specific control block to pass into the flash API calls. Implemented asflash_lp_instance_ctrl_tflash_hp_instance_ctrl_t

8.16.8.19 startupAreaSelect

```
ssp_err_t(* flash_api_t::startupAreaSelect) (flash_ctrl_t *const p_ctrl,
flash_startup_area_swap_t swap_type, bool is_temporary)
```

Detailed description

Select which block - Default (Block 0) or Alternate (Block 1) is used as the start-up area block. Implemented as

- R_FLASH_LP_StartUpAreaSelect
- R_FLASH_HP_StartUpAreaSelect

Table 1142:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to FLASH device control.
swap_type	in	FLASH_STARTUP_AREA_BLOCK0, FLASH_STARTUP_AREA_BLOCK1 or FLASH_STARTUP_AREA_BTFLG.
is_temporary	in	True or false. See table below.

swap_type | is_temporary | Operation FLASH_STARTUP_AREA_BLOCK0: false On next reset Startup area will be Block 0.

FLASH_STARTUP_AREA_BLOCK0 | false | On next reset Startup area will be Block 0. Block 0.

FLASH_STARTUP_AREA_BLOCK1: false On next reset Startup area will be Block 1.

FLASH_STARTUP_AREA_BLOCK1 | true | Startup area is immediately, but temporarily switched to Block 1. Block 1.

FLASH_STARTUP_AREA_BTFLG | true | Startup area is immediately, but temporarily switched to... taken.

the Block determined by the Configuration BTFLG.

Parameter p_ctrl

Definition: `flash_ctrl_t*const p_ctrl`

Flash control block. Allocate an instance specific control block to pass into the flash API calls. Implemented asflash_lp_instance_ctrl_tflash_hp_instance_ctrl_t

Parameter swap_type

Definition: `flash_startup_area_swap_tswap_type`

Parameter for specifying the startup area swap being requested by startupAreaSelect()

Parameter is_temporary

`const`

8.16.8.20 versionGet

```
ssp_err_t(* flash_api_t::versionGet) (ssp_version_t *p_version)
```

Detailed description

Get Flash driver version. Implemented as

- R_FLASH_LP_VersionGet
- R_FLASH_HP_VersionGet

Table 1143:Parameters

Name	Direction	Description
p_version	out	Returns version.

Parameter p_version

8.16.8.21 flash_instance_t

flash_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- flash_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- flash_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- flash_api_t const * p_api

  Pointer to the API structure for this instance.

8.17 FMI Interface

Interface for reading on-chip factory information.

8.17.1 Summary

The FMI (Factory MCU Information) module provides a function for reading the Product Information record.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

FMI Interface description: FMI Driver

8.17.2 Interface API

fmi_api_t

Function name	Description
.init	Initialize the FMI base pointer.
.productInfoGet	Get product information record address into caller's pointer.
.uniqueIdGet	Copy unique ID into p_unique_id.
.productFeatureGet	Get feature information and store it in p_info.
.eventInfoGet	Get event information and store it in p_info.
.versionGet	Get the driver version based on compile time macros.

8.17.3 Data structures

- fmi_header_t
- fmi_product_info_t
- fmi_unique_id_t
- fmi_feature_info_t
- fmi_event_info_t
- fmi_instance_t

8.17.4 Defines

- #define FMI_API_VERSION_MAJOR

  Initial value: `(1U)`

  Register definitions, common services and error codes.
- #define FMI_API_VERSION_MINOR

  Initial value: `(3U)`

8.17.5 API Structures

8.17.5.1 fmi_header_t

fmi_header_t

Detailed description
Variables

- uint32_t contents
- uint32_t variant
- uint32_t count
- uint32_t minor
- uint32_t major

8.17.5.2 fmi_product_info_t

fmi_product_info_t

Detailed description
Variables

- fmi_header_t header
- uint8_t unique_id[16]

  DEPRECATED, use uniqueIdGet instead.
- uint8_t product_name[16]
- uint8_t product_marking[16]
- uint32_t mask_revision
- uint32_t pin_count
- uint32_t pkg_type
- uint32_t temp_range
- uint32_t quality_code
- uint32_t reserved
- struct{} struct{}

  See source code for definition of this member.
- uint32_t max_freq
- uint32_t reserved1
- struct{} struct{}

  See source code for definition of this member.

8.17.5.3 fmi_unique_id_t

fmi_unique_id_t

Detailed description

Variables

- uint32_t unique_id[4]

8.17.5.4 fmi_feature_info_t

fmi_feature_info_t

Detailed description

Variables

- void * ptr
- uint32_t channel_count
- uint32_t variant_data
- uint32_t extended_data_count
- uint32_t version_major
- uint32_t version_minor
- struct{} struct{}

  See source code for definition of this member.
- void * ptr_extended_data

8.17.5.5 fmi_event_info_t

fmi_event_info_t

Detailed description

Variables

- IRQn_Type irq
- elc_event_t event

8.17.5.6 fmi_api_t

fmi_api_t

Detailed description

fmi driver structure. General fmi functions implemented at the HAL layer will follow this API.

8.17.5.7 init

```
(* fmi_api_t::init) (void)
```

Detailed description

Initialize the FMI base pointer. Implemented as

- R_FMI_Init()

8.17.5.8 productInfoGet

```
(* fmi_api_t::productInfoGet) (fmi_product_info_t **pp_product_info)
```

Detailed description

Get product information record address into caller's pointer.

ATTENTION: fmi_product_info_t::unique_id is deprecated and will not contain a unique ID if the factory MCU information is linked in by the application code. Use uniqueIdGet for the unique ID.

Table 1144:Parameters

Name	Direction	Description
pp_product_info	inout	Pointer to store pointer to product info.

Implemented as

- R_FMI_ProductInfoGet()

Parameter pp_product_info

Definition: `fmi_product_info_t**pp_product_info`

- `fmi_product_info_t::header`
- `fmi_product_info_t::unique_id`

  DEPRECATED, use uniqueIdGet instead.
- `fmi_product_info_t::product_name`
- `fmi_product_info_t::product_marking`
- `fmi_product_info_t::mask_revision`
- `fmi_product_info_t::pin_count`

- fmi_product_info_t::pkg_type
- fmi_product_info_t::temp_range
- fmi_product_info_t::quality_code
- fmi_product_info_t::reserved
- fmi_product_info_t::struct{}
- fmi_product_info_t::max_freq
- fmi_product_info_t::reserved1
- fmi_product_info_t::struct{}

8.17.5.9 uniqueIdGet

```
(* fmi_api_t::uniqueIdGet) (fmi_unique_id_t *p_unique_id)
```

Detailed description

Copy unique ID into p_unique_id.

Table 1145:Parameters

Name	Direction	Description
p_unique_id	out	Pointer to unique ID.

Implemented as

- R_FMI_UniqueIdGet()

Parameter p_unique_id

Definition: fmi_unique_id_t*p_unique_id

- fmi_unique_id_t::unique_id

8.17.5.10 productFeatureGet

```
(* fmi_api_t::productFeatureGet) ( const *const p_feature, fmi_feature_info_t
*const p_info)
```

Detailed description

Get feature information and store it in p_info.

Table 1146:Parameters

Name	Direction	Description
p_feature	in	Definition of SSP feature.
p_info	out	Feature specific information.

Implemented as

- R_FMI_ProductFeatureGet()

Parameter p_feature
Parameter p_info

Definition: fmi_feature_info_t*const p_info

- fmi_feature_info_t::ptr
- fmi_feature_info_t::channel_count
- fmi_feature_info_t::variant_data
- fmi_feature_info_t::extended_data_count
- fmi_feature_info_t::version_major
- fmi_feature_info_t::version_minor
- fmi_feature_info_t::struct{}
- fmi_feature_info_t::ptr_extended_data

8.17.5.11 eventInfoGet

```
(* fmi_api_t::eventInfoGet) ( const *const p_feature, signal, fmi_event_info_t
*const p_info)
```

Detailed description

Get event information and store it in p_info.

Table 1147:Parameters

Name	Direction	Description
p_feature	in	Definition of SSP feature.
signal	in	Feature signal ID.
p_info	out	Event information for feature signal.

Implemented as

- R_FMI_EventInfoGet()

Parameter p_feature
Parameter signal
Parameter p_info

Definition: `fmi_event_info_t*const p_info`

- `fmi_event_info_t::irq`
- `fmi_event_info_t::event`

8.17.5.12 versionGet

```
(* fmi_api_t::versionGet) ( *const p_version)
```

Detailed description

Get the driver version based on compile time macros. Implemented as

- R_FMI_VersionGet()

8.17.5.13 fmi_instance_t

fmi_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- fmi_api_t const * p_api

  Pointer to the API structure for this instance.

8.18 HASH Algorithm Interface

HASH_Interface Hash algorithm APIs.

8.18.1 Interface API

hash_api_t

Function name	Description
.open	HASH_Interface: Initial configuration
.updateHash	update hash for the words from source buffer .This function will be deprecated. The function hashUpdate should be used instead.
.hashUpdate	update hash for the words from source buffer .
.close	
.versionGet	Gets version and stores it in provided pointer p_version.

8.18.2 Data structures

- hash_cfg_t
- hash_ctrl_t
- hash_instance_t

8.18.3 Variables

- g_md5_hash_on_sce
- g_sha1_hash_on_sce
- g_sha256_hash_on_sce

8.18.4 Defines

- #define HASH_API_VERSION_MAJOR

  Initial value: `(1U)`

  Register definitions, common services and error codes.
- #define HASH_API_VERSION_MINOR

  Initial value: `(0U)`
- #define HASH_MESSAGE_BLOCK_SIZE_WORDS

  Initial value: `(16U)`
- #define HASH_MESSAGE_BLOCK_SIZE_BYTES

  Initial value: `((HASH_MESSAGE_BLOCK_SIZE_WORDS) * 4)`
- #define HASH_MAX_DIGEST_SIZE_WORDS

  Initial value: `(8)`
- #define HASH_MAX_DIGEST_SIZE_BYTES

  Initial value: `((HASH_DIGEST_SIZE_WORDS) * 4)`

8.18.5 g_md5_hash_on_sce

`hash_api_t::g_md5_hash_on_sce`

8.18.5.1 Detailed description

HASH interface is only available on S7G2, S5D9 and S5D5.
MD5 implementation of HASH API.

8.18.6 g_sha1_hash_on_sce

`hash_api_t::g_sha1_hash_on_sce`

8.18.6.1 Detailed description

SHA1 implementation of HASH API.

8.18.7 g_sha256_hash_on_sce

`hash_api_t::g_sha256_hash_on_sce`

8.18.7.1 Detailed description

SHA256 implementation of HASH API.

8.18.8 API Structures

8.18.8.1 hash_cfg_t

hash_cfg_t

Detailed description

HASH_Interface configuration structure

Variables

- crypto_ctrl_t * p_crypto_ctrl
  pointer to crypto engine control structure
- crypto_api_t const * p_crypto_api
  pointer to crypto engine API structure

8.18.8.2 hash_ctrl_t

hash_ctrl_t

Detailed description

HASH_Interface control structure

Variables

- uint32_t msgbuf[HASH_MESSAGE_BLOCK_SIZE_WORDS]
  message buffer to be hashed
- uint32_t hash[HASH_MAX_DIGEST_SIZE_WORDS]
  current hash value
- uint64_t length
  64-bit message length (number of bits)

8.18.8.3 hash_api_t

hash_api_t

Detailed description

HASH_Interface SCE functions implemented at the HAL layer will follow this API.

8.18.8.4 open

```
uint32_t(* hash_api_t::open) (hash_ctrl_t *const p_ctrl, hash_cfg_t const *const p_cfg)
```

Detailed description

HASH_Interface: Initial configuration

NOTE: HASH_Interface: Peripheral clocks and any required output pins should be configured prior to calling this function.

Table 1148:Parameters

Name	Direction	Description
HASH_Interface	inout	p_ctrl Pointer to control structure. Must be declared by user. Elements set here.
HASH_Interface	in	p_cfg Pointer to configuration structure. All elements of this structure must be set by user.

Parameter HASH_Interface
Parameter HASH_Interface

8.18.8.5 updateHash

```
uint32_t(* hash_api_t::updateHash) (const uint32_t *p_source, uint32_t num_words, uint32_t *p_dest)
```

Detailed description

update hash for the num_words words from source buffer p_source.

Table 1149:Parameters

Name	Direction	Description
p_source	in	pointer to input message buffer. size must be a multiple of 64-bytes
num_words	in	number of words to be hashed from the buffer p_source
p_dest	inout	pointer to the message digest. on input contains initialization value.

This function will be deprecated. The function hashUpdate should be used instead.

Parameter p_source

`uint32_t`

Parameter num_words

`uint32_t`

Parameter p_dest

`uint32_t`

8.18.8.6 hashUpdate

```
uint32_t(* hash_api_t::hashUpdate) (hash_ctrl_t *const p_ctrl, const uint32_t
*p_source, uint32_t num_words, uint32_t *p_dest)
```

Detailed description

update hash for the num_words words from source buffer p_source.

Table 1150:Parameters

Name	Direction	Description
p_ctrl	in	pointer to hash_ctrl_t control structure.
p_source	in	pointer to input message buffer. size must be a multiple of 64-bytes
num_words	in	number of words to be hashed from the buffer p_source
p_dest	inout	pointer to the message digest. on input contains initialization value.

Parameter p_ctrl

Definition: `hash_ctrl_t`

HASH_Interface control structure

Parameter p_source

`uint32_t`

Parameter num_words

`uint32_t`

Parameter p_dest

`uint32_t`

8.18.8.7 close

```
uint32_t(* hash_api_t::close) (hash_ctrl_t *const p_ctrl)
```

8.18.8.8 versionGet

```
uint32_t(* hash_api_t::versionGet) ( *const p_version)
```

Detailed description

Gets version and stores it in provided pointer p_version.

Table 1151:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.18.8.9 hash_instance_t

hash_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- hash_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- hash_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- hash_api_t const * p_api

  Pointer to the API structure for this instance.

8.19 I2C Master Interface

Interface for I2C communication.

8.19.1 Summary

The I2C master interface provides a common API for I2C HAL drivers. The I2C master interface supports:

- Interrupt driven transmit/receive processing

- Callback function support which can return an event code

Implemented by:

- Simple I2C on SCI
- IIC

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

I2C Interface description: I2C Master Driver and I2C Slave Driver

8.19.2 Interface API

i2c_api_master_t

Function name	Description
.open	Opens the I2C driver and initializes the hardware.
.close	Closes the driver and releases the I2C device.
.read	Performs a read operation on an I2C device.
.write	Performs a write operation on an I2C device.
.reset	Performs a reset of the peripheral.
.slaveAddressSet	Sets address of the slave device without reconfiguring the bus.
.versionGet	Gets version information and stores it in the provided version struct.

i2c_api_slave_t

Function name	Description
.open	Opens the I2C driver and initializes the hardware.
.close	Closes the driver and releases the I2C device.
.masterWriteSlaveRead	Performs a read operation on an I2C device.

Function name	Description
.masterReadSlaveWrite	Performs a write operation on an I2C device.
.versionGet	Gets version information and stores it in the provided version struct.

8.19.3 Data structures

- i2c_callback_args_t
- i2c_cfg_t
- i2c_api_master_t
- i2c_api_slave_t
- i2c_master_instance_t
- i2c_slave_instance_t

8.19.4 Enumerations

- i2c_rate_t
- i2c_addr_mode_t
- i2c_event_t

8.19.5 Typedefs

- i2c_ctrl_t

8.19.6 Defines

- #define I2C_MASTER_API_VERSION_MAJOR
  Initial value: `(1U)`
- #define I2C_MASTER_API_VERSION_MINOR
  Initial value: `(4U)`

8.19.7 API Data

8.19.7.1 i2c_rate_t

```
i2c_rate_t
```

Detailed description

Communication speed options

Enumerated values

Name	Description
I2C_RATE_STANDARD	100 kHz
I2C_RATE_FAST	400 kHz
I2C_RATE_FASTPLUS	1 MHz

8.19.7.2 i2c_addr_mode_t

`i2c_addr_mode_t`

Detailed description

Addressing mode options

Enumerated values

Name	Description
I2C_ADDR_MODE_7BIT	Use 7-bit addressing mode.
I2C_ADDR_MODE_10BIT	Use 10-bit addressing mode.

8.19.7.3 i2c_event_t

`i2c_event_t`

Detailed description

Callback events

Enumerated values

Name	Description
I2C_EVENT_ABORTED	A transfer was aborted.
I2C_EVENT_RX_COMPLETE	A receive operation was completed successfully.

Name	Description
I2C_EVENT_TX_COMPLETE	A transmit operation was completed successfully.

8.19.7.4 i2c_ctrl_t

```
typedef void i2c_ctrl_t
```

Detailed description

I2C control block. Allocate an instance specific control block to pass into the I2C API calls. Implemented as

- sci_i2c_instance_ctrl_t
- riic_instance_ctrl_t

8.19.8 API Structures

8.19.8.1 i2c_callback_args_t

i2c_callback_args_t

Detailed description

I2C callback parameter definition

Variables

- void const *const p_context

  Pointer to user-provided context.
- uint32_t const bytes

  Number of received/transmitted bytes in buff.
- i2c_event_t const event

  Event code.

8.19.8.2 i2c_cfg_t

i2c_cfg_t

Detailed description

I2C configuration block

Variables

- uint8_t channel

  Identifier recognizable by implementation.

  Generic configuration
- i2c_rate_t rate

  Device's maximum clock rate from enum i2c_rate_t.
- uint16_t slave

  The address of the slave device.
- i2c_addr_mode_t addr_mode

  Indicates how slave fields should be interpreted.
- uint16_t sda_delay

  The SDA output delay.
- uint8_t rxi_ipl

  Receive interrupt priority.
- uint8_t txi_ipl

  Transmit interrupt priority.
- uint8_t tei_ipl

  Transmit end interrupt priority.
- uint8_t eri_ipl

  Error interrupt priority.
- transfer_instance_t const * p_transfer_tx

  DTC instance for I2C transmit.Set to NULL if unused.

  DTC/DMA support
- transfer_instance_t const * p_transfer_rx

  DTC instance for I2C receive. Set to NULL if unused.
- void(* p_callback)(i2c_callback_args_t *p_args)

  Pointer to callback function.

  Parameters to control software behavior
- void const * p_context

  Pointer to the user-provided context.
- void const * p_extend

  Any configuration data needed by the hardware.

  Implementation-specific configuration

8.19.8.3 i2c_api_master_t

i2c_api_master_t

Detailed description

Interface definition for I2C access as master

8.19.8.4 open

```
ssp_err_t(* i2c_api_master_t::open) (i2c_ctrl_t *const p_ctrl, i2c_cfg_t const
*const p_cfg)
```

Detailed description

Opens the I2C driver and initializes the hardware. Implemented as

- R_RIIC_MasterOpen
- R_SCI_SIIC_MasterOpen

Table 1152:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block. Must be declared by user. Elements are set here.
p_cfg	in	Pointer to configuration structure.

Parameter p_ctrl

Definition: `i2c_ctrl_t*const p_ctrl`

I2C control block. Allocate an instance specific control block to pass into the I2C API calls. Implemented assci_i2c_instance_ctrl_triic_instance_ctrl_t

Parameter p_cfg

Definition: `i2c_cfg_t const *const p_cfg`

I2C configuration block

- `i2c_cfg_t`::channel

  Identifier recognizable by implementation.

  Generic configuration

- `i2c_cfg_t`::`i2c_rate_t`

  Device's maximum clock rate from enum i2c_rate_t.

  Enumerated as:

  - I2C_RATE_STANDARD
  - I2C_RATE_FAST
  - I2C_RATE_FASTPLUS
- i2c_cfg_t::slave

  The address of the slave device.
- i2c_cfg_t::i2c_addr_mode_t

  Indicates how slave fields should be interpreted.

  Enumerated as:
  - I2C_ADDR_MODE_7BIT
  - I2C_ADDR_MODE_10BIT
- i2c_cfg_t::sda_delay

  The SDA output delay.
- i2c_cfg_t::rxi_ipl

  Receive interrupt priority.
- i2c_cfg_t::txi_ipl

  Transmit interrupt priority.
- i2c_cfg_t::tei_ipl

  Transmit end interrupt priority.
- i2c_cfg_t::eri_ipl

  Error interrupt priority.
- i2c_cfg_t::transfer_instance_t

  DTC instance for I2C transmit.Set to NULL if unused.

  DTC/DMA support
- i2c_cfg_t::transfer_instance_t

  DTC instance for I2C receive. Set to NULL if unused.
- i2c_cfg_t::p_callback

  Pointer to callback function.

  Parameters to control software behavior
- i2c_cfg_t::p_context

  Pointer to the user-provided context.
- i2c_cfg_t::p_extend

  Any configuration data needed by the hardware.

  Implementation-specific configuration

8.19.8.5 close

```
ssp_err_t(* i2c_api_master_t::close) (i2c_ctrl_t *const p_ctrl)
```

Detailed description

Closes the driver and releases the I2C device. Implemented as

- R_RIIC_MasterClose
- R_SCI_SIIC_MasterClose

Table 1153:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block set in open call.

Parameter p_ctrl

Definition: `i2c_ctrl_t*const p_ctrl`

I2C control block. Allocate an instance specific control block to pass into the I2C API calls. Implemented assci_i2c_instance_ctrl_triic_instance_ctrl_t

8.19.8.6 read

```
ssp_err_t(* i2c_api_master_t::read) (i2c_ctrl_t *const p_ctrl, uint8_t *const
p_dest, uint32_t const bytes, bool const restart)
```

Detailed description

Performs a read operation on an I2C device. Implemented as

- R_RIIC_MasterRead
- R_SCI_SIIC_MasterRead

Table 1154:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block set in open call.
p_dest	in	Pointer to the location to store read data.
bytes	in	Number of bytes to read.

Table 1154:Parameters (Continued)

Name	Direction	Description
restart	in	Specify if the restart condition should be issued after reading.

Parameter p_ctrl

Definition: `i2c_ctrl_t*const p_ctrl`

I2C control block. Allocate an instance specific control block to pass into the I2C API calls. Implemented assci_i2c_instance_ctrl_triic_instance_ctrl_t

Parameter p_dest

`uint8_t`

Parameter bytes

`uint32_t`

Parameter restart

`const`

8.19.8.7 write

```
ssp_err_t(* i2c_api_master_t::write) (i2c_ctrl_t *const p_ctrl, uint8_t *const
p_src, uint32_t const bytes, bool const restart)
```

Detailed description

Performs a write operation on an I2C device. Implemented as

- R_RIIC_MasterWrite
- R_SCI_SIIC_MasterWrite

Table 1155:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block set in open call.
p_src	in	Pointer to the location to get write data from.
bytes	in	Number of bytes to write.
restart	in	Specify if the restart condition should be issued after writing.

Parameter p_ctrl

Definition: i2c_ctrl_t*const p_ctrl

I2C control block. Allocate an instance specific control block to pass into the I2C API calls. Implemented assci_i2c_instance_ctrl_triic_instance_ctrl_t

Parameter p_src

uint8_t

Parameter bytes

uint32_t

Parameter restart

const

8.19.8.8 reset

```
ssp_err_t(* i2c_api_master_t::reset) (i2c_ctrl_t *const p_ctrl)
```

Detailed description

Performs a reset of the peripheral. Implemented as

- R_RIIC_MasterReset
- R_SCI_SIIC_MasterReset

Table 1156:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block set in open call.

Parameter p_ctrl

Definition: i2c_ctrl_t*const p_ctrl

I2C control block. Allocate an instance specific control block to pass into the I2C API calls. Implemented assci_i2c_instance_ctrl_triic_instance_ctrl_t

8.19.8.9 slaveAddressSet

```
ssp_err_t(* i2c_api_master_t::slaveAddressSet) (i2c_ctrl_t *const p_ctrl,
uint16_t const slave, i2c_addr_mode_t const addr_mode)
```

Detailed description

Sets address of the slave device without reconfiguring the bus. Implemented as

- R_RIIC_MasterSlaveAddressSet

- R_SCI_SIIC_MasterSlaveAddressSet

Table 1157:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block set in open call.
slave_address	in	Address of the slave device.
address_mode	in	Addressing mode.

Parameter p_ctrl

Definition: `i2c_ctrl_t*const p_ctrl`

I2C control block. Allocate an instance specific control block to pass into the I2C API calls. Implemented assci_i2c_instance_ctrl_triic_instance_ctrl_t

Parameter slave_address

Parameter address_mode

8.19.8.10 versionGet

```
ssp_err_t(* i2c_api_master_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Gets version information and stores it in the provided version struct. Implemented as

- R_RIIC_MasterVersionGet
- R_SCI_SIIC_MasterVersionGet

Table 1158:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.19.8.11 i2c_api_slave_t

i2c_api_slave_t

Detailed description

Interface definition for I2C access as slave

8.19.8.12 open

```
ssp_err_t(* i2c_api_slave_t::open) (i2c_ctrl_t *const p_ctrl, i2c_cfg_t const
*const p_cfg)
```

Detailed description

Opens the I2C driver and initializes the hardware. Implemented as

- R_RIIC_SlaveOpen

Table 1159:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block. Must be declared by user. Elements are set here.
p_cfg	in	Pointer to configuration structure.

Parameter p_ctrl

Definition: `i2c_ctrl_t*const p_ctrl`

I2C control block. Allocate an instance specific control block to pass into the I2C API calls. Implemented assci_i2c_instance_ctrl_triic_instance_ctrl_t

Parameter p_cfg

Definition: `i2c_cfg_t const *const p_cfg`

I2C configuration block

- `i2c_cfg_t`::`channel`

  Identifier recognizable by implementation.

  Generic configuration

- `i2c_cfg_t`::`i2c_rate_t`

  Device's maximum clock rate from enum i2c_rate_t.

  Enumerated as:

  – I2C_RATE_STANDARD

  – I2C_RATE_FAST

  – I2C_RATE_FASTPLUS

- i2c_cfg_t::slave

  The address of the slave device.
- i2c_cfg_t::i2c_addr_mode_t

  Indicates how slave fields should be interpreted.

  Enumerated as:

  – I2C_ADDR_MODE_7BIT

  – I2C_ADDR_MODE_10BIT
- i2c_cfg_t::sda_delay

  The SDA output delay.
- i2c_cfg_t::rxi_ipl

  Receive interrupt priority.
- i2c_cfg_t::txi_ipl

  Transmit interrupt priority.
- i2c_cfg_t::tei_ipl

  Transmit end interrupt priority.
- i2c_cfg_t::eri_ipl

  Error interrupt priority.
- i2c_cfg_t::transfer_instance_t

  DTC instance for I2C transmit.Set to NULL if unused.

  DTC/DMA support
- i2c_cfg_t::transfer_instance_t

  DTC instance for I2C receive. Set to NULL if unused.
- i2c_cfg_t::p_callback

  Pointer to callback function.

  Parameters to control software behavior
- i2c_cfg_t::p_context

  Pointer to the user-provided context.
- i2c_cfg_t::p_extend

  Any configuration data needed by the hardware.

  Implementation-specific configuration

8.19.8.13 close

```
ssp_err_t(* i2c_api_slave_t::close) (i2c_ctrl_t *const p_ctrl)
```

Detailed description

Closes the driver and releases the I2C device. Implemented as

- R_RIIC_SlaveClose

Table 1160:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block set in open call.

Parameter p_ctrl

Definition: `i2c_ctrl_t*const p_ctrl`

I2C control block. Allocate an instance specific control block to pass into the I2C API calls. Implemented assci_i2c_instance_ctrl_triic_instance_ctrl_t

8.19.8.14 masterWriteSlaveRead

```
ssp_err_t(* i2c_api_slave_t::masterWriteSlaveRead) (i2c_ctrl_t *const p_ctrl,
uint8_t *const p_dest, uint32_t const bytes)
```

Detailed description

Performs a read operation on an I2C device. Implemented as

- R_RIIC_MasterWriteSlaveRead

Table 1161:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block set in open call.
p_dest	in	Pointer to the location to store read data.
bytes	in	Number of bytes to read.

Parameter p_ctrl

Definition: `i2c_ctrl_t*const p_ctrl`

I2C control block. Allocate an instance specific control block to pass into the I2C API calls. Implemented assci_i2c_instance_ctrl_triic_instance_ctrl_t

Parameter p_dest

`uint8_t`

Parameter bytes

`uint32_t`

8.19.8.15 masterReadSlaveWrite

```
ssp_err_t(* i2c_api_slave_t::masterReadSlaveWrite) (i2c_ctrl_t *const p_ctrl,
uint8_t *const p_src, uint32_t const bytes)
```

Detailed description

Performs a write operation on an I2C device. Implemented as

- R_RIIC_MasterReadSlaveWrite

Table 1162:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block set in open call.
p_src	in	Pointer to the location to get write data from.
bytes	in	Number of bytes to write.

Parameter p_ctrl

Definition: `i2c_ctrl_t*const p_ctrl`

I2C control block. Allocate an instance specific control block to pass into the I2C API calls. Implemented assci_i2c_instance_ctrl_triic_instance_ctrl_t

Parameter p_src

`uint8_t`

Parameter bytes

`uint32_t`

8.19.8.16 versionGet

```
ssp_err_t(* i2c_api_slave_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Gets version information and stores it in the provided version struct. Implemented as

- R_RIIC_SlaveVersionGet

Table 1163:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.19.8.17 i2c_master_instance_t

i2c_master_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- i2c_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- i2c_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- i2c_api_master_t const * p_api

  Pointer to the API structure for this instance.

8.19.8.18 i2c_slave_instance_t

i2c_slave_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- i2c_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- i2c_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- i2c_api_slave_t const * p_api

  Pointer to the API structure for this instance.

8.20 I2S Interface

The I2S (Inter-IC Sound) interface provides APIs and definitions for I2S audio communication.

8.20.1 Summary

The I2S (Inter-IC Sound) interface provides APIs and definitions for I2S audio communication.

8.20.2 Known Implementations

SSI

8.20.3 Related modules

See also: I2S Audio Playback Framework

8.20.4 Interface API

i2s_api_t

Function name	Description
.open	Initial configuration.
.stop	Stop communication. Transmission is stopped when callback is called with I2S_EVENT_IDLE. Reception is stopped when callback is called with I2S_EVENT_RX_EMPTY.
.mute	Enable or disable mute.
.write	Write I2S data. All transmit data is queued when callback is called with I2S_EVENT_TX_EMPTY. Transmission is complete when callback is called with I2S_EVENT_IDLE.
.read	Read I2S data. Reception is complete when callback is called with I2S_EVENT_RX_EMPTY.
.writeRead	Simultaneously write and read I2S data. Transmission and reception are complete when callback is called with I2S_EVENT_IDLE.
.infoGet	Get instance specific information and store it in provided pointer p_info.

Function name	Description
.close	Allows driver to be reconfigured and may reduce power consumption.
.versionGet	Get version and store it in provided pointer p_version.

8.20.5 Data structures

- i2s_callback_args_t
- i2s_info_t
- i2s_cfg_t
- i2s_instance_t

8.20.6 Enumerations

- i2s_pcm_width_t
- i2s_word_length_t
- i2s_event_t
- i2s_dir_t
- i2s_mute_t
- i2s_ws_continue_t
- i2s_status_t

8.20.7 Typedefs

- i2s_ctrl_t

8.20.8 Defines

- #define I2S_API_VERSION_MAJOR

  Initial value: `(1U)`

  Register definitions, common services and error codes.
- #define I2S_API_VERSION_MINOR

  Initial value: `(2U)`

8.20.9 API Data

8.20.9.1 i2s_pcm_width_t

`i2s_pcm_width_t`

Detailed description

Audio PCM width

Enumerated values

Name	Description
I2S_PCM_WIDTH_8_BITS	Using 8-bit PCM.
I2S_PCM_WIDTH_16_BITS	Using 16-bit PCM.
I2S_PCM_WIDTH_18_BITS	Using 18-bit PCM.
I2S_PCM_WIDTH_20_BITS	Using 20-bit PCM.
I2S_PCM_WIDTH_22_BITS	Using 22-bit PCM.
I2S_PCM_WIDTH_24_BITS	Using 24-bit PCM.

8.20.9.2 i2s_word_length_t

`i2s_word_length_t`

Detailed description

Audio system word length.

Enumerated values

Name	Description
I2S_WORD_LENGTH_8_BITS	Using 8-bit system word length.
I2S_WORD_LENGTH_16_BITS	Using 16-bit system word length.
I2S_WORD_LENGTH_24_BITS	Using 24-bit system word length.
I2S_WORD_LENGTH_32_BITS	Using 32-bit system word length.

8.20.9.3 i2s_event_t

`i2s_event_t`

Detailed description

Events that can trigger a callback function

Enumerated values

Name	Description
I2S_EVENT_IDLE	Communication is idle.
I2S_EVENT_TX_EMPTY	Transmit buffer is below FIFO trigger level.
I2S_EVENT_RX_FULL	Receive buffer is above FIFO trigger level.

8.20.9.4 i2s_dir_t

`i2s_dir_t`

Detailed description

I2S communication direction

Enumerated values

Name	Description
I2S_DIR_TX	Transmit direction only.
I2S_DIR_RX	Receive direction only.
I2S_DIR_TX_RX	Transmit and receive direction.

8.20.9.5 i2s_mute_t

`i2s_mute_t`

Detailed description

Mute audio samples.

Enumerated values

Name	Description
I2S_MUTE_ON	Enable mute.
I2S_MUTE_OFF	Disable mute.

8.20.9.6 i2s_ws_continue_t

`i2s_ws_continue_t`

Detailed description

Whether to continue WS (word select line) transmission during idle state.

Enumerated values

Name	Description
I2S_WS_CONTINUE_ON	Enable WS continue mode.
I2S_WS_CONTINUE_OFF	Disable WS continue mode.

8.20.9.7 i2s_status_t

`i2s_status_t`

Detailed description

Possible status values returned by infoGet.

Enumerated values

Name	Description
I2S_STATUS_IN_USE	I2S is in use.
I2S_STATUS_STOPPED	I2S is stopped.

8.20.9.8 i2s_ctrl_t

`typedef void i2s_ctrl_t`

Detailed description

I2S control block. Allocate an instance specific control block to pass into the I2S API calls. Implemented as

- ssi_instance_ctrl_t

8.20.10 API Structures

8.20.10.1 i2s_callback_args_t

i2s_callback_args_t

Detailed description

Callback function parameter data

Variables

- void const * p_context

  Placeholder for user data. Set in i2s_api_t::open function in i2s_cfg_t.

- i2s_event_t event

  The event can be used to identify what caused the callback (overflow or error).

8.20.10.2 i2s_info_t

i2s_info_t

Detailed description

Timer information structure to store various information for an I2S instance.

Variables

- i2s_status_t status
- uint32_t sampling_freq_hz

  Sampling frequency in Hertz.

8.20.10.3 i2s_cfg_t

i2s_cfg_t

Detailed description

User configuration structure, used in open function

Variables

- uint8_t channel

  Select a channel corresponding to the channel number of the hardware.

- i2s_pcm_width_t pcm_width

  Audio PCM data width.

- i2s_word_length_t word_length

  Audio word length, bits must be >= i2s_cfg_t::pcm_width bits.

- i2s_ws_continue_t ws_continue

  Whether to continue WS transmission during idle state.

- uint32_t sampling_freq_hz

  Sampling frequency in Hertz.

- uint32_t audio_clk_freq_hz

  Audio clock frequency in Hertz. Must be a multiple between 1 and 128 of (16 * i2s_cfg_t::sampling_freq_hz * (i2s_cfg_t::word_length <enum_value> + 1)

- timer_instance_t const * p_timer

  To generate audio clock with GPT, link a timer instance here. Set to NULL if unused.
- transfer_instance_t const * p_transfer_tx

  To use DTC during write, link a DTC instance here. Set to NULL if unused.
- transfer_instance_t const * p_transfer_rx

  To use DTC during read, link a DTC instance here. Set to NULL if unused.
- void(* p_callback)(i2s_callback_args_t *p_args)

  Callback provided when an I2S ISR occurs. Set to NULL for no CPU interrupt.
- void const * p_context

  Placeholder for user data. Passed to the user callback in i2s_callback_args_t.
- void const * p_extend

  Extension parameter for hardware specific settings.
- uint8_t rxi_ipl

  Receive interrupt priority.
- uint8_t txi_ipl

  Transmit interrupt priority.
- uint8_t idle_err_ipl

  Idle/Error interrupt priority.

8.20.10.4 i2s_api_t

i2s_api_t

Detailed description

I2S functions implemented at the HAL layer will follow this API.

8.20.10.5 open

```
ssp_err_t(* i2s_api_t::open) (i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const
p_cfg)
```

Detailed description

Initial configuration. Implemented as

- R_SSI_Open

NOTE: Peripheral clocks and any required output pins should be configured prior to calling this function.

NOTE: To reconfigure after calling this function, call close first.

Table 1164:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block. Must be declared by user. Elements set here.
p_cfg	in	Pointer to configuration structure. All elements of this structure must be set by user.

Parameter p_ctrl

Definition: `i2s_ctrl_t*const p_ctrl`

I2S control block. Allocate an instance specific control block to pass into the I2S API calls. Implemented asssi_instance_ctrl_t

Parameter p_cfg

Definition: `i2s_cfg_t const *const p_cfg`

User configuration structure, used in open function

- `i2s_cfg_t`::`channel`

  Select a channel corresponding to the channel number of the hardware.

- `i2s_cfg_t`::`i2s_pcm_width_t`

  Audio PCM data width.

  Enumerated as:

  – I2S_PCM_WIDTH_8_BITS

  – I2S_PCM_WIDTH_16_BITS

  – I2S_PCM_WIDTH_18_BITS

  – I2S_PCM_WIDTH_20_BITS

  – I2S_PCM_WIDTH_22_BITS

  – I2S_PCM_WIDTH_24_BITS

- `i2s_cfg_t`::`i2s_word_length_t`

  Audio word length, bits must be >= i2s_cfg_t::pcm_width bits.

  Enumerated as:

  – I2S_WORD_LENGTH_8_BITS

  - I2S_WORD_LENGTH_16_BITS
  - I2S_WORD_LENGTH_24_BITS
  - I2S_WORD_LENGTH_32_BITS
- i2s_cfg_t::i2s_ws_continue_t

  Whether to continue WS transmission during idle state.

  Enumerated as:

  - I2S_WS_CONTINUE_ON
  - I2S_WS_CONTINUE_OFF
- i2s_cfg_t::sampling_freq_hz

  Sampling frequency in Hertz.
- i2s_cfg_t::audio_clk_freq_hz

  Audio clock frequency in Hertz. Must be a multiple between 1 and 128 of (16 * i2s_cfg_t::sampling_freq_hz * (i2s_cfg_t::word_length <enum_value> + 1)
- i2s_cfg_t::timer_instance_t

  To generate audio clock with GPT, link a timer instance here. Set to NULL if unused.
- i2s_cfg_t::transfer_instance_t

  To use DTC during write, link a DTC instance here. Set to NULL if unused.
- i2s_cfg_t::transfer_instance_t

  To use DTC during read, link a DTC instance here. Set to NULL if unused.
- i2s_cfg_t::p_callback

  Callback provided when an I2S ISR occurs. Set to NULL for no CPU interrupt.
- i2s_cfg_t::p_context

  Placeholder for user data. Passed to the user callback in i2s_callback_args_t.
- i2s_cfg_t::p_extend

  Extension parameter for hardware specific settings.
- i2s_cfg_t::rxi_ipl

  Receive interrupt priority.
- i2s_cfg_t::txi_ipl

  Transmit interrupt priority.
- i2s_cfg_t::idle_err_ipl

  Idle/Error interrupt priority.

8.20.10.6 stop

```
ssp_err_t(* i2s_api_t::stop) (i2s_ctrl_t *const p_ctrl, i2s_dir_t const dir)
```

Detailed description

Stop communication. Transmission is stopped when callback is called with I2S_EVENT_IDLE. Reception is stopped when callback is called with I2S_EVENT_RX_EMPTY. Implemented as

- R_SSI_Stop

Table 1165:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this instance.
dir	in	Direction of communication to stop.

Parameter p_ctrl

Definition: `i2s_ctrl_t*const p_ctrl`

I2S control block. Allocate an instance specific control block to pass into the I2S API calls. Implemented asssi_instance_ctrl_t

Parameter dir

8.20.10.7 mute

```
ssp_err_t(* i2s_api_t::mute) (i2s_ctrl_t *const p_ctrl, i2s_mute_t const
mute_enable)
```

Detailed description

Enable or disable mute. Implemented as

- R_SSI_Mute

Table 1166:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this instance.
mute_enable	in	Whether to enable or disable mute.

Parameter p_ctrl

Definition: `i2s_ctrl_t*const p_ctrl`

I2S control block. Allocate an instance specific control block to pass into the I2S API calls. Implemented asssi_instance_ctrl_t

Parameter mute_enable

Definition: `i2s_mute_tconst mute_enable`

Mute audio samples.

8.20.10.8 write

```
ssp_err_t(* i2s_api_t::write) (i2s_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint16_t const bytes)
```

Detailed description

Write I2S data. All transmit data is queued when callback is called with I2S_EVENT_TX_EMPTY. Transmission is complete when callback is called with I2S_EVENT_IDLE. Implemented as

- R_SSI_Write

Table 1167:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this instance.
p_src	in	Buffer of PCM samples. Must be 4 byte aligned.
bytes	in	Number of bytes in the buffer. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, padding 0s will be added to transmission to make it a multiple of 8.

Parameter p_ctrl

Definition: `i2s_ctrl_t*const p_ctrl`

I2S control block. Allocate an instance specific control block to pass into the I2S API calls. Implemented asssi_instance_ctrl_t

Parameter p_src

`uint8_t`

Parameter bytes

`uint16_t`

8.20.10.9 read

```
ssp_err_t(* i2s_api_t::read) (i2s_ctrl_t *const p_ctrl, uint8_t *const p_dest,
uint16_t const bytes)
```

Detailed description

Read I2S data. Reception is complete when callback is called with I2S_EVENT_RX_EMPTY. Implemented as

- R_SSI_Read

Table 1168:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this instance.
p_dest	in	Buffer to store PCM samples. Must be 4 byte aligned.
bytes	in	Number of bytes in the buffer. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, receive will stop at the multiple of 8 below requested bytes.

Parameter p_ctrl

Definition: `i2s_ctrl_t*const p_ctrl`

I2S control block. Allocate an instance specific control block to pass into the I2S API calls. Implemented asssi_instance_ctrl_t

Parameter p_dest

`uint8_t`

Parameter bytes

`uint16_t`

8.20.10.10 writeRead

```
ssp_err_t(* i2s_api_t::writeRead) (i2s_ctrl_t *const p_ctrl, uint8_t const *const
p_src, uint8_t *const p_dest, uint16_t const bytes)
```

Detailed description

Simultaneously write and read I2S data. Transmission and reception are complete when callback is called with I2S_EVENT_IDLE. Implemented as

- R_SSI_WriteRead

Table 1169:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this instance.
p_src	in	Buffer of PCM samples. Must be 4 byte aligned.
p_dest	in	Buffer to store PCM samples. Must be 4 byte aligned.
bytes	in	Number of bytes in the buffers. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, padding 0s will be added to transmission to make it a multiple of 8, and receive will stop at the multiple of 8 below requested bytes.

Parameter p_ctrl

Definition: `i2s_ctrl_t*const p_ctrl`

I2S control block. Allocate an instance specific control block to pass into the I2S API calls. Implemented asssi_instance_ctrl_t

Parameter p_src

`uint8_t`

Parameter p_dest

`uint8_t`

Parameter bytes

`uint16_t`

8.20.10.11 infoGet

```
ssp_err_t(* i2s_api_t::infoGet) (i2s_ctrl_t *const p_ctrl, i2s_info_t *const
p_info)
```

Detailed description

Get instance specific information and store it in provided pointer p_info. Implemented as

- R_SSI_InfoGet

Table 1170:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this instance.
p_info	out	Collection of information for this instance.

Parameter p_ctrl

Definition: `i2s_ctrl_t*const p_ctrl`

I2S control block. Allocate an instance specific control block to pass into the I2S API calls. Implemented asssi_instance_ctrl_t

Parameter p_info

Definition: `i2s_info_t*const p_info`

Timer information structure to store various information for an I2S instance.

- `i2s_info_t::status`
- `i2s_info_t::sampling_freq_hz`

  Sampling frequency in Hertz.

8.20.10.12 close

```
ssp_err_t(* i2s_api_t::close) (i2s_ctrl_t *const p_ctrl)
```

Detailed description

Allows driver to be reconfigured and may reduce power consumption. Implemented as

- R_SSI_Close

Table 1171:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this instance.

Parameter p_ctrl

Definition: `i2s_ctrl_t*const p_ctrl`

I2S control block. Allocate an instance specific control block to pass into the I2S API calls. Implemented asssi_instance_ctrl_t

8.20.10.13 versionGet

```
ssp_err_t(* i2s_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Get version and store it in provided pointer p_version. Implemented as

- R_SSI_VersionGet

Table 1172:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.20.10.14 i2s_instance_t

i2s_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- i2s_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- i2s_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- i2s_api_t const * p_api

  Pointer to the API structure for this instance.

8.21 I/O Port Interface

Interface for accessing I/O ports and configuring I/O functionality.

The IOPort shared interface provides the ability to access the IOPorts of a device at both bit and port level. Port and pin direction can be changed.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

IOPORT Interface description: I/O Port Driver

8.21.1 Interface API

ioport_api_t

Function name	Description
.init	Initialize internal driver data and initial pin configurations. Called during startup. Do not call this API during runtime. Use pinsCfg for runtime reconfiguration of multiple pins.
.pinsCfg	Configure multiple pins.
.pinCfg	Configure settings for an individual pin.
.pinDirectionSet	Set the pin direction of a pin.
.pinEventInputRead	Read the event input data of the specified pin and return the level.
.pinEventOutputWrite	Write pin event data.
.pinEthernetModeCfg	Configure the PHY mode of the Ethernet channels.
.pinRead	Read level of a pin.
.pinWrite	Write specified level to a pin.
.portDirectionSet	Set the direction of one or more pins on a port.
.portEventInputRead	Read captured event data for a port.
.portEventOutputWrite	Write event output data for a port.
.portRead	Read states of pins on the specified port.
.portWrite	Write to multiple pins on a port.
.versionGet	Return the version of the IOPort driver.

8.21.2 Data structures

- ioport_pin_cfg_t
- ioport_cfg_t
- ioport_instance_t

8.21.3 Enumerations

- ioport_level_t
- ioport_direction_t
- ioport_port_t
- ioport_port_pin_t
- ioport_peripheral_t
- ioport_ethernet_channel_t
- ioport_ethernet_mode_t
- ioport_cfg_options_t

8.21.4 Typedefs

- ioport_size_t

8.21.5 Defines

- #define IOPORT_API_VERSION_MAJOR
  Initial value: `(1U)`
- #define IOPORT_API_VERSION_MINOR
  Initial value: `(2U)`
- #define IOPORT_PRV_PFS_PSEL_OFFSET
  Initial value: `(24)`

8.21.6 API Data

8.21.6.1 ioport_level_t

`ioport_level_t`

Detailed description

Levels that can be set and read for individual pins

Enumerated values

Name	Description
IOPORT_LEVEL_LOW	Low.
IOPORT_LEVEL_HIGH	High.

8.21.6.2 ioport_direction_t

`ioport_direction_t`

Detailed description

Direction of individual pins

Enumerated values

Name	Description
IOPORT_DIRECTION_INPUT	Input.
IOPORT_DIRECTION_OUTPUT	Output.

8.21.6.3 ioport_port_t

`ioport_port_t`

Detailed description

Superset list of all possible IO ports.

Enumerated values

Name	Description
IOPORT_PORT_00	IO port 0.
IOPORT_PORT_01	IO port 1.
IOPORT_PORT_02	IO port 2.
IOPORT_PORT_03	IO port 3.
IOPORT_PORT_04	IO port 4.
IOPORT_PORT_05	IO port 5.
IOPORT_PORT_06	IO port 6.
IOPORT_PORT_07	IO port 7.
IOPORT_PORT_08	IO port 8.
IOPORT_PORT_09	IO port 9.
IOPORT_PORT_10	IO port 10.

Name	Description
IOPORT_PORT_11	IO port 11.

8.21.6.4 ioport_port_pin_t

```
ioport_port_pin_t
```

Detailed description

Superset list of all possible IO port pins.

Enumerated values

Name	Description
IOPORT_PORT_00_PIN_00	IO port 0 pin 0.
IOPORT_PORT_00_PIN_01	IO port 0 pin 1.
IOPORT_PORT_00_PIN_02	IO port 0 pin 2.
IOPORT_PORT_00_PIN_03	IO port 0 pin 3.
IOPORT_PORT_00_PIN_04	IO port 0 pin 4.
IOPORT_PORT_00_PIN_05	IO port 0 pin 5.
IOPORT_PORT_00_PIN_06	IO port 0 pin 6.
IOPORT_PORT_00_PIN_07	IO port 0 pin 7.
IOPORT_PORT_00_PIN_08	IO port 0 pin 8.
IOPORT_PORT_00_PIN_09	IO port 0 pin 9.
IOPORT_PORT_00_PIN_10	IO port 0 pin 10.
IOPORT_PORT_00_PIN_11	IO port 0 pin 11.
IOPORT_PORT_00_PIN_12	IO port 0 pin 12.
IOPORT_PORT_00_PIN_13	IO port 0 pin 13.
IOPORT_PORT_00_PIN_14	IO port 0 pin 14.
IOPORT_PORT_00_PIN_15	IO port 0 pin 15.
IOPORT_PORT_01_PIN_00	IO port 1 pin 0.

Name	Description
IOPORT_PORT_01_PIN_01	IO port 1 pin 1.
IOPORT_PORT_01_PIN_02	IO port 1 pin 2.
IOPORT_PORT_01_PIN_03	IO port 1 pin 3.
IOPORT_PORT_01_PIN_04	IO port 1 pin 4.
IOPORT_PORT_01_PIN_05	IO port 1 pin 5.
IOPORT_PORT_01_PIN_06	IO port 1 pin 6.
IOPORT_PORT_01_PIN_07	IO port 1 pin 7.
IOPORT_PORT_01_PIN_08	IO port 1 pin 8.
IOPORT_PORT_01_PIN_09	IO port 1 pin 9.
IOPORT_PORT_01_PIN_10	IO port 1 pin 10.
IOPORT_PORT_01_PIN_11	IO port 1 pin 11.
IOPORT_PORT_01_PIN_12	IO port 1 pin 12.
IOPORT_PORT_01_PIN_13	IO port 1 pin 13.
IOPORT_PORT_01_PIN_14	IO port 1 pin 14.
IOPORT_PORT_01_PIN_15	IO port 1 pin 15.
IOPORT_PORT_02_PIN_00	IO port 2 pin 0.
IOPORT_PORT_02_PIN_01	IO port 2 pin 1.
IOPORT_PORT_02_PIN_02	IO port 2 pin 2.
IOPORT_PORT_02_PIN_03	IO port 2 pin 3.
IOPORT_PORT_02_PIN_04	IO port 2 pin 4.
IOPORT_PORT_02_PIN_05	IO port 2 pin 5.
IOPORT_PORT_02_PIN_06	IO port 2 pin 6.
IOPORT_PORT_02_PIN_07	IO port 2 pin 7.
IOPORT_PORT_02_PIN_08	IO port 2 pin 8.

Name	Description
IOPORT_PORT_02_PIN_09	IO port 2 pin 9.
IOPORT_PORT_02_PIN_10	IO port 2 pin 10.
IOPORT_PORT_02_PIN_11	IO port 2 pin 11.
IOPORT_PORT_02_PIN_12	IO port 2 pin 12.
IOPORT_PORT_02_PIN_13	IO port 2 pin 13.
IOPORT_PORT_02_PIN_14	IO port 2 pin 14.
IOPORT_PORT_02_PIN_15	IO port 2 pin 15.
IOPORT_PORT_03_PIN_00	IO port 3 pin 0.
IOPORT_PORT_03_PIN_01	IO port 3 pin 1.
IOPORT_PORT_03_PIN_02	IO port 3 pin 2.
IOPORT_PORT_03_PIN_03	IO port 3 pin 3.
IOPORT_PORT_03_PIN_04	IO port 3 pin 4.
IOPORT_PORT_03_PIN_05	IO port 3 pin 5.
IOPORT_PORT_03_PIN_06	IO port 3 pin 6.
IOPORT_PORT_03_PIN_07	IO port 3 pin 7.
IOPORT_PORT_03_PIN_08	IO port 3 pin 8.
IOPORT_PORT_03_PIN_09	IO port 3 pin 9.
IOPORT_PORT_03_PIN_10	IO port 3 pin 10.
IOPORT_PORT_03_PIN_11	IO port 3 pin 11.
IOPORT_PORT_03_PIN_12	IO port 3 pin 12.
IOPORT_PORT_03_PIN_13	IO port 3 pin 13.
IOPORT_PORT_03_PIN_14	IO port 3 pin 14.
IOPORT_PORT_03_PIN_15	IO port 3 pin 15.
IOPORT_PORT_04_PIN_00	IO port 4 pin 0.

Name	Description
IOPORT_PORT_04_PIN_01	IO port 4 pin 1.
IOPORT_PORT_04_PIN_02	IO port 4 pin 2.
IOPORT_PORT_04_PIN_03	IO port 4 pin 3.
IOPORT_PORT_04_PIN_04	IO port 4 pin 4.
IOPORT_PORT_04_PIN_05	IO port 4 pin 5.
IOPORT_PORT_04_PIN_06	IO port 4 pin 6.
IOPORT_PORT_04_PIN_07	IO port 4 pin 7.
IOPORT_PORT_04_PIN_08	IO port 4 pin 8.
IOPORT_PORT_04_PIN_09	IO port 4 pin 9.
IOPORT_PORT_04_PIN_10	IO port 4 pin 10.
IOPORT_PORT_04_PIN_11	IO port 4 pin 11.
IOPORT_PORT_04_PIN_12	IO port 4 pin 12.
IOPORT_PORT_04_PIN_13	IO port 4 pin 13.
IOPORT_PORT_04_PIN_14	IO port 4 pin 14.
IOPORT_PORT_04_PIN_15	IO port 4 pin 15.
IOPORT_PORT_05_PIN_00	IO port 5 pin 0.
IOPORT_PORT_05_PIN_01	IO port 5 pin 1.
IOPORT_PORT_05_PIN_02	IO port 5 pin 2.
IOPORT_PORT_05_PIN_03	IO port 5 pin 3.
IOPORT_PORT_05_PIN_04	IO port 5 pin 4.
IOPORT_PORT_05_PIN_05	IO port 5 pin 5.
IOPORT_PORT_05_PIN_06	IO port 5 pin 6.
IOPORT_PORT_05_PIN_07	IO port 5 pin 7.
IOPORT_PORT_05_PIN_08	IO port 5 pin 8.

Name	Description
IOPORT_PORT_05_PIN_09	IO port 5 pin 9.
IOPORT_PORT_05_PIN_10	IO port 5 pin 10.
IOPORT_PORT_05_PIN_11	IO port 5 pin 11.
IOPORT_PORT_05_PIN_12	IO port 5 pin 12.
IOPORT_PORT_05_PIN_13	IO port 5 pin 13.
IOPORT_PORT_05_PIN_14	IO port 5 pin 14.
IOPORT_PORT_05_PIN_15	IO port 5 pin 15.
IOPORT_PORT_06_PIN_00	IO port 6 pin 0.
IOPORT_PORT_06_PIN_01	IO port 6 pin 1.
IOPORT_PORT_06_PIN_02	IO port 6 pin 2.
IOPORT_PORT_06_PIN_03	IO port 6 pin 3.
IOPORT_PORT_06_PIN_04	IO port 6 pin 4.
IOPORT_PORT_06_PIN_05	IO port 6 pin 5.
IOPORT_PORT_06_PIN_06	IO port 6 pin 6.
IOPORT_PORT_06_PIN_07	IO port 6 pin 7.
IOPORT_PORT_06_PIN_08	IO port 6 pin 8.
IOPORT_PORT_06_PIN_09	IO port 6 pin 9.
IOPORT_PORT_06_PIN_10	IO port 6 pin 10.
IOPORT_PORT_06_PIN_11	IO port 6 pin 11.
IOPORT_PORT_06_PIN_12	IO port 6 pin 12.
IOPORT_PORT_06_PIN_13	IO port 6 pin 13.
IOPORT_PORT_06_PIN_14	IO port 6 pin 14.
IOPORT_PORT_06_PIN_15	IO port 6 pin 15.
IOPORT_PORT_07_PIN_00	IO port 7 pin 0.

Name	Description
IOPORT_PORT_07_PIN_01	IO port 7 pin 1.
IOPORT_PORT_07_PIN_02	IO port 7 pin 2.
IOPORT_PORT_07_PIN_03	IO port 7 pin 3.
IOPORT_PORT_07_PIN_04	IO port 7 pin 4.
IOPORT_PORT_07_PIN_05	IO port 7 pin 5.
IOPORT_PORT_07_PIN_06	IO port 7 pin 6.
IOPORT_PORT_07_PIN_07	IO port 7 pin 7.
IOPORT_PORT_07_PIN_08	IO port 7 pin 8.
IOPORT_PORT_07_PIN_09	IO port 7 pin 9.
IOPORT_PORT_07_PIN_10	IO port 7 pin 10.
IOPORT_PORT_07_PIN_11	IO port 7 pin 11.
IOPORT_PORT_07_PIN_12	IO port 7 pin 12.
IOPORT_PORT_07_PIN_13	IO port 7 pin 13.
IOPORT_PORT_07_PIN_14	IO port 7 pin 14.
IOPORT_PORT_07_PIN_15	IO port 7 pin 15.
IOPORT_PORT_08_PIN_00	IO port 8 pin 0.
IOPORT_PORT_08_PIN_01	IO port 8 pin 1.
IOPORT_PORT_08_PIN_02	IO port 8 pin 2.
IOPORT_PORT_08_PIN_03	IO port 8 pin 3.
IOPORT_PORT_08_PIN_04	IO port 8 pin 4.
IOPORT_PORT_08_PIN_05	IO port 8 pin 5.
IOPORT_PORT_08_PIN_06	IO port 8 pin 6.
IOPORT_PORT_08_PIN_07	IO port 8 pin 7.
IOPORT_PORT_08_PIN_08	IO port 8 pin 8.

Name	Description
IOPORT_PORT_08_PIN_09	IO port 8 pin 9.
IOPORT_PORT_08_PIN_10	IO port 8 pin 10.
IOPORT_PORT_08_PIN_11	IO port 8 pin 11.
IOPORT_PORT_08_PIN_12	IO port 8 pin 12.
IOPORT_PORT_08_PIN_13	IO port 8 pin 13.
IOPORT_PORT_08_PIN_14	IO port 8 pin 14.
IOPORT_PORT_08_PIN_15	IO port 8 pin 15.
IOPORT_PORT_09_PIN_00	IO port 9 pin 0.
IOPORT_PORT_09_PIN_01	IO port 9 pin 1.
IOPORT_PORT_09_PIN_02	IO port 9 pin 2.
IOPORT_PORT_09_PIN_03	IO port 9 pin 3.
IOPORT_PORT_09_PIN_04	IO port 9 pin 4.
IOPORT_PORT_09_PIN_05	IO port 9 pin 5.
IOPORT_PORT_09_PIN_06	IO port 9 pin 6.
IOPORT_PORT_09_PIN_07	IO port 9 pin 7.
IOPORT_PORT_09_PIN_08	IO port 9 pin 8.
IOPORT_PORT_09_PIN_09	IO port 9 pin 9.
IOPORT_PORT_09_PIN_10	IO port 9 pin 10.
IOPORT_PORT_09_PIN_11	IO port 9 pin 11.
IOPORT_PORT_09_PIN_12	IO port 9 pin 12.
IOPORT_PORT_09_PIN_13	IO port 9 pin 13.
IOPORT_PORT_09_PIN_14	IO port 9 pin 14.
IOPORT_PORT_09_PIN_15	IO port 9 pin 15.
IOPORT_PORT_10_PIN_00	IO port 10 pin 0.

Name	Description
IOPORT_PORT_10_PIN_01	IO port 10 pin 1.
IOPORT_PORT_10_PIN_02	IO port 10 pin 2.
IOPORT_PORT_10_PIN_03	IO port 10 pin 3.
IOPORT_PORT_10_PIN_04	IO port 10 pin 4.
IOPORT_PORT_10_PIN_05	IO port 10 pin 5.
IOPORT_PORT_10_PIN_06	IO port 10 pin 6.
IOPORT_PORT_10_PIN_07	IO port 10 pin 7.
IOPORT_PORT_10_PIN_08	IO port 10 pin 8.
IOPORT_PORT_10_PIN_09	IO port 10 pin 9.
IOPORT_PORT_10_PIN_10	IO port 10 pin 10.
IOPORT_PORT_10_PIN_11	IO port 10 pin 11.
IOPORT_PORT_10_PIN_12	IO port 10 pin 12.
IOPORT_PORT_10_PIN_13	IO port 10 pin 13.
IOPORT_PORT_10_PIN_14	IO port 10 pin 14.
IOPORT_PORT_10_PIN_15	IO port 10 pin 15.
IOPORT_PORT_11_PIN_00	IO port 11 pin 0.
IOPORT_PORT_11_PIN_01	IO port 11 pin 1.
IOPORT_PORT_11_PIN_02	IO port 11 pin 2.
IOPORT_PORT_11_PIN_03	IO port 11 pin 3.
IOPORT_PORT_11_PIN_04	IO port 11 pin 4.
IOPORT_PORT_11_PIN_05	IO port 11 pin 5.
IOPORT_PORT_11_PIN_06	IO port 11 pin 6.
IOPORT_PORT_11_PIN_07	IO port 11 pin 7.
IOPORT_PORT_11_PIN_08	IO port 11 pin 8.

Name	Description
IOPORT_PORT_11_PIN_09	IO port 11 pin 9.
IOPORT_PORT_11_PIN_10	IO port 11 pin 10.
IOPORT_PORT_11_PIN_11	IO port 11 pin 11.
IOPORT_PORT_11_PIN_12	IO port 11 pin 12.
IOPORT_PORT_11_PIN_13	IO port 11 pin 13.
IOPORT_PORT_11_PIN_14	IO port 11 pin 14.
IOPORT_PORT_11_PIN_15	IO port 11 pin 15.

8.21.6.5 ioport_peripheral_t

`ioport_peripheral_t`

Detailed description

Superset of all peripheral functions.

Enumerated values

Name	Description
IOPORT_PERIPHERAL_IO	Pin will functions as an IO pin.
IOPORT_PERIPHERAL_DEBUG	Pin will function as a DEBUG pin.
IOPORT_PERIPHERAL_AGT	Pin will function as an AGT.
IOPORT_PERIPHERAL_GPT0	Pin will function as a GPT.
IOPORT_PERIPHERAL_GPT1	Pin will function as a GPT.
IOPORT_PERIPHERAL_SCI0_2_4_6_8	Pin will function as an SCI.
IOPORT_PERIPHERAL_SCI1_3_5_7_9	Pin will function as an SCI.
IOPORT_PERIPHERAL_RSPI	Pin will function as a RSPI.
IOPORT_PERIPHERAL_RIIC	Pin will function as a RIIC.
IOPORT_PERIPHERAL_KEY	Pin will function as a KEY.
IOPORT_PERIPHERAL_CLKOUT_COMP_RTC	Pin will function as a.

Name	Description
IOPORT_PERIPHERAL_CAC_AD	Pin will function as a CAC/ADC.
IOPORT_PERIPHERAL_BUS	Pin will function as a BUS.
IOPORT_PERIPHERAL_CTSU	Pin will function as a CTSU.
IOPORT_PERIPHERAL_LCDC	Pin will function as a segment LCD.
IOPORT_PERIPHERAL_DALI	Pin will function as a DALI.
IOPORT_PERIPHERAL_CAN	Pin will function as a CAN.
IOPORT_PERIPHERAL_QSPI	Pin will function as a QSPI.
IOPORT_PERIPHERAL_SSI	Pin will function as an SSI.
IOPORT_PERIPHERAL_USB_FS	Pin will function as a USB.
IOPORT_PERIPHERAL_USB_HS	Pin will function as a USB.
IOPORT_PERIPHERAL_SDHI_MMC	Pin will function as an SD/MMC.
IOPORT_PERIPHERAL_ETHER_MII	Pin will function as an Ethernet.
IOPORT_PERIPHERAL_ETHER_RMII	Pin will function as an Ethernet.
IOPORT_PERIPHERAL_PDC	Pin will function as a PDC.
IOPORT_PERIPHERAL_LCD_GRAPHICS	Pin will function as a graphics.
IOPORT_PERIPHERAL_TRACE	Pin will function as a debug trace.
IOPORT_PERIPHERAL_END	Marks end of enum - used by.

8.21.6.6 ioport_ethernet_channel_t

`ioport_ethernet_channel_t`

Detailed description

Superset of Ethernet channels.

Enumerated values

Name	Description
IOPORT_ETHERNET_CHANNEL_0	Used to select Ethernet channel 0.

Name	Description
IOPORT_ETHERNET_CHANNEL_1	Used to select Ethernet channel 1.
IOPORT_ETHERNET_CHANNEL_END	Marks end of enum - used by parameter checking.

8.21.6.7 ioport_ethernet_mode_t

`ioport_ethernet_mode_t`

Detailed description

Superset of Ethernet PHY modes.

Enumerated values

Name	Description
IOPORT_ETHERNET_MODE_MII	Ethernet PHY mode set to MII.
IOPORT_ETHERNET_MODE_RMII	Ethernet PHY mode set to RMII.
IOPORT_ETHERNET_MODE_END	Marks end of enum - used by parameter checking.

8.21.6.8 ioport_cfg_options_t

`ioport_cfg_options_t`

Detailed description

Options to configure pin functions

Enumerated values

Name	Description
IOPORT_CFG_PORT_DIRECTION_INPUT	Sets the pin direction to input (default)
IOPORT_CFG_PORT_DIRECTION_OUTPUT	Sets the pin direction to output.
IOPORT_CFG_PORT_OUTPUT_LOW	Sets the pin level to low.
IOPORT_CFG_PORT_OUTPUT_HIGH	Sets the pin level to high.
IOPORT_CFG_PULLUP_ENABLE	Enables the pin's internal pull-up.
IOPORT_CFG_PIM_TTL	Enables the pin's input mode.
IOPORT_CFG_NMOS_ENABLE	Enables the pin's NMOS open-drain output.

Name	Description
IOPORT_CFG_PMOS_ENABLE	Enables the pin's PMOS open-drain ouput.
IOPORT_CFG_DRIVE_MID	Sets pin drive output to medium.
IOPORT_CFG_DRIVE_MID_IIC	Sets pin to drive output needed for IIC on a 20mA port.
IOPORT_CFG_DRIVE_HIGH	Sets pin drive output to high.
IOPORT_CFG_EVENT_RISING_EDGE	Sets pin event trigger to rising edge.
IOPORT_CFG_EVENT_FALLING_EDGE	Sets pin event trigger to falling edge.
IOPORT_CFG_EVENT_BOTH_EDGES	Sets pin event trigger to both edges.
IOPORT_CFG_IRQ_ENABLE	Sets pin as an IRQ pin.
IOPORT_CFG_ANALOG_ENABLE	Enables pin to operate as an analog pin.
IOPORT_CFG_PERIPHERAL_PIN	Enables pin to operate as a peripheral pin.

8.21.6.9 ioport_size_t

```
typedef uint16_t ioport_size_t
```

Brief description

IO port size on this device.

Detailed description

IO port type used with ports

8.21.7 API Structures

8.21.7.1 ioport_pin_cfg_t

ioport_pin_cfg_t

Detailed description

Pin identifier and pin PFS pin configuration value

Variables

- uint32_t pin_cfg

  Pin PFS configuration - Use ioport_cfg_options_t parameters to configure.

- ioport_port_pin_t pin

  Pin identifier.

8.21.7.2 ioport_cfg_t

ioport_cfg_t

Detailed description

Multiple pin configuration data for loading into PFS registers by R_IOPORT_Init

Variables

- uint16_t number_of_pins

  Number of pins for which there is configuration data.

- ioport_pin_cfg_t const * p_pin_cfg_data

  Pin configuration data.

8.21.7.3 ioport_api_t

ioport_api_t

Detailed description

IOPort driver structure. IOPort functions implemented at the HAL layer will follow this API.

8.21.7.4 init

```
ssp_err_t(* ioport_api_t::init) (const ioport_cfg_t *p_cfg)
```

Detailed description

Initialize internal driver data and initial pin configurations. Called during startup. Do not call this API during runtime. Use pinsCfg for runtime reconfiguration of multiple pins. Implemented as

- R_IOPORT_Init

Table 1173:Parameters

Name	Direction	Description
p_cfg	in	Pointer to pin configuration data array.

Parameter p_cfg

Definition: `ioport_cfg_t ioport_cfg_t *p_cfg`

Multiple pin configuration data for loading into PFS registers by R_IOPORT_Init

- `ioport_cfg_t`::number_of_pins

  Number of pins for which there is configuration data.

- `ioport_cfg_t`::`ioport_pin_cfg_t`

  Pin configuration data.

8.21.7.5 pinsCfg

```
ssp_err_t(* ioport_api_t::pinsCfg) (const ioport_cfg_t *p_cfg)
```

Detailed description

Configure multiple pins. Implemented as

- R_IOPORT_PinsCfg

Table 1174:Parameters

Name	Direction	Description
p_cfg	in	Pointer to pin configuration data array.

Parameter p_cfg

Definition: `ioport_cfg_t ioport_cfg_t *p_cfg`

Multiple pin configuration data for loading into PFS registers by R_IOPORT_Init

- `ioport_cfg_t`::`number_of_pins`

  Number of pins for which there is configuration data.

- `ioport_cfg_t`::`ioport_pin_cfg_t`

  Pin configuration data.

8.21.7.6 pinCfg

```
ssp_err_t(* ioport_api_t::pinCfg) (ioport_port_pin_t pin, uint32_t cfg)
```

Detailed description

Configure settings for an individual pin. Implemented as

- R_IOPORT_PinCfg

Table 1175:Parameters

Name	Direction	Description
pin	in	Pin to be read.
cfg	in	Configuration options for the pin.

Parameter pin
Parameter cfg

`uint32_t`

8.21.7.7 pinDirectionSet

```
ssp_err_t(* ioport_api_t::pinDirectionSet) (ioport_port_pin_t pin,
ioport_direction_t direction)
```

Detailed description

Set the pin direction of a pin. Implemented as

- R_IOPORT_PinDirectionSet

Table 1176:Parameters

Name	Direction	Description
pin	in	Pin being configured.
direction	in	Direction to set pin to which is a member of ioport_direction_t.

Parameter pin
Parameter direction

8.21.7.8 pinEventInputRead

```
ssp_err_t(* ioport_api_t::pinEventInputRead) (ioport_port_pin_t pin,
ioport_level_t *p_pin_event)
```

Detailed description

Read the event input data of the specified pin and return the level. Implemented as

- R_IOPORT_PinEventInputRead

Table 1177:Parameters

Name	Direction	Description
pin	in	Pin to be read.
p_pin_event	in	Pointer to return the event data.

Parameter pin
Parameter p_pin_event

Definition: `ioport_level_t*p_pin_event`

Levels that can be set and read for individual pins

8.21.7.9 pinEventOutputWrite

```
ssp_err_t(* ioport_api_t::pinEventOutputWrite) (ioport_port_pin_t pin,
ioport_level_t pin_value)
```

Detailed description

Write pin event data. Implemented as

- R_IOPORT_PinEventOutputWrite

Table 1178:Parameters

Name	Direction	Description
pin	in	Pin event data is to be written to.
pin_value	in	Level to be written to pin output event.

Parameter pin
Parameter pin_value

Definition: `ioport_level_tpin_value`

Levels that can be set and read for individual pins

8.21.7.10 pinEthernetModeCfg

```
ssp_err_t(* ioport_api_t::pinEthernetModeCfg) (ioport_ethernet_channel_t channel,
ioport_ethernet_mode_t mode)
```

Detailed description

Configure the PHY mode of the Ethernet channels. Implemented as

- R_IOPORT_EthernetModeCfg

Table 1179:Parameters

Name	Direction	Description
channel	in	Channel configuration will be set for.
mode	in	PHY mode to set the channel to.

Parameter channel
Parameter mode

8.21.7.11 pinRead

```
ssp_err_t(* ioport_api_t::pinRead) (ioport_port_pin_t pin, ioport_level_t
*p_pin_value)
```

Detailed description

Read level of a pin. Implemented as

- R_IOPORT_PinRead

Table 1180:Parameters

Name	Direction	Description
pin	in	Pin to be read.
p_pin_value	in	Pointer to return the pin level.

Parameter pin
Parameter p_pin_value

Definition: `ioport_level_t*p_pin_value`

Levels that can be set and read for individual pins

8.21.7.12 pinWrite

```
ssp_err_t(* ioport_api_t::pinWrite) (ioport_port_pin_t pin, ioport_level_t level)
```

Detailed description

Write specified level to a pin. Implemented as

- R_IOPORT_PinWrite

Table 1181:Parameters

Name	Direction	Description
pin	in	Pin to be written to.
level	in	State to be written to the pin.

Parameter pin
Parameter level

8.21.7.13 portDirectionSet

```
ssp_err_t(* ioport_api_t::portDirectionSet) (ioport_port_t port, ioport_size_t
direction_values, ioport_size_t mask)
```

Detailed description

Set the direction of one or more pins on a port. Implemented as

- R_IOPORT_PortDirectionSet

Table 1182:Parameters

Name	Direction	Description
port	in	Port being configured.
direction_values	in	Value controlling direction of pins on port (1 - output, 0 - input).
mask	in	Mask controlling which pins on the port are to be configured.

Parameter port
Parameter direction_values

Definition: `ioport_size_tdirection_values`

IO port type used with ports

Parameter mask

Definition: `ioport_size_tmask`

IO port type used with ports

8.21.7.14 portEventInputRead

```
ssp_err_t(* ioport_api_t::portEventInputRead) (ioport_port_t port, ioport_size_t
*p_event_data)
```

Detailed description

Read captured event data for a port. Implemented as

- R_IOPORT_PortEventInputRead

Table 1183:Parameters

Name	Direction	Description
port	in	Port to be read.
p_event_data	in	Pointer to return the event data.

Parameter port
Parameter p_event_data

Definition: `ioport_size_t*p_event_data`

IO port type used with ports

8.21.7.15 portEventOutputWrite

```
ssp_err_t(* ioport_api_t::portEventOutputWrite) (ioport_port_t port,
ioport_size_t event_data, ioport_size_t mask_value)
```

Detailed description

Write event output data for a port. Implemented as

- R_IOPORT_PortEventOutputWrite

Table 1184:Parameters

Name	Direction	Description
port	in	Port event data will be written to.
event_data	in	Data to be written as event data to specified port.
mask_value	in	Each bit set to 1 in the mask corresponds to that bit's value in event data. being written to port.

Parameter port
Parameter event_data

Definition: ioport_size_tevent_data

IO port type used with ports

Parameter mask_value

Definition: ioport_size_tmask_value

IO port type used with ports

8.21.7.16 portRead

```
ssp_err_t(* ioport_api_t::portRead) (ioport_port_t port, ioport_size_t
*p_port_value)
```

Detailed description

Read states of pins on the specified port. Implemented as

- R_IOPORT_PortRead

Table 1185:Parameters

Name	Direction	Description
port	in	Port to be read.

Table 1185:Parameters (Continued)

Name	Direction	Description
p_port_value	in	Pointer to return the port value.

Parameter port
Parameter p_port_value

Definition: `ioport_size_t*p_port_value`

IO port type used with ports

8.21.7.17 portWrite

```
ssp_err_t(* ioport_api_t::portWrite) (ioport_port_t port, ioport_size_t value,
ioport_size_t mask)
```

Detailed description

Write to multiple pins on a port. Implemented as

- R_IOPORT_PortWrite

Table 1186:Parameters

Name	Direction	Description
port	in	Port to be written to.
value	in	Value to be written to the port.
mask	in	Mask controlling which pins on the port are written to.

Parameter port
Parameter value

Definition: `ioport_size_tvalue`

IO port type used with ports

Parameter mask

Definition: `ioport_size_tmask`

IO port type used with ports

8.21.7.18 versionGet

```
ssp_err_t(* ioport_api_t::versionGet) (ssp_version_t *p_data)
```

Detailed description

Return the version of the IOPort driver. Implemented as

- R_IOPORT_VersionGet

Table 1187:Parameters

Name	Direction	Description
p_data	out	Memory address to return version information to.

Parameter p_data

Definition: `ssp_version_t *p_data`

- `ssp_version_t::version_id`
  Version id
- `ssp_version_t::code_version_minor`
  Code minor version.
- `ssp_version_t::code_version_major`
  Code major version.
- `ssp_version_t::api_version_minor`
  API minor version.
- `ssp_version_t::api_version_major`
  API major version.
- `ssp_version_tstruct{}`
  Code version parameters

8.21.7.19 ioport_instance_t

ioport_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- ioport_cfg_t const * p_cfg
  Pointer to the configuration structure for this instance.
- ioport_api_t const * p_api
  Pointer to the API structure for this instance.

8.22 Input Capture Interface

Interface for sampling input signals for pulse width.

8.22.1 Summary

The input capture interface provides for sampling of input signals to determine the width of a pulse (from one edge to the opposite edge). An interrupt can be triggered after each measurement is captured.

Implemented by: GPT Input Capture

See also: Timer Interface

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

8.22.2 Interface API

input_capture_api_t

Function name	Description
.open	Initial configuration.
.disable	Disables input capture measurement.
.enable	Enables input capture measurement.
.infoGet	Gets the status (running or not) of the measurement counter.
.lastCaptureGet	Gets the last captured timer/counter value
.close	Close the input capture operation. Allows driver to be reconfigured, and may reduce power consumption.
.versionGet	Gets the version of this API and stores it in structure pointed to by p_version.

8.22.3 Data structures

- input_capture_callback_args_t
- input_capture_capture_t

- input_capture_info_t
- input_capture_cfg_t
- input_capture_instance_t

8.22.4 Enumerations

- input_capture_mode_t
- input_capture_signal_edge_t
- input_capture_signal_level_t
- input_capture_repetition_t
- input_capture_event_t
- input_capture_status_t
- input_capture_variant_t

8.22.5 Typedefs

- input_capture_ctrl_t

8.22.6 Defines

- #define INPUT_CAPTURE_API_VERSION_MAJOR

  Initial value: `(1U)`
- #define INPUT_CAPTURE_API_VERSION_MINOR

  Initial value: `(4U)`
- #define info_capture_info_t

  Initial value:`input_capture_info_t`

  Mapping of deprecated info_capture_info_t.

8.22.7 API Data

8.22.7.1 input_capture_mode_t

`input_capture_mode_t`

Detailed description

Input capture operational modes

Enumerated values

Name	Description
INPUT_CAPTURE_MODE_PULSE_WIDTH	Measure a signal pulse width.
INPUT_CAPTURE_MODE_PERIOD	Measure a signal Cycle period.

8.22.7.2 input_capture_signal_edge_t

`input_capture_signal_edge_t`

Detailed description

Input capture signal edge trigger

Enumerated values

Name	Description
INPUT_CAPTURE_SIGNAL_EDGE_RISING	The capture begins at the rising edge.
INPUT_CAPTURE_SIGNAL_EDGE_FALLING	The capture begins at the falling edge.

8.22.7.3 input_capture_signal_level_t

`input_capture_signal_level_t`

Detailed description

Input capture signal level, primarily used for the enable signal

Enumerated values

Name	Description
INPUT_CAPTURE_SIGNAL_LEVEL_NONE	Use this if signal_level is not applicable to a particular measurement.
INPUT_CAPTURE_SIGNAL_LEVEL_LOW	The capture is enabled at the low level.
INPUT_CAPTURE_SIGNAL_LEVEL_HIGH	The capture is enabled at the high level.

8.22.7.4 input_capture_repetition_t

`input_capture_repetition_t`

Detailed description

Specifies either a one-time or continuous measurements

Enumerated values

Name	Description
INPUT_CAPTURE_REPETITION_PERIODIC	Capture continuous measurements, until explicitly stopped or disabled.
INPUT_CAPTURE_REPETITION_ONE_SHOT	Capture a single measurement, then interrupts are disabled.

8.22.7.5 input_capture_event_t

`input_capture_event_t`

Detailed description

Events that can trigger a callback function

Enumerated values

Name	Description
INPUT_CAPTURE_EVENT_MEASUREMENT	A capture measurement has been captured.
INPUT_CAPTURE_EVENT_OVERFLOW	A capture measurement overflowed the counter.

8.22.7.6 input_capture_status_t

`input_capture_status_t`

Detailed description

Input capture status.

Enumerated values

Name	Description
INPUT_CAPTURE_STATUS_IDLE	The input capture timer is idle.
INPUT_CAPTURE_STATUS_CAPTURING	A capture measurement is in progress.

8.22.7.7 input_capture_variant_t

`input_capture_variant_t`

Detailed description

Input capture timer variant types.

Enumerated values

Name	Description
INPUT_CAPTURE_VARIANT_32_BIT	32-bit timer
INPUT_CAPTURE_VARIANT_16_BIT	16-bit timer

8.22.7.8 input_capture_ctrl_t

```
typedef void input_capture_ctrl_t
```

Detailed description

Input capture control block. Allocate an instance specific control block to pass into the input capture API calls. Implemented as

- gpt_input_capture_instance_ctrl_t

8.22.8 API Structures

8.22.8.1 input_capture_callback_args_t

input_capture_callback_args_t

Detailed description

Callback function parameter data

Variables

- uint8_t channel

  The channel being used.
- input_capture_event_t event

  The event that caused the interrupt and callback.
- uint32_t counter

  The value of the timer captured at the time of interrupt.
- uint32_t overflows

  The number of counter overflows that occurred during this measurement.
- void const * p_context

  Placeholder for user data, set in input_capture_cfg_t::p_context.

8.22.8.2 input_capture_capture_t

input_capture_capture_t

Detailed description

Capture data

Variables

- uint32_t counter

  The value of the timer captured at the time of interrupt.

- uint32_t overflows

  The number of counter overflows that occurred during this measurement.

8.22.8.3 input_capture_info_t

input_capture_info_t

Detailed description

Driver information

Variables

- input_capture_status_t status

  Whether or not a capture is in progress.

- input_capture_variant_t variant

  Whether timer is 16 or 32 bits.

8.22.8.4 input_capture_cfg_t

input_capture_cfg_t

Detailed description

User configuration structure, passed to open function

Variables

- uint8_t channel

  The channel in use.

- uint8_t capture_irq_ipl

  Capture interrupt priority.

- uint8_t overflow_irq_ipl

  Overflow interrupt priority.

- input_capture_mode_t mode

  The mode of measurement to be performed.

- input_capture_signal_edge_t edge

  The triggering edge to start a measurement (rise or fall).

- input_capture_repetition_t repetition

  One-shot or periodic measurement.
- bool autostart

  Specifies whether interrupts are enabled or not after open.
- void const * p_extend

  REQUIRED. Pointer to peripheral-specific extension parameters. See gpt_input_capture_extend_t for GPT.
- void(* p_callback)(input_capture_callback_args_t *p_args)

  Pointer to user's callback function, or NULL if no interrupt desired.
- void const * p_context

  Pointer to user's context data, to be passed to the callback.

8.22.8.5 input_capture_api_t

input_capture_api_t

Detailed description

Input capture API structure. Functions implemented at the HAL layer will implement this API.

8.22.8.6 open

```
ssp_err_t(* input_capture_api_t::open) (input_capture_ctrl_t *const p_ctrl,
input_capture_cfg_t const *const p_cfg)
```

Detailed description

Initial configuration. Implemented as

- R_GPT_InputCaptureOpen

NOTE: To reconfigure after calling this function, call close first.

Table 1188:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block: memory allocated by caller, contents filled in by open.
p_cfg	in	Pointer to configuration structure. All elements of this structure must be set by user.

Parameter p_ctrl

Definition: input_capture_ctrl_t*const p_ctrl

Input capture control block. Allocate an instance specific control block to pass into the input capture API calls. Implemented asgpt_input_capture_instance_ctrl_t

Parameter p_cfg

Definition: input_capture_cfg_t const *const p_cfg

User configuration structure, passed to open function

- input_capture_cfg_t::channel

  The channel in use.

- input_capture_cfg_t::capture_irq_ipl

  Capture interrupt priority.

- input_capture_cfg_t::overflow_irq_ipl

  Overflow interrupt priority.

- input_capture_cfg_t::input_capture_mode_t

  The mode of measurement to be performed.

  Enumerated as:

  – INPUT_CAPTURE_MODE_PULSE_WIDTH

  – INPUT_CAPTURE_MODE_PERIOD

- input_capture_cfg_t::input_capture_signal_edge_t

  The triggering edge to start a measurement (rise or fall).

  Enumerated as:

  – INPUT_CAPTURE_SIGNAL_EDGE_RISING

  – INPUT_CAPTURE_SIGNAL_EDGE_FALLING

- input_capture_cfg_t::input_capture_repetition_t

  One-shot or periodic measurement.

  Enumerated as:

  – INPUT_CAPTURE_REPETITION_PERIODIC

  – INPUT_CAPTURE_REPETITION_ONE_SHOT

- input_capture_cfg_t::autostart

  Specifies whether interrupts are enabled or not after open.

- input_capture_cfg_t::p_extend

  REQUIRED. Pointer to peripheral-specific extension parameters. See gpt_input_capture_extend_t for GPT.

- input_capture_cfg_t::p_callback

  Pointer to user's callback function, or NULL if no interrupt desired.

- input_capture_cfg_t::p_context

  Pointer to user's context data, to be passed to the callback.

8.22.8.7 disable

```
ssp_err_t(* input_capture_api_t::disable) (input_capture_ctrl_t const *const
p_ctrl)
```

Detailed description

Disables input capture measurement. Implemented as

- R_GPT_InputCaptureDisable

Table 1189:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block initialized by open call.

Parameter p_ctrl

Definition: input_capture_ctrl_tconst *const p_ctrl

Input capture control block. Allocate an instance specific control block to pass into the input capture API calls. Implemented asgpt_input_capture_instance_ctrl_t

8.22.8.8 enable

```
ssp_err_t(* input_capture_api_t::enable) (input_capture_ctrl_t const *const
p_ctrl)
```

Detailed description

Enables input capture measurement. Implemented as

- R_GPT_InputCaptureEnable

Table 1190:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block initialized by open call.

NOTE: Interrupts may already be enabled if specified by input_capture_cfg_t::irq_enable.

Parameter p_ctrl

Definition: input_capture_ctrl_tconst *const p_ctrl

Input capture control block. Allocate an instance specific control block to pass into the input capture API calls. Implemented asgpt_input_capture_instance_ctrl_t

8.22.8.9 infoGet

```
ssp_err_t(* input_capture_api_t::infoGet) (input_capture_ctrl_t const *const
p_ctrl, input_capture_info_t *const p_info)
```

Detailed description

Gets the status (running or not) of the measurement counter. Implemented as

- R_GPT_InputCaptureInfoGet

Table 1191:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block initialized by open call.
p_info	out	Pointer to returned status. Result will be one of input_capture_status_t.

Parameter p_ctrl

Definition: input_capture_ctrl_tconst *const p_ctrl

Input capture control block. Allocate an instance specific control block to pass into the input capture API calls. Implemented asgpt_input_capture_instance_ctrl_t

Parameter p_info

Definition: input_capture_info_t*const p_info

Driver information

- input_capture_info_t::status

  Whether or not a capture is in progress.

- input_capture_info_t::variant

  Whether timer is 16 or 32 bits.

8.22.8.10 lastCaptureGet

```
ssp_err_t(* input_capture_api_t::lastCaptureGet) (input_capture_ctrl_t const
*const p_ctrl, input_capture_capture_t *const p_counter)
```

Detailed description

Gets the last captured timer/counter value Implemented as

- R_GPT_InputCaptureLastCaptureGet

Table 1192:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block initialized by open call.
p_counter	out	Pointer to location to store last captured counter value.

Parameter p_ctrl

Definition: `input_capture_ctrl_tconst *const p_ctrl`

Input capture control block. Allocate an instance specific control block to pass into the input capture API calls. Implemented asgpt_input_capture_instance_ctrl_t

Parameter p_counter

Definition: `input_capture_capture_t*const p_counter`

Capture data

- `input_capture_capture_t::counter`

  The value of the timer captured at the time of interrupt.

- `input_capture_capture_t::overflows`

  The number of counter overflows that occurred during this measurement.

8.22.8.11 close

```
ssp_err_t(* input_capture_api_t::close) (input_capture_ctrl_t *const p_ctrl)
```

Detailed description

Close the input capture operation. Allows driver to be reconfigured, and may reduce power consumption. Implemented as

- R_GPT_InputCaptureClose

Table 1193:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block initialized by open call.

Parameter p_ctrl

Definition: `input_capture_ctrl_t*const p_ctrl`

Input capture control block. Allocate an instance specific control block to pass into the input capture API calls. Implemented asgpt_input_capture_instance_ctrl_t

8.22.8.12 versionGet

```
ssp_err_t(* input_capture_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Gets the version of this API and stores it in structure pointed to by p_version. Implemented as

- R_GPT_InputCaptureVersionGet

Table 1194:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.22.8.13 input_capture_instance_t

input_capture_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- input_capture_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- input_capture_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- input_capture_api_t const * p_api

  Pointer to the API structure for this instance.

8.23 JPEG Decode Interface

Interface for JPEG decode functions.

8.23.1 Summary

The JPEG DECODE interface provides JPEG decoder functionality. It allows application to convert a JPEG image into bitmap data suitable for display frame buffer.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

JPEG DECODE Interface description: JPEG Decode Driver

8.23.2 Interface API

jpeg_decode_api_t

Function name	Description
.open	Initial configuration
.outputBufferSet	Assign output buffer to JPEG codec for storing output data.
.horizontalStrideSet	Configure the horizontal stride value.
.imageSubsampleSet	Configure the horizontal and vertical subsample settings.
.inputBufferSet	Assign input data buffer to JPEG codec.
.linesDecodedGet	Return the number of lines decoded into the output buffer.
.imageSizeGet	Retrieve image size during decoding operation.
.statusGet	Retrieve current status of the JPEG codec module.
.close	Cancel an outstanding operation.
.versionGet	Get version and store it in provided pointer p_version.
.pixelFormatGet	Get the input pixel format.

8.23.3 Data structures

- jpeg_decode_callback_args_t
- jpeg_decode_cfg_t
- jpeg_decode_instance_t

8.23.4 Enumerations

- jpeg_decode_color_space_t
- jpeg_decode_data_format_t
- jpeg_decode_pixel_format_t
- jpeg_decode_status_t
- jpeg_decode_subsample_t
- jpeg_decode_count_enable_t
- jpeg_decode_resume_mode_t

8.23.5 Typedefs

- jpeg_decode_ctrl_t

8.23.6 Defines

- #define JPEG_DECODE_API_VERSION_MAJOR

  Initial value: `(1U)`

  Register definitions, common services and error codes. Configuration for this module
- #define JPEG_DECODE_API_VERSION_MINOR

  Initial value: `(3U)`

8.23.7 API Data

8.23.7.1 jpeg_decode_color_space_t

`jpeg_decode_color_space_t`

Detailed description

Image color space definitions

Enumerated values

Name	Description
JPEG_DECODE_COLOR_SPACE_YCBCR444	Color Space YCbCr 444.
JPEG_DECODE_COLOR_SPACE_YCBCR422	Color Space YCbCr 422.
JPEG_DECODE_COLOR_SPACE_YCBCR420	Color Space YCbCr 420.

Name	Description
JPEG_DECODE_COLOR_SPACE_YCBCR411	Color Space YCbCr 411.

8.23.7.2 jpeg_decode_data_format_t

`jpeg_decode_data_format_t`

Detailed description

Multi-byte Data Format

Enumerated values

Name	Description
JPEG_DECODE_DATA_FORMAT_NORMAL	(1)(2)(3)(4)(5)(6)(7)(8) Normal byte order
JPEG_DECODE_DATA_FORMAT_BYTE_SWAP	(2)(1)(4)(3)(6)(5)(8)(7) Byte Swap
JPEG_DECODE_DATA_FORMAT_WORD_SWAP	(3)(4)(1)(2)(7)(8)(5)(6) Word Swap
JPEG_DECODE_DATA_FORMAT_WORD_BYTE_SWAP	(4)(3)(2)(1)(8)(7)(6)(5) Word-Byte Swap
JPEG_DECODE_DATA_FORMAT_LONGWORD_SWAP	(5)(6)(7)(8)(1)(2)(3)(4) Longword Swap
JPEG_DECODE_DATA_FORMAT_LONGWORD_BYTE_SWAP	(6)(5)(8)(7)(2)(1)(4)(3) Longword Byte Swap
JPEG_DECODE_DATA_FORMAT_LONGWORD_WORD_SWAP	(7)(8)(5)(6)(3)(4)(1)(2) Longword Word Swap
JPEG_DECODE_DATA_FORMAT_LONGWORD_WORD_BYTE_SWAP	(8)(7)(6)(5)(4)(3)(2)(1) Longword Word Byte Swap
JPEG_DECODE_DATA_FORMAT_MAX	

8.23.7.3 jpeg_decode_pixel_format_t

`jpeg_decode_pixel_format_t`

Detailed description

Pixel Data Format

Enumerated values

Name	Description
JPEG_DECODE_PIXEL_FORMAT_ARGB8888	Pixel Data ARGB8888 format.
JPEG_DECODE_PIXEL_FORMAT_RGB565	Pixel Data RGB565 format.

8.23.7.4 jpeg_decode_status_t

`jpeg_decode_status_t`

Detailed description

JPEG HLD driver internal status information. The driver can simultaneously be in more than any one status at the same time. Parse the status bit-fields using the definitions in this enum to determine driver status

Enumerated values

Name	Description
JPEG_DECODE_STATUS_FREE	JPEG codec module is not yet open.
JPEG_DECODE_STATUS_IDLE	JPEG Codec module is open, and is not operational.
JPEG_DECODE_STATUS_RUNNING	JPEG Codec is running.
JPEG_DECODE_STATUS_DONE	JPEG Codec has successfully finished the operation.
JPEG_DECODE_STATUS_INPUT_PAUSE	JPEG Codec paused waiting for more input data.
JPEG_DECODE_STATUS_OUTPUT_PAUSE	JPEG Codec paused after decoded the number of lines specified by user.
JPEG_DECODE_STATUS_IMAGE_SIZE_READY	JPEG decoding operation obtained image size, and paused.
JPEG_DECODE_STATUS_ERROR	JPEG Codec module encountered an error.
JPEG_DECODE_STATUS_HEADER_PROCESSING	JPEG Codec module is reading the JPEG header information.

8.23.7.5 jpeg_decode_subsample_t

`jpeg_decode_subsample_t`

Detailed description

Data type for horizontal and vertical subsample settings. This setting applies only to the decoding operation.

Enumerated values

Name	Description
JPEG_DECODE_OUTPUT_NO_SUBSAMPLE	No subsample. The image is decoded with no reduction in size.
JPEG_DECODE_OUTPUT_SUBSAMPLE_HALF	The output image size is reduced by half.
JPEG_DECODE_OUTPUT_SUBSAMPLE_ONE_QUARTER	The output image size is reduced to one-quarter.
JPEG_DECODE_OUTPUT_SUBSAMPLE_ONE_EIGHTH	The output image size is reduced to one-eighth.

8.23.7.6 jpeg_decode_count_enable_t

`jpeg_decode_count_enable_t`

Detailed description

Data type for decoding count mode enable.

Enumerated values

Name	Description
JPEG_DECODE_COUNT_DISABLE	Count mode disable.
JPEG_DECODE_COUNT_ENABLE	Count mode enable.

8.23.7.7 jpeg_decode_resume_mode_t

`jpeg_decode_resume_mode_t`

Detailed description

Data type for decoding count mode enable.

Enumerated values

Name	Description
JPEG_DECODE_COUNT_MODE_ADDRESS_CONTINUE	The data buffer address will not be initialized when resuming image data lines.
JPEG_DECODE_COUNT_MODE_ADDRESS_REINITIALIZE	The data buffer address will be initialized when resuming image data lines.

8.23.7.8 jpeg_decode_ctrl_t

```
typedef void jpeg_decode_ctrl_t
```

Detailed description

JPEG decode control block. Allocate an instance specific control block to pass into the JPEG decode API calls. Implemented as

- jpeg_decode_instance_ctrl_t

8.23.8 API Structures

8.23.8.1 jpeg_decode_callback_args_t

jpeg_decode_callback_args_t

Detailed description

Callback status structure

Variables

- jpeg_decode_status_t status

  JPEG status.
- void const * p_context

  Pointer to user-provided context.

8.23.8.2 jpeg_decode_cfg_t

jpeg_decode_cfg_t

Detailed description

User configuration structure, used in open function.

Variables

- jpeg_decode_color_space_t color_space

  Color space.
- jpeg_decode_data_format_t input_data_format

  Input data stream byte order.
- jpeg_decode_data_format_t output_data_format

  Output data stream byte order.
- jpeg_decode_pixel_format_t pixel_format

  Pixel format.
- uint8_t alpha_value

  Alpha value to be applied to decoded pixel data. Only valid for ARGB888 format.

- uint8_t jdti_ipl

  Data transfer interrupt priority.
- uint8_t jedi_ipl

  Decompression interrupt priority.
- void(* p_callback)(jpeg_decode_callback_args_t *p_args)

  User-supplied callback functions.
- void const * p_context

  Placeholder for user data. Passed to user callback in jpeg_decode_callback_args_t.

8.23.8.3 jpeg_decode_api_t

jpeg_decode_api_t

Detailed description

JPEG functions implemented at the HAL layer will follow this API.

8.23.8.4 open

```
ssp_err_t(* jpeg_decode_api_t::open) (jpeg_decode_ctrl_t *const p_ctrl,
jpeg_decode_cfg_t const *const p_cfg)
```

Detailed description

Initial configuration Implemented as

- R_JPEG_Decode_Open

NOTE: none

Table 1195:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to control block. Must be declared by user. Elements set here.
p_cfg	in	Pointer to configuration structure. All elements of this structure must be set by user.

Parameter p_ctrl

Definition: `jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode control block. Allocate an instance specific control block to pass into the JPEG decode API calls. Implemented asjpeg_decode_instance_ctrl_t

Parameter p_cfg

Definition: `jpeg_decode_cfg_t const *const p_cfg`

User configuration structure, used in open function.

- `jpeg_decode_cfg_t`::`jpeg_decode_color_space_t`

  Color space.

  Enumerated as:

  – JPEG_DECODE_COLOR_SPACE_YCBCR444

  – JPEG_DECODE_COLOR_SPACE_YCBCR422

  – JPEG_DECODE_COLOR_SPACE_YCBCR420

  – JPEG_DECODE_COLOR_SPACE_YCBCR411

- `jpeg_decode_cfg_t`::`jpeg_decode_data_format_t`

  Input data stream byte order.

  Enumerated as:

  – JPEG_DECODE_DATA_FORMAT_NORMAL

  – JPEG_DECODE_DATA_FORMAT_BYTE_SWAP

  – JPEG_DECODE_DATA_FORMAT_WORD_SWAP

  – JPEG_DECODE_DATA_FORMAT_WORD_BYTE_SWAP

  – JPEG_DECODE_DATA_FORMAT_LONGWORD_SWAP

  – JPEG_DECODE_DATA_FORMAT_LONGWORD_BYTE_SWAP

  – JPEG_DECODE_DATA_FORMAT_LONGWORD_WORD_SWAP

  – JPEG_DECODE_DATA_FORMAT_LONGWORD_WORD_BYTE_SWAP

  – JPEG_DECODE_DATA_FORMAT_MAX

- `jpeg_decode_cfg_t`::`jpeg_decode_data_format_t`

  Output data stream byte order.

  Enumerated as:

  – JPEG_DECODE_DATA_FORMAT_NORMAL

  – JPEG_DECODE_DATA_FORMAT_BYTE_SWAP

  – JPEG_DECODE_DATA_FORMAT_WORD_SWAP

  – JPEG_DECODE_DATA_FORMAT_WORD_BYTE_SWAP

  – JPEG_DECODE_DATA_FORMAT_LONGWORD_SWAP

  - JPEG_DECODE_DATA_FORMAT_LONGWORD_BYTE_SWAP
  - JPEG_DECODE_DATA_FORMAT_LONGWORD_WORD_SWAP
  - JPEG_DECODE_DATA_FORMAT_LONGWORD_WORD_BYTE_SWAP
  - JPEG_DECODE_DATA_FORMAT_MAX
- jpeg_decode_cfg_t::jpeg_decode_pixel_format_t

  Pixel format.

  Enumerated as:

  - JPEG_DECODE_PIXEL_FORMAT_ARGB8888
  - JPEG_DECODE_PIXEL_FORMAT_RGB565
- jpeg_decode_cfg_t::alpha_value

  Alpha value to be applied to decoded pixel data. Only valid for ARGB888 format.
- jpeg_decode_cfg_t::jdti_ipl

  Data transfer interrupt priority.
- jpeg_decode_cfg_t::jedi_ipl

  Decompression interrupt priority.
- jpeg_decode_cfg_t::p_callback

  User-supplied callback functions.
- jpeg_decode_cfg_t::p_context

  Placeholder for user data. Passed to user callback in jpeg_decode_callback_args_t.

8.23.8.5 outputBufferSet

```
ssp_err_t(* jpeg_decode_api_t::outputBufferSet) (jpeg_decode_ctrl_t *const
p_ctrl, void *p_buffer, uint32_t buffer_size)
```

Detailed description

Assign output buffer to JPEG codec for storing output data. Implemented as

- R_JPEG_Decode_OutputBufferSet

NOTE: The JPEG codec module must have been opened properly.

NOTE: The buffer starting address must be 8-byte aligned. For the decoding process, the HLD driver automatically computes the number of lines of the image to decoded so the output data fits into the given space. If the supplied output buffer is not able to hold the entire frame, the application should call the Output Full Callback function so it can be notified when additional buffer space is needed.

Table 1196:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call.
p_buffer	in	Pointer to the output buffer space
buffer_size	in	Size of the output buffer

Parameter p_ctrl

Definition: `jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode control block. Allocate an instance specific control block to pass into the JPEG decode API calls. Implemented asjpeg_decode_instance_ctrl_t

Parameter p_buffer

`const`

Parameter buffer_size

`uint32_t`

8.23.8.6 horizontalStrideSet

```
ssp_err_t(* jpeg_decode_api_t::horizontalStrideSet) (jpeg_decode_ctrl_t *const
p_ctrl, uint32_t horizontal_stride)
```

Detailed description

Configure the horizontal stride value. Implemented as

- R_JPEG_Decode_HorizontalStrideSet

NOTE: The JPEG codec module must have been opened properly.

Table 1197:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call.
horizontal_stride	in	Horizontal stride value to be used for the decoded image data.

Table 1197:Parameters (Continued)

Name	Direction	Description
buffer_size	in	Size of the output buffer

Parameter p_ctrl

Definition: `jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode control block. Allocate an instance specific control block to pass into the JPEG decode API calls. Implemented asjpeg_decode_instance_ctrl_t

Parameter horizontal_stride

`uint32_t`

Parameter buffer_size

8.23.8.7 imageSubsampleSet

```
ssp_err_t(* jpeg_decode_api_t::imageSubsampleSet) (jpeg_decode_ctrl_t *const
p_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t
vertical_subsample)
```

Detailed description

Configure the horizontal and vertical subsample settings. Implemented as

- R_JPEG_Decode_ImageSubsampleSet

NOTE: The JPEG codec module must have been opened properly.

Table 1198:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call.
horizontal_subsample	in	Horizontal subsample value
vertical_subsample	in	Vertical subsample value

Parameter p_ctrl

Definition: `jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode control block. Allocate an instance specific control block to pass into the JPEG decode API calls. Implemented asjpeg_decode_instance_ctrl_t

Parameter horizontal_subsample

Definition: `jpeg_decode_subsample_thorizontal_subsample`

Data type for horizontal and vertical subsample settings. This setting applies only to the decoding operation.

Parameter vertical_subsample

Definition: `jpeg_decode_subsample_tvertical_subsample`

Data type for horizontal and vertical subsample settings. This setting applies only to the decoding operation.

8.23.8.8 inputBufferSet

```
ssp_err_t(* jpeg_decode_api_t::inputBufferSet) (jpeg_decode_ctrl_t *const p_ctrl,
void *p_buffer, uint32_t buffer_size)
```

Detailed description

Assign input data buffer to JPEG codec. Implemented as

- R_JPEG_Decode_InputBufferSet

NOTE: the JPEG codec module must have been opened properly.

NOTE: The buffer starting address must be 8-byte aligned.

Table 1199:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call.
p_buffer	in	Pointer to the input buffer space
buffer_size	in	Size of the input buffer

Parameter p_ctrl

Definition: `jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode control block. Allocate an instance specific control block to pass into the JPEG decode API calls. Implemented asjpeg_decode_instance_ctrl_t

Parameter p_buffer

`const`

Parameter buffer_size

`uint32_t`

8.23.8.9 linesDecodedGet

```
ssp_err_t(* jpeg_decode_api_t::linesDecodedGet) (jpeg_decode_ctrl_t *const
p_ctrl, uint32_t *const p_lines)
```

Detailed description

Return the number of lines decoded into the output buffer. Implemented as

- R_JPEG_Decode_LinesDecodedGet

NOTE: the JPEG codec module must have been opened properly.

Table 1200:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call.
p_lines	out	Number of lines decoded

Parameter p_ctrl

Definition: `jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode control block. Allocate an instance specific control block to pass into the JPEG decode API calls. Implemented asjpeg_decode_instance_ctrl_t

Parameter p_lines

`uint32_t`

8.23.8.10 imageSizeGet

```
ssp_err_t(* jpeg_decode_api_t::imageSizeGet) (jpeg_decode_ctrl_t *const p_ctrl,
uint16_t *p_horizontal_size, uint16_t *p_vertical_size)
```

Detailed description

Retrieve image size during decoding operation. Implemented as

- R_JPEG_Decode_ImageSizeGet

NOTE: the JPEG codec module must have been opened properly.

NOTE: If the encoding or the decoding operation is finished without errors, the HLD driver automatically closes the device. In this case, application does not need to explicitly close the JPEG device.

Table 1201:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call.
p_horizontal_size	out	Image horizontal size, in number of pixels.
p_vertical_size	out	Image vertical size, in number of pixels.

Parameter p_ctrl

Definition: `jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode control block. Allocate an instance specific control block to pass into the JPEG decode API calls. Implemented asjpeg_decode_instance_ctrl_t

Parameter p_horizontal_size

`uint16_t`

Parameter p_vertical_size

`uint16_t`

8.23.8.11 statusGet

```
ssp_err_t(* jpeg_decode_api_t::statusGet) (jpeg_decode_ctrl_t *const p_ctrl,
jpeg_decode_status_t *const p_status)
```

Detailed description

Retrieve current status of the JPEG codec module. Implemented as

- R_JPEG_Decode_StatusGet

NOTE: the JPEG codec module must have been opened properly.

Table 1202:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call.
p_status	out	JPEG module status

Parameter p_ctrl

Definition: jpeg_decode_ctrl_t*const p_ctrl

JPEG decode control block. Allocate an instance specific control block to pass into the JPEG decode API calls. Implemented asjpeg_decode_instance_ctrl_t

Parameter p_status

Definition: jpeg_decode_status_t*const p_status

JPEG HLD driver internal status information. The driver can simultaneously be in more than any one status at the same time. Parse the status bit-fields using the definitions in this enum to determine driver status

8.23.8.12 close

```
ssp_err_t(* jpeg_decode_api_t::close) (jpeg_decode_ctrl_t *const p_ctrl)
```

Detailed description

Cancel an outstanding operation. Implemented as

- R_JPEG_Decode_Close

NOTE: the JPEG codec module must have been opened properly.

NOTE: If the encoding or the decoding operation is finished without errors, the HLD driver automatically closes the device. In this case, application does not need to explicitly close the JPEG device.

Table 1203:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in jpeg_decode_api_t::Open call.

Parameter p_ctrl

Definition: jpeg_decode_ctrl_t*const p_ctrl

JPEG decode control block. Allocate an instance specific control block to pass into the JPEG decode API calls. Implemented asjpeg_decode_instance_ctrl_t

8.23.8.13 versionGet

```
ssp_err_t(* jpeg_decode_api_t::versionGet) (ssp_version_t *p_version)
```

Detailed description

Get version and store it in provided pointer p_version. Implemented as

- R_JPEG_Decode_VersionGet

Table 1204:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.23.8.14 pixelFormatGet

```
ssp_err_t(* jpeg_decode_api_t::pixelFormatGet) (jpeg_decode_ctrl_t *const p_ctrl,
jpeg_decode_color_space_t *const p_color_space)
```

Detailed description

Get the input pixel format. Implemented as

- R_JPEG_Decode_PixelFormatGet

NOTE: the JPEG codec module must have been opened properly.

Table 1205:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call.
p_color_space	out	JPEG input format.

Parameter p_ctrl

Definition: `jpeg_decode_ctrl_t*const p_ctrl`

JPEG decode control block. Allocate an instance specific control block to pass into the JPEG decode API calls. Implemented asjpeg_decode_instance_ctrl_t

Parameter p_color_space

Definition: `jpeg_decode_color_space_t*const p_color_space`

Image color space definitions

8.23.8.15 jpeg_decode_instance_t

jpeg_decode_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- jpeg_decode_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- jpeg_decode_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- jpeg_decode_api_t const * p_api

  Pointer to the API structure for this instance.

8.24 JPEG Encode Interface

Interface for JPEG encode functions.

8.24.1 Summary

The JPEG ENCODE interface provides JPEG encoder functionality. It allows application to convert a JPEG image into bitmap data suitable for display frame buffer.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

JPEG ENCODE Interface description: JPEG Decode Driver

8.24.2 Interface API

jpeg_encode_api_t

Function name	Description
.open	Initial configuration
.imageParameterSet	Set image parameters to JPEG Codec
.outputBufferSet	Assign output buffer to JPEG codec for storing output data.
.inputBufferSet	Assign input data buffer to JPEG codec.
.statusGet	Retrieve current status of the JPEG codec module.

Function name	Description
.close	Cancel an outstanding operation.
.versionGet	Get version and store it in provided pointer p_version.

8.24.3 Data structures

- jpeg_encode_raw_image_parameters
- jpeg_encode_callback_args_t
- jpeg_encode_cfg_t
- jpeg_encode_instance_t

8.24.4 Enumerations

- jpeg_encode_data_format_t
- jpeg_encode_status_t
- jpeg_encode_count_t
- jpeg_encode_resume_mode_t

8.24.5 Typedefs

- jpeg_encode_ctrl_t

8.24.6 Defines

- #define JPEG_ENCODE_API_VERSION_MAJOR

  Initial value: `(1U)`

  Register definitions, common services and error codes.
- #define JPEG_ENCODE_API_VERSION_MINOR

  Initial value: `(0U)`

8.24.7 API Data

8.24.7.1 jpeg_encode_data_format_t

`jpeg_encode_data_format_t`

Detailed description

Multi-byte Data Format

Enumerated values

Name	Description
JPEG_ENCODE_DATA_FORMAT_NORMAL	(1)(2)(3)(4)(5)(6)(7)(8) Normal byte order
JPEG_ENCODE_DATA_FORMAT_BYTE_SWAP	(2)(1)(4)(3)(6)(5)(8)(7) Byte Swap
JPEG_ENCODE_DATA_FORMAT_WORD_SWAP	(3)(4)(1)(2)(7)(8)(5)(6) Word Swap
JPEG_ENCODE_DATA_FORMAT_WORD_BYTE_SWAP	(4)(3)(2)(1)(8)(7)(6)(5) Word-Byte Swap
JPEG_ENCODE_DATA_FORMAT_LONGWORD_SWAP	(5)(6)(7)(8)(1)(2)(3)(4) Longword Swap
JPEG_ENCODE_DATA_FORMAT_LONGWORD_BYTE_SWAP	(6)(5)(8)(7)(2)(1)(4)(3) Longword Byte Swap
JPEG_ENCODE_DATA_FORMAT_LONGWORD_WORD_SWAP	(7)(8)(5)(6)(3)(4)(1)(2) Longword Word Swap
JPEG_ENCODE_DATA_FORMAT_LONGWORD_WORD_BYTE_SWAP	(8)(7)(6)(5)(4)(3)(2)(1) Longword Word Byte Swap
JPEG_ENCODE_DATA_FORMAT_MAX	

8.24.7.2 jpeg_encode_status_t

```
jpeg_encode_status_t
```

Detailed description

JPEG HLD driver internal status information. The driver can simultaneously be in more than any one status at the same time. Parse the status bit-fields using the definitions in this enum to determine driver status

Enumerated values

Name	Description
JPEG_ENCODE_STATUS_FREE	JPEG codec module is not yet open.
JPEG_ENCODE_STATUS_IDLE	JPEG Codec module is open, and is not operational.
JPEG_ENCODE_STATUS_RUNNING	JPEG Codec is running.
JPEG_ENCODE_STATUS_DONE	JPEG Codec has successfully finished the operation.

Name	Description
JPEG_ENCODE_STATUS_INPUT_PAUSE	JPEG Codec paused waiting for more input data.

8.24.7.3 jpeg_encode_count_t

```
jpeg_encode_count_t
```

Detailed description

Data type for encoding count mode enable.

Enumerated values

Name	Description
JPEG_ENCODE_COUNT_DISABLE	Count mode disable.
JPEG_ENCODE_COUNT_ENABLE	Count mode enable.

8.24.7.4 jpeg_encode_resume_mode_t

```
jpeg_encode_resume_mode_t
```

Detailed description

Data type for encoding resume mode

Enumerated values

Name	Description
JPEG_ENCODE_COUNT_MODE_ADDRESS_CONTINUE	The data buffer address will not be initialized when resuming image data lines.
JPEG_ENCODE_COUNT_MODE_ADDRESS_REINITIALIZE	The data buffer address will be initialized when resuming image data lines.

8.24.7.5 jpeg_encode_ctrl_t

```
typedef void jpeg_encode_ctrl_t
```

Detailed description

JPEG encode control block. Allocate an instance specific control block to pass into the JPEG encode API calls. Implemented as

- jpeg_encode_instance_ctrl_t

8.24.8 API Structures

8.24.8.1 jpeg_encode_raw_image_parameters

jpeg_encode_raw_image_parameters

Detailed description

Image parameter structure

Variables

- uint16_t horizontal_stride

  Horizontal stride.

- uint16_t horizontal_resolution

  Horizontal Resolution in pixel.

- uint16_t vertical_resolution

  Vertical Resolution in pixel.

8.24.8.2 jpeg_encode_callback_args_t

jpeg_encode_callback_args_t

Detailed description

Callback status structure

Variables

- jpeg_encode_status_t status

  JPEG status.

- uint32_t image_size

  JPEG image size.

- void const * p_context

  Pointer to user-provided context.

8.24.8.3 jpeg_encode_cfg_t

jpeg_encode_cfg_t

Detailed description

User configuration structure, used in open function.

Variables

- jpeg_encode_data_format_t input_data_format

  Input data stream byte order.

- jpeg_encode_data_format_t output_data_format

  Output data stream byte order.
- uint16_t dri_marker

  DRI Marker setting 0 :- No DRI and RST marker.
- uint8_t jdti_ipl

  Data transfer interrupt priority.
- uint8_t jedi_ipl

  Decompression interrupt priority.
- uint8_t quality_factor

  JPEG image quality.
- uint16_t vertical_resolution

  vertical resolution of input image
- uint16_t horizontal_resolution

  horizontal resolution of input image
- uint8_t const * p_quant_luma_table

  Luma table.
- uint8_t const * p_quant_croma_table

  croma table
- uint8_t const * p_huffman_luma_ac_table

  Huffman AC table for luma.
- uint8_t const * p_huffman_luma_dc_table

  Huffman DC table for luma.
- uint8_t const * p_huffman_croma_ac_table

  Huffman AC table for croma.
- uint8_t const * p_huffman_croma_dc_table

  Huffman DC table for croma.
- void(* p_callback)(jpeg_encode_callback_args_t *p_args)

  User-supplied callback functions.
- void const * p_context

  Placeholder for user data. Passed to user callback in jpeg_encode_callback_args_t.

8.24.8.4 jpeg_encode_api_t

jpeg_encode_api_t

Detailed description

JPEG functions implemented at the HAL layer will follow this API.

8.24.8.5 open

```
ssp_err_t(* jpeg_encode_api_t::open) (jpeg_encode_ctrl_t *const p_ctrl,
jpeg_encode_cfg_t const *const p_cfg)
```

Detailed description

Initial configuration Implemented as

- R_JPEG_Encode_Open

NOTE: none

Table 1206:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to control block. Must be declared by user. Elements set here.
p_cfg	in	Pointer to configuration structure. All elements of this structure must be set by user.

Parameter p_ctrl

Definition: `jpeg_encode_ctrl_t*const p_ctrl`

JPEG encode control block. Allocate an instance specific control block to pass into the JPEG encode API calls. Implemented asjpeg_encode_instance_ctrl_t

Parameter p_cfg

Definition: `jpeg_encode_cfg_t const *const p_cfg`

User configuration structure, used in open function.

- `jpeg_encode_cfg_t`::`jpeg_encode_data_format_t`

  Input data stream byte order.

  Enumerated as:

  – JPEG_ENCODE_DATA_FORMAT_NORMAL

  – JPEG_ENCODE_DATA_FORMAT_BYTE_SWAP

  – JPEG_ENCODE_DATA_FORMAT_WORD_SWAP

  – JPEG_ENCODE_DATA_FORMAT_WORD_BYTE_SWAP

  – JPEG_ENCODE_DATA_FORMAT_LONGWORD_SWAP

  - JPEG_ENCODE_DATA_FORMAT_LONGWORD_BYTE_SWAP
  - JPEG_ENCODE_DATA_FORMAT_LONGWORD_WORD_SWAP
  - JPEG_ENCODE_DATA_FORMAT_LONGWORD_WORD_BYTE_SWAP
  - JPEG_ENCODE_DATA_FORMAT_MAX
- jpeg_encode_cfg_t::jpeg_encode_data_format_t

  Output data stream byte order.

  Enumerated as:
  - JPEG_ENCODE_DATA_FORMAT_NORMAL
  - JPEG_ENCODE_DATA_FORMAT_BYTE_SWAP
  - JPEG_ENCODE_DATA_FORMAT_WORD_SWAP
  - JPEG_ENCODE_DATA_FORMAT_WORD_BYTE_SWAP
  - JPEG_ENCODE_DATA_FORMAT_LONGWORD_SWAP
  - JPEG_ENCODE_DATA_FORMAT_LONGWORD_BYTE_SWAP
  - JPEG_ENCODE_DATA_FORMAT_LONGWORD_WORD_SWAP
  - JPEG_ENCODE_DATA_FORMAT_LONGWORD_WORD_BYTE_SWAP
  - JPEG_ENCODE_DATA_FORMAT_MAX
- jpeg_encode_cfg_t::dri_marker

  DRI Marker setting 0 :- No DRI and RST marker.
- jpeg_encode_cfg_t::jdti_ipl

  Data transfer interrupt priority.
- jpeg_encode_cfg_t::jedi_ipl

  Decompression interrupt priority.
- jpeg_encode_cfg_t::quality_factor

  JPEG image quality.
- jpeg_encode_cfg_t::vertical_resolution

  vertical resolution of input image
- jpeg_encode_cfg_t::horizontal_resolution

  horizontal resolution of input image
- jpeg_encode_cfg_t::p_quant_luma_table

  Luma table.
- jpeg_encode_cfg_t::p_quant_croma_table

  croma table

- jpeg_encode_cfg_t::p_huffman_luma_ac_table

  Huffman AC table for luma.
- jpeg_encode_cfg_t::p_huffman_luma_dc_table

  Huffman DC table for luma.
- jpeg_encode_cfg_t::p_huffman_croma_ac_table

  Huffman AC table for croma.
- jpeg_encode_cfg_t::p_huffman_croma_dc_table

  Huffman DC table for croma.
- jpeg_encode_cfg_t::p_callback

  User-supplied callback functions.
- jpeg_encode_cfg_t::p_context

  Placeholder for user data. Passed to user callback in jpeg_encode_callback_args_t.

8.24.8.6 imageParameterSet

```
ssp_err_t(* jpeg_encode_api_t::imageParameterSet) (jpeg_encode_ctrl_t *const
p_ctrl, jpeg_encode_raw_image_parameters *p_raw_image_parameters)
```

Detailed description

Set image parameters to JPEG Codec Implemented as

- R_JPEG_Encode_ImageParameterSet

NOTE: The JPEG codec module must have been opened properly.

Table 1207:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to control block. Must be declared by user. Elements set here.
p_raw_image_parameters	in	Pointer to the RAW image parameters

Parameter p_ctrl

Definition: jpeg_encode_ctrl_t*const p_ctrl

JPEG encode control block. Allocate an instance specific control block to pass into the JPEG encode API calls. Implemented asjpeg_encode_instance_ctrl_t

Parameter p_raw_image_parameters

```
const
```

8.24.8.7 outputBufferSet

```
ssp_err_t(* jpeg_encode_api_t::outputBufferSet) (jpeg_encode_ctrl_t *const
p_ctrl, void *p_buffer)
```

Detailed description

Assign output buffer to JPEG codec for storing output data. Implemented as

- R_JPEG_Encode_OutputBufferSet

NOTE: The JPEG codec module must have been opened properly.

NOTE: The buffer starting address must be 8-byte aligned.

Table 1208:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call.
p_buffer	in	Pointer to the output buffer space

Parameter p_ctrl

Definition: `jpeg_encode_ctrl_t*const p_ctrl`

JPEG encode control block. Allocate an instance specific control block to pass into the JPEG encode API calls. Implemented asjpeg_encode_instance_ctrl_t

Parameter p_buffer

```
const
```

8.24.8.8 inputBufferSet

```
ssp_err_t(* jpeg_encode_api_t::inputBufferSet) (jpeg_encode_ctrl_t *const p_ctrl,
void *p_buffer, uint32_t buffer_size)
```

Detailed description

Assign input data buffer to JPEG codec. Implemented as

- R_JPEG_Encode_InputBufferSet

NOTE: the JPEG codec module must have been opened properly, output buffer and image parameter must be set prior to

call this function.

NOTE: The buffer starting address must be 8-byte aligned.

Table 1209:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call.
p_buffer	in	Pointer to the input buffer space
buffer_size	in	Size of the input buffer

Parameter p_ctrl

Definition: `jpeg_encode_ctrl_t*const p_ctrl`

JPEG encode control block. Allocate an instance specific control block to pass into the JPEG encode API calls. Implemented asjpeg_encode_instance_ctrl_t

Parameter p_buffer

`const`

Parameter buffer_size

`uint32_t`

8.24.8.9 statusGet

```
ssp_err_t(* jpeg_encode_api_t::statusGet) (jpeg_encode_ctrl_t *const p_ctrl,
volatile jpeg_encode_status_t *p_status)
```

Detailed description

Retrieve current status of the JPEG codec module. Implemented as

- R_JPEG_Encode_StatusGet

NOTE: the JPEG codec module must have been opened properly.

Table 1210:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call.
p_status	out	JPEG module status

Parameter p_ctrl

Definition: `jpeg_encode_ctrl_t*const p_ctrl`

JPEG encode control block. Allocate an instance specific control block to pass into the JPEG encode API calls. Implemented asjpeg_encode_instance_ctrl_t

Parameter p_status

Definition: `jpeg_encode_status_tjpeg_encode_status_t *p_status`

JPEG HLD driver internal status information. The driver can simultaneously be in more than any one status at the same time. Parse the status bit-fields using the definitions in this enum to determine driver status

8.24.8.10 close

```
ssp_err_t(* jpeg_encode_api_t::close) (jpeg_encode_ctrl_t *const p_ctrl)
```

Detailed description

Cancel an outstanding operation. Implemented as

- R_JPEG_Encode_Close

NOTE: the JPEG codec module must have been opened properly.

Table 1211:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in jpeg_encode_api_t::Open call.

Parameter p_ctrl

Definition: `jpeg_encode_ctrl_t*const p_ctrl`

JPEG encode control block. Allocate an instance specific control block to pass into the JPEG encode API calls. Implemented asjpeg_encode_instance_ctrl_t

8.24.8.11 versionGet

```
ssp_err_t(* jpeg_encode_api_t::versionGet) (ssp_version_t *p_version)
```

Detailed description

Get version and store it in provided pointer p_version. Implemented as

- R_JPEG_Encode_VersionGet

Table 1212:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.24.8.12 jpeg_encode_instance_t

jpeg_encode_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- jpeg_encode_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- jpeg_encode_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- jpeg_encode_api_t const * p_api

  Pointer to the API structure for this instance.

8.25 Key Matrix Interface

Interface for key matrix functions.

8.25.1 Summary

The KEYMATRIX interface provides standard KeyMatrix functionality including event generation on a rising or falling edge for one or more channels at the same time. The generated event indicates all channels that are active in that instant via a bit mask. This allows the interface to be used with a matrix configuration or a one-to-one hardware implementation that is triggered on either a rising or a falling edge.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Key Matrix Interface description: Key Matrix Driver

8.25.2 Interface API

keymatrix_api_t

Function name	Description
.open	Initial configuration.
.enable	Enable Key interrupt
.disable	Disable Key interrupt.
.triggerSet	Set trigger for Key interrupt.
.close	Allow driver to be reconfigured. May reduce power consumption.
.versionGet	Get version and store it in provided pointer p_version.

8.25.3 Data structures

- keymatrix_callback_args_t
- keymatrix_cfg_t
- keymatrix_instance_t

8.25.4 Enumerations

- keymatrix_trigger_t

8.25.5 Typedefs

- keymatrix_channels_t
- keymatrix_ctrl_t

8.25.6 Defines

- #define KEYMATRIX_API_VERSION_MAJOR

  Initial value: `(1U)`

  KEY MATRIX API version number (Major)
- #define KEYMATRIX_API_VERSION_MINOR

  Initial value: `(1U)`

  KEY MATRIX API version number (Minor)

8.25.7 API Data

8.25.7.1 keymatrix_trigger_t

```
keymatrix_trigger_t
```

Detailed description

Trigger type: rising edge, falling edge

Enumerated values

Name	Description
KEYMATRIX_TRIG_FALLING	Falling edge trigger.
KEYMATRIX_TRIG_RISING	Rising edge trigger.

8.25.7.2 keymatrix_channels_t

```
typedef uint32_t keymatrix_channels_t
```

Detailed description

Channel definition. This is a bit mask with each bit from 0-7 representing channels 0-7 respectively.

8.25.7.3 keymatrix_ctrl_t

```
typedef void keymatrix_ctrl_t
```

Detailed description

Key matrix control block. Allocate an instance specific control block to pass into the key matrix API calls. Implemented as

- kint_instance_ctrl_t

8.25.8 API Structures

8.25.8.1 keymatrix_callback_args_t

keymatrix_callback_args_t

Detailed description

Callback function parameter data

Variables

- void const * p_context

  Holder for user data. Set in keymatrix_api_t::open function in keymatrix_cfg_t.

- keymatrix_channels_t channels

  Bit vector representing the physical hardware channel(s) that caused the interrupt. The bit vector is used for compatibility with matrix designs where more than one input will be active at once. Not all HAL drivers support matrix mode. See r_kint.h for details.

8.25.8.2 keymatrix_cfg_t

keymatrix_cfg_t

Detailed description

User configuration structure, used in open function

Variables

- keymatrix_channels_t channels

  Key Input channel(s). Bit mask of channels to open.

- keymatrix_trigger_t trigger

  Key Input trigger setting.

- bool autostart

  Start operation and enable interrupts during open().

- void(* p_callback)(keymatrix_callback_args_t *p_args)

  Callback for key interrupt ISR.

- void const * p_context

  Holder for user data. Passed to callback in keymatrix_user_cb_data_t.

- void const * p_extend

  Extension parameter for hardware specific settings.

- uint8_t irq_ipl

  Interrupt priority level.

8.25.8.3 keymatrix_api_t

keymatrix_api_t

Detailed description

Key Matrix driver structure. Key Matrix functions implemented at the HAL layer will use this API.

8.25.8.4 open

```
ssp_err_t(* keymatrix_api_t::open) (keymatrix_ctrl_t *const p_ctrl,
keymatrix_cfg_t const *const p_cfg)
```

Detailed description

Initial configuration. Implemented as

- R_KINT_KEYMATRIX_Open

Table 1213:Parameters

Name	Direction	Description
p_ctrl	out	Pointer to control block. Must be declared by user. Value set in this function.
p_cfg	in	Pointer to configuration structure. All elements of the structure must be set by user.

Parameter p_ctrl

Definition: `keymatrix_ctrl_t*const p_ctrl`

Key matrix control block. Allocate an instance specific control block to pass into the key matrix API calls. Implemented askint_instance_ctrl_t

Parameter p_cfg

Definition: `keymatrix_cfg_t const *const p_cfg`

User configuration structure, used in open function

- `keymatrix_cfg_t`::`keymatrix_channels_t`

  Key Input channel(s). Bit mask of channels to open.

- `keymatrix_cfg_t`::`keymatrix_trigger_t`

  Key Input trigger setting.

  Enumerated as:

  – KEYMATRIX_TRIG_FALLING

– KEYMATRIX_TRIG_RISING

- keymatrix_cfg_t::autostart

  Start operation and enable interrupts during open().

- keymatrix_cfg_t::p_callback

  Callback for key interrupt ISR.

- keymatrix_cfg_t::p_context

  Holder for user data. Passed to callback in keymatrix_user_cb_data_t.

- keymatrix_cfg_t::p_extend

  Extension parameter for hardware specific settings.

- keymatrix_cfg_t::irq_ipl

  Interrupt priority level.

8.25.8.5 enable

```
ssp_err_t(* keymatrix_api_t::enable) (keymatrix_ctrl_t *const p_ctrl)
```

Detailed description

Enable Key interrupt Implemented as

- R_KINT_KEYMATRIX_Enable

Table 1214:Parameters

Name	Direction	Description
p_ctrl	in	Control block pointer set in Open call for this Key interrupt.

Parameter p_ctrl

Definition: keymatrix_ctrl_t*const p_ctrl

Key matrix control block. Allocate an instance specific control block to pass into the key matrix API calls. Implemented askint_instance_ctrl_t

8.25.8.6 disable

```
ssp_err_t(* keymatrix_api_t::disable) (keymatrix_ctrl_t *const p_ctrl)
```

Detailed description

Disable Key interrupt. Implemented as

- R_KINT_KEYMATRIX_Disable

Table 1215:Parameters

Name	Direction	Description
p_ctrl	in	Control block pointer set in Open call for this Key interrupt.

Parameter p_ctrl

Definition: `keymatrix_ctrl_t*const p_ctrl`

Key matrix control block. Allocate an instance specific control block to pass into the key matrix API calls. Implemented askint_instance_ctrl_t

8.25.8.7 triggerSet

```
ssp_err_t(* keymatrix_api_t::triggerSet) (keymatrix_ctrl_t *const p_ctrl,
keymatrix_trigger_t const trigger)
```

Detailed description

Set trigger for Key interrupt. Implemented as

- R_KINT_KEYMATRIX_TriggerSet

Table 1216:Parameters

Name	Direction	Description
p_ctrl	in	Control block pointer set in Open call for this Key interrupt.
trigger	in	Trigger source for key interrupt; defined in enumeration of keymatrix_trigger_t.

Parameter p_ctrl

Definition: `keymatrix_ctrl_t*const p_ctrl`

Key matrix control block. Allocate an instance specific control block to pass into the key matrix API calls. Implemented askint_instance_ctrl_t

Parameter trigger

8.25.8.8 close

```
ssp_err_t(* keymatrix_api_t::close) (keymatrix_ctrl_t *const p_ctrl)
```

Detailed description

Allow driver to be reconfigured. May reduce power consumption. Implemented as

- R_KINT_KEYMATRIX_Close

Table 1217:Parameters

Name	Direction	Description
p_ctrl	in	Control block pointer set in Open call for this Key interrupt.

Parameter p_ctrl

Definition: `keymatrix_ctrl_t*const p_ctrl`

Key matrix control block. Allocate an instance specific control block to pass into the key matrix API calls. Implemented askint_instance_ctrl_t

8.25.8.9 versionGet

```
ssp_err_t(* keymatrix_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Get version and store it in provided pointer p_version. Implemented as

- R_KINT_VersionGet

Table 1218:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.25.8.10 keymatrix_instance_t

keymatrix_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- keymatrix_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- keymatrix_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- keymatrix_api_t const * p_api

  Pointer to the API structure for this instance.

8.26 Low Power Modes Interface

Interface for accessing low power modes.

8.26.1 Summary

This section defines the API for the LPM (Low Power Mode) Driver. The LPM Driver provides several functions for controlling power consumption including stopping modules, selecting the operating mode, configuring low power modes, and transitioning to low power modes. The LPM driver supports configuration of MCU operating modes and mcu low power modes using the LPM hardware peripheral. The LPM driver supports operating modes low-voltage, low-speed, middle-speed, high-speed, and suboscillator mode. The LPM driver supports low power modes deep standby, standby, sleep, and snooze. The LPM driver supports reducing power consumption when in deep standby mode via internal power supply control and reseting the states of IO ports. The LPM driver supports disabling and enabling of the MCU's other hardware peripherals.
NOTE: Not all operating modes are available on all MCUs. Not all low power modes are available on all MCUs.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

LPM Interface description: Low Power Modes Driver

8.26.2 Interface API

lpm_api_t

Function name	Description
.init	Open the LPM driver module Initialized the LPM block according to the passed in config structure
.mstpcrSet	Set the value of all the Module Stop Control Registers
.mstpcrGet	Get the values of all the Module Stop Control Registers
.moduleStop	Stop a module

Function name	Description
.moduleStart	Run a module
.operatingPowerModeSet	Set the operating power mode
.snoozeEnable	Configure and enable snooze mode
.snoozeDisable	Disable snooze mode
.lowPowerCfg	Configure a low power mode
.wupenSet	Set the value of the Wake Up Interrupt Enable Register WUPEN
.wupenGet	Get the value of the Wake Up Interrupt Enable Register WUPEN
.deepStandbyCancelRequestEnable	Enable a Deep Standby Cancel Request
.deepStandbyCancelRequestDisable	Disable a Deep Standby Cancel Request
.lowPowerModeEnter	Enter low power mode (sleep/standby/deep standby) using WFI macro. Function will return after waking from low power mode.
.versionGet	Get the driver version based on compile time macros.

8.26.3 Data structures

- lpm_cfg_t
- lpm_instance_t

8.26.4 Enumerations

- lpm_cancel_request_edge_t
- lpm_deep_standby_t
- lpm_low_power_mode_t
- lpm_output_port_enable_t
- lpm_dpsby_t
- lpm_io_port_t
- lpm_power_supply_t
- lpm_power_save_memory_t
- lpm_code_flash_t

- lpm_snooze_request_t
- lpm_snooze_end_t
- lpm_snooze_rxd0_t
- lpm_snooze_dtc_t
- lpm_operating_power_t
- lpm_subosc_t
- lpm_mstp_t

8.26.5 Defines

- #define LPM_API_VERSION_MAJOR

  Initial value: `(2U)`

  Register definitions, common services and error codes.
- #define LPM_API_VERSION_MINOR

  Initial value: `(3U)`

8.26.6 API Data

8.26.6.1 lpm_cancel_request_edge_t

`lpm_cancel_request_edge_t`

Detailed description

Deep Standby Interrupt Edge

Enumerated values

Name	Description
LPM_CANCEL_REQUEST_EDGE_FALLING	A cancel request is generated at a falling edge.
LPM_CANCEL_REQUEST_EDGE_RISING	A cancel request is generated at a rising edge.

8.26.6.2 lpm_deep_standby_t

`lpm_deep_standby_t`

Detailed description

Deep Standby pins and signals

Enumerated values

Name	Description
LPM_DEEP_STANDBY_IRQ0_DS	IRQ0-DS Pin.
LPM_DEEP_STANDBY_IRQ1_DS	IRQ1-DS Pin.
LPM_DEEP_STANDBY_IRQ2_DS	IRQ2-DS Pin.
LPM_DEEP_STANDBY_IRQ3_DS	IRQ3-DS Pin.
LPM_DEEP_STANDBY_IRQ4_DS	IRQ4-DS Pin.
LPM_DEEP_STANDBY_IRQ5_DS	IRQ5-DS Pin.
LPM_DEEP_STANDBY_IRQ6_DS	IRQ6-DS Pin.
LPM_DEEP_STANDBY_IRQ7_DS	IRQ7-DS Pin.
LPM_DEEP_STANDBY_IRQ8_DS	IRQ8-DS Pin.
LPM_DEEP_STANDBY_IRQ9_DS	IRQ9-DS Pin.
LPM_DEEP_STANDBY_IRQ10_DS	IRQ10-DS Pin.
LPM_DEEP_STANDBY_IRQ11_DS	IRQ11-DS Pin.
LPM_DEEP_STANDBY_IRQ12_DS	IRQ12-DS Pin.
LPM_DEEP_STANDBY_IRQ13_DS	IRQ13-DS Pin.
LPM_DEEP_STANDBY_IRQ14_DS	IRQ14-DS Pin.
LPM_DEEP_STANDBY_IRQ15_DS	IRQ15-DS Pin.
LPM_DEEP_STANDBY_LVD1	LVD1.
LPM_DEEP_STANDBY_LVD2	LVD2.
LPM_DEEP_STANDBY_RTC_INTERVAL	RTC Interval Interrupt.
LPM_DEEP_STANDBY_RTC_ALARM	RTC Alarm Interrupt.
LPM_DEEP_STANDBY_NMI	NMI Pin.
LPM_DEEP_STANDBY_USBFS	USBFS Suspend/Resume.
LPM_DEEP_STANDBY_USBHS	USBHS Suspend/Resume.

Name	Description
LPM_DEEP_STANDBY_AGT1	AGT1 Underflow.

8.26.6.3 lpm_low_power_mode_t

`lpm_low_power_mode_t`

Detailed description

Low power modes

Enumerated values

Name	Description
LPM_LOW_POWER_MODE_SLEEP	Sleep mode (ARM Cortex sleep mode)
LPM_LOW_POWER_MODE_STANDBY	Software Standby mode.
LPM_LOW_POWER_MODE_DEEP	Deep Software Standby mode.

8.26.6.4 lpm_output_port_enable_t

`lpm_output_port_enable_t`

Detailed description

Output port enable

Enumerated values

Name	Description
LPM_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode.
LPM_OUTPUT_PORT_ENABLE_RETAIN	1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state.

8.26.6.5 lpm_dpsby_t

`lpm_dpsby_t`

Detailed description

Select between Software Standby and Deep Software Standby

Enumerated values

Name	Description
LPM_DPSBY_SOFTWARE_STANDBY	Sleep mode (SBYCR.SSBY = 0) / Software Standby mode (SBYCR.SSBY = 1)
LPM_DPSBY_DEEP_SOFTWARE_STANDBY	Sleep mode (SBYCR.SSBY = 0) / Deep Software Standby mode (SBYCR.SSBY = 1)

8.26.6.6 lpm_io_port_t

`lpm_io_port_t`

Detailed description

I/O port state after Deep Software Standby mode

Enumerated values

Name	Description
LPM_IO_PORT_RESET	When the Deep Software Standby mode is canceled, the I/O ports are in the reset state.
LPM_IO_PORT_NO_CHANGE	When the Deep Software Standby mode is canceled, the I/O ports are in the same state as in the Deep Software Standby mode

8.26.6.7 lpm_power_supply_t

`lpm_power_supply_t`

Detailed description

Power supply control

Enumerated values

Name	Description
LPM_POWER_SUPPLY_DEEPCUT0	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is supplied in deep software standby mode

Name	Description
LPM_POWER_SUPPLY_DEEPCUT1	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode
LPM_POWER_SUPPLY_DEEPCUT3	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode. In addition, LVD is disabled and the low power function in a poweron reset circuit is enabled

8.26.6.8 lpm_power_save_memory_t

`lpm_power_save_memory_t`

Detailed description

Power save memory control

Enumerated values

Name	Description
LPM_POWER_SAVE_MEMORY_ALL_RAM	All RAM on in Software Standby mode.
LPM_POWER_SAVE_MEMORY_48KB	48KB RAM (2000 0000h to 2000 BFFFh) on in Software Standby mode

8.26.6.9 lpm_code_flash_t

`lpm_code_flash_t`

Detailed description

Code flash operation mode

Enumerated values

Name	Description
LPM_CODE_FLASH_OPERATES	Code flash memory operates.
LPM_CODE_FLASH_STOPS	Code flash memory stops.

8.26.6.10 lpm_snooze_request_t

`lpm_snooze_request_t`

Detailed description

Snooze end control

Enumerated values

Name	Description
LPM_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request.
LPM_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request.
LPM_SNOOZE_REQUEST_KR	Enable KR snooze request.
LPM_SNOOZE_REQUEST_COMPARATOR_OC0	Enable Comparator-OC0 snooze request.
LPM_SNOOZE_REQUEST_COMPARATOR_LP	Enable Comparator-LP snooze request.
LPM_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request.

Name	Description
LPM_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request.
LPM_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request.
LPM_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request.
LPM_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request.

8.26.6.11 lpm_snooze_end_t

`lpm_snooze_end_t`

Detailed description

Snooze end control

Enumerated values

Name	Description
LPM_SNOOZE_END_VIA_WUPEN	Transition from Snooze to Normal mode directly.
LPM_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow.
LPM_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion.
LPM_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion.
LPM_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match.
LPM_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC hannel 0 compare mismatch.
LPM_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match.
LPM_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch.
LPM_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address unmatch.

8.26.6.12 lpm_snooze_rxd0_t

`lpm_snooze_rxd0_t`

Detailed description

RXD0 Snooze Request Enable

Enumerated values

Name	Description
LPM_SNOOZE_RXD0_FALLING_EDGE_IGNORE	Ignore RXD0 falling edge in Software Standby mode.
LPM_SNOOZE_RXD0_FALLING_EDGE_DETECT	Detect RXD0 falling edge in Software Standby mode as a request to transit to Snooze mode. Do not set to 1 other than in asynchronous mode.

8.26.6.13 lpm_snooze_dtc_t

`lpm_snooze_dtc_t`

Detailed description

DTC Enable in Snooze Mode

Enumerated values

Name	Description
LPM_SNOOZE_DTC_DISABLE	Disable DTC operation.
LPM_SNOOZE_DTC_ENABLE	Enable DTC operation.

8.26.6.14 lpm_operating_power_t

`lpm_operating_power_t`

Detailed description

Operating power mode

Enumerated values

Name	Description
LPM_OPERATING_POWER_HIGH_SPEED_MODE	High speed mode.
LPM_OPERATING_POWER_MIDDLE_SPEED_MODE	Middle speed mode.
LPM_OPERATING_POWER_LOW_VOLTAGE_MODE	Low voltage mode.
LPM_OPERATING_POWER_LOW_SPEED_MODE	Low speed mode.

8.26.6.15 lpm_subosc_t

`lpm_subosc_t`

Detailed description

Subosc speed mode select

Enumerated values

Name	Description
LPM_SUBOSC_OTHER	Other than Subosc-speed mode.
LPM_SUBOSC_SELECT	Subosc-speed mode.

8.26.6.16 lpm_mstp_t

`lpm_mstp_t`

Detailed description

Module stop bits

Enumerated values

Name	Description
LPM_MSTP_RAM0	RAM0.
LPM_MSTP_RAM1	RAM1.
LPM_MSTP_HSRAM	HSRAM.
LPM_MSTP_ECCRAM	ECCRAM.
LPM_MSTP_STANDBYRAM	STANDBYRAM.
LPM_MSTP_DMAC_DTC	DMAC_DTC.
LPM_MSTP_RCAN1	RCAN1.
LPM_MSTP_RCAN0	RCAN0.
LPM_MSTP_IRDA	IRDA.
LPM_MSTP_QSPI	QSPI.
LPM_MSTP_I2C2	I2C2.
LPM_MSTP_I2C1	I2C1.
LPM_MSTP_I2C0	I2C0.

Name	Description
LPM_MSTP_USBFS	USBFS.
LPM_MSTP_USBHS	USBHS.
LPM_MSTP_EPTPC_PTPEDMAC	EPTPC_PTPEDMAC.
LPM_MSTP_ETHERC1_EDMAC1	ETHERC1_EDMAC1.
LPM_MSTP_ETHERC0_EDMAC0	ETHERC0_EDMAC0.
LPM_MSTP_RSPI1	RSPI1.
LPM_MSTP_RSPI0	RSPI0.
LPM_MSTP_SCI9	SCI9.
LPM_MSTP_SCI8	SCI8.
LPM_MSTP_SCI7	SCI7.
LPM_MSTP_SCI6	SCI6.
LPM_MSTP_SCI5	SCI5.
LPM_MSTP_SCI4	SCI4.
LPM_MSTP_SCI3	SCI3.
LPM_MSTP_SCI2	SCI2.
LPM_MSTP_SCI1	SCI1.
LPM_MSTP_SCI0	SCI0.
LPM_MSTP_CAC	CAC.
LPM_MSTP_CRC	CRC.
LPM_MSTP_PDC	PDC.
LPM_MSTP_CTSU	CTSU.
LPM_MSTP_GLCDC	GLCDC.
LPM_MSTP_JPEG	JPEG.
LPM_MSTP_2GD	2GD

Name	Description
LPM_MSTP_SSI1	SSI1.
LPM_MSTP_SSI0	SSI0.
LPM_MSTP_SRC	SRC.
LPM_MSTP_SDH_MMC1	SDH_MMC1.
LPM_MSTP_SDH_MMC0	SDH_MMC0.
LPM_MSTP_DOC	DOC.
LPM_MSTP_ELC	ELC.
LPM_MSTP_TSIP	TSIP.
LPM_MSTP_AGT1	AGT1.
LPM_MSTP_AGT0	AGT0.
LPM_MSTP_GPT_CH7_0	GPT_CH7_0.
LPM_MSTP_GPT_CH13_8	GPT_CH13_8.
LPM_MSTP_PGI	PGI.
LPM_MSTP_S12AD_UNIT1	S12AD_UNIT1.
LPM_MSTP_S12AD_UNIT0	S12AD_UNIT0.
LPM_MSTP_D_A0	D_A0.
LPM_MSTP_TEMP_SENSE	TEMP_SENSE.
LPM_MSTP_COMP_OC5	COMP_OC5.
LPM_MSTP_COMP_OC4	COMP_OC4.
LPM_MSTP_COMP_OC3	COMP_OC3.
LPM_MSTP_COMP_OC2	COMP_OC2.
LPM_MSTP_COMP_OC1	COMP_OC1.
LPM_MSTP_COMP_OC0	COMP_OC0.
LPM_MSTP_COMP_LP	COMP_LP.

Name	Description
LPM_MSTP_COMP_RD	COMP_RD.
LPM_MSTP_COMP_OPAMP	COMP_OPAMP.

8.26.7 API Structures

8.26.7.1 lpm_cfg_t

lpm_cfg_t

Detailed description

User configuration structure, used in open function

Variables

- lpm_operating_power_t operating_power

  Operating power mode.
- lpm_subosc_t sub_oscillator

  Sub oscillator.
- lpm_code_flash_t code_flash

  Enable the code flash.

8.26.7.2 lpm_api_t

lpm_api_t

Detailed description

lpm driver structure. General lpm functions implemented at the HAL layer will follow this API.

8.26.7.3 init

```
ssp_err_t(* lpm_api_t::init) (lpm_cfg_t const *const p_cfg)
```

Detailed description

Open the LPM driver module Initialized the LPM block according to the passed in config structure

Table 1219:Parameters

Name	Direction	Description
p_cfg	in	Pointer to a configuration structure

Parameter p_cfg

Definition: lpm_cfg_t const *const p_cfg

User configuration structure, used in open function

- lpm_cfg_t::lpm_operating_power_t

  Operating power mode.

  Enumerated as:

  – LPM_OPERATING_POWER_HIGH_SPEED_MODE

  – LPM_OPERATING_POWER_MIDDLE_SPEED_MODE

  – LPM_OPERATING_POWER_LOW_VOLTAGE_MODE

  – LPM_OPERATING_POWER_LOW_SPEED_MODE

- lpm_cfg_t::lpm_subosc_t

  Sub oscillator.

  Enumerated as:

  – LPM_SUBOSC_OTHER

  – LPM_SUBOSC_SELECT

- lpm_cfg_t::lpm_code_flash_t

  Enable the code flash.

  Enumerated as:

  – LPM_CODE_FLASH_OPERATES

  – LPM_CODE_FLASH_STOPS

8.26.7.4 mstpcrSet

```
ssp_err_t(* lpm_api_t::mstpcrSet) (uint32_t mstpcra_value, uint32_t
mstpcrb_value, uint32_t mstpcrc_value, uint32_t mstpcrd_value)
```

Detailed description

Set the value of all the Module Stop Control Registers

Table 1220:Parameters

Name	Direction	Description
mstpcra_value	in	The value for MSTPCRA
mstpcrb_value	in	The value for MSTPCRB
mstpcrc_value	in	The value for MSTPCRC

Table 1220:Parameters (Continued)

Name	Direction	Description
mstpcrd_value	in	The value for MSTPCRD

Parameter mstpcra_value

`uint32_t`

Parameter mstpcrb_value

`uint32_t`

Parameter mstpcrc_value

`uint32_t`

Parameter mstpcrd_value

`uint32_t`

8.26.7.5 mstpcrGet

```
ssp_err_t(* lpm_api_t::mstpcrGet) (uint32_t *mstpcra_value, uint32_t
*mstpcrb_value, uint32_t *mstpcrc_value, uint32_t *mstpcrd_value)
```

Detailed description

Get the values of all the Module Stop Control Registers

Table 1221:Parameters

Name	Direction	Description
mstpcra_value	inout	The value from MSTPCRA
mstpcrb_value	inout	The value from MSTPCRB
mstpcrc_value	inout	The value from MSTPCRC
mstpcrd_value	inout	The value from MSTPCRD

Parameter mstpcra_value

`uint32_t`

Parameter mstpcrb_value

`uint32_t`

Parameter mstpcrc_value

`uint32_t`

Parameter mstpcrd_value

`uint32_t`

8.26.7.6 moduleStop

```
ssp_err_t(* lpm_api_t::moduleStop) (lpm_mstp_t module)
```

Detailed description

Stop a module

Table 1222:Parameters

Name	Direction	Description
module	in	The module to set the state of

Parameter module

Definition: lpm_mstp_tmodule

Module stop bits

8.26.7.7 moduleStart

```
ssp_err_t(* lpm_api_t::moduleStart) (lpm_mstp_t module)
```

Detailed description

Run a module

Table 1223:Parameters

Name	Direction	Description
module	in	The module to set the state of

Parameter module

Definition: lpm_mstp_tmodule

Module stop bits

8.26.7.8 operatingPowerModeSet

```
ssp_err_t(* lpm_api_t::operatingPowerModeSet) (lpm_operating_power_t power_mode,
lpm_subosc_t subosc)
```

Detailed description

Set the operating power mode

Table 1224:Parameters

Name	Direction	Description
power_mode	in	The power mode to set the chip to
subosc	in	Select the sub oscillator or other oscillator

Parameter power_mode

Definition: `lpm_operating_power_tpower_mode`

Operating power mode

Parameter subosc

8.26.7.9 snoozeEnable

```
ssp_err_t(* lpm_api_t::snoozeEnable) (lpm_snooze_rxd0_t rdx0_mode,
lpm_snooze_dtc_t dtc_mode, lpm_snooze_request_t requests, lpm_snooze_end_t
triggers)
```

Detailed description

Configure and enable snooze mode

Table 1225:Parameters

Name	Direction	Description
rdx0_mode	in	Enter Snooze mode on the falling edge of RXD0 (only in asynchronous mode)
dtc_mode	in	Use DTC and RAM while in Snooze Mode
requests	in	Specify the events that cause an entry into Snooze mode
triggers	in	Specify the events that cause an exit from snooze mode

Parameter rdx0_mode

Definition: `lpm_snooze_rxd0_trdx0_mode`

RXD0 Snooze Request Enable

Parameter dtc_mode

Definition: `lpm_snooze_dtc_tdtc_mode`

DTC Enable in Snooze Mode

Parameter requests

Definition: `lpm_snooze_request_t requests`

Snooze end control

Parameter triggers

Definition: `lpm_snooze_end_t triggers`

Snooze end control

8.26.7.10 snoozeDisable

```
ssp_err_t(* lpm_api_t::snoozeDisable) (void)
```

Detailed description

Disable snooze mode

Table 1226:Return values

Name	Description
SSP_SUCCESS	Snooze mode successfully disabled

Parameter SSP_SUCCESS

8.26.7.11 lowPowerCfg

```
ssp_err_t(* lpm_api_t::lowPowerCfg) (lpm_low_power_mode_t power_mode,
lpm_output_port_enable_t output_port_enable, lpm_power_supply_t power_supply,
lpm_io_port_t io_port_state)
```

Detailed description

Configure a low power mode

Table 1227:Parameters

Name	Direction	Description
low_power_mode	in	Which low power mode to enter
output_port_enable	in	Retain port output status in sleep or standby
power_supply	in	What remains powered in Deep Software Standby

Table 1227:Parameters (Continued)

Name	Direction	Description
io_port_state	in	I/O port state after Deep Software Standby mode

Parameter low_power_mode
Parameter output_port_enable
Parameter power_supply
Parameter io_port_state

Definition: lpm_io_port_tio_port_state

I/O port state after Deep Software Standby mode

8.26.7.12 wupenSet

```
ssp_err_t(* lpm_api_t::wupenSet) (uint32_t wupen_value)
```

Detailed description

Set the value of the Wake Up Interrupt Enable Register WUPEN

Table 1228:Parameters

Name	Direction	Description
wupen_value	in	The value for WUPEN

Parameter wupen_value

```
uint32_t
```

8.26.7.13 wupenGet

```
ssp_err_t(* lpm_api_t::wupenGet) (uint32_t *wupen_value)
```

Detailed description

Get the value of the Wake Up Interrupt Enable Register WUPEN

Table 1229:Parameters

Name	Direction	Description
wupen_value	inout	The value from WUPEN

Parameter wupen_value

```
uint32_t
```

8.26.7.14 deepStandbyCancelRequestEnable

```
ssp_err_t(* lpm_api_t::deepStandbyCancelRequestEnable) (lpm_deep_standby_t
pin_signal, lpm_cancel_request_edge_t rising_falling)
```

Detailed description

Enable a Deep Standby Cancel Request

Table 1230:Parameters

Name	Direction	Description
pin_signal	in	The pin or signal associated with deep standby mode
rising_falling	in	Which edge causes the cancel request (ignore with RTC interrupts)

Parameter pin_signal

Definition: `lpm_deep_standby_tpin_signal`

Deep Standby pins and signals

Parameter rising_falling

Definition: `lpm_cancel_request_edge_trising_falling`

Deep Standby Interrupt Edge

8.26.7.15 deepStandbyCancelRequestDisable

```
ssp_err_t(* lpm_api_t::deepStandbyCancelRequestDisable) (lpm_deep_standby_t
pin_signal)
```

Detailed description

Disable a Deep Standby Cancel Request

Table 1231:Parameters

Name	Direction	Description
pin_signal	in	The pin or signal associated with deep standby mode

Parameter pin_signal

Definition: `lpm_deep_standby_tpin_signal`

Deep Standby pins and signals

8.26.7.16 lowPowerModeEnter

```
ssp_err_t(* lpm_api_t::lowPowerModeEnter) (void)
```

Detailed description

Enter low power mode (sleep/standby/deep standby) using WFI macro. Function will return after waking from low power mode.

8.26.7.17 versionGet

```
ssp_err_t(* lpm_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Get the driver version based on compile time macros.

Table 1232:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_INVALID_PTR	p_version is NULL.

Parameter SSP_SUCCESS
Parameter SSP_ERR_INVALID_PTR

8.26.7.18 lpm_instance_t

lpm_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- lpm_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- lpm_api_t const * p_api

  Pointer to the API structure for this instance.

8.27 Low Power Modes V2 Interface

Interface for accessing low power modes.

8.27.1 Summary

This section defines the API for the LPMV2 (Low Power Mode) Driver. The LPMV2 Driver provides functions for controlling power consumption by configuring and transitioning to a low power mode. The LPMV2 driver supports configuration of MCU low power modes using the LPMV2 hardware peripheral. The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

NOTE: Not all low power modes are available on all MCUs.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

LPMV2 Interface description: HAL LPMV2 Interface

8.27.2 Interface API

lpmv2_api_t

Function name	Description
.init	Initialization function
.lowPowerCfg	Configure a low power mode.
.lowPowerModeEnter	Enter low power mode (sleep/standby/deep standby) using WFI macro. Function will return after waking from low power mode.
.versionGet	Get the driver version based on compile time macros.

8.27.3 Data structures

- lpmv2_cfg_t
- lpmv2_instance_t

8.27.4 Enumerations

- lpmv2_low_power_mode_t

8.27.5 Defines

- #define LPMV2_API_VERSION_MAJOR

  Initial value: `(2U)`

  Register definitions, common services and error codes.
- #define LPMV2_API_VERSION_MINOR

  Initial value: `(3U)`

8.27.6 API Data

8.27.6.1 lpmv2_low_power_mode_t

`lpmv2_low_power_mode_t`

Detailed description

Low power modes

Enumerated values

Name	Description
LPMV2_LOW_POWER_MODE_SLEEP	Sleep mode.
LPMV2_LOW_POWER_MODE_STANDBY	Software Standby mode.
LPMV2_LOW_POWER_MODE_STANDBY_SNOOZE	Software Standby mode with Snooze mode enabled.
LPMV2_LOW_POWER_MODE_DEEP	Deep Software Standby mode.

8.27.7 API Structures

8.27.7.1 lpmv2_cfg_t

lpmv2_cfg_t

Detailed description

User configuration structure, used in open function

Variables

- lpmv2_low_power_mode_t low_power_mode

  Low Power Mode
- void const * p_extend

  MCU Specific configuration

8.27.7.2 lpmv2_api_t

lpmv2_api_t

Detailed description

lpmv2 driver structure. General lpmv2 functions implemented at the HAL layer will follow this API.

8.27.7.3 init

```
ssp_err_t(* lpmv2_api_t::init) (void)
```

Detailed description

Initialization function Implemented as

- R_LPMV2_Init

8.27.7.4 lowPowerCfg

```
ssp_err_t(* lpmv2_api_t::lowPowerCfg) (lpmv2_cfg_t const *const p_cfg)
```

Detailed description

Configure a low power mode. Implemented as

- R_LPMV2_LowPowerConfigure

Table 1233:Parameters

Name	Direction	Description
p_cfg	in	Pointer to configuration structure. All elements of this structure must be set by user.

Parameter p_cfg

Definition: `lpmv2_cfg_t const *const p_cfg`

User configuration structure, used in open function

- `lpmv2_cfg_t`::`lpmv2_low_power_mode_t`

  Low Power Mode

  Enumerated as:

  – LPMV2_LOW_POWER_MODE_SLEEP

  – LPMV2_LOW_POWER_MODE_STANDBY

  – LPMV2_LOW_POWER_MODE_STANDBY_SNOOZE

  – LPMV2_LOW_POWER_MODE_DEEP

- lpmv2_cfg_t::p_extend

  MCU Specific configuration

8.27.7.5 lowPowerModeEnter

```
ssp_err_t(* lpmv2_api_t::lowPowerModeEnter) (void)
```

Detailed description

Enter low power mode (sleep/standby/deep standby) using WFI macro. Function will return after waking from low power mode. Implemented as

- R_LPMV2_LowPowerModeEnter

8.27.7.6 versionGet

```
ssp_err_t(* lpmv2_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Get the driver version based on compile time macros. Implemented as

- R_LPMV2_VersionGet

Table 1234:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.27.7.7 lpmv2_instance_t

lpmv2_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- lpmv2_cfg_t const *const p_cfg

  Pointer to the configuration structure for this instance.

- lpmv2_api_t const *const p_api

  Pointer to the API structure for this instance.

8.28 Low Voltage Detection Interface

This section defines the API for the LVD (Low Voltage Detection) Driver.

The LVD driver provides functions for configuring the LVD hardware peripheral.

The process of configuring and enabling a Low Voltage Detection monitor has very specific timing constraints and register write ordering. Because of these constraints, the entire process of configuring and enabling a voltage monitor is most effectively performed by a single function. The API function configure performs configuration and enables the monitor in order to properly enforce the timing and register write ordering constraints.

The LVD driver configures all of the settings of the available configurable LVD monitors.

The settings include:

- voltage_threshold: Determines the voltage detection threshold (i.e. 2.99 Volts).
- sample_clock_divisor: Determines the sample clock rate of the digital filter, based on division of the LOCO clock. Also disables or enables the digital filter if available on the MCU.
- detection_response: Determines which event will occur, reset, interrupt, non-maskable interrupt, or no response, when the voltage threshold is crossed
- voltage_slope: Choose either rising or falling voltage as the trigger for a voltage detection interrupt.
- negation_delay: Determine whether timing of the negation of the voltage event is based upon the reset event or based on the voltage event itself.
- p_callback: Address of user defined function to be called when the voltage event interrupt occurs.

NOTE: Low Voltage Monitor 0 (LVD0) is not configurable at runtime but can be configured by changing the OFS1 register value on the BSP Properties tab of the Synergy Project Configurator in the e^2 studio ISDE.

NOTE: Digital filter is not to be used with standby modes. If software standby or deep standby mode is to be used, the digital filter should be disabled.

For details about the implementation of the driver functions see section LVD.

8.28.1 Interface API

lvd_api_t

Function name	Description
.open	Initializes a low voltage detection driver according to the passed in configuration structure. Enables an LVD peripheral based on configuration structure.

Function name	Description
.statusGet	Get the current state of the monitor, (threshold crossing detected, voltage currently within range) Can be used to poll the state of the LVD monitor at any time. Must be used if the peripheral was initialized with lvd_response_t set to LVD_RESPONSE_NONE.
.statusClear	Clears the latched status of the monitor. Must be used if the peripheral was initialized with lvd_response_t set to LVD_RESPONSE_NONE.
.close	Disables the LVD peripheral. Closes the driver instance.
.versionGet	Returns the LVD driver version based on compile time macros.

8.28.2 Data structures

- lvd_status_t
- lvd_callback_args_t
- lvd_cfg_t
- lvd_instance_t

8.28.3 Enumerations

- lvd_threshold_t
- lvd_response_t
- lvd_voltage_slope_t
- lvd_threshold_crossing_t
- lvd_current_state_t

8.28.4 Typedefs

- lvd_ctrl_t

8.28.5 Defines

- #define LVD_API_VERSION_MAJOR

  Initial value: `(1U)`

  Register definitions, common services and error codes.

- #define LVD_API_VERSION_MINOR

  Initial value: `(1U)`

8.28.6 API Data

8.28.6.1 lvd_threshold_t

`lvd_threshold_t`

Detailed description

Voltage detection level The thresholds supported by each MCU is in the MCU User's Manual as well as in the r_lvd module description on the threads tab of the Synergy project.

Enumerated values

Name	Description
LVD_THRESHOLD_MONITOR_1_LEVEL_0	4.29V (Vdet1_0)
LVD_THRESHOLD_MONITOR_1_LEVEL_1	4.14V (Vdet1_1)
LVD_THRESHOLD_MONITOR_1_LEVEL_2	4.02V (Vdet1_2)
LVD_THRESHOLD_MONITOR_1_LEVEL_3	3.84V (Vdet1_3)
LVD_THRESHOLD_MONITOR_1_LEVEL_4	3.10V (Vdet1_4)
LVD_THRESHOLD_MONITOR_1_LEVEL_5	3.00V (Vdet1_5)
LVD_THRESHOLD_MONITOR_1_LEVEL_6	2.90V (Vdet1_6)
LVD_THRESHOLD_MONITOR_1_LEVEL_7	2.79V (Vdet1_7)
LVD_THRESHOLD_MONITOR_1_LEVEL_8	2.68V (Vdet1_8)
LVD_THRESHOLD_MONITOR_1_LEVEL_9	2.58V (Vdet1_9)
LVD_THRESHOLD_MONITOR_1_LEVEL_A	2.48V (Vdet1_A)
LVD_THRESHOLD_MONITOR_1_LEVEL_B	2.20V (Vdet1_B)
LVD_THRESHOLD_MONITOR_1_LEVEL_C	1.96V (Vdet1_C)
LVD_THRESHOLD_MONITOR_1_LEVEL_D	1.86V (Vdet1_D)
LVD_THRESHOLD_MONITOR_1_LEVEL_E	1.75V (Vdet1_E)
LVD_THRESHOLD_MONITOR_1_LEVEL_F	1.65V (Vdet1_F)

Name	Description
LVD_THRESHOLD_MONITOR_1_LEVEL_11	2.99V (Vdet1_11)
LVD_THRESHOLD_MONITOR_1_LEVEL_12	2.92V (Vdet1_12)
LVD_THRESHOLD_MONITOR_1_LEVEL_13	2.85V (Vdet1_13)
LVD_THRESHOLD_MONITOR_2_LEVEL_0	4.29V (Vdet2_0)
LVD_THRESHOLD_MONITOR_2_LEVEL_1	4.14V (Vdet2_1)
LVD_THRESHOLD_MONITOR_2_LEVEL_2	4.02V (Vdet2_2)
LVD_THRESHOLD_MONITOR_2_LEVEL_3	3.84V (Vdet2_3)
LVD_THRESHOLD_MONITOR_2_LEVEL_5	2.99V (Vdet2_5)
LVD_THRESHOLD_MONITOR_2_LEVEL_6	2.92V (Vdet2_6)
LVD_THRESHOLD_MONITOR_2_LEVEL_7	2.85V (Vdet2_7)

8.28.6.2 lvd_response_t

`lvd_response_t`

Detailed description

Response types to a threshold crossing event, interrupt, reset, NMI...

Enumerated values

Name	Description
LVD_RESPONSE_NMI	Non-maskable interrupt.
LVD_RESPONSE_INTERRUPT	Maskable interrupt.
LVD_RESPONSE_RESET	Reset.
LVD_RESPONSE_NONE	No response, status must be requested via statusGet function.

8.28.6.3 lvd_voltage_slope_t

`lvd_voltage_slope_t`

Detailed description

Voltage slope, rising, falling, or both

Enumerated values

Name	Description
LVD_VOLTAGE_SLOPE_RISING	When VCC >= Vdet2 (rise) is detected.
LVD_VOLTAGE_SLOPE_FALLING	When VCC < Vdet2 (drop) is detected.
LVD_VOLTAGE_SLOPE_BOTH	When drop and rise are detected.

8.28.6.4 lvd_threshold_crossing_t

```
lvd_threshold_crossing_t
```

Detailed description

Threshold crossing detection (latched)

Enumerated values

Name	Description
LVD_THRESHOLD_CROSSING_NOT_DETECTED	Threshold crossing has not been detected.
LVD_THRESHOLD_CROSSING_DETECTED	Threshold crossing has been detected.

8.28.6.5 lvd_current_state_t

```
lvd_current_state_t
```

Detailed description

Instantaneous status of VCC (above or below threshold)

Enumerated values

Name	Description
LVD_CURRENT_STATE_BELOW_THRESHOLD	VCC < threshold.
LVD_CURRENT_STATE_ABOVE_THRESHOLD	VCC >= threshold or monitor is disabled.

8.28.6.6 lvd_ctrl_t

```
typedef void lvd_ctrl_t
```

Detailed description

LVD control block. Allocate an instance specific control block to pass into the LVD API calls. Implemented as

- lvd_instance_ctrl_t

8.28.7 API Structures

8.28.7.1 lvd_status_t

lvd_status_t

Detailed description

Voltage monitor status structure, used with statusGet function and p_callback to provide current state of the monitor, (threshold crossing detected, vcc currently within range).

Variables

- lvd_threshold_crossing_t crossing_detected

  Threshold crossing detection (latched)

- lvd_current_state_t current_state

  Instantaneous status of monitored voltage (above or below threshold)

8.28.7.2 lvd_callback_args_t

lvd_callback_args_t

Detailed description

LVD callback parameter definition

Variables

- uint32_t monitor_number

  Monitor number.

- lvd_status_t status

  Status of monitor.

- void const * p_context

  Placeholder for user data.

8.28.7.3 lvd_cfg_t

lvd_cfg_t

Detailed description

LVD configuration structure

Variables

- const uint32_t monitor_number

  Monitor number, 1, 2, ...

- lvd_threshold_t voltage_threshold

  Threshold for out of range voltage detection
- lvd_response_t detection_response

  Response on detecting a threshold crossing
- lvd_voltage_slope_t voltage_slope

  Rising or falling voltage is to be detected
- uint8_t monitor_ipl

  Interrupt priority level.
- void(* p_callback)(lvd_callback_args_t *p_args)

  User function to be called from interrupt
- void const * p_context

  Placeholder for user data. Passed to the user callback in
- void const * p_extend

  Extension parameter for hardware specific settings

8.28.7.4 lvd_api_t

lvd_api_t

Detailed description

LVD driver API structure. LVD driver functions implemented at the HAL layer will adhere to this API.

8.28.7.5 open

```
ssp_err_t(* lvd_api_t::open) (lvd_ctrl_t *const p_ctrl, lvd_cfg_t const *const
p_cfg)
```

Detailed description

Initializes a low voltage detection driver according to the passed in configuration structure. Enables an LVD peripheral based on configuration structure. Implemented as

- R_LVD_Open

Table 1235:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to monitor control structure for the driver instance
p_cfg	in	Pointer to the configuration structure for the driver instance

Parameter p_ctrl

Definition: `lvd_ctrl_t*const p_ctrl`

LVD control block. Allocate an instance specific control block to pass into the LVD API calls. Implemented aslvd_instance_ctrl_t

Parameter p_cfg

Definition: `lvd_cfg_t const *const p_cfg`

LVD configuration structure

- `lvd_cfg_t`::`monitor_number`

  Monitor number, 1, 2, ...

- `lvd_cfg_t`::`lvd_threshold_t`

  Threshold for out of range voltage detection

  Enumerated as:

  – LVD_THRESHOLD_MONITOR_1_LEVEL_0
  – LVD_THRESHOLD_MONITOR_1_LEVEL_1
  – LVD_THRESHOLD_MONITOR_1_LEVEL_2
  – LVD_THRESHOLD_MONITOR_1_LEVEL_3
  – LVD_THRESHOLD_MONITOR_1_LEVEL_4
  – LVD_THRESHOLD_MONITOR_1_LEVEL_5
  – LVD_THRESHOLD_MONITOR_1_LEVEL_6
  – LVD_THRESHOLD_MONITOR_1_LEVEL_7
  – LVD_THRESHOLD_MONITOR_1_LEVEL_8
  – LVD_THRESHOLD_MONITOR_1_LEVEL_9
  – LVD_THRESHOLD_MONITOR_1_LEVEL_A
  – LVD_THRESHOLD_MONITOR_1_LEVEL_B
  – LVD_THRESHOLD_MONITOR_1_LEVEL_C
  – LVD_THRESHOLD_MONITOR_1_LEVEL_D
  – LVD_THRESHOLD_MONITOR_1_LEVEL_E
  – LVD_THRESHOLD_MONITOR_1_LEVEL_F
  – LVD_THRESHOLD_MONITOR_1_LEVEL_11
  – LVD_THRESHOLD_MONITOR_1_LEVEL_12
  – LVD_THRESHOLD_MONITOR_1_LEVEL_13

  - LVD_THRESHOLD_MONITOR_2_LEVEL_0
  - LVD_THRESHOLD_MONITOR_2_LEVEL_1
  - LVD_THRESHOLD_MONITOR_2_LEVEL_2
  - LVD_THRESHOLD_MONITOR_2_LEVEL_3
  - LVD_THRESHOLD_MONITOR_2_LEVEL_5
  - LVD_THRESHOLD_MONITOR_2_LEVEL_6
  - LVD_THRESHOLD_MONITOR_2_LEVEL_7
- lvd_cfg_t::lvd_response_t

  Response on detecting a threshold crossing

  Enumerated as:
  - LVD_RESPONSE_NMI
  - LVD_RESPONSE_INTERRUPT
  - LVD_RESPONSE_RESET
  - LVD_RESPONSE_NONE
- lvd_cfg_t::lvd_voltage_slope_t

  Rising or falling voltage is to be detected

  Enumerated as:
  - LVD_VOLTAGE_SLOPE_RISING
  - LVD_VOLTAGE_SLOPE_FALLING
  - LVD_VOLTAGE_SLOPE_BOTH
- lvd_cfg_t::monitor_ipl

  Interrupt priority level.
- lvd_cfg_t::p_callback

  User function to be called from interrupt
- lvd_cfg_t::p_context

  Placeholder for user data. Passed to the user callback in
- lvd_cfg_t::p_extend

  Extension parameter for hardware specific settings

8.28.7.6 statusGet

```
ssp_err_t(* lvd_api_t::statusGet) (lvd_ctrl_t *const p_ctrl, lvd_status_t
*p_lvd_status)
```

Detailed description

Get the current state of the monitor, (threshold crossing detected, voltage currently within range) Can be used to poll the state of the LVD monitor at any time. Must be used if the peripheral was initialized with lvd_response_t set to LVD_RESPONSE_NONE. Implemented as

- R_LVD_StatusGet

Table 1236:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control structure for the driver instance
p_lvd_status	inout	Pointer to an lvd_status_t instance

Parameter p_ctrl

Definition: `lvd_ctrl_t*const p_ctrl`

LVD control block. Allocate an instance specific control block to pass into the LVD API calls. Implemented aslvd_instance_ctrl_t

Parameter p_lvd_status

Definition: `lvd_status_t*p_lvd_status`

Voltage monitor status structure, used with statusGet function and p_callback to provide current state of the monitor, (threshold crossing detected, vcc currently within range).

- `lvd_status_t::crossing_detected`

  Threshold crossing detection (latched)

- `lvd_status_t::current_state`

  Instantaneous status of monitored voltage (above or below threshold)

8.28.7.7 statusClear

```
ssp_err_t(* lvd_api_t::statusClear) (lvd_ctrl_t *const p_ctrl)
```

Detailed description

Clears the latched status of the monitor. Must be used if the peripheral was initialized with lvd_response_t set to LVD_RESPONSE_NONE. Implemented as

- R_LVD_StatusClear

Table 1237:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control structure for the driver instance

Parameter p_ctrl

Definition: `lvd_ctrl_t*const p_ctrl`

LVD control block. Allocate an instance specific control block to pass into the LVD API calls. Implemented aslvd_instance_ctrl_t

8.28.7.8 close

```
ssp_err_t(* lvd_api_t::close) (lvd_ctrl_t *const p_ctrl)
```

Detailed description

Disables the LVD peripheral. Closes the driver instance. Implemented as

- R_LVD_Close

Table 1238:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control structure for the driver instance

Parameter p_ctrl

Definition: `lvd_ctrl_t*const p_ctrl`

LVD control block. Allocate an instance specific control block to pass into the LVD API calls. Implemented aslvd_instance_ctrl_t

8.28.7.9 versionGet

```
ssp_err_t(* lvd_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Returns the LVD driver version based on compile time macros. Implemented as

- R_LVD_VersionGet

Table 1239:Parameters

Name	Direction	Description
p_version	inout	Pointer to version structure

Parameter p_version

8.28.7.10 lvd_instance_t

lvd_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- lvd_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- lvd_cfg_t const * p_cfg

  Pointer to the configuration structure for this interface instance.
- lvd_api_t const * p_api

  Pointer to the API structure for this interface instance.

8.29 OPAMP Interface

Interface for Operational Amplifiers.

8.29.1 Summary

The OPAMP interface provides standard operational amplifier functionality, including starting and stopping the amplifier.

Implemented by: Operational Amplifier (OPAMP)

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

OPAMP Interface description: OPAMP Driver

8.29.2 Interface API

opamp_api_t

Function name	Description
.open	Initialize the operational amplifier.
.start	Start the op-amp(s).
.stop	Stop the op-amp(s).
.trim	Trim the op-amp(s). Not supported on all MCUs. See implementation for procedure details.
.infoGet	Provide information such as the recommended minimum stabilization wait time.
.statusGet	Provide status of each op-amp channel.
.close	Close the specified OPAMP unit by ending any scan in progress, disabling interrupts, and removing power to the specified A/D unit.
.versionGet	Retrieve the API version.

8.29.3 Data structures

- opamp_trim_args_t
- opamp_info_t
- opamp_status_t
- opamp_cfg_t
- opamp_instance_t

8.29.4 Enumerations

- opamp_trim_cmd_t
- opamp_trim_input_t

8.29.5 Typedefs

- opamp_ctrl_t

8.29.6 Defines

- #define OPAMP_API_VERSION_MAJOR

  Initial value: `(1U)`

  Includes board and MCU related header files. Version Number of API.

- #define OPAMP_API_VERSION_MINOR

  Initial value: `(0U)`

8.29.7 API Data

8.29.7.1 opamp_trim_cmd_t

`opamp_trim_cmd_t`

Detailed description

Trim command.

Enumerated values

Name	Description
OPAMP_TRIM_CMD_START	Initialize trim state machine.
OPAMP_TRIM_CMD_NEXT_STEP	Move to next step in state machine.
OPAMP_TRIM_CMD_CLEAR_BIT	Clear trim bit.

8.29.7.2 opamp_trim_input_t

`opamp_trim_input_t`

Detailed description

Trim input.

Enumerated values

Name	Description
OPAMP_TRIM_INPUT_PCH	Trim non-inverting (+) input.
OPAMP_TRIM_INPUT_NCH	Trim inverting (-) input.

8.29.7.3 opamp_ctrl_t

```
typedef void opamp_ctrl_t
```

Detailed description

OPAMP control block. Allocate using driver instance control structure from driver instance header file.

8.29.8 API Structures

8.29.8.1 opamp_trim_args_t

opamp_trim_args_t

Detailed description

OPAMP trim arguments.

Variables

- uint8_t channel

  Channel.

- opamp_trim_input_t input

  Which input of the channel above.

8.29.8.2 opamp_info_t

opamp_info_t

Detailed description

OPAMP information.

Variables

- uint32_t min_stabilization_wait_us

  Minimum stabilization wait time in microseconds.

8.29.8.3 opamp_status_t

opamp_status_t

Detailed description

OPAMP status.

Variables

- uint32_t operating_channel_mask

  Bitmask of channels currently operating.

8.29.8.4 opamp_cfg_t

opamp_cfg_t

Detailed description

OPAMP general configuration.

Variables

- void const * p_extend

  Extension parameter for hardware specific settings.

8.29.8.5 opamp_api_t

opamp_api_t

Detailed description

OPAMP functions implemented at the HAL layer will follow this API.

8.29.8.6 open

```
ssp_err_t(* opamp_api_t::open) (opamp_ctrl_t *const p_ctrl, opamp_cfg_t const
*const p_cfg)
```

Detailed description

Initialize the operational amplifier. Implemented as

- R_OPAMP_Open

Table 1240:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to instance control block
p_cfg	in	Pointer to configuration

Parameter p_ctrl
Parameter p_cfg

8.29.8.7 start

```
ssp_err_t(* opamp_api_t::start) (opamp_ctrl_t *const p_ctrl, uint32_t const
channel_mask)
```

Detailed description

Start the op-amp(s). Implemented as

- R_OPAMP_Start

Table 1241:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to instance control block
channel_mask	in	Bitmask of channels to start

Parameter p_ctrl
Parameter channel_mask

`uint32_t`

8.29.8.8 stop

```
ssp_err_t(* opamp_api_t::stop) (opamp_ctrl_t *const p_ctrl, uint32_t const
channel_mask)
```

Detailed description

Stop the op-amp(s). Implemented as

- R_OPAMP_Stop

Table 1242:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to instance control block
channel_mask	in	Bitmask of channels to stop

Parameter p_ctrl
Parameter channel_mask

`uint32_t`

8.29.8.9 trim

```
ssp_err_t(* opamp_api_t::trim) (opamp_ctrl_t *const p_ctrl, opamp_trim_cmd_t
const cmd, opamp_trim_args_t const *const p_args)
```

Detailed description

Trim the op-amp(s). Not supported on all MCUs. See implementation for procedure details. Implemented as

- R_OPAMP_Trim

Table 1243:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to instance control block
cmd	in	Trim command
p_args	in	Pointer to arguments for the command

Parameter p_ctrl
Parameter cmd
Parameter p_args

Definition: `opamp_trim_args_tconst *const p_args`

OPAMP trim arguments.

- `opamp_trim_args_t::channel`

  Channel.

- `opamp_trim_args_t::input`

  Which input of the channel above.

8.29.8.10 infoGet

```
ssp_err_t(* opamp_api_t::infoGet) (opamp_ctrl_t *const p_ctrl, opamp_info_t
*const p_info)
```

Detailed description

Provide information such as the recommended minimum stabilization wait time. Implemented as

- R_OPAMP_InfoGet

Table 1244:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to instance control block
p_info	out	OPAMP information stored here

Parameter p_ctrl
Parameter p_info

8.29.8.11 statusGet

```
ssp_err_t(* opamp_api_t::statusGet) (opamp_ctrl_t *const p_ctrl, opamp_status_t
*const p_status)
```

Detailed description

Provide status of each op-amp channel. Implemented as

- R_OPAMP_StatusGet

Table 1245:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to instance control block
p_status	out	Status stored here

Parameter p_ctrl
Parameter p_status

8.29.8.12 close

```
ssp_err_t(* opamp_api_t::close) (opamp_ctrl_t *const p_ctrl)
```

Detailed description

Close the specified OPAMP unit by ending any scan in progress, disabling interrupts, and removing power to the specified A/D unit. Implemented as

- R_OPAMP_Close

Table 1246:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to instance control block

Parameter p_ctrl

8.29.8.13 versionGet

```
ssp_err_t(* opamp_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Retrieve the API version. Implemented as

- R_OPAMP_VersionGet

NOTE: This function retrieves the API version.

Table 1247:Parameters

Name	Direction	Description
p_version	in	Pointer to version structure

Parameter p_version

8.29.8.14 opamp_instance_t

opamp_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- opamp_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- opamp_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- opamp_api_t const * p_api

  Pointer to the API structure for this instance.

8.30 PDC Interface

Interface for PDC functions.

8.30.1 Summary

The PDC interface provides the functionality for capturing an image from a camera. When a capture is complete a transfer complete interrupt is triggered.

8.30.2 Known Implementations

PDCRelated SSP architecture topics:

- What is an SSP Interface? SSP Interfaces
- What is a SSP Layer? SSP Predefined Layers

- How to use SSP Interfaces and Modules? Using SSP Modules

8.30.3 Interface API

pdc_api_t

Function name	Description
.open	Initial configuration.
.close	Closes the driver and allows reconfiguration. May reduce power consumption.
.captureStart	Start a capture.
.stateGet	Get the state of the VSYNC and HSYNC pins.
.versionGet	Return the version of the driver.

8.30.4 Data structures

- pdc_state_t
- pdc_callback_args_t
- pdc_cfg_t
- pdc_instance_t

8.30.5 Enumerations

- pdc_clock_division_t
- pdc_endian_t
- pdc_hsync_polarity_t
- pdc_vsync_polarity_t
- pdc_event_t
- pdc_vsync_state_t
- pdc_hsync_state_t

8.30.6 Typedefs

- pdc_ctrl_t

8.30.7 Defines

- #define PDC_API_VERSION_MAJOR

  Initial value: `(1U)`

  Register definitions, common services and error codes.
- #define PDC_API_VERSION_MINOR

  Initial value: `(2U)`

8.30.8 API Data

8.30.8.1 pdc_clock_division_t

`pdc_clock_division_t`

Detailed description

Clock divider applied to PDC clock to provide PCKO output frequency

Enumerated values

Name	Description
PDC_CLOCK_DIVISION_2	CLK / 2.
PDC_CLOCK_DIVISION_4	CLK / 4.
PDC_CLOCK_DIVISION_6	CLK / 6.
PDC_CLOCK_DIVISION_8	CLK / 8.
PDC_CLOCK_DIVISION_10	CLK / 10.
PDC_CLOCK_DIVISION_12	CLK / 12.
PDC_CLOCK_DIVISION_14	CLK / 14.
PDC_CLOCK_DIVISION_16	CLK / 16.

8.30.8.2 pdc_endian_t

`pdc_endian_t`

Detailed description

Endian of captured data

Enumerated values

Name	Description
PDC_ENDIAN_LITTLE	Data is in little endian format.
PDC_ENDIAN_BIG	Data is in big endian format.

8.30.8.3 pdc_hsync_polarity_t

`pdc_hsync_polarity_t`

Detailed description

Polarity of input HSYNC signal

Enumerated values

Name	Description
PDC_HSYNC_POLARITY_HIGH	HSYNC signal is active high.
PDC_HSYNC_POLARITY_LOW	HSYNC signal is active low.

8.30.8.4 pdc_vsync_polarity_t

`pdc_vsync_polarity_t`

Detailed description

Polarity of input VSYNC signal

Enumerated values

Name	Description
PDC_VSYNC_POLARITY_HIGH	VSYNC signal is active high.
PDC_VSYNC_POLARITY_LOW	VSYNC signal is active low.

8.30.8.5 pdc_event_t

`pdc_event_t`

Detailed description

PDC events

Enumerated values

Name	Description
PDC_EVENT_TRANSFER_COMPLETE	Complete frame transferred by DMAC/DTC.
PDC_EVENT_RX_DATA_READY	Receive data ready interrupt.
PDC_EVENT_FRAME_END	Frame end interrupt.
PDC_EVENT_ERR_OVERRUN	Overrun interrupt.
PDC_EVENT_ERR_UNDERRUN	Underrun interrupt.
PDC_EVENT_ERR_V_SET	Vertical line setting error interrupt.
PDC_EVENT_ERR_H_SET	Horizontal byte number setting error interrupt.

8.30.8.6 pdc_vsync_state_t

`pdc_vsync_state_t`

Detailed description

VSYNC signal state

Enumerated values

Name	Description
PDC_VSYNC_STATE_LOW	VSYNC signal is low.
PDC_VSYNC_STATE_HIGH	VSYNC signal is high.

8.30.8.7 pdc_hsync_state_t

`pdc_hsync_state_t`

Detailed description

HSYNC signal state

Enumerated values

Name	Description
PDC_HSYNC_STATE_LOW	HSYNC signal is low.
PDC_HSYNC_STATE_HIGH	HSYNC signal is high.

8.30.8.8 pdc_ctrl_t

```
typedef void pdc_ctrl_t
```

Detailed description

PDC control block. Allocate an instance specific control block to pass into the PDC API calls. Implemented as

- pdc_instance_ctrl_t

8.30.9 API Structures

8.30.9.1 pdc_state_t

pdc_state_t

Detailed description

PDC VSYNC/HSYNC state

Variables

- pdc_vsync_state_t vsync

  VSYNC signal state.
- pdc_hsync_state_t hsync

  HSYNC signal state.

8.30.9.2 pdc_callback_args_t

pdc_callback_args_t

Detailed description

Callback function parameter data

Variables

- pdc_event_t event

  Event causing the callback.
- uint8_t * p_buffer

  Pointer to buffer containing the captured data.
- void const * p_context

  Placeholder for user data. Set in pdc_api_t::open function in pdc_cfg_t.

8.30.9.3 pdc_cfg_t

pdc_cfg_t

Detailed description

PDC configuration parameters.

Variables

- uint16_t x_capture_start_pixel

  Horizontal position to start capture.
- uint16_t x_capture_pixels

  Number of horizontal pixels to capture.
- uint16_t y_capture_start_pixel

  Vertical position to start capture.
- uint16_t y_capture_pixels

  Number of vertical lines/pixels to capture.
- pdc_clock_division_t clock_division

  Clock divider.
- pdc_endian_t endian

  Endian of capture data.
- pdc_hsync_polarity_t hsync_polarity

  Polarity of HSYNC input.
- pdc_vsync_polarity_t vsync_polarity

  Polarity of VSYNC input.
- uint8_t * p_buffer

  Pointer to buffer to write image into.
- uint8_t bytes_per_pixel

  Number of bytes per pixel.
- uint8_t frame_end_ipl

  Frame end interrupt priority.
- uint8_t irq_ipl

  PDC interrupt priority.
- transfer_instance_t const *const p_lower_lvl_transfer

  Pointer to the transfer instance the PDC should use.
- void(* p_callback)(pdc_callback_args_t *p_args)

  Callback provided when a PDC transfer ISR occurs.
- void const * p_context

  Placeholder for user data. Passed to the user callback in pdc_callback_args_t.
- void const * p_extend

8.30.9.4 pdc_api_t

pdc_api_t

Detailed description

PDC functions implemented at the HAL layer will follow this API.

8.30.9.5 open

```
ssp_err_t(* pdc_api_t::open) (pdc_ctrl_t *const p_ctrl, pdc_cfg_t const *const
p_cfg)
```

Detailed description

Initial configuration. Implemented as

- R_PDC_Open

NOTE: To reconfigure after calling this function, call close first.

Table 1248:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure.
p_cfg	in	Pointer to pin configuration structure.

Parameter p_ctrl

Definition: `pdc_ctrl_t*const p_ctrl`

PDC control block. Allocate an instance specific control block to pass into the PDC API calls. Implemented aspdc_instance_ctrl_t

Parameter p_cfg

Definition: `pdc_cfg_t const *const p_cfg`

PDC configuration parameters.

- `pdc_cfg_t::x_capture_start_pixel`

  Horizontal position to start capture.

- `pdc_cfg_t::x_capture_pixels`

  Number of horizontal pixels to capture.

- `pdc_cfg_t::y_capture_start_pixel`

  Vertical position to start capture.

- pdc_cfg_t::y_capture_pixels

  Number of vertical lines/pixels to capture.
- pdc_cfg_t::pdc_clock_division_t

  Clock divider.

  Enumerated as:

  – PDC_CLOCK_DIVISION_2

  – PDC_CLOCK_DIVISION_4

  – PDC_CLOCK_DIVISION_6

  – PDC_CLOCK_DIVISION_8

  – PDC_CLOCK_DIVISION_10

  – PDC_CLOCK_DIVISION_12

  – PDC_CLOCK_DIVISION_14

  – PDC_CLOCK_DIVISION_16
- pdc_cfg_t::pdc_endian_t

  Endian of capture data.

  Enumerated as:

  – PDC_ENDIAN_LITTLE

  – PDC_ENDIAN_BIG
- pdc_cfg_t::pdc_hsync_polarity_t

  Polarity of HSYNC input.

  Enumerated as:

  – PDC_HSYNC_POLARITY_HIGH

  – PDC_HSYNC_POLARITY_LOW
- pdc_cfg_t::pdc_vsync_polarity_t

  Polarity of VSYNC input.

  Enumerated as:

  – PDC_VSYNC_POLARITY_HIGH

  – PDC_VSYNC_POLARITY_LOW
- pdc_cfg_t::p_buffer

  Pointer to buffer to write image into.

- pdc_cfg_t::bytes_per_pixel

  Number of bytes per pixel.
- pdc_cfg_t::frame_end_ipl

  Frame end interrupt priority.
- pdc_cfg_t::irq_ipl

  PDC interrupt priority.
- pdc_cfg_t::transfer_instance_t

  Pointer to the transfer instance the PDC should use.
- pdc_cfg_t::p_callback

  Callback provided when a PDC transfer ISR occurs.
- pdc_cfg_t::p_context

  Placeholder for user data. Passed to the user callback in pdc_callback_args_t.
- pdc_cfg_t::p_extend

8.30.9.6 close

```
ssp_err_t(* pdc_api_t::close) (pdc_ctrl_t *const p_ctrl)
```

Detailed description

Closes the driver and allows reconfiguration. May reduce power consumption. Implemented as

- R_PDC_Close

Table 1249:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure.

Parameter p_ctrl

Definition: pdc_ctrl_t*const p_ctrl

PDC control block. Allocate an instance specific control block to pass into the PDC API calls. Implemented aspdc_instance_ctrl_t

8.30.9.7 captureStart

```
ssp_err_t(* pdc_api_t::captureStart) (pdc_ctrl_t *const p_ctrl, uint8_t *const
p_buffer)
```

Detailed description

Start a capture. Implemented as

- R_PDC_CaptureStart

Table 1250:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure.
p_buffer	in	Pointer to store captured image data.

Parameter p_ctrl

Definition: `pdc_ctrl_t*const p_ctrl`

PDC control block. Allocate an instance specific control block to pass into the PDC API calls. Implemented aspdc_instance_ctrl_t

Parameter p_buffer

`uint8_t`

8.30.9.8 stateGet

```
ssp_err_t(* pdc_api_t::stateGet) (pdc_ctrl_t *const p_ctrl, pdc_state_t *p_state)
```

Detailed description

Get the state of the VSYNC and HSYNC pins. Implemented as

- R_PDC_StateGet

Table 1251:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure.
p_state	in	Pointer to store state data.

Parameter p_ctrl

Definition: `pdc_ctrl_t*const p_ctrl`

PDC control block. Allocate an instance specific control block to pass into the PDC API calls. Implemented aspdc_instance_ctrl_t

Parameter p_state

Definition: `pdc_state_t*p_state`

PDC VSYNC/HSYNC state

- `pdc_state_t::vsync`

  VSYNC signal state.

- pdc_state_t::hsync

  HSYNC signal state.

8.30.9.9 versionGet

```
ssp_err_t(* pdc_api_t::versionGet) (ssp_version_t *const p_data)
```

Detailed description

Return the version of the driver. Implemented as

- R_PDC_VersionGet

Table 1252:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure.
p_data	out	Memory address to return version information to.

Parameter p_ctrl
Parameter p_data

Definition: ssp_version_t *const p_data

- ssp_version_t::version_id

  Version id

- ssp_version_t::code_version_minor

  Code minor version.

- ssp_version_t::code_version_major

  Code major version.

- ssp_version_t::api_version_minor

  API minor version.

- ssp_version_t::api_version_major

  API major version.

- ssp_version_tstruct{}

  Code version parameters

8.30.9.10 pdc_instance_t

pdc_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- pdc_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- pdc_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- pdc_api_t const * p_api

  Pointer to the API structure for this instance.

8.31 Quad SPI Flash Interface

Interface for accessing external SPI flash devices.

8.31.1 Summary

The QSPI module provides an interface that writes and erases sectors in quad SPI flash devices connected to the QSPI interface.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

QSPI Interface description: QSPI Driver

8.31.2 Interface API

qspi_api_t

Function name	Description
.open	Open the QSPI driver module.
.close	Close the QSPI driver module.
.read	Read a block of data from the flash.
.pageProgram	Program a page of data to the flash.
.erase	Erase a certain number of bytes of the flash.

Function name	Description
.infoGet	Get information about this specific QSPI flash.
.sectorErase	Erase a sector of the flash.
.statusGet	Get the write or erase status of the flash.
.bankSelect	Select the bank to access.
.versionGet	Get the driver version based on compile time macros.

8.31.3 Data structures

- qspi_cfg_t
- qspi_info_t
- qspi_instance_t

8.31.4 Typedefs

- qspi_ctrl_t

8.31.5 Defines

- #define QSPI_API_VERSION_MAJOR
  Initial value: `(1U)`
- #define QSPI_API_VERSION_MINOR
  Initial value: `(2U)`

8.31.6 API Data

8.31.6.1 qspi_ctrl_t

```
typedef void qspi_ctrl_t
```

Detailed description

QSPI control block. Allocate an instance specific control block to pass into the QSPI API calls. Implemented as

- qspi_instance_ctrl_t

8.31.7 API Structures

8.31.7.1 qspi_cfg_t

qspi_cfg_t

Detailed description

User configuration structure used by the open function

Variables

- void * p_extend

  place holder for future development

8.31.7.2 qspi_info_t

qspi_info_t

Detailed description

QSPI information structure to store information specific to the currently used QSPI.

Variables

- uint32_t total_size_bytes

  Size of this QSPI in bytes.
- uint32_t min_program_size_bytes

  Minimum program size in bytes.
- uint32_t * p_erase_sizes_bytes

  Array of available erase sizes. Each entry is in bytes.
- uint8_t num_erase_sizes

  Number of available erase sizes.

8.31.7.3 qspi_api_t

qspi_api_t

Detailed description

QSPI functions implemented at the HAL layer follow this API.

8.31.7.4 open

```
(* qspi_api_t::open) (qspi_ctrl_t *p_ctrl, qspi_cfg_t const *const p_cfg)
```

Detailed description

Open the QSPI driver module. Implemented as

- R_QSPI_Open

Table 1253:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to a driver handle
p_cfg	in	Pointer to a configuration structure

Parameter p_ctrl

Definition: `qspi_ctrl_t*p_ctrl`

QSPI control block. Allocate an instance specific control block to pass into the QSPI API calls. Implemented asqspi_instance_ctrl_t

Parameter p_cfg

Definition: `qspi_cfg_t const *const p_cfg`

User configuration structure used by the open function

- `qspi_cfg_t`::`p_extend`

  place holder for future development

8.31.7.5 close

```
(* qspi_api_t::close) (qspi_ctrl_t *p_ctrl)
```

Detailed description

Close the QSPI driver module. Implemented as

- R_QSPI_Close

Table 1254:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to a driver handle

Parameter p_ctrl

Definition: `qspi_ctrl_t*p_ctrl`

QSPI control block. Allocate an instance specific control block to pass into the QSPI API calls. Implemented asqspi_instance_ctrl_t

8.31.7.6 read

```
(* qspi_api_t::read) (qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint8_t
*p_memory_address, uint32_t byte_count)
```

Detailed description

Read a block of data from the flash. Implemented as

- R_QSPI_Read

Table 1255:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to a driver handle
p_device_address	in	The location in the flash device address space to read
p_memory_address	in	The memory address of a buffer to place the read data in
byte_count	in	The number of bytes to read

Parameter p_ctrl

Definition: qspi_ctrl_t*p_ctrl

QSPI control block. Allocate an instance specific control block to pass into the QSPI API calls. Implemented asqspi_instance_ctrl_t

Parameter p_device_address

uint8_t

Parameter p_memory_address

uint8_t

Parameter byte_count

uint32_t

8.31.7.7 pageProgram

```
(* qspi_api_t::pageProgram) (qspi_ctrl_t *p_ctrl, uint8_t *p_device_address,
uint8_t *p_memory_address, uint32_t byte_count)
```

Detailed description

Program a page of data to the flash. Implemented as

- R_QSPI_PageProgram

Table 1256:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to a driver handle
p_device_address	in	The location in the flash device address space to write the data to
p_memory_address	in	The memory address of the data to write to the flash device
byte_count	in	The number of bytes to write

Parameter p_ctrl

Definition: `qspi_ctrl_t*p_ctrl`

QSPI control block. Allocate an instance specific control block to pass into the QSPI API calls. Implemented asqspi_instance_ctrl_t

Parameter p_device_address

`uint8_t`

Parameter p_memory_address

`uint8_t`

Parameter byte_count

`uint32_t`

8.31.7.8 erase

```
(* qspi_api_t::erase) (qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint32_t byte_count)
```

Detailed description

Erase a certain number of bytes of the flash. Implemented as

- R_QSPI_Erase

Table 1257:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to a driver handle

Table 1257:Parameters (Continued)

Name	Direction	Description
p_device_address	in	The location in the flash device address space to start the erase from
byte_count	in	The number of bytes to erase

Parameter p_ctrl

Definition: `qspi_ctrl_t*p_ctrl`

QSPI control block. Allocate an instance specific control block to pass into the QSPI API calls. Implemented asqspi_instance_ctrl_t

Parameter p_device_address

`uint8_t`

Parameter byte_count

`uint32_t`

8.31.7.9 infoGet

```
(* qspi_api_t::infoGet) (qspi_ctrl_t *const p_ctrl, qspi_info_t *const p_info)
```

Detailed description

Get information about this specific QSPI flash. Implemented as

- R_QSPI_InfoGet

Table 1258:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this timer.
p_info	out	Collection of information for this QSPI.

Parameter p_ctrl

Definition: `qspi_ctrl_t*const p_ctrl`

QSPI control block. Allocate an instance specific control block to pass into the QSPI API calls. Implemented asqspi_instance_ctrl_t

Parameter p_info

Definition: qspi_info_t*const p_info

QSPI information structure to store information specific to the currently used QSPI.

- qspi_info_t::total_size_bytes

  Size of this QSPI in bytes.

- qspi_info_t::min_program_size_bytes

  Minimum program size in bytes.

- qspi_info_t::p_erase_sizes_bytes

  Array of available erase sizes. Each entry is in bytes.

- qspi_info_t::num_erase_sizes

  Number of available erase sizes.

8.31.7.10 sectorErase

```
(* qspi_api_t::sectorErase) (qspi_ctrl_t *p_ctrl, uint8_t *p_device_address)
```

Detailed description

Erase a sector of the flash. Implemented as

- R_QSPI_SectorErase

Table 1259:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to a driver handle
p_device_address	in	The location in the flash device address space to start the erase from

Parameter p_ctrl

Definition: qspi_ctrl_t*p_ctrl

QSPI control block. Allocate an instance specific control block to pass into the QSPI API calls. Implemented asqspi_instance_ctrl_t

Parameter p_device_address

uint8_t

8.31.7.11 statusGet

```
(* qspi_api_t::statusGet) (qspi_ctrl_t *p_ctrl, bool *p_write_in_progress)
```

Detailed description

Get the write or erase status of the flash. Implemented as

- R_QSPI_StatusGet

Table 1260:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to a driver handle
p_write_in_progress	in	The write or erase status - TRUE = write/erase in progress

Parameter p_ctrl

Definition: `qspi_ctrl_t*p_ctrl`

QSPI control block. Allocate an instance specific control block to pass into the QSPI API calls. Implemented asqspi_instance_ctrl_t

Parameter p_write_in_progress

`const`

8.31.7.12 bankSelect

```
(* qspi_api_t::bankSelect) (uint32_t bank)
```

Detailed description

Select the bank to access. Implemented as

- R_QSPI_BankSelect

Table 1261:Parameters

Name	Direction	Description
bank	in	The bank number

Parameter bank

`uint32_t`

8.31.7.13 versionGet

```
(* qspi_api_t::versionGet) ( *const p_version)
```

Detailed description

Get the driver version based on compile time macros. Implemented as

- R_QSPI_VersionGet

Table 1262:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.31.7.14 qspi_instance_t

qspi_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- qspi_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- qspi_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- qspi_api_t const * p_api

  Pointer to the API structure for this instance.

8.32 RTC Interface

Interface for accessing the Realtime Clock.

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

RTC description: RTC Driver

8.32.1 Interface API

rtc_api_t

Function name	Description
.open	Open the RTC driver.
.close	Close the RTC driver.
.configure	Configure the RTC driver.
.calendarTimeSet	Set the calendar time.
.calendarTimeGet	Get the calendar time.
.calendarAlarmSet	Set the calendar alarm time.
.calendarAlarmGet	Get the calendar alarm time.
.calendarCounterStart	Start the calendar counter.
.calendarCounterStop	Stop the calendar counter.
.irqEnable	Enable the alarm irq.
.irqDisable	Disable the alarm irq.
.periodicIrqRateSet	Set the periodic irq rate
.infoGet	Return the currently configure clock source for the RTC
.errorAdjustmentModeSet	Set time error adjustment mode.
.errorAdjustmentSet	Set time error adjustment.
.versionGet	Gets version and stores it in provided pointer p_version.

8.32.2 Data structures

- rtc_callback_args_t
- rtc_error_adjustment_cfg_t
- rtc_error_adjustment_mode_cfg_t
- rtc_alarm_time_t
- rtc_info_t

- rtc_cfg_t
- rtc_instance_t

8.32.3 Enumerations

- rtc_event_t
- rtc_clock_source_t
- rtc_status_t
- rtc_error_adjustment_t
- rtc_error_adjustment_mode_t
- rtc_error_adjustment_period_t
- rtc_periodic_irq_select_t

8.32.4 Typedefs

- rtc_time_t
- rtc_ctrl_t

8.32.5 Defines

- #define RTC_API_VERSION_MAJOR

  Initial value: `(1U)`

  Use of time structure, tm
- #define RTC_API_VERSION_MINOR

  Initial value: `(4U)`

8.32.6 API Data

8.32.6.1 rtc_event_t

`rtc_event_t`

Detailed description

Events that can trigger a callback function

Enumerated values

Name	Description
RTC_EVENT_ALARM_IRQ	Real Time Clock ALARM IRQ.
RTC_EVENT_PERIODIC_IRQ	Real Time Clock PERIODIC IRQ.
RTC_EVENT_CARRY_IRQ	Real Time Clock CARRY IRQ.

8.32.6.2 rtc_clock_source_t

`rtc_clock_source_t`

Detailed description

Clock source for the RTC block

Enumerated values

Name	Description
RTC_CLOCK_SOURCE_SUBCLK	Sub-clock oscillator.
RTC_CLOCK_SOURCE_LOCO	Low power On Chip Oscillator.

8.32.6.3 rtc_status_t

`rtc_status_t`

Detailed description

RTC run state

Enumerated values

Name	Description
RTC_STATUS_STOPPED	RTC counter is stopped.
RTC_STATUS_RUNNING	RTC counter is running.

8.32.6.4 rtc_error_adjustment_t

`rtc_error_adjustment_t`

Detailed description

Time error adjustment settings

Enumerated values

Name	Description
RTC_ERROR_ADJUSTMENT_NONE	Adjustment is not performed.
RTC_ERROR_ADJUSTMENT_ADD_PRESCALER	Adjustment is performed by the addition to the prescaler.
RTC_ERROR_ADJUSTMENT_SUBTRACT_PRESCALER	Adjustment is performed by the subtraction from the prescaler.

8.32.6.5 rtc_error_adjustment_mode_t

`rtc_error_adjustment_mode_t`

Detailed description

Time error adjustment mode settings

Enumerated values

Name	Description
RTC_ERROR_ADJUSTMENT_MODE_MANUAL	Adjustment mode is set to manual.
RTC_ERROR_ADJUSTMENT_MODE_AUTOMATIC	Adjustment mode is set to automatic.

8.32.6.6 rtc_error_adjustment_period_t

`rtc_error_adjustment_period_t`

Detailed description

Time error adjustment period settings

Enumerated values

Name	Description
RTC_ERROR_ADJUSTMENT_PERIOD_1_MINUTE	Adjustment period is set to every one minute.
RTC_ERROR_ADJUSTMENT_PERIOD_10_SECOND	Adjustment period is set to every ten second.
RTC_ERROR_ADJUSTMENT_PERIOD_NONE	Adjustment period not supported in manual mode.

8.32.6.7 rtc_periodic_irq_select_t

`rtc_periodic_irq_select_t`

Detailed description

Periodic Interrupt select

Enumerated values

Name	Description
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_256_SECOND	A periodic irq is generated every 1/256 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_128_SECOND	A periodic irq is generated every 1/128 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_64_SECOND	A periodic irq is generated every 1/64 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_32_SECOND	A periodic irq is generated every 1/32 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_16_SECOND	A periodic irq is generated every 1/16 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_8_SECOND	A periodic irq is generated every 1/8 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_4_SECOND	A periodic irq is generated every 1/4 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_2_SECOND	A periodic irq is generated every 1/2 second.
RTC_PERIODIC_IRQ_SELECT_1_SECOND	A periodic irq is generated every 1 second.
RTC_PERIODIC_IRQ_SELECT_2_SECONDS	A periodic irq is generated every 2 seconds.

8.32.6.8 rtc_time_t

```
typedef struct tm rtc_time_t
```

Detailed description

Date and time structure defined in C standard library <time.h>

8.32.6.9 rtc_ctrl_t

```
typedef void rtc_ctrl_t
```

Detailed description

RTC control block. Allocate an instance specific control block to pass into the RTC API calls. Implemented as

- rtc_instance_ctrl_t

8.32.7 API Structures

8.32.7.1 rtc_callback_args_t

rtc_callback_args_t

Detailed description

Callback function parameter data

Variables

- rtc_event_t event

  The event can be used to identify what caused the callback (compare match or error).

- void const * p_context

  Placeholder for user data. Set in r_timer_t::open function in timer_cfg_t.

8.32.7.2 rtc_error_adjustment_cfg_t

rtc_error_adjustment_cfg_t

Detailed description

Time error adjustment value configuration

Variables

- rtc_error_adjustment_t adjustment_type

  Time error adjustment type setting.

- uint8_t adjustment_value

  Time error adjustment value.

8.32.7.3 rtc_error_adjustment_mode_cfg_t

rtc_error_adjustment_mode_cfg_t

Detailed description

Time error adjustment mode and period configuration

Variables

- rtc_error_adjustment_mode_t adjustment_mode

  Time error adjustment mode setting.

- rtc_error_adjustment_period_t adjustment_period

  Time error adjustment period setting.

8.32.7.4 rtc_alarm_time_t

rtc_alarm_time_t

Detailed description

Alarm time setting structure

Variables

- rtc_time_t time

  Time structure.
- bool sec_match

  Enable the alarm based on a match of the seconds field.
- bool min_match

  Enable the alarm based on a match of the minutes field.
- bool hour_match

  Enable the alarm based on a match of the hours field.
- bool mday_match

  Enable the alarm based on a match of the days field.
- bool mon_match

  Enable the alarm based on a match of the months field.
- bool year_match

  Enable the alarm based on a match of the years field.
- bool dayofweek_match

  Enable the alarm based on a match of the dayofweek field.

8.32.7.5 rtc_info_t

rtc_info_t

Detailed description

RTC Information Structure for information returned by infoGet()

Variables

- rtc_clock_source_t clock_source

  Clock source for the RTC block.
- rtc_status_t status

  RTC run status.

8.32.7.6 rtc_cfg_t

rtc_cfg_t

Detailed description

User configuration structure, used in open function

Variables

- rtc_clock_source_t clock_source

  Clock source for the RTC block.
- bool hw_cfg

  Initialize RTC in Open()
- uint32_t error_adjustment_value

  Value of the prescaler for error adjustment.
- rtc_error_adjustment_t error_adjustment_type

  How the prescaler value is applied.
- uint8_t alarm_ipl

  Alarm interrupt priority.
- uint8_t periodic_ipl

  Periodic interrupt priority.
- uint8_t carry_ipl

  Carry interrupt priority.
- void(* p_callback)(rtc_callback_args_t *p_args)

  Called from the ISR.
- void const * p_context

  Passed to the callback.
- void const * p_extend

  RTC hardware dependant configuration.

8.32.7.7 rtc_api_t

rtc_api_t

Detailed description

RTC driver structure. General RTC functions implemented at the HAL layer follow this API.

8.32.7.8 open

```
ssp_err_t(* rtc_api_t::open) (rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const
p_cfg)
```

Detailed description

Open the RTC driver. Implemented as

- R_RTC_Open

Table 1263:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to RTC device handle
p_cfg	in	Pointer to the configuration structure

Parameter p_ctrl

Definition: `rtc_ctrl_t*const p_ctrl`

RTC control block. Allocate an instance specific control block to pass into the RTC API calls. Implemented asrtc_instance_ctrl_t

Parameter p_cfg

Definition: `rtc_cfg_t const *const p_cfg`

User configuration structure, used in open function

- `rtc_cfg_t`::`rtc_clock_source_t`

  Clock source for the RTC block.

  Enumerated as:

  – RTC_CLOCK_SOURCE_SUBCLK

  – RTC_CLOCK_SOURCE_LOCO

- `rtc_cfg_t`::`hw_cfg`

  Initialize RTC in Open()

- `rtc_cfg_t`::`error_adjustment_value`

  Value of the prescaler for error adjustment.

- `rtc_cfg_t`::`rtc_error_adjustment_t`

  How the prescaler value is applied.

  Enumerated as:

  – RTC_ERROR_ADJUSTMENT_NONE

  – RTC_ERROR_ADJUSTMENT_ADD_PRESCALER

  – RTC_ERROR_ADJUSTMENT_SUBTRACT_PRESCALER

- `rtc_cfg_t`::`alarm_ipl`

  Alarm interrupt priority.

- `rtc_cfg_t`::`periodic_ipl`

  Periodic interrupt priority.

- rtc_cfg_t::carry_ipl

  Carry interrupt priority.
- rtc_cfg_t::p_callback

  Called from the ISR.
- rtc_cfg_t::p_context

  Passed to the callback.
- rtc_cfg_t::p_extend

  RTC hardware dependant configuration.

8.32.7.9 close

```
ssp_err_t(* rtc_api_t::close) (rtc_ctrl_t *const p_ctrl)
```

Detailed description

Close the RTC driver. Implemented as

- R_RTC_Close

Table 1264:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to RTC device handle.

Parameter p_ctrl

Definition: rtc_ctrl_t*const p_ctrl

RTC control block. Allocate an instance specific control block to pass into the RTC API calls. Implemented asrtc_instance_ctrl_t

8.32.7.10 configure

```
ssp_err_t(* rtc_api_t::configure) (rtc_ctrl_t *const p_ctrl, void *const p_extend)
```

Detailed description

Configure the RTC driver. Implemented as

- R_RTC_Configure

Table 1265:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to RTC device handle.

Table 1265:Parameters (Continued)

Name	Direction	Description
p_extend	in	Currently not implemented, pass NULL.

Parameter p_ctrl

Definition: `rtc_ctrl_t*const p_ctrl`

RTC control block. Allocate an instance specific control block to pass into the RTC API calls. Implemented asrtc_instance_ctrl_t

Parameter p_extend

`const`

8.32.7.11 calendarTimeSet

```
ssp_err_t(* rtc_api_t::calendarTimeSet) (rtc_ctrl_t *const p_ctrl, rtc_time_t
*p_time, bool clock_start)
```

Detailed description

Set the calendar time. Implemented as

- R_RTC_CalendarTimeSet

Table 1266:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to RTC device handle
p_time	in	Pointer to a time structure that contains the time to set
clock_start	in	Flag that starts the clock right after it is set

Parameter p_ctrl

Definition: `rtc_ctrl_t*const p_ctrl`

RTC control block. Allocate an instance specific control block to pass into the RTC API calls. Implemented asrtc_instance_ctrl_t

Parameter p_time

Definition: `rtc_time_t*p_time`

Date and time structure defined in C standard library <time.h>

Parameter clock_start

`const`

8.32.7.12 calendarTimeGet

```
ssp_err_t(* rtc_api_t::calendarTimeGet) (rtc_ctrl_t *const p_ctrl, rtc_time_t
*p_time)
```

Detailed description

Get the calendar time. Implemented as

- R_RTC_CalendarTimeGet

Table 1267:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to RTC device handle
p_time	out	Pointer to a time structure that contains the time to get

Parameter p_ctrl

Definition: `rtc_ctrl_t*const p_ctrl`

RTC control block. Allocate an instance specific control block to pass into the RTC API calls. Implemented asrtc_instance_ctrl_t

Parameter p_time

Definition: `rtc_time_t*p_time`

Date and time structure defined in C standard library <time.h>

8.32.7.13 calendarAlarmSet

```
ssp_err_t(* rtc_api_t::calendarAlarmSet) (rtc_ctrl_t *const p_ctrl,
rtc_alarm_time_t *p_alarm, bool irq_enable_flag)
```

Detailed description

Set the calendar alarm time. Implemented as

- R_RTC_CalendarAlarmSet

Table 1268:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to RTC device handle
p_alarm	in	Pointer to an alarm structure that contains the alarm time to set
irq_enable_flag	in	Enable the ALARM irq if set

Parameter p_ctrl

Definition: `rtc_ctrl_t*const p_ctrl`

RTC control block. Allocate an instance specific control block to pass into the RTC API calls. Implemented asrtc_instance_ctrl_t

Parameter p_alarm

Definition: `rtc_alarm_time_t*p_alarm`

Alarm time setting structure

- `rtc_alarm_time_t::time`

  Time structure.

- `rtc_alarm_time_t::sec_match`

  Enable the alarm based on a match of the seconds field.

- `rtc_alarm_time_t::min_match`

  Enable the alarm based on a match of the minutes field.

- `rtc_alarm_time_t::hour_match`

  Enable the alarm based on a match of the hours field.

- `rtc_alarm_time_t::mday_match`

  Enable the alarm based on a match of the days field.

- `rtc_alarm_time_t::mon_match`

  Enable the alarm based on a match of the months field.

- `rtc_alarm_time_t::year_match`

  Enable the alarm based on a match of the years field.

- `rtc_alarm_time_t::dayofweek_match`

  Enable the alarm based on a match of the dayofweek field.

Parameter irq_enable_flag

`const`

8.32.7.14 calendarAlarmGet

```
ssp_err_t(* rtc_api_t::calendarAlarmGet) (rtc_ctrl_t *const p_ctrl,
rtc_alarm_time_t *p_alarm)
```

Detailed description

Get the calendar alarm time. Implemented as

- R_RTC_CalendarAlarmGet

Table 1269:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to RTC device handle
p_alarm	out	Pointer to an alarm structure to fill up with the alarm time

Parameter p_ctrl

Definition: `rtc_ctrl_t*const p_ctrl`

RTC control block. Allocate an instance specific control block to pass into the RTC API calls. Implemented asrtc_instance_ctrl_t

Parameter p_alarm

Definition: `rtc_alarm_time_t*p_alarm`

Alarm time setting structure

- `rtc_alarm_time_t::time`

  Time structure.

- `rtc_alarm_time_t::sec_match`

  Enable the alarm based on a match of the seconds field.

- `rtc_alarm_time_t::min_match`

  Enable the alarm based on a match of the minutes field.

- `rtc_alarm_time_t::hour_match`

  Enable the alarm based on a match of the hours field.

- `rtc_alarm_time_t::mday_match`

  Enable the alarm based on a match of the days field.

- `rtc_alarm_time_t::mon_match`

  Enable the alarm based on a match of the months field.

- `rtc_alarm_time_t::year_match`

  Enable the alarm based on a match of the years field.

- `rtc_alarm_time_t::dayofweek_match`

  Enable the alarm based on a match of the dayofweek field.

8.32.7.15 calendarCounterStart

```
ssp_err_t(* rtc_api_t::calendarCounterStart) (rtc_ctrl_t *const p_ctrl)
```

Detailed description

Start the calendar counter. Implemented as

- R_RTC_CalendarCounterStart

Table 1270:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to RTC device handle

Parameter p_ctrl

Definition: `rtc_ctrl_t*const p_ctrl`

RTC control block. Allocate an instance specific control block to pass into the RTC API calls. Implemented asrtc_instance_ctrl_t

8.32.7.16 calendarCounterStop

```
ssp_err_t(* rtc_api_t::calendarCounterStop) (rtc_ctrl_t *const p_ctrl)
```

Detailed description

Stop the calendar counter. Implemented as

- R_RTC_CalendarCounterStop

Table 1271:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to RTC device handle

Parameter p_ctrl

Definition: `rtc_ctrl_t*const p_ctrl`

RTC control block. Allocate an instance specific control block to pass into the RTC API calls. Implemented asrtc_instance_ctrl_t

8.32.7.17 irqEnable

```
ssp_err_t(* rtc_api_t::irqEnable) (rtc_ctrl_t *const p_ctrl, rtc_event_t irq)
```

Detailed description

Enable the alarm irq. Implemented as

- R_RTC_IrqEnable

Table 1272:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to RTC device handle

Parameter p_ctrl

Definition: `rtc_ctrl_t*const p_ctrl`

RTC control block. Allocate an instance specific control block to pass into the RTC API calls. Implemented asrtc_instance_ctrl_t

8.32.7.18 irqDisable

```
ssp_err_t(* rtc_api_t::irqDisable) (rtc_ctrl_t *const p_ctrl, rtc_event_t irq)
```

Detailed description

Disable the alarm irq. Implemented as

- R_RTC_IrqDisable

Table 1273:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to RTC device handle

Parameter p_ctrl

Definition: `rtc_ctrl_t*const p_ctrl`

RTC control block. Allocate an instance specific control block to pass into the RTC API calls. Implemented asrtc_instance_ctrl_t

8.32.7.19 periodicIrqRateSet

```
ssp_err_t(* rtc_api_t::periodicIrqRateSet) (rtc_ctrl_t *const p_ctrl,
rtc_periodic_irq_select_t rate)
```

Detailed description

Set the periodic irq rate Implemented as

- R_RTC_PeriodicIrqRateSet

Table 1274:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to RTC device handle
rate	in	Rate of periodic interrupts

Parameter p_ctrl

Definition: `rtc_ctrl_t*const p_ctrl`

RTC control block. Allocate an instance specific control block to pass into the RTC API calls. Implemented asrtc_instance_ctrl_t

Parameter rate

Definition: `rtc_periodic_irq_select_trate`

Periodic Interrupt select

8.32.7.20 infoGet

```
ssp_err_t(* rtc_api_t::infoGet) (rtc_ctrl_t *p_ctrl, rtc_info_t *p_rtc_info)
```

Detailed description

Return the currently configure clock source for the RTC

Implemented as

- R_RTC_InfoGet

Table 1275:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
p_rtc_info	out	Pointer to RTC information structure

Parameter p_ctrl

Definition: `rtc_ctrl_t*p_ctrl`

RTC control block. Allocate an instance specific control block to pass into the RTC API calls. Implemented asrtc_instance_ctrl_t

Parameter p_rtc_info

Definition: `rtc_info_t*p_rtc_info`

RTC Information Structure for information returned by infoGet()

- rtc_info_t::clock_source

  Clock source for the RTC block.

- rtc_info_t::status

  RTC run status.

8.32.7.21 errorAdjustmentModeSet

```
ssp_err_t(* rtc_api_t::errorAdjustmentModeSet) (rtc_ctrl_t *p_ctrl,
rtc_error_adjustment_mode_cfg_t *p_error_adjustment_mode)
```

Detailed description

Set time error adjustment mode.

Implemented as

- R_RTC_ErrorAdjustmentModeSet

Table 1276:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
p_error_adjustment_mode	in	Pointer to error adjustment mode configuration structure

Parameter p_ctrl

Definition: rtc_ctrl_t*p_ctrl

RTC control block. Allocate an instance specific control block to pass into the RTC API calls. Implemented asrtc_instance_ctrl_t

Parameter p_error_adjustment_mode

Definition: rtc_error_adjustment_mode_cfg_t *p_error_adjustment_mode

Time error adjustment mode and period configuration

- rtc_error_adjustment_mode_cfg_t::rtc_error_adjustment_mode_t

  Time error adjustment mode setting.

  Enumerated as:

  – RTC_ERROR_ADJUSTMENT_MODE_MANUAL

  – RTC_ERROR_ADJUSTMENT_MODE_AUTOMATIC

- rtc_error_adjustment_mode_cfg_t::rtc_error_adjustment_period_t

  Time error adjustment period setting.

  Enumerated as:

– RTC_ERROR_ADJUSTMENT_PERIOD_1_MINUTE

– RTC_ERROR_ADJUSTMENT_PERIOD_10_SECOND

– RTC_ERROR_ADJUSTMENT_PERIOD_NONE

8.32.7.22 errorAdjustmentSet

```
ssp_err_t(* rtc_api_t::errorAdjustmentSet) (rtc_ctrl_t *p_ctrl,
rtc_error_adjustment_cfg_t *p_error_adjustment_config)
```

Detailed description

Set time error adjustment.

Implemented as

- R_RTC_ErrorAdjustmentSet

Table 1277:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control handle structure
p_error_adjustment_config	in	Pointer to error adjustment structure

Parameter p_ctrl

Definition: `rtc_ctrl_t*p_ctrl`

RTC control block. Allocate an instance specific control block to pass into the RTC API calls. Implemented asrtc_instance_ctrl_t

Parameter p_error_adjustment_config

Definition: `rtc_error_adjustment_cfg_t *p_error_adjustment_config`

Time error adjustment value configuration

- `rtc_error_adjustment_cfg_t`::`rtc_error_adjustment_t`

  Time error adjustment type setting.

  Enumerated as:

  – RTC_ERROR_ADJUSTMENT_NONE

  – RTC_ERROR_ADJUSTMENT_ADD_PRESCALER

  – RTC_ERROR_ADJUSTMENT_SUBTRACT_PRESCALER

- `rtc_error_adjustment_cfg_t`::`adjustment_value`

  Time error adjustment value.

8.32.7.23 versionGet

```
ssp_err_t(* rtc_api_t::versionGet) (ssp_version_t *version)
```

Detailed description

Gets version and stores it in provided pointer p_version. Implemented as

- R_RTC_VersionGet

Table 1278:Parameters

Name	Direction	Description
p_version	out	Code and API version used

Parameter p_version

8.32.7.24 rtc_instance_t

rtc_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- rtc_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- rtc_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- rtc_api_t const * p_api

  Pointer to the API structure for this instance.

8.33 SD/MMC Interface

Interface for accessing SD, eMMC, and SDIO devices.

8.33.1 Summary

The r_sdmmc interface provides standard SD and eMMC media functionality. A complete file system can be implemented with FileX, sf_el_fx, sf_block_media_sdmmc and r_sdmmc modules. This driver also supports SDIO. The SD/MMC interface is implemented by:

- SDMMC

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

SD/MMC description: SD/MMC Driver and SDIO Driver

8.33.2 Interface API

sdmmc_api_t

Function name	Description
.open	Open an SD/MMC device. If the device is a card, the card must be plugged in prior to calling this API. This API blocks until the device initialization procedure is complete.
.close	Close open SD/MMC device.
.read	Read data from an SD/MMC channel. This API is not supported for SDIO devices.
.write	Write data to SD/MMC channel. This API is not supported for SDIO devices.
.control	The Control function sends control commands to and receives info from the SD/MMC port.
.readIo	Read one byte of I/O data from an SDIO device. This API is not supported for SD or eMMC memory devices.
.writeIo	Write one byte of I/O data to an SDIO device. This API is not supported for SD or eMMC memory devices.
.readIoExt	Read multiple bytes or blocks of I/O data from an SDIO device. This API is not supported for SD or eMMC memory devices.
.writeIoExt	Write multiple bytes or blocks of I/O data to an SDIO device. This API is not supported for SD or eMMC memory devices.
.ioIntEnable	Enables SDIO interrupt for SD/MMC instance. This API is not supported for SD or eMMC memory devices.
.versionGet	Returns the version of the SD/MMC driver.
.infoGet	Get SD/MMC device info.

Function name	Description
.erase	Erase SD/MMC sectors. The sector size for erase is fixed at 512 bytes. This API is not supported for SDIO devices.

8.33.3 Data structures

- sdmmc_hw_t
- sdmmc_info_t
- sdmmc_callback_args_t
- sdmmc_cfg_t
- sdmmc_instance_t

8.33.4 Enumerations

- sdmmc_ready_status_t
- sdmmc_card_type_t
- sdmmc_media_type_t
- sdmmc_bus_width_t
- sdmmc_io_transfer_mode_t
- sdmmc_io_address_mode_t
- sdmmc_io_write_mode_t
- sdmmc_event_t

8.33.5 Typedefs

- sdmmc_ctrl_t

8.33.6 Defines

- #define SDMMC_API_VERSION_MAJOR

  Initial value: `(1U)`
- #define SDMMC_API_VERSION_MINOR

  Initial value: `(2U)`
- #define SDMMC_MAX_BLOCK_SIZE

  Initial value: `(512U)`

8.33.7 API Data

8.33.7.1 sdmmc_ready_status_t

`sdmmc_ready_status_t`

Detailed description

SD/MMC status

Enumerated values

Name	Description
SDMMC_STATUS_CARD_NOT_READY	SD card or eMMC device has not been initialized.
SDMMC_STATUS_CARD_READY	SD card or eMMC device has been initialized and is ready to access.

8.33.7.2 sdmmc_card_type_t

`sdmmc_card_type_t`

Detailed description

SD/MMC media uses SD protocol or MMC protocol.

Enumerated values

Name	Description
SDMMC_CARD_TYPE_MMC	The media is an eMMC device.
SDMMC_CARD_TYPE_SD	The media is an SD card.

8.33.7.3 sdmmc_media_type_t

`sdmmc_media_type_t`

Detailed description

SD/MMC media is embedded or it can be inserted and removed.

Enumerated values

Name	Description
SDMMC_MEDIA_TYPE_EMBEDDED	The media is an embedded card, or eMMC device.

Name	Description
SDMMC_MEDIA_TYPE_CARD	The media is an pluggable card.

8.33.7.4 sdmmc_bus_width_t

`sdmmc_bus_width_t`

Detailed description

SD/MMC data bus is 1, 4 or 8 bits wide.

Enumerated values

Name	Description
SDMMC_BUS_WIDTH_1_BIT	Data bus is 1 bit wide.
SDMMC_BUS_WIDTH_4_BITS	Data bus is 4 bits wide.
SDMMC_BUS_WIDTH_8_BITS	Data bus is 8 bits wide.

8.33.7.5 sdmmc_io_transfer_mode_t

`sdmmc_io_transfer_mode_t`

Enumerated values

Name	Description
SDMMC_IO_MODE_TRANSFER_BYTE	
SDMMC_IO_MODE_TRANSFER_BLOCK	

8.33.7.6 sdmmc_io_address_mode_t

`sdmmc_io_address_mode_t`

Enumerated values

Name	Description
SDMMC_IO_ADDRESS_MODE_FIXED	
SDMMC_IO_ADDRESS_MODE_INCREMENT	

8.33.7.7 sdmmc_io_write_mode_t

```
sdmmc_io_write_mode_t
```

Enumerated values

Name	Description
SDMMC_IO_WRITE_MODE_NO_READ	
SDMMC_IO_WRITE_READ_AFTER_WRITE	

8.33.7.8 sdmmc_event_t

```
sdmmc_event_t
```

Detailed description

Events that can trigger a callback function

Enumerated values

Name	Description
SDMMC_EVENT_CARD_REMOVED	Card removed event.
SDMMC_EVENT_CARD_INSERTED	Card inserted event.
SDMMC_EVENT_ACCESS	Access event.
SDMMC_EVENT_SDIO	IO event.
SDMMC_EVENT_TRANSFER_COMPLETE	Read or write complete.
SDMMC_EVENT_TRANSFER_ERROR	Read or write failed.
SDMMC_EVENT_NONE	No event.

8.33.7.9 sdmmc_ctrl_t

```
typedef void sdmmc_ctrl_t
```

Detailed description

SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls. Implemented as

- sdmmc_instance_ctrl_t

8.33.8 API Structures

8.33.8.1 sdmmc_hw_t

sdmmc_hw_t

Detailed description

Channel, media type, bus width defined by the hardware.

Variables

- uint8_t channel

  Channel of SD/MMC host interface.

- sdmmc_media_type_t media_type

  Embedded or pluggable card.

- sdmmc_bus_width_t bus_width

  Device bus width is 1, 4 or 8 bits wide.

8.33.8.2 sdmmc_info_t

sdmmc_info_t

Detailed description

Status and other information obtained from the media device.

Variables

- sdmmc_card_type_t card_type

  SD or eMMC.

- bool ready

  False if card was removed (only applies if MCU supports card detection and SDnCD pin is connected).True otherwise.If ready is false, the driver must be closed, then reopened with a card inserted.

- bool hc

  true = Card is High Capacity card

- bool sdio

  true = SDIO present

- bool write_protected

  true = Card is write protected

- bool transfer_in_progress

  true = Card is busy

- uint8_t csd_version

  CSD version.

- uint8_t device_type

  Speed and data rate (eMMC)
- sdmmc_bus_width_t bus_width

  Current media bus width.
- uint8_t hs_timing

  High Speed status.
- uint32_t sdhi_rca

  Relative Card Address.
- uint32_t max_clock_rate

  Maximum clock rate for media card.
- uint32_t clock_rate

  Current clock rate.
- uint32_t sector_size

  Sector size.
- uint32_t sector_count

  Sector count.
- uint32_t erase_sector_count

  Minimum erasable unit (in 512 byte sectors)

8.33.8.3 sdmmc_callback_args_t

sdmmc_callback_args_t

Detailed description

Callback function parameter data

Variables

- sdmmc_event_t event

  The event can be used to identify what caused the callback.
- void const * p_context

  Placeholder for user data.

8.33.8.4 sdmmc_cfg_t

sdmmc_cfg_t

Detailed description

SD/MMC Configuration

Variables

- sdmmc_hw_t hw

  Channel, media type, bus width defined by the hardware.
- transfer_instance_t const * p_lower_lvl_transfer

  Transfer instance used to move data with DMA or DTC.
- void(* p_callback)(sdmmc_callback_args_t *p_args)

  Pointer to callback function.
- void const * p_context

  User defined context passed into callback function.
- void const * p_extend

  SD/MMC hardware dependent configuration.
- uint8_t access_ipl

  Access interrupt priority.
- uint8_t sdio_ipl

  SDIO interrupt priority.
- uint8_t card_ipl

  Card interrupt priority.
- uint8_t dma_req_ipl

  DMA request interrupt priority.

8.33.8.5 sdmmc_api_t

sdmmc_api_t

Detailed description

SD/MMC functions implemented at the HAL layer API.

8.33.8.6 open

```
ssp_err_t(* sdmmc_api_t::open) (sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const
*const p_cfg)
```

Detailed description

Open an SD/MMC device. If the device is a card, the card must be plugged in prior to calling this API. This API blocks until the device initialization procedure is complete.

Implemented asR_SDMMC_Open

Table 1279:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to SD/MMC instance control block.
p_cfg	in	Pointer to SD/MMC instance configuration structure.

Parameter p_ctrl

Definition: `sdmmc_ctrl_t*const p_ctrl`

SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls. Implemented assdmmc_instance_ctrl_t

Parameter p_cfg

Definition: `sdmmc_cfg_t const *const p_cfg`

SD/MMC Configuration

- `sdmmc_cfg_t`::`sdmmc_hw_t`

  Channel, media type, bus width defined by the hardware.

- `sdmmc_cfg_t`::`transfer_instance_t`

  Transfer instance used to move data with DMA or DTC.

- `sdmmc_cfg_t`::`p_callback`

  Pointer to callback function.

- `sdmmc_cfg_t`::`p_context`

  User defined context passed into callback function.

- `sdmmc_cfg_t`::`p_extend`

  SD/MMC hardware dependent configuration.

- `sdmmc_cfg_t`::`access_ipl`

  Access interrupt priority.

- `sdmmc_cfg_t`::`sdio_ipl`

  SDIO interrupt priority.

- `sdmmc_cfg_t`::`card_ipl`

  Card interrupt priority.

- `sdmmc_cfg_t`::`dma_req_ipl`

  DMA request interrupt priority.

8.33.8.7 close

```
ssp_err_t(* sdmmc_api_t::close) (sdmmc_ctrl_t *const p_ctrl)
```

Detailed description

Close open SD/MMC device.

Implemented asR_SDMMC_Close

Table 1280:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to an open SD/MMC instance control block.

Parameter p_ctrl

Definition: `sdmmc_ctrl_t*const p_ctrl`

SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls. Implemented assdmmc_instance_ctrl_t

8.33.8.8 read

```
ssp_err_t(* sdmmc_api_t::read) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest,
uint32_t const start_sector, uint32_t const sector_count)
```

Detailed description

Read data from an SD/MMC channel. This API is not supported for SDIO devices.

Implemented asR_SDMMC_Read

Table 1281:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to an open SD/MMC instance control block.
p_dest	out	Pointer to data buffer to read data to.
start_sector	in	First sector address to read.
sector_count	in	Number of sectors to read. All sectors must be in the range of sector_count.

Parameter p_ctrl

Definition: `sdmmc_ctrl_t*const p_ctrl`

SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls. Implemented assdmmc_instance_ctrl_t

Parameter p_dest

`uint8_t`

Parameter start_sector

`uint32_t`

Parameter sector_count

`uint32_t`

8.33.8.9 write

```
ssp_err_t(* sdmmc_api_t::write) (sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count)
```

Detailed description

Write data to SD/MMC channel. This API is not supported for SDIO devices.

Implemented asR_SDMMC_Write

Table 1282:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to an open SD/MMC instance control block.
p_source	in	Pointer to data buffer to write data from.
start_sector	in	First sector address to write to.
sector_count	in	Number of sectors to write. All sectors must be in the range of sector_count.

Parameter p_ctrl

Definition: `sdmmc_ctrl_t*const p_ctrl`

SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls. Implemented assdmmc_instance_ctrl_t

Parameter p_source

`uint8_t`

Parameter start_sector

`uint32_t`

Parameter sector_count

`uint32_t`

8.33.8.10 control

```
ssp_err_t(* sdmmc_api_t::control) (sdmmc_ctrl_t *const p_ctrl, ssp_command_t
const command, void *p_data)
```

Detailed description

The Control function sends control commands to and receives info from the SD/MMC port.

Implemented asR_SDMMC_Control

Table 1283:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to an open SD/MMC instance control block.
command	in	Command to execute. The list of supported commands is below.
p_data	inout	Pointer to data in or out. For each command, this data should be cast as follows: • SSP_COMMAND_GET_SECTOR_COUNT : [out] (uint32_t *) p_data • SSP_COMMAND_GET_SECTOR_SIZE : [out] (uint32_t *) p_data • SSP_COMMAND_GET_WRITE_PROTECTED : [out] (bool *) p_data • SSP_COMMAND_SET_BLOCK_SIZE : [in] (uint32_t *) p_data

Parameter p_ctrl

Definition: sdmmc_ctrl_t*const p_ctrl

SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls. Implemented assdmmc_instance_ctrl_t

Parameter command
Parameter p_data

```
const
```

8.33.8.11 readIo

```
ssp_err_t(* sdmmc_api_t::readIo) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const
p_data, uint32_t const function, uint32_t const address)
```

Detailed description

Read one byte of I/O data from an SDIO device. This API is not supported for SD or eMMC memory devices.

Implemented asR_SDMMC_ReadIo

Table 1284:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to an open SD/MMC instance control block.
p_data	out	Pointer to location to store data byte.
function	in	SDIO Function Number.
address	in	SDIO register address.

Parameter p_ctrl

Definition: `sdmmc_ctrl_t*const p_ctrl`

SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls. Implemented assdmmc_instance_ctrl_t

Parameter p_data

`uint8_t`

Parameter function

`uint32_t`

Parameter address

`uint32_t`

8.33.8.12 writeIo

```
ssp_err_t(* sdmmc_api_t::writeIo) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const
p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t
const read_after_write)
```

Detailed description

Write one byte of I/O data to an SDIO device. This API is not supported for SD or eMMC memory devices.

Implemented asR_SDMMC_WriteIo

Table 1285:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to an open SD/MMC instance control block.

Table 1285:Parameters (Continued)

Name	Direction	Description
p_data	inout	Pointer to data byte to write. Read data is also provided here if read_after_write is true.
function	in	SDIO Function Number.
address	in	SDIO register address.
read_after_write	in	Set to true for reading after writing.

Parameter p_ctrl

Definition: `sdmmc_ctrl_t*const p_ctrl`

SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls. Implemented assdmmc_instance_ctrl_t

Parameter p_data

`uint8_t`

Parameter function

`uint32_t`

Parameter address

`uint32_t`

Parameter read_after_write

Definition: `sdmmc_io_write_mode_tconst read_after_write`

8.33.8.13 readIoExt

```
ssp_err_t(* sdmmc_api_t::readIoExt) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const
p_dest, uint32_t const function, uint32_t const address, uint32_t *const count,
sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
```

Detailed description

Read multiple bytes or blocks of I/O data from an SDIO device. This API is not supported for SD or eMMC memory devices.

Implemented asR_SDMMC_ReadIoExt

Table 1286:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to an open SD/MMC instance control block.
p_dest	out	Pointer to data buffer to read data to.
function	in	SDIO Function Number.
address	in	SDIO register address.
count	in	Number of bytes or blocks to read, maximum 512 bytes or 511 blocks.
transfer_mode	in	Byte mode = 0, Block mode = 1.
address_mode	in	0 = fixed address, 1 = Incrementing address.

Parameter p_ctrl

Definition: `sdmmc_ctrl_t*const p_ctrl`

SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls. Implemented assdmmc_instance_ctrl_t

Parameter p_dest

`uint8_t`

Parameter function

`uint32_t`

Parameter address

`uint32_t`

Parameter count

`uint32_t`

Parameter transfer_mode
Parameter address_mode

8.33.8.14 writeIoExt

```
ssp_err_t(* sdmmc_api_t::writeIoExt) (sdmmc_ctrl_t *const p_ctrl, uint8_t const
*const p_source, uint32_t const function, uint32_t const address, uint32_t const
count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t
address_mode)
```

Detailed description

Write multiple bytes or blocks of I/O data to an SDIO device. This API is not supported for SD or eMMC memory devices.

Implemented asR_SDMMC_WriteIoExt

Table 1287:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to an open SD/MMC instance control block.
p_source	in	Pointer to data buffer to write data from.
function_number	in	SDIO Function Number.
address	in	SDIO register address.
count	in	Number of bytes or blocks to write, maximum 512 bytes or 511 blocks.
transfer_mode	in	Byte mode = 0, Block mode = 1.
address_mode	in	0 = fixed address, 1 = Incrementing address.

Parameter p_ctrl

Definition: `sdmmc_ctrl_t*const p_ctrl`

SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls. Implemented assdmmc_instance_ctrl_t

Parameter p_source

`uint8_t`

Parameter function_number
Parameter address

`uint32_t`

Parameter count

`uint32_t`

Parameter transfer_mode
Parameter address_mode

8.33.8.15 IoIntEnable

```
ssp_err_t(* sdmmc_api_t::IoIntEnable) (sdmmc_ctrl_t *const p_ctrl, bool enable)
```

Detailed description

Enables SDIO interrupt for SD/MMC instance. This API is not supported for SD or eMMC memory devices.

Implemented asR_SDMMC_IoIntEnable

Table 1288:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to an open SD/MMC instance control block.
enable	in	Interrupt enable = true, interrupt disable = false.

Parameter p_ctrl

Definition: `sdmmc_ctrl_t*const p_ctrl`

SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls. Implemented assdmmc_instance_ctrl_t

Parameter enable

`const`

8.33.8.16 versionGet

```
ssp_err_t(* sdmmc_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Returns the version of the SD/MMC driver.

Implemented asR_SDMMC_VersionGet

Table 1289:Parameters

Name	Direction	Description
p_version	out	Pointer to return version information to.

Parameter p_version

8.33.8.17 infoGet

```
ssp_err_t(* sdmmc_api_t::infoGet) (sdmmc_ctrl_t *const p_ctrl, sdmmc_info_t
*const p_info)
```

Detailed description

Get SD/MMC device info.

Implemented asR_SDMMC_InfoGet

Table 1290:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to an open SD/MMC instance control block.
p_info	out	Pointer to return device information to.

Parameter p_ctrl

Definition: `sdmmc_ctrl_t*const p_ctrl`

SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls. Implemented assdmmc_instance_ctrl_t

Parameter p_info

Definition: `sdmmc_info_t*const p_info`

Status and other information obtained from the media device.

- `sdmmc_info_t::card_type`

  SD or eMMC.

- `sdmmc_info_t::ready`

  False if card was removed (only applies if MCU supports card detection and SDnCD pin is connected). True otherwise. If ready is false, the driver must be closed, then reopened with a card inserted.

- `sdmmc_info_t::hc`

  true = Card is High Capacity card

- `sdmmc_info_t::sdio`

  true = SDIO present

- `sdmmc_info_t::write_protected`

  true = Card is write protected

- `sdmmc_info_t::transfer_in_progress`

  true = Card is busy

- `sdmmc_info_t::csd_version`

  CSD version.

- `sdmmc_info_t::device_type`

  Speed and data rate (eMMC)

- `sdmmc_info_t::bus_width`

  Current media bus width.

- sdmmc_info_t::hs_timing

  High Speed status.
- sdmmc_info_t::sdhi_rca

  Relative Card Address.
- sdmmc_info_t::max_clock_rate

  Maximum clock rate for media card.
- sdmmc_info_t::clock_rate

  Current clock rate.
- sdmmc_info_t::sector_size

  Sector size.
- sdmmc_info_t::sector_count

  Sector count.
- sdmmc_info_t::erase_sector_count

  Minimum erasable unit (in 512 byte sectors)

8.33.8.18 erase

```
ssp_err_t(* sdmmc_api_t::erase) (sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector, uint32_t const sector_count)
```

Detailed description

Erase SD/MMC sectors. The sector size for erase is fixed at 512 bytes. This API is not supported for SDIO devices.

Implemented asR_SDMMC_Erase

Table 1291:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to an open SD/MMC instance control block.
start_sector	in	First sector to erase. Must be a multiple of erase_sector_count.
sector_count	in	Number of sectors to erase. Must be a multiple of erase_sector_count. All sectors must be in the range of sector_count.

Parameter p_ctrl

Definition: sdmmc_ctrl_t*const p_ctrl

SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls. Implemented assdmmc_instance_ctrl_t

Parameter start_sector

`uint32_t`

Parameter sector_count

`uint32_t`

8.33.8.19 sdmmc_instance_t

sdmmc_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- sdmmc_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- sdmmc_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- sdmmc_api_t const * p_api

  Pointer to the API structure for this instance.

8.34 SLCDC Interface

Interface for Segment LCD controllers.

8.34.1 Summary

This driver uses the Segment LCD controller (SLCDC) to display data on a Segment LCD.

Implemented by: SLCDC

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

SLCDC Interface description: Segment LCD Driver

8.34.2 Interface API

slcdc_api_t

Function name	Description
.open	Open SLCD device.
.write	Write data to SLCD segment. Specifies the initial display data. Except for 8-time slice, store values in the lower 4 bits when writing to the A-pattern area, and in the upper 4 bits when writing to the B-pattern area. The display data is stored in the display data register.
.modify	Rewrite data in the SLCD segment. Rewrites the LCD display data in 1-bit units. If a bit is not specified for rewriting, the value stored in the bit is held as it is. Specifies the data to rewrite
.start	Enable display on the SLCD. Displays the specified data on the LCD. Before that data should be written to the segments.
.stop	Disable display on the SLCD. Stops displaying data on the SLCD.
.contrastIncrease	Increase the display contrast. Increase by 1 unit. This function can be selected when the internal voltage boosting method is used for the drive voltage generator
.contrastDecrease	Decrease the display contrast. Decrease by 1 unit. This function can be selected when the internal voltage boosting method is used for the drive voltage generator
.setdisplayArea	Set LCD display area. This function sets a specified display area, A-pattern or B-pattern. This function can be used to set blink on where A-pattern and B-pattern area data will be alternately displayed.When using blinking, the RTC is required to operate before this function is executed. To configure the RTC, follow the steps below. 1) Open RTC 2) Set Periodic interrupt request, 1/2 second 3) Start RTC counter 4) Enable IRQ, RTC_EVENT_PERIODIC_IRQ Refer to the User's Manual: Hardware for the detailed procedure.
.close	Close display device.
.versionGet	Get version.

8.34.3 Data structures

- slcdc_cfg_t

- slcdc_instance_t

8.34.4 Enumerations

- slcdc_display_state_t
- slcdc_bias_method_t
- slcdc_time_slice_t
- slcdc_wave_form_t
- slcdc_drive_volt_gen_t
- slcdc_display_area_control_blink_t
- slcdc_display_area_t
- slcdc_display_on_off_t
- slcdc_display_enable_disable_t
- slcdc_display_clock_t
- slcdc_clk_div_t

8.34.5 Typedefs

- slcdc_size_t
- slcdc_ctrl_t

8.34.6 Defines

- #define SLCDC_API_VERSION_MAJOR

  Initial value: `(1U)`

  Register definitions, common services and error codes.
- #define SLCDC_API_VERSION_MINOR

  Initial value: `(3U)`
- #define SLCDC_VOL_MIN

  Initial value: `(4)`
- #define SLCDC_VOL_MAX

  Initial value: `(19)`
- #define SLCDC_VOL_MAX_4BIAS

  Initial value: `(10)`
- #define SLCDC_CFG_REF_VCC

  Initial value: `(12)`

- #define SLCDC_BOOST_COUNTER

  Initial value: `(5) /* The number of times of performing waiting for 100 ms */`
- #define SLCDC_BOOST_WAIT

  Initial value: `(uint32_t)(100000) /* Waiting for 100ms */`
- #define SLCDC_SETUP_WAIT

  Initial value: `(5) /* Waiting for 5ms */`
- #define MAX_NUM_SEG

  Initial value: `(51U)`

8.34.7 API Data

8.34.7.1 slcdc_display_state_t

`slcdc_display_state_t`

Detailed description

Display interface operation state

Enumerated values

Name	Description
SLCDC_DISPLAY_STATE_CLOSED	Display closed.
SLCDC_DISPLAY_STATE_OPENED	Display opened.

8.34.7.2 slcdc_bias_method_t

`slcdc_bias_method_t`

Detailed description

LCD display bias method.

Enumerated values

Name	Description
SLCDC_BIAS_2	1/2 bias method
SLCDC_BIAS_3	1/3 bias method
SLCDC_BIAS_4	1/4 bias method

8.34.7.3 slcdc_time_slice_t

`slcdc_time_slice_t`

Detailed description

Time slice of LCD display.

Enumerated values

Name	Description
SLCDC_STATIC	Static.
SLCDC_SLICE_2	2-time slice
SLCDC_SLICE_3	3-time slice
SLCDC_SLICE_4	4-time slice
SLCDC_SLICE_8	8-time slice

8.34.7.4 slcdc_wave_form_t

`slcdc_wave_form_t`

Detailed description

LCD display waveform select.

Enumerated values

Name	Description
SLCDC_WAVE_A	Waveform A.
SLCDC_WAVE_B	Waveform B.

8.34.7.5 slcdc_drive_volt_gen_t

`slcdc_drive_volt_gen_t`

Detailed description

LCD Drive Voltage Generator Select.

Enumerated values

Name	Description
SLCDC_VOLT_EXTERNAL	External resistance division method.
SLCDC_VOLT_INTERNAL	Internal voltage boosting method.
SLCDC_VOLT_CAPACITOR	Capacitor split method.

8.34.7.6 slcdc_display_area_control_blink_t

`slcdc_display_area_control_blink_t`

Detailed description

Display Data Area Control

Enumerated values

Name	Description
SLCDC_NOT_BLINKING	Alternately displaying A-pattern and B-pattern area data.
SLCDC_BLINKING	Displaying an A-pattern or B-pattern area data.

8.34.7.7 slcdc_display_area_t

`slcdc_display_area_t`

Detailed description

Display Area data

Enumerated values

Name	Description
SLCDC_DISP_A	Displaying an A-pattern area data.
SLCDC_DISP_B	Displaying an B-pattern area data.
SLCDC_DISP_BLINK	

8.34.7.8 slcdc_display_on_off_t

`slcdc_display_on_off_t`

Detailed description

LCD Display Enable/Disable

Enumerated values

Name	Description
SLCDC_DISP_OFF	Display off.
SLCDC_DISP_ON	Display on.

8.34.7.9 slcdc_display_enable_disable_t

`slcdc_display_enable_disable_t`

Detailed description

LCD Display output enable

Enumerated values

Name	Description
SLCDC_DISP_DISABLE	Output ground level to segment/common pin.
SLCDC_DISP_ENABLE	Output enable.

8.34.7.10 slcdc_display_clock_t

`slcdc_display_clock_t`

Detailed description

LCD Display clock selection

Enumerated values

Name	Description
SLCDC_CLOCK_LOCO	Display clock source LOCO.
SLCDC_CLOCK_SOSC	Display clock source SOSC.
SLCDC_CLOCK_MOSC	Display clock source MOSC.
SLCDC_CLOCK_HOCO	Display clock source HOCO.

8.34.7.11 slcdc_clk_div_t

`slcdc_clk_div_t`

Detailed description

LCD clock settings

Enumerated values

Name	Description
SLCDC_CLK_DIVISOR_LOCO_4	LOCO Clock/4.
SLCDC_CLK_DIVISOR_LOCO_8	LOCO Clock/8.
SLCDC_CLK_DIVISOR_LOCO_16	LOCO Clock/16.
SLCDC_CLK_DIVISOR_LOCO_32	LOCO Clock/32.
SLCDC_CLK_DIVISOR_LOCO_64	LOCO Clock/64.
SLCDC_CLK_DIVISOR_LOCO_128	LOCO Clock/128.
SLCDC_CLK_DIVISOR_LOCO_256	LOCO Clock/256.
SLCDC_CLK_DIVISOR_LOCO_512	LOCO Clock/512.
SLCDC_CLK_DIVISOR_LOCO_1024	LOCO Clock/1024.
SLCDC_CLK_DIVISOR_HOCO_256	HOCO Clock/256.
SLCDC_CLK_DIVISOR_HOCO_512	HOCO Clock/512.
SLCDC_CLK_DIVISOR_HOCO_1024	HOCO Clock/1024.
SLCDC_CLK_DIVISOR_HOCO_2048	HOCO Clock/2048.
SLCDC_CLK_DIVISOR_HOCO_4096	HOCO Clock/4096.
SLCDC_CLK_DIVISOR_HOCO_8192	HOCO Clock/8192.
SLCDC_CLK_DIVISOR_HOCO_16384	HOCO Clock/16384.
SLCDC_CLK_DIVISOR_HOCO_32768	HOCO Clock/32768.
SLCDC_CLK_DIVISOR_HOCO_65536	HOCO Clock/65536.
SLCDC_CLK_DIVISOR_HOCO_131072	HOCO Clock/131072.
SLCDC_CLK_DIVISOR_HOCO_262144	HOCO Clock/262144.

Name	Description
SLCDC_CLK_DIVISOR_HOCO_524288	HOCO Clock/524288.

8.34.7.12 slcdc_size_t

```
typedef uint8_t slcdc_size_t
```

Detailed description

Size definition for slcdc

8.34.7.13 slcdc_ctrl_t

```
typedef void slcdc_ctrl_t
```

Detailed description

SLCDC control block. Allocate an instance specific control block to pass into the SLCDC API calls. Implemented as

- slcdc_instance_ctrl_tSLCDC control block

8.34.8 API Structures

8.34.8.1 slcdc_cfg_t

slcdc_cfg_t

Detailed description

SLCDC configuration block

Variables

- slcdc_display_clock_t slcdc_clock
  LCD clock source (LCDSCKSEL)
- slcdc_clk_div_t slcdc_clock_setting
  LCD clock setting (LCDC0)
- slcdc_bias_method_t bias_method
  LCD display bias method select (LBAS bit).
- slcdc_time_slice_t time_slice
  Time slice of LCD display select (LDTY bit)
- slcdc_wave_form_t wave_form
  LCD display waveform select (LWAVE bit).
- slcdc_drive_volt_gen_t drive_volt_gen
  LCD Drive Voltage Generator Select (MDSTET bit).

8.34.8.2 slcdc_api_t

slcdc_api_t

Detailed description

SLCDC functions implemented at the HAL layer will follow this API.

8.34.8.3 open

```
ssp_err_t(* slcdc_api_t::open) (slcdc_ctrl_t *const p_ctrl, slcdc_cfg_t const
*const p_cfg)
```

Detailed description

Open SLCD device. Implemented as

- R_SLCDC_Open

Table 1292:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to display interface control block. Must be declared by user. Value set here.
p_cfg	in	Pointer to display configuration structure. All elements of this structure must be set by user.

Parameter p_ctrl

Definition: `slcdc_ctrl_t*const p_ctrl`

SLCDC control block. Allocate an instance specific control block to pass into the SLCDC API calls. Implemented asslcdc_instance_ctrl_tSLCDC control block

Parameter p_cfg

Definition: `slcdc_cfg_t const *const p_cfg`

SLCDC configuration block

- `slcdc_cfg_t`::`slcdc_display_clock_t`

  LCD clock source (LCDSCKSEL)

  Enumerated as:

  – SLCDC_CLOCK_LOCO

  – SLCDC_CLOCK_SOSC

  – SLCDC_CLOCK_MOSC

  – SLCDC_CLOCK_HOCO

- slcdc_cfg_t::slcdc_clk_div_t

  LCD clock setting (LCDC0)

  Enumerated as:

  – SLCDC_CLK_DIVISOR_LOCO_4
  – SLCDC_CLK_DIVISOR_LOCO_8
  – SLCDC_CLK_DIVISOR_LOCO_16
  – SLCDC_CLK_DIVISOR_LOCO_32
  – SLCDC_CLK_DIVISOR_LOCO_64
  – SLCDC_CLK_DIVISOR_LOCO_128
  – SLCDC_CLK_DIVISOR_LOCO_256
  – SLCDC_CLK_DIVISOR_LOCO_512
  – SLCDC_CLK_DIVISOR_LOCO_1024
  – SLCDC_CLK_DIVISOR_HOCO_256
  – SLCDC_CLK_DIVISOR_HOCO_512
  – SLCDC_CLK_DIVISOR_HOCO_1024
  – SLCDC_CLK_DIVISOR_HOCO_2048
  – SLCDC_CLK_DIVISOR_HOCO_4096
  – SLCDC_CLK_DIVISOR_HOCO_8192
  – SLCDC_CLK_DIVISOR_HOCO_16384
  – SLCDC_CLK_DIVISOR_HOCO_32768
  – SLCDC_CLK_DIVISOR_HOCO_65536
  – SLCDC_CLK_DIVISOR_HOCO_131072
  – SLCDC_CLK_DIVISOR_HOCO_262144
  – SLCDC_CLK_DIVISOR_HOCO_524288

- slcdc_cfg_t::slcdc_bias_method_t

  LCD display bias method select (LBAS bit).

  Enumerated as:

  – SLCDC_BIAS_2
  – SLCDC_BIAS_3
  – SLCDC_BIAS_4

- slcdc_cfg_t::slcdc_time_slice_t

  Time slice of LCD display select (LDTY bit)

  Enumerated as:

  – SLCDC_STATIC

  – SLCDC_SLICE_2

  – SLCDC_SLICE_3

  – SLCDC_SLICE_4

  – SLCDC_SLICE_8

- slcdc_cfg_t::slcdc_wave_form_t

  LCD display waveform select (LWAVE bit).

  Enumerated as:

  – SLCDC_WAVE_A

  – SLCDC_WAVE_B

- slcdc_cfg_t::slcdc_drive_volt_gen_t

  LCD Drive Voltage Generator Select (MDSTET bit).

  Enumerated as:

  – SLCDC_VOLT_EXTERNAL

  – SLCDC_VOLT_INTERNAL

  – SLCDC_VOLT_CAPACITOR

8.34.8.4 write

```
ssp_err_t(* slcdc_api_t::write) (slcdc_ctrl_t *const p_ctrl, slcdc_size_t const
start_segment, slcdc_size_t const *const p_data, slcdc_size_t const segment_count)
```

Detailed description

Write data to SLCD segment. Specifies the initial display data. Except for 8-time slice, store values in the lower 4 bits when writing to the A-pattern area, and in the upper 4 bits when writing to the B-pattern area. The display data is stored in the display data register. Implemented as

- R_SLCDC_Write

Table 1293:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to display interface control block.

Table 1293:Parameters (Continued)

Name	Direction	Description
start_segment	in	Specify the start segment number to be written.
p_data	in	pointer to the display data to be written to the specified segments
segment_count	in	Number of segments to be written

Parameter p_ctrl

Definition: `slcdc_ctrl_t*const p_ctrl`

SLCDC control block. Allocate an instance specific control block to pass into the SLCDC API calls. Implemented asslcdc_instance_ctrl_tSLCDC control block

Parameter start_segment

Definition: `slcdc_size_tconst start_segment`

Size definition for slcdc

Parameter p_data

Definition: `slcdc_size_tconst *const p_data`

Size definition for slcdc

Parameter segment_count

Definition: `slcdc_size_tconst segment_count`

Size definition for slcdc

8.34.8.5 modify

```
ssp_err_t(* slcdc_api_t::modify) (slcdc_ctrl_t *const p_ctrl, slcdc_size_t const
segment, slcdc_size_t const data_mask, slcdc_size_t const data)
```

Detailed description

Rewrite data in the SLCD segment. Rewrites the LCD display data in 1-bit units. If a bit is not specified for rewriting, the value stored in the bit is held as it is. Specifies the data to rewrite Implemented as

- R_SLCDC_Modify

Table 1294:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to display interface control block.
seg	in	Specify the segment to be written.

Table 1294:Parameters (Continued)

Name	Direction	Description
data_mask	in	Mask the data being displayed. Set 0 to the bit to be rewritten and set 1 to the other bits. Multiple bits can be rewritten. Setting value of data_mask, Bit 3 - 0xf7 Bit 2 - 0xfb Bit 1 - 0xfd Bit 0 - 0xfe
data	in	Specify display data to rewrite to the specified segment.

Parameter p_ctrl

Definition: `slcdc_ctrl_t*const p_ctrl`

SLCDC control block. Allocate an instance specific control block to pass into the SLCDC API calls. Implemented asslcdc_instance_ctrl_tSLCDC control block

Parameter seg

Definition: `slcdc_size_tconst segment`

Size definition for slcdc

Parameter data_mask

Definition: `slcdc_size_tconst data_mask`

Size definition for slcdc

Parameter data

Definition: `slcdc_size_tconst data`

Size definition for slcdc

8.34.8.6 start

```
ssp_err_t(* slcdc_api_t::start) (slcdc_ctrl_t *const p_ctrl)
```

Detailed description

Enable display on the SLCD. Displays the specified data on the LCD. Before that data should be written to the segments. Implemented as

- R_SLCDC_Start

Table 1295:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to display interface control block.

Parameter p_ctrl

Definition: `slcdc_ctrl_t*const p_ctrl`

SLCDC control block. Allocate an instance specific control block to pass into the SLCDC API calls. Implemented asslcdc_instance_ctrl_tSLCDC control block

8.34.8.7 stop

```
ssp_err_t(* slcdc_api_t::stop) (slcdc_ctrl_t *const p_ctrl)
```

Detailed description

Disable display on the SLCD. Stops displaying data on the SLCD. Implemented as

- R_SLCDC_Stop

Table 1296:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to display interface control block.

Parameter p_ctrl

Definition: `slcdc_ctrl_t*const p_ctrl`

SLCDC control block. Allocate an instance specific control block to pass into the SLCDC API calls. Implemented asslcdc_instance_ctrl_tSLCDC control block

8.34.8.8 contrastIncrease

```
ssp_err_t(* slcdc_api_t::contrastIncrease) (slcdc_ctrl_t *const p_ctrl)
```

Detailed description

Increase the display contrast. Increase by 1 unit. This function can be selected when the internal voltage boosting method is used for the drive voltage generator Implemented as

- R_SLCDC_ContrastIncrease

Table 1297:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to display interface control block.

Parameter p_ctrl

Definition: `slcdc_ctrl_t*const p_ctrl`

SLCDC control block. Allocate an instance specific control block to pass into the SLCDC API calls. Implemented asslcdc_instance_ctrl_tSLCDC control block

8.34.8.9 contrastDecrease

```
ssp_err_t(* slcdc_api_t::contrastDecrease) (slcdc_ctrl_t *const p_ctrl)
```

Detailed description

Decrease the display contrast. Decrease by 1 unit. This function can be selected when the internal voltage boosting method is used for the drive voltage generator Implemented as

- R_SLCDC_ContrastDecrease

Table 1298:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to display interface control block.

Parameter p_ctrl

Definition: `slcdc_ctrl_t*const p_ctrl`

SLCDC control block. Allocate an instance specific control block to pass into the SLCDC API calls. Implemented asslcdc_instance_ctrl_tSLCDC control block

8.34.8.10 setdisplayArea

```
ssp_err_t(* slcdc_api_t::setdisplayArea) (slcdc_ctrl_t *const p_ctrl,
slcdc_display_area_t const display_area)
```

Detailed description

Set LCD display area. This function sets a specified display area, A-pattern or B-pattern. This function can be used to set blink on where A-pattern and B-pattern area data will be alternately displayed.

When using blinking, the RTC is required to operate before this function is executed. To configure the RTC, follow the steps below. 1) Open RTC 2) Set Periodic interrupt request, 1/2 second 3) Start RTC counter 4) Enable IRQ, RTC_EVENT_PERIODIC_IRQ Refer to the User's Manual: Hardware for the detailed procedure.

Implemented as

- R_SLCDC_SetdisplayArea()

Table 1299:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to display interface control block.

Table 1299:Parameters (Continued)

Name	Direction	Description
display_area	in	Display area to be used, A-pattern or B-pattern area.

Parameter p_ctrl

Definition: `slcdc_ctrl_t*const p_ctrl`

SLCDC control block. Allocate an instance specific control block to pass into the SLCDC API calls. Implemented asslcdc_instance_ctrl_tSLCDC control block

Parameter display_area

8.34.8.11 close

```
ssp_err_t(* slcdc_api_t::close) (slcdc_ctrl_t *const p_ctrl)
```

Detailed description

Close display device. Implemented as

- R_SLCDC_Close

Table 1300:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to display interface control block.

Parameter p_ctrl

Definition: `slcdc_ctrl_t*const p_ctrl`

SLCDC control block. Allocate an instance specific control block to pass into the SLCDC API calls. Implemented asslcdc_instance_ctrl_tSLCDC control block

8.34.8.12 versionGet

```
ssp_err_t(* slcdc_api_t::versionGet) (ssp_version_t *p_version)
```

Detailed description

Get version. Implemented as

- R_SLCDC_VersionGet

Table 1301:Parameters

Name	Direction	Description
p_version	in	Pointer to the memory to store the version information.

Parameter p_version

8.34.8.13 slcdc_instance_t

slcdc_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- slcdc_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- slcdc_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- slcdc_api_t const * p_api

  Pointer to the API structure for this instance.

8.35 SPI Interface

Interface for SPI communications.

8.35.1 Summary

The SPI Interface provides access to the SPI bus API. The Interface implements the Simple SPI on SCI HAL layer driver module.

Implemented by:

- SPI
- Simple SPI on SCI

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

SPI Interface description: SPI Driver

8.35.2 Interface API

spi_api_t

Function name	Description
.open	Initialize a channel for SPI communication mode.
.read	Receive data from an SPI device.
.write	Transmit data to an SPI device.
.writeRead	Simultaneously transmit data to an SPI device while receiving data from a SPI device (full duplex).
.close	Remove power to the SPI channel designated by the handle and disable the associated interrupts.
.versionGet	Get the version information of the underlying driver.

8.35.3 Data structures

- spi_callback_args_t
- spi_cfg_t
- spi_instance_t

8.35.4 Enumerations

- spi_bit_width_t
- spi_mode_t
- spi_clk_phase_t
- spi_clk_polarity_t
- spi_mode_fault_t
- spi_bit_order_t
- spi_event_t
- spi_operation_t

8.35.5 Typedefs

- spi_ctrl_t

8.35.6 Defines

- #define SPI_API_VERSION_MAJOR

  Initial value: `(1U)`
- #define SPI_API_VERSION_MINOR

  Initial value: `(4U)`

8.35.7 API Data

8.35.7.1 spi_bit_width_t

`spi_bit_width_t`

Detailed description

Data bit width

Enumerated values

Name	Description
SPI_BIT_WIDTH_8_BITS	Data bit width is 8 bits byte.
SPI_BIT_WIDTH_16_BITS	Data bit width is 16 bits word.
SPI_BIT_WIDTH_32_BITS	Data bit width is 32 bits long word.

8.35.7.2 spi_mode_t

`spi_mode_t`

Detailed description

Master or slave operating mode

Enumerated values

Name	Description
SPI_MODE_MASTER	Channel operates as SPI master.
SPI_MODE_SLAVE	Channel operates as SPI slave.

8.35.7.3 spi_clk_phase_t

`spi_clk_phase_t`

Detailed description

Clock phase

Enumerated values

Name	Description
SPI_CLK_PHASE_EDGE_ODD	0: Data sampling on odd edge, data variation on even edge
SPI_CLK_PHASE_EDGE_EVEN	1: Data variation on odd edge, data sampling on even edge

8.35.7.4 spi_clk_polarity_t

`spi_clk_polarity_t`

Detailed description

Clock polarity

Enumerated values

Name	Description
SPI_CLK_POLARITY_LOW	0: Clock polarity is low when idle
SPI_CLK_POLARITY_HIGH	1: Clock polarity is high when idle

8.35.7.5 spi_mode_fault_t

`spi_mode_fault_t`

Detailed description

Mode fault error flag. This error occurs when the device is setup as a master, but the SSLA line does not seem to be controlled by the master. This usually happens when the connecting device is also acting as master. A similar situation can also happen when configured as a slave.

Enumerated values

Name	Description
SPI_MODE_FAULT_ERROR_ENABLE	Mode fault error flag on.

Name	Description
SPI_MODE_FAULT_ERROR_DISABLE	Mode fault error flag off.

8.35.7.6 spi_bit_order_t

`spi_bit_order_t`

Detailed description

Bit-order

Enumerated values

Name	Description
SPI_BIT_ORDER_MSB_FIRST	Send MSB first in transmission.
SPI_BIT_ORDER_LSB_FIRST	Send LSB first in transmission.

8.35.7.7 spi_event_t

`spi_event_t`

Detailed description

SPI events

Enumerated values

Name	Description
SPI_EVENT_TRANSFER_COMPLETE	The data transfer was completed.
SPI_EVENT_TRANSFER_ABORTED	The data transfer was aborted.
SPI_EVENT_ERR_MODE_FAULT	Mode fault error.
SPI_EVENT_ERR_READ_OVERFLOW	Read overflow error.
SPI_EVENT_ERR_PARITY	Parity error.
SPI_EVENT_ERR_OVERRUN	Overrun error.
SPI_EVENT_ERR_FRAMING	Framing error.
SPI_EVENT_ERR_MODE_UNDERRUN	Underrun error.

8.35.7.8 spi_operation_t

```
spi_operation_t
```

Detailed description

Used by control block only.

Enumerated values

Name	Description
SPI_OPERATION_DO_TX	perform SPI transmission operation
SPI_OPERATION_DO_RX	perform SPI reception operation
SPI_OPERATION_DO_TX_RX	perform SPI Transmission and reception operation

8.35.7.9 spi_ctrl_t

```
typedef void spi_ctrl_t
```

Detailed description

SPI control block. Allocate an instance specific control block to pass into the SPI API calls. Implemented as

- sci_spi_instance_ctrl_t
- rspi_instance_ctrl_t

8.35.8 API Structures

8.35.8.1 spi_callback_args_t

spi_callback_args_t

Detailed description

Common callback parameter definition

Variables

- uint32_t channel

  Device channel number.

- spi_event_t event

  Event code.

- void const * p_context

  Context provided to user during callback.

8.35.8.2 spi_cfg_t

spi_cfg_t

Detailed description

SPI interface configuration

Variables

- uint8_t channel

  Channel number to be used.
- uint8_t rxi_ipl

  Receive interrupt priority.
- uint8_t txi_ipl

  Transmit interrupt priority.
- uint8_t tei_ipl

  Transmit end interrupt priority.
- uint8_t eri_ipl

  Error interrupt priority.
- spi_mode_t operating_mode

  Select master or slave operating mode.
- spi_clk_phase_t clk_phase

  Data sampling on odd or even clock edge.
- spi_clk_polarity_t clk_polarity

  Clock level when idle.
- spi_mode_fault_t mode_fault

  Mode fault error (master/slave conflict) flag.
- spi_bit_order_t bit_order

  Select to transmit MSB/LSB first.
- uint32_t bitrate

  Bits Per Second.
- transfer_instance_t const * p_transfer_tx

  To use SPI DTC/DMA write transfer, link a DTC/DMA instance here. Set to NULL if unused.
- transfer_instance_t const * p_transfer_rx

  To use SPI DTC/DMA read transfer, link a DTC/DMA instance here. Set to NULL if unused.
- void(* p_callback)(spi_callback_args_t *p_args)

  Pointer to user callback function.

- void const * p_context

  User defined context passed to callback function.

- void const * p_extend

  Extended SPI hardware dependent configuration.

8.35.8.3 spi_api_t

spi_api_t

Detailed description

Shared Interface definition for SPI

8.35.8.4 open

```
ssp_err_t(* spi_api_t::open) (spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg)
```

Detailed description

Initialize a channel for SPI communication mode. Implemented as

- R_RSPI_Open
- R_SCI_SPI_Open

Table 1302:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to user-provided storage for the control block.
p_cfg	in	Pointer to SPI configuration structure.

Parameter p_ctrl

Definition: `spi_ctrl_t*p_ctrl`

SPI control block. Allocate an instance specific control block to pass into the SPI API calls. Implemented assci_spi_instance_ctrl_trspi_instance_ctrl_t

Parameter p_cfg

Definition: `spi_cfg_t const *const p_cfg`

SPI interface configuration

- `spi_cfg_t::channel`

  Channel number to be used.

- `spi_cfg_t::rxi_ipl`

  Receive interrupt priority.

- spi_cfg_t::txi_ipl

  Transmit interrupt priority.

- spi_cfg_t::tei_ipl

  Transmit end interrupt priority.

- spi_cfg_t::eri_ipl

  Error interrupt priority.

- spi_cfg_t::spi_mode_t

  Select master or slave operating mode.

  Enumerated as:

  – SPI_MODE_MASTER

  – SPI_MODE_SLAVE

- spi_cfg_t::spi_clk_phase_t

  Data sampling on odd or even clock edge.

  Enumerated as:

  – SPI_CLK_PHASE_EDGE_ODD

  – SPI_CLK_PHASE_EDGE_EVEN

- spi_cfg_t::spi_clk_polarity_t

  Clock level when idle.

  Enumerated as:

  – SPI_CLK_POLARITY_LOW

  – SPI_CLK_POLARITY_HIGH

- spi_cfg_t::spi_mode_fault_t

  Mode fault error (master/slave conflict) flag.

  Enumerated as:

  – SPI_MODE_FAULT_ERROR_ENABLE

  – SPI_MODE_FAULT_ERROR_DISABLE

- spi_cfg_t::spi_bit_order_t

  Select to transmit MSB/LSB first.

  Enumerated as:

  – SPI_BIT_ORDER_MSB_FIRST

  – SPI_BIT_ORDER_LSB_FIRST

- spi_cfg_t::bitrate

  Bits Per Second.

- spi_cfg_t::transfer_instance_t

  To use SPI DTC/DMA write transfer, link a DTC/DMA instance here. Set to NULL if unused.

- spi_cfg_t::transfer_instance_t

  To use SPI DTC/DMA read transfer, link a DTC/DMA instance here. Set to NULL if unused.

- spi_cfg_t::p_callback

  Pointer to user callback function.

- spi_cfg_t::p_context

  User defined context passed to callback function.

- spi_cfg_t::p_extend

  Extended SPI hardware dependent configuration.

8.35.8.5 read

```
ssp_err_t(* spi_api_t::read) (spi_ctrl_t *const p_ctrl, void const *p_dest,
uint32_t const length, spi_bit_width_t const bit_width)
```

Detailed description

Receive data from an SPI device. Implemented as

- R_RSPI_Read
- R_SCI_SPI_Read

Table 1303:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the channel.
length	in	Number of units of data to be transferred (unit size specified by the bit_width).
bit_width	in	Data bit width to be transferred.
p_dest	out	Pointer to destination buffer into which data will be copied that is received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count.

Parameter p_ctrl

Definition: `spi_ctrl_t*const p_ctrl`

SPI control block. Allocate an instance specific control block to pass into the SPI API calls. Implemented assci_spi_instance_ctrl_trspi_instance_ctrl_t

Parameter length

`uint32_t`

Parameter bit_width

Parameter p_dest

`const`

8.35.8.6 write

```
ssp_err_t(* spi_api_t::write) (spi_ctrl_t *const p_ctrl, void const *p_src,
uint32_t const length, spi_bit_width_t const bit_width)
```

Detailed description

Transmit data to an SPI device. Implemented as

- R_RSPI_Write
- R_SCI_SPI_Write

Table 1304:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the channel.
p_src	in	Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL.
length	in	Number of units of data to be transferred (unit size specified by the bit_width).
bit_width	in	Data bit width to be transferred.

Parameter p_ctrl

Definition: `spi_ctrl_t*const p_ctrl`

SPI control block. Allocate an instance specific control block to pass into the SPI API calls. Implemented assci_spi_instance_ctrl_trspi_instance_ctrl_t

Parameter p_src

`const`

Parameter length

uint32_t

Parameter bit_width

8.35.8.7 writeRead

```
ssp_err_t(* spi_api_t::writeRead) (spi_ctrl_t *const p_ctrl, void const *p_src,
void const *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
```

Detailed description

Simultaneously transmit data to an SPI device while receiving data from a SPI device (full duplex). Implemented as

- R_RSPI_WriteRead
- R_SCI_SPI_WriteRead

Table 1305:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the channel.
p_src	in	Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL.
p_dest	out	Pointer to destination buffer into which data will be copied that is received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count. The argument must not be NULL.
length	in	Number of units of data to be transferred (unit size specified by the bit_width).
bit_width	in	Data bit width to be transferred.

Parameter p_ctrl

Definition: spi_ctrl_t*const p_ctrl

SPI control block. Allocate an instance specific control block to pass into the SPI API calls. Implemented assci_spi_instance_ctrl_trspi_instance_ctrl_t

Parameter p_src

const

Parameter p_dest

`const`

Parameter length

`uint32_t`

Parameter bit_width

8.35.8.8 close

```
ssp_err_t(* spi_api_t::close) (spi_ctrl_t *const p_ctrl)
```

Detailed description

Remove power to the SPI channel designated by the handle and disable the associated interrupts. Implemented as

- R_RSPI_Close
- R_SCI_SPI_Close

Table 1306:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the channel.

Parameter p_ctrl

Definition: `spi_ctrl_t*const p_ctrl`

SPI control block. Allocate an instance specific control block to pass into the SPI API calls. Implemented assci_spi_instance_ctrl_trspi_instance_ctrl_t

8.35.8.9 versionGet

```
ssp_err_t(* spi_api_t::versionGet) (ssp_version_t *p_version)
```

Detailed description

Get the version information of the underlying driver. Implemented as

- R_RSPI_VersionGet
- R_SCI_SPI_VersionGet

Table 1307:Parameters

Name	Direction	Description
p_version	out	pointer to memory location to return version number

Parameter p_version

8.35.8.10 spi_instance_t

spi_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- spi_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- spi_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- spi_api_t const * p_api

  Pointer to the API structure for this instance.

8.36 Timer Interface

Interface for timer functions.

8.36.1 Summary

The general timer interface provides standard timer functionality including periodic mode, one-shot mode, and free-running timer mode. After each timer cycle (overflow or underflow), an interrupt can be triggered.

If an instance supports output compare mode, it is provided in the extension configuration timer_on_<instance>_cfg_t defined in r_<instance>.h.

Implemented by:

- GPT
- AGT

See also: Input Capture Interface

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Timer Interface description: Timer Driver

8.36.2 Interface API

timer_api_t

Function name	Description
.open	Initial configuration.
.stop	Stop the counter.
.start	Start the counter.
.reset	Reset the counter to the initial value.
.counterGet	Get current counter value and store it in provided pointer p_value.
.periodSet	Set the time until the timer expires.
.dutyCycleSet	Sets the time until the duty cycle expires.
.infoGet	Get the time until the timer expires in clock counts and store it in provided pointer p_period_counts.
.close	Allows driver to be reconfigured and may reduce power consumption.
.versionGet	Get version and store it in provided pointer p_version.

8.36.3 Data structures

- timer_callback_args_t
- timer_info_t
- timer_cfg_t
- timer_instance_t

8.36.4 Enumerations

- timer_event_t
- timer_variant_t
- timer_status_t
- timer_mode_t
- timer_unit_t
- timer_pwm_unit_t
- timer_direction_t

8.36.5 Typedefs

- timer_size_t
- timer_ctrl_t
- timer_period_t

8.36.6 Defines

- #define TIMER_API_VERSION_MAJOR

  Initial value: `(1U)`
- #define TIMER_API_VERSION_MINOR

  Initial value: `(2U)`

8.36.7 API Data

8.36.7.1 timer_event_t

`timer_event_t`

Detailed description

Events that can trigger a callback function

Enumerated values

Name	Description
TIMER_EVENT_EXPIRED	Requested timer delay has expired or timer has wrapped around.

8.36.7.2 timer_variant_t

`timer_variant_t`

Detailed description

Timer variant types.

Enumerated values

Name	Description
TIMER_VARIANT_32_BIT	32-bit timer
TIMER_VARIANT_16_BIT	16-bit timer

8.36.7.3 timer_status_t

`timer_status_t`

Detailed description

Possible status values returned by infoGet.

Enumerated values

Name	Description
TIMER_STATUS_COUNTING	Timer is running.
TIMER_STATUS_STOPPED	Timer is stopped.

8.36.7.4 timer_mode_t

`timer_mode_t`

Detailed description

Timer operational modes

Enumerated values

Name	Description
TIMER_MODE_PERIODIC	Timer will restart after delay periods.
TIMER_MODE_ONE_SHOT	Timer will stop after delay periods.
TIMER_MODE_PWM	Timer generate PWM output.

8.36.7.5 timer_unit_t

```
timer_unit_t
```

Detailed description

Units of timer period value.

Enumerated values

Name	Description
TIMER_UNIT_PERIOD_RAW_COUNTS	Period in clock counts.
TIMER_UNIT_PERIOD_NSEC	Period in nanoseconds.
TIMER_UNIT_PERIOD_USEC	Period in microseconds.
TIMER_UNIT_PERIOD_MSEC	Period in milliseconds.
TIMER_UNIT_PERIOD_SEC	Period in seconds.
TIMER_UNIT_FREQUENCY_HZ	Frequency in Hz.
TIMER_UNIT_FREQUENCY_KHZ	Frequency in kHz.

8.36.7.6 timer_pwm_unit_t

```
timer_pwm_unit_t
```

Detailed description

Units of timer duty cycle value.

Enumerated values

Name	Description
TIMER_PWM_UNIT_RAW_COUNTS	Period in clock counts.
TIMER_PWM_UNIT_PERCENT	Percent unit used for duty cycle.
TIMER_PWM_UNIT_PERCENT_X_1000	Percent multiplied by 1000 for extra resolution.

8.36.7.7 timer_direction_t

```
timer_direction_t
```

Detailed description

Direction of timer count

Enumerated values

Name	Description
TIMER_DIRECTION_DOWN	Timer count goes up.
TIMER_DIRECTION_UP	Timer count goes down.

8.36.7.8 timer_size_t

```
typedef uint32_t timer_size_t
```

Detailed description

Largest supported timer size. Currently up to 32-bit timers are supported. If a 16-bit timer is used, only the bottom 16 bits of any timer_size_t parameter can be used. Passing in values larger than 16 bits would result in an error.

8.36.7.9 timer_ctrl_t

```
typedef void timer_ctrl_t
```

Detailed description

Timer control block. Allocate an instance specific control block to pass into the timer API calls. Implemented as

- gpt_instance_ctrl_t
- agt_instance_ctrl_t

8.36.7.10 timer_period_t

```
typedef timer_size_t timer_period_t
```

Detailed description

DEPRECATED: Recommend using timer_size_t for period.

8.36.8 API Structures

8.36.8.1 timer_callback_args_t

timer_callback_args_t

Detailed description

Callback function parameter data

Variables

- void const * p_context

  Placeholder for user data. Set in timer_api_t::open function in timer_cfg_t.

- timer_event_t event

  The event can be used to identify what caused the callback (overflow or error).

8.36.8.2 timer_info_t

timer_info_t

Detailed description

Timer information structure to store various information for a timer resource

Variables

- timer_direction_t count_direction

  Clock counting direction of the timer resource.

- uint32_t clock_frequency

  Clock frequency of the timer resource.

- timer_size_t period_counts

  Time in clock counts until timer will expire.

- timer_status_t status
- elc_event_t elc_event

  ELC event associated with the count operation of the timer resource.

8.36.8.3 timer_cfg_t

timer_cfg_t

Detailed description

User configuration structure, used in open function

Variables

- timer_mode_t mode

  Select enumerated value from timer_mode_t.

- uint32_t period

  Defines when the timer should expire. For a free running counter, set to TIMER_MAX_CLOCK with unit TIMER_UNIT_PERIOD_RAW_COUNTS
- timer_unit_t unit

  Units of timer_cfg_t::period.
- timer_size_t duty_cycle

  Duty cycle in units timer_cfg_t::duty_cycle_unit.
- timer_pwm_unit_t duty_cycle_unit

  Units of timer_cfg_t::duty_cycle.
- uint8_t channel

  Select a channel corresponding to the channel number of the hardware.
- uint8_t irq_ipl

  Timer interrupt priority.
- bool autostart

  Whether to start during Open call or not. True: Start during open. False: Don't start during open.
- void(* p_callback)(timer_callback_args_t *p_args)

  Callback provided when a timer ISR occurs. Set to NULL for no CPU interrupt.
- void const * p_context

  Placeholder for user data. Passed to the user callback in timer_callback_args_t.
- void const * p_extend

  Extension parameter for hardware specific settings.

8.36.8.4 timer_api_t

timer_api_t

Detailed description

Timer API structure. General timer functions implemented at the HAL layer follow this API.

8.36.8.5 open

```
ssp_err_t(* timer_api_t::open) (timer_ctrl_t *const p_ctrl, timer_cfg_t const
*const p_cfg)
```

Detailed description

Initial configuration. Implemented as

- R_GPT_TimerOpen
- R_AGT_TimerOpen

NOTE: To reconfigure after calling this function, call close first.

Table 1308:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control block. Must be declared by user. Elements set here.
p_cfg	in	Pointer to configuration structure. All elements of this structure must be set by user.

Parameter p_ctrl

Definition: `timer_ctrl_t*const p_ctrl`

Timer control block. Allocate an instance specific control block to pass into the timer API calls. Implemented asgpt_instance_ctrl_tagt_instance_ctrl_t

Parameter p_cfg

Definition: `timer_cfg_t const *const p_cfg`

User configuration structure, used in open function

- `timer_cfg_t`::`timer_mode_t`

  Select enumerated value from timer_mode_t.

  Enumerated as:

  – TIMER_MODE_PERIODIC

  – TIMER_MODE_ONE_SHOT

  – TIMER_MODE_PWM

- `timer_cfg_t`::`period`

  Defines when the timer should expire. For a free running counter, set to TIMER_MAX_CLOCK with unit TIMER_UNIT_PERIOD_RAW_COUNTS

- `timer_cfg_t`::`timer_unit_t`

  Units of timer_cfg_t::period.

  Enumerated as:

  – TIMER_UNIT_PERIOD_RAW_COUNTS

  – TIMER_UNIT_PERIOD_NSEC

  – TIMER_UNIT_PERIOD_USEC

  - TIMER_UNIT_PERIOD_MSEC
  - TIMER_UNIT_PERIOD_SEC
  - TIMER_UNIT_FREQUENCY_HZ
  - TIMER_UNIT_FREQUENCY_KHZ
- timer_cfg_t::timer_size_t

  Duty cycle in units timer_cfg_t::duty_cycle_unit.
- timer_cfg_t::timer_pwm_unit_t

  Units of timer_cfg_t::duty_cycle.

  Enumerated as:
  - TIMER_PWM_UNIT_RAW_COUNTS
  - TIMER_PWM_UNIT_PERCENT
  - TIMER_PWM_UNIT_PERCENT_X_1000
- timer_cfg_t::channel

  Select a channel corresponding to the channel number of the hardware.
- timer_cfg_t::irq_ipl

  Timer interrupt priority.
- timer_cfg_t::autostart

  Whether to start during Open call or not. True: Start during open. False: Don't start during open.
- timer_cfg_t::p_callback

  Callback provided when a timer ISR occurs. Set to NULL for no CPU interrupt.
- timer_cfg_t::p_context

  Placeholder for user data. Passed to the user callback in timer_callback_args_t.
- timer_cfg_t::p_extend

  Extension parameter for hardware specific settings.

8.36.8.6 stop

```
ssp_err_t(* timer_api_t::stop) (timer_ctrl_t *const p_ctrl)
```

Detailed description

Stop the counter. Implemented as

- R_GPT_Stop
- R_AGT_Stop

Table 1309:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this timer.

Parameter p_ctrl

Definition: `timer_ctrl_t*const p_ctrl`

Timer control block. Allocate an instance specific control block to pass into the timer API calls. Implemented asgpt_instance_ctrl_tagt_instance_ctrl_t

8.36.8.7 start

```
ssp_err_t(* timer_api_t::start) (timer_ctrl_t *const p_ctrl)
```

Detailed description

Start the counter. Implemented as

- R_GPT_Start
- R_AGT_Start

NOTE: The counter can also be started in the open function if autostart is true.

Table 1310:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this timer.

Parameter p_ctrl

Definition: `timer_ctrl_t*const p_ctrl`

Timer control block. Allocate an instance specific control block to pass into the timer API calls. Implemented asgpt_instance_ctrl_tagt_instance_ctrl_t

8.36.8.8 reset

```
ssp_err_t(* timer_api_t::reset) (timer_ctrl_t *const p_ctrl)
```

Detailed description

Reset the counter to the initial value. Implemented as

- R_GPT_Reset
- R_AGT_Reset

Table 1311:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this timer.

Parameter p_ctrl

Definition: `timer_ctrl_t*const p_ctrl`

Timer control block. Allocate an instance specific control block to pass into the timer API calls. Implemented asgpt_instance_ctrl_tagt_instance_ctrl_t

8.36.8.9 counterGet

```
ssp_err_t(* timer_api_t::counterGet) (timer_ctrl_t *const p_ctrl, timer_size_t
*const p_value)
```

Detailed description

Get current counter value and store it in provided pointer p_value. Implemented as

- R_GPT_CounterGet
- R_AGT_CounterGet

Table 1312:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this timer.
p_value	out	Pointer to store current counter value.

Parameter p_ctrl

Definition: `timer_ctrl_t*const p_ctrl`

Timer control block. Allocate an instance specific control block to pass into the timer API calls. Implemented asgpt_instance_ctrl_tagt_instance_ctrl_t

Parameter p_value

Definition: `timer_size_t*const p_value`

Largest supported timer size. Currently up to 32-bit timers are supported. If a 16-bit timer is used, only the bottom 16 bits of any timer_size_t parameter can be used. Passing in values larger than 16 bits would result in an error.

8.36.8.10 periodSet

```
ssp_err_t(* timer_api_t::periodSet) (timer_ctrl_t *const p_ctrl, timer_size_t
const period, timer_unit_t const unit)
```

Detailed description

Set the time until the timer expires. Implemented as

- R_GPT_PeriodSet
- R_AGT_PeriodSet

NOTE: Timer expiration may or may not generate a CPU interrupt based on how the timer is configured in open.

Table 1313:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this timer.
period	in	Time until timer should expire.
unit	in	Units of period parameter.

Parameter p_ctrl

Definition: `timer_ctrl_t*const p_ctrl`

Timer control block. Allocate an instance specific control block to pass into the timer API calls. Implemented asgpt_instance_ctrl_tagt_instance_ctrl_t

Parameter period

Definition: `timer_size_tconst period`

Largest supported timer size. Currently up to 32-bit timers are supported. If a 16-bit timer is used, only the bottom 16 bits of any timer_size_t parameter can be used. Passing in values larger than 16 bits would result in an error.

Parameter unit

8.36.8.11 dutyCycleSet

```
ssp_err_t(* timer_api_t::dutyCycleSet) (timer_ctrl_t *const p_ctrl, timer_size_t
const duty_cycle, timer_pwm_unit_t const duty_cycle_unit, uint8_t const pin)
```

Detailed description

Sets the time until the duty cycle expires.

Table 1314:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this timer.
duty_cycle	in	Time until duty cycle should expire.
duty_cycle_unit	in	Units of duty_cycle parameter.
pin	in	Which output pin to update. Enter the pin number or if pins are identified by letters, enter 0 for A, 1 for B, 2 for C, etc.

Parameter p_ctrl

Definition: `timer_ctrl_t*const p_ctrl`

Timer control block. Allocate an instance specific control block to pass into the timer API calls. Implemented asgpt_instance_ctrl_tagt_instance_ctrl_t

Parameter duty_cycle

Definition: `timer_size_tconst duty_cycle`

Largest supported timer size. Currently up to 32-bit timers are supported. If a 16-bit timer is used, only the bottom 16 bits of any timer_size_t parameter can be used. Passing in values larger than 16 bits would result in an error.

Parameter duty_cycle_unit

Definition: `timer_pwm_unit_tconst duty_cycle_unit`

Units of timer duty cycle value.

Parameter pin

`uint8_t`

8.36.8.12 infoGet

```
ssp_err_t(* timer_api_t::infoGet) (timer_ctrl_t *const p_ctrl, timer_info_t
*const p_info)
```

Detailed description

Get the time until the timer expires in clock counts and store it in provided pointer p_period_counts. Implemented as

- R_GPT_InfoGet
- R_AGT_InfoGet

Table 1315:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this timer.
p_info	out	Collection of information for this timer.

Parameter p_ctrl

Definition: `timer_ctrl_t*const p_ctrl`

Timer control block. Allocate an instance specific control block to pass into the timer API calls. Implemented asgpt_instance_ctrl_tagt_instance_ctrl_t

Parameter p_info

Definition: `timer_info_t*const p_info`

Timer information structure to store various information for a timer resource

- `timer_info_t::count_direction`

  Clock counting direction of the timer resource.
- `timer_info_t::clock_frequency`

  Clock frequency of the timer resource.
- `timer_info_t::period_counts`

  Time in clock counts until timer will expire.
- `timer_info_t::status`
- `timer_info_t::elc_event`

  ELC event associated with the count operation of the timer resource.

8.36.8.13 close

```
ssp_err_t(* timer_api_t::close) (timer_ctrl_t *const p_ctrl)
```

Detailed description

Allows driver to be reconfigured and may reduce power consumption. Implemented as

- R_GPT_Close
- R_AGT_Close

Table 1316:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this timer.

Parameter p_ctrl

Definition: `timer_ctrl_t*const p_ctrl`

Timer control block. Allocate an instance specific control block to pass into the timer API calls. Implemented asgpt_instance_ctrl_tagt_instance_ctrl_t

8.36.8.14 versionGet

```
ssp_err_t(* timer_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Get version and store it in provided pointer p_version. Implemented as

- R_GPT_VersionGet
- R_AGT_VersionGet

Table 1317:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.36.8.15 timer_instance_t

timer_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- timer_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- timer_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- timer_api_t const * p_api

  Pointer to the API structure for this instance.

8.37 Transfer Interface

Interface for data transfer functions.

8.37.1 Summary

The transfer interface supports background data transfer (no CPU intervention).

The transfer interface can be implemented by:

- DTC
- DMAC

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

Transfer Interface description: Transfer Driver

8.37.2 Interface API

transfer_api_t

Function name	Description
.open	Initial configuration. Enables the transfer if auto_enable is true and p_src, p_dest, and length are valid. Transfers can also be enabled using enable or reset.

Function name	Description
.reset	Reset source address pointer, destination address pointer, and/or length, keeping all other settings the same. Enable the transfer if p_src, p_dest, and length are valid.
.enable	Enable transfer. Transfers occur after the activation source event (or when start is called if ELC_EVENT_ELC_SOFTWARE_EVENT_0 or ELC_EVENT_ELC_SOFTWARE_EVENT_0 is chosen as activation source).
.disable	Disable transfer. Transfers do not occur after the transfer_info_t::activation source event (or when start is called if ELC_EVENT_ELC_SOFTWARE_EVENT_0 or ELC_EVENT_ELC_SOFTWARE_EVENT_0 is chosen as transfer_info_t::activation_source).
.start	Start transfer in software.
.stop	Stop transfer in software. The transfer will stop after completion of the current transfer.
.infoGet	Provides information about this transfer.
.close	Releases hardware lock. This allows a transfer to be reconfigured using open.
.versionGet	Gets version and stores it in provided pointer p_version.
.blockReset	Reset source address pointer, destination address pointer, and/or length, for block transfer keeping all other settings the same. Enable the transfer if p_src, p_dest, and length are valid.

8.37.3 Data structures

- transfer_properties_t
- transfer_info_t
- transfer_callback_args_t
- transfer_cfg_t
- transfer_instance_t

8.37.4 Enumerations

- transfer_mode_t

- transfer_size_t
- transfer_addr_mode_t
- transfer_repeat_area_t
- transfer_chain_mode_t
- transfer_irq_t
- transfer_start_mode_t

8.37.5 Typedefs

- transfer_ctrl_t

8.37.6 Defines

- #define TRANSFER_API_VERSION_MAJOR

  Initial value: `(1U)`
- #define TRANSFER_API_VERSION_MINOR

  Initial value: `(2U)`

8.37.7 API Data

8.37.7.1 transfer_mode_t

`transfer_mode_t`

Detailed description

Transfer mode describes what will happen when a transfer request occurs.

Enumerated values

Name	Description
TRANSFER_MODE_NORMAL	In normal mode, each transfer request causes a transfer of transfer_size_t from the source pointer to the destination pointer. The transfer length is decremented and the source and address pointers are updated according to transfer_addr_mode_t. After the transfer length reaches 0, transfer requests will not cause any further transfers.

Name	Description
TRANSFER_MODE_REPEAT	Repeat mode is like normal mode, except that when the transfer length reaches 0, the pointer to the repeat area and the transfer length will be reset to their initial values. If DMAC is used, the transfer repeats only transfer_info_t::num_blocks times. After the transfer repeats transfer_info_t::num_blocks times, transfer requests will not cause any further transfers. If DTC is used, the transfer repeats continuously (no limit to the number of repeat transfers).
TRANSFER_MODE_BLOCK	In block mode, each transfer request causes transfer_info_t::length transfers of transfer_size_t. After each individual transfer, the source and destination pointers are updated according to transfer_addr_mode_t. After the block transfer is complete, transfer_info_t::num_blocks is decremented. After the transfer_info_t::num_blocks reaches 0, transfer requests will not cause any further transfers.

8.37.7.2 transfer_size_t

`transfer_size_t`

Detailed description

Transfer size specifies the size of each individual transfer. Total transfer length = transfer_size_t * transfer_length_t

Enumerated values

Name	Description
TRANSFER_SIZE_1_BYTE	Each transfer transfers an 8-bit value.
TRANSFER_SIZE_2_BYTE	Address pointer is incremented after each transfer.
TRANSFER_SIZE_4_BYTE	Address pointer is incremented after each transfer.

8.37.7.3 transfer_addr_mode_t

`transfer_addr_mode_t`

Detailed description

Address mode specifies whether to modify (increment or decrement) pointer after each transfer.

Enumerated values

Name	Description
TRANSFER_ADDR_MODE_FIXED	Address pointer remains fixed after each transfer.
TRANSFER_ADDR_MODE_INCREMENTED	Address pointer is incremented by associated transfer_size_t after each transfer.
TRANSFER_ADDR_MODE_DECREMENTED	Address pointer is decremented by associated transfer_size_t after each transfer.

8.37.7.4 transfer_repeat_area_t

`transfer_repeat_area_t`

Detailed description

Repeat area options (source or destination). In TRANSFER_MODE_REPEAT, the selected pointer returns to its original value after length transfers. In TRANSFER_MODE_BLOCK, the selected pointer returns to its original value after each transfer.

Enumerated values

Name	Description
TRANSFER_REPEAT_AREA_DESTINATION	Destination area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK.
TRANSFER_REPEAT_AREA_SOURCE	Source area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK.

8.37.7.5 transfer_chain_mode_t

`transfer_chain_mode_t`

Detailed description

Chain transfer mode options.
NOTE: Only applies for DTC.

Enumerated values

Name	Description
TRANSFER_CHAIN_MODE_DISABLED	Chain mode not used.

Name	Description
TRANSFER_CHAIN_MODE_EACH	Switch to next transfer after a single transfer from this transfer_info_t.
TRANSFER_CHAIN_MODE_END	Complete the entire transfer defined in this transfer_info_t before chaining to next transfer.

8.37.7.6 transfer_irq_t

`transfer_irq_t`

Detailed description

Interrupt options.

Enumerated values

Name	Description
TRANSFER_IRQ_END	Interrupt occurs only after last transfer. If this transfer is chained to a subsequent transfer, the interrupt will occur only after subsequent chained transfer(s) are complete. DTC triggers the interrupt of the activation source. Choosing TRANSFER_IRQ_END with DTC will prevent activation source interrupts until the transfer is complete.
TRANSFER_IRQ_EACH	Interrupt occurs after each transfer. Not available in all HAL drivers. See HAL driver for details. This will prevent chained transfers that would have happened after this one until the next activation source.

8.37.7.7 transfer_start_mode_t

`transfer_start_mode_t`

Detailed description

Select whether to start single or repeated transfer with software start.

Enumerated values

Name	Description
TRANSFER_START_MODE_SINGLE	Software start triggers single transfer.
TRANSFER_START_MODE_REPEAT	Software start transfer continues until transfer is complete.

8.37.7.8 transfer_ctrl_t

```
typedef void transfer_ctrl_t
```

Detailed description

Transfer control block. Allocate an instance specific control block to pass into the transfer API calls. Implemented as

- dtc_instance_ctrl_t
- dmac_instance_ctrl_t

8.37.8 API Structures

8.37.8.1 transfer_properties_t

transfer_properties_t

Detailed description

Driver specific information.

Variables

- uint32_t transfer_length_max

  Maximum number of transfers.

- uint16_t transfer_length_remaining

  Number of transfers remaining.

- bool in_progress

  Whether or not this transfer is in progress.

8.37.8.2 transfer_info_t

transfer_info_t

Detailed description

This structure specifies the properties of the transfer.
ATTENTION: When using DTC, this structure corresponds to the descriptor block registers required by the DTC. The following components may be modified by the driver: p_src, p_dest, num_blocks, and length.

ATTENTION: When using DTC, do NOT reuse this structure to configure multiple transfers. Each transfer must have a unique .

ATTENTION: When using DTC, this structure must not be allocated in a temporary location. Any instance of this structure must remain in scope until the transfer it is used for is closed.

NOTE: When using DTC, consider placing instances of this structure in a protected section of memory.

Variables

- uint32_t __pad0__
- uint32_t __pad1__
- transfer_addr_mode_t dest_addr_mode

  Select what happens to destination pointer after each transfer.
- transfer_repeat_area_t repeat_area

  Select to repeat source or destination area, unused in TRANSFER_MODE_NORMAL.
- transfer_irq_t irq

  Select if interrupts should occur after each individual transfer or after the completion of all planned transfers.
- transfer_chain_mode_t chain_mode

  Select when the chain transfer ends.
- uint32_t __pad2__
- transfer_addr_mode_t src_addr_mode

  Select what happens to source pointer after each transfer.
- transfer_size_t size

  Select number of bytes to transfer at once. transfer_info_t::length.
- transfer_mode_t mode

  Select mode from transfer_mode_t.
- void const *volatile p_src

  Source pointer.
- void *volatile p_dest

  Destination pointer.
- uint16_t num_blocks

  Number of blocks to transfer when using TRANSFER_MODE_BLOCK (both DTC an DMAC) and TRANSFER_MODE_REPEAT (DMAC only), unused in other modes.
- uint16_t length

  Length of each transfer. Range limited for TRANSFER_MODE_BLOCK and TRANSFER_MODE_REPEAT, see HAL driver for details.

8.37.8.3 transfer_callback_args_t

transfer_callback_args_t

Detailed description

Callback function parameter data.

Variables

- void const * p_context

  Placeholder for user data. Set in r_transfer_t::open function in transfer_cfg_t.

8.37.8.4 transfer_cfg_t

transfer_cfg_t

Detailed description

Driver configuration set in open. All elements except p_extend are required and must be initialized.

Variables

- transfer_info_t * p_info

  Pointer to transfer configuration options. If using chain transfer (DTC only), this can be a pointer to an array of chained transfers that will be completed in order.

- elc_event_t activation_source

  Select which event will trigger the transfer. Select ELC_EVENT_ELC_SOFTWARE_EVENT_0 or ELC_EVENT_ELC_SOFTWARE_EVENT_0 for software activation. When using DTC, these events may only be used once each. DMAC uses internal software start when either of these events are selected.

- bool auto_enable

  Select whether the transfer should be enabled after open.

- uint8_t irq_ipl

  Interrupt priority level. Unsupported for DTC except when ELC software events are used. DTC transfers trigger the interrupt associated with the activation source.

- void(* p_callback)(transfer_callback_args_t *p_args)

  Callback for transfer end interrupt. Set to NULL for no CPU interrupt. Unsupported for DTC except when ELC software events are used. DTC transfers trigger the interrupt associated with the activation source.

- void const * p_context

  Placeholder for user data. Passed to the user p_callback in transfer_callback_args_t.

- void const * p_extend

  Extension parameter for hardware specific settings.

8.37.8.5 transfer_api_t

transfer_api_t

Detailed description

Transfer functions implemented at the HAL layer will follow this API.

8.37.8.6 open

```
ssp_err_t(* transfer_api_t::open) (transfer_ctrl_t *const p_ctrl, transfer_cfg_t
const *const p_cfg)
```

Detailed description

Initial configuration. Enables the transfer if auto_enable is true and p_src, p_dest, and length are valid. Transfers can also be enabled using enable or reset. Implemented as

- R_DTC_Open
- R_DMAC_Open

Table 1318:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to control block. Must be declared by user. Elements set here.
p_cfg	in	Pointer to configuration structure. All elements of this structure must be set by user.

Parameter p_ctrl

Definition: `transfer_ctrl_t*const p_ctrl`

Transfer control block. Allocate an instance specific control block to pass into the transfer API calls. Implemented asdtc_instance_ctrl_tdmac_instance_ctrl_t

Parameter p_cfg

Definition: `transfer_cfg_t const *const p_cfg`

Driver configuration set in open. All elements except p_extend are required and must be initialized.

- `transfer_cfg_t`::`transfer_info_t`

  Pointer to transfer configuration options. If using chain transfer (DTC only), this can be a pointer to an array of chained transfers that will be completed in order.

- `transfer_cfg_t`::`elc_event_t`

  Select which event will trigger the transfer. Select ELC_EVENT_ELC_SOFTWARE_EVENT_0 or ELC_EVENT_ELC_SOFTWARE_EVENT_0 for software activation. When using DTC, these events may only be used once each. DMAC uses internal software start when either of these events are selected.

- `transfer_cfg_t`::`auto_enable`

  Select whether the transfer should be enabled after open.

- `transfer_cfg_t`::`irq_ipl`

  Interrupt priority level. Unsupported for DTC except when ELC software events are used. DTC transfers trigger the interrupt associated with the activation source.

- `transfer_cfg_t`::`p_callback`

  Callback for transfer end interrupt. Set to NULL for no CPU interrupt. Unsupported for DTC except when ELC software events are used. DTC transfers trigger the interrupt associated with the activation source.

- transfer_cfg_t::p_context

  Placeholder for user data. Passed to the user p_callback in transfer_callback_args_t.

- transfer_cfg_t::p_extend

  Extension parameter for hardware specific settings.

8.37.8.7 reset

```
ssp_err_t(* transfer_api_t::reset) (transfer_ctrl_t *const p_ctrl, void const
*p_src, void *p_dest, uint16_t const num_transfers)
```

Detailed description

Reset source address pointer, destination address pointer, and/or length, keeping all other settings the same. Enable the transfer if p_src, p_dest, and length are valid. Implemented as

- R_DTC_Reset
- R_DMAC_Reset

Table 1319:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this transfer.
p_src	in	Pointer to source. Set to NULL if source pointer should not change.
p_dest	in	Pointer to destination. Set to NULL if destination pointer should not change.
num_transfers	in	Transfer length in normal mode or number of blocks in block mode. In DMAC only, resets number of repeats (initially stored in num_blocks) in repeat mode. Not used in repeat mode for DTC.

Parameter p_ctrl

Definition: transfer_ctrl_t*const p_ctrl

Transfer control block. Allocate an instance specific control block to pass into the transfer API calls. Implemented asdtc_instance_ctrl_tdmac_instance_ctrl_t

Parameter p_src

const

Parameter p_dest

const

Parameter num_transfers

`uint16_t`

8.37.8.8 enable

```
ssp_err_t(* transfer_api_t::enable) (transfer_ctrl_t *const p_ctrl)
```

Detailed description

Enable transfer. Transfers occur after the activation source event (or when start is called if ELC_EVENT_ELC_SOFTWARE_EVENT_0 or ELC_EVENT_ELC_SOFTWARE_EVENT_0 is chosen as activation source). Implemented as

- R_DMAC_Enable
- R_DTC_Enable

Table 1320:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this transfer.

Parameter p_ctrl

Definition: `transfer_ctrl_t*const p_ctrl`

Transfer control block. Allocate an instance specific control block to pass into the transfer API calls. Implemented asdtc_instance_ctrl_tdmac_instance_ctrl_t

8.37.8.9 disable

```
ssp_err_t(* transfer_api_t::disable) (transfer_ctrl_t *const p_ctrl)
```

Detailed description

Disable transfer. Transfers do not occur after the transfer_info_t::activation source event (or when start is called if ELC_EVENT_ELC_SOFTWARE_EVENT_0 or ELC_EVENT_ELC_SOFTWARE_EVENT_0 is chosen as transfer_info_t::activation_source).

NOTE: If a transfer is in progress, it will be completed. Subsequent transfer requests do not cause a transfer.

Implemented as

- R_DMAC_Disable
- R_DTC_Disable

Table 1321:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this transfer.

Parameter p_ctrl

Definition: `transfer_ctrl_t*const p_ctrl`

Transfer control block. Allocate an instance specific control block to pass into the transfer API calls. Implemented asdtc_instance_ctrl_tdmac_instance_ctrl_t

8.37.8.10 start

```
ssp_err_t(* transfer_api_t::start) (transfer_ctrl_t *const p_ctrl,
transfer_start_mode_t mode)
```

Detailed description

Start transfer in software.

ATTENTION: Only works if ELC_EVENT_ELC_SOFTWARE_EVENT_0 or ELC_EVENT_ELC_SOFT-WARE_EVENT_0 is chosen as transfer_info_t::activation_source.

NOTE: DTC only supports TRANSFER_START_MODE_SINGLE. DTC does not support TRANSFER_START_-MODE_REPEAT.

Implemented as

- R_DMAC_Start
- R_DTC_Start

Table 1322:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this transfer.
mode	in	Select mode from transfer_start_mode_t.

Parameter p_ctrl

Definition: transfer_ctrl_t*const p_ctrl

Transfer control block. Allocate an instance specific control block to pass into the transfer API calls. Implemented asdtc_instance_ctrl_tdmac_instance_ctrl_t

Parameter mode

8.37.8.11 stop

```
ssp_err_t(* transfer_api_t::stop) (transfer_ctrl_t *const p_ctrl)
```

Detailed description

Stop transfer in software. The transfer will stop after completion of the current transfer.

NOTE: Not supported for DTC.

NOTE: Only applies for transfers started with TRANSFER_START_MODE_REPEAT.

ATTENTION: Only works if ELC_EVENT_ELC_SOFTWARE_EVENT_0 or ELC_EVENT_ELC_SOFTWARE_EVENT_0 is chosen as transfer_info_t::activation_source.

Implemented as

- R_DMAC_Stop

Table 1323:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this transfer.

Parameter p_ctrl

Definition: transfer_ctrl_t*const p_ctrl

Transfer control block. Allocate an instance specific control block to pass into the transfer API calls. Implemented asdtc_instance_ctrl_tdmac_instance_ctrl_t

8.37.8.12 infoGet

```
ssp_err_t(* transfer_api_t::infoGet) (transfer_ctrl_t *const p_ctrl,
transfer_properties_t *const p_info)
```

Detailed description

Provides information about this transfer. Implemented as

- R_DTC_InfoGet
- R_DMAC_InfoGet

Table 1324:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this transfer.
p_info	out	Driver specific information.

Parameter p_ctrl

Definition: `transfer_ctrl_t*const p_ctrl`

Transfer control block. Allocate an instance specific control block to pass into the transfer API calls. Implemented asdtc_instance_ctrl_tdmac_instance_ctrl_t

Parameter p_info

Definition: `transfer_properties_t*const p_info`

Driver specific information.

- `transfer_properties_t::transfer_length_max`

  Maximum number of transfers.

- `transfer_properties_t::transfer_length_remaining`

  Number of transfers remaining.

- `transfer_properties_t::in_progress`

  Whether or not this transfer is in progress.

8.37.8.13 close

```
ssp_err_t(* transfer_api_t::close) (transfer_ctrl_t *const p_ctrl)
```

Detailed description

Releases hardware lock. This allows a transfer to be reconfigured using open. Implemented as

- R_DTC_Close
- R_DMAC_Close

Table 1325:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this transfer.

Parameter p_ctrl

Definition: `transfer_ctrl_t*const p_ctrl`

Transfer control block. Allocate an instance specific control block to pass into the transfer API calls. Implemented asdtc_instance_ctrl_tdmac_instance_ctrl_t

8.37.8.14 versionGet

```
ssp_err_t(* transfer_api_t::versionGet) (ssp_version_t *const p_version)
```

Detailed description

Gets version and stores it in provided pointer p_version. Implemented as

- R_DTC_VersionGet
- R_DMAC_VersionGet

Table 1326:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.37.8.15 blockReset

```
ssp_err_t(* transfer_api_t::blockReset) (transfer_ctrl_t *const p_ctrl, void
const *p_src, void *p_dest, uint16_t const length, transfer_size_t size, uint16_t
const num_transfers)
```

Detailed description

Reset source address pointer, destination address pointer, and/or length, for block transfer keeping all other settings the same. Enable the transfer if p_src, p_dest, and length are valid. Implemented as

- R_DMAC_BlockReset
- R_DTC_BlockReset

Table 1327:Parameters

Name	Direction	Description
p_ctrl	in	Control block set in open call for this transfer.
p_src	in	Pointer to source. Set to NULL if source pointer should not change.
p_dest	in	Pointer to destination. Set to NULL if destination pointer should not change.
length	in	Transfer length in block mode.In DMAC only.
size	in	Transfer size in block mode. In DMAC only.
num_transfers	in	number of blocks in block mode. In DMAC only.

Parameter p_ctrl

Definition: `transfer_ctrl_t*const p_ctrl`

Transfer control block. Allocate an instance specific control block to pass into the transfer API calls. Implemented asdtc_instance_ctrl_tdmac_instance_ctrl_t

Parameter p_src

`const`

Parameter p_dest

`const`

Parameter length

`uint16_t`

Parameter size
Parameter num_transfers

`uint16_t`

8.37.8.16 transfer_instance_t

transfer_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- transfer_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.
- transfer_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.
- transfer_api_t const * p_api

  Pointer to the API structure for this instance.

8.38 Random number generation

RNG_Interface Random number generation.

8.38.1 Interface API

trng_api_t

Function name	Description
.open	Open the TRNG driver for reading random data from the hardware TRNG module
.read	Generate of random number words and store them in buffer
.close	Close the TRNG interface driver
.versionGet	Gets version and stores it in provided pointer p_version.

8.38.2 Data structures

- trng_ctrl_t
- trng_cfg_t
- trng_instance_t

8.38.3 Defines

- #define TRNG_API_VERSION_MAJOR

  Initial value: `(01)`

  Register definitions, common services and error codes.

- #define TRNG_API_VERSION_MINOR

  Initial value: `(00)`
- #define TRNG_REGISTER_SIZE_WORDS

  Initial value: `(4)`
- #define TRNG_REGISTER_SIZE_BYTES

  Initial value: `((TRNG_REGISTER_SIZE_WORDS) * 4)`

8.38.4 API Structures

8.38.4.1 trng_ctrl_t

trng_ctrl_t

Detailed description

TRNG_Interface control structure.

Variables

- uint32_t nattempts

  number of retries
- crypto_ctrl_t * p_crypto_ctrl

  pointer to crypto control structure
- crypto_api_t const * p_crypto_api

  pointer to crypto-engine API
- uint32_t prevbuf[TRNG_REGISTER_SIZE_WORDS]

  previous random data
- uint32_t currbuf[TRNG_REGISTER_SIZE_WORDS]

  current random data

8.38.4.2 trng_cfg_t

trng_cfg_t

Detailed description

TRNG interface configuration parameters

Variables

- crypto_api_t const * p_crypto_api

  pointer to crypto API
- uint32_t nattempts

  number of retries when a continuous test failure occurs

8.38.4.3 trng_api_t

trng_api_t

Detailed description

TRNG_Interface SCE functions implemented at the HAL layer will follow this API.

8.38.4.4 open

```
uint32_t(* trng_api_t::open) (trng_ctrl_t *const p_ctrl, trng_cfg_t const *const p_cfg)
```

Detailed description

Open the TRNG driver for reading random data from the hardware TRNG module

Table 1328:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to control structure. Must be declared by user. Elements set here.
p_cfg	in	Pointer to configuration structure. All elements of this structure must be set by user.

Parameter p_ctrl

Definition: `trng_ctrl_t`

TRNG_Interface control structure.

Parameter p_cfg

Definition: `trng_cfg_t const *const p_cfg`

TRNG interface configuration parameters

- `trng_cfg_t`::`p_crypto_api`

  pointer to crypto API

- `trng_cfg_t`::`nattempts`

  number of retries when a continuous test failure occurs

8.38.4.5 read

```
uint32_t(* trng_api_t::read) (trng_ctrl_t *const p_ctrl, uint32_t *const p_rngbuf, uint32_t nwords)
```

Detailed description

Generate nwords of random number words and store them in p_rngbuf buffer

Table 1329:Parameters

Name	Direction	Description
p_ctrl	in	pointer to trng control structure
p_rngbuf	out	generated random numbers will be stored to the buffer p_rngbuf
nwords	in	number of random words to generate

Parameter p_ctrl

Definition: `trng_ctrl_t`

TRNG_Interface control structure.

Parameter p_rngbuf

`uint32_t`

Parameter nwords

`uint32_t`

8.38.4.6 close

```
uint32_t(* trng_api_t::close) (trng_ctrl_t *const p_ctrl)
```

Detailed description

Close the TRNG interface driver

Table 1330:Parameters

Name	Direction	Description
p_ctrl	in	pointer to trng interface control structure

Parameter p_ctrl

Definition: `trng_ctrl_t`

TRNG_Interface control structure.

8.38.4.7 versionGet

```
uint32_t(* trng_api_t::versionGet) ( *const p_version)
```

Detailed description

Gets version and stores it in provided pointer p_version.

Table 1331:Parameters

Name	Direction	Description
p_version	out	Code and API version used.

Parameter p_version

8.38.4.8 trng_instance_t

trng_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- trng_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- trng_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- trng_api_t const * p_api

  Pointer to the API structure for this instance.

8.39 UART Interface

Interface for UART communications.

8.39.1 Summary

The UART interface provides common APIs for UART HAL drivers. The UART interface supports the following features:

- Full-duplex UART communication
- Generic UART parameter setting
- Interrupt driven transmit/receive processing
- Callback function with returned event code
- Runtime baud-rate change
- Hardware resource locking during a transaction
- CTS/RTS hardware flow control support (with an associated IOPORT pin)

- Circular buffer support
- Runtime Transmit/Receive circular buffer flushing

Implemented by:

- UART on SCI

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

UART Interface description: UART Driver

8.39.2 Interface API

uart_api_t

Function name	Description
.open	Open UART device.
.read	Read from UART device. If a transfer instance is used for reception, the received bytes are stored directly in the read input buffer. When a transfer is complete, the callback is called with event UART_EVENT_RX_COMPLETE. Bytes received outside an active transfer are received in the callback function with event UART_EVENT_RX_CHAR. The maximum transfer size is reported by infoGet.
.write	Write to UART device. The write buffer is used until write is complete. Do not overwrite write buffer contents until the write is finished. When the write is complete (all bytes are fully transmitted on the wire), the callback called with event UART_EVENT_TX_COMPLETE. The maximum transfer size is reported by infoGet.
.baudSet	Change baud rate.
.infoGet	Get the driver specific information.
.close	Close UART device.
.versionGet	Get version.
.communicationAbort	Abort ongoing transfer.

8.39.3 Data structures

- uart_info_t
- uart_callback_args_t
- uart_cfg_t
- uart_instance_t

8.39.4 Enumerations

- uart_event_t
- uart_data_bits_t
- uart_parity_t
- uart_stop_bits_t
- uart_dir_t

8.39.5 Typedefs

- uart_ctrl_t

8.39.6 Defines

- #define UART_API_VERSION_MAJOR
  Initial value: `(1U)`
- #define UART_API_VERSION_MINOR
  Initial value: `(5U)`

8.39.7 API Data

8.39.7.1 uart_event_t

`uart_event_t`

Detailed description

UART Event codes

Enumerated values

Name	Description
UART_EVENT_RX_COMPLETE	Receive complete event.

Name	Description
UART_EVENT_TX_COMPLETE	Transmit complete event.
UART_EVENT_ERR_PARITY	Parity error event.
UART_EVENT_ERR_FRAMING	Mode fault error event.
UART_EVENT_BREAK_DETECT	Break detect error event.
UART_EVENT_ERR_OVERFLOW	FIFO Overflow error event.
UART_EVENT_ERR_RXBUF_OVERFLOW	DEPRECATED: Receive buffer overflow error event.
UART_EVENT_RX_CHAR	Character received.
UART_EVENT_TX_DATA_EMPTY	Last byte is transmitting, ready for more data.

8.39.7.2 uart_data_bits_t

`uart_data_bits_t`

Detailed description

UART Data bit length definition

Enumerated values

Name	Description
UART_DATA_BITS_8	Data bits 8-bit.
UART_DATA_BITS_7	Data bits 7-bit.
UART_DATA_BITS_9	Data bits 9-bit.

8.39.7.3 uart_parity_t

`uart_parity_t`

Detailed description

UART Parity definition

Enumerated values

Name	Description
UART_PARITY_OFF	No parity.

Name	Description
UART_PARITY_EVEN	Even parity.
UART_PARITY_ODD	Odd parity.

8.39.7.4 uart_stop_bits_t

```
uart_stop_bits_t
```

Detailed description

UART Stop bits definition

Enumerated values

Name	Description
UART_STOP_BITS_1	Stop bit 1-bit.
UART_STOP_BITS_2	Stop bits 2-bit.

8.39.7.5 uart_dir_t

```
uart_dir_t
```

Detailed description

UART transaction definition

Enumerated values

Name	Description
UART_DIR_RX_TX	Both RX and TX.
UART_DIR_RX	Only RX.
UART_DIR_TX	Only TX.

8.39.7.6 uart_ctrl_t

```
typedef void uart_ctrl_t
```

Detailed description

UART control block. Allocate an instance specific control block to pass into the UART API calls. Implemented as

- sci_uart_instance_ctrl_t

8.39.8 API Structures

8.39.8.1 uart_info_t

uart_info_t

Detailed description

UART driver specific information

Variables

- uint32_t write_bytes_max

  Maximum bytes that can be written at this time. Only applies if uart_cfg_t::p_transfer_tx is not NULL.

- uint32_t read_bytes_max

  Maximum bytes that are available to read at one time. Only applies if uart_cfg_t::p_transfer_rx is not NULL.

8.39.8.2 uart_callback_args_t

uart_callback_args_t

Detailed description

UART Callback parameter definition

Variables

- uint32_t channel

  Device channel number.

- uart_event_t event

  Event code.

- uint32_t data

  Contains the next character received for the events UART_EVENT_RX_CHAR, UART_EVENT_ERR_PARITY, UART_EVENT_ERR_FRAMING, or UART_EVENT_ERR_OVERFLOW. Otherwise unused.

- void const * p_context

  Context provided to user during callback.

8.39.8.3 uart_cfg_t

uart_cfg_t

Detailed description

UART Configuration

Variables

- uint8_t channel

  Select a channel corresponding to the channel number of the hardware.

- uint32_t baud_rate

  Baud rate, i.e. 9600, 19200, 115200.

- uart_data_bits_t data_bits

  Data bit length (8 or 7 or 9)

- uart_parity_t parity

  Parity type (none or odd or even)

- uart_stop_bits_t stop_bits

  Stop bit length (1 or 2)

- bool ctsrts_en

  CTS/RTS hardware flow control enable.

- uint8_t rxi_ipl

  Receive interrupt priority.

- uint8_t txi_ipl

  Transmit interrupt priority.

- uint8_t tei_ipl

  Transmit end interrupt priority.

- uint8_t eri_ipl

  Error interrupt priority.

- transfer_instance_t const * p_transfer_rx

  Optional transfer instance used to receive multiple bytes without interrupts. Set to NULL if unused. If NULL, the number of bytes allowed in the read API is limited to one byte at a time.

- transfer_instance_t const * p_transfer_tx

  Optional transfer instance used to send multiple bytes without interrupts. Set to NULL if unused. If NULL, the number of bytes allowed in the write APIs is limited to one byte at a time.

- void(* p_callback)(uart_callback_args_t *p_args)

  Pointer to callback function.

- void const * p_context

  User defined context passed into callback function.

- void const * p_extend

  UART hardware dependent configuration.

8.39.8.4 uart_api_t

uart_api_t

Detailed description

Shared Interface definition for UART

8.39.8.5 open

```
ssp_err_t(* uart_api_t::open) (uart_ctrl_t *const p_ctrl, uart_cfg_t const *const p_cfg)
```

Detailed description

Open UART device. Implemented as

- R_SCI_UartOpen

Table 1332:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to the UART control block Must be declared by user. Value set here.
uart_cfg_t	in	Pointer to UART configuration structure. All elements of this structure must be set by user.

Parameter p_ctrl

Definition: `uart_ctrl_t*const p_ctrl`

UART control block. Allocate an instance specific control block to pass into the UART API calls. Implemented assci_uart_instance_ctrl_t

Parameter uart_cfg_t

Definition: `uart_cfg_t const *const p_cfg`

UART Configuration

- `uart_cfg_t`::channel

  Select a channel corresponding to the channel number of the hardware.

- `uart_cfg_t`::baud_rate

  Baud rate, i.e. 9600, 19200, 115200.

- `uart_cfg_t`::`uart_data_bits_t`

  Data bit length (8 or 7 or 9)

  Enumerated as:

  – UART_DATA_BITS_8

  – UART_DATA_BITS_7

  – UART_DATA_BITS_9

- uart_cfg_t::uart_parity_t

  Parity type (none or odd or even)

  Enumerated as:

  – UART_PARITY_OFF

  – UART_PARITY_EVEN

  – UART_PARITY_ODD

- uart_cfg_t::uart_stop_bits_t

  Stop bit length (1 or 2)

  Enumerated as:

  – UART_STOP_BITS_1

  – UART_STOP_BITS_2

- uart_cfg_t::ctsrts_en

  CTS/RTS hardware flow control enable.

- uart_cfg_t::rxi_ipl

  Receive interrupt priority.

- uart_cfg_t::txi_ipl

  Transmit interrupt priority.

- uart_cfg_t::tei_ipl

  Transmit end interrupt priority.

- uart_cfg_t::eri_ipl

  Error interrupt priority.

- uart_cfg_t::transfer_instance_t

  Optional transfer instance used to receive multiple bytes without interrupts. Set to NULL if unused. If NULL, the number of bytes allowed in the read API is limited to one byte at a time.

- uart_cfg_t::transfer_instance_t

  Optional transfer instance used to send multiple bytes without interrupts. Set to NULL if unused. If NULL, the number of bytes allowed in the write APIs is limited to one byte at a time.

- uart_cfg_t::p_callback

  Pointer to callback function.

- uart_cfg_t::p_context

  User defined context passed into callback function.

- uart_cfg_t::p_extend

  UART hardware dependent configuration.

8.39.8.6 read

```
ssp_err_t(* uart_api_t::read) (uart_ctrl_t *const p_ctrl, uint8_t const *const
p_dest, uint32_t const bytes)
```

Detailed description

Read from UART device. If a transfer instance is used for reception, the received bytes are stored directly in the read input buffer. When a transfer is complete, the callback is called with event UART_EVENT_RX_COMPLETE. Bytes received outside an active transfer are received in the callback function with event UART_EVENT_RX_CHAR. The maximum transfer size is reported by infoGet. Implemented as

- R_SCI_UartRead

Table 1333:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the UART control block for the channel.
p_dest	in	Destination address to read data from.
bytes	in	Read data length. Only applicable if p_transfer_rx is not NULL. Otherwise all read bytes will be provided through the callback set in p_callback.

Parameter p_ctrl

Definition: `uart_ctrl_t*const p_ctrl`

UART control block. Allocate an instance specific control block to pass into the UART API calls. Implemented assci_uart_instance_ctrl_t

Parameter p_dest

`uint8_t`

Parameter bytes

`uint32_t`

8.39.8.7 write

```
ssp_err_t(* uart_api_t::write) (uart_ctrl_t *const p_ctrl, uint8_t const *const
p_src, uint32_t const bytes)
```

Detailed description

Write to UART device. The write buffer is used until write is complete. Do not overwrite write buffer contents until the write is finished. When the write is complete (all bytes are fully transmitted on the wire), the callback called with event UART_EVENT_TX_COMPLETE. The maximum transfer size is reported by infoGet. Implemented as

- R_SCI_UartWrite

Table 1334:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the UART control block.
p_src	in	Source address to write data to.
bytes	in	Write data length.

Parameter p_ctrl

Definition: `uart_ctrl_t*const p_ctrl`

UART control block. Allocate an instance specific control block to pass into the UART API calls. Implemented assci_uart_instance_ctrl_t

Parameter p_src

`uint8_t`

Parameter bytes

`uint32_t`

8.39.8.8 baudSet

```
ssp_err_t(* uart_api_t::baudSet) (uart_ctrl_t *const p_ctrl, uint32_t const
baudrate)
```

Detailed description

Change baud rate.

ATTENTION: Calling this API aborts any in-progress transmission and disables reception until the new baud settings have been applied.

Implemented as

- R_SCI_UartBaudSet

Table 1335:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the UART control block.
baudrate	in	Baud rate in bps.

Parameter p_ctrl

Definition: uart_ctrl_t*const p_ctrl

UART control block. Allocate an instance specific control block to pass into the UART API calls. Implemented assci_uart_instance_ctrl_t

Parameter baudrate

uint32_t

8.39.8.9 infoGet

```
ssp_err_t(* uart_api_t::infoGet) (uart_ctrl_t *const p_ctrl, uart_info_t *const p_info)
```

Detailed description

Get the driver specific information. Implemented as

- R_SCI_UartInfoGet

Table 1336:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the UART control block.
baudrate	in	Baud rate in bps.

Parameter p_ctrl

Definition: uart_ctrl_t*const p_ctrl

UART control block. Allocate an instance specific control block to pass into the UART API calls. Implemented assci_uart_instance_ctrl_t

Parameter baudrate

8.39.8.10 close

```
ssp_err_t(* uart_api_t::close) (uart_ctrl_t *const p_ctrl)
```

Detailed description

Close UART device. Implemented as

- R_SCI_UartClose

Table 1337:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the UART control block.

Parameter p_ctrl

Definition: uart_ctrl_t*const p_ctrl

UART control block. Allocate an instance specific control block to pass into the UART API calls. Implemented assci_uart_instance_ctrl_t

8.39.8.11 versionGet

```
ssp_err_t(* uart_api_t::versionGet) (ssp_version_t *p_version)
```

Detailed description

Get version. Implemented as

- R_SCI_UartVersionGet

Table 1338:Parameters

Name	Direction	Description
p_version	in	Pointer to the memory to store the version information.

Parameter p_version

8.39.8.12 communicationAbort

```
ssp_err_t(* uart_api_t::communicationAbort) (uart_ctrl_t *const p_ctrl,
uart_dir_t communication_to_abort)
```

Detailed description

Abort ongoing transfer. Implemented as

- R_SCI_UartAbort

Table 1339:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the UART control block.
communication_to_abort	in	Type of abort request.

Parameter p_ctrl

Definition: uart_ctrl_t*const p_ctrl

UART control block. Allocate an instance specific control block to pass into the UART API calls. Implemented assci_uart_instance_ctrl_t

Parameter communication_to_abort

Definition: uart_dir_tcommunication_to_abort

UART transaction definition

8.39.8.13 uart_instance_t

uart_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- uart_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- uart_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- uart_api_t const * p_api

  Pointer to the API structure for this instance.

8.40 WDT Interface

Interface for watch dog timer functions.

8.40.1 Summary

The WDT interface for the Watchdog Timer (WDT) peripheral provides watchdog functionality including resetting the device or generating an interrupt.

See Also and Thread Monitor Framework Interface

The watchdog timer interface can be implemented by:

- WDT
- IWDT

Related SSP architecture topics:

- SSP Interfaces
- SSP Predefined Layers
- Using SSP Modules

WDT Interface description: Watchdog Driver

8.40.2 Interface API

wdt_api_t

Function name	Description
.cfgGet	Initialize the WDT in register start mode. In auto-start mode with NMI output it registers the NMI callback.
.open	Initialize the WDT in register start mode. In auto-start mode with NMI output it registers the NMI callback.
.refresh	Refresh the watchdog timer.
.statusGet	Read the status of the WDT.
.statusClear	Clear the status flags of the WDT.
.counterGet	Read the current WDT counter value.
.timeoutGet	Read the watchdog timeout values.
.versionGet	Return the version of the IOPort driver.

8.40.3 Data structures

- wdt_callback_args_t
- wdt_timeout_values_t
- wdt_cfg_t
- wdt_instance_t

8.40.4 Enumerations

- wdt_timeout_t
- wdt_clock_division_t
- wdt_window_start_t
- wdt_window_end_t
- wdt_reset_control_t
- wdt_stop_control_t
- wdt_status_t
- wdt_start_mode_t

8.40.5 Typedefs

- wdt_ctrl_t

8.40.6 Defines

- #define WDT_API_VERSION_MAJOR

  Initial value: `(1U)`
- #define WDT_API_VERSION_MINOR

  Initial value: `(2U)`

8.40.7 API Data

8.40.7.1 wdt_timeout_t

`wdt_timeout_t`

Detailed description

WDT time-out periods.

Enumerated values

Name	Description
WDT_TIMEOUT_128	128 clock cycles
WDT_TIMEOUT_512	512 clock cycles
WDT_TIMEOUT_1024	1024 clock cycles
WDT_TIMEOUT_2048	2048 clock cycles
WDT_TIMEOUT_4096	4096 clock cycles
WDT_TIMEOUT_8192	8192 clock cycles
WDT_TIMEOUT_16384	16384 clock cycles

8.40.7.2 wdt_clock_division_t

`wdt_clock_division_t`

Detailed description

WDT clock division ratio.

Enumerated values

Name	Description
WDT_CLOCK_DIVISION_1	CLK/1.
WDT_CLOCK_DIVISION_4	CLK/4.
WDT_CLOCK_DIVISION_16	CLK/16.
WDT_CLOCK_DIVISION_32	CLK/32.
WDT_CLOCK_DIVISION_64	CLK/64.
WDT_CLOCK_DIVISION_128	CLK/128.
WDT_CLOCK_DIVISION_256	CLK/256.
WDT_CLOCK_DIVISION_512	CLK/512.
WDT_CLOCK_DIVISION_2048	CLK/2048.
WDT_CLOCK_DIVISION_8192	CLK/8192.

8.40.7.3 wdt_window_start_t

`wdt_window_start_t`

Detailed description

WDT refresh permitted period window start position.

Enumerated values

Name	Description
WDT_WINDOW_START_25	Start position = 25%.
WDT_WINDOW_START_50	Start position = 50%.
WDT_WINDOW_START_75	Start position = 75%.
WDT_WINDOW_START_100	Start position = 100%.

8.40.7.4 wdt_window_end_t

`wdt_window_end_t`

Detailed description

WDT refresh permitted period window end position.

Enumerated values

Name	Description
WDT_WINDOW_END_75	End position = 75%.
WDT_WINDOW_END_50	End position = 50%.
WDT_WINDOW_END_25	End position = 25%.
WDT_WINDOW_END_0	End position = 0%.

8.40.7.5 wdt_reset_control_t

`wdt_reset_control_t`

Detailed description

WDT Counter underflow and refresh error control.

Enumerated values

Name	Description
WDT_RESET_CONTROL_NMI	NMI request when counter underflows.
WDT_RESET_CONTROL_RESET	Reset request when counter underflows.

8.40.7.6 wdt_stop_control_t

`wdt_stop_control_t`

Detailed description

WDT Counter operation in sleep mode.

Enumerated values

Name	Description
WDT_STOP_CONTROL_DISABLE	Count will not stop when device enters sleep mode.
WDT_STOP_CONTROL_ENABLE	Count will automatically stop when device enters sleep mode.

8.40.7.7 wdt_status_t

```
wdt_status_t
```

Detailed description

WDT status

Enumerated values

Name	Description
WDT_STATUS_NO_ERROR	No status flags set.
WDT_STATUS_UNDERFLOW_ERROR	Underflow flag set.
WDT_STATUS_REFRESH_ERROR	Refresh error flag set. Refresh outside of permitted.
WDT_STATUS_UNDERFLOW_AND_REFRESH_ERRO R	Underflow and refresh error flags set.

8.40.7.8 wdt_start_mode_t

```
wdt_start_mode_t
```

Detailed description

WDT start mode. Used to check the WDT is configured correctly.

Enumerated values

Name	Description
WDT_START_MODE_REGISTER	WDT is to be configured using the WDT registers.
WDT_START_MODE_AUTO	WDT is to be configured using OFS0 hardware register.
WDT_START_MODE_DISABLED	WDT is disabled.

8.40.7.9 wdt_ctrl_t

```
typedef void wdt_ctrl_t
```

Detailed description

WDT control block. Allocate an instance specific control block to pass into the WDT API calls. Implemented as

- wdt_instance_ctrl_t
- iwdt_instance_ctrl_t

8.40.8 API Structures

8.40.8.1 wdt_callback_args_t

wdt_callback_args_t

Detailed description

Callback function parameter data

Variables

- void const * p_context

  Placeholder for user data. Set in wdt_api_t::open function in wdt_cfg_t.

8.40.8.2 wdt_timeout_values_t

wdt_timeout_values_t

Detailed description

WDT timeout data. Used to return frequency of WDT clock and timeout period

Variables

- uint32_t clock_frequency_hz

  Frequency of watchdog clock after divider.

- uint32_t timeout_clocks

  Timeout period in units of watchdog clock ticks.

8.40.8.3 wdt_cfg_t

wdt_cfg_t

Detailed description

WDT configuration parameters.

Variables

- wdt_start_mode_t start_mode

  The expected start mode for the WDT.

- bool autostart

  When true the WDT is started as part of its configuration (register start mode). If false the WDT needs to be started manually by calling the refresh API.

- wdt_timeout_t timeout

  Timeout period.

- wdt_clock_division_t clock_division

  Clock divider.

- wdt_window_start_t window_start

  Refresh permitted window start position.
- wdt_window_end_t window_end

  Refresh permitted window end position.
- wdt_reset_control_t reset_control

  Select NMI or reset generated on underflow.
- wdt_stop_control_t stop_control

  Select whether counter operates in sleep mode.
- void(* p_callback)(wdt_callback_args_t *p_args)

  Callback provided when a WDT NMI ISR occurs.
- void const * p_context

  Placeholder for user data. Passed to the user callback in wdt_callback_args_t.
- void const * p_extend

  Placeholder for user extension.

8.40.8.4 wdt_api_t

wdt_api_t

Detailed description

WDT functions implemented at the HAL layer will follow this API.

8.40.8.5 cfgGet

```
ssp_err_t(* wdt_api_t::cfgGet) (wdt_ctrl_t *const p_ctrl, wdt_cfg_t *const p_cfg)
```

Detailed description

Initialize the WDT in register start mode. In auto-start mode with NMI output it registers the NMI callback. Implemented as

- R_WDT_CfgGet
- R_IWDT_CfgGet

Table 1340:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure.
p_cfg	out	Pointer to pin configuration structure for reading WDT configuration.

Parameter p_ctrl

Definition: `wdt_ctrl_t*const p_ctrl`

WDT control block. Allocate an instance specific control block to pass into the WDT API calls. Implemented aswdt_instance_ctrl_tiwdt_instance_ctrl_t

Parameter p_cfg

Definition: `wdt_cfg_t *const p_cfg`

WDT configuration parameters.

- `wdt_cfg_t`::`wdt_start_mode_t`

  The expected start mode for the WDT.

  Enumerated as:

  – WDT_START_MODE_REGISTER

  – WDT_START_MODE_AUTO

  – WDT_START_MODE_DISABLED

- `wdt_cfg_t`::`autostart`

  When true the WDT is started as part of its configuration (register start mode). If false the WDT needs to be started manually by calling the refresh API.

- `wdt_cfg_t`::`wdt_timeout_t`

  Timeout period.

  Enumerated as:

  – WDT_TIMEOUT_128

  – WDT_TIMEOUT_512

  – WDT_TIMEOUT_1024

  – WDT_TIMEOUT_2048

  – WDT_TIMEOUT_4096

  – WDT_TIMEOUT_8192

  – WDT_TIMEOUT_16384

- `wdt_cfg_t`::`wdt_clock_division_t`

  Clock divider.

  Enumerated as:

  – WDT_CLOCK_DIVISION_1

  – WDT_CLOCK_DIVISION_4

  – WDT_CLOCK_DIVISION_16

  – WDT_CLOCK_DIVISION_32

  - WDT_CLOCK_DIVISION_64
  - WDT_CLOCK_DIVISION_128
  - WDT_CLOCK_DIVISION_256
  - WDT_CLOCK_DIVISION_512
  - WDT_CLOCK_DIVISION_2048
  - WDT_CLOCK_DIVISION_8192
- wdt_cfg_t::wdt_window_start_t

  Refresh permitted window start position.

  Enumerated as:
  - WDT_WINDOW_START_25
  - WDT_WINDOW_START_50
  - WDT_WINDOW_START_75
  - WDT_WINDOW_START_100
- wdt_cfg_t::wdt_window_end_t

  Refresh permitted window end position.

  Enumerated as:
  - WDT_WINDOW_END_75
  - WDT_WINDOW_END_50
  - WDT_WINDOW_END_25
  - WDT_WINDOW_END_0
- wdt_cfg_t::wdt_reset_control_t

  Select NMI or reset generated on underflow.

  Enumerated as:
  - WDT_RESET_CONTROL_NMI
  - WDT_RESET_CONTROL_RESET
- wdt_cfg_t::wdt_stop_control_t

  Select whether counter operates in sleep mode.

  Enumerated as:
  - WDT_STOP_CONTROL_DISABLE
  - WDT_STOP_CONTROL_ENABLE

- wdt_cfg_t::p_callback

  Callback provided when a WDT NMI ISR occurs.

- wdt_cfg_t::p_context

  Placeholder for user data. Passed to the user callback in wdt_callback_args_t.

- wdt_cfg_t::p_extend

  Placeholder for user extension.

8.40.8.6 open

```
ssp_err_t(* wdt_api_t::open) (wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)
```

Detailed description

Initialize the WDT in register start mode. In auto-start mode with NMI output it registers the NMI callback. Implemented as

- R_WDT_Open
- R_IWDT_Open

Table 1341:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure.
p_cfg	in	Pointer to pin configuration structure.

Parameter p_ctrl

Definition: wdt_ctrl_t*const p_ctrl

WDT control block. Allocate an instance specific control block to pass into the WDT API calls. Implemented aswdt_instance_ctrl_tiwdt_instance_ctrl_t

Parameter p_cfg

Definition: wdt_cfg_t const *const p_cfg

WDT configuration parameters.

- wdt_cfg_t::wdt_start_mode_t

  The expected start mode for the WDT.

  Enumerated as:

  – WDT_START_MODE_REGISTER

  – WDT_START_MODE_AUTO

  – WDT_START_MODE_DISABLED

- wdt_cfg_t::autostart

  When true the WDT is started as part of its configuration (register start mode). If false the WDT needs to be started manually by calling the refresh API.

- wdt_cfg_t::wdt_timeout_t

  Timeout period.

  Enumerated as:

  – WDT_TIMEOUT_128
  – WDT_TIMEOUT_512
  – WDT_TIMEOUT_1024
  – WDT_TIMEOUT_2048
  – WDT_TIMEOUT_4096
  – WDT_TIMEOUT_8192
  – WDT_TIMEOUT_16384

- wdt_cfg_t::wdt_clock_division_t

  Clock divider.

  Enumerated as:

  – WDT_CLOCK_DIVISION_1
  – WDT_CLOCK_DIVISION_4
  – WDT_CLOCK_DIVISION_16
  – WDT_CLOCK_DIVISION_32
  – WDT_CLOCK_DIVISION_64
  – WDT_CLOCK_DIVISION_128
  – WDT_CLOCK_DIVISION_256
  – WDT_CLOCK_DIVISION_512
  – WDT_CLOCK_DIVISION_2048
  – WDT_CLOCK_DIVISION_8192

- wdt_cfg_t::wdt_window_start_t

  Refresh permitted window start position.

  Enumerated as:

  – WDT_WINDOW_START_25
  – WDT_WINDOW_START_50

  - WDT_WINDOW_START_75
  - WDT_WINDOW_START_100
- wdt_cfg_t::wdt_window_end_t

  Refresh permitted window end position.

  Enumerated as:
  - WDT_WINDOW_END_75
  - WDT_WINDOW_END_50
  - WDT_WINDOW_END_25
  - WDT_WINDOW_END_0
- wdt_cfg_t::wdt_reset_control_t

  Select NMI or reset generated on underflow.

  Enumerated as:
  - WDT_RESET_CONTROL_NMI
  - WDT_RESET_CONTROL_RESET
- wdt_cfg_t::wdt_stop_control_t

  Select whether counter operates in sleep mode.

  Enumerated as:
  - WDT_STOP_CONTROL_DISABLE
  - WDT_STOP_CONTROL_ENABLE
- wdt_cfg_t::p_callback

  Callback provided when a WDT NMI ISR occurs.
- wdt_cfg_t::p_context

  Placeholder for user data. Passed to the user callback in wdt_callback_args_t.
- wdt_cfg_t::p_extend

  Placeholder for user extension.

8.40.8.7 refresh

```
ssp_err_t(* wdt_api_t::refresh) (wdt_ctrl_t *const p_ctrl)
```

Detailed description

Refresh the watchdog timer. Implemented as

- R_WDT_Refresh
- R_IWDT_Refresh

NOTE: If the WDT is in auto-start mode ensure the OFS0 register is configured before using this function.

ATTENTION: Calling this function in register-start mode before calling R_WDT_Open will start the WDT in it's default state and further changes to the configuration will not be possible.

Table 1342:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure.

Parameter p_ctrl

Definition: `wdt_ctrl_t*const p_ctrl`

WDT control block. Allocate an instance specific control block to pass into the WDT API calls. Implemented aswdt_instance_ctrl_tiwdt_instance_ctrl_t

8.40.8.8 statusGet

```
ssp_err_t(* wdt_api_t::statusGet) (wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)
```

Detailed description

Read the status of the WDT. Implemented as

- R_WDT_StatusGet
- R_IWDT_StatusGet

Table 1343:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure.
p_status	out	Pointer to variable to return status information through.

Parameter p_ctrl

Definition: `wdt_ctrl_t*const p_ctrl`

WDT control block. Allocate an instance specific control block to pass into the WDT API calls. Implemented aswdt_instance_ctrl_tiwdt_instance_ctrl_t

Parameter p_status

Definition: `wdt_status_t*const p_status`

WDT status

8.40.8.9 statusClear

```
ssp_err_t(* wdt_api_t::statusClear) (wdt_ctrl_t *const p_ctrl, const wdt_status_t status)
```

Detailed description

Clear the status flags of the WDT. Implemented as

- R_WDT_StatusClear
- R_IWDT_StatusClear

Table 1344:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure.
status	in	Status condition(s) to clear.

Parameter p_ctrl

Definition: `wdt_ctrl_t*const p_ctrl`

WDT control block. Allocate an instance specific control block to pass into the WDT API calls. Implemented aswdt_instance_ctrl_tiwdt_instance_ctrl_t

Parameter status

8.40.8.10 counterGet

```
ssp_err_t(* wdt_api_t::counterGet) (wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)
```

Detailed description

Read the current WDT counter value. Implemented as

- R_WDT_CounterGet
- R_IWDT_CounterGet

Table 1345:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure.

Table 1345:Parameters (Continued)

Name	Direction	Description
p_count	out	Pointer to variable to return current WDT counter value.

Parameter p_ctrl

Definition: `wdt_ctrl_t*const p_ctrl`

WDT control block. Allocate an instance specific control block to pass into the WDT API calls. Implemented aswdt_instance_ctrl_tiwdt_instance_ctrl_t

Parameter p_count

`uint32_t`

8.40.8.11 timeoutGet

```
ssp_err_t(* wdt_api_t::timeoutGet) (wdt_ctrl_t *const p_ctrl,
wdt_timeout_values_t *const p_timeout)
```

Detailed description

Read the watchdog timeout values. Implemented as

- R_WDT_TimeoutGet
- R_IWDT_TimeoutGet

Table 1346:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure.
p_timeout	out	Pointer to structure to return timeout values.

Parameter p_ctrl

Definition: `wdt_ctrl_t*const p_ctrl`

WDT control block. Allocate an instance specific control block to pass into the WDT API calls. Implemented aswdt_instance_ctrl_tiwdt_instance_ctrl_t

Parameter p_timeout

Definition: `wdt_timeout_values_t*const p_timeout`

WDT timeout data. Used to return frequency of WDT clock and timeout period

- `wdt_timeout_values_t::clock_frequency_hz`

  Frequency of watchdog clock after divider.

- wdt_timeout_values_t::timeout_clocks

  Timeout period in units of watchdog clock ticks.

8.40.8.12 versionGet

```
ssp_err_t(* wdt_api_t::versionGet) (ssp_version_t *const p_data)
```

Detailed description

Return the version of the IOPort driver. Implemented as

- R_WDT_VersionGet
- R_IWDT_VersionGet

Table 1347:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure.
p_data	out	Memory address to return version information to.

Parameter p_ctrl
Parameter p_data

Definition: ssp_version_t *const p_data

- ssp_version_t::version_id

  Version id

- ssp_version_t::code_version_minor

  Code minor version.

- ssp_version_t::code_version_major

  Code major version.

- ssp_version_t::api_version_minor

  API minor version.

- ssp_version_t::api_version_major

  API major version.

- ssp_version_tstruct{}

  Code version parameters

8.40.8.13 wdt_instance_t

wdt_instance_t

Detailed description

This structure encompasses everything that is needed to use an instance of this interface.

Variables

- wdt_ctrl_t * p_ctrl

  Pointer to the control structure for this instance.

- wdt_cfg_t const * p_cfg

  Pointer to the configuration structure for this instance.

- wdt_api_t const * p_api

  Pointer to the API structure for this instance.

Chapter 9 API Reference: HAL Drivers

9.1 API Reference: HAL Layer

The hardware abstraction layer provides drivers for Renesas peripherals. HAL drivers typically implement Interfaces and provide additional hardware specific APIs.

- High-Speed Analog Comparator
- Low Power Analog Comparator
- ADC
- AGT
- CAC
- CAN
- CGC
- CRC
- CTSU
- DAC
- DAC
- DMAC
- DOC
- DTC
- ELC
- High-performance Flash
- Low Power Flash
- FMI
- GLCDC
- GPT
- GPT Input Capture
- ICU
- IOPORT
- IWDT
- JPEG Decode
- JPEG Encode
- Key Interrupts

- LPM
- LPMV2 S124
- LPMV2 S128
- LPMV2 S1JA
- LPMV2 S3A3
- LPMV2 S3A6
- LPMV2 S3A7
- LPMV2 S5D4
- LPMV2 S5D9
- LPMV2 S7G2
- LVD
- Operational Amplifier (OPAMP)
- PDC
- QSPI
- IIC
- IIC Slave
- SPI
- RTC
- Simple I2C on SCI
- Simple SPI on SCI
- UART on SCI
- Sigma Delta ADC (SDADC)
- SDMMC
- SLCDC
- SSI
- WDT
- SCE Module

9.2 High-Speed Analog Comparator

Driver for the High-Speed Analog Comparator.

9.2.1 Summary

Extends COMPARATOR Interface.

This module implements the COMPARATOR Interface using the high-speed analog comparator.

9.2.2 Functions

- R_ACMPHS_Open
- R_ACMPHS_InfoGet
- R_ACMPHS_OutputEnable
- R_ACMPHS_StatusGet
- R_ACMPHS_Close
- R_ACMPHS_VersionGet

9.2.3 Defines

- #define ACMPHS_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define ACMPHS_CODE_VERSION_MINOR

  Initial value: `(0U)`

9.2.4 R_ACMPHS_Open

```
ssp_err_t R_ACMPHS_Open ( comparator_ctrl_t  *const p_api_ctrl ,
 comparator_cfg_t  const *const p_cfg )
```

9.2.4.1 Detailed description

Configures the comparator and starts operation. Callbacks and pin output are not active until outputEnable() is called. outputEnable should be called after the output has stabilized. Implements open.

Comparator inputs must be configured in the application code prior to calling this function.

Table 1348:Return values

Name	Description
SSP_SUCCESS	Open successful.
SSP_ERR_ASSERTION	An input pointer is NULL
SSP_ERR_INVALID_ARGUMENT	An argument is invalid. Window mode (COMPARATOR_MODE_WINDOW) and filter of 1 (COMPARATOR_FILTER_1) are not supported in this implementation.

Table 1348:Return values (Continued)

Name	Description
SSP_ERR_IN_USE	The control block is already open or the hardware lock is taken.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet

9.2.4.2 Function steps

- Verify the control block has not already been initialized.
- Configure interrupt priority. The interrupt is disabled until outputEnable is called.
- Enable clocks to the ACMPHS hardware.
- Set registers controlled by this driver to their default values.
- Configure the output polarity.
- Configure the trigger edge.
- Configure the hardware debounce filter.
- Enable the comparator.

9.2.5 R_ACMPHS_InfoGet

```
ssp_err_t R_ACMPHS_InfoGet ( comparator_ctrl_t  *const p_api_ctrl ,
 comparator_info_t  *const p_info )
```

9.2.5.1 Detailed description

Provides the minimum stabilization wait time in microseconds. Implements infoGet.

Table 1349:Return values

Name	Description
SSP_SUCCESS	Information stored in p_info.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.2.5.2 Function steps

- Perform parameter checking
- Get the base stabilization time.
- Add 4 filter clocks if the filter is enabled.

9.2.6 R_ACMPHS_OutputEnable

```
ssp_err_t R_ACMPHS_OutputEnable ( comparator_ctrl_t  *const p_api_ctrl )
```

9.2.6.1 Detailed description

Enables the comparator output, which can be polled using statusGet. Also enables pin output and interrupts as configured during open. Implements outputEnable.

Table 1350:Return values

Name	Description
SSP_SUCCESS	Comparator output is enabled.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.2.6.2 Function steps

- Enable the ACMPHS output.
- Set the VCOUT output setting for this channel (enabled or disabled).
- Enable interrupts for this channel.

9.2.7 R_ACMPHS_StatusGet

```
ssp_err_t R_ACMPHS_StatusGet ( comparator_ctrl_t  *const p_api_ctrl ,
 comparator_status_t  *const p_status )
```

9.2.7.1 Detailed description

Provides the operating status of the comparator. Implements statusGet.

Table 1351:Return values

Name	Description
SSP_SUCCESS	Operating status of the comparator is provided in p_status.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.
SSP_ERR_TIMEOUT	The debounce filter is off and 2 consecutive matching values were not read within 1024 attempts.

9.2.7.2 Function steps

- Read the operating status of the comparator.

9.2.8 R_ACMPHS_Close

```
ssp_err_t R_ACMPHS_Close ( comparator_ctrl_t  * p_api_ctrl )
```

9.2.8.1 Detailed description

Stops the comparator. Implements close.

Table 1352:Return values

Name	Description
SSP_SUCCESS	Instance control block closed successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.2.8.2 Function steps

- Perform parameter checking
- Mark driver as closed
- Disable interrupts.
- Stop the comparator and disable output to VCOUT.
- Enter the module-stop state.

- Release the hardware lock

9.2.9 R_ACMPHS_VersionGet

```
ssp_err_t R_ACMPHS_VersionGet ( ssp_version_t  *const p_version )
```

9.2.9.1 Detailed description

Gets the API and code version. Implements versionGet.

Table 1353:Return values

Name	Description
SSP_SUCCESS	Version information available in p_version.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

9.2.9.2 Function steps

- Return the version number

9.2.10 Extensions

9.2.10.1 acmphs_instance_ctrl_t

acmphs_instance_ctrl_t

Detailed description

Channel instance control block. DO NOT INITIALIZE. Initialization occurs in open.

Variables

- uint32_t open
- R_ACMPHS0_Type * p_reg
- comparator_pin_output_t pin_output
- void(* p_callback)(*p_args)
- void const * p_context
- IRQn_Type irq
- uint8_t channel

9.3 Low Power Analog Comparator

Driver for the Low Power Analog Comparator.

9.3.1 Summary

Extends COMPARATOR Interface.

This module implements the COMPARATOR Interface using the low power analog comparator.

9.3.2 Functions

- R_ACMPLP_Open
- R_ACMPLP_InfoGet
- R_ACMPLP_OutputEnable
- R_ACMPLP_StatusGet
- R_ACMPLP_Close
- R_ACMPLP_VersionGet

9.3.3 Defines

- #define ACMPLP_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define ACMPLP_CODE_VERSION_MINOR

  Initial value: `(0U)`

9.3.4 R_ACMPLP_Open

```
ssp_err_t R_ACMPLP_Open ( comparator_ctrl_t  *const p_api_ctrl ,
 comparator_cfg_t  const *const p_cfg )
```

9.3.4.1 Detailed description

Configures the comparator and starts operation. Callbacks and pin output are not active until outputEnable() is called. outputEnable should be called after the output has stabilized. Implements open.

Comparator inputs must be configured in the application code prior to calling this function.

Table 1354:Return values

Name	Description
SSP_SUCCESS	Open successful.
SSP_ERR_ASSERTION	An input pointer is NULL
SSP_ERR_INVALID_ARGUMENT	An argument is invalid. Window mode (COMPARATOR_MODE_WINDOW) and filter of 1 (COMPARATOR_FILTER_1) are not supported in this implementation.
SSP_ERR_IN_USE	The control block is already open or the hardware lock is taken.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet

9.3.4.2 Function steps

- Verify the control block has not already been initialized.
- Configure interrupt priority. The interrupt is disabled until outputEnable is called.
- Enable clocks to the ACMPLP hardware.
- Set registers controlled by this channel to their default values.
- Set the mode.
- Configure the output polarity.
- Configure the trigger edge.
- Configure the hardware debounce filter.
- Enable the comparator.

9.3.5 R_ACMPLP_InfoGet

```
ssp_err_t R_ACMPLP_InfoGet ( comparator_ctrl_t  *const p_api_ctrl ,
 comparator_info_t  *const p_info )
```

9.3.5.1 Detailed description

Provides the minimum stabilization wait time in microseconds. Implements infoGet.

Table 1355:Return values

Name	Description
SSP_SUCCESS	Information stored in p_info.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.3.5.2 Function steps

- Perform parameter checking

9.3.6 R_ACMPLP_OutputEnable

```
ssp_err_t R_ACMPLP_OutputEnable ( comparator_ctrl_t  *const p_api_ctrl )
```

9.3.6.1 Detailed description

Enables the comparator output, which can be polled using statusGet. Also enables pin output and interrupts as configured during open. Implements outputEnable.

Table 1356:Return values

Name	Description
SSP_SUCCESS	Comparator output is enabled.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.3.6.2 Function steps

- Set the VCOUT output setting for this channel (enabled or disabled).
- Enable interrupts for this channel.

9.3.7 R_ACMPLP_StatusGet

```
ssp_err_t R_ACMPLP_StatusGet ( comparator_ctrl_t  *const p_api_ctrl ,
 comparator_status_t  *const p_status )
```

9.3.7.1 Detailed description

Provides the operating status of the comparator. Implements statusGet.

Table 1357:Return values

Name	Description
SSP_SUCCESS	Operating status of the comparator is provided in p_status.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.3.7.2 Function steps

- Read the operating status of the comparator.

9.3.8 R_ACMPLP_Close

```
ssp_err_t R_ACMPLP_Close ( comparator_ctrl_t  * p_api_ctrl )
```

9.3.8.1 Detailed description

Stops the comparator. Implements close.

Table 1358:Return values

Name	Description
SSP_SUCCESS	Instance control block closed successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.3.8.2 Function steps

- Perform parameter checking
- Mark driver as closed
- Disable interrupts.
- Stop the comparator and disable output to VCOUT.

- Enter the module-stop state.
- Release the hardware lock

9.3.9 R_ACMPLP_VersionGet

```
ssp_err_t R_ACMPLP_VersionGet ( ssp_version_t  *const p_version )
```

9.3.9.1 Detailed description

Gets the API and code version. Implements versionGet.

Table 1359:Return values

Name	Description
SSP_SUCCESS	Version information available in p_version.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

9.3.9.2 Function steps

- Return the version number

9.3.10 Extensions

9.3.10.1 acmplp_instance_ctrl_t

acmplp_instance_ctrl_t

Detailed description

Channel instance control block. DO NOT INITIALIZE. Initialization occurs in open.

Variables

- uint32_t open
- R_ACMPLP_Type * p_reg
- comparator_pin_output_t pin_output
- void(* p_callback)(*p_args)
- void const * p_context
- IRQn_Type irq
- uint8_t channel
- uint8_t output_enabled

- comparator_polarity_invert_t invert

9.4 ADC

Driver for the 14-Bit A/D Converter (ADC14) and 12-bit A/D Converter (ADC12).

This module supports the ADC14 and ADC12 peripherals. It implements the following interfaces:

- ADC Interface

9.4.1 Functions

- R_ADC_Open
- R_ADC_SetSampleStateCount
- R_ADC_ScanConfigure
- R_ADC_InfoGet
- R_ADC_ScanStart
- R_ADC_ScanStop
- R_ADC_CheckScanDone
- R_ADC_Read
- R_ADC_Read32
- R_ADC_Close
- R_ADC_VersionGet
- R_ADC_Calibrate
- R_ADC_OffsetSet

9.4.2 Defines

- #define ADC_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file
- #define ADC_CODE_VERSION_MINOR

  Initial value: `(5U)`
- #define ADC_SAMPLE_STATE_COUNT_MIN

  Initial value: `(7U)`

  Typical values that can be used to modify the sample states. The minimum sample state count value is either 6 or 7 depending on the clock ratios. It is fixed to 7 based on the fact that at the lowest ADC conversion clock supported

(1 MHz) this extra state will lead to at worst a "1 microsecond" increase in conversion time. At 60 MHz the extra sample state will add 16.7 ns to the conversion time.

- #define ADC_SAMPLE_STATE_COUNT_MAX

  Initial value: `(255U)`

- #define ADC_SAMPLE_STATE_HOLD_COUNT_MIN

  Initial value: `(4U)`

  Typical values that can be used for the sample and hold counts for the channels 0-2 Minimum sample and hold states

- #define ADC_SAMPLE_STATE_HOLD_COUNT_DEFAULT

  Initial value: `(24U)`

  Default sample and hold states

- #define ADC_MASK_CHANNEL_0

  Initial value: `(1U<<0U)`

  For ADC Scan configuration adc_channel_cfg_t::scan_mask, scan_mask_group_b, add_mask and sample_hold_mask Use bitwise OR to combine these masks for desired channels and sensors.

- #define ADC_MASK_CHANNEL_1

  Initial value: `(1U<<1U)`

- #define ADC_MASK_CHANNEL_2

  Initial value: `(1U<<2U)`

- #define ADC_MASK_CHANNEL_3

  Initial value: `(1U<<3U)`

- #define ADC_MASK_CHANNEL_4

  Initial value: `(1U<<4U)`

- #define ADC_MASK_CHANNEL_5

  Initial value: `(1U<<5U)`

- #define ADC_MASK_CHANNEL_6

  Initial value: `(1U<<6U)`

- #define ADC_MASK_CHANNEL_7

  Initial value: `(1U<<7U)`

- #define ADC_MASK_CHANNEL_8

  Initial value: `(1U<<8U)`

- #define ADC_MASK_CHANNEL_9

  Initial value: `(1U<<9U)`

- #define ADC_MASK_CHANNEL_10
  Initial value: `(1U<<10U)`
- #define ADC_MASK_CHANNEL_11
  Initial value: `(1U<<11U)`
- #define ADC_MASK_CHANNEL_12
  Initial value: `(1U<<12U)`
- #define ADC_MASK_CHANNEL_13
  Initial value: `(1U<<13U)`
- #define ADC_MASK_CHANNEL_14
  Initial value: `(1U<<14U)`
- #define ADC_MASK_CHANNEL_15
  Initial value: `(1U<<15U)`
- #define ADC_MASK_CHANNEL_16
  Initial value: `(1U<<16U)`
- #define ADC_MASK_CHANNEL_17
  Initial value: `(1U<<17U)`
- #define ADC_MASK_CHANNEL_18
  Initial value: `(1U<<18U)`
- #define ADC_MASK_CHANNEL_19
  Initial value: `(1U<<19U)`
- #define ADC_MASK_CHANNEL_20
  Initial value: `(1U<<20U)`
- #define ADC_MASK_CHANNEL_21
  Initial value: `(1U<<21U)`
- #define ADC_MASK_CHANNEL_22
  Initial value: `(1U<<22U)`
- #define ADC_MASK_CHANNEL_23
  Initial value: `(1U<<23U)`
- #define ADC_MASK_CHANNEL_24
  Initial value: `(1U<<24U)`
- #define ADC_MASK_CHANNEL_25
  Initial value: `(1U<<25U)`
- #define ADC_MASK_CHANNEL_26
  Initial value: `(1U<<26U)`

- #define ADC_MASK_CHANNEL_27

  Initial value: `(1U<<27U)`
- #define ADC_MASK_TEMPERATURE

  Initial value: `(1U<<28UL)`
- #define ADC_MASK_VOLT

  Initial value: `(1U<<29UL)`
- #define ADC_MASK_SENSORS

  Initial value: `(ADC_MASK_TEMPERATURE | ADC_MASK_VOLT)`
- #define ADC_MASK_GROUP_B_OFF

  Initial value: `(0UL)`
- #define ADC_MASK_ADD_OFF

  Initial value: `(0UL)`
- #define ADC_MASK_SAMPLE_HOLD_OFF

  Initial value: `(0U)`
- #define ADC_SAMPLE_HOLD_CHANNELS

  Initial value: `(0x07U)`

  Sample and hold Channel mask. Sample and hold is only available for channel 0,1,2

9.4.3 R_ADC_Open

```
ssp_err_t R_ADC_Open ( adc_ctrl_t  * p_api_ctrl ,  adc_cfg_t  const
*const p_cfg )
```

9.4.3.1 Brief description

The Open function applies power to the A/D peripheral, sets the operational mode, trigger sources, interrupt priority, and configurations for the peripheral as a whole. If interrupt priority is non-zero in BSP_IRQ_Cfg.h, the function takes a callback function pointer for notifying the user at interrupt level whenever a scan has completed. On MCUs where calibration is possible, this function will only return after calibration is completed if enabled in the user configuration. The calibration times vary depending on PCLKB and ADCLK.

9.4.3.2 Detailed description

Table 1360:Return values

Name	Description
SSP_SUCCESS	Call successful.

Table 1360:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	The parameter p_api_ctrl or p_cfg is NULL.
SSP_ERR_INVALID_ARGUMENT	Mode or element of p_cfg structure has invalid value or is illegal based on mode.
SSP_ERR_IN_USE	Calibration timed out.

9.4.3.3 Function steps

- Perform parameter checking
- Verify this unit has not already been initialized
- Set all p_ctrl fields prior to using it in any functions
- Save callback function pointer
- Store the Unit number into the control structure
- Store the user context into the control structure
- Store the mode into the control structure
- Store the alignment into the control structure
- Save the regular mode/Group A trigger in the internal control block
- Save the context
- Confirm the requested unit exists on this MCU and record available channels.
- Lock specified ADC channel
- Retrieve temperature sensor information into control block
- Set ADC and Temperature sensors to a stop state
- Initialize the hardware based on the configuration
- Set ADC and Temperature sensors to a stop state
- Set unused registers to known state (Disable ADC PGA on MCUs that have it)
- Invalid scan mask (initialized for later).
- Mark driver as opened by initializing it to "RADC" in its ASCII equivalent for this unit.
- Return the error code

9.4.4 R_ADC_SetSampleStateCount

```
ssp_err_t R_ADC_SetSampleStateCount ( adc_ctrl_t  * p_api_ctrl ,
 adc_sample_state_t  * p_sample )
```

9.4.4.1 Brief description

Set the sample state count for individual channels. This only needs to be set for special use cases. Normally, use the default values out of Reset.

9.4.4.2 Detailed description

Table 1361:Return values

Name	Description
SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	The parameter p_api_ctrl or p_sample is NULL.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.

9.4.4.3 Function steps

- Perform parameter checking
- Set the sample state count for the specified register
- Return the error code

9.4.5 R_ADC_ScanConfigure

```
ssp_err_t R_ADC_ScanConfigure ( adc_ctrl_t  * p_api_ctrl ,  adc_channel_cfg_t
const *const p_channel_cfg )
```

9.4.5.1 Brief description

Configure the ADC scan parameters. Channel specific settings are set in this function.

9.4.5.2 Detailed description

Table 1362:Return values

Name	Description
SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	The parameter p_api_ctrl or p_channel_cfg is NULL.

Table 1362:Return values (Continued)

Name	Description
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.

NOTE: If the Group Mode Priority configuration is set to ADC_GROUP_A_GROUP_B_CONTINUOUS_SCAN, then since Group B will be scanning continuously, Group B Interrupts are disabled and the application will not receive a call-back for Group B scan completion even if a callback is provided. The application will still receive a callback for Group A scan completion if a callback is provided.

NOTE: If the ADC conversion clock is faster than 50 MHz, the Temperature and Voltage sensor will not be accurate across the operating temperature range, so an error is returned.

9.4.5.3 Function steps

- Perform parameter checking
- Configure the hardware based on the configuration
- Save the scan mask locally; this is required for the infoGet function
- Return the error code

9.4.6 R_ADC_InfoGet

```
ssp_err_t R_ADC_InfoGet ( adc_ctrl_t  * p_api_ctrl ,  adc_info_t  * p_adc_info )
```

9.4.6.1 Brief description

This function returns the address of the lowest number configured channel and the total number of bytes to be read in order to read the results of the configured channels and return the ELC Event name. If no channels are configured, then a length of 0 is returned. This function retrieves the temperature sensor slope. It also returns the calibration data for the sensor if available on this MCU otherwise an invalid calibration data of 0xFFFFFFFF will be returned.

9.4.6.2 Detailed description

Table 1363:Return values

Name	Description
SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- eventInfoGet

NOTE: : Currently this function call does not support Group Mode operation.

9.4.6.3 Function steps

- Perform parameter checking
- Get a pointer to the base register for the current unit
- Retrieve the scan mask of active channels from the control structure
- If at least one channel is configured, determine the highest and lowest configured channels
- Determine the lowest channel that is configured
- Determine the highest channel that is configured
- Set the mask count so that we start with the highest bit of the 32 bit mask
- Initialize the mask result
- Determine the size of data that must be read to read all the channels between and including the highest and lowest channels.
- If no channels are configured, set the return length 0
- Specify the peripheral name in the ELC list
- Set Temp Sensor calibration data to invalid value
- If calibration register is available, retrieve it from the MCU
- Provide the previously retrieved slope information

9.4.7 R_ADC_ScanStart

```
ssp_err_t R_ADC_ScanStart ( adc_ctrl_t  * p_api_ctrl )
```

9.4.7.1 Brief description

This function starts a software scan or enables the hardware trigger for a scan depending on how the triggers were configured in the Open() call. If the Unit was configured for hardware triggering, then this function simply allows the trigger signal (hardware or software) to get to the ADC Unit. The function is not able to control the generation of the trigger itself. If the Unit was configured for software triggering, then this function starts the software triggered scan.

9.4.7.2 Detailed description

Table 1364:Return values

Name	Description
SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_IN_USE	Running scan is still in progress

9.4.7.3 Function steps

- Perform parameter checking
- If the the normal/GroupA trigger is not set to software, then that the Unit is configured for hardware triggering
- Otherwise, enable software triggering
- Check to see if there is an ongoing scan else start the scan
- Return the error code

9.4.8 R_ADC_ScanStop

```
ssp_err_t R_ADC_ScanStop ( adc_ctrl_t  * p_api_ctrl )
```

9.4.8.1 Brief description

This function stops the software scan or disables the Unit from being triggered by the hardware trigger (internal or external) based on what type of trigger the unit was configured for in the Open() function. Stopping a hardware triggered scan via this function does not abort an ongoing scan, but prevents the next scan from occurring. Stopping a software triggered scan aborts an ongoing scan.

9.4.8.2 Detailed description

Table 1365:Return values

Name	Description
SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Unit is not open.

NOTE: Stopping a software scan results in immediate stoppage of the scan irrespective of current state of of the scan. Stopping the hardware scan results in disabling the trigger to prevent future scans from starting but does not affect the current scan.

9.4.8.3 Function steps

- Perform parameter checking
- If the trigger is not software scan, then disallow hardware triggering
- Otherwise, disable software triggering
- Return the error code

9.4.9 R_ADC_CheckScanDone

```
ssp_err_t R_ADC_CheckScanDone ( adc_ctrl_t  * p_api_ctrl )
```

9.4.9.1 Brief description

This function returns the status of any scan process that was started. On supported MCUs, the status of the ADC calibration is returned.

9.4.9.2 Detailed description

Table 1366:Return values

Name	Description
SSP_SUCCESS	Successful; the scan is complete.

Table 1366:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_IN_USE	Running scan or calibration is still in progress.

NOTE: If the peripheral was configured in single scan mode, then the return value of this function is an indication of the scan status. However, if the peripheral was configured in group mode, then the return value of this function could be an indication of either the group A or group B scan state. This is because the ADST bit is set when a scan is ongoing and cleared when the scan is done. This function should normally only be used when using software trigger in single scan mode.

9.4.9.3 Function steps

- Perform parameter checking
- Read status of ADC calibration and return busy status if calibration is ongoing
- Read the status of the ADST bit
- Return the error code

9.4.10 R_ADC_Read

```
ssp_err_t R_ADC_Read ( adc_ctrl_t * p_api_ctrl , adc_register_t const reg_id ,
  adc_data_size_t *const p_data )
```

9.4.10.1 Brief description

This function reads conversion results from a single channel or sensor register.

9.4.10.2 Detailed description

Table 1367:Return values

Name	Description
SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.

Table 1367:Return values (Continued)

Name	Description
SSP_ERR_INVALID_POINTER	The parameter p_data is NULL.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.

9.4.10.3 Function steps

- Perform parameter checking
- Get pointer to appropriate base address. This is repeated here in case parameter checking is disabled.
- Read the data from the requested ADC conversion register and return it
- Return the error code

9.4.11 R_ADC_Read32

```
ssp_err_t R_ADC_Read32 ( adc_ctrl_t  * p_api_ctrl ,  adc_register_t
const reg_id ,   uint32_t *const p_data )
```

9.4.11.1 Brief description

This function reads conversion results from a single channel or sensor register into a 32-bit result.

9.4.11.2 Detailed description

Table 1368:Return values

Name	Description
SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
SSP_ERR_INVALID_POINTER	The parameter p_data is NULL.
SSP_ERR_NOT_OPEN	Unit is not open.
SSP_ERR_INVALID_ARGUMENT	Parameter has invalid value.

9.4.11.3 Function steps

- Read the 16-bit result.
- Left shift the result into the upper 16 bits if the unit is configured for left alignment.

9.4.12 R_ADC_Close

```
ssp_err_t R_ADC_Close ( adc_ctrl_t  * p_api_ctrl )
```

9.4.12.1 Brief description

This function ends any scan in progress, disables interrupts, and removes power to the A/D peripheral.

9.4.12.2 Detailed description

Table 1369:Return values

Name	Description
SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Unit is not open.

9.4.12.3 Function steps

- Perform parameter checking
- Mark driver as closed
- Perform hardware stop for the specific unit
- Release the lock
- Return the error code

9.4.13 R_ADC_VersionGet

```
ssp_err_t R_ADC_VersionGet ( ssp_version_t  *const p_version )
```

9.4.13.1 Brief description

Retrieve the API version number.

9.4.13.2 Detailed description

Table 1370:Return values

Name	Description
SSP_SUCCESS	Successful return.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

9.4.13.3 Function steps

- Return the version number

9.4.14 R_ADC_Calibrate

```
ssp_err_t R_ADC_Calibrate ( adc_ctrl_t  *const p_api_ctrl ,   void
*const p_extend )
```

9.4.14.1 Brief description

This function initiates calibration of the ADC on supported MCUs. Calibration will take a minimum of 24 milliseconds at 32 MHz PCLKB and ADCLK. If ADC interrupts are enabled, a notification is provided via callback when calibration is complete. Otherwise, if the ADC interrupts are disabled then no notification will be provided and the application must check calibration status using infoGet() to determine if the calibration is complete before using the ADC API. Interrupts are enabled in scanStatusGet.

9.4.14.2 Detailed description

Table 1371:Parameters

Name	Direction	Description
p_api_ctrl	in	Pointer to control handle structure
p_extend	in	Unused argument. Pass NULL.

Table 1372:Return values

Name	Description
SSP_SUCCESS	Calibration successfully initiated.

Table 1372:Return values (Continued)

Name	Description
SSP_ERR_INVALID_HW_CONDITION	Hardware is in invalid state to perform calibration due to ongoing scan or scan trigger is enabled.
SSP_ERR_UNSUPPORTED	Calibration not supported on this MCU.
SSP_ERR_ASSERTION	The parameter p_api_ctrl or p_extend is NULL.

9.4.14.3 Function steps

- Perform parameter checking
- ADC Calibration can only happen if there is no ongoing scan and if the scan trigger is disabled
- Set the normal mode interrupt request to occur when calibration is complete
- Initiate calibration
- Return the unsupported error.

9.4.15 R_ADC_OffsetSet

```
ssp_err_t R_ADC_OffsetSet ( adc_ctrl_t  *const p_api_ctrl ,  adc_register_t
const reg_id ,   int32_t offset )
```

9.4.15.1 Detailed description

offsetSet is not supported on the ADC.

9.4.15.2 Function steps

- Return the unsupported error.

9.4.16 Extensions

9.4.16.1 adc_instance_ctrl_t

adc_instance_ctrl_t

Detailed description

ADC instance control block. DO NOT INITIALIZE. Initialized in open.

Variables

- uint16_t unit

  ADC Unit in use.

- int16_t slope_microvolts

  Temperature sensor slope in microvolts/°C.
- adc_mode_t mode

  operational mode
- adc_alignment_t alignment

  alignment
- uint8_t max_resolution

  ADC max resolution: 8, 10, 12, or 14-bit.
- uint8_t pga_available

  PGA available or not on MCU.
- uint8_t tsn_ctrl_available

  Availability of TSN control register.
- uint8_t tsn_calib_available

  Availability of TSn calibration register.
- uint8_t adc_calib_available

  Availability of ADC calibration feature.
- R_TSN_Control_Type * p_tsn_ctrl_regs

  Pointer to temperature control register.
- R_TSN_Calibration_Type * p_tsn_calib_regs

  Pointer to temperature calibration register.
- void const * p_context

  Placeholder for user data.
- void * p_reg

  Base register for this unit.
- void(* callback)(*p_args)

  User callback pointer.
- adc_trigger_t trigger

  Trigger defined for normal mode.
- uint32_t opened

  Boolean to verify that the Unit has been initialized.
- uint32_t scan_mask

  Scan mask used for Normal scan.
- IRQn_Type scan_end_irq

  Scan end IRQ number.

- IRQn_Type scan_end_b_irq

  Scan end group B IRQ number.

9.5 AGT

Driver for the Asynchronous General Purpose Timer (AGT).

9.5.1 Summary

Extends Timer Interface.

HAL High-Level Driver for accessing and configuring AGT timer modes.

The AGT timer functions are used by the Timer to provide timer services.

9.5.2 Functions

- R_AGT_TimerOpen
- R_AGT_Close
- R_AGT_CounterGet
- R_AGT_PeriodSet
- R_AGT_DutyCycleSet
- R_AGT_Reset
- R_AGT_Start
- R_AGT_InfoGet
- R_AGT_Stop
- R_AGT_VersionGet

9.5.3 R_AGT_TimerOpen

```
ssp_err_t R_AGT_TimerOpen ( timer_ctrl_t  *const p_api_ctrl ,  timer_cfg_t  const
*const p_cfg )
```

9.5.3.1 Brief description

Open the AGT channel as a timer, handles required initialization described in hardware manual. Implements open.

9.5.3.2 Detailed description

The Timer Open function configures a single AGT channel for timer mode with parameters specified in the timer Configuration structure. It also sets up the control block for use with subsequent AGT Timer APIs.

This function must be called once prior to calling any other AGT API functions. After a channel is opened, the Open function should not be called again for the same channel without first calling the associated Close function.

The AGT hardware does not support one-shot functionality natively. The one-shot feature is therefore implemented in the AGT HAL layer. For a timer configured as a one-shot timer, the timer is stopped upon the first timer expiration.

The AGT implementation of the general timer can accept an optional timer_on_agt_cfg_t extension parameter. For AGT, the extension specifies the clock to be used as timer source and the output pin configurations. If the extension parameter is not specified (NULL), the default clock PCLKB is used and the output pins are disabled.

The clock divider is selected based on the source clock frequency and the timer period supplied by the caller.

Table 1373:Return values

Name	Description
SSP_SUCCESS	Initialization was successful and timer has started.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_cfg, p_ctrl, or the configuration channel ID exceeds AGT_MAX_CH, or the configuration mode is invalid.
SSP_ERR_IN_USE	The channel specified has already been opened.
SSP_ERR_IRQ_BSP_DISABLED	A required interrupt has not been enabled in the BSP.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet
- eventInfoGet
- systemClockFreqGet

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

9.5.3.3 Function steps

- Verify channel is not already used
- Power on the AGT channel.
- Wait for counter to stop.
- Clear AGTO output
- Clear TEDGSEL bit to normal output.
- Set the AGT mode based on agt_mode value.
- Make sure period is valid, then set period
- Start the timer if requested by user

- All done.

9.5.4 R_AGT_Close

```
ssp_err_t R_AGT_Close ( timer_ctrl_t  *const p_api_ctrl )
```

9.5.4.1 Detailed description

Stops counter, disables interrupts, disables output pins, and clears internal driver data. Implements close.

Table 1374:Return values

Name	Description
SSP_SUCCESS	The AGT Timer channel is successfully closed.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_NOT_OPEN	The AGT channel is not opened.

9.5.4.2 Function steps

- Cleanup the device: Stop counter, disable interrupts, and power down if no other channels are in use.
- Clear the TOE (output enable) bit
- Clear the TEDGSEL bit
- Unlock channel

9.5.5 R_AGT_CounterGet

```
ssp_err_t R_AGT_CounterGet ( timer_ctrl_t  *const p_api_ctrl ,  timer_size_t
*const p_value )
```

9.5.5.1 Detailed description

Retrieve and store counter value in provided p_value pointer. Implements counterGet.

Table 1375:Return values

Name	Description
SSP_SUCCESS	Counter value read, p_value is valid.
SSP_ERR_ASSERTION	The p_ctrl or p_value parameter was null

Table 1375:Return values (Continued)

Name	Description
SSP_ERR_NOT_OPEN	The channel is not opened.

9.5.5.2 Function steps

- Read counter value

9.5.6 R_AGT_PeriodSet

```
ssp_err_t R_AGT_PeriodSet ( timer_ctrl_t  *const p_api_ctrl ,  timer_size_t
const period ,  timer_unit_t  const unit )
```

9.5.6.1 Detailed description

Sets period value provided. Implements periodSet.

Table 1376:Return values

Name	Description
SSP_SUCCESS	Period value written successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_INVALID_ARG	One of the following is invalid: • p_period->unit: must be one of the options from timer_size_t::unit • p_period->value: must result in a period supported by the clock source specified during the Open call.
SSP_ERR_NOT_OPEN	The channel is not opened.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet
- systemClockFreqGet

9.5.6.2 Function steps

- Make sure period is valid, then set period

9.5.7 R_AGT_DutyCycleSet

```
ssp_err_t R_AGT_DutyCycleSet ( timer_ctrl_t  *const p_api_ctrl ,  timer_size_t
const duty_cycle ,  timer_pwm_unit_t  const unit ,   uint8_t const pin )
```

9.5.7.1 Detailed description

Setting duty cycle is not supported by this driver. Implements dutyCycleSet.

Table 1377:Return values

Name	Description
SSP_SUCCESS	Once duty cycle set successfully.
SSP_ERR_INVALID_ARGUMENT	If any of the argument is invalid
SSP_ERR_NOT_OPEN	The channel is not opened.

9.5.7.2 Function steps

- Set duty cycle.

9.5.8 R_AGT_Reset

```
ssp_err_t R_AGT_Reset ( timer_ctrl_t  *const p_api_ctrl )
```

9.5.8.1 Detailed description

Resets the counter value to the period that was originally set. Implements reset.

Table 1378:Return values

Name	Description
SSP_SUCCESS	Counter value written successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.5.8.2 Function steps

- Save running status for restart, if already running.

- Reset the counter to the period originally sent in.
- Restart the AGT channel if it was running.

9.5.9 R_AGT_Start

```
ssp_err_t R_AGT_Start ( timer_ctrl_t  *const p_api_ctrl )
```

9.5.9.1 Detailed description

Starts timer. Implements start.

Table 1379:Return values

Name	Description
SSP_SUCCESS	Timer successfully started.
SSP_ERR_ASSERTION	The p_ctrl parameter is null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.5.9.2 Function steps

- Start timer

9.5.10 R_AGT_InfoGet

```
ssp_err_t R_AGT_InfoGet ( timer_ctrl_t  *const p_api_ctrl ,  timer_info_t
*const p_info )
```

9.5.10.1 Brief description

Get timer information and store it in provided pointer p_info. Implements infoGet.

9.5.10.2 Detailed description

Table 1380:Return values

Name	Description
SSP_SUCCESS	Period, status, count direction, frequency value written to caller's structure successfully.

Table 1380:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	The p_ctrl or p_period_counts parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_INVALID_HW_CONDITION	Invalid hardware setting is detected.

9.5.10.3 Function steps

- Get and store period
- Get and store clock frequency
- AGT supports only counting down direction

9.5.11 R_AGT_Stop

```
ssp_err_t R_AGT_Stop ( timer_ctrl_t  *const p_api_ctrl )
```

9.5.11.1 Detailed description

Stops the AGT channel specified by the handle (control block). This API implements stop. This API does not reset the channel or power it down.

Table 1381:Return values

Name	Description
SSP_SUCCESS	Timer successfully stopped.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.5.12 R_AGT_VersionGet

```
ssp_err_t R_AGT_VersionGet ( ssp_version_t  *const p_version )
```

9.5.12.1 Detailed description

Sets driver version based on compile time macros. Implements versionGet.

Table 1382:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

9.5.13 API Data

9.5.13.1 agt_count_source_t

agt_count_source_t

Detailed description

Count source

Enumerated values

Name	Description
AGT_CLOCK_PCLKB	Counter clock source is PCLKB when AGT_CLOCK_PCLKB, AGT_CLOCK_PCLKB_DIV_2, or AGT_CLOCK_PCLKB_DIV_8 is selected. The PCLKB divisor is selected automatically at runtime to the optimal value of PCLKB/1, PCLKB/2, or PCLKB/8. If the timer_cfg_t::unit is TIMER_UNIT_PERIOD_RAW_COUNTS, the timer_cfg_t::period should be the desired value in PCLKB counts, even if the value would exceed 16 bits. For example, if a period of 0x30000 counts is requested, a divisor of PCLKB/8 is be selected and the counter underflows after 0x6000 counts.
AGT_CLOCK_PCLKB_DIV_8	Superseded: See AGT_CLOCK_PCLKB.
AGT_CLOCK_PCLKB_DIV_2	Superseded: See AGT_CLOCK_PCLKB.
AGT_CLOCK_LOCO	Divided clock LOCO specified y bits CKS[2:0] in the AGTMR2 register.
AGT_CLOCK_AGT0_UNDERFLOW	Underflow event signal from AGT0.
AGT_CLOCK_FSUB	Divided clock fSUB specified by bits CKS[2:0] in the AGTMR2 register.

9.5.14 Extensions

9.5.14.1 agt_instance_ctrl_t

agt_instance_ctrl_t

Detailed description

Channel control block. DO NOT INITIALIZE. Initialization occurs when open is called.

Variables

- void(* p_callback)(*p_args)

  Callback provided when a timer ISR occurs. NULL indicates no CPU interrupt.
- void const * p_context

  Placeholder for user data. Passed to the user callback in timer_callback_args_t.
- void * p_reg

  Base register for this channel.
- uint32_t open

  Whether or not channel is open.
- uint16_t period

  Current timer period (counts)
- uint8_t channel

  Channel number.
- IRQn_Type irq

  Counter overflow IRQ number.
- timer_mode_t mode

  Timer mode.

9.5.14.2 timer_on_agt_cfg_t

timer_on_agt_cfg_t

Detailed description

Optional AGT extension data structure.

Variables

- agt_count_source_t count_source

  AGT channel clock source. Valid values are: AGT_CLOCK_PCLKB, AGT_CLOCK_LOCO, AGT_-CLOCK_FSUB.
- bool agto_output_enabled

  AGTO pin is enabled for output compare (true, false)

- bool agtio_output_enabled

  AGTIO pin is enabled for output compare (true, false)
- bool output_inverted

  Output inverted (true, false)
- bool agtoa_output_enable

  Enable comparator A output pin (true, false)
- bool agtob_output_enable

  Enable comparator B output pin (true, false)

9.6 CAC

Driver for the Clock Frequency Accuracy Measurement Circuit (CAC).

9.6.1 Summary

This module supports the CAC peripheral.

9.6.2 Functions

- R_CAC_Open
- R_CAC_Close
- R_CAC_StopMeasurement
- R_CAC_StartMeasurement
- R_CAC_Reset
- R_CAC_Read
- R_CAC_VersionGet
- cac_setup
- cac_disable_nvic_interrupts
- cac_fmi_setup
- cac_setup_vectors
- cac_irq_cfg
- cac_enable_interrupts
- cac_frequency_error_isr
- cac_overflow_isr
- cac_measurement_end_isr

9.6.3 Defines

- #define CAC_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define CAC_CODE_VERSION_MINOR

  Initial value: `(7U)`

9.6.4 R_CAC_Open

```
ssp_err_t R_CAC_Open ( cac_ctrl_t  *const p_api_ctrl ,  cac_cfg_t  const
*const p_cfg )
```

9.6.4.1 Brief description

Initialize the CAC peripheral.

9.6.4.2 Detailed description

The Open function applies power to the CAC peripheral, checks/sets the interrupt priority, and configures the CAC based on the provided user configuration settings. If a user defined callback function has been provided in the configuration, then the CAC interrupt(s) will be enabled and the user callback function called accordingly. Implements r_cac_t::open.

Table 1383:Return values

Name	Description
SSP_SUCCESS	CAC is available and available for measurement(s).
SSP_ERR_ASSERTION	Null Pointer.
SSP_ERR_INVALID_ARGUMENT	One or more configuration options are invalid.
SSP_ERR_HW_LOCKED	Hardware lock for CAC peripheral is already taken.
SSP_ERR_INVALID_CAC_REF_CLOCK	Measured clock rate smaller than reference clock rate.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet
- eventInfoGet

NOTE: There is only a single CAC peripheral. It is not reentrant.

9.6.4.3 Function steps

- g_cac_version is accessed by the ASSERT macro only and so compiler toolchain can issue a warning that they are not accessed. The code below eliminates this warning and also ensures these data structures are not optimised away.
- Eliminate warning if parameter checking is disabled.
- Take the hardware lock for the CAC.
- Setup the interrupt vectors and priorities
- Return the hardware lock for the CAC.
- Apply power to the peripheral
- Configure the CAC per the configuration.
- Store the callback and context information
- Mark driver as open by initializing it to "CAC" - its ASCII equivalent.

9.6.5 R_CAC_Close

```
ssp_err_t R_CAC_Close ( cac_ctrl_t  *const p_api_ctrl )
```

9.6.5.1 Brief description

Release any resources that were allocated by the Open() or any subsequent CAC operations. Implements r_cac_t::close.

9.6.5.2 Detailed description

Table 1384:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.
SSP_ERR_NOT_OPEN	R_CAC_Open has not been successfully called.

9.6.5.3 Function steps

- Eliminate warning if parameter checking is disabled.
- < Disable interrupts in the peripheral and NVIC
- Disable interrupts in NVIC.
- Disable the CAC ints.
- Stop measuring.

- Power down peripheral.
- Return the hardware lock for the CAC.

9.6.6 R_CAC_StopMeasurement

```
ssp_err_t R_CAC_StopMeasurement ( cac_ctrl_t  *const p_api_ctrl )
```

9.6.6.1 Brief description

Stop the CAC measurement process. Implements r_cac_t::stopMeasurement.

9.6.6.2 Detailed description

Table 1385:Return values

Name	Description
SSP_SUCCESS	CAC measuring has been stoped.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.
SSP_ERR_NOT_OPEN	R_CAC_Open has not been successfully called.

9.6.6.3 Function steps

- Eliminate warning if parameter checking is disabled.
- Stop measuring.

9.6.7 R_CAC_StartMeasurement

```
ssp_err_t R_CAC_StartMeasurement ( cac_ctrl_t  *const p_api_ctrl )
```

9.6.7.1 Brief description

Start the CAC measurement process. Implements r_cac_t::startMeasurement.

9.6.7.2 Detailed description

Table 1386:Return values

Name	Description
SSP_SUCCESS	CAC measurement started.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.
SSP_ERR_NOT_OPEN	R_CAC_Open has not been successfully called.
SSP_ERR_CLOCK_INACTIVE	Either the provided Measurement or Reference clock is not running

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- clockCheck

9.6.7.3 Function steps

- Eliminate warnings if parameter checking is disabled.
- Start measuring.

9.6.8 R_CAC_Reset

```
ssp_err_t R_CAC_Reset ( cac_ctrl_t  *const p_api_ctrl )
```

9.6.8.1 Brief description

Resets the Overflow, Measurement End and Frequency Error interrupt flags. This will clear any of the CASTR status bits that have been set, but only if the CFME bit is off (Not measuring). Implements r_cac_t::reset.

9.6.8.2 Detailed description

Table 1387:Return values

Name	Description
SSP_SUCCESS	CAC reset completed.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.
SSP_ERR_NOT_OPEN	R_CAC_Open has not been successfully called.

9.6.8.3 Function steps

- Eliminate warning if parameter checking is disabled.
- Reset the CAC.

9.6.9 R_CAC_Read

```
ssp_err_t R_CAC_Read ( cac_ctrl_t  *const p_api_ctrl ,   uint8_t
*const p_status ,   uint16_t *const p_counter )
```

9.6.9.1 Brief description

Read and return the CAC status and counter registers. Implements r_cac_t::read.

9.6.9.2 Detailed description

Table 1388:Return values

Name	Description
SSP_SUCCESS	CAC read successful.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.
SSP_ERR_NOT_OPEN	R_CAC_Open has not been successfully called.

9.6.9.3 Function steps

- Eliminate warning if parameter checking is disabled.

9.6.10 R_CAC_VersionGet

```
ssp_err_t R_CAC_VersionGet ( ssp_version_t  *const p_version )
```

9.6.10.1 Brief description

Get the API and code version information.

9.6.10.2 Detailed description

Table 1389:Return values

Name	Description
SSP_SUCCESS	Version info returned.

NOTE: This function is reentrant.

9.6.11 cac_setup

```
cac_setup ( cac_instance_ctrl_t  *const p_ctrl ,  cac_cfg_t  const
*const p_cfg )
```

9.6.11.1 Detailed description

Configure the CAC per the user's configuration

Table 1390:Parameters

Name	Direction	Description
p_ctrl	in	- CAC instance control block
p_cfg	in	- pointer to the CAC configuration settings.

9.6.11.2 Function steps

- Disable interrupt in ICU

9.6.12 cac_disable_nvic_interrupts

```
cac_disable_nvic_interrupts ( cac_instance_ctrl_t  *const p_ctrl )
```

9.6.12.1 Detailed description

Disable the CAC interrupts in the NVIC

Table 1391:Parameters

Name	Direction	Description
p_ctrl	in	- CAC instance control block

None

9.6.13 cac_fmi_setup

```
ssp_err_t cac_fmi_setup ( cac_instance_ctrl_t  *const p_ctrl ,  ssp_feature_t
* p_ssp )
```

9.6.13.1 Brief description

This function gets the interrupt vector indexes from the FMI for the CAC interrupts.

9.6.13.2 Detailed description

Table 1392:Parameters

Name	Direction	Description
p_ctrl	in	- CAC instance control block
p_ssp	in	- pointer to ssp feature information

Table 1393:Return values

Name	Description
SSP_SUCCESS	FMI based setup success.
SSP_ERR_ASSERTION	Problem getting FMI information.

9.6.14 cac_setup_vectors

```
ssp_err_t cac_setup_vectors ( cac_instance_ctrl_t  *const p_ctrl ,  cac_cfg_t
const *const p_cfg )
```

9.6.14.1 Brief description

This function verfies that there is a callback function specified if one or more interrupts are enabled and sets the vector table entries and priority for enabled CAC interrupts.

9.6.14.2 Detailed description

Table 1394:Parameters

Name	Direction	Description
p_ctrl	in	- CAC instance control block
p_cfg	in	- pointer to the CAC configuration settings.

Table 1395:Return values

Name	Description
SSP_SUCCESS	Enabled vectors and priorities configured.
SSP_ERR_INVALID_POINTER	Interrupt specified with NULL callback.

9.6.15 cac_irq_cfg

```
 cac_irq_cfg ( cac_instance_ctrl_t  *const p_ctrl ,   bool state ,  cac_cfg_t
const *const p_cfg )
```

9.6.15.1 Brief description

This function enables or disables the CAC interrupts. There are 3 possible CAC interrupts, each of which the user may choose to enable or disable. These are the Frequency Error interrupt, the Measurement Complete interrupt and the Overflow interrupt. If the caller has provided a callback function as part of the provided p_cfg pointer, then that function will be called as a result of the interrupt.

9.6.15.2 Detailed description

Table 1396:Parameters

Name	Direction	Description
p_ctrl	in	CAC instance control block

Table 1396:Parameters (Continued)

Name	Direction	Description
state	in	true ==> enable interrupts, false ==> disable interrupts.
p_cfg	in	Pointer to the CAC configuration structure.

9.6.15.3 Function steps

- Regardless of whether we will enable or disable, Clear the Interrupt Request bits
- Enable the Interrupts if requested
- Assign the callback
- Disable interrupt in NVIC
- < Disable interrupts in the peripheral

9.6.16 cac_enable_interrupts

```
 cac_enable_interrupts ( cac_instance_ctrl_t  *const p_ctrl ,  cac_cfg_t  const
*const p_cfg )
```

9.6.16.1 Brief description

This function enables the CAC interrupts enabled in the supplied configuration.

9.6.16.2 Detailed description

Table 1397:Parameters

Name	Direction	Description
p_ctrl	in	CAC instance control block
p_cfg	in	Pointer to the CAC configuration structure.

9.6.16.3 Function steps

- < Disable interrupts in the peripheral
- Enable/Disable CAC interrupts as requested by the user.

9.6.17 cac_frequency_error_isr

```
cac_frequency_error_isr ( void )
```

9.6.17.1 Brief description

CAC Frequency error interrupt routine.

9.6.17.2 Detailed description

This function implements the CAC Frequency error isr. The function clears the interrupt request source on entry populates the callback structure with the relevant event, and providing a callback routine has been provided, calls the callback function with the event.

Table 1398:Return values

Name	Description
none	

9.6.17.3 Function steps

- The CAC interrupts are level based which means the bits in the peripheral must be cleared before clearing the IR bit in the IELSRn register.
- Clear the interrupt condition in the CAC peripheral
- Clear the IR bit in the IELSRn register
- Set data to identify callback to user, then call user callback.

9.6.18 cac_overflow_isr

```
cac_overflow_isr ( void )
```

9.6.18.1 Brief description

CAC Overflow interrupt routine.

9.6.18.2 Detailed description

This function implements the CAC overflow isr. The function clears the interrupt request source on entry populates the callback structure with the relevant event, and providing a callback routine has been provided, calls the callback function with the event.

Table 1399:Return values

Name	Description
none	

9.6.18.3 Function steps

- The CAC interrupts are level based which means the bits in the peripheral must be cleared before clearing the IR bit in the IELSRn register.
- Clear the interrupt condition in the CAC peripheral
- Clear the IR bit in the IELSRn register
- Set data to identify callback to user, then call user callback.

9.6.19 cac_measurement_end_isr

```
cac_measurement_end_isr (  void  )
```

9.6.19.1 Brief description

CAC Measurement Complete interrupt routine.

9.6.19.2 Detailed description

This function implements the CAC measurement complete isr. The function clears the interrupt request source populates the callback structure with the relevant event, and providing a callback routine has been provided, calls the callback function with the event.

Table 1400:Return values

Name	Description
none	

9.6.19.3 Function steps

- The CAC interrupts are level based which means the bits in the peripheral must be cleared before clearing the IR bit in the IELSRn register.
- Clear the interrupt condition in the CAC peripheral
- Clear the IR bit in the IELSRn register

- Set data to identify callback to user, then call user callback.

9.6.20 Extensions

9.6.20.1 cac_instance_ctrl_t

cac_instance_ctrl_t

Detailed description

CAC instance control block. DO NOT INITIALIZE.

Variables

- void * p_reg

  Pointer to register base address.
- void(* p_callback)(*cb_data)

  Called from the ISR.
- void const * p_context

  Passed to the callback.
- IRQn_Type frequency_error_irq

  Frequency error IRQ number.
- IRQn_Type measurement_end_irq

  Measurement end IRQ number.
- IRQn_Type overflow_irq

  Overflow IRQ number.
- uint32_t cac_api_open

  Set to "CAC" once API has been successfully opened.
- bool cac_continous_mode

  Set as a result of the Open() call.
- bsp_lock_t cac_lock

  CAC commands software lock.
- cac_clock_source_t measurement_clock

  Clock specified in Open() as the measurement clock.
- cac_clock_source_t reference_clock

  Clock specified in Open() as the reference clock.
- void const * p_extend

9.7 CAN

Driver for CAN, Controller Area Network.

This module supports the Controller Area Network peripheral. It implements the following interfaces:

- CAN Interface

9.7.1 Functions

- R_CAN_Open
- R_CAN_Close
- R_CAN_Read
- R_CAN_Write
- R_CAN_Control
- R_CAN_InfoGet
- R_CAN_VersionGet

9.7.2 Defines

- #define CAN_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define CAN_CODE_VERSION_MINOR

  Initial value: `(4U)`

9.7.3 R_CAN_Open

```
ssp_err_t R_CAN_Open ( can_ctrl_t  *const p_ctrl ,  can_cfg_t  const
*const p_cfg )
```

9.7.3.1 Brief description

Open and configure the CAN channel for operation. Implements open

9.7.3.2 Detailed description

Table 1401:Return values

Name	Description
SSP_SUCCESS	Channel opened successfully
SSP_ERR_INVALID_ARGUMENT	Invalid channel passed as argument.
SSP_ERR_HW_LOCKED	Lock already owned by another user.
SSP_ERR_CAN_MODE_SWITCH_FAILED	Channel failed to switch modes.
SSP_ERR_CAN_INIT_FAILED	Channel failed to initialize.
SSP_ERR_ASSERTION	Null pointer presented.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet
- eventInfoGet
- clockCheck

9.7.3.3 Function steps

- Check for valid parameters.
- Make sure the feature exists on this MCU.
- Return if failed to get feature information.
- Try to get channel lock.
- Return if channel is already open so return error
- Enter module start state.
- Disable interrupts while initializing
- Initialize and configure CAN module to run.
- Set channel, callback function, context, id mode, mailbox count, message mode, op mode and opened status.
- If successful, Lookup and store IRQ numbers. Enable interrupts.
- If successful, Mark the control block as open
- If the device failed to initialize, disable interrupts, stop and unlock the hardware and mark the control block as closed.
- Process errors before returning.
- Log error or assertion.

9.7.4 R_CAN_Close

```
ssp_err_t R_CAN_Close ( can_ctrl_t  *const p_ctrl )
```

9.7.4.1 Brief description

Close the CAN channel. Implements close

9.7.4.2 Detailed description

End of function R_CAN_Open

Table 1402:Return values

Name	Description
SSP_SUCCESS	Channel closed successfully.
SSP_ERR_NOT_OPEN	Control block not open.
SSP_ERR_ASSERTION	Null pointer presented.

9.7.4.3 Function steps

- Mark the channel not open so other APIs cannot use it.
- Disable transmit, receive and error interrupts
- Enable module stop for the CAN channel
- Unlock the CAN channel

9.7.5 R_CAN_Read

```
ssp_err_t R_CAN_Read ( can_ctrl_t  *const p_ctrl ,   uint32_t mailbox ,
 can_frame_t  *const p_frame )
```

9.7.5.1 Brief description

Read data from the CAN channel. Return up to eight bytes read from the channel mailbox. Implements read

9.7.5.2 Detailed description

Table 1403:Return values

Name	Description
SSP_SUCCESS	Data successfully read.
SSP_ERR_NOT_OPEN	Control block not open.
SSP_ERR_CAN_DATA_UNAVAILABLE	No data available.
SSP_ERR_CAN_TRANSMIT_MAILBOX	Mailbox is not setup for receive.
SSP_ERR_ASSERTION	Null pointer presented.

9.7.5.3 Function steps

- Check for receive data
- Get frame data.
- Check for other mailboxes in an overrun state.
- Check for other mailboxes with received messages pending.

9.7.6 R_CAN_Write

```
ssp_err_t R_CAN_Write ( can_ctrl_t  *const p_ctrl ,   uint32_t mailbox ,
 can_frame_t  *const p_frame )
```

9.7.6.1 Brief description

Write data to the CAN channel. Write up to eight bytes to the channel mailbox. Implements write

9.7.6.2 Detailed description

Table 1404:Return values

Name	Description
SSP_SUCCESS	Operation succeeded.
SSP_ERR_NOT_OPEN	Control block not open.
SSP_ERR_CAN_TRANSMIT_NOT_READY	Transmit in progress, cannot write data at this time.

Table 1404:Return values (Continued)

Name	Description
SSP_ERR_CAN_RECEIVE_MAILBOX	Mailbox is setup for receive and cannot send.
SSP_ERR_INVALID_ARGUMENT	Data length or frame type invalid.
SSP_ERR_ASSERTION	Null pointer presented

9.7.6.3 Function steps

- Check transmit ready flag.
- Transmit ready flag is not set, return error/status.
- Transmit ready flag set, so clear it.
- Send transmit frame.

9.7.7 R_CAN_Control

```
ssp_err_t R_CAN_Control ( can_ctrl_t  *const p_ctrl ,  can_command_t
const command ,   void * p_data )
```

9.7.7.1 Brief description

CAN Control is used to control extended features. Implements control

9.7.7.2 Detailed description

Table 1405:Return values

Name	Description
SSP_SUCCESS	Operation succeeded.
SSP_ERR_NOT_OPEN	Control block not open.
SSP_ERR_INVALID_ARGUMENT	Invalid command.
SSP_ERR_ASSERTION	Null pointer presented
SSP_ERR_CAN_MODE_SWITCH_FAILED	Switching modes failed.

9.7.7.3 Function steps

- Verify command is CAN_COMMAND_MODE_SWITCH
- Change operating mode. Returns false if invalid mode or mode switch failed.
- Save mode for diagnostic purposes.

9.7.8 R_CAN_InfoGet

```
ssp_err_t R_CAN_InfoGet ( can_ctrl_t  *const p_ctrl ,   can_info_t
*const p_info )
```

9.7.8.1 Brief description

Get CAN state and status information for the channel. Implements infoGet

9.7.8.2 Detailed description

Table 1406:Return values

Name	Description
SSP_SUCCESS	Operation succeeded.
SSP_ERR_NOT_OPEN	Control block not open.
SSP_ERR_CAN_DATA_UNAVAILABLE	Channel failed to return info.
SSP_ERR_ASSERTION	Null pointer presented

9.7.8.3 Function steps

- Get status for channel.
- Error encountered when retrieving info.
- Save the operation mode

9.7.9 R_CAN_VersionGet

```
ssp_err_t R_CAN_VersionGet ( ssp_version_t  *const p_version )
```

9.7.9.1 Brief description

Get CAN module code and API versions. Implements versionGet

9.7.9.2 Detailed description

Table 1407:Return values

Name	Description
SSP_SUCCESS	Operation succeeded.
SSP_ERR_ASSERTION	Null pointer presented note This function is reentrant.

9.7.9.3 Function steps

- Return module version information.

9.7.10 Extensions

9.7.10.1 can_instance_ctrl_t

can_instance_ctrl_t

Detailed description

CAN Instance Control Block

Variables

- uint32_t channel

  Channel number.

  Parameters to control CAN peripheral device
- uint32_t open

  Open status of channel.
- can_mode_t operation_mode

  Can operation mode.
- can_id_mode_t id_mode

  Standard or Extended ID mode.
- uint32_t mailbox_count

  Number of mailboxes.
- can_mailbox_t * p_mailbox

  Pointer to mailboxes.

- can_message_mode_t message_mode

  Overwrite message or overrun.
- can_clock_source_t clock_source

  Clock source. CANMCLK or PCLKB.
- void(* p_callback)(*p_args)

  Pointer to callback function.

  Parameters to process CAN Event
- void const * p_context

  Pointer to the higher level device context.
- void * p_reg

  Pointer to register base address.
- IRQn_Type error_irq

  Error IRQ number.
- IRQn_Type mailbox_rx_irq

  Receive mailbox IRQ number.
- IRQn_Type mailbox_tx_irq

  Transmit mailbox IRQ number.

9.7.10.2 can_extended_cfg_t

can_extended_cfg_t

Detailed description

CAN clock configuration and mailbox mask to be pointed to by p_extend.

Variables

- can_clock_source_t clock_source

  Source of the CAN clock.
- uint32_t * p_mailbox_mask

  Mailbox mask, one for every 4 mailboxes.

9.8 CGC

Driver for the Clock Generation Circuit.

Clock Generation Circuit API.

This module supports the Clock Generation Circuit. It implements the following interfaces:

- CGC Interface

9.8.1 Functions

- R_CGC_Init
- R_CGC_ClocksCfg
- R_CGC_ClockStart
- R_CGC_ClockStop
- R_CGC_SystemClockSet
- R_CGC_SystemClockGet
- R_CGC_SystemClockFreqGet
- R_CGC_ClockCheck
- R_CGC_OscStopDetect
- R_CGC_OscStopStatusClear
- R_CGC_BusClockOutCfg
- R_CGC_BusClockOutEnable
- R_CGC_BusClockOutDisable
- R_CGC_ClockOutCfg
- R_CGC_ClockOutEnable
- R_CGC_ClockOutDisable
- R_CGC_LCDClockCfg
- R_CGC_LCDClockEnable
- R_CGC_LCDClockDisable
- R_CGC_SDADCClockCfg
- R_CGC_SDADCClockEnable
- R_CGC_SDADCClockDisable
- R_CGC_SDRAMClockOutEnable
- R_CGC_SDRAMClockOutDisable
- R_CGC_USBClockCfg
- R_CGC_SystickUpdate
- R_CGC_VersionGet
- r_cgc_stabilization_wait
- r_cgc_clock_start_stop

9.8.2 Defines

- #define CGC_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define CGC_CODE_VERSION_MINOR

  Initial value: `(9U)`

9.8.3 R_CGC_Init

```
ssp_err_t R_CGC_Init (  void  )
```

9.8.3.1 Brief description

Initialize the CGC API.

9.8.3.2 Detailed description

Configures the following for the clock generator module

- If CGC_CFG_SUBCLOCK_AT_RESET_ENABLE is set to true:
  - SubClock drive capacity (Compile time configurable: CGC_CFG_SUBCLOCK_DRIVE)
  - Initial setting for the SubClock

THIS FUNCTION MUST BE EXECUTED ONCE AT STARTUP BEFORE ANY OF THE OTHER CGC FUNCTIONS CAN BE USED OR THE CLOCK SOURCE IS CHANGED FROM THE MOCO.

Table 1408:Return values

Name	Description
SSP_SUCCESS	Clock initialized successfully.
SSP_ERR_HARDWARE_TIMEOUT	Hardware timed out.

9.8.3.3 Function steps

- SubClock will stop only if configurable setting is Enabled

9.8.4 R_CGC_ClocksCfg

```
ssp_err_t R_CGC_ClocksCfg ( cgc_clocks_cfg_t  const *const p_clock_cfg )
```

9.8.4.1 Brief description

Reconfigure all main system clocks.

9.8.4.2 Detailed description

Table 1409:Return values

Name	Description
SSP_SUCCESS	Clock initialized successfully.
SSP_ERR_INVALID_ARGUMENT	Invalid argument used.
SSP_ERR_MAIN_OSC_INACTIVE	PLL Initialization attempted with Main OCO turned off/unstable.
SSP_ERR_CLOCK_ACTIVE	Active clock source specified for modification. This applies specifically to the PLL dividers/multipliers which cannot be modified if the PLL is active. It has to be stopped first before modification.
SSP_ERR_NOT_STABILIZED	The Clock source is not stabilized after being turned off.
SSP_ERR_CLKOUT_EXCEEDED	The main oscillator can be only 8 or 16 MHz.
SSP_ERR_ASSERTION	A NULL is passed for configuration data when PLL is the clock_source.
SSP_ERR_INVALID_MODE	Attempt to start a clock in a restricted operating power control mode.

9.8.5 R_CGC_ClockStart

```
ssp_err_t R_CGC_ClockStart ( cgc_clock_t  clock_source ,  cgc_clock_cfg_t
* p_clock_cfg )
```

9.8.5.1 Brief description

Start the specified clock if it is not currently active.

9.8.5.2 Detailed description

Configures the following when starting the Main Clock Oscillator:

- MainClock drive capacity (Configured based on external clock frequency)
- MainClock stabilization wait time (Compile time configurable: CGC_CFG_MAIN_OSC_WAIT)

- To update the subclock driven capacity, stop the subclock first before calling this function.

Table 1410:Return values

Name	Description
SSP_SUCCESS	Clock initialized successfully.
SSP_ERR_INVALID_ARGUMENT	Invalid argument used.
SSP_ERR_MAIN_OSC_INACTIVE	PLL Initialization attempted with Main OCO turned off/unstable.
SSP_ERR_CLOCK_ACTIVE	Active clock source specified for modification. This applies specifically to the PLL dividers/multipliers which cannot be modified if the PLL is active. It has to be stopped first before modification.
SSP_ERR_NOT_STABILIZED	The Clock source is not stabilized after being turned off.
SSP_ERR_CLKOUT_EXCEEDED	The main oscillator can be only 8 or 16 MHz.
SSP_ERR_ASSERTION	A NULL is passed for configuration data when PLL is the clock_source.
SSP_ERR_INVALID_MODE	Attempt to start a clock in a restricted operating power control mode.
SSP_ERR_HARDWARE_TIMEOUT	Hardware timed out.

9.8.5.3 Function steps

- See if we need to switch to Middle or High Speed mode before starting the PLL.

9.8.6 R_CGC_ClockStop

```
ssp_err_t R_CGC_ClockStop ( cgc_clock_t  clock_source )
```

9.8.6.1 Brief description

Stop the specified clock if it is active and not configured as the system clock.

9.8.6.2 Detailed description

Table 1411:Return values

Name	Description
SSP_SUCCESS	Clock stopped successfully.
SSP_ERR_CLOCK_ACTIVE	Current System clock source specified for stopping. This is not allowed.
SSP_ERR_OSC_STOP_DET_ENABLED	Illegal attempt to stop MOCO when Oscillation stop is enabled.
SSP_ERR_NOT_STABILIZED	Clock not stabilized after starting. A finite stabilization time after starting the clock has to elapse before it can be stopped.
SSP_ERR_INVALID_ARGUMENT	Invalid argument used.
SSP_ERR_HARDWARE_TIMEOUT	Hardware timed out.

9.8.7 R_CGC_SystemClockSet

```
ssp_err_t R_CGC_SystemClockSet ( cgc_clock_t  clock_source ,
 cgc_system_clock_cfg_t  const *const p_clock_cfg )
```

9.8.7.1 Brief description

Set the specified clock as the system clock and configure the internal dividers for ICLK, PCLKA, PCLKB, PCLKC, PCLKD and FCLK.

9.8.7.2 Detailed description

THIS FUNCTION DOES NOT CHECK TO SEE IF THE OPERATING MODE SUPPORTS THE SPECIFIED CLOCK SOURCE AND DIVIDER VALUES. SETTING A CLOCK SOURCE AND DVIDER OUTSIDE THE RANGE SUPPORTED BY THE CURRENT OPERATING MODE WILL RESULT IN UNDEFINED OPERATION.

IF THE LOCO MOCO OR SUBCLOCK ARE CHOSEN AS THE SYSTEM CLOCK, THIS FUNCTION WILL SET THOSE AS THE SYSTEM CLOCK WITHOUT CHECKING FOR STABILIZATION. IT IS UP TO THE USER TO ENSURE THAT LOCO, MOCO OR SUBCLOCK ARE STABLE BEFORE USING THEM AS THE SYSTEM CLOCK.

Additionally this function sets the RAM and ROM wait states for the MCU. For the S7 MCU the ROMWT register controls ROM wait states. For the S3 MCU the MEMWAIT register controls ROM wait states.

Table 1412:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.
SSP_ERR_CLOCK_INACTIVE	The specified clock source is inactive.
SSP_ERR_ASSERTION	The p_clock_cfg parameter is NULL.
SSP_ERR_NOT_STABILIZED	The clock source has not stabilized
SSP_ERR_INVALID_ARGUMENT	Invalid argument used. ICLK is not set as the fastest clock.
SSP_ERR_INVALID_MODE	Peripheral divisions are not valid in sub-osc mode
SSP_ERR_INVALID_MODE	Oscillator stop detect not allowed in sub-osc mode

9.8.7.3 Function steps

- In order to correctly set the ROM and RAM wait state registers we need to know the current (S3A7 only) and requested iclk frequencies.

9.8.8 R_CGC_SystemClockGet

```
ssp_err_t R_CGC_SystemClockGet ( cgc_clock_t  * clock_source ,
 cgc_system_clock_cfg_t  * p_set_clock_cfg )
```

9.8.8.1 Brief description

Return the current system clock source and configuration.

9.8.8.2 Detailed description

Table 1413:Return values

Name	Description
SSP_SUCCESS	Parameters returned successfully.
SSP_ERR_ASSERTION	A NULL is passed for configuration data.
SSP_ERR_ASSERTION	A NULL is passed for clock source.

9.8.9 R_CGC_SystemClockFreqGet

```
ssp_err_t R_CGC_SystemClockFreqGet ( cgc_system_clocks_t  clock ,  uint32_t
* p_freq_hz )
```

9.8.9.1 Brief description

Return the requested internal clock frequency in Hz.

9.8.9.2 Detailed description

Table 1414:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.
SSP_ERR_INVALID_ARGUMENT	Invalid clock specified.
SSP_ERR_ASSERTION	A NULL is passed for frequency data.

9.8.10 R_CGC_ClockCheck

```
ssp_err_t R_CGC_ClockCheck ( cgc_clock_t  clock_source )
```

9.8.10.1 Brief description

Check the specified clock for stability.

9.8.10.2 Detailed description

Table 1415:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.
SSP_ERR_NOT_STABILIZED	Clock not stabilized.
SSP_ERR_CLOCK_ACTIVE	Clock active but not able to check for stability.
SSP_ERR_CLOCK_INACTIVE	Clock not turned on.
SSP_ERR_INVALID_ARGUMENT	Illegal parameter passed.

Table 1415:Return values (Continued)

Name	Description
SSP_ERR_STABILIZED	Clock stabilized.

9.8.11 R_CGC_OscStopDetect

```
ssp_err_t R_CGC_OscStopDetect ( cgc_callback_args_t void(*)(
*p_args) p_callback ,   bool enable )
```

9.8.11.1 Brief description

Enable or disable the oscillation stop detection for the main clock. The MCU will automatically switch the system clock to MOCO when a stop is detected if Main Clock is the system clock. If the system clock is the PLL, then the clock source will not be changed and the PLL free running frequency will be the system clock frequency.

9.8.11.2 Detailed description

Table 1416:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.
SSP_ERR_OSC_STOP_DETECTED	The Oscillation stop detect status flag is set. Under this condition it is not possible to disable the Oscillation stop detection function.
SSP_ERR_ASSERTION	Null pointer passed for callback function when the second argument is "true".
SSP_ERR_ASSERTION	Cannot enable oscillator stop detect in sub-osc speed mode
SSP_ERR_ASSERTION	Invalid peripheral clock divisions for oscillator stop detect

9.8.12 R_CGC_OscStopStatusClear

```
ssp_err_t R_CGC_OscStopStatusClear (  void  )
```

9.8.12.1 Brief description

Clear the Oscillation Stop Detection Status register.

9.8.12.2 Detailed description

This register is not cleared automatically if the stopped clock is restarted. This function blocks for about 3 ICLK cycles until the status register is cleared.

Table 1417:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.
SSP_ERR_OSC_STOP_CLOCK_ACTIVE	The Oscillation Detect Status flag cannot be cleared if the Main Osc or PLL is set as the system clock. Change the system clock before attempting to clear this bit.

9.8.13 R_CGC_BusClockOutCfg

```
ssp_err_t R_CGC_BusClockOutCfg ( cgc_bclockout_dividers_t  divider )
```

9.8.13.1 Brief description

Configure the secondary dividers for BCLKOUT. The primary divider is set using the bsp clock configuration and the R_CGC_SystemClockSet function.

9.8.13.2 Detailed description

Table 1418:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.

9.8.14 R_CGC_BusClockOutEnable

```
ssp_err_t R_CGC_BusClockOutEnable (  void  )
```

9.8.14.1 Brief description

Enable the BCLKOUT output.

9.8.14.2 Detailed description

Table 1419:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.

9.8.15 R_CGC_BusClockOutDisable

```
ssp_err_t R_CGC_BusClockOutDisable (  void  )
```

9.8.15.1 Brief description

Disable the BCLKOUT output.

9.8.15.2 Detailed description

Table 1420:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.

9.8.16 R_CGC_ClockOutCfg

```
ssp_err_t R_CGC_ClockOutCfg ( cgc_clock_t  clock ,
 cgc_clockout_dividers_t  divider )
```

9.8.16.1 Brief description

Configure the dividers for CLKOUT.

9.8.16.2 Detailed description

Table 1421:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.

9.8.17 R_CGC_ClockOutEnable

```
ssp_err_t R_CGC_ClockOutEnable ( void )
```

9.8.17.1 Brief description

Enable the CLKOUT output.

9.8.17.2 Detailed description

Table 1422:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.

9.8.18 R_CGC_ClockOutDisable

```
ssp_err_t R_CGC_ClockOutDisable ( void )
```

9.8.18.1 Brief description

Disable the CLKOUT output.

9.8.18.2 Detailed description

Table 1423:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.

9.8.19 R_CGC_LCDClockCfg

```
ssp_err_t R_CGC_LCDClockCfg ( cgc_clock_t clock )
```

9.8.19.1 Brief description

Configure the source for the segment LCDCLK.

9.8.19.2 Detailed description

Table 1424:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.
SSP_ERR_TIMEOUT	Timed out.
SSP_ERR_INVALID_ARGUMENT	Invalid argument used.

9.8.20 R_CGC_LCDClockEnable

```
ssp_err_t R_CGC_LCDClockEnable (  void  )
```

9.8.20.1 Brief description

Enable the segment LCDCLK output.

9.8.20.2 Detailed description

Table 1425:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.
SSP_ERR_TIMEOUT	Timed out.

9.8.21 R_CGC_LCDClockDisable

```
ssp_err_t R_CGC_LCDClockDisable (  void  )
```

9.8.21.1 Brief description

Disable the segment LCDCLK output.

9.8.21.2 Detailed description

Table 1426:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.
SSP_ERR_TIMEOUT	Timed out.

9.8.22 R_CGC_SDADCClockCfg

```
ssp_err_t R_CGC_SDADCClockCfg ( cgc_clock_t  clock )
```

9.8.22.1 Brief description

Configure the source for the SDADCCLK.

9.8.22.2 Detailed description

Table 1427:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.

9.8.23 R_CGC_SDADCClockEnable

```
ssp_err_t R_CGC_SDADCClockEnable (  void  )
```

9.8.23.1 Brief description

Enable the SDADCCLK output.

9.8.23.2 Detailed description

Table 1428:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.

9.8.24 R_CGC_SDADCClockDisable

```
ssp_err_t R_CGC_SDADCClockDisable ( void )
```

9.8.24.1 Brief description

Disable the SDADCCLK output.

9.8.24.2 Detailed description

Table 1429:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.

9.8.25 R_CGC_SDRAMClockOutEnable

```
ssp_err_t R_CGC_SDRAMClockOutEnable ( void )
```

9.8.25.1 Brief description

Enable the SDCLK output.

9.8.25.2 Detailed description

Table 1430:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.

9.8.26 R_CGC_SDRAMClockOutDisable

```
ssp_err_t R_CGC_SDRAMClockOutDisable ( void )
```

9.8.26.1 Brief description

Disable the SDCLK output.

9.8.26.2 Detailed description

Table 1431:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.

9.8.27 R_CGC_USBClockCfg

```
ssp_err_t R_CGC_USBClockCfg ( cgc_usb_clock_div_t  divider )
```

9.8.27.1 Brief description

Configure the dividers for UCLK.

9.8.27.2 Detailed description

Table 1432:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.
SSP_ERR_INVALID_ARGUMENT	Invalid divider specified.

9.8.28 R_CGC_SystickUpdate

```
ssp_err_t R_CGC_SystickUpdate (  uint32_t period_count ,
 cgc_systick_period_units_t  units )
```

9.8.28.1 Brief description

Re-Configure the systick based on the provided period and current system clock frequency.

9.8.28.2 Detailed description

Table 1433:Parameters

Name	Direction	Description
period_count	in	The duration for the systick period.
units	in	The units for the provided period.

Table 1434:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.
SSP_ERR_INVALID_ARGUMENT	Invalid period specified.
SSP_ERR_ABORTED	Attempt to update systick timer failed.

9.8.29 R_CGC_VersionGet

```
ssp_err_t R_CGC_VersionGet ( ssp_version_t  *const p_version )
```

9.8.29.1 Brief description

Return the driver version.

9.8.29.2 Detailed description

Table 1435:Return values

Name	Description
SSP_SUCCESS	Operation performed successfully.
SSP_ERR_ASSERTION	The parameter p_version is NULL..

9.8.30 r_cgc_stabilization_wait

```
ssp_err_t r_cgc_stabilization_wait ( cgc_clock_t  clock )
```

9.8.31 r_cgc_clock_start_stop

```
ssp_err_t r_cgc_clock_start_stop ( cgc_clock_change_t  clock_state ,
 cgc_clock_t  clock_to_change )
```

9.9 CRC

Driver for the CRC Calculator (CRC).

This module supports the CRC Calculator (CRC). It implements the following interface:

- CRC Interface

9.9.1 Functions

- calculate_polynomial
- R_CRC_Open
- R_CRC_Close
- R_CRC_CalculatedValueGet
- R_CRC_SnoopEnable
- R_CRC_SnoopDisable
- R_CRC_SnoopCfg
- R_CRC_Calculate
- R_CRC_VersionGet

9.9.2 Variables

- s_crc_version
- g_crc_on_crc

9.9.3 Defines

- #define CRC_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define CRC_CODE_VERSION_MINOR

  Initial value: `(4U)`

- #define CRC_SNOOP_MAX_CHANNEL

  Initial value: `(10UL)`

  Maximum number of channels supported by CRC

- #define CRC_OPEN

  Initial value: `(0x00435243ULL)`

  "CRC" in ASCII, used to determine if channel is open.

- #define CRC_ERROR_RETURN

  Initial value:`#define SSP_ERROR_RETURN((a), (err), &g_module_name[0], &s_crc_version)`

  Macro for error logger.

9.9.4 calculate_polynomial

```
ssp_err_t calculate_polynomial ( crc_ctrl_t *const p_api_ctrl , void
* inputBuffer , uint32_t length , uint32_t crc_seed , uint32_t
* calculatedValue )
```

9.9.4.1 Brief description

Perform a CRC calculation on a block of 8-bit data.

9.9.4.2 Detailed description

(end defgroup CRC) Implements calculate

This function performs a CRC calculation on an array of 8-bit/32-bit(for 32-bit polynomial) values and returns an 8-bit32-bit(for 32-bit polynomial) calculated value.

Table 1436:Return values

Name	Description
SSP_SUCCESS	Calculation successful.

9.9.5 R_CRC_Open

```
ssp_err_t R_CRC_Open ( crc_ctrl_t *const p_api_ctrl , crc_cfg_t const
*const p_cfg )
```

9.9.5.1 Brief description

Open the CRC driver module.

9.9.5.2 Detailed description

Implements open

Open the CRC driver module and initialize the driver control block according to the passed-in configuration structure.

Table 1437:Return values

Name	Description
SSP_SUCCESS	Configuration was successful.
SSP_ERR_ASSERTION	p_ctrl or p_cfg is NULL.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet

9.9.5.3 Function steps

- Mark driver as initialized by setting the open value to the ASCII equivalent of "CRC"

9.9.6 R_CRC_Close

```
ssp_err_t R_CRC_Close ( crc_ctrl_t  *const p_api_ctrl )
```

9.9.6.1 Brief description

Close the CRC module driver.

9.9.6.2 Detailed description

Implements close

Table 1438:Return values

Name	Description
SSP_SUCCESS	Configuration was successful.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	The driver is not opened.

9.9.7 R_CRC_CalculatedValueGet

```
ssp_err_t R_CRC_CalculatedValueGet ( crc_ctrl_t  *const p_api_ctrl ,   uint32_t
* calculatedValue )
```

9.9.7.1 Brief description

Return the current calculated value.

9.9.7.2 Detailed description

Implements crcResultGet

CRC calculation operates on a running value. This function returns the current calculated value.

Table 1439:Return values

Name	Description
SSP_SUCCESS	Return of calculated value successful.
SSP_ERR_ASSERTION	Either p_ctrl or calculatedValue is NULL.
SSP_ERR_NOT_OPEN	The driver is not opened.

9.9.8 R_CRC_SnoopEnable

```
ssp_err_t R_CRC_SnoopEnable ( crc_ctrl_t  *const p_api_ctrl ,
  uint32_t crc_seed )
```

9.9.8.1 Brief description

Enable snooping.

9.9.8.2 Detailed description

Implements snoopEnable

Table 1440:Return values

Name	Description
SSP_SUCCESS	Snoop enabled.
SSP_ERR_ASSERTION	Either p_ctrl or crc_seed is NULL.

Table 1440:Return values (Continued)

Name	Description
SSP_ERR_NOT_OPEN	The driver is not opened.

9.9.9 R_CRC_SnoopDisable

```
ssp_err_t R_CRC_SnoopDisable ( crc_ctrl_t  *const p_api_ctrl )
```

9.9.9.1 Brief description

Disable snooping.

9.9.9.2 Detailed description

Implements snoopDisable

Table 1441:Return values

Name	Description
SSP_SUCCESS	Snoop disabled.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	The driver is not opened.

9.9.10 R_CRC_SnoopCfg

```
ssp_err_t R_CRC_SnoopCfg ( crc_ctrl_t  *const p_api_ctrl ,  crc_snoop_cfg_t
*const p_snoop_cfg )
```

9.9.10.1 Brief description

Configure the snoop channel and direction.

9.9.10.2 Detailed description

Implements snoopCfg

The CRC calculator can operate on reads and writes over any of the first ten SCI channels. For example, if set to channel 0, transmit, every byte written out SCI channel 0 is also sent to the CRC calculator as if the value was explicitly written directly to the CRC calculator.

Table 1442:Return values

Name	Description
SSP_SUCCESS	Snoop configured successfully.
SSP_ERR_ASSERTION	- This is due to below conditions • Either p_ctrl or p_snoop_cfg is NULL • snoop_channel is greater than or equal to CRC_SNOOP_MAX_CHANNEL.
SSP_ERR_NOT_OPEN	The driver is not opened.

9.9.11 R_CRC_Calculate

```
ssp_err_t R_CRC_Calculate ( crc_ctrl_t  *const p_api_ctrl ,   void
* inputBuffer ,   uint32_t length ,   uint32_t crc_seed ,   uint32_t
* calculatedValue )
```

9.9.11.1 Brief description

Perform a CRC calculation on a block of 8-bit/32-bit(for 32-bit polynomial) data.

9.9.11.2 Detailed description

Implements calculate

This function performs a CRC calculation on an array of 8-bit/32-bit(for 32-bit polynomial) values and returns an 8-bit/32-bit(for 32-bit polynomial) calculated value

Table 1443:Return values

Name	Description
SSP_SUCCESS	Calculation successful.
SSP_ERR_ASSERTION	Either p_ctrl, inputBuffer, or calculatedValue is NULL.
SSP_ERR_INVALID_ARGUMENT	length value is NULL.
SSP_ERR_NOT_OPEN	The driver is not opened.
SSP_ERR_IN_USE	CRC peripheral is currently in use by another instance of the driver.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet

9.9.11.3 Function steps

- Lock the peripheral during calculation

9.9.12 R_CRC_VersionGet

```
ssp_err_t R_CRC_VersionGet ( ssp_version_t  *const p_version )
```

9.9.12.1 Brief description

Get the driver version based on compile time macros.

9.9.12.2 Detailed description

Implements versionGet

Table 1444:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_version is NULL.

9.9.13 g_crc_on_crc

```
crc_api_t::g_crc_on_crc
```

9.9.13.1 Detailed description

Name of module used by error logger macro Filled in Interface API structure for this Instance.

9.9.13.2 Initialized as

```
g_crc_on_crc=
{
    .open           = R_CRC_Open,
    .close          = R_CRC_Close,
    .crcResultGet   = R_CRC_CalculatedValueGet,
    .snoopEnable    = R_CRC_SnoopEnable,
    .snoopDisable   = R_CRC_SnoopDisable,
```

```
    .snoopCfg       = R_CRC_SnoopCfg,
    .calculate      = R_CRC_Calculate,
    .versionGet     = R_CRC_VersionGet
}
```

9.9.14 Extensions

9.9.14.1 crc_instance_ctrl_t

crc_instance_ctrl_t

Detailed description

Driver instance control structure.

Variables

- R_CRC_Type * p_reg

  Pointer to register base address.

- uint32_t open

  Whether or not channel is open.

- crc_polynomial_t polynomial

  CRC Generating Polynomial Switching (GPS).

- crc_bit_order_t bit_order

  CRC Calculation Switching (LMS).

9.10 CTSU

Driver for the Capacitive Touch Sensing Unit (CTSU).

Driver for operating the CTSU. Provides functions to start up the CTSU and maintain necessary parameters to determine if a channel is being touched or not.

It implements the following interface:

- CTSU Interface defined in r_ctsu_api.h

9.10.1 Functions

- R_CTSU_VersionGet
- R_CTSU_Open
- R_CTSU_Start_Scan
- R_CTSU_Update_Parameters
- R_CTSU_Process

- R_CTSU_Read_Results
- R_CTSU_Control
- R_CTSU_Close

9.10.2 Defines

- #define CTSU_CODE_VERSION_MAJOR
  Initial value: `(1U)`
- #define CTSU_CODE_VERSION_MINOR
  Initial value: `(3U)`
- #define CTSU_MAX_CHANNELS
  Initial value: `(36)`

9.10.3 R_CTSU_VersionGet

```
ssp_err_t R_CTSU_VersionGet ( ssp_version_t  *const p_version )
```

9.10.4 R_CTSU_Open

```
ssp_err_t R_CTSU_Open ( ctsu_ctrl_t  * p_api_ctrl ,  ctsu_cfg_t  * p_cfg )
```

9.10.4.1 Brief description

Initializes the CTSU, initializes software driver parameters for touch determination, with initial auto-tuning(optional) and baseline initialization(optional).

9.10.4.2 Detailed description

Table 1445:Return values

Name	Description
SSP_SUCCESS	No errors. CTSU succesfully initialized.
SSP_ERR_HW_LOCKED	CTSU driver already in use
SSP_ERR_ASSERTION	Invalid mode of operation (SELF or MUTUAL capacitance), invalid soft option, or invalid close option
SSP_ERR_INVALID_POINTER	NULL pointer passed as an argument for a required parameter -OR- Memory allocation failed.

Table 1445:Return values (Continued)

Name	Description
SSP_ERR_INVALID_ARGUMENT	Illegal parameter was found.
SSP_ERR_CTSU_OFFSET_ADJUSTMENT_FAILED	Initial auto tuning failed because either the filter depth is too large, or the tuning is not valid for either the board or the current conditions.

NOTE: Data processing functions should be used by advanced users only. Beginners should provide a dummy argument (except NULL). Members inside should be kept NULL.

NOTE: If CFG_AUTO_TUNE is enabled, then the user must ensure that global interrupt operation is enabled before Open gets invoked.

9.10.4.3 Function steps

- Perform quick checks
- Set the valid channel mask based on variant data.
- Cannot reconfigure CTSU. You should unload active configuration by calling Close before proceeding.
- Power ON the CTSU
- Blind copy the configuration settings to initialize CTSU
- Discover excluded channels
- The number of active channels must be greater than the number of excluded channels
- Variables to be used in the loop below
- Initialize buffers used
- Populate primary buffers
- Select operation functions
- Enable interrupt operation
- Perform optional actions
- Indicate that R_Touch is ready
- Save which CTSU configuration has been opened
- Save the CTSU options used during open
- Save the CTSU options to use when updating parameters
- Save the notification function argument to be passed

9.10.5 R_CTSU_Start_Scan

```
ssp_err_t R_CTSU_Start_Scan ( ctsu_ctrl_t  * p_api_ctrl )
```

9.10.5.1 Brief description

Starts off a CTSU hardware scan. The rate at which this function is called is essentially the scan rate.

9.10.5.2 Detailed description

Table 1446:Return values

Name	Description
SSP_SUCCESS	No errors. CTSU scan successfully initialized.
SSP_ERR_IN_USE	A scan is ongoing

9.10.5.3 Function steps

- CTSU h/w is ready to start a new scan.
- Start a new scan.
- Hardware is busy

9.10.6 R_CTSU_Update_Parameters

```
ssp_err_t R_CTSU_Update_Parameters ( ctsu_ctrl_t  * p_api_ctrl )
```

9.10.6.1 Brief description

Function that must be called after each scan is complete to process the results of the scanned data.

9.10.6.2 Detailed description

It performs the following tasks:

```
             0. Checks for any errors in CTSU operation.
             1. Runs a filter on the raw values output by the CTSU.
             2. Determines if a channel is being touched or not and updates the
binary and difference between
                 sensor_baseline and the current sensor value.
          3. When the update is complete, the user specified callback is called
from this function with
               the CTSU_EVENT_PARAMETERS_UPDATED event.
```

```
            4. (Optional) Performs auto-tuning if (a single/all) channels are not
being touched.
             5. (Optional) Performs drift compensation if (a single/all channels)
are not being touched.
                     Drift compensation := move sensor baseline after N readings.
```

Table 1447:Return values

Name	Description
SSP_SUCCESS	No errors.
SSP_ERR_INVALID_POINTER	NULL pointer passed as an argument for a required parameter. -OR- Memory allocation failed.
SSP_ERR_INVALID_ARGUMENT	Illegal parameter was found.
SSP_ERR_CTSU_OFFSET_ADJUSTMENT_FAILED	Auto tuning failed because CTSU encountered a ICOMP error or Check the configuration parameters for each channel by locating the point of failure.

9.10.6.3 Function steps

- Check for CTSU error conditions
- Update each channel measured
- Initialize variables for loop below
- Channel is to be excluded. Bypass calculations.
- Check which channels are being touched.
- Initialize variables for loop below
- Channel is to be excluded. Bypass calculations.
- Do the callback to provide a notification
- Perform drift compensation

9.10.7 R_CTSU_Process

```
ssp_err_t R_CTSU_Process ( ctsu_ctrl_t  * p_api_ctrl ,
 ctsu_process_option_t  opts )
```

9.10.7.1 Brief description

Function that the user must call from the main loop. This function is deprecated by the R_CTSU_Update_Parameters function and is not exposed to the user in the interface API.

9.10.7.2 Detailed description

It performs the following tasks:

```
             0. Checks for any errors in CTSU operation.
             1. Runs a filter on the raw values output by the CTSU.
             2. Determines if a channel is being touched or not and updates the
binary and difference between
                  sensor_baseline and the current sensor value.
           3. (Optional) Performs auto-tuning if (a single/all) channels are not
being touched.
            4. (Optional) Performs drift compensation if (a single/all channels)
are not being touched.
                     Drift compensation := move sensor baseline after N readings.
             5. (Optional) Starts off a new scan.
```

Table 1448:Return values

Name	Description
SSP_SUCCESS	No errors.
SSP_ERR_INVALID_POINTER	NULL pointer passed as an argument for a required parameter. -OR- Memory allocation failed.
SSP_ERR_INVALID_ARGUMENT	Illegal parameter was found.
SSP_ERR_CTSU_OFFSET_ADJUSTMENT_FAILED	Auto tuning failed because CTSU encountered a ICOMP error or Check the configuration parameters for each channel by locating the point of failure.

9.10.7.3 Function steps

- Check for CTSU error conditions
- Update each channel measured
- Initialize variables for loop below
- Initialize variables for loop below
- Check which channels are being touched.
- Update buttons
- Update Slider
- Perform drift compensation
- CTSU is ready to run
- Start a new scan.

- Save a copy of the state machine

9.10.8 R_CTSU_Read_Results

```
ssp_err_t R_CTSU_Read_Results ( ctsu_ctrl_t  * p_api_ctrl ,   void * p_dest ,
 ctsu_read_t  opts ,  ctsu_channel_pair_t const * channels ,   const
uint16_t count )
```

9.10.8.1 Brief description

Function to allow the user to read the results after executing a scan.

9.10.8.2 Detailed description

Table 1449:Return values

Name	Description
SSP_SUCCESS	No errors.
CTSU_ERR_INVALID_CMD	Command option provided is invalid.
SSP_ERR_INVALID_POINTER	NULL pointer passed as an argument for a required parameter. -OR- Memory allocation failed.
SSP_ERR_INVALID_ARGUMENT	Illegal parameter was found.
SSP_ERR_CTSU_OFFSET_ADJUSTMENT_FAILED	Auto tuning failed because CTSU encountered a ICOMP error. Check the configuration parameters for each channel by locating the point of failure.

9.10.8.3 Function steps

- Get results for the requested options
- Assemble arr[3] and arr[2] as the high and low byte
- Assemble arr[3] and arr[2] as the high and low byte
- Assemble arr[3] and arr[2] as the high and low byte
- Assemble arr[3] and arr[2] as the high and low byte
- Assemble arr[3] and arr[2] as the high and low byte
- Assemble arr[3] and arr[2] as the high and low byte

9.10.9 R_CTSU_Control

```
ssp_err_t R_CTSU_Control (  ctsu_cmd_t cmd ,   void * p_data )
```

9.10.9.1 Brief description

Allows the user to query manipulate R_Touch layer, and CTSU operation. This function is not exposed for usage in the interface API.

9.10.9.2 Detailed description

Table 1450:Return values

Name	Description
SSP_SUCCESS	No errors.
CTSU_ERR_INVALID_CMD	Command option provided is invalid.
SSP_ERR_IN_USE	CTSU Scan is ongoing.

9.10.9.3 Function steps

- Execute the control request

9.10.10 R_CTSU_Close

```
ssp_err_t R_CTSU_Close ( ctsu_ctrl_t  * p_api_ctrl ,
 ctsu_close_option_t  opts )
```

9.10.10.1 Brief description

Function used to close the CTSU driver, stop clocking of the CTSU peripheral, reset all internal structures and close the DTC Transfer instances. The function can also optionally save the configuration for a rapid wake-up on re-opening.

9.10.10.2 Detailed description

Table 1451:Return values

Name	Description
SSP_SUCCESS	No errors.

9.10.11 Extensions

9.10.11.1 ctsu_instance_ctrl_t

ctsu_instance_ctrl_t

Detailed description

Instance control structure

Variables

- transfer_api_t const * p_api_transfer

  Pointer to lower level Transfer driver function pointers.
- transfer_ctrl_t * p_lowerl_lvl_transfer_read_ctrl

  Pointer to the Transfer Read control.
- transfer_ctrl_t * p_lowerl_lvl_transfer_write_ctrl

  Pointer to the Transfer Write control.
- bool ctsu_opened

  Store initialization state.
- uint8_t ctsu_unit

  CTSU Unit in use.
- ctsu_hw_cfg_t * p_ctsu_hw_cfg

  Pointer to a CTSU configuration.
- ctsu_process_option_t ctsu_open_option

  Software options to use when performing Open and Process.
- ctsu_process_option_t ctsu_update_option

  Software options to use when performing parameter Update process.
- ctsu_close_option_t ctsu_close_option

  Software options to use when closing touch operation.
- ctsu_action_t ctsu_process_state

  Variable to observe CTSU processing state machine operation.
- void(* p_callback)(*p_args)

  Callback to the function to use when updating dependent parameters is complete.
- void * p_context

  Pointer to data that should be passed to update_complete notification.
- R_CTSU_Type * p_reg

  Pointer to base register address.

9.11 DAC

Driver for the 12-Bit D/C Converter (DAC12).

9.11.1 Summary

This module implements the following interface: DAC Interface.

Name of module used by error logger macro

9.11.2 Functions

- R_DAC_Open
- R_DAC_Close
- R_DAC_Write
- R_DAC_Start
- R_DAC_Stop
- R_DAC_VersionGet
- R_DAC_InfoGet

9.11.3 Defines

- #define DAC_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define DAC_CODE_VERSION_MINOR

  Initial value: `(4U)`

9.11.4 R_DAC_Open

```
ssp_err_t R_DAC_Open ( dac_ctrl_t  * p_api_ctrl ,  dac_cfg_t  const
*const p_cfg )
```

9.11.4.1 Brief description

Perform required initialization described in hardware manual. Implements open. Configures a single DAC channel, starts the channel, and provides a handle for use with the DAC API Write and Close functions. Must be called once prior to calling any other DAC API functions. After a channel is opened, Open should not be called again for the same channel without calling Close first.

9.11.4.2 Detailed description

Table 1452:Return values

Name	Description
SSP_SUCCESS	The channel was successfully opened.
SSP_ERR_ASSERTION	One or both of the following parameters may be NULL: p_api_ctrl or p_cfg Channel ID requested in p_cfg may not available on the device selected in r_bsp_config.h data_format value in p_cfg is out of range. ad_da_synchronized value in p_cfg is out of range.
SSP_ERR_IN_USE	DAC resource is locked.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

9.11.4.3 Function steps

- Validate the input parameter.
- Make sure the peripheral exists.
- Lock the DAC Hardware Resource.
- Power on the DAC device.
- Stop the channel.
- Configure data format: left or right justified.
- Configure D/A-A/D Synchronous Start Control Register(DAADSCR).
- Set output amplifier configuration for the channel.
- Set the reference voltage.
- Initialize the channel state information.
- All done. Return.

9.11.5 R_DAC_Close

```
ssp_err_t R_DAC_Close ( dac_ctrl_t  * p_api_ctrl )
```

9.11.5.1 Brief description

Stop the D/A conversion, stop output, and close the DAC channel.

9.11.5.2 Detailed description

Table 1453:Return values

Name	Description
SSP_SUCCESS	The channel is successfully closed.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

9.11.5.3 Function steps

- Validate that the channel is opened.
- Stop the channel
- Update the channel state information.
- Unlock the DAC Hardware Resource
- Power down the DAC device.
- Unlock the DAC Hardware Resource

9.11.6 R_DAC_Write

```
ssp_err_t R_DAC_Write ( dac_ctrl_t  * p_api_ctrl ,  dac_size_t  value )
```

9.11.6.1 Brief description

Write data to the D/A converter and enable the output if it has not been enabled.

9.11.6.2 Detailed description

Table 1454:Return values

Name	Description
SSP_SUCCESS	Data is successfully written to the D/A Converter.

Table 1454:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

NOTE: Write function automatically starts the D/A conversion after data is successfully written to the channel.

9.11.6.3 Function steps

- Validate that the channel is opened.
- Write the value to D/A converter.
- Start the converter if it has been idle.
- Start the channel

9.11.7 R_DAC_Start

```
ssp_err_t R_DAC_Start ( dac_ctrl_t  * p_api_ctrl )
```

9.11.7.1 Brief description

Start the D/A conversion output if it has not been started.

9.11.7.2 Detailed description

Table 1455:Return values

Name	Description
SSP_SUCCESS	The channel is started successfully.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

9.11.7.3 Function steps

- Validate that the channel is opened.

- Enable the output.
- Update the internal state.

9.11.8 R_DAC_Stop

```
ssp_err_t R_DAC_Stop ( dac_ctrl_t  * p_api_ctrl )
```

9.11.8.1 Brief description

Stop the D/A conversion and disable the output signal.

9.11.8.2 Detailed description

Table 1456:Return values

Name	Description
SSP_SUCCESS	The control is successfully stopped.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

9.11.8.3 Function steps

- Validate that the channel is opened.
- Disable the output.
- Mark the internal state.

9.11.9 R_DAC_VersionGet

```
ssp_err_t R_DAC_VersionGet ( ssp_version_t  * p_version )
```

9.11.9.1 Brief description

Get version and store it in provided pointer p_version.

9.11.9.2 Detailed description

Table 1457:Return values

Name	Description
SSP_SUCCESS	Successfully retrieved version information.
SSP_ERR_ASSERTION	p_version is NULL.

9.11.10 R_DAC_InfoGet

```
ssp_err_t R_DAC_InfoGet ( dac_info_t  *const p_info )
```

9.11.10.1 Brief description

Get information about DAC Resolution and store it in provided pointer p_info.

9.11.10.2 Detailed description

Table 1458:Return values

Name	Description
SSP_SUCCESS	Value of DAC resolution written to caller's structure successfully.
SSP_ERR_ASSERTION	The p_info parameter was null.

9.11.11 Extensions

9.11.11.1 dac_instance_ctrl_t

dac_instance_ctrl_t

Detailed description

DAC instance control block. DO NOT INITIALIZE.

Variables

- void * p_reg

  Pointer to DAC base register.

- uint8_t channel

  ID associated with this DAC channel.
- uint8_t channel_started

  DAC operation on channel started.
- uint8_t channel_opened

  DAC channel open.

9.12 DAC

Driver for the 8-Bit D/C Converter (DAC8).

9.12.1 Summary

This module implements the following interface: DAC Interface.

Name of module used by error logger macro

9.12.2 Functions

- R_DAC8_Open
- R_DAC8_Close
- R_DAC8_Write
- R_DAC8_Start
- R_DAC8_Stop
- R_DAC8_VersionGet
- R_DAC8_InfoGet

9.12.3 Defines

- #define DAC8_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define DAC8_CODE_VERSION_MINOR

  Initial value: `(3U)`

9.12.4 R_DAC8_Open

```
ssp_err_t R_DAC8_Open ( dac_ctrl_t  * p_api_ctrl ,  dac_cfg_t  const
*const p_cfg )
```

9.12.4.1 Brief description

Perform required initialization described in hardware manual. Implements open. Configures a single DAC channel, starts the channel, and provides a handle for use with the DAC API Write and Close functions. Must be called once prior to calling any other DAC API functions. After a channel is opened, Open should not be called again for the same channel without calling Close first.

9.12.4.2 Detailed description

Table 1459:Return values

Name	Description
SSP_SUCCESS	The channel was successfully opened.
SSP_ERR_ASSERTION	One or both of the following parameters may be NULL: p_api_ctrl or p_cfg
SSP_ERR_IN_USE	DAC8 resource is locked.
SSP_ERR_IP_CHANNEL_NOT_PRESENT	An invalid channel was requested.
SSP_ERR_UNSUPPORTED	Output amplifier is not supported. Real time mode is not supported. Charge pump is not supported. A/D - D/A synchronization is not supported.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

9.12.4.3 Function steps

- Validate the input parameter.
- Make sure the peripheral exists.
- Lock the DAC Hardware Resource.
- Power on the DAC device.
- Stop the channel.
- Set defaults.
- Configure the charge pump.
- Configure the DAC mode.

- Configure DA/AD mode.
- Initialize the channel state information.
- All done. Return.

9.12.5 R_DAC8_Close

```
ssp_err_t R_DAC8_Close ( dac_ctrl_t  * p_api_ctrl )
```

9.12.5.1 Brief description

Stop the D/A conversion, stop output, and close the DAC channel.

9.12.5.2 Detailed description

Table 1460:Return values

Name	Description
SSP_SUCCESS	The channel is successfully closed.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

9.12.5.3 Function steps

- Stop the channel
- Update the channel state information.
- Unlock the DAC Hardware Resource
- Unlock the DAC Hardware Resource

9.12.6 R_DAC8_Write

```
ssp_err_t R_DAC8_Write ( dac_ctrl_t  * p_api_ctrl ,  dac_size_t  value )
```

9.12.6.1 Brief description

Write data to the D/A converter and enable the output if it has not been enabled.

9.12.6.2 Detailed description

Table 1461:Return values

Name	Description
SSP_SUCCESS	Data is successfully written to the D/A Converter.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

NOTE: Write function automatically starts the D/A conversion after data is successfully written to the channel.

9.12.6.3 Function steps

- Handle data format.
- Convert to 8 bits.
- Write the value to D/A converter.
- Start the converter if it has been idle.
- Start the channel

9.12.7 R_DAC8_Start

```
ssp_err_t R_DAC8_Start ( dac_ctrl_t  * p_api_ctrl )
```

9.12.7.1 Brief description

Start the D/A conversion output.

9.12.7.2 Detailed description

Table 1462:Return values

Name	Description
SSP_SUCCESS	The channel is started successfully.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.

Table 1462:Return values (Continued)

Name	Description
SSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

9.12.7.3 Function steps

- Enable the output.
- Update the internal state.

9.12.8 R_DAC8_Stop

```
ssp_err_t R_DAC8_Stop ( dac_ctrl_t  * p_api_ctrl )
```

9.12.8.1 Brief description

Stop the D/A conversion and disable the output signal.

9.12.8.2 Detailed description

Table 1463:Return values

Name	Description
SSP_SUCCESS	The control is successfully stopped.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

9.12.8.3 Function steps

- Disable the output.
- Mark the internal state.

9.12.9 R_DAC8_VersionGet

```
ssp_err_t R_DAC8_VersionGet ( ssp_version_t  * p_version )
```

9.12.9.1 Brief description

Get version and store it in provided pointer p_version.

9.12.9.2 Detailed description

Table 1464:Return values

Name	Description
SSP_SUCCESS	Successfully retrieved version information.
SSP_ERR_ASSERTION	p_version is NULL.

9.12.10 R_DAC8_InfoGet

```
ssp_err_t R_DAC8_InfoGet ( dac_info_t  *const p_info )
```

9.12.10.1 Brief description

Get information about DAC Resolution and store it in provided pointer p_info.

9.12.10.2 Detailed description

Table 1465:Return values

Name	Description
SSP_SUCCESS	Value of DAC resolution written to caller's structure successfully.
SSP_ERR_ASSERTION	The p_info parameter was null.

9.12.11 API Data

9.12.11.1 dac8_mode_t

`dac8_mode_t`

Detailed description

DAC8 mode

Enumerated values

Name	Description
DAC8_MODE_NORMAL	DAC Normal mode.
DAC8_MODE_REAL_TIME	DAC Real-time (event link) mode.

9.12.12 Extensions

9.12.12.1 dac8_instance_ctrl_t

dac8_instance_ctrl_t

Detailed description

DAC8 instance control block. DO NOT INITIALIZE.

Variables

- void * p_reg

  Pointer to DAC base register.
- uint8_t channel

  ID associated with this DAC channel.
- uint8_t channel_started

  DAC operation on channel started.
- uint32_t channel_opened

  DAC channel open.
- dac_data_format_t data_format

  DAC data format.

9.12.12.2 dac8_extended_cfg_t

dac8_extended_cfg_t

Detailed description

DAC8 extended configuration

Variables

- bool enable_charge_pump

  Enable DAC charge pump.
- dac8_mode_t dac_mode

  DAC mode.

9.13 DMAC

DMA Controller (DMAC).

9.13.1 Summary

Extends the Transfer Interface.

The Direct Memory Access (DMA) Controller allows data transfers to occur in place of or in addition to any interrupt. It also supports data transfers using software start.

NOTE: The transfer length is limited to 1024 (10 bits) in TRANSFER_MODE_BLOCK and TRANSFER_MODE_REPEAT.

NOTE: This driver supports only TRANSFER_IRQ_END from transfer_irq_t.

9.13.2 Functions

- R_DMAC_Open
- R_DMAC_Reset
- R_DMAC_Start
- R_DMAC_Stop
- R_DMAC_Enable
- R_DMAC_Disable
- R_DMAC_InfoGet
- R_DMAC_Close
- R_DMAC_VersionGet
- R_DMAC_BlockReset

9.13.3 Defines

- #define DMAC_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define DMAC_CODE_VERSION_MINOR

  Initial value: `(4U)`
- #define DMAC_REPEAT_BLOCK_MAX_LENGTH

  Initial value: `(0x400)`

  Length limited to 1024 transfers for repeat and block mode

- #define DMAC_NORMAL_MAX_LENGTH

  Initial value: (0xFFFF)

  Length limited to 65535 transfers for normal mode

9.13.4 R_DMAC_Open

```
ssp_err_t R_DMAC_Open ( transfer_ctrl_t  *const p_api_ctrl ,  transfer_cfg_t
const *const p_cfg )
```

9.13.4.1 Brief description

Initialize transfer and enables transfer in ICU. Implements open.

9.13.4.2 Detailed description

Table 1466:Return values

Name	Description
SSP_SUCCESS	Successful open. Transfer is configured and will start when trigger occurs.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_ENABLED	Auto-enable was requested, but enable failed.
SSP_ERR_IRQ_BSP_DISABLED	The IRQ associated with the activation source is not enabled in the BSP.
SSP_ERR_IN_USE	The BSP hardware lock for the DMAC is not available.

See Common Error Codes for other possible return codes.This function calls

- productFeatureGet
- eventInfoGet

9.13.4.3 Function steps

- Acquire BSP hardware lock for channel used.
- Configure the DMAC according to the flowchart "Activating the DMAC" in chapter 16.3.7 of hardware manual NoSecurity_r01uh0488ej0040_sc32.pdf.
- Configure DMA transfer and sources
-

  NOTE: Transfer escape interrupts not supported.

Update internal variables.

- Mark driver as open by initializing "DMAC" in its ASCII equivalent.
- If auto_enable is true, enable transfer and ELC events if software start is used.

9.13.5 R_DMAC_Reset

```
ssp_err_t R_DMAC_Reset ( transfer_ctrl_t  *const p_api_ctrl ,   void const
*volatile p_src ,   void *volatile p_dest ,   uint16_t const num_transfers )
```

9.13.5.1 Brief description

Reset transfer source, destination, and number of transfers.

9.13.5.2 Detailed description

Table 1467:Return values

Name	Description
SSP_SUCCESS	Transfer reset successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_ENABLED	DMAC is not enabled. A valid source and destination must be provided in either open() or reset().
SSP_ERR_IN_USE	Transfer is in progress. Wait for transfer to complete.

9.13.5.3 Function steps

- Enables transfers on this activation source.

9.13.6 R_DMAC_Start

```
ssp_err_t R_DMAC_Start ( transfer_ctrl_t  *const p_api_ctrl ,
 transfer_start_mode_t  mode )
```

9.13.6.1 Brief description

Start transfer. Implements start.

9.13.6.2 Detailed description

Table 1468:Return values

Name	Description
SSP_SUCCESS	Transfer started written successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.
SSP_ERR_UNSUPPORTED	Handle was not configured for software activation.

9.13.6.3 Function steps

- Set autoclear bit and software start bit.

9.13.7 R_DMAC_Stop

```
ssp_err_t R_DMAC_Stop ( transfer_ctrl_t  *const p_api_ctrl )
```

9.13.7.1 Brief description

Stop transfer. Implements stop.

9.13.7.2 Detailed description

Table 1469:Return values

Name	Description
SSP_SUCCESS	Transfer stopped written successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

9.13.7.3 Function steps

- Reset auto clear bit and clear software transfer request.

9.13.8 R_DMAC_Enable

```
ssp_err_t R_DMAC_Enable ( transfer_ctrl_t  *const p_api_ctrl )
```

9.13.8.1 Brief description

Enable transfer. Implements enable.

9.13.8.2 Detailed description

Table 1470:Return values

Name	Description
SSP_SUCCESS	Counter value written successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

9.13.8.3 Function steps

- Enable transfer.

9.13.9 R_DMAC_Disable

```
ssp_err_t R_DMAC_Disable ( transfer_ctrl_t  *const p_api_ctrl )
```

9.13.9.1 Brief description

Disable transfer. Implements disable.

9.13.9.2 Detailed description

Table 1471:Return values

Name	Description
SSP_SUCCESS	Counter value written successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.

Table 1471:Return values (Continued)

Name	Description
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

9.13.9.3 Function steps

- Disable transfer.

9.13.10 R_DMAC_InfoGet

```
ssp_err_t R_DMAC_InfoGet ( transfer_ctrl_t  *const p_api_ctrl ,
 transfer_properties_t  *const p_info )
```

9.13.10.1 Brief description

Set driver specific information in provided pointer. Implements infoGet.

9.13.10.2 Detailed description

Table 1472:Return values

Name	Description
SSP_SUCCESS	Counter value written successfully.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.
SSP_ERR_ASSERTION	An input parameter is invalid.

9.13.10.3 Function steps

- If a transfer is active, store it in p_in_progress.
- Store maximum transfer length.
- Store remaining transfer length.

9.13.11 R_DMAC_Close

```
ssp_err_t R_DMAC_Close ( transfer_ctrl_t  *const p_api_ctrl )
```

9.13.11.1 Brief description

Disable transfer and clean up internal data. Implements close.

9.13.11.2 Detailed description

Table 1473:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.
SSP_ERR_IN_USE	Transfer is in progress. Wait for transfer to complete.

9.13.11.3 Function steps

- Disable DMAC transfers, disable DMAC interrupts, and remove DMAC trigger from ICU register.
- Clear ID so control block can be reused.
- Release BSP hardware lock on this channel

9.13.12 R_DMAC_VersionGet

```
ssp_err_t R_DMAC_VersionGet ( ssp_version_t  *const p_version )
```

9.13.12.1 Brief description

Set driver version based on compile time macros.

9.13.12.2 Detailed description

Table 1474:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	An input parameter is invalid.

9.13.13 R_DMAC_BlockReset

```
ssp_err_t R_DMAC_BlockReset ( transfer_ctrl_t  *const p_api_ctrl ,   void const
*volatile p_src ,   void *volatile p_dest ,   uint16_t const length ,
 transfer_size_t  size ,   uint16_t const num_transfers )
```

9.13.13.1 Brief description

Reset transfer source, destination, length and number of transfers for block transfer.

9.13.13.2 Detailed description

Table 1475:Return values

Name	Description
SSP_SUCCESS	Transfer reset successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_IN_USE	Transfer is in progress. Wait for transfer to complete.
SSP_ERR_NOT_ENABLED	DMAC is not enabled. A valid source and destination must be provided in either open() or blockReset().

9.13.13.3 Function steps

- Enables transfers on this activation source.

9.13.14 Extensions

9.13.14.1 dmac_instance_ctrl_t

dmac_instance_ctrl_t

Detailed description

Control block used by driver. DO NOT INITIALIZE - this structure will be initialized in open.

Variables

- uint32_t id

  Driver ID.

- elc_event_t trigger

  Transfer activation event. Matches event returned by transfer_api_t::infoGet.

- IRQn_Type irq

  Transfer activation IRQ.
- uint8_t channel

  Channel number.
- void(* p_callback)(*cb_data)

  Callback for transfer end interrupt.
- void const * p_context

  Placeholder for user data. Passed to the user p_callback in transfer_callback_args_t.
- void * p_reg

  Pointer to base register.

9.13.14.2 transfer_on_dmac_cfg_t

transfer_on_dmac_cfg_t

Detailed description

DMAC transfer configuration extension. This extension is required.

Variables

- uint8_t channel

  Channel number, does not apply to all HAL drivers.

9.14 DOC

Driver for the Data Operation Circuit (DOC).

9.14.1 Summary

This module implements the DOC Interface using the Data Operation Circuit (DOC).

9.14.2 Functions

- R_DOC_Open
- R_DOC_Close
- R_DOC_StatusGet
- R_DOC_StatusClear
- R_DOC_Write

- R_DOC_InputRegisterWrite
- R_DOC_VersionGet

9.14.3 Defines

- #define DOC_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define DOC_CODE_VERSION_MINOR

  Initial value: `(4U)`

9.14.4 R_DOC_Open

```
ssp_err_t R_DOC_Open ( doc_ctrl_t  *const p_api_ctrl ,  doc_cfg_t  const
*const p_cfg )
```

9.14.4.1 Brief description

Configure the Data Operation Circuit (DOC) in comparison, addition or subtraction mode. Implements open.

9.14.4.2 Detailed description

If callback is not NULL in the configuration structure the DOC IRQ will be enabled.

Table 1476:Return values

Name	Description
SSP_SUCCESS	DOC successfully configured.
SSP_ERR_IN_USE	Module already open.
SSP_ERR_ASSERTION	One or more pointers point to NULL.
SSP_ERR_INVALID_ARGUMENT	ISR is not enabled. Enable the ISR in bsp_irq_cfg.h.
SSP_ERR_HW_LOCKED	DOC resource is locked.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet
- eventInfoGet

NOTE: This function is reentrant.

9.14.4.3 Function steps

- Make sure the DOC exists on this part.
- Lock the DOC Hardware Resource
- Configure the DOC via DOCR register.
- If callback parameter is not NULL configure the IRQ
- Mark driver as open by initializing it to "DOCO" in its ASCII equivalent.

9.14.5 R_DOC_Close

```
ssp_err_t R_DOC_Close ( doc_ctrl_t  *const p_api_ctrl )
```

9.14.5.1 Brief description

Close the module driver. Enable module stop mode. Implements close.

9.14.5.2 Detailed description

To close the DOC it must have been opened via the open API. When opened a control structure of type doc_ctrl_t is passed to the open API. The same control structure must be passed to the close API.

Table 1477:Return values

Name	Description
SSP_SUCCESS	Module successfully closed.
SSP_ERR_NOT_OPEN	Driver not open.
SSP_ERR_ASSERTION	Pointer pointing to NULL.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- eventInfoGet

NOTE: This function is reentrant.

NOTE: This function will disable the DOC interrupt in the NVIC.

9.14.5.3 Function steps

- Disable the IRQ in the NVIC in case it has been left enabled.
- Clear DOPCF in DOCR
- Mark driver as closed.
- Unlock the DOC Hardware Resource

9.14.6 R_DOC_StatusGet

```
ssp_err_t R_DOC_StatusGet ( doc_ctrl_t  *const p_api_ctrl ,  doc_status_t
* p_status )
```

9.14.6.1 Brief description

Return the comparison/addition/subtraction status. Implements statusGet.

9.14.6.2 Detailed description

The status is read from the Data Operation Circuit Flag (DOPCF).

Table 1478:Return values

Name	Description
SSP_SUCCESS	Status successfully read.
SSP_ERR_NOT_OPEN	Driver not open.
SSP_ERR_ASSERTION	One or more pointers point to NULL.

NOTE: This function is reentrant.

9.14.7 R_DOC_StatusClear

```
ssp_err_t R_DOC_StatusClear ( doc_ctrl_t  *const p_api_ctrl )
```

9.14.7.1 Brief description

Clear the DOPCF status flag at the hardware layer. This flag indicates the result of a DOC operation. Implements statusClear.

9.14.7.2 Detailed description

Table 1479:Return values

Name	Description
SSP_SUCCESS	Interrupt successfully cleared.
SSP_ERR_NOT_OPEN	Driver not open.
SSP_ERR_ASSERTION	Pointer pointing to NULL.

NOTE: This function is reentrant.

9.14.8 R_DOC_Write

```
ssp_err_t R_DOC_Write ( doc_ctrl_t  *const p_api_ctrl ,  doc_data_t
*const p_data )
```

9.14.8.1 Brief description

Write to the DODIR and DODSR registers. Implements write,.

9.14.8.2 Detailed description

In comparison mode the 16-bit reference data is written to the DODSR register and the data for the comparison written to the DODIR. In addition mode the initial data is written to the DODSR and the data to be added to the DODIR. In subtraction mode the initial data is written to the DODSR and the data to be subtracted to the DODIR.

When changing both the DODSR and DODIR the DODSR should be updated first.

Table 1480:Return values

Name	Description
SSP_SUCCESS	Values successfully written to the registers.
SSP_ERR_NOT_OPEN	Driver not open.

Table 1480:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	One or more pointers point to NULL.

NOTE: This function is reentrant.

9.14.9 R_DOC_InputRegisterWrite

```
ssp_err_t R_DOC_InputRegisterWrite ( doc_ctrl_t  *const p_api_ctrl ,
 doc_size_t  data )
```

9.14.9.1 Brief description

Write to the DODIR register. Implements inputRegisterWrite,.

9.14.9.2 Detailed description

Writes to the DODIR register only.

Table 1481:Return values

Name	Description
SSP_SUCCESS	Value successfully written to the DODIR register.
SSP_ERR_NOT_OPEN	Driver not open.
SSP_ERR_ASSERTION	One or more pointers point to NULL.

NOTE: This function is reentrant.

9.14.10 R_DOC_VersionGet

```
ssp_err_t R_DOC_VersionGet ( ssp_version_t  *const p_version )
```

9.14.10.1 Brief description

Return DOC HAL driver version. Implements versionGet.

9.14.10.2 Detailed description

Table 1482:Return values

Name	Description
SSP_SUCCESS	Version information successfully read.
SSP_ERR_ASSERTION	Pointer pointing to NULL.

NOTE: This function is reentrant.

9.14.11 Extensions

9.14.11.1 doc_instance_ctrl_t

doc_instance_ctrl_t

Detailed description

DOC instance control block. Do not initialize. Initialization occurs when the open function is called.

Variables

- uint32_t open

  Used by driver to check if the control structure is valid.
- void(* p_callback)(*p_args)

  Callback provided when a DOC ISR occurs. NULL indicates no CPU interrupt.
- doc_event_t event

  The event DOC is configured for. Passed in ISR callback.
- void const * p_context

  Placeholder for user data. Passed to the user callback in doc_callback_args_t.
- void * p_reg

  Base register.

9.15 DTC

Driver for the Data Transfer Controller (DTC).

9.15.1 Summary

Extends Transfer Interface.

The Data Transfer Controller allows data transfers to occur in place of or in addition to any interrupt. It does not support data transfers using software start.

NOTE: The transfer length is limited to 256 (8 bits) in TRANSFER_MODE_BLOCK and TRANSFER_MODE_REPEAT.

9.15.2 Functions

- R_DTC_Open
- R_DTC_Reset
- R_DTC_Start
- R_DTC_Stop
- R_DTC_Enable
- R_DTC_Disable
- R_DTC_InfoGet
- R_DTC_Close
- R_DTC_VersionGet
- R_DTC_BlockReset

9.15.3 Defines

- #define DTC_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define DTC_CODE_VERSION_MINOR

  Initial value: `(8U)`
- #define DTC_REPEAT_BLOCK_MAX_LENGTH

  Initial value: `(0x100)`

  Length limited to 256 transfers for repeat and block mode

- #define DTC_NORMAL_MAX_LENGTH

  Initial value: (0x10000)

  Length limited to 65536 transfers for normal mode

9.15.4 R_DTC_Open

```
ssp_err_t R_DTC_Open ( transfer_ctrl_t  *const p_api_ctrl ,  transfer_cfg_t
const *const p_cfg )
```

9.15.4.1 Brief description

Set transfer data in the vector table and enable transfer in ICU. Implements open.

9.15.4.2 Detailed description

Table 1483:Return values

Name	Description
SSP_SUCCESS	Successful open. Transfer is configured and will start when trigger occurs.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_IN_USE	The BSP hardware lock for the DTC is not available, or the index for this IRQ in the DTC vector table is already configured.
SSP_ERR_HW_LOCKED	DTC hardware resource is locked.
SSP_ERR_IRQ_BSP_DISABLED	The IRQ associated with the activation source is not enabled in the BSP.
SSP_ERR_NOT_ENABLED	Auto-enable was requested, but enable failed due to an invalid input parameter.

9.15.4.3 Function steps

- Make sure the activation source is mapped in the ICU.
- Make sure the activation source is not already being used by the DTC.
- Configure the DTC transfer. See the hardware manual for details.
- Update internal variables.
- Mark driver as open by initializing it to "DTC" in its ASCII equivalent.

- If auto_enable is true, enable transfer and ELC events if software start is used.
- Enable read skip after all settings are complete.

9.15.5 R_DTC_Reset

```
ssp_err_t R_DTC_Reset ( transfer_ctrl_t  *const p_api_ctrl ,   void const
*volatile p_src ,   void *volatile p_dest ,   uint16_t const num_transfers )
```

9.15.5.1 Brief description

Reset transfer source, destination, and number of transfers. Implements reset.

9.15.5.2 Detailed description

Table 1484:Return values

Name	Description
SSP_SUCCESS	Transfer reset successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.
SSP_ERR_NOT_ENABLED	Transfer length must not be 0 for repeat and block mode, or enable failed due to an invalid input parameter: • Transfer source must not be NULL. • Transfer destination must not be NULL.

9.15.5.3 Function steps

- Disable transfers on this activation source.
- Disable read skip prior to modifying settings. It will be enabled later.
- Reset transfer based on input parameters.
- Enables transfers on this activation source.
- Enable read skip after all settings are complete.

9.15.6 R_DTC_Start

```
ssp_err_t R_DTC_Start ( transfer_ctrl_t  *const p_api_ctrl ,
 transfer_start_mode_t  mode )
```

9.15.6.1 Brief description

Start transfer. Implements start.

9.15.6.2 Detailed description

Table 1485:Return values

Name	Description
SSP_SUCCESS	Transfer started successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.
SSP_ERR_UNSUPPORTED	One of the following is invalid: • Handle was not configured for software activation. • Mode not set to TRANSFER_START_MODE_SINGLE. • DTC_SOFTWARE_START_ENABLE set to 0 (disabled) in ssp_cfg/driver/r_dtc_cfg.h.

9.15.7 R_DTC_Stop

```
ssp_err_t R_DTC_Stop ( transfer_ctrl_t  *const p_ctrl )
```

9.15.7.1 Brief description

Placeholder for unsupported stop function. Implements stop.

9.15.7.2 Detailed description

Table 1486:Return values

Name	Description
SSP_ERR_UNSUPPORTED	DTC software start is not supported.

9.15.8 R_DTC_Enable

```
ssp_err_t R_DTC_Enable ( transfer_ctrl_t  *const p_api_ctrl )
```

9.15.8.1 Brief description

Enable transfer and ELC events if they are used for software start. Implements enable.

9.15.8.2 Detailed description

Table 1487:Return values

Name	Description
SSP_SUCCESS	Counter value written successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.
SSP_ERR_IRQ_BSP_DISABLED	The IRQ associated with the p_ctrl is not enabled in the BSP.

9.15.8.3 Function steps

- Enable transfer.

9.15.9 R_DTC_Disable

```
ssp_err_t R_DTC_Disable ( transfer_ctrl_t  *const p_api_ctrl )
```

9.15.9.1 Brief description

Disable transfer. Implements disable.

9.15.9.2 Detailed description

Table 1488:Return values

Name	Description
SSP_SUCCESS	Counter value written successfully.

Table 1488:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.

9.15.9.3 Function steps

- Disable transfer.

9.15.10 R_DTC_InfoGet

```
ssp_err_t R_DTC_InfoGet ( transfer_ctrl_t  *const p_api_ctrl ,
 transfer_properties_t  *const p_info )
```

9.15.10.1 Brief description

Set driver specific information. Implements infoGet.

9.15.10.2 Detailed description

Table 1489:Return values

Name	Description
SSP_SUCCESS	Counter value written successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.

9.15.10.3 Function steps

- If a transfer is active, store it in p_in_progress.
- Transfer information for the activation source is taken from DTC vector table.
- Mask out the high byte in case of repeat transfer.
- Store maximum transfer length.

9.15.11 R_DTC_Close

```
ssp_err_t R_DTC_Close ( transfer_ctrl_t  *const p_api_ctrl )
```

9.15.11.1 Brief description

Disables transfer in the ICU, then clears transfer data from the DTC vector table. Implements close.

9.15.11.2 Detailed description

Table 1490:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.
SSP_ERR_IRQ_BSP_DISABLED	The IRQ associated with the p_ctrl is not enabled in the BSP.

9.15.11.3 Function steps

- Clear DTC enable bit in ICU.
- Clear pointer in vector table.

9.15.12 R_DTC_VersionGet

```
ssp_err_t R_DTC_VersionGet ( ssp_version_t  *const p_version )
```

9.15.12.1 Brief description

Set driver version based on compile time macros. Implements versionGet.

9.15.12.2 Detailed description

Table 1491:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	An input parameter is invalid.

9.15.13 R_DTC_BlockReset

```
ssp_err_t R_DTC_BlockReset ( transfer_ctrl_t  *const p_api_ctrl ,   void const
*volatile p_src ,   void *volatile p_dest ,   uint16_t const length ,
 transfer_size_t  size ,   uint16_t const num_transfers )
```

9.15.13.1 Brief description

BlockReset transfer source, destination, length and number of transfers. Implements blockReset.

9.15.13.2 Detailed description

Table 1492:Return values

Name	Description
SSP_SUCCESS	Transfer reset successfully.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.
SSP_ERR_UNSUPPORTED	If mode is other than Block Transfer Mode.
SSP_ERR_NOT_ENABLED	Enable failed due to an invalid input parameter: • Transfer source must not be NULL. • Transfer destination must not be NULL.

9.15.13.3 Function steps

- Disable transfers on this activation source.
- Disable read skip prior to modifying settings. It will be enabled later.

- Reset transfer based on input parameters.
- Enables transfers on this activation source.
- Enable read skip after all settings are complete.

9.15.14 Extensions

9.15.14.1 dtc_instance_ctrl_t

dtc_instance_ctrl_t

Detailed description

Control block used by driver. DO NOT INITIALIZE - this structure will be initialized in open.

Variables

- uint32_t id

  Driver ID.
- elc_event_t trigger

  Transfer activation event. Matches event returned by transfer_api_t::infoGet.
- IRQn_Type irq

  Transfer activation IRQ, does not apply to all HAL drivers.
- void(* p_callback)(*cb_data)

  Callback for transfer end interrupt used for ELC software trigger.
- void const * p_context

  Placeholder for user data. Passed to the user p_callback in transfer_callback_args_t.

9.15.14.2 dtc_reg_t

dtc_reg_t

Detailed description

DTC Registers. Same as transfer_info_t, but uses register names. Provided as service to typecast transfer_info_t.

Variables

- uint32_t __pad0__
- uint8_t MRB

  Mode Register B
- uint8_t __pad0__
- uint8_t DM

  Transfer Destination Address mode.

- uint8_t DTS

  DTC Transfer Mode Select.
- uint8_t DISEL

  DTC Interrupt Select.
- uint8_t CHNS

  DTC Chain Transfer Select.
- uint8_t CHNE

  DTC CHain Transfer Enable.
- struct{} MRB_b

  MRB bits */

  See source code for definition of this member.
- uint8_t MRA

  Mode Register A
- uint8_t SM

  Transfer Source Address mode.
- uint8_t SZ

  DTC Data Transfer Size.
- uint8_t MD

  DTC Transfer Mode Select.
- struct{} MRA_b

  MRA bits */

  See source code for definition of this member.
- struct{} struct{}

  Mode registers */

  See source code for definition of this member.
- void *volatile SAR

  Source address register.
- void *volatile DAR

  Destination address register
- uint16_t CRB

  Transfer count register B
- uint16_t CRA

  Transfer count register A

- uint8_t CRAL

  Transfer counter A lower register.
- uint8_t CRAH

  Transfer counter B upper register.
- struct{} CRA_b

  bits */

  See source code for definition of this member.
- struct{} struct{}

  Transfer count registers */

  See source code for definition of this member.

9.16 ELC

Driver for the Event Link Controller (ELC).

This module supports the Event Link Controller (ELC). It implements the following interface: ELC Interface

9.16.1 Functions

- R_ELC_Init
- R_ELC_SoftwareEventGenerate
- R_ELC_LinkSet
- R_ELC_LinkBreak
- R_ELC_Enable
- R_ELC_Disable
- R_ELC_VersionGet

9.16.2 Defines

- #define ELC_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define ELC_CODE_VERSION_MINOR

  Initial value: `(3U)`

9.16.3 R_ELC_Init

```
ssp_err_t R_ELC_Init ( elc_cfg_t  const *const p_cfg )
```

9.16.3.1 Brief description

Initialize all the links in the Event Link Controller.

9.16.3.2 Detailed description

Implements init

The configuration structure passed in to this function includes links for every event source included in the ELC and sets them all at once. To set an individual link use R_ELC_LinkSet

Table 1493:Return values

Name	Description
SSP_SUCCESS	Initialization was successful
SSP_ERR_ASSERTION	p_config was NULL

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet

9.16.3.3 Function steps

- Power on ELC
- Enable the operation of the Event Link Controller

9.16.4 R_ELC_SoftwareEventGenerate

```
ssp_err_t R_ELC_SoftwareEventGenerate ( elc_software_event_t  event_number )
```

9.16.4.1 Brief description

Generate a software event in the Event Link Controller.

9.16.4.2 Detailed description

Implements softwareEventGenerate

Table 1494:Return values

Name	Description
SSP_SUCCESS	Initialization was successful
SSP_ERR_ASSERTION	Invalid event number

9.16.5 R_ELC_LinkSet

```
ssp_err_t R_ELC_LinkSet ( elc_peripheral_t  peripheral ,  elc_event_t  signal )
```

9.16.5.1 Brief description

Create a single event link.

9.16.5.2 Detailed description

Implements linkSet

Table 1495:Return values

Name	Description
SSP_SUCCESS	Initialization was successful

9.16.6 R_ELC_LinkBreak

```
ssp_err_t R_ELC_LinkBreak ( elc_peripheral_t  peripheral )
```

9.16.6.1 Brief description

Break an event link.

9.16.6.2 Detailed description

Implements linkBreak

Table 1496:Return values

Name	Description
SSP_SUCCESS	Event link broken

9.16.7 R_ELC_Enable

```
ssp_err_t R_ELC_Enable ( void )
```

9.16.7.1 Brief description

Enable the operation of the Event Link Controller.

9.16.7.2 Detailed description

Implements enable

Table 1497:Return values

Name	Description
SSP_SUCCESS	ELC enabled.

9.16.8 R_ELC_Disable

```
ssp_err_t R_ELC_Disable ( void )
```

9.16.8.1 Brief description

Disable the operation of the Event Link Controller.

9.16.8.2 Detailed description

Implements disable

Table 1498:Return values

Name	Description
SSP_SUCCESS	ELC disabled.

9.16.9 R_ELC_VersionGet

```
ssp_err_t R_ELC_VersionGet ( ssp_version_t  *const p_version )
```

9.16.9.1 Brief description

Get the driver version based on compile time macros.

9.16.9.2 Detailed description

Implements versionGet

Table 1499:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_version is NULL.

9.17 Low Power Flash

Driver for the Low power Flash Memory (S3A7 and S124).

This module supports the Flash interface for the Low-power FLASH peripheral.

9.17.1 Functions

- R_FLASH_LP_Open
- R_FLASH_LP_Write
- R_FLASH_LP_Read
- R_FLASH_LP_Erase
- R_FLASH_LP_BlankCheck
- R_FLASH_LP_StatusGet
- R_FLASH_LP_AccessWindowSet
- R_FLASH_LP_AccessWindowClear
- R_FLASH_LP_Reset
- R_FLASH_LP_StartUpAreaSelect
- R_FLASH_LP_UpdateFlashClockFreq
- R_FLASH_LP_InfoGet

- R_FLASH_LP_Close
- R_FLASH_LP_VersionGet

9.17.2 Defines

- #define FLASH_LP_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define FLASH_LP_CODE_VERSION_MINOR

  Initial value: `(8U)`
- #define PLACE_IN_RAM_SECTION

  Initial value:

9.17.3 R_FLASH_LP_Open

```
ssp_err_t R_FLASH_LP_Open ( flash_ctrl_t *const p_api_ctrl , flash_cfg_t const
*const p_cfg )
```

9.17.3.1 Brief description

Initialize the Low Power flash peripheral. Implements open.

9.17.3.2 Detailed description

The Open function initializes the Flash. It first copies the FCU firmware to FCURAM and sets the FCU Clock based on the current FCLK frequency. In addition, if Code Flash programming is enabled, the API code responsible for Code Flash programming will be copied to RAM.

This function must be called once prior to calling any other FLASH API functions. If a user supplied callback function is supplied, then the Flash Ready interrupt will be configured to call the users callback routine with an Event type describing the source of the interrupt for Data Flash operations. Subsequent to successfully completing this call p_ctrl->opened will be true.

NOTE: Providing a callback function in the supplied p_cfg->callback field, automatically configures the Flash for Data Flash to operate in non-blocking (BGO) mode.

Subsequent to a successful Open(), the Flash is ready to process additional Flash commands.

Table 1500:Return values

Name	Description
SSP_SUCCESS	Initialization was successful and timer has started.
SSP_FLASH_ERR_FAILURE	Failed to successfully enter Programming/Erase mode.

Table 1500:Return values (Continued)

Name	Description
SSP_ERR_TIMEOUT	Timed out waiting for FCU to be ready.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg or problem getting FMI info.
SSP_ERR_IRQ_BSP_DISABLED	Caller is requesting BGO but the Flash interrupts are not enabled.
SSP_ERR_FCLK	FCLK must be a minimum of 4 MHz for Flash operations.
SSP_ERR_IN_USE	Flash Open() has already been called.
SSP_ERR_HW_LOCKED	Flash module unable to get the Hardware lock for the Flash LP Periperhal.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet
- eventInfoGet

9.17.3.3 Function steps

- g_flash_lp_version is accessed by the ASSERT macro only, and so compiler toolchain can issue a warning that it is not accessed. The code below eliminates this warning and also ensures data structures are not optimized away.
- Insure API has not been opened
- While the Flash API is in use we will disable the FLash Cache.
- Allow Initialization if not initialized or if no operation is ongoing and re-initialization is desired
- Check that API is not already Open
- Set the parameters struct based on the user supplied settings
- Setup the Flash interrupt callback based on the caller's info. If the Flash interrupt is not enabled in the BSP then this will return SSP_ERR_IRQ_BSP_DISABLED
- Make sure Flash interrupts are disabled, they are only used in BGO mode
- Save callback function pointer

9.17.4 R_FLASH_LP_Write

```
ssp_err_t R_FLASH_LP_Write ( flash_ctrl_t  *const p_api_ctrl ,   uint32_t
const src_address ,   uint32_t flash_address ,   uint32_t const num_bytes )
```

9.17.4.1 Brief description

Write to the specified Code or Data Flash memory area. Implements write.

9.17.4.2 Detailed description

The minimum/maximum number of bytes, as well as the invalid address and programming boundaries supported and enforced by this function are dependent on the MCU in use as well as the part package size. Please see the User manual and/or requirements document for additional information.

Table 1501:Return values

Name	Description
SSP_SUCCESS	Operation successful. If BGO is enabled this means the operation was started successfully.
SSP_ERR_IN_USE	The Flash peripheral is busy with a prior on-going transaction.
SSP_ERR_NOT_OPEN	The Flash API is not Open.
SSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of attempting to Write an area that is protected by an Access Window.
SSP_ERR_WRITE_FAILED	Status is indicating a Programming error for the requested operation. This may be returned if the requested Flash area is not blank.
SSP_ERR_TIMEOUT	Timed out waiting for FCU operation to complete.
SSP_ERR_INVALID_SIZE	Number of bytes provided was not a multiple of the programming size or exceeded the maximum range.
SSP_ERR_INVALID_ADDRESS	Invalid address was input or address not on programming boundary.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled and a request to write CF was requested.

9.17.4.3 Function steps

- Get the block information for this address

9.17.5 R_FLASH_LP_Read

```
ssp_err_t R_FLASH_LP_Read ( flash_ctrl_t  *const p_api_ctrl ,   uint8_t
* p_dest_address ,   uint32_t const flash_address ,   uint32_t const num_bytes )
```

9.17.5.1 Brief description

Read the requested number of bytes from the supplied Data or Code Flash address. Implements read.

9.17.5.2 Detailed description

The minimum/maximum number of blocks, as well as the invalid address and programming boundaries supported and enforced by this function are dependent on the MCU in use as well as the part package size. Please see the User manual and/or requirements document for additional information.

NOTE: This function is provided simply for the purposes of maintaining a complete interface. It is possible (and recommended), to read Flash memory directly.

Table 1502:Return values

Name	Description
SSP_SUCCESS	Operation successful.
SSP_ERR_INVALID_ADDRESS	Invalid Flash address was supplied.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_dest_address
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

9.17.5.3 Function steps

- Eliminate warning if parameter checking is disabled.

9.17.6 R_FLASH_LP_Erase

```
ssp_err_t R_FLASH_LP_Erase ( flash_ctrl_t  *const p_api_ctrl ,   uint32_t
const address ,   uint32_t const num_blocks )
```

9.17.6.1 Brief description

Erase the specified Code or Data Flash blocks. Implements erase.

9.17.6.2 Detailed description

The minimum/maximum number of blocks, as well as the invalid address and programming boundaries supported and enforced by this function are dependent on the MCU in use as well as the part package size. Please see the User manual and/or requirements document for additional information.

Table 1503:Return values

Name	Description
SSP_SUCCESS	Successful open.
SSP_ERR_INVALID_BLOCKS	Invalid number of blocks specified
SSP_ERR_INVALID_ADDRESS	Invalid address specified
SSP_ERR_IN_USE	Other flash operation in progress, or API not initialized
SSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of attempting to Erase an area that is protected by an Access Window.
SSP_ERR_ASSERTION	NULL provided for p_ctrl
SSP_ERR_NOT_OPEN	The Flash API is not Open.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled and a request to erase CF was requested.

9.17.6.3 Function steps

- Get the block information for this address
- Update Flash state and enter the respective Code or Data Flash P/E mode, may return SSP_ERR_IN_USE
- Still good to go?
- Is this a request to Erase Code Flash?
- This is a request to erase Data Flash
- Erase the Blocks, give this function the details about this block
- If in non-BGO mode, the current operation is complete. Exit PE mode and return status

9.17.7 R_FLASH_LP_BlankCheck

```
ssp_err_t R_FLASH_LP_BlankCheck ( flash_ctrl_t  *const p_api_ctrl ,   uint32_t
const address ,   uint32_t num_bytes ,  flash_result_t  * p_blank_check_result )
```

9.17.7.1 Brief description

Perform a blank check on the specified address area. Implements blankCheck.

9.17.7.2 Detailed description

The minimum/maximum number of bytes, as well as the invalid address and programming boundaries supported and enforced by this function are dependent on the MCU in use as well as the part package size. Please see the User manual and/or requirements document for additional information. The number of bytes for Data Flash blank checking must be between (1 and FLASH_DATA_BLANK_CHECK_MAX). The number of bytes for Code Flash blank checking must be between (1 and FLASH_CODE_BLANK_CHECK_MAX).

Table 1504:Return values

Name	Description
SSP_SUCCESS	Blankcheck operation completed with result in p_blank_check_result, or blankcheck started and in-progess (BGO mode).
SSP_ERR_INVALID_ADDRESS	Invalid data flash address was input
SSP_ERR_INVALID_SIZE	'num_bytes' was either too large or not aligned for the CF/DF boundary size.
SSP_ERR_IN_USE	Flash is busy with an on-going operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled and a request to Blank Check CF was requested.

9.17.7.3 Function steps

- Get the block information for this address
- Update Flash state and enter the respective Code or Data Flash P/E mode, may return SSP_ERR_IN_USE
- SSP_ERR_IN_USE would indicate that a BGO operation is underway, so don't reset in that case

9.17.8 R_FLASH_LP_StatusGet

```
ssp_err_t R_FLASH_LP_StatusGet ( flash_ctrl_t  *const p_api_ctrl )
```

9.17.8.1 Brief description

Query the FLASH for its status. Implements statusGet.

9.17.8.2 Detailed description

Table 1505:Return values

Name	Description
SSP_SUCCESS	Flash is ready and available to accept commands.
SSP_ERR_IN_USE	Flash is busy with an on-going operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

9.17.8.3 Function steps

- Eliminate warning if parameter checking is disabled.
- Return flash status

9.17.9 R_FLASH_LP_AccessWindowSet

```
ssp_err_t R_FLASH_LP_AccessWindowSet ( flash_ctrl_t  *const p_api_ctrl ,
  uint32_t const start_addr ,   uint32_t const end_addr )
```

9.17.9.1 Brief description

Configure an access window for the Code Flash memory. Implements accessWindowSet.

9.17.9.2 Detailed description

An access window defines a contiguous area in Code Flash for which programming/erase is enabled. This area is on block boundaries. The block containing start_addr is the first block. The block containing end_addr is the last block. The access window then becomes first block > last block inclusive. Anything outside this range of Code Flash is then write protected. As an example, if you wanted to place an accesswindow on Code Flash Blocks 0 and 1, such that only those two blocks were writable, you would need to specify (address in block 0, address in block 2) as the respective start and end address. NOTE: If the start address and end address are set to the same value, then the access window is effectively removed. This accomplishes the same functionality as R_FLASH_LP_AccessWindowClear.

The invalid address and programming boundaries supported and enforced by this function are dependent on the MCU in use as well as the part package size. Please see the User manual and/or requirements document for additional information.

Table 1506:Return values

Name	Description
SSP_SUCCESS	Access window successfully configured.
SSP_ERR_INVALID_ADDRESS	Invalid settings for start_addr and/or end_addr.
SSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled.
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

9.17.9.3 Function steps

- Eliminate warning if parameter checking is disabled.
- Remove warnings generated when Code Flash code is not compiled in.
- < For consistency with _LP API we return error if Code Flash not enabled

9.17.10 R_FLASH_LP_AccessWindowClear

```
ssp_err_t R_FLASH_LP_AccessWindowClear ( flash_ctrl_t  *const p_api_ctrl )
```

9.17.10.1 Brief description

Remove any access window that is configured in the Code Flash. Implements accessWindowClear. On successful return from this call all Code Flash is writable.

9.17.10.2 Detailed description

Table 1507:Return values

Name	Description
SSP_SUCCESS	Access window successfully removed.
SSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.

Table 1507:Return values (Continued)

Name	Description
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled.
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

9.17.10.3 Function steps

- Eliminate warning if parameter checking is disabled.
- < Return error if Code Flash not enabled

9.17.11 R_FLASH_LP_Reset

```
ssp_err_t R_FLASH_LP_Reset ( flash_ctrl_t  *const p_api_ctrl )
```

9.17.11.1 Brief description

Reset the FLASH peripheral. Implements reset.

9.17.11.2 Detailed description

No attempt is made to grab the Flash software lock before executing the reset since the assumption is that a reset will terminate any existing operation.

Table 1508:Return values

Name	Description
SSP_SUCCESS	Flash circuit successfully reset.
SSP_ERR_ASSERTION	NULL provided for p_ctrl
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

9.17.12 R_FLASH_LP_StartUpAreaSelect

```
ssp_err_t R_FLASH_LP_StartUpAreaSelect ( flash_ctrl_t  *const p_api_ctrl ,
 flash_startup_area_swap_t  swap_type ,   bool is_temporary )
```

9.17.12.1 Brief description

Select which block is used as the startup area block. Implements startupAreaSelect.

9.17.12.2 Detailed description

Selects which block - Default (Block 0) or Alternate (Block 1) is used as the startup area block. The provided parameters determine which block will become the active startup block and whether that action will be immediate (but temporary), or permanent subsequent to the next reset. Doing a temporary switch might appear to have limited usefulness. If there is an access window in place such that Block 0 is write protected, then one could do a temporary switch, update the block and switch them back without having to touch the access window.

Table 1509:Return values

Name	Description
SSP_SUCCESS	Start-up area successfully toggled.
SSP_ERR_IN_USE	Flash is busy with an on-going operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

9.17.12.3 Function steps

- This is already the normal reset behavior. There's nothing to do
- Update Flash state and enter the respective Code or Data Flash P/E mode, may return SSP_ERR_IN_USE
- Success?
- Return to read mode
- If there is an error, then reset the Flash. This will clear error flags and exit the P/E mode

9.17.13 R_FLASH_LP_UpdateFlashClockFreq

```
ssp_err_t R_FLASH_LP_UpdateFlashClockFreq ( flash_ctrl_t  *const p_api_ctrl )
```

9.17.13.1 Brief description

Indicate to the already open Flash API that the FCLK has changed. Implements r_flash_t::updateFlashClockFreq.

9.17.13.2 Detailed description

This could be the case if the application has changed the system clock, and therefore the FCLK. Failure to call this function subsequent to changing the FCLK could result in damage to the flash macro. This function uses R_CGC_SystemClockFreqGet to get the current FCLK frequency.

Table 1510:Return values

Name	Description
SSP_SUCCESS	Start-up area successfully toggled.
SSP_ERR_IN_USE	Flash is busy with an on-going operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

9.17.13.3 Function steps

- Grab the current flash state
- API already in this state
- < Check FCLK, calculate timeout values

9.17.14 R_FLASH_LP_InfoGet

```
ssp_err_t R_FLASH_LP_InfoGet ( flash_ctrl_t  *const p_api_ctrl ,  flash_info_t
*const p_info )
```

9.17.14.1 Brief description

Returns the information about the flash regions. Implements infoGet.

9.17.14.2 Detailed description

Table 1511:Return values

Name	Description
SSP_SUCCESS	Successful retrieved the request information.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_info.

9.17.15 R_FLASH_LP_Close

```
ssp_err_t R_FLASH_LP_Close ( flash_ctrl_t  *const p_api_ctrl )
```

9.17.15.1 Brief description

Release any resources that were allocated by the Flash API. Implements close.

9.17.15.2 Detailed description

Table 1512:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.

9.17.15.3 Function steps

- Eliminate warning if parameter checking is disabled.
- Return the hardware lock for the Flash
- Restore the FLash Cache state to what it was before we opened the Flash API.
- API is now closed
- Release the lock

9.17.16 R_FLASH_LP_VersionGet

```
ssp_err_t R_FLASH_LP_VersionGet ( ssp_version_t  *const p_version )
```

9.17.16.1 Brief description

Get FLASH HAL driver version.

9.17.16.2 Detailed description

Table 1513:Return values

Name	Description
SSP_SUCCESS	- operation performed successfully

NOTE: This function is reentrant.

9.17.17 Extensions

9.17.17.1 flash_lp_instance_ctrl_t

flash_lp_instance_ctrl_t

Detailed description

Flash instance control block. DO NOT INITIALIZE.

Variables

- uint32_t opened

  To check whether api has been opened or not.
- void * p_reg

  Base address of flash registers.
- void(* p_callback)(*p_args)
- bsp_cache_state_t cache_state

  Used to disable and then restore Flash Cache while API is open.
- IRQn_Type irq

  Flash ready interrupt number.

9.18 High-performance Flash

Driver for the High-performance Flash Memory (S7G2 and S5D9).

This module supports the Flash interface for the High Performance FLASH peripheral.

9.18.1 Functions

- R_FLASH_HP_Open
- R_FLASH_HP_Write
- R_FLASH_HP_Read
- R_FLASH_HP_Erase
- R_FLASH_HP_BlankCheck
- R_FLASH_HP_StatusGet
- R_FLASH_HP_AccessWindowSet
- R_FLASH_HP_AccessWindowClear
- R_FLASH_HP_Reset
- R_FLASH_HP_StartUpAreaSelect

- R_FLASH_HP_UpdateFlashClockFreq
- R_FLASH_HP_InfoGet
- R_FLASH_HP_Close
- R_FLASH_HP_VersionGet
- flash_lock_state
- flash_ReleaseState
- setup_for_pe_mode
- flash_get_block_info
- flash_setup
- flash_open_setup
- flash_write_initiate
- flash_operation_complete
- flash_blank_check_address_checking
- flash_blank_check_setup
- flash_blank_check_initiate
- flash_fmi_setup

9.18.2 Defines

- #define FLASH_HP_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define FLASH_HP_CODE_VERSION_MINOR

  Initial value: `(5U)`
- #define PLACE_IN_RAM_SECTION

  Initial value:

  S5D9 and S5D5 MCUs uses RV40F Phase 2 Flash technology. This macro will eventually be migrated to bsp_feature.h.

9.18.3 R_FLASH_HP_Open

```
ssp_err_t R_FLASH_HP_Open ( flash_ctrl_t *const p_api_ctrl , flash_cfg_t const
*const p_cfg )
```

9.18.3.1 Brief description

Initializes the flash peripheral. Implements open.

9.18.3.2 Detailed description

The Open function initializes the Flash. It first copies the FCU firmware to FCURAM and sets the FCU Clock based on the current FCLK frequency. In addition, if Code Flash programming is enabled, the API code responsible for Code Flash programming will be copied to RAM.

This function must be called once prior to calling any other FLASH API functions. If a user supplied callback function is supplied, then the Flash Ready and Error interrupts will be configured to call the users callback routine with an Event type describing the source of the interrupt.

NOTE: Providing a callback function in the supplied p_cfg->callback field, automatically configures the Flash for Data Flash to operate in non-blocking (BGO) mode.

Subsequent to a successful Open(), the Flash is ready to process additional Flash commands.

Table 1514:Return values

Name	Description
SSP_SUCCESS	Initialization was successful and timer has started.
SSP_FLASH_ERR_FAILURE	Failed to successfully enter Programming/Erase mode.
SSP_ERR_TIMEOUT	Timed out waiting for FCU to be ready.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg or problem getting FMI info.
SSP_ERR_IRQ_BSP_DISABLED	Caller is requesting BGO but the Flash interrupts are not enabled.
SSP_ERR_FCLK	FCLK must be a minimum of 4 MHz for Flash operations.
SSP_ERR_HW_LOCKED	FLASH peripheral has already been initialized and is in use.

9.18.3.3 Function steps

- g_flash_hp_version is accessed by the ASSERT macro only, and so compiler toolchain can issue a warning that it is not accessed. The code below eliminates this warning and also ensures data structures are not optimized away.
- While the Flash API is in use we will disable the FLash Cache.
- Allow Initialization if not initialized or if no operation is ongoing and re-initialization is desired
- Check that API is not already Open
- Set the parameters struct based on the user supplied settings
- Setup the Flash interrupt callback based on the caller's info. If both Flash interrupts are not enabled in the BSP then this will return SSP_ERR_IRQ_BSP_DISABLED
- Make sure Flash interrupts are disabled, they are only used in BGO mode

- Save callback function pointer

9.18.4 R_FLASH_HP_Write

```
ssp_err_t R_FLASH_HP_Write ( flash_ctrl_t  *const p_api_ctrl ,   uint32_t
const src_address ,   uint32_t flash_address ,   uint32_t const num_bytes )
```

9.18.4.1 Brief description

Writes to the specified Code or Data Flash memory area. Implements write.

9.18.4.2 Detailed description

Table 1515:Return values

Name	Description
SSP_SUCCESS	Operation successful. If BGO is enabled this means the operation was started successfully.
SSP_ERR_IN_USE	The Flash peripheral is busy with a prior on-going transaction.
SSP_ERR_NOT_OPEN	The Flash API is not Open.
SSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of attempting to Write an area that is protected by an Access Window.
SSP_ERR_WRITE_FAILED	Status is indicating a Programming error for the requested operation. This may be returned if the requested Flash area is not blank.
SSP_ERR_TIMEOUT	Timed out waiting for FCU operation to complete.
SSP_ERR_INVALID_SIZE	Number of bytes provided was not a multiple of the programming size or exceeded the maximum range.
SSP_ERR_INVALID_ADDRESS	Invalid address was input or address not on programming boundary.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled and a request to write CF was requested.

9.18.5 R_FLASH_HP_Read

```
ssp_err_t R_FLASH_HP_Read ( flash_ctrl_t  *const p_api_ctrl ,   uint8_t
* p_dest_address ,   uint32_t const flash_address ,   uint32_t const num_bytes )
```

9.18.5.1 Brief description

Reads the requested number of bytes from the supplied Data or Code Flash memory address. Implements read.

9.18.5.2 Detailed description

NOTE: This function is provided simply for the purposes of maintaining a complete interface. It is possible (and recommended), to read Flash memory directly.

Table 1516:Return values

Name	Description
SSP_SUCCESS	Operation successful.
SSP_ERR_INVALID_ADDRESS	Invalid Flash address was supplied.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_dest_address

9.18.5.3 Function steps

- Eliminate warning if parameter checking is disabled.

9.18.6 R_FLASH_HP_Erase

```
ssp_err_t R_FLASH_HP_Erase ( flash_ctrl_t  *const p_api_ctrl ,   uint32_t
const address ,   uint32_t const num_blocks )
```

9.18.6.1 Brief description

Erases the specified Code or Data Flash blocks. Implements erase by the block_erase_address.

9.18.6.2 Detailed description

Table 1517:Return values

Name	Description
SSP_SUCCESS	Successful open.
SSP_ERR_INVALID_BLOCKS	Invalid number of blocks specified
SSP_ERR_INVALID_ADDRESS	Invalid address specified
SSP_ERR_IN_USE	Other flash operation in progress, or API not initialized
SSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of attempting to Erase an area that is protected by an Access Window.
SSP_ERR_ASSERTION	NULL provided for p_ctrl
SSP_ERR_NOT_OPEN	The Flash API is not Open.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled and a request to erase CF was requested.

9.18.6.3 Function steps

- Update Flash state and enter the respective Code or Data Flash P/E mode, may return SSP_ERR_IN_USE

9.18.7 R_FLASH_HP_BlankCheck

```
ssp_err_t R_FLASH_HP_BlankCheck ( flash_ctrl_t  *const p_api_ctrl ,   uint32_t
const address ,   uint32_t num_bytes ,  flash_result_t  * p_blank_check_result )
```

9.18.7.1 Brief description

Performs a blank check on the specified address area. Implements blankCheck.

9.18.7.2 Detailed description

Table 1518:Return values

Name	Description
SSP_SUCCESS	Blankcheck operation completed with result in p_blank_check_result, or blankcheck started and in-progess (BGO mode).
SSP_ERR_INVALID_ADDRESS	Invalid data flash address was input.
SSP_ERR_INVALID_SIZE	'num_bytes' was either too large or not aligned for the CF/DF boundary size.
SSP_ERR_IN_USE	Other flash operation in progress or API not initialized.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.

9.18.8 R_FLASH_HP_StatusGet

```
ssp_err_t R_FLASH_HP_StatusGet ( flash_ctrl_t  *const p_api_ctrl )
```

9.18.8.1 Brief description

Query the FLASH peripheral for its status. Implements statusGet.

9.18.8.2 Detailed description

Table 1519:Return values

Name	Description
SSP_SUCCESS	FLASH peripheral is ready to use.
SSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.

9.18.8.3 Function steps

- Eliminate warning if parameter checking is disabled.

9.18.9 R_FLASH_HP_AccessWindowSet

```
ssp_err_t R_FLASH_HP_AccessWindowSet ( flash_ctrl_t  *const p_api_ctrl ,
  uint32_t const start_addr ,   uint32_t const end_addr )
```

9.18.9.1 Brief description

Configure an access window for the Code Flash memory using the provided start and end address. An access window defines a contiguous area in Code Flash for which programming/erase is enabled. This area is on block boundaries. The block containing start_addr is the first block. The block containing end_addr is the last block. The access window then becomes first block > last block inclusive. Anything outside this range of Code Flash is then write protected.

9.18.9.2 Detailed description

NOTE: If the start address and end address are set to the same value, then the access window is effectively removed. This accomplishes the same functionality as R_FLASH_HP_AccessWindowClear.

Implements accessWindowSet.

Table 1520:Return values

Name	Description
SSP_SUCCESS	Access window successfully configured.
SSP_ERR_INVALID_ADDRESS	Invalid settings for start_addr and/or end_addr.
SSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled.
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

9.18.9.3 Function steps

- Eliminate warning if parameter checking is disabled.
- Remove warnings generated when Code Flash code is not compiled in.
- < For consistency with _LP API we return error if Code Flash not enabled

9.18.10 R_FLASH_HP_AccessWindowClear

```
ssp_err_t R_FLASH_HP_AccessWindowClear ( flash_ctrl_t  *const p_api_ctrl )
```

9.18.10.1 Brief description

Remove any access window that is currently configured in the Code Flash. Subsequent to this call all Code Flash is writable. Implements accessWindowClear.

9.18.10.2 Detailed description

Table 1521:Return values

Name	Description
SSP_SUCCESS	Access window successfully removed.
SSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.
SSP_ERR_INVALID_ARGUMENT	Code Flash Programming is not enabled.
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

9.18.10.3 Function steps

- Eliminate warning if parameter checking is disabled.
- < For consistency with _LP API we return error if Code Flash not enabled

9.18.11 R_FLASH_HP_Reset

```
ssp_err_t R_FLASH_HP_Reset ( flash_ctrl_t  *const p_api_ctrl )
```

9.18.11.1 Brief description

Resets the FLASH peripheral. Implements reset. No attempt is made to grab the Flash software lock before executing the reset since the assumption is that a reset will terminate any existing operation.

9.18.11.2 Detailed description

Table 1522:Return values

Name	Description
SSP_SUCCESS	Flash circuit successfully reset.

Table 1522:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	NULL provided for p_ctrl.

9.18.11.3 Function steps

- Eliminate warning if parameter checking is disabled.

9.18.12 R_FLASH_HP_StartUpAreaSelect

```
ssp_err_t R_FLASH_HP_StartUpAreaSelect ( flash_ctrl_t  *const p_api_ctrl ,
 flash_startup_area_swap_t  swap_type ,   bool is_temporary )
```

9.18.12.1 Brief description

Selects which block - Default (Block 0) or Alternate (Block 1) is used as the startup area block. The provided parameters determine which block will become the active startup block and whether that action will be immediate (but temporary), or permanent subsequent to the next reset. Doing a temporary switch might appear to have limited usefulness. If there is an access window in place such that Block 0 is write protected, then one could do a temporary switch, update the block and switch them back without having to touch the access window. Implements startupAreaSelect.

9.18.12.2 Detailed description

Table 1523:Return values

Name	Description
SSP_SUCCESS	Start-up area successfully toggled.
SSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl.

9.18.12.3 Function steps

- Eliminate warning if parameter checking is disabled.
- This is already the normal reset behavior. There's nothing to do
- Update Flash state and set the P/E mode. 'Standard' Flash requires that Configuration commands be issued while in Data Flash P/E mode. PHASE 2 Flash requires that Configuration commands be issued while in Code Flash P/E mode.

9.18.13 R_FLASH_HP_UpdateFlashClockFreq

```
ssp_err_t R_FLASH_HP_UpdateFlashClockFreq ( flash_ctrl_t  *const p_api_ctrl )
```

9.18.13.1 Brief description

Indicate to the already open Flash API, that the FCLK has changed since the Open(). This could be the case if the application has changed the system clock, and therefore the FCLK. Failure to call this function subsequent to changing the FCLK could result in damage to the flash macro. This function uses R_CGC_SystemClockFreqGet to get the current FCLK frequency. Implements updateFlashClockFreq.

9.18.13.2 Detailed description

Table 1524:Return values

Name	Description
SSP_SUCCESS	Start-up area successfully toggled.
SSP_ERR_IN_USE	Flash is busy with an on-going operation.
SSP_ERR_ASSERTION	NULL provided for p_ctrl
SSP_ERR_NOT_OPEN	Flash API has not yet been opened.

9.18.13.3 Function steps

- Eliminate warning if parameter checking is disabled.
- Grab the current flash state
- API already in this state
- < Check FCLK, calculate timeout values

9.18.14 R_FLASH_HP_InfoGet

```
ssp_err_t R_FLASH_HP_InfoGet ( flash_ctrl_t  *const p_api_ctrl ,  flash_info_t
*const p_info )
```

9.18.14.1 Brief description

Returns the information about the flash regions. Implements infoGet.

9.18.14.2 Detailed description

Table 1525:Return values

Name	Description
SSP_SUCCESS	Successful retrieved the request information.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_info.

9.18.14.3 Function steps

- Eliminate warning if parameter checking is disabled.

9.18.15 R_FLASH_HP_Close

```
ssp_err_t R_FLASH_HP_Close ( flash_ctrl_t  *const p_api_ctrl )
```

9.18.15.1 Brief description

Releases any resources that were allocated by the Open() or any subsequent Flash operations. Implements close.

9.18.15.2 Detailed description

Table 1526:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.

9.18.15.3 Function steps

- Eliminate warning if parameter checking is disabled.
- Return the hardware lock for the Flash
- Restore the FLash Cache state to what it was before we opened the Flash API.
- Release the lock
- Close the API

9.18.16 R_FLASH_HP_VersionGet

```
ssp_err_t R_FLASH_HP_VersionGet ( ssp_version_t  *const p_version )
```

9.18.16.1 Brief description

This function gets FLASH HAL driver version.

9.18.16.2 Detailed description

Table 1527:Return values

Name	Description
SSP_SUCCESS	- operation performed successfully

NOTE: This function is reentrant.

9.18.17 flash_lock_state

```
ssp_err_t flash_lock_state (  flash_states_t new_state )
```

9.18.17.1 Brief description

This function attempts to change the flash state to the new requested state. This can only happen if we are able to take the FLASH lock. If the lock is currently in use then we will return FLASH_ERR_BUSY, otherwise we will take the lock and change the state.

9.18.17.2 Detailed description

Internal functions.

Table 1528:Parameters

Name	Direction	Description
new_state	in	Which state to attempt to transition to.

Table 1529:Return values

Name	Description
SSP_SUCCESS	Successful.
SSP_ERR_IN_USE	Flash is busy with another operation.

9.18.18 flash_ReleaseState

```
flash_ReleaseState (  void  )
```

9.18.18.1 Brief description

This function releases the flash state so another flash operation can take place. This function is called by both the HLD and LLD layers (interrupt routines).

9.18.18.2 Detailed description

Table 1530:Return values

Name	Description
None	

9.18.19 setup_for_pe_mode

```
ssp_err_t setup_for_pe_mode ( flash_hp_instance_ctrl_t  *const p_ctrl ,
  flash_type_t flash_type ,   flash_states_t flash_state )
```

9.18.19.1 Brief description

This function places the flash in the requested Code or Data P/E mode.

9.18.19.2 Detailed description

Table 1531:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the Flash control block.

Table 1531:Parameters (Continued)

Name	Direction	Description
flash_type	in	FLASH_TYPE_CODE_FLASH or FLASH_TYPE_DATA_FLASH.
flash_state	in	Desired Flash state to transition into (ie FLASH_STATE_WRITING).

Table 1532:Return values

Name	Description
SSP_SUCCESS	Successful.
SSP_ERR_IN_USE	Flash is busy with another operation.

9.18.20 flash_get_block_info

```
flash_get_block_info (  uint32_t addr ,   flash_block_info_t * p_block_info )
```

9.18.20.1 Brief description

This function takes the supplied address and fills in the supplied block info structure with the respective details about the block in which it resides.

9.18.20.2 Detailed description

Table 1533:Return values

Name	Description
false	Supplied address is valid flash address on this MCU.
true	Supplied address is valid and p_block_info contains the details on this addresses block

9.18.20.3 Function steps

- Determine if this is a Code or Data Flash address, or something else (invalid)
- Is this address within this section of blocks?
- Make sure we know the end of the Code or Data Flash memory space.

- add in # of blocks in this section

9.18.21 flash_setup

```
ssp_err_t flash_setup (  R_FACI_Type * p_faci_reg )
```

9.18.21.1 Brief description

This function verifies that FCLK falls within the allowable range and calculates the timeout values based on the current FCLK frequency.

9.18.21.2 Detailed description

Table 1534:Return values

Name	Description
SSP_SUCCESS	Setup complete
SSP_ERR_FCLK	FCLK must be a minimum of 4 MHz for Flash operations.

9.18.21.3 Function steps

- We need clock frequencies to calculate the worst case timeout values.
- FCLK must be a minimum of 4 MHz for Flash operations

9.18.22 flash_open_setup

```
ssp_err_t flash_open_setup ( flash_hp_instance_ctrl_t  *const p_ctrl ,
  ssp_feature_t * p_ssp )
```

9.18.22.1 Brief description

This function does the completion setup for the R_FLASH_HP_Open function.

9.18.22.2 Function steps

- Check FCLK, calculate timeout values
- If we failed to successfully open then return the hardware lock
- API is now open

9.18.23 flash_write_initiate

```
ssp_err_t flash_write_initiate ( flash_hp_instance_ctrl_t  *const p_ctrl ,
  uint32_t const * src_start_address ,   uint32_t * dest_start_address ,
  uint32_t num_bytes )
```

9.18.23.1 Brief description

This function initiates the write sequence for the R_FLASH_HP_Write function.

9.18.23.2 Function steps

- Update Flash state and enter the respective Code or Data Flash P/E mode

9.18.24 flash_operation_complete

```
flash_operation_complete ( flash_hp_instance_ctrl_t  *const p_ctrl ,
ssp_err_t  err )
```

9.18.24.1 Brief description

This function performs the final cleanup for the erase, write and blankcheck functions.

9.18.24.2 Function steps

- SSP_ERR_IN_USE will be returned if we are in the process of executing a BGO operation. In this case we do not want to take any action as that will terminate the in process operation

9.18.25 flash_blank_check_address_checking

```
ssp_err_t flash_blank_check_address_checking (  uint32_t const address ,
  uint32_t num_bytes )
```

9.18.25.1 Brief description

This function performs the address checking required by the R_FLASH_HP_BlankCheck function.

9.18.25.2 Detailed description

Table 1535:Return values

Name	Description
SSP_SUCCESS	Parameter checking completed without error.

Table 1535:Return values (Continued)

Name	Description
SSP_ERR_INVALID_ADDRESS	Invalid data flash address was input.
SSP_ERR_INVALID_SIZE	'num_bytes' was either too large or not aligned for the CF/DF boundary size.

9.18.25.3 Function steps

- Is this a request to Blank Check Data Flash?. If so, num_bytes must be a multiple of the programming size

9.18.26 flash_blank_check_setup

```
ssp_err_t flash_blank_check_setup ( flash_ctrl_t  *const p_api_ctrl ,   uint8_t
* p_address ,   uint32_t num_bytes ,  flash_result_t  * p_blank_check_result )
```

9.18.26.1 Brief description

This function performs the setup phase required by the R_FLASH_HP_BlankCheck function.

9.18.26.2 Detailed description

Table 1536:Return values

Name	Description
SSP_SUCCESS	Setup completed completed without error.
SSP_ERR_PE_FAILURE	Failed to enter P/E mode

9.18.26.3 Function steps

- Blank checking for Code Flash does not require any FCU operations. The specified address area Can simply be checked for non 0xFF.
- Assume blank until we know otherwise

9.18.27 flash_blank_check_initiate

```
ssp_err_t flash_blank_check_initiate ( flash_hp_instance_ctrl_t  *const p_ctrl ,
  uint32_t const address ,   uint32_t num_bytes ,  flash_result_t
* p_blank_check_result )
```

9.18.27.1 Brief description

This function performs the Blank check phase required by the R_FLASH_HP_BlankCheck function.

9.18.27.2 Detailed description

Table 1537:Return values

Name	Description
SSP_SUCCESS	Setup completed completed without error.
SSP_ERR_PE_FAILURE	Failed to enter P/E mode

9.18.28 flash_fmi_setup

```
ssp_err_t flash_fmi_setup ( flash_hp_instance_ctrl_t  *const p_ctrl ,
 flash_cfg_t  const *const p_cfg ,   ssp_feature_t * p_ssp )
```

9.18.28.1 Brief description

This function initializes data required by the Flash based on information read from the FMI.

9.18.28.2 Detailed description

Table 1538:Return values

Name	Description
SSP_SUCCESS	FMI based setup success.
SSP_ERR_IN_USE	The Flash peripheral is busy with a prior on-going transaction.
SSP_ERR_ASSERTION	Problem getting FMI information.
SSP_ERR_HW_LOCKED	FLASH peripheral has already been initialized and is in use.

9.18.29 Extensions

9.18.29.1 flash_hp_instance_ctrl_t

flash_hp_instance_ctrl_t

Detailed description

Flash HP instance control block. DO NOT INITIALIZE.

Variables

- uint32_t opened

  To check whether api has been opened or not.
- R_FACI_Type * p_reg

  Base address of flash registers.
- void(* p_callback)(*p_args)
- bsp_cache_state_t cache_state

  User Callback function.

  Used to disable and then restore Flash Cache while API is open.
- IRQn_Type irq

  Flash ready interrupt number.
- IRQn_Type err_irq

  Flash error interrupt number.

9.19 FMI

Driver for accessing Factory MCU Information (FMI).
Read Factory MCU Information flash memory.

9.19.1 Functions

- R_FMI_Init
- R_FMI_ProductInfoGet
- R_FMI_UniqueIdGet
- R_FMI_FeatureGet
- R_FMI_EventInfoGet
- R_FMI_VersionGet

9.19.2 Variables

- g_fmi_on_fmi

9.19.3 Defines

- #define FMI_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define FMI_CODE_VERSION_MINOR

  Initial value: `(4U)`

9.19.4 R_FMI_Init

```
ssp_err_t R_FMI_Init ( void )
```

9.19.4.1 Detailed description

Initializes factory flash base pointer. Implements init.

Table 1539:Return values

Name	Description
SSP_SUCCESS	Factory flash initialization successful.
SSP_ERR_INVALID_FMI_DATA	The FMI data table provided is not valid, or at least one field in the part number is not compatible with the FMI data table provided.

9.19.4.2 Function steps

- Verify the provided MCU information data table is valid. This table is required to use the SSP. If the data in this table is not valid, log an unrecoverable error.
- Check to see if the factory flash is valid.
- The upper 16 bits of the base address of the factory flash must be 0x0100.
- If the factory flash is valid, store the base address for later and compare the part number to the part number mask match.

9.19.5 R_FMI_ProductInfoGet

```
ssp_err_t R_FMI_ProductInfoGet ( fmi_product_info_t ** pp_product_info )
```

9.19.5.1 Detailed description

Get pointer to Factory MCU Information product information record. Implements productInfoGet.

Table 1540:Return values

Name	Description
SSP_SUCCESS	Caller's pointer set to Product Information Record.
SSP_ERR_ASSERTION	Caller's pointer is NULL.

9.19.5.2 Function steps

- Use the factory flash if it is valid. Otherwise use the SSP MCU information.

9.19.6 R_FMI_UniqueIdGet

```
ssp_err_t R_FMI_UniqueIdGet (  fmi_unique_id_t * p_unique_id )
```

9.19.6.1 Detailed description

Get unique ID for this device. Implements uniqueIdGet.

Table 1541:Return values

Name	Description
SSP_SUCCESS	Unique ID stored in p_unique_id
SSP_ERR_ASSERTION	p_unique_id was NULL
SSP_ERR_INVALID_FACTORY_FLASH	Factory flash is not valid

9.19.6.2 Function steps

- If the factory flash is not valid, return an error.
- Store unique ID in p_unique_id.

9.19.7 R_FMI_FeatureGet

```
ssp_err_t R_FMI_FeatureGet (  ssp_feature_t const *const p_feature ,
  fmi_feature_info_t *const p_info )
```

9.19.7.1 Detailed description

Get feature information for the requested feature. Implements productFeatureGet.

Table 1542:Return values

Name	Description
SSP_SUCCESS	Feature information stored in p_info
SSP_ERR_ASSERTION	p_feature was NULL or p_info was NULL
SSP_ERR_IP_CHANNEL_NOT_PRESENT	Requested channel does not exist on this MCU
SSP_ERR_IP_UNIT_NOT_PRESENT	Requested unit does not exist on this MCU
SSP_ERR_INTERNAL	Requested feature is in a format not supported at this time

9.19.7.2 Function steps

- Get the factory flash base address for this IP.
- Find the factory flash base address for this unit.
- Populate the feature information.

9.19.8 R_FMI_EventInfoGet

```
ssp_err_t R_FMI_EventInfoGet (  ssp_feature_t const *const p_feature ,
  ssp_signal_t signal ,   fmi_event_info_t *const p_info )
```

9.19.8.1 Brief description

Get event information for the requested feature and signal. Implements eventInfoGet.

9.19.8.2 Detailed description

Table 1543:Return values

Name	Description
SSP_SUCCESS	Event information stored in p_info
SSP_ERR_ASSERTION	p_feature was NULL or p_info was NULL

Table 1543:Return values (Continued)

Name	Description
SSP_ERR_IRQ_BSP_DISABLED	Event information could not be found. p_info::irq is set to SSP_INVALID_VECTOR and p_info::event is set to 0xFF.

NOTE: The ssp_signal_t enums and ssp_ip_t are collectively defined for different modules. When used with the FMI API, the signal and IP should correspond as follows:

1) IP name (ssp_ip_t enum : SSP_IP_<peripheral>)
2) Signal name (ssp_signal_t enum: SSP_SIGNAL_<peripheral>_ <signal>), where <peripheral> is the same as the one used in IP name above.

9.19.9 R_FMI_VersionGet

```
ssp_err_t R_FMI_VersionGet ( ssp_version_t  *const p_version )
```

9.19.9.1 Detailed description

Get the driver version based on compile time macros. Implements versionGet.

Table 1544:Return values

Name	Description
SSP_SUCCESS	Caller's structure written.
SSP_ERR_ASSERTION	Caller's pointer is NULL.

9.19.10 g_fmi_on_fmi

```
fmi_api_t::g_fmi_on_fmi
```

9.20 GLCDC

Driver for the Graphics LCD Controller (GLCDC).

9.20.1 Summary

Implements Display Interface. This module supports the Graphics LCD Controller (GLCDC). It implements the display interface and drives LCD panels connected to the GLCDC pins.

9.20.2 Functions

- r_glcd_sync_signal_set
- r_glcd_background_screen_set
- r_glcd_graphics_layer_set
- r_glcd_output_block_set
- r_glcd_gamma_correction
- r_glcd_hsync_set
- r_glcd_vsync_set
- r_glcd_data_enable_set
- r_glcd_clock_set
- r_glcd_brightness_correction
- r_glcd_contrast_correction
- r_glcd_get_bit_size
- r_glcd_graphics_plane_format_set
- r_glcd_interrupt_enable
- r_glcd_output_data_order_set
- r_glcd_pixel_size_recalculate
- glcdc_line_detect_isr
- glcdc_underflow_1_isr
- glcdc_underflow_2_isr
- R_GLCD_Open
- R_GLCD_Close
- R_GLCD_Start
- R_GLCD_Stop
- R_GLCD_LayerChange
- R_GLCD_ColorCorrection
- R_GLCD_ClutUpdate
- R_GLCD_StatusGet
- R_GLCD_VersionGet

9.20.3 Variables

- module_version
- g_display_on_glcd
- default_glcd_cfg
- ctrl_blk

9.20.4 Defines

- #define GLCD_ERROR_RETURN

  Initial value:`#define SSP_ERROR_RETURN((a), (err), &g_module_name[0], &module_version)`

- #define GLCD_ADDRESS_ALIGNMENT_64B

  Initial value: `(64U)`

  The macro to use for 64-byte alignment checking, calculation

- #define OFFSET_MARGIN_MINUS_64PIX

  Initial value: `(64U)`

  This enables the GLCD to locate the foreground/background layer image, at the physical start of the left side of the display, if the offset of layer start position is a negative value, compared to the active video area. Shifting the base address of the layer image can be used in this operation, but it must be aligned to 64 byte boundary, so that the layer position adjustment can be made.

9.20.5 r_glcd_sync_signal_set

```
r_glcd_sync_signal_set ( R_GLCDC_Type * p_glcd_reg , display_cfg_t const *const p_cfg )
```

9.20.5.1 Brief description

Subroutine to configure the sync signal setting (TCON block setting)

9.20.5.2 Detailed description

Table 1545:Parameters

Name	Direction	Description
p_cfg	in	Pointer to the configuration structure for display interface

Table 1545:Parameters (Continued)

Name	Direction	Description
p_glcd_reg	in	Pointer to GLCD registers

Table 1546:Return values

Name	Description
void	

9.20.5.3 Function steps

- Applied default configuration if GLCD HAL configuration is NULL

9.20.6 r_glcd_background_screen_set

```
r_glcd_background_screen_set (  R_GLCDC_Type * p_glcd_reg ,  display_cfg_t
const *const p_cfg )
```

9.20.6.1 Brief description

Subroutine to configure the background screen setting.

9.20.6.2 Detailed description

- Panel timing setting
- Color setting for the background screen

Table 1547:Parameters

Name	Direction	Description
p_cfg	in	Pointer to the configuration structure for display interface
p_glcd_reg	in	Pointer to GLCD registers

Table 1548:Return values

Name	Description
void	

9.20.6.3 Function steps

- Set number of total cycle for a line including Sync & Back poach, Front poach
- Set the start position of Background screen
- Set the width of Background screen
- Set the Background color

9.20.7 r_glcd_graphics_layer_set

```
 r_glcd_graphics_layer_set (  R_GLCDC_Type * p_glcd_reg ,  display_input_cfg_t
const *const p_input ,  display_layer_t  const *const p_layer ,
 display_frame_layer_t  const frame )
```

9.20.7.1 Brief description

Subroutine to configure the graphics layer register settings which includes...

9.20.7.2 Detailed description

- Blend setting of foreground or background plane on background plane
- Rectangle area blending settings

Table 1549:Parameters

Name	Direction	Description
p_input	in	The input frame buffer configuration
p_layer	in	The layer configuration
frame	in	The number of input frame buffer
p_glcd_reg	in	Pointer to GLCD registers

Table 1550:Return values

Name	Description
void	

NOTE: This function does not perform parameter check and it would be expected to be done in the caller function.

9.20.7.3 Function steps

- If the base address is NULL, just set the later transparent and disable read memory access
- Set the base address of graphics plane
- Set the background color on graphics plane
- Set the number of data transfer times per line, 64 bytes are transferred in each transfer
- If line number descending mode is enable, change its sign
- Set the line offset address for accessing the graphics data on graphics plane
- When line repeating mode, always read data on same line(s)
- Set the line offset address for accessing the graphics data
- Set the line offset address for accessing the graphics data on graphics plane
- Set the frame number of the graphics plane
- Set the frame offset for accessing the graphics data on the graphics plane
- Set the start position of the graphics layers
- Set the start position of the rectangle area in the graphics layers
- Set the width of the graphics layers
- Set the alpha blending condition

9.20.8 r_glcd_output_block_set

```
 r_glcd_output_block_set (  R_GLCDC_Type * p_glcd_reg ,  display_cfg_t  const
*const p_cfg )
```

9.20.8.1 Brief description

Subroutine to configure the output control block register settings which includes...

9.20.8.2 Detailed description

- Bit endian / color order setting
- Output color setting
- Color correction setting

Table 1551:Parameters

Name	Direction	Description
p_cfg	in	Pointer to the configuration structure for display interface
p_glcd_reg	in	Pointer to GLCD registers

Table 1552:Return values

Name	Description
void	

9.20.8.3 Function steps

- Applied default configuration if the GLCD HAL configuration is NULL
- selects the output format
- In case of serial RGB, set as RGB888 format
- sets the pixel clock (the GLCD internal signal) frequency in case that the output format is 8-bit serial RGB
- sets the Brightness/contrast and Gamma Correction processing order
- Set the dithering mode

9.20.9 r_glcd_gamma_correction

```
 r_glcd_gamma_correction (  R_GLCDC_Type * p_glcd_reg ,  display_cfg_t  const
*const p_cfg )
```

9.20.9.1 Brief description

Subroutine to configure the gamma correction register setting.

9.20.9.2 Detailed description

Table 1553:Parameters

Name	Direction	Description
p_cfg	in	Pointer to the configuration structure for the display interface
p_glcd_reg	in	Pointer to GLCD registers

Table 1554:Return values

Name	Description
void	

9.20.10 r_glcd_hsync_set

```
r_glcd_hsync_set (  R_GLCDC_Type * p_glcd_reg ,  glcd_tcon_pin_t  tcon ,
display_timing_t  const * timing )
```

9.20.10.1 Brief description

Subroutine to configure the horizontal signal setting.

9.20.10.2 Detailed description

Table 1555:Parameters

Name	Direction	Description
p_glcd_reg	in	Pointer to GLCD registers
tcon	in	TCON pin select(GLCD_TCON_PIN_0|GLCD_TCON_PIN_1|GLCD_TCON_PIN_2|GLCD_TCON_PIN_3)
timing	in	Hsync signal timing

Table 1556:Return values

Name	Description
void	

9.20.11 r_glcd_vsync_set

```
r_glcd_vsync_set (  R_GLCDC_Type * p_glcd_reg ,  glcd_tcon_pin_t  tcon ,
display_timing_t  const *const timing )
```

9.20.11.1 Brief description

Subroutine to configure the vertical signal setting.

9.20.11.2 Detailed description

Table 1557:Parameters

Name	Direction	Description
p_glcd_reg	in	Pointer to GLCD registers
tcon	in	TCON pin select(GLCD_TCON_PIN_0|GLCD_TCON_PIN_1|GLCD_TCON_PIN_2|GLCD_TCON_PIN_3)
timing	in	Vsync signal timing

Table 1558:Return values

Name	Description
void	

9.20.12 r_glcd_data_enable_set

```
 r_glcd_data_enable_set (  R_GLCDC_Type * p_glcd_reg ,  glcd_tcon_pin_t
const tcon ,  display_timing_t  const *const vtiming ,  display_timing_t  const
*const htiming ,  display_signal_polarity_t  const polarity )
```

9.20.12.1 Brief description

Subroutine to configure the data enable(DE) signal setting.

9.20.12.2 Detailed description

Table 1559:Parameters

Name	Direction	Description
p_glcd_reg	in	Pointer to GLCD registers
tcon	in	TCON pin select(GLCD_TCON_PIN_0|GLCD_TCON_PIN_1|GLCD_TCON_PIN_2|GLCD_TCON_PIN_3)
vtiming	in	DE signal vertical timing
htiming	in	DE signal horizontal timing
polarity	in	DE signal porarity(DISPLAY_SIGNAL_POLARITY_LOACTIVE|DISPLAY_SIGNAL_POLARITY_HIACTIVE)

Table 1560:Return values

Name	Description
void	

9.20.13 r_glcd_clock_set

```
 r_glcd_clock_set (  R_GLCDC_Type * p_glcd_reg ,  display_cfg_t  const
*const p_cfg )
```

9.20.13.1 Brief description

Subroutine to configure dot clock setting.

9.20.13.2 Detailed description

Table 1561:Parameters

Name	Direction	Description
p_cfg	in	Pointer to the configuration structure for display interface
p_glcd_reg	in	Pointer to GLCD registers

Table 1562:Return values

Name	Description
void	

9.20.13.3 Function steps

- Selects input source for dot clock
- Sets division ratio
- Selects pixel clock output

9.20.14 r_glcd_brightness_correction

```
 r_glcd_brightness_correction (  R_GLCDC_Type * p_glcd_reg ,
 glcd_instance_ctrl_t  const *const p_ctrl ,  display_brightness_t  const
*const p_brightness )
```

9.20.14.1 Brief description

Subroutine to configure the brightness register settings. Pixel color output comes to be the value shown below processed by the brightness control block.

9.20.14.2 Detailed description

- Gout = Gin + p_cfg->output.brightness.g - 512 (output.brightness.g must be 10 bits value; up to 512)
- Bout = Bin + p_cfg->output.brightness.b - 512 (output.brightness.b must be 10 bits value; up to 512)
- Rout = Rin + p_cfg->output.brightness.r - 512 (output.brightness.r must be 10 bits value; up to 512)

Table 1563:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for Display Interface
p_brightness	in	Pointer to brightness configuration structure
p_glcd_reg	in	Pointer to GLCD registers

Table 1564:Return values

Name	Description
void	

9.20.14.3 Function steps

- Sets brightness correction register for each color in a pixel.
- If brightness setting in configuration is 'off', apply default value

9.20.15 r_glcd_contrast_correction

```
 r_glcd_contrast_correction (  R_GLCDC_Type * p_glcd_reg ,
 glcd_instance_ctrl_t  const *const p_ctrl ,  display_contrast_t  const
*const p_contrast )
```

9.20.15.1 Brief description

Subroutine to configure the contrast register settings. Pixel color output becomes the value shown below, processed by the contrast control block. Contrast can be changed between x0.000 to x1.992 (0x0:x0.000 / 0x80:x1.000 / 0xFF:x1.992).

9.20.15.2 Detailed description

- Gout = (Gin + p_contrast->g)/128
- Bout = (Bin + p_contrast->b)/128
- Rout = (Rin + p_contrast->r)/128

Table 1565:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block for the Display Interface
p_contrast	in	Pointer to the contrast configuration structure
p_glcd_reg	in	Pointer to GLCD registers

Table 1566:Return values

Name	Description
void	

9.20.15.3 Function steps

- Sets the contrast correction register for each color in a pixel.
- If the contrast setting in the configuration is set to 'off', apply default value

9.20.16 r_glcd_get_bit_size

```
r_glcd_get_bit_size ( display_in_format_t  const format )
```

9.20.16.1 Brief description

Subroutine to get the bit size of the specified format.

9.20.16.2 Detailed description

Table 1567:Parameters

Name	Direction	Description
format	in	Color format (specify display_in_format_t type enumeration value)

Table 1568:Return values

Name	Description
Bit	size

9.20.16.3 Function steps

- Get bit size and set color format
- < ARGB8888, 32bits
- < RGB888, 32bits
- < RGB565, 16bits
- < ARGB1555, 16bits
- < ARGB4444, 16bits
- < CLUT8
- < CLUT4
- < CLUT1

9.20.17 r_glcd_graphics_plane_format_set

```
r_glcd_graphics_plane_format_set (  R_GLCDC_Type * p_glcd_reg ,
display_in_format_t  const format ,  display_frame_layer_t  const frame )
```

9.20.17.1 Brief description

Subroutine to set the color format of graphics plane to the GLCD register.

9.20.17.2 Detailed description

Table 1569:Parameters

Name	Direction	Description
format	in	Color format (specify display_in_format_t type enumeration value)
frame	in	The number of input frame buffer
p_glcd_reg	in	Pointer to GLCD registers

Table 1570:Return values

Name	Description
void	

9.20.17.3 Function steps

- < ARGB8888, 32bits
- < RGB888, 32bits
- < RGB565, 16bits
- < ARGB1555, 16bits
- < ARGB4444, 16bits
- < CLUT8
- < CLUT4
- < CLUT1

9.20.18 r_glcd_interrupt_enable

```
 r_glcd_interrupt_enable (  R_GLCDC_Type * p_glcd_reg ,   IRQn_Type
* line_detect_irq ,   IRQn_Type * underflow_1_irq ,   IRQn_Type
* underflow_2_irq )
```

9.20.18.1 Brief description

Enable the glcd interrupt.

9.20.18.2 Detailed description

Table 1571:Parameters

Name	Direction	Description
p_glcd_reg	in	Pointer to GLCD registers
line_detect_irq	in	Pointer to IRQ
underflow_1_irq	in	Pointer to IRQ
underflow_2_irq	in	Pointer to IRQ

Table 1572:Return values

Name	Description
void	

9.20.18.3 Function steps

- Enable the GLCD interrupts

9.20.19 r_glcd_output_data_order_set

```
r_glcd_output_data_order_set (  R_GLCDC_Type * p_glcd_reg ,  display_cfg_t
const *const p_cfg )
```

9.20.19.1 Brief description

Configure endianness for output data and output byte order swapping.

9.20.19.2 Detailed description

Table 1573:Parameters

Name	Direction	Description
p_cfg	in	Pointer to the configuration structure for the display interface
p_glcd_reg	in	Pointer to GLCD registers

Table 1574:Return values

Name	Description
void	

9.20.19.3 Function steps

- selects big or little endian for output data
- selects the output byte order swapping

9.20.20 r_glcd_pixel_size_recalculate

```
 r_glcd_pixel_size_recalculate ( display_input_cfg_t  const *const p_input ,
 display_layer_t  const *const p_layer ,   recalculated_param_t
* p_recalculated ,   uint16_t * bit_size )
```

9.20.20.1 Brief description

Calculate the pixels to be displayed on window based on the offset of the graphic layer.

9.20.20.2 Detailed description

Table 1575:Parameters

Name	Direction	Description
p_input	in	The input frame buffer configuration
p_layer	in	The layer configuration
p_recalculated	inout	Pointer to store recalculated parameter
bit_size	in	Pointer to bit size of the color format

Table 1576:Return values

Name	Description
void	

9.20.20.3 Function steps

- If the offset of the graphics layer is greater than or equal to zero and it is less than or equal to the size of the display window, calculate actual pixel size to display.
- Calculate actual pixel size, the pixels to be displayed is less than the display window size
- Actual pixel size to display is same as the display window size
- If the offset of the graphics layer is less than zero, calculate the actual pixel size to display.
- If coordinate.x is a minus value, the layer image position can be adjusted not only by adjusting the horizontal offset but also by changing the base address of layer image. Since the base address has to be aligned to 64 bytes, we need to adjust the horizontal offset, which is the cycles from the 'internal zero' to the start cycle of active video region to achieve 1 pixel unit offset. We need to adjust the size of image to display as well.
- Base address must be aligned to a 64-bit address

- If graphics layer offset is beyond the display window size, set the pixel size to display to zero

9.20.21 glcdc_line_detect_isr

```
glcdc_line_detect_isr ( void )
```

9.20.21.1 Brief description

The line detection interrupt service routine. This ISR is called when the number of the display line reaches the designated number of lines. If a callback function is registered in R_GLCD_Open, it is called from this ISR and the DISPLAY_EVENT_LINE_DETECTION event code is set as its argument.

9.20.21.2 Detailed description

Table 1577:Return values

Name	Description
none	

9.20.21.3 Function steps

- Call back callback function if it is registered
- Clear interrupt flag in the register of the GLCD module
- Clear interrupt flag in the register of the NVIC module

9.20.22 glcdc_underflow_1_isr

```
glcdc_underflow_1_isr ( void )
```

9.20.22.1 Brief description

The graphics plane 1 underflow detection interrupt service routine. This ISR is called when the underflow occurs in the graphics plane 1 control block. If a callback function is registered in R_GLCD_Open, it is called back from this ISR and the DISPLAY_EVENT_GR1_UNDERFLOW event code is set as its argument.

9.20.22.2 Detailed description

Table 1578:Return values

Name	Description
none	

9.20.22.3 Function steps

- Call back callback function if it is registered
- Clear interrupt flag in the register of the GLCD module
- Clear interrupt flag in the register of the NVIC module

9.20.23 glcdc_underflow_2_isr

```
glcdc_underflow_2_isr (  void  )
```

9.20.23.1 Brief description

The graphics plane 2 underflow detection interrupt service routine. This ISR is called when the underflow occurs in the graphics plane 2 control block. If a callback function is registered in R_GLCD_Open, it is called from this ISR and the DISPLAY_EVENT_GR2_UNDERFLOW event code is set as its argument.

9.20.23.2 Detailed description

Table 1579:Return values

Name	Description
none	

9.20.23.3 Function steps

- Call the callback function if it is registered
- Clear interrupt flag in the register of the GLCD module
- Clear interrupt flag in the register of the NVIC module

9.20.24 R_GLCD_Open

```
ssp_err_t R_GLCD_Open ( display_ctrl_t *const p_api_ctrl , display_cfg_t const
*const p_cfg )
```

9.20.24.1 Brief description

Open GLCDC module.

9.20.24.2 Detailed description

Implements

- open.

Table 1580:Return values

Name	Description
SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter in the argument.
SSP_ERR_HW_LOCKED	GLCDC resource is locked.
SSP_ERR_CLOCK_GENERATION	Dot clock cannot be generated from clock source.
SSP_ERR_INVALID_TIMING_SETTING	Invalid panel timing parameter.
SSP_ERR_INVALID_LAYER_SETTING	Invalid layer setting found.
SSP_ERR_INVALID_LAYER_FORMAT	Invalid format is specified.
SSP_ERR_INVALID_GAMMA_SETTING	Invalid gamma correction setting found.

NOTE: PCLKA must be supplied to Graphics LCD Controller (GLCDC) and GLCDC pins must be set in IOPORT before calling this API.

9.20.24.3 Function steps

- Lock the GLCD resource

- Supply the peripheral clock to the GLCD module
- Release GLCD from a SW reset status.
- Set the dot clock frequency
- Set the panel signal timing
- Configure the background screen
- Store back poach position to the control block (needed to define the layer blending position later)
- Configure the graphics plane layers
- Configure the output control block
- Configure the color correction setting (brightness, brightness and gamma correction)
- Change GLCD driver state
- Save callback function
- Save user defined context
- Save the display interface context into GLCD HAL control block
- Set the line number which is suppose to happen the line detect interrupt

9.20.25 R_GLCD_Close

```
ssp_err_t R_GLCD_Close ( display_ctrl_t  *const p_api_ctrl )
```

9.20.25.1 Brief description

Close GLCDC module.

9.20.25.2 Detailed description

Implements

- close.

Table 1581:Return values

Name	Description
SSP_SUCCESS	Device was closed successfully.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_NOT_OPEN	The function call is performed when the driver state is not equal to DISPLAY_STATE_CLOSED.

Table 1581:Return values (Continued)

Name	Description
SSP_ERR_INVALID_UPDATE_TIMING	A function call is performed when the GLCD is updating register values internally.

NOTE: This API can be called when the driver is not in DISPLAY_STATE_CLOSED state. It returns an error if the register update operation for the background screen generation block is being held.

9.20.25.3 Function steps

- Disable the GLCD interrupts
- Reset the GLCD hardware
- Halt the peripheral clock to the GLCD module
- Unlock the GLCD resource

9.20.26 R_GLCD_Start

```
ssp_err_t R_GLCD_Start ( display_ctrl_t  *const p_api_ctrl )
```

9.20.26.1 Brief description

Start GLCDC module.

9.20.26.2 Detailed description

Implements

- start.

Table 1582:Return values

Name	Description
SSP_SUCCESS	Device was started successfully.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_MODE	Function call is performed when the driver state is not DISPLAY_STATE_OPENED.

NOTE: This API can be called when the driver is not in DISPLAY_STATE_OPENED status.

9.20.26.3 Function steps

- Start to output the vertical and horizontal synchronization signals and screen data.

9.20.27 R_GLCD_Stop

```
ssp_err_t R_GLCD_Stop ( display_ctrl_t  *const p_api_ctrl )
```

9.20.27.1 Brief description

Stop GLCDC module.

9.20.27.2 Detailed description

Implements

- stop.

Table 1583:Return values

Name	Description
SSP_SUCCESS	Device was stopped successfully
SSP_ERR_ASSERTION	Pointer to the control block is NULL
SSP_ERR_INVALID_MODE	Function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING.
SSP_ERR_INVALID_UPDATE_TIMING	The function call is performed while the GLCD is updating register values internally.

NOTE: This API can be called when the driver is in the DISPLAY_STATE_DISPLAYING state. It returns an error if the register update operation for the background screen generation blocks, the graphics data I/F blocks, or the output control block is being held.

9.20.27.3 Function steps

- Stop outputting the vertical and horizontal synchronization signals and screen data.

9.20.28 R_GLCD_LayerChange

```
ssp_err_t R_GLCD_LayerChange ( display_ctrl_t  const *const p_api_ctrl ,
 display_runtime_cfg_t  const *const p_cfg ,  display_frame_layer_t  frame )
```

9.20.28.1 Brief description

Change layer parameters of GLCDC module at runtime.

9.20.28.2 Detailed description

Implements

- layerChange.

Table 1584:Return values

Name	Description
SSP_SUCCESS	Changed layer parameters of GLCDC module successfully.
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
SSP_ERR_INVALID_MODE	A function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING.
SSP_ERR_INVALID_ARGUMENT	An invalid parameter is found in the argument.
SSP_ERR_INVALID_UPDATE_TIMING	A function call is performed while the GLCD is updating register values internally.

NOTE: This API can be called when the driver is in DISPLAY_STATE_DISPLAYING state. It returns an error if the register update operation for the background screen generation blocks or the graphics data I/F block is being held.

9.20.28.3 Function steps

- Configure the graphics plane layers

- Reflect the graphics module register value to the GLCD internal operations (at the timing of the next Vsync assertion)

9.20.29 R_GLCD_ColorCorrection

```
ssp_err_t R_GLCD_ColorCorrection ( display_ctrl_t  const *const p_api_ctrl ,
 display_correction_t  const *const p_correction )
```

9.20.29.1 Brief description

Perform color correction by GLCDC module.

9.20.29.2 Detailed description

Implements

- correction.

Table 1585:Return values

Name	Description
SSP_SUCCESS	Color correction by GLCDC module was performed successfully.
SSP_ERR_ASSERTION	Pointer to the control block or the display correction structure is NULL.
SSP_ERR_INVALID_MODE	Function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING.
SSP_ERR_INVALID_UPDATE_TIMING	A function call is performed while the GLCDC is updating registers internally.

NOTE: This API can be called when the driver is in the DISPLAY_STATE_DISPLAYING state. It returns an error if the register update operation for the background screen generation blocks or the output control block is being held.

9.20.29.3 Function steps

- Configure the brightness and contrast correction register setting.
- Update the Output block register setting.

9.20.30 R_GLCD_ClutUpdate

```
ssp_err_t R_GLCD_ClutUpdate ( display_ctrl_t  const *const p_api_ctrl ,
 display_clut_cfg_t  const *const p_clut_cfg ,  display_frame_layer_t  frame )
```

9.20.30.1 Brief description

Update Color Look Up Table of GLCDC module.

9.20.30.2 Detailed description

Implements

- clut.

Table 1586:Return values

Name	Description
SSP_SUCCESS	CLUT updated successfully.
SSP_ERR_ASSERTION	Pointer to the control block or CLUT source data is NULL.
SSP_ERR_INVALID_CLUT_ACCESS	Illegal CLUT entry or size is specified.

NOTE: This API can be called any time.

9.20.30.3 Function steps

- Check the CLUT table current used
- Copy the new CLUT data on the source memory to the CLUT SRAM in the GLCD module
- Make the GLCD module read the new CLUT table data from the next frame

9.20.31 R_GLCD_StatusGet

```
ssp_err_t R_GLCD_StatusGet ( display_ctrl_t  const *const p_api_ctrl ,
 display_status_t  *const p_status )
```

9.20.31.1 Brief description

Get status of GLCDC module.

9.20.31.2 Detailed description

Implements

- statusGet.

Table 1587:Return values

Name	Description
SSP_SUCCESS	Got status successfully.
SSP_ERR_ASSERTION	Pointer to the control block or the status structure is NULL.

NOTE: The GLCD hardware starts the fading processing at the first Vsync after the previous LayerChange() call is held. Due to this behavior of the hardware, this API may not return DISPLAY_FADE_STATUS_FADING_UNDERWAY as the fading status, if it is called before the first Vsync after LayerChange() is called. In this case, the API returns DISPLAY_FADE_STATUS_UNCERTAIN, instead of DISPLAY_FADE_STATUS_NOT_UNDERWAY.

9.20.31.3 Function steps

- Return the GLCD HAL driver state
- Return the fading status for the layers

9.20.32 R_GLCD_VersionGet

```
ssp_err_t R_GLCD_VersionGet ( ssp_version_t  * p_version )
```

9.20.32.1 Brief description

Get version of R_GLCDC module.

9.20.32.2 Detailed description

Implements

- versionGet.

Table 1588:Return values

Name	Description
p_version.	The version number.

NOTE: This function is re-entrant.

9.20.33 g_display_on_glcd

`display_api_t::g_display_on_glcd`

9.20.33.1 Detailed description

GLCD HAL module API function pointer list

9.20.33.2 Initialized as

```
g_display_on_glcd=
{
    .open        = R_GLCD_Open,
    .close       = R_GLCD_Close,
    .start       = R_GLCD_Start,
    .stop        = R_GLCD_Stop,
    .layerChange = R_GLCD_LayerChange,
    .clut        = R_GLCD_ClutUpdate,
    .correction  = R_GLCD_ColorCorrection,
    .statusGet   = R_GLCD_StatusGet,
    .versionGet  = R_GLCD_VersionGet
}
```

9.20.34 API Data

9.20.34.1 glcd_clk_src_t

`glcd_clk_src_t`

Detailed description

Clock source select

Enumerated values

Name	Description
GLCD_CLK_SRC_INTERNAL	Internal.
GLCD_CLK_SRC_EXTERNAL	External.

9.20.34.2 glcd_panel_clk_div_t

`glcd_panel_clk_div_t`

Detailed description

Clock frequency division ratio

Enumerated values

Name	Description
GLCD_PANEL_CLK_DIVISOR_1	Division Ratio 1/1.
GLCD_PANEL_CLK_DIVISOR_2	Division Ratio 1/2.
GLCD_PANEL_CLK_DIVISOR_3	Division Ratio 1/3.
GLCD_PANEL_CLK_DIVISOR_4	Division Ratio 1/4.
GLCD_PANEL_CLK_DIVISOR_5	Division Ratio 1/5.
GLCD_PANEL_CLK_DIVISOR_6	Division Ratio 1/6.
GLCD_PANEL_CLK_DIVISOR_7	Division Ratio 1/7.
GLCD_PANEL_CLK_DIVISOR_8	Division Ratio 1/8.
GLCD_PANEL_CLK_DIVISOR_9	Division Ratio 1/9.
GLCD_PANEL_CLK_DIVISOR_12	Division Ratio 1/12.
GLCD_PANEL_CLK_DIVISOR_16	Division Ratio 1/16.
GLCD_PANEL_CLK_DIVISOR_24	Division Ratio 1/24.
GLCD_PANEL_CLK_DIVISOR_32	Division Ratio 1/32.

9.20.34.3 glcd_tcon_pin_t

`glcd_tcon_pin_t`

Detailed description

LCD TCON output pin select

Enumerated values

Name	Description
GLCD_TCON_PIN_NONE	No output.
GLCD_TCON_PIN_0	LCD_TCON0.
GLCD_TCON_PIN_1	LCD_TCON1.
GLCD_TCON_PIN_2	LCD_TCON2.
GLCD_TCON_PIN_3	LCD_TCON3.
GLCD_TCON_PIN_NUM	

9.20.34.4 glcd_bus_arbitration_t

`glcd_bus_arbitration_t`

Detailed description

Bus Arbitration setting

Enumerated values

Name	Description
GLCD_BUS_ARBITRATION_ROUNDROBIN	Round robin.
GLCD_BUS_ARBITRATION_FIX_PRIORITY	Fixed.

9.20.34.5 glcd_correction_proc_order_t

`glcd_correction_proc_order_t`

Detailed description

Correction circuit sequence control

Enumerated values

Name	Description
GLCD_CORRECTION_PROC_ORDER_BRIGHTNESS_CONTRAST2GAMMA	Brightness -> contrast -> gamma correction.
GLCD_CORRECTION_PROC_ORDER_GAMMA2BRIGHTNESS_CONTRAST	Gamma correction -> brightness -> contrast.

9.20.34.6 glcd_tcon_signal_select_t

`glcd_tcon_signal_select_t`

Detailed description

Timing signals for driving the LCD panel

Enumerated values

Name	Description
GLCD_TCON_SIGNAL_SELECT_STVA_VS	STVA/VS.
GLCD_TCON_SIGNAL_SELECT_STVB_VE	STVB/VE.
GLCD_TCON_SIGNAL_SELECT_STHA_HS	STH/SP/HS.
GLCD_TCON_SIGNAL_SELECT_STHB_HE	STB/LP/HE.
GLCD_TCON_SIGNAL_SELECT_DE	DE.

9.20.34.7 glcd_clut_plane_t

`glcd_clut_plane_t`

Detailed description

Clock phase adjustment for serial RGB output

Enumerated values

Name	Description
GLCD_CLUT_PLANE_0	GLCD CLUT plane 0.
GLCD_CLUT_PLANE_1	GLCD CLUT plane 1.

9.20.34.8 glcd_dithering_mode_t

`glcd_dithering_mode_t`

Detailed description

Dithering mode

Enumerated values

Name	Description
GLCD_DITHERING_MODE_TRUNCATE	No dithering (truncate)
GLCD_DITHERING_MODE_ROUND_OFF	Dithering with round off.
GLCD_DITHERING_MODE_2X2PATTERN	Dithering with 2x2 pattern.
GLCD_DITHERING_MODE_SETTING_MAX	

9.20.34.9 glcd_dithering_pattern_t

`glcd_dithering_pattern_t`

Detailed description

Dithering mode

Enumerated values

Name	Description
GLCD_DITHERING_PATTERN_00	2x2 pattern '00'
GLCD_DITHERING_PATTERN_01	2x2 pattern '01'
GLCD_DITHERING_PATTERN_10	2x2 pattern '10'
GLCD_DITHERING_PATTERN_11	2x2 pattern '11'

9.20.34.10 glcd_input_interface_format_t

`glcd_input_interface_format_t`

Detailed description

Output interface format

Enumerated values

Name	Description
GLCD_INPUT_INTERFACE_FORMAT_RGB565	Input interface format RGB565.
GLCD_INPUT_INTERFACE_FORMAT_RGB888	Input interface format RGB888.
GLCD_INPUT_INTERFACE_FORMAT_ARGB1555	Input interface format ARGB1555.
GLCD_INPUT_INTERFACE_FORMAT_ARGB4444	Input interface format ARGB4444.
GLCD_INPUT_INTERFACE_FORMAT_ARGB8888	Input interface format ARGB8888.
GLCD_INPUT_INTERFACE_FORMAT_CLUT8	Input interface format CLUT8.
GLCD_INPUT_INTERFACE_FORMAT_CLUT4	Input interface format CLUT4.
GLCD_INPUT_INTERFACE_FORMAT_CLUT1	Input interface format CLUT1.

9.20.34.11 glcd_output_interface_format_t

`glcd_output_interface_format_t`

Detailed description

Output interface format

Enumerated values

Name	Description
GLCD_OUTPUT_INTERFACE_FORMAT_RGB888	Output interface format RGB888.
GLCD_OUTPUT_INTERFACE_FORMAT_RGB666	Output interface format RGB666.
GLCD_OUTPUT_INTERFACE_FORMAT_RGB565	Output interface format RGB565.
GLCD_OUTPUT_INTERFACE_FORMAT_SERIAL_RGB	Output interface format Serial RGB.

9.20.34.12 glcd_dithering_output_format_t

`glcd_dithering_output_format_t`

Detailed description

Dithering output format

Enumerated values

Name	Description
GLCD_DITHERING_OUTPUT_FORMAT_RGB888	Dithering output format RGB888.
GLCD_DITHERING_OUTPUT_FORMAT_RGB666	Dithering output format RGB666.
GLCD_DITHERING_OUTPUT_FORMAT_RGB565	Dithering output format RGB565.

9.20.34.13 display_plane_blend_t

`display_plane_blend_t`

Detailed description

RGB color order select

Enumerated values

Name	Description
DISPLAY_PLANE_BLEND_TRANSPARENT	Current graphics layer is transparent and the lower layer is.
DISPLAY_PLANE_BLEND_NON_TRANSPARENT	Current graphics layer is displayed.
DISPLAY_PLANE_BLEND_ON_LOWER_LAYER	Current graphics layer is blended with the lower layer.

9.20.34.14 glcd_fading_control_initial_alpha_t

`glcd_fading_control_initial_alpha_t`

Detailed description

RGB color order select

Enumerated values

Name	Description
GLCD_FADING_CONTROL_INITIAL_ALPHA_MIN	Initial alpha value setting for a graphics plane is zero.
GLCD_FADING_CONTROL_INITIAL_ALPHA_MAX	Initial alpha value setting for a graphics plane is maximum.

9.20.35 Extensions

9.20.35.1 glcd_instance_ctrl_t

glcd_instance_ctrl_t

Detailed description

Display control block. DO NOT INITIALIZE.

Variables

- display_state_t state

  Status of GLCD module.
- void(* p_callback)(*p_args)

  Pointer to callback function.
- void const * p_context

  Pointer to the higher level device context.
- R_GLCDC_Type * p_reg

  Base register address.

9.20.35.2 glcd_cfg_t

glcd_cfg_t

Detailed description

GLCD hardware specific configuration

Variables

- glcd_tcon_pin_t tcon_hsync

  GLCD TCON output pin select.
- glcd_tcon_pin_t tcon_vsync

  GLCD TCON output pin select.
- glcd_tcon_pin_t tcon_de

  GLCD TCON output pin select.
- glcd_correction_proc_order_t correction_proc_order

  Correction control route select.
- glcd_clk_src_t clksrc

  Clock Source selection.
- glcd_panel_clk_div_t clock_div_ratio

  Clock divide ratio for dot clock.

- glcd_dithering_mode_t dithering_mode

  Dithering mode.
- glcd_dithering_pattern_t dithering_pattern_A

  Dithering pattern A.
- glcd_dithering_pattern_t dithering_pattern_B

  Dithering pattern B.
- glcd_dithering_pattern_t dithering_pattern_C

  Dithering pattern C.
- glcd_dithering_pattern_t dithering_pattern_D

  Dithering pattern D.

9.20.35.3 glcd_ctrl_t

glcd_ctrl_t

Detailed description

GLCD hardware specific control block

Variables

- display_coordinate_t back_porch

  Zero coordinate for graphics plane(Bach porch End)
- uint16_t hsize

  Horizontal pixel size in a line.
- uint16_t vsize

  Vertical pixel size in a frame.
- bsp_lock_t resource_lock

  Resource lock.
- void * p_context

  Pointer to the function level device context (e.g. display_ctrl_t type data)

9.20.35.4 tcon_func_t

tcon_func_t

Detailed description

Function pointers to control TCON pin settings

Variables

- void(* tcon_select)(R_GLCDC_Type *p_glcd_reg,)
- void(* invert)(R_GLCDC_Type *p_glcd_reg)

9.20.35.5 recalculated_param_t

recalculated_param_t

Detailed description

The structure for the layer parameter recalculation

Variables

- uint16_t hpix_size
- uint16_t vpix_size
- int16_t hpix_offset
- int16_t vpix_offset
- uint32_t hread_size
- uint32_t base_address

9.21 GPT

Driver for the General PWM Timer (GPT).

9.21.1 Summary

Extends Timer Interface.

This module implements the Timer Interface using the General PWM Timer (GPT) peripherals GPT32EH, GPT32E, GPT32. It also provides an output compare extension to output the timer signal to the GTIOC pin.

9.21.2 Functions

- R_GPT_TimerOpen
- R_GPT_Stop
- R_GPT_Start
- R_GPT_CounterGet
- R_GPT_Reset
- R_GPT_PeriodSet
- R_GPT_DutyCycleSet
- R_GPT_InfoGet
- R_GPT_Close
- R_GPT_VersionGet

9.21.3 Defines

- #define GPT_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define GPT_CODE_VERSION_MINOR

  Initial value: `(7U)`

9.21.4 R_GPT_TimerOpen

```
ssp_err_t R_GPT_TimerOpen ( timer_ctrl_t *const p_api_ctrl , timer_cfg_t const
*const p_cfg )
```

9.21.4.1 Brief description

Powers on GPT, handles required initialization described in hardware manual. Implements open.

9.21.4.2 Detailed description

The Open function configures a single GPT channel, starts the channel, and provides a handle for use with the GPT API Control and Close functions. This function must be called once prior to calling any other GPT API functions. After a channel is opened, the Open function should not be called again for the same channel without first calling the associated Close function.

GPT hardware does not support one-shot functionality natively. When using one-shot mode, the timer will be stopped in an ISR after the requested period has elapsed.

The GPT implementation of the general timer can accept a timer_on_gpt_cfg_t extension parameter.

Table 1589:Return values

Name	Description
SSP_SUCCESS	Initialization was successful and timer has started.
SSP_ERR_ASSERTION	One of the following parameters is incorrect. Either • p_cfg is NULL, OR • p_ctrl is NULL, OR

Table 1589:Return values (Continued)

Name	Description
SSP_ERR_INVALID_ARGUMENT	One of the following parameters is invalid: • p_cfg->period: must be in the following range: – Lower bound: (1 / (PCLK frequency) – Upper bound: (0xFFFFFFFF * 1024 / (PCLK frequency)) • p_cfg->p_callback not NULL, but ISR is not enabled. ISR must be enabled to use callback function. Enable channel's overflow ISR in bsp_irq_cfg.h.
SSP_ERR_IN_USE	The channel specified has already been opened. No configurations were changed. Call the associated Close function or use associated Control commands to reconfigure the channel.
SSP_ERR_IRQ_BSP_DISABLED	- p_cfg->mode is TIMER_MODE_ONE_SHOT, but ISR is not enabled. ISR must be enabled to use one-shot mode.
SSP_ERR_IP_CHANNEL_NOT_PRESENT	- The channel requested in the p_cfg parameter is not available on this device.

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

9.21.4.3 Function steps

- Calculate period and store internal variables
- Calculate duty cycle
- Verify channel is not already used
- Power on GPT before setting any hardware registers. Make sure the counter is stopped before setting mode register, PCLK divisor register, and counter register.

9.21.5 R_GPT_Stop

```
ssp_err_t R_GPT_Stop ( timer_ctrl_t  *const p_api_ctrl )
```

9.21.5.1 Brief description

Stops timer. Implements stop.

9.21.5.2 Detailed description

Table 1590:Return values

Name	Description
SSP_SUCCESS	Timer successfully stopped.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.21.5.3 Function steps

- Stop timer

9.21.6 R_GPT_Start

```
ssp_err_t R_GPT_Start ( timer_ctrl_t  *const p_api_ctrl )
```

9.21.6.1 Brief description

Starts timer. Implements start.

9.21.6.2 Detailed description

Table 1591:Return values

Name	Description
SSP_SUCCESS	Timer successfully started.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.21.6.3 Function steps

- Start timer

9.21.7 R_GPT_CounterGet

```
ssp_err_t R_GPT_CounterGet ( timer_ctrl_t  *const p_api_ctrl ,  timer_size_t
*const p_value )
```

9.21.7.1 Brief description

Sets counter value in provided p_value pointer. Implements counterGet.

9.21.7.2 Detailed description

Table 1592:Return values

Name	Description
SSP_SUCCESS	Counter value read, p_value is valid.
SSP_ERR_ASSERTION	The p_ctrl or p_value parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.21.7.3 Function steps

- Read counter value

9.21.8 R_GPT_Reset

```
ssp_err_t R_GPT_Reset ( timer_ctrl_t  *const p_api_ctrl )
```

9.21.8.1 Brief description

Resets the counter value to 0. Implements reset.

9.21.8.2 Detailed description

Table 1593:Return values

Name	Description
SSP_SUCCESS	Counter value written successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.

Table 1593:Return values (Continued)

Name	Description
SSP_ERR_NOT_OPEN	The channel is not opened.

9.21.8.3 Function steps

- Write the counter value

9.21.9 R_GPT_PeriodSet

```
ssp_err_t R_GPT_PeriodSet ( timer_ctrl_t  *const p_api_ctrl ,  timer_size_t
const period ,  timer_unit_t  const unit )
```

9.21.9.1 Brief description

Sets period value provided. Implements periodSet.

9.21.9.2 Detailed description

Table 1594:Return values

Name	Description
SSP_SUCCESS	Period value written successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_INVALID_ARGUMENT	One of the following is invalid: • p_period->unit: must be one of the options from timer_unit_t • p_period->value: must result in a period in the following range: – Lower bound: (1 / (PCLK frequency)) – Upper bound: (0xFFFFFFFF * 1024 / (PCLK frequency))
SSP_ERR_NOT_OPEN	The channel is not opened.

9.21.9.3 Function steps

- Delay must be converted to PCLK counts before it can be set in registers

- Make sure period is valid.
- Store current status, then stop timer before setting divisor register
- Reset counter in case new cycle is less than current count value, then restore state (counting or stopped).

9.21.10 R_GPT_DutyCycleSet

```
ssp_err_t R_GPT_DutyCycleSet ( timer_ctrl_t  *const p_api_ctrl ,  timer_size_t
const duty_cycle ,  timer_pwm_unit_t  const unit ,   uint8_t const pin )
```

9.21.10.1 Brief description

Sets status in provided p_status pointer. Implements pwm_api_t::dutyCycleSet.

9.21.10.2 Detailed description

Table 1595:Return values

Name	Description
SSP_SUCCESS	Counter value written successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.21.10.3 Function steps

- Converted duty cycle to PCLK counts before it can be set in registers
- Set duty cycle.

9.21.11 R_GPT_InfoGet

```
ssp_err_t R_GPT_InfoGet ( timer_ctrl_t  *const p_api_ctrl ,  timer_info_t
*const p_info )
```

9.21.11.1 Brief description

Get timer information and store it in provided pointer p_info. Implements infoGet.

9.21.11.2 Detailed description

Table 1596:Return values

Name	Description
SSP_SUCCESS	Period, count direction, frequency, and status value written to caller's structure successfully.
SSP_ERR_ASSERTION	The p_ctrl or p_info parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.21.11.3 Function steps

- Get and store period
- Get and store clock frequency
- Get and store clock counting direction

9.21.12 R_GPT_Close

```
ssp_err_t R_GPT_Close ( timer_ctrl_t  *const p_api_ctrl )
```

9.21.12.1 Brief description

Stops counter, disables interrupts, disables output pins, and clears internal driver data.

9.21.12.2 Detailed description

Table 1597:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.21.12.3 Function steps

- Cleanup. Disable interrupts, stop counter, and disable output.

- Unlock channel
- Clear stored internal driver data

9.21.13 R_GPT_VersionGet

```
ssp_err_t R_GPT_VersionGet ( ssp_version_t  *const p_version )
```

9.21.13.1 Brief description

Sets driver version based on compile time macros.

9.21.13.2 Detailed description

Table 1598:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

9.21.14 API Data

9.21.14.1 gpt_pin_level_t

`gpt_pin_level_t`

Detailed description

Level of GPT pin

Enumerated values

Name	Description
GPT_PIN_LEVEL_LOW	Pin level low.
GPT_PIN_LEVEL_HIGH	Pin level high.
GPT_PIN_LEVEL_RETAINED	Pin level retained.

9.21.14.2 gpt_trigger_t

`gpt_trigger_t`

Detailed description

Sources can be used to start the timer, stop the timer, count up, or count down.

Enumerated values

Name	Description
GPT_TRIGGER_NONE	No action performed.
GPT_TRIGGER_GTIOCA_RISING_WHILE_GTIOCB_LOW	Action performed when GTIOCA input rises while GTIOCB is low.
GPT_TRIGGER_GTIOCA_RISING_WHILE_GTIOCB_HIGH	Action performed when GTIOCA input rises while GTIOCB is high.
GPT_TRIGGER_GTIOCA_FALLING_WHILE_GTIOCB_LOW	Action performed when GTIOCA input falls while GTIOCB is low.
GPT_TRIGGER_GTIOCA_FALLING_WHILE_GTIOCB_HIGH	Action performed when GTIOCA input falls while GTIOCB is high.
GPT_TRIGGER_GTIOCB_RISING_WHILE_GTIOCA_LOW	Action performed when GTIOCB input rises while GTIOCA is low.
GPT_TRIGGER_GTIOCB_RISING_WHILE_GTIOCA_HIGH	Action performed when GTIOCB input rises while GTIOCA is high.
GPT_TRIGGER_GTIOCB_FALLING_WHILE_GTIOCA_LOW	Action performed when GTIOCB input falls while GTIOCA is low.
GPT_TRIGGER_GTIOCB_FALLING_WHILE_GTIOCA_HIGH	Action performed when GTIOCB input falls while GTIOCA is high.
GPT_TRIGGER_SOURCE_REGISTER_ENABLE	Enables settings in the Source Select Register.

9.21.14.3 gpt_output_t

```
gpt_output_t
```

Detailed description

Output level used when selecting what happens at compare match or cycle end.

Enumerated values

Name	Description
GPT_OUTPUT_RETAINED	Output retained.

Name	Description
GPT_OUTPUT_LOW	Output low.
GPT_OUTPUT_HIGH	Output high.
GPT_OUTPUT_TOGGLED	Output toggled.

9.21.15 Extensions

9.21.15.1 gpt_instance_ctrl_t

gpt_instance_ctrl_t

Detailed description

Channel control block. DO NOT INITIALIZE. Initialization occurs when open is called.

Variables

- void(* p_callback)(*p_args)

  Callback provided when a timer ISR occurs. NULL indicates no CPU interrupt.
- void const * p_context

  Placeholder for user data. Passed to the user callback in timer_callback_args_t.
- void * p_reg

  Base register for this channel.
- uint32_t open

  Whether or not channel is open.
- uint8_t channel

  Channel number.
- bool one_shot

  Whether or not timer is in one shot mode.
- IRQn_Type irq

  Counter overflow IRQ number.
- timer_variant_t variant

  Timer variant.

9.21.15.2 gpt_output_pin_t

gpt_output_pin_t

Detailed description

Configurations for output pins.

Variables

- bool output_enabled

  Set to true to enable output, false to disable output.

- gpt_pin_level_t stop_level

  Select a stop level from gpt_pin_level_t.

9.21.15.3 timer_on_gpt_cfg_t

timer_on_gpt_cfg_t

Detailed description

GPT extension configures the output pins for GPT.

Variables

- gpt_output_pin_t gtioca

  Configuration for GPT I/O pin A.

- gpt_output_pin_t gtiocb

  Configuration for GPT I/O pin B.

9.22 GPT Input Capture

Driver for the General PWM Timer (GPT) with Input Capture.

9.22.1 Summary

Extends Input Capture Interface.

This module implements the Input Capture Interface for the General PWM Timer (GPT) peripherals GPT32EH, GPT32E, GPT32.

9.22.2 Functions

- R_GPT_InputCaptureOpen
- R_GPT_InputCaptureClose
- R_GPT_InputCaptureVersionGet
- R_GPT_InputCaptureDisable
- R_GPT_InputCaptureEnable
- R_GPT_InputCaptureInfoGet
- R_GPT_InputCaptureLastCaptureGet

9.22.3 Defines

- #define GPT_INPUT_CAPTURE_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Includes
- #define GPT_INPUT_CAPTURE_CODE_VERSION_MINOR

  Initial value: `(7U)`
- #define GPT_INPUT_CAPTURE_MAX_COUNT

  Initial value: `(0xFFFFFFFFUL)`

  Maximum value of GPT counter.

9.22.4 R_GPT_InputCaptureOpen

```
ssp_err_t R_GPT_InputCaptureOpen ( input_capture_ctrl_t  *const p_api_ctrl ,
 input_capture_cfg_t  const *const p_cfg )
```

9.22.4.1 Brief description

Open a GPT Timer for Input Capture. Implements open.

9.22.4.2 Detailed description

The Open function configures a single GPT channel for input capture and provides a handle for use with the other Input Capture API functions. This function must be called once prior to calling any other Input Capture API function. After a channel is opened, the Open function should not be called again for the same channel without first calling the associated Close function.

Table 1599:Return values

Name	Description
SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the parameters is NULL: p_cfg, p_ctrl, p_extend. Or the channel requested in the p_cfg parameter may not be available on the device selected in r_bsp_cfg.h. Or p_cfg->mode is invalid.
SSP_ERR_IN_USE	The channel specified has already been opened. No configurations were changed. Call the associated Close function or use associated Control commands to reconfigure the channel.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet
- eventInfoGet

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

9.22.4.3 Function steps

- Verify channel is not already used

9.22.5 R_GPT_InputCaptureClose

```
ssp_err_t R_GPT_InputCaptureClose ( input_capture_ctrl_t  *const p_api_ctrl )
```

9.22.5.1 Brief description

Close a GPT Timer Channel for Input Capture. Implements close.

9.22.5.2 Detailed description

Clears Timer settings, disables interrupts, and clears internal driver data.

Table 1600:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.22.5.3 Function steps

- Cleanup. Disable interrupts and stop measurements.
- Unlock channel
- Clear stored internal driver data

9.22.6 R_GPT_InputCaptureVersionGet

```
ssp_err_t R_GPT_InputCaptureVersionGet ( ssp_version_t  *const p_version )
```

9.22.6.1 Brief description

Gets driver version based on compile time macros. Implements versionGet.

9.22.6.2 Detailed description

Table 1601:Return values

Name	Description
SSP_SUCCESS	Success.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

9.22.7 R_GPT_InputCaptureDisable

```
ssp_err_t R_GPT_InputCaptureDisable ( input_capture_ctrl_t  const
*const p_api_ctrl )
```

9.22.7.1 Brief description

Disables GPT Input Capture RegA interrupt for specified channel at NVIC. Implements disable.

9.22.7.2 Detailed description

Table 1602:Return values

Name	Description
SSP_SUCCESS	Interrupt disabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.22.7.3 Function steps

- Disable interrupts
- Clearing the input capture source select registers.

9.22.8 R_GPT_InputCaptureEnable

```
ssp_err_t R_GPT_InputCaptureEnable ( input_capture_ctrl_t  const
*const p_api_ctrl )
```

9.22.8.1 Brief description

Enables GPT Input Capture RegA interrupt for specified channel at NVIC. Implements enable.

9.22.8.2 Detailed description

Table 1603:Return values

Name	Description
SSP_SUCCESS	Interrupt enabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.22.8.3 Function steps

- Configuring the input capture source select registers.
- Enabling the overflow and capture registers.

9.22.9 R_GPT_InputCaptureInfoGet

```
ssp_err_t R_GPT_InputCaptureInfoGet ( input_capture_ctrl_t  const
*const p_api_ctrl ,  input_capture_info_t  *const p_info )
```

9.22.9.1 Brief description

Gets status into provided p_info pointer. Implements infoGet.

9.22.9.2 Detailed description

Table 1604:Return values

Name	Description
SSP_SUCCESS	Success.

Table 1604:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.22.10 R_GPT_InputCaptureLastCaptureGet

```
ssp_err_t R_GPT_InputCaptureLastCaptureGet ( input_capture_ctrl_t  const
*const p_api_ctrl ,  input_capture_capture_t  *const p_capture )
```

9.22.10.1 Brief description

Update the last captured value and overflow count, in provided p_capture pointer. Implements lastCaptureGet.

9.22.10.2 Detailed description

Table 1605:Return values

Name	Description
SSP_SUCCESS	Period value written successfully.
SSP_ERR_ASSERTION	The p_ctrl or p_value parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.22.10.3 Function steps

- Set capture value

9.22.11 API Data

9.22.11.1 gpt_input_capture_signal_t

`gpt_input_capture_signal_t`

Detailed description

Input capture signal selection

Enumerated values

Name	Description
GPT_INPUT_CAPTURE_SIGNAL_PIN_GTIOCA	GTIOCxA pin, where x is channel number.
GPT_INPUT_CAPTURE_SIGNAL_PIN_GTIOCB	GTIOCxB pin, where x is channel number.

9.22.11.2 gpt_input_capture_signal_filter_t

`gpt_input_capture_signal_filter_t`

Detailed description

Input capture signal noise filter (debounce) setting. Only available for input signals GTIOCxA and GTIOCxB. The noise filter samples the external signal at intervals of the PCLK divided by one of the values. When 3 consecutive samples are at the same level (high or low), then that level is passed on as the observed state of the signal. See "Noise Filter Function" in the hardware manual, GPT section.

Enumerated values

Name	Description
GPT_INPUT_CAPTURE_SIGNAL_FILTER_NONE	NONE - no filtering.
GPT_INPUT_CAPTURE_SIGNAL_FILTER_1	PCLK/1 - fast sampling.
GPT_INPUT_CAPTURE_SIGNAL_FILTER_4	PCLK/4.
GPT_INPUT_CAPTURE_SIGNAL_FILTER_16	PCLK/16.
GPT_INPUT_CAPTURE_SIGNAL_FILTER_64	PCLK/64 - slow sampling.

9.22.11.3 gpt_input_capture_clock_divider_t

`gpt_input_capture_clock_divider_t`

Detailed description

Input capture PCLK divider. Used to scale the timer counter.

Enumerated values

Name	Description
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_1	/ 1
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_4	/ 4
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_16	/ 16

Name	Description
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_64	/ 64
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_256	/ 256
GPT_INPUT_CAPTURE_CLOCK_DIVIDER_1024	/ 1024

9.22.12 Extensions

9.22.12.1 gpt_input_capture_extend_t

gpt_input_capture_extend_t

Brief description

Extension configuration struct for TU Input Capture.

Detailed description

Pointed to by p_extend

Variables

- gpt_input_capture_signal_t signal

  One of gpt_input_capture_signal_t.
- gpt_input_capture_signal_filter_t signal_filter

  One of gpt_input_capture_signal_filter_t.
- gpt_input_capture_clock_divider_t clock_divider

  One of gpt_input_capture_clock_divider_t.
- input_capture_signal_level_t enable_level

  The unused GTIOCx pin can be used as an enable signal to enable captures. If the Input Capture Signal Pin is GTIOCA, then the enable pin is GTIOCB. The enable level is set here if used.
- gpt_input_capture_signal_filter_t enable_filter

  One of gpt_input_capture_signal_filter_t.

9.22.12.2 gpt_input_capture_instance_ctrl_t

gpt_input_capture_instance_ctrl_t

Detailed description

Channel control block. DO NOT INITIALIZE. Initialization occurs when open is called.

Variables

- uint32_t open

  Whether or not channel is open.

- uint8_t channel

  The channel in use.
- input_capture_mode_t mode

  The mode of measurement being performed.
- input_capture_repetition_t repetition

  One-shot or periodic measurement.
- uint32_t capture_count

  The value of the timer captured at the time of interrupt.
- uint32_t overflows_last

  Overflow count that occurred during last measurement.
- uint32_t overflows_current

  Running count of overflows in current measurement.
- void(* p_callback)(*p_args)

  Pointer to user callback.
- void const * p_context

  Pointer to user's context data, to be passed to the callback function.
- void * p_reg

  GPT base register for this channel.
- IRQn_Type capture_irq

  Capture IRQ number.
- IRQn_Type overflow_irq

  Overflow IRQ number.
- input_capture_variant_t variant

  Timer variant.
- uint32_t start_bitmask

  Start and Clear bitmask for input capture.
- uint32_t stop_bitmask

  Stop and capture bitmask for input capture.

9.23 ICU

Driver for the Interrupt Controller Unit (ICU) External pin interrupts function.

9.23.1 Summary

Extends External IRQ Interface.

This module implements the External IRQ Interface using the external input pins in the Interrupt Controller Unit (ICU).

9.23.2 Functions

- R_ICU_ExternalIrqOpen
- R_ICU_ExternalIrqEnable
- R_ICU_ExternalIrqDisable
- R_ICU_ExternalIrqTriggerSet
- R_ICU_ExternalIrqFilterEnable
- R_ICU_ExternalIrqFilterDisable
- R_ICU_ExternalIrqVersionGet
- R_ICU_ExternalIrqClose

9.23.3 Defines

- #define ICU_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define ICU_CODE_VERSION_MINOR

  Initial value: `(4U)`

9.23.4 R_ICU_ExternalIrqOpen

```
ssp_err_t R_ICU_ExternalIrqOpen ( external_irq_ctrl_t  *const p_api_ctrl ,
 external_irq_cfg_t  const *const p_cfg )
```

9.23.4.1 Brief description

Configure an external input pin for use with the button interface. Implements open.

9.23.4.2 Detailed description

The Open function is responsible for preparing an external input pin for operation. After completion of the Open function the external input pin shall be enabled and ready to service interrupts. This function must be called once prior to calling any other external input pin API functions. Once successfully completed, the status of the selected external input pin will be set to "open". After that this function should not be called again for the same external input pin without first performing a "close" by calling R_ICU_ExternalIrqClose.

Table 1606:Return values

Name	Description
SSP_SUCCESS	Open successful.
SSP_ERR_ASSERTION	One of the following is invalid: • p_ctrl or p_cfg is NULL • The channel requested in p_cfg is not available on the device selected in r_bsp_cfg.h.
SSP_ERR_INVALID_ARGUMENT	p_cfg->p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use callback function. Enable channel's overflow ISR in bsp_irq_cfg.h.
SSP_ERR_IN_USE	The channel specified has already been opened. No configurations were changed. Call the associated Close function to reconfigure the channel.
SSP_ERR_IP_CHANNEL_NOT_PRESENT	Requested channel does not exist on this device.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet
- eventInfoGet

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

9.23.5 R_ICU_ExternalIrqEnable

```
ssp_err_t R_ICU_ExternalIrqEnable ( external_irq_ctrl_t  *const p_api_ctrl )
```

9.23.5.1 Brief description

Enable external interrupt for specified channel at NVIC. Implements enable.

9.23.5.2 Detailed description

Table 1607:Return values

Name	Description
SSP_SUCCESS	Interrupt Enabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_INTERNAL	Requested IRQ is not defined in this system

9.23.6 R_ICU_ExternalIrqDisable

```
ssp_err_t R_ICU_ExternalIrqDisable ( external_irq_ctrl_t  *const p_api_ctrl )
```

9.23.6.1 Brief description

Disable external interrupt for specified channel at NVIC. Implements disable.

9.23.6.2 Detailed description

Table 1608:Return values

Name	Description
SSP_SUCCESS	Interrupt disabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.
SSP_ERR_INTERNAL	Requested IRQ is not defined in this system

9.23.7 R_ICU_ExternalIrqTriggerSet

```
ssp_err_t R_ICU_ExternalIrqTriggerSet ( external_irq_ctrl_t  *const p_api_ctrl ,
 external_irq_trigger_t  hw_trigger )
```

9.23.7.1 Brief description

Set trigger value provided. Implements triggerSet.

9.23.7.2 Detailed description

Table 1609:Return values

Name	Description
SSP_SUCCESS	Period value written successfully.
SSP_ERR_ASSERTION	The p_ctrl or p_period parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.23.8 R_ICU_ExternalIrqFilterEnable

```
ssp_err_t R_ICU_ExternalIrqFilterEnable ( external_irq_ctrl_t
*const p_api_ctrl )
```

9.23.8.1 Brief description

Enable external interrupt digital filter for specified channel. Implements filterEnable.

9.23.8.2 Detailed description

Table 1610:Return values

Name	Description
SSP_SUCCESS	External interrupt digital filter enabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.23.9 R_ICU_ExternalIrqFilterDisable

```
ssp_err_t R_ICU_ExternalIrqFilterDisable ( external_irq_ctrl_t
*const p_api_ctrl )
```

9.23.9.1 Brief description

Enable external interrupt digital filter for specified channel. Implements filterDisable.

9.23.9.2 Detailed description

Table 1611:Return values

Name	Description
SSP_SUCCESS	External interrupt digital filter disabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.23.10 R_ICU_ExternalIrqVersionGet

```
ssp_err_t R_ICU_ExternalIrqVersionGet ( ssp_version_t  *const p_version )
```

9.23.10.1 Brief description

Set driver version based on compile time macros. Implements versionGet.

9.23.10.2 Detailed description

Table 1612:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

9.23.11 R_ICU_ExternalIrqClose

```
ssp_err_t R_ICU_ExternalIrqClose ( external_irq_ctrl_t  *const p_api_ctrl )
```

9.23.11.1 Brief description

Disable external interrupt. Implements close.

9.23.11.2 Detailed description

Table 1613:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.23.11.3 Function steps

- Cleanup. Disable interrupt
- Release BSP hardware lock

9.23.12 Extensions

9.23.12.1 icu_instance_ctrl_t

icu_instance_ctrl_t

Detailed description

Channel instance control block. DO NOT INITIALIZE. Initialization occurs in open.

Variables

- uint32_t open

  Used to determine if channel control block is in use.
- R_ICU_Type * p_reg

  Pointer to register base address.
- void(* p_callback)(*p_args)

  Callback provided when a external IRQ ISR occurs. Set to NULL for no CPU interrupt.
- void const * p_context

  Placeholder for user data. Passed to the user callback in external_irq_callback_args_t.
- IRQn_Type irq

  NVIC interrupt number.
- uint8_t channel

  Channel.

9.24 IOPORT

Driver for the I/O Ports.

The IOPort HAL drivers provide the ability to access the I/O Ports of a device at both bit and port level. Port and pin direction can be changed. In addition a number of configuration APIs are provided to change the functionality of individual pins.

9.24.1 Functions

- R_IOPORT_Init
- R_IOPORT_PinsCfg
- R_IOPORT_PinCfg
- R_IOPORT_PinRead
- R_IOPORT_PortRead
- R_IOPORT_PortWrite
- R_IOPORT_PinWrite
- R_IOPORT_PortDirectionSet
- R_IOPORT_PinDirectionSet
- R_IOPORT_PortEventInputRead
- R_IOPORT_PinEventInputRead
- R_IOPORT_PortEventOutputWrite
- R_IOPORT_PinEventOutputWrite
- R_IOPORT_VersionGet
- R_IOPORT_EthernetModeCfg

9.24.2 Defines

- #define IOPORT_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define IOPORT_CODE_VERSION_MINOR

  Initial value: `(4U)`

9.24.3 R_IOPORT_Init

```
ssp_err_t R_IOPORT_Init ( ioport_cfg_t const * p_cfg )
```

9.24.3.1 Brief description

Initializes internal driver data, then calls R_IOPORT_PinsCfg to configure pins.

9.24.3.2 Detailed description

Table 1614:Return values

Name	Description
SSP_SUCCESS	Pin configuration data written to PFS register(s)
SSP_ERR_ASSERTION	NULL pointer

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet

9.24.4 R_IOPORT_PinsCfg

```
ssp_err_t R_IOPORT_PinsCfg ( ioport_cfg_t const * p_cfg )
```

9.24.4.1 Brief description

Configures the functions of multiple pins by loading configuration data into pin PFS registers. Implements pinsCfg.

9.24.4.2 Detailed description

This function initializes the supplied list of PmnPFS registers with the supplied values. This data can be generated by the ISDE pin configurator or manually by the developer. Different pin configurations can be loaded for different situations such as low power modes and test.*

Table 1615:Return values

Name	Description
SSP_SUCCESS	Pin configuration data written to PFS register(s)
SSP_ERR_ASSERTION	NULL pointer

9.24.5 R_IOPORT_PinCfg

```
ssp_err_t R_IOPORT_PinCfg ( ioport_port_pin_t  pin ,   uint32_t cfg )
```

9.24.5.1 Brief description

Configures the settings of a pin. Implements pinCfg.

9.24.5.2 Detailed description

Table 1616:Return values

Name	Description
SSP_SUCCESS	Pin configured.
SSP_ERR_INVALID_ARGUMENT	Invalid pin

NOTE: This function is re-entrant for different pins. This function will change the configuration of the pin with the new configuration. For example it is not possible with this function to change the drive strength of a pin while leaving all the other pin settings unchanged. To achieve this the original settings with the required change will need to be written using this function.

9.24.6 R_IOPORT_PinRead

```
ssp_err_t R_IOPORT_PinRead ( ioport_port_pin_t  pin ,  ioport_level_t
* p_pin_value )
```

9.24.6.1 Brief description

Reads the level on a pin. Implements pinRead.

9.24.6.2 Detailed description

Table 1617:Return values

Name	Description
SSP_SUCCESS	Pin read.
SSP_ERR_INVALID_ARGUMENT	Invalid argument
SSP_ERR_ASSERTION	NULL pointer

NOTE: This function is re-entrant for different pins.

9.24.7 R_IOPORT_PortRead

```
ssp_err_t R_IOPORT_PortRead ( ioport_port_t  port ,  ioport_size_t
* p_port_value )
```

9.24.7.1 Brief description

Reads the value on an IO port. Implements portRead.

9.24.7.2 Detailed description

The specified port will be read, and the levels for all the pins will be returned. Each bit in the returned value corresponds to a pin on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on. *

Table 1618:Return values

Name	Description
SSP_SUCCESS	Port read.
SSP_ERR_INVALID_ARGUMENT	Port not valid.
SSP_ERR_ASSERTION	NULL pointer

NOTE: This function is re-entrant for different ports.

9.24.8 R_IOPORT_PortWrite

```
ssp_err_t R_IOPORT_PortWrite ( ioport_port_t  port ,  ioport_size_t  value ,
 ioport_size_t  mask )
```

9.24.8.1 Brief description

Writes to multiple pins on a port. Implements portWrite.

9.24.8.2 Detailed description

The input value will be written to the specified port. Each bit in the value parameter corresponds to a bit on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on. Each bit in the mask parameter corresponds to a pin on the port.

Only the bits with the corresponding bit in the mask value set will be updated. e.g. value = 0xFFFF, mask = 0x0003 results in only bits 0 and 1 being updated.

Table 1619:Return values

Name	Description
SSP_SUCCESS	Port written to.
SSP_ERR_INVALID_ARGUMENT	The port and/or mask not valid.

NOTE: This function is re-entrant for different ports. This function makes use of the PCNTR3 register to atomically modify the levels on the specified pins on a port.

9.24.8.3 Function steps

- High bits
- Low bits

9.24.9 R_IOPORT_PinWrite

```
ssp_err_t R_IOPORT_PinWrite ( ioport_port_pin_t  pin ,  ioport_level_t  level )
```

9.24.9.1 Brief description

Sets a pin's output either high or low. Implements pinWrite.

9.24.9.2 Detailed description

Table 1620:Return values

Name	Description
SSP_SUCCESS	Pin written to.
SSP_ERR_INVALID_ARGUMENT	The pin and/or level not valid.

NOTE: This function is re-entrant for different pins. This function makes use of the PCNTR3 register to atomically modify the level on the specified pin on a port.

9.24.10 R_IOPORT_PortDirectionSet

```
ssp_err_t R_IOPORT_PortDirectionSet ( ioport_port_t  port ,
 ioport_size_t  direction_values ,  ioport_size_t  mask )
```

9.24.10.1 Brief description

Sets the direction of individual pins on a port. Implements portDirectionSet.

9.24.10.2 Detailed description

Multiple pins on a port can be set to inputs or outputs at once. Each bit in the mask parameter corresponds to a pin on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on. If a bit is set to 1 then the corresponding pin will be changed to an input or an output as specified by the direction values. If a mask bit is set to 0 then the direction of the pin will not be changed.

Table 1621:Return values

Name	Description
SSP_SUCCESS	Port direction updated.
SSP_ERR_INVALID_ARGUMENT	The port and/or mask not valid.

NOTE: This function is re-entrant for different ports.

9.24.10.3 Function steps

- High bits
- Low bits
- New value to write to port direction register

9.24.11 R_IOPORT_PinDirectionSet

```
ssp_err_t R_IOPORT_PinDirectionSet ( ioport_port_pin_t  pin ,
 ioport_direction_t  direction )
```

9.24.11.1 Brief description

Sets the direction of an individual pin on a port. Implements pinDirectionSet.

9.24.11.2 Detailed description

Table 1622:Return values

Name	Description
SSP_SUCCESS	Pin direction updated.
SSP_ERR_INVALID_ARGUMENT	The pin and/or direction not valid.

NOTE: This function is re-entrant for different pins.

9.24.12 R_IOPORT_PortEventInputRead

```
ssp_err_t R_IOPORT_PortEventInputRead ( ioport_port_t  port ,  ioport_size_t
* p_event_data )
```

9.24.12.1 Brief description

Reads the value of the event input data. Implements portEventInputRead.

9.24.12.2 Detailed description

The event input data for the port will be read. Each bit in the returned value corresponds to a pin on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on.

The port event data is captured in response to a trigger from the ELC. This function enables this data to be read. Using the event system allows the captured data to be stored when it occurs and then read back at a later time.

Table 1623:Return values

Name	Description
SSP_SUCCESS	Port read.
SSP_ERR_INVALID_ARGUMENT	Port not valid.
SSP_ERR_ASSERTION	NULL pointer

NOTE: This function is re-entrant for different ports.

9.24.13 R_IOPORT_PinEventInputRead

```
ssp_err_t R_IOPORT_PinEventInputRead ( ioport_port_pin_t  pin ,  ioport_level_t
* p_pin_event )
```

9.24.13.1 Brief description

Reads the value of the event input data of a specific pin. Implements pinEventInputRead.

9.24.13.2 Detailed description

The pin event data is captured in response to a trigger from the ELC. This function enables this data to be read. Using the event system allows the captured data to be stored when it occurs and then read back at a later time.

Table 1624:Return values

Name	Description
SSP_SUCCESS	Pin read.
SSP_ERR_INVALID_ARGUMENT	Pin not valid.
SSP_ERR_ASSERTION	NULL pointer

NOTE: This function is re-entrant.

9.24.14 R_IOPORT_PortEventOutputWrite

```
ssp_err_t R_IOPORT_PortEventOutputWrite ( ioport_port_t  port ,
 ioport_size_t  event_data ,  ioport_size_t  mask_value )
```

9.24.14.1 Brief description

This function writes the set and reset event output data for a port. Implements portEventOutputWrite.

9.24.14.2 Detailed description

Using the event system enables a port state to be stored by this function in advance of being output on the port. The output to the port will occur when the ELC event occurs.

The input value will be written to the specified port when an ELC event configured for that port occurs. Each bit in the value parameter corresponds to a bit on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on. Each bit in the mask parameter corresponds to a pin on the port.

Table 1625:Return values

Name	Description
SSP_SUCCESS	Port event data written.
SSP_ERR_INVALID_ARGUMENT	Port and/or mask not valid. •

NOTE: This function is re-entrant for different ports.

9.24.15 R_IOPORT_PinEventOutputWrite

```
ssp_err_t R_IOPORT_PinEventOutputWrite ( ioport_port_pin_t  pin ,
 ioport_level_t  pin_value )
```

9.24.15.1 Brief description

This function writes the event output data value to a pin. Implements pinEventOutputWrite.

9.24.15.2 Detailed description

Using the event system enables a pin state to be stored by this function in advance of being output on the pin. The output to the pin will occur when the ELC event occurs.

Table 1626:Return values

Name	Description
SSP_SUCCESS	Pin event data written.
SSP_ERR_INVALID_ARGUMENT	Pin or value not valid.

NOTE: This function is re-entrant for different ports.

9.24.16 R_IOPORT_VersionGet

```
ssp_err_t R_IOPORT_VersionGet ( ssp_version_t  * p_data )
```

9.24.16.1 Brief description

Returns IOPort HAL driver version. Implements versionGet.

9.24.16.2 Detailed description

Table 1627:Return values

Name	Description
SSP_SUCCESS	Version information read.
SSP_ERR_ASSERTION	The parameter p_data is NULL.

NOTE: This function is reentrant.

9.24.17 R_IOPORT_EthernetModeCfg

```
ssp_err_t R_IOPORT_EthernetModeCfg ( ioport_ethernet_channel_t  channel ,
 ioport_ethernet_mode_t  mode )
```

9.24.17.1 Brief description

Configures Ethernet channel PHY mode. Implements ioport_api_t::ethModeCfg.

9.24.17.2 Detailed description

Table 1628:Return values

Name	Description
SSP_SUCCESS	Ethernet PHY mode set.

Table 1628:Return values (Continued)

Name	Description
SSP_ERR_INVALID_ARGUMENT	Channel or mode not valid.
SSP_ERR_UNSUPPORTED	Ethernet configuration not supported on this device.

NOTE: This function is not re-entrant.

9.25 IWDT

Driver for the Independent Watchdog Timer (IWDT).

9.25.1 Summary

This module supports the Independent Watchdog Timer (IWDT). It implements the WDT Interface. Extends WDT_API HAL layer drivers for interfacing with the Independent Watchdog Timer (IWDT) peripheral.

The IWDT HAL APIs provide the ability to refresh the independent watchdog, read the timer value and read and clear status flags. When used in NMI output mode the callback to be called by the NMI ISR can be registered.

9.25.2 Functions

- R_IWDT_Open
- R_IWDT_CfgGet
- R_IWDT_Refresh
- R_IWDT_StatusGet
- R_IWDT_StatusClear
- R_IWDT_CounterGet
- R_IWDT_TimeoutGet
- R_IWDT_VersionGet

9.25.3 Defines

- #define IWDT_CODE_VERSION_MAJOR

  Initial value: `(1U)`

- #define IWDT_CODE_VERSION_MINOR

  Initial value: (4U)

9.25.4 R_IWDT_Open

```
ssp_err_t R_IWDT_Open ( wdt_ctrl_t  *const p_api_ctrl ,  wdt_cfg_t  const
*const p_cfg )
```

9.25.4.1 Brief description

Register the IWDT NMI callback.

9.25.4.2 Detailed description

Table 1629:Return values

Name	Description
SSP_SUCCESS	IWDT NMI callback successfully configured.
SSP_ERR_ASSERTION	Null Pointer.
SSP_ERR_INVALID_MODE	An attempt to open the IWDT when the OFS0 register is not configured for auto-start mode.
SSP_ERR_HW_LOCKED	IWDT module has already been called.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet

NOTE: This function is not reentrant.

9.25.4.3 Function steps

- g_iwdt_version is accessed by the ASSERT macro only and so compiler toolchain can issue a warning that it is not accessed. The code below eliminates this warning and also ensures these data structures are not optimized away.
- Eliminate toolchain warning when NMI output is not being used.
- Lock the IWDT Hardware Resource
- Initialize global pointer to WDT for NMI callback use.
- Check for NMI output mode

- NMI output mode
- Enable the IWDT underflow/refresh error interrupt (will generate an NMI).

9.25.5 R_IWDT_CfgGet

```
ssp_err_t R_IWDT_CfgGet ( wdt_ctrl_t  *const p_api_ctrl ,  wdt_cfg_t
*const p_cfg )
```

9.25.5.1 Brief description

Read the configuration of the IWDT. Implements cfgGet.

9.25.5.2 Detailed description

Table 1630:Return values

Name	Description
SSP_SUCCESS	IWDT configuration successfully read.
SSP_ERR_ASSERTION	Null Pointer.
SSP_ERR_INVALID_ARGUMENT	One or more configuration options is invalid.

NOTE: This function is reentrant.

9.25.5.3 Function steps

- Get timeout value from OFS0 register.

9.25.6 R_IWDT_Refresh

```
ssp_err_t R_IWDT_Refresh ( wdt_ctrl_t  *const p_api_ctrl )
```

9.25.6.1 Brief description

Refresh the Independent Watchdog Timer. Implements refresh.

9.25.6.2 Detailed description

Table 1631:Return values

Name	Description
SSP_SUCCESS	IWDT successfully refreshed.

NOTE: This function is reentrant. This function only returns SSP_SUCCESS. If the refresh fails due to being performed outside of the permitted refresh period the device will either reset or trigger an NMI ISR to run.

9.25.7 R_IWDT_StatusGet

```
ssp_err_t R_IWDT_StatusGet ( wdt_ctrl_t  *const p_api_ctrl ,  wdt_status_t
*const p_status )
```

9.25.7.1 Brief description

Read the IWDT status flags.

9.25.7.2 Detailed description

Indicates both status and error conditions.

Table 1632:Return values

Name	Description
SSP_SUCCESS	IWDT status successfully read.
SSP_ERR_ASSERTION	Null pointer as a parameter.

NOTE: This function is reentrant. When the IWDT is configured to output a reset on underflow or refresh error reading the status and error flags can be read after reset to establish if the IWDT caused the reset. Reading the status and error flags in NMI output mode indicates whether the IWDT generated the NMI interrupt.

9.25.8 R_IWDT_StatusClear

```
ssp_err_t R_IWDT_StatusClear ( wdt_ctrl_t  *const p_api_ctrl ,
 wdt_status_t const  status )
```

9.25.8.1 Brief description

Clear the IWDT status and error flags. Implements statusClear.

9.25.8.2 Detailed description

Table 1633:Return values

Name	Description
SSP_SUCCESS	IWDT flag(s) successfully cleared.
SSP_ERR_ASSERTION	Null pointer as a parameter.

NOTE: This function is reentrant.

9.25.8.3 Function steps

- Write zero to clear flags

9.25.9 R_IWDT_CounterGet

```
ssp_err_t R_IWDT_CounterGet ( wdt_ctrl_t  *const p_api_ctrl ,   uint32_t
*const p_count )
```

9.25.9.1 Brief description

Read the current count value of the IWDT. Implements counterGet.

9.25.9.2 Detailed description

Table 1634:Return values

Name	Description
SSP_SUCCESS	IWDT current count successfully read.
SSP_ERR_ASSERTION	Null pointer passed as a parameter.

NOTE: This function is reentrant.

9.25.10 R_IWDT_TimeoutGet

```
ssp_err_t R_IWDT_TimeoutGet ( wdt_ctrl_t  *const p_api_ctrl ,
 wdt_timeout_values_t  *const p_timeout )
```

9.25.10.1 Brief description

Read timeout information for the watchdog timer. Implements timeoutGet.

9.25.10.2 Detailed description

Table 1635:Return values

Name	Description
SSP_SUCCESS	WDT successfully refreshed.
SSP_ERR_ASSERTION	Null Pointer.
SSP_ERR_ABORTED	Invalid clock divider for this watchdog

NOTE: This function is reentrant. This function must not be called before calling R_WDT_Open.

9.25.11 R_IWDT_VersionGet

```
ssp_err_t R_IWDT_VersionGet ( ssp_version_t  *const p_data )
```

9.25.11.1 Brief description

Return IWDT HAL driver version. Implements versionGet.

9.25.11.2 Detailed description

Table 1636:Return values

Name	Description
SSP_SUCCESS	Call successful.
SSP_ERR_ASSERTION	Null pointer passed as a parameter.

NOTE: This function is reentrant.

9.25.12 Extensions

9.25.12.1 iwdt_instance_ctrl_t

iwdt_instance_ctrl_t

Detailed description

WDT control block. DO NOT INITIALIZE. Initialization occurs when open is called.

Variables

- bool wdt_open

  Indicates whether the open() API has been successfully called.
- void const * p_context

  Placeholder for user data. Passed to the user callback in wdt_callback_args_t.
- R_IWDT_Type * p_reg

  Pointer to register base address.
- void(* p_callback)(*p_args)

  Callback provided when a WDT NMI ISR occurs.

9.26 JPEG Decode

Driver for the JPEG Decoder.

9.26.1 Functions

- R_JPEG_Decode_Open
- R_JPEG_Decode_OutputBufferSet
- R_JPEG_Decode_LinesDecodedGet
- R_JPEG_Decode_InputBufferSet
- R_JPEG_Decode_ImageSubsampleSet
- R_JPEG_Decode_HorizontalStrideSet
- R_JPEG_Decode_Close
- R_JPEG_Decode_ImageSizeGet
- R_JPEG_Decode_StatusGet
- R_JPEG_Decode_PixelFormatGet
- R_JPEG_Decode_VersionGet

9.26.2 Defines

- #define JPEG_DECODE_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define JPEG_DECODE_CODE_VERSION_MINOR

  Initial value: `(6U)`

9.26.3 R_JPEG_Decode_Open

```
ssp_err_t R_JPEG_Decode_Open ( jpeg_decode_ctrl_t  *const p_api_ctrl ,
 jpeg_decode_cfg_t  const *const p_cfg )
```

9.26.3.1 Brief description

Initialize the JPEG Codec module. This function configures the JPEG Codec for decoding operation, sets up the registers for data format and pixel format based on user-supplied configuration parameters. Interrupts are enabled to support image size read operation and callback functions.

9.26.3.2 Detailed description

Table 1637:Return values

Name	Description
SSP_SUCCESS	JPEG Codec module is properly configured and is ready to take input data.
SSP_ERR_IN_USE	JPEG Codec is already in use.
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
SSP_ERR_HW_LOCKED	JPEG Codec resource is locked.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet
- eventInfoGet

9.26.3.3 Function steps

- Verify JPEG Codec is not already used.
- Update the common control parameter with the control and JEDI and JDTI callback handler for JPEG decode, the handlers will be called from r_jpeg_common, which implements JPEG JDTI and JEDI ISR for r_jpeg_decode and r_jpeg_encode driver.
- Record the configuration settings.
- Initialize horizontal stride value.
- Initialize output buffer size.
- Initialize total_lines_decoded
- Initialize horizontal sub-sample setting.
- Provide power to the JPEG module.
- Perform bus reset
- Reset the destination buffer address.
- Reset the source buffer address.
- Reset the horizontal stride.
- Configure the JPEG module for decode operation.
- Set image format for the decoded image.
- If the output pixel format is ARGB8888, also configure the alpha value.
- Set the alpha value for the decoded image.

- Set the output data format.
- The following interrupts are enabled: Interrupt on all errors Interrupt on Image Size
- Record user supplied callback routine.
- Set the driver status.
- All done. Return success.

9.26.4 R_JPEG_Decode_OutputBufferSet

```
ssp_err_t R_JPEG_Decode_OutputBufferSet ( jpeg_decode_ctrl_t  * p_api_ctrl ,
  void * p_output_buffer ,   uint32_t output_buffer_size )
```

9.26.4.1 Brief description

Assign output buffer to the JPEG Codec for storing output data.

9.26.4.2 Detailed description

NOTE: The number of image lines to be decoded depends on the size of the buffer and the horizontal stride settings. Once the output buffer size is known, the horizontal stride value is known, and the input pixel format is known (the input pixel format is obtained by the JPEG decoder from the JPEG headers), the driver automatically computes the number of lines that can be decoded into the output buffer. After these lines are decoded, the JPEG engine pauses and a callback function is triggered, so the application is able to provide the next buffer for the JPEG module to resume the operation.

The JPEG decoding operation automatically starts after both the input buffer and the output buffer are set, and the output buffer is big enough to hold at least eight lines of decoded image data.

Table 1638:Return values

Name	Description
SSP_SUCCESS	The output buffer is properly assigned to JPEG codec device.
SSP_ERR_ASSERTION	Pointer to the control block is NULL, or the pointer to the output_buffer. is NULL, or the output_buffer_size is 0.
SSP_ERR_INVALID_ALIGNMENT	Buffer starting address is not 8-byte aligned.
SSP_ERR_NOT_OPEN	JPEG not opened.
SSP_ERR_JPEG_BUFFERSIZE_NOT_ENOUGH	Invalid buffer size

9.26.4.3 Function steps

- Set the decoding destination address.

- Record the size of the output buffer.
- If the image size is not ready yet, the driver does not know the input pixel format. Without that information, the driver is unable to compute the number of lines of image to decode. In this case, the driver would record the output buffer size. Once all the information is ready, the driver would attempt to start the decoding process.
- For a given buffer size, compute number of lines to decode if the image size acquisition is known.
- If the driver status is IMAGE_SIZE_READY with no other flags, that means the driver just received IMAGE_SIZE. It has not started the decoding process yet.
- If Input buffer is set, output buffer is set, and horizontal stride is set, the driver is able to determine the number of lines to decode, and start the decoding operation.
- If the current status is OUTPUT_PAUSE, the driver needs to resume the operation.

9.26.5 R_JPEG_Decode_LinesDecodedGet

```
ssp_err_t R_JPEG_Decode_LinesDecodedGet ( jpeg_decode_ctrl_t  * p_api_ctrl ,
  uint32_t * p_lines )
```

9.26.5.1 Brief description

Returns the number of lines decoded into the output buffer.

9.26.5.2 Detailed description

NOTE: Use this function to retrieve number of image lines written to the output buffer after JPEG decoded a partial image. Combined with the horizontal stride settings and the output pixel format, the application can compute the amount of data to read from the output buffer.

Table 1639:Return values

Name	Description
SSP_SUCCESS	The output buffer is properly assigned to JPEG codec device.
SSP_ERR_ASSERTION	Pointer to the control block is NULL, or the pointer to the output_buffer. is NULL, or the output_buffer_size is 0.
SSP_ERR_NOT_OPEN	JPEG not opened.

9.26.6 R_JPEG_Decode_InputBufferSet

```
ssp_err_t R_JPEG_Decode_InputBufferSet ( jpeg_decode_ctrl_t  *const p_api_ctrl ,
  void * p_data_buffer ,   uint32_t data_buffer_size )
```

9.26.6.1 Brief description

Assign input data buffer to JPEG codec for processing.

9.26.6.2 Detailed description

NOTE: After the amount of data is processed, the JPEG driver triggers a callback function with the flag JPEG_OPERATION_INPUT_PAUSE set. The application supplies the next chunk of data to the driver so JPEG decoding can resume.

The JPEG decoding operation automatically starts after both the input buffer and the output buffer are set, and the output buffer is big enough to hold at least one line of decoded image data.

Table 1640:Return values

Name	Description
SSP_SUCCESS	The input data buffer is properly assigned to JPEG Codec device.
SSP_ERR_ASSERTION	Pointer to the control block is NULL, or the pointer to the input_buffer is NULL, or the input_buffer_size is 0.
SSP_ERR_INVALID_ALIGNMENT	Buffer starting address is not 8-byte aligned.
SSP_ERR_NOT_OPEN	JPEG not opened.

9.26.6.3 Function steps

- Configure the input buffer address.
- If the system is idle, start the JPEG engine. This allows the system to obtain image information (image size and input pixel format). This information is needed to drive the decode process later on.
- Based on buffer size, detect the in count mode setting. The driver is able to read input data in chunks. However the size of each chunk is limited to BUFFER_MAX_SIZE. Therefore, if the input data size is larger than BUFFER_MAX_SIZE, the driver assumes the entire input data is present, and can be decoded without additional input data. Otherwise, the driver enables input stream feature. This works even if the entire input size is smaller than BUFFER_MAX_SIZE.

9.26.7 R_JPEG_Decode_ImageSubsampleSet

```
ssp_err_t R_JPEG_Decode_ImageSubsampleSet ( jpeg_decode_ctrl_t
*const p_api_ctrl ,  jpeg_decode_subsample_t  horizontal_subsample ,
 jpeg_decode_subsample_t  vertical_subsample )
```

9.26.7.1 Brief description

Configure horizontal and vertical subsample.

9.26.7.2 Detailed description

NOTE: Use for scaling the decoded image.

Table 1641:Return values

Name	Description
SSP_SUCCESS	Horizontal Stride value is properly configured.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_ARGUMENT	Sub-sample setting is invalid.
SSP_ERR_NOT_OPEN	JPEG not opened.

9.26.7.3 Function steps

- Update horizontal sub-sample setting.

9.26.8 R_JPEG_Decode_HorizontalStrideSet

```
ssp_err_t R_JPEG_Decode_HorizontalStrideSet ( jpeg_decode_ctrl_t  * p_api_ctrl ,
  uint32_t horizontal_stride )
```

9.26.8.1 Brief description

Configure horizontal stride setting.

9.26.8.2 Detailed description

NOTE: Use when the horizontal stride needs to match the image width and the image size is unknown when opening the JPEG driver. (If the image size is known prior to the open call, pass the horizontal stride value in the jpef_cfg_t structure.) After the image size becomes available, use this function to update the horizontal stride value. If the driver must decode one line at a time, the horizontal stride can be set to zero.

Table 1642:Return values

Name	Description
SSP_SUCCESS	Horizontal Stride value is properly configured.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_INVALID_ALIGNMENT	Horizontal stride is zero or not 8-byte aligned.
SSP_ERR_NOT_OPEN	JPEG not opened.

9.26.8.3 Function steps

- Record the horizontal stride value in the control block
- Set the horizontal stride.
- If the parameters all are set, resume the core to decode.
- For the given buffer size, compute number of lines to decode.

9.26.9 R_JPEG_Decode_Close

```
ssp_err_t R_JPEG_Decode_Close ( jpeg_decode_ctrl_t  * p_api_ctrl )
```

9.26.9.1 Brief description

Cancel an outstanding JPEG codec operation and close the device.

9.26.9.2 Detailed description

Table 1643:Return values

Name	Description
SSP_SUCCESS	The input data buffer is properly assigned to JPEG Codec device.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_NOT_OPEN	JPEG not opened.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- eventInfoGet

9.26.9.3 Function steps

- Clear JPEG JINTE0 interrupt and JINTE1 interrupt.
- Disable JEDI and JDTI at NVIC
- Power off the JPEG codec.
- Reset the jpeg status flag in the driver.
- Unlock module at BSP level.

9.26.10 R_JPEG_Decode_ImageSizeGet

```
ssp_err_t R_JPEG_Decode_ImageSizeGet ( jpeg_decode_ctrl_t  * p_api_ctrl ,
  uint16_t * p_horizontal_size ,   uint16_t * p_vertical_size )
```

9.26.10.1 Brief description

Obtain the size of the image. This operation is valid during JPEG decoding operation.

9.26.10.2 Detailed description

Table 1644:Return values

Name	Description
SSP_SUCCESS	The image size is available and the horizontal and vertical values are stored in the memory pointed to by p_horizontal_size and p_vertical_size.

Table 1644:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_IMAGE_SIZE_UNKNOWN	The image size is unknown. More input data may be needed.
SSP_ERR_INVALID_MODE	JPEG Codec module is not decoding.
SSP_ERR_NOT_OPEN	JPEG is not opened.

9.26.11 R_JPEG_Decode_StatusGet

```
ssp_err_t R_JPEG_Decode_StatusGet ( jpeg_decode_ctrl_t  * p_api_ctrl ,
 jpeg_decode_status_t  * p_status )
```

9.26.11.1 Brief description

Get the status of the JPEG codec. This function can also be used to poll the device.

9.26.11.2 Detailed description

Table 1645:Return values

Name	Description
SSP_SUCCESS	The status information is successfully retrieved.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_NOT_OPEN	JPEG is not opened.

9.26.11.3 Function steps

- HW does not report error. Return internal status information.

9.26.12 R_JPEG_Decode_PixelFormatGet

```
ssp_err_t R_JPEG_Decode_PixelFormatGet ( jpeg_decode_ctrl_t  * p_api_ctrl ,
 jpeg_decode_color_space_t  * p_color_space )
```

9.26.12.1 Brief description

Get the input pixel format.

9.26.12.2 Detailed description

Table 1646:Return values

Name	Description
SSP_SUCCESS	The status information is successfully retrieved.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_NOT_OPEN	JPEG is not opened.

9.26.12.3 Function steps

- HW does not report error. Return internal status information.

9.26.13 R_JPEG_Decode_VersionGet

```
ssp_err_t R_JPEG_Decode_VersionGet ( ssp_version_t  * p_version )
```

9.26.13.1 Brief description

Get version of the display interface and GLCD HAL code.

9.26.13.2 Detailed description

Table 1647:Return values

Name	Description
SSP_SUCCESS	Version number
SSP_ERR_ASSERTION	The parameter p_version is NULL.

NOTE: This function is reentrant.

9.26.14 Extensions

9.26.14.1 jpeg_decode_instance_ctrl_t

jpeg_decode_instance_ctrl_t

Detailed description

JPEG Codec module control block. DO NOT INITIALIZE. Initialization occurs when jpep_api_t::open is called.

Variables

- jpeg_decode_status_t status
  JPEG Codec module status.
- ssp_err_t error_code
  JPEG Codec error code (if any).
- void(* p_callback)(*p_args)
  User-supplied callback functions.
- void const * p_extend
  JPEG Codec hardware dependent configuration */.
- void const * p_context
  Placeholder for user data. Passed to user callback in jpeg_decode_callback_args_t.
- R_JPEG_Type * p_reg
  Pointer to register base address.
- jpeg_decode_pixel_format_t pixel_format
  Pixel format.
- uint32_t horizontal_stride
  Horizontal Stride settings.
- uint32_t outbuffer_size
  out buffer size
- uint16_t total_lines_decoded
  Track the number of lines decoded so far.
- jpeg_decode_subsample_t horizontal_subsample
  Horizontal sub-sample setting.

9.27 JPEG Encode

Driver for the JPEG Encoder.

9.27.1 Functions

- R_JPEG_Encode_Open
- R_JPEG_Encode_OutputBufferSet
- R_JPEG_Encode_InputBufferSet
- R_JPEG_Encode_ImageParameterSet
- R_JPEG_Encode_StatusGet
- R_JPEG_Encode_Close
- R_JPEG_Encode_VersionGet

9.27.2 Defines

- #define JPEG_ENCODE_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Configuration for this module
- #define JPEG_ENCODE_CODE_VERSION_MINOR

  Initial value: `(0U)`

9.27.3 R_JPEG_Encode_Open

```
ssp_err_t R_JPEG_Encode_Open ( jpeg_encode_ctrl_t  *const p_api_ctrl ,
 jpeg_encode_cfg_t  const *const p_cfg )
```

9.27.3.1 Brief description

Initialize the JPEG Codec module. This function configures the JPEG Codec for encoding operation, sets up the registers for data format, pixel format, vertical and horizontal resolution stride based on user-supplied configuration parameters.

9.27.3.2 Detailed description

Table 1648:Return values

Name	Description
SSP_SUCCESS	JPEG Codec module is properly configured and is ready to take input data.
SSP_ERR_IN_USE	JPEG Codec is already in use.
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.

Table 1648:Return values (Continued)

Name	Description
SSP_ERR_HW_LOCKED	JPEG Codec resource is locked.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter is passed.
SSP_ERR_INVALID_ALIGNMENT	Horizontal stride is not 8-byte aligned.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet
- eventInfoGet

9.27.3.3 Function steps

- Verify JPEG Codec is not already used.
- Update the common control parameter with the control and JDEI and JDTI callback handler for JPEG encode, the handlers will be called from r_jpeg_common, which implements JPEG JDTI and JDEI ISR for r_jpeg_decode and r_jpeg_encode driver.
- Get the JDTI event information from FMI
- Get the vector table information of JDTI event
- Record the JPEG Encoder ctrl and internal ISR callback handler to JDTI event vector table
- Get the JEDI event information from FMI
- Get the vector table information of JEDI event
- Record the JPEG Encoder ctrl and internal ISR callback handler to JEDI event vector table
- Provide power to the JPEG module.
- Perform bus reset
- Reset the destination buffer address.
- Reset the source buffer address.
- Set the horizontal stride.
- Set the image horizontal and vertical size.
- Configure the JPEG module for encode operation.
- Set the output data format.
- Set input pixel format for JPEG Encoder NOTE : only ycbcr422 is valid for encoder
- Quantization table setting NOTE: Table 0(zero) is used for Luminance and Table 1(one) is used for Chrominance - Cr and Cb component
- Upload Luminance and Chrominance table to JPEG Codec
- Huffman table setting

- Upload the Huffman table to JPEG Codec
- Reset Marker setting
- Record image parameters to ctrl
- Set the driver status.
- Record the user context information
- Record user supplied callback routine.
- Enabled JPEG Compression data transfer complete interrupt and Count mode Interrupt
- Enable the JDTI and JDEI interrupts
- All done. Return success.

9.27.4 R_JPEG_Encode_OutputBufferSet

```
ssp_err_t R_JPEG_Encode_OutputBufferSet ( jpeg_encode_ctrl_t  * p_api_ctrl ,
  void * p_output_buffer )
```

9.27.4.1 Brief description

Assign output buffer to the JPEG Codec for storing output data.

9.27.4.2 Detailed description

NOTE: Buffer size should be sufficient to hold the encoded jpeg image.

Table 1649:Return values

Name	Description
SSP_SUCCESS	The output buffer is properly assigned to JPEG codec device.
SSP_ERR_ASSERTION	Pointer to the control block is NULL, or the pointer to the output_buffer. is NULL, or the output_buffer_size is 0.
SSP_ERR_INVALID_ALIGNMENT	Buffer starting address is not 8-byte aligned.
SSP_ERR_NOT_OPEN	JPEG not opened.
SSP_ERR_INVALID_CALL	An invalid call has been made, Codec output buffer address is attempted to changed during codec operation

9.27.4.3 Function steps

- Output buffer cannot be change during codec operation
- Set the encoding destination address.

9.27.5 R_JPEG_Encode_InputBufferSet

```
ssp_err_t R_JPEG_Encode_InputBufferSet ( jpeg_encode_ctrl_t  *const p_api_ctrl ,
  void * p_data_buffer ,   uint32_t data_buffer_size )
```

9.27.5.1 Brief description

Assign input data buffer to JPEG codec for processing.

9.27.5.2 Detailed description

NOTE: 1.After the amount of data is processed, the JPEG driver triggers a callback function with the flag JPEG_OPERATION_INPUT_PAUSE set. The application supplies the next chunk of data to the driver so JPEG encoding can resume. 2.Image size should be greater or equal to minimum coded unit (MCU) for YCbCr422 (only supported color space) the MCU is 8 lines by 16 pixel (where 1 pixel = 2 bytes) hence size can not be less than 8x16x2 = 256

The JPEG encoding operation automatically starts after setting the input buffer.

Table 1650:Return values

Name	Description
SSP_SUCCESS	The input data buffer is properly assigned to JPEG Codec device.
SSP_ERR_ASSERTION	Pointer to the control block is NULL, or the pointer to the p_data_buffer is NULL.
SSP_ERR_INVALID_ALIGNMENT	Buffer starting address or image line to encode or size of buffer is not 8-byte aligned.
SSP_ERR_NOT_OPEN	JPEG not opened.
SSP_ERR_INVALID_CALL	An invalid call has been made, set output buffer first
SSP_ERR_JPEG_IMAGE_SIZE_ERROR	Image size is not supported by JPEG Codec

9.27.5.3 Function steps

- Validate the the size : JPEG Codec can process minimum up to 8 lines by 16 pixel for YCbCr422 meaning 8x16x2 = 256

- Calculate the number of lines to be encode
- JPEG Codec required Lines to be byte aligned
- Check, If output image buffer is set or not
- Configure the input buffer address.
- if JPEG is just opened or completed one image, make DONE and IDLE status flag zero to encode next image
- Remove the Done and IDLE status flag
- Set the driver status to JPEG_ENCODE_STATUS_RUNNING.
- Start the encoder
- JPEG is PAUSE for next chunk of image
- Clear internal status information.
- Set RUNNING status
- Resume the count mode
- JPEG is running Notify the user, return error

9.27.6 R_JPEG_Encode_ImageParameterSet

```
ssp_err_t R_JPEG_Encode_ImageParameterSet ( jpeg_encode_ctrl_t
*const p_api_ctrl ,  jpeg_encode_raw_image_parameters  * p_image_parameters )
```

9.27.6.1 Brief description

Setup the image parameters to JPEG Codec device.

9.27.6.2 Detailed description

NOTE: Image parameters needs to be set before the setting the input buffer.DO NOT call this function during the JPEG Codec operation.

Table 1651:Return values

Name	Description
SSP_SUCCESS	Image parameter is properly assigned to JPEG Codec device.
SSP_ERR_ASSERTION	Pointer to the control block is NULL,
SSP_ERR_INVALID_ALIGNMENT	Horizontal stride is not 8-byte aligned.

Table 1651:Return values (Continued)

Name	Description
SSP_ERR_INVALID_ARGUMENT	Horizontal and Vertical resolution is invalid or zero.
SSP_ERR_NOT_OPEN	JPEG not opened.
SSP_ERR_INVALID_CALL	An invalid call has been made.

9.27.6.3 Function steps

- Do not change the JPEG Codec image setting while JPEG Codec is in progress
- Record the horizontal stride and vertical size value in the control block
- Set the horizontal stride.
- Set the image horizontal and vertical size.

9.27.7 R_JPEG_Encode_StatusGet

```
ssp_err_t R_JPEG_Encode_StatusGet ( jpeg_encode_ctrl_t  * p_api_ctrl ,
 jpeg_encode_status_t volatile * p_status )
```

9.27.7.1 Brief description

Get the status of the JPEG codec. This function can also be used to poll the device.

9.27.7.2 Detailed description

Table 1652:Return values

Name	Description
SSP_SUCCESS	The status information is successfully retrieved.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.

9.27.7.3 Function steps

- HW does not report error. Return internal status information.

9.27.8 R_JPEG_Encode_Close

```
ssp_err_t R_JPEG_Encode_Close ( jpeg_encode_ctrl_t  * p_api_ctrl )
```

9.27.8.1 Brief description

Cancel an outstanding JPEG codec operation and close the device.

9.27.8.2 Detailed description

Table 1653:Return values

Name	Description
SSP_SUCCESS	The input data buffer is properly assigned to JPEG Codec device.
SSP_ERR_ASSERTION	Pointer to the control block is NULL.
SSP_ERR_NOT_OPEN	JPEG not opened.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- eventInfoGet

9.27.8.3 Function steps

- Clear JPEG JINTE0 interrupt and JINTE1 interrupt.
- Disable JEDI and JDTI at NVIC
- Power off the JPEG codec.
- Reset the jpeg status flag in the driver.
- Unlock module at BSP level.

9.27.9 R_JPEG_Encode_VersionGet

```
ssp_err_t R_JPEG_Encode_VersionGet ( ssp_version_t  * p_version )
```

9.27.9.1 Brief description

Get version of the display interface and GLCD HAL code.

9.27.9.2 Detailed description

Table 1654:Return values

Name	Description
SSP_SUCCESS	Version number
SSP_ERR_ASSERTION	The parameter p_version is NULL.

NOTE: This function is reentrant.

9.27.10 Extensions

9.27.10.1 jpeg_encode_instance_ctrl_t

jpeg_encode_instance_ctrl_t

Detailed description

JPEG Codec module control block. DO NOT INITIALIZE. Initialization occurs when jpep_api_t::open is called.

Variables

- jpeg_encode_status_t status

  JPEG Codec module status.
- void(* p_callback)(*p_args)

  User-supplied callback functions.
- void const * p_extend

  JPEG Codec hardware dependent configuration */.
- void const * p_context

  Placeholder for user data. Passed to user callback in jpeg_encode_callback_args_t.
- R_JPEG_Type * p_reg

  Pointer to register base address.
- uint32_t horizontal_stride

  Horizontal Stride settings (Line offset).
- uint32_t output_buffer_size

  out buffer size

- uint16_t lines_to_encoded

  Number of lines to encode.

- uint16_t vertical_resolution

  vertical size

9.28 Key Interrupts

Driver for the Key Interrupt Function.

The Key input driver can be used for one to eight channels or in a matrix format. This module implements the following interface: Key Matrix Interface.

9.28.1 Functions

- R_KINT_KEYMATRIX_Open
- R_KINT_KEYMATRIX_Close
- R_KINT_KEYMATRIX_Enable
- R_KINT_KEYMATRIX_Disable
- R_KINT_KEYMATRIX_TriggerSet
- R_KINT_VersionGet

9.28.2 Defines

- #define KINT_CODE_VERSION_MAJOR

  Initial value: `(1U)`

- #define KINT_CODE_VERSION_MINOR

  Initial value: `(4U)`

9.28.3 R_KINT_KEYMATRIX_Open

```
ssp_err_t R_KINT_KEYMATRIX_Open ( keymatrix_ctrl_t  *const p_api_ctrl ,
 keymatrix_cfg_t  const *const p_cfg )
```

9.28.3.1 Brief description

Power on KINT, handle required initialization described in hardware manual. Implements open.

9.28.3.2 Detailed description

The Open function configures all the Key Input (KINT) channels and provides a handle for use with the rest of the KINT API functions. This function must be called at least once prior to calling any other KINT API functions. After the peripheral is initialized, the Open function should not be called again without first calling the Close function.

Table 1655:Return values

Name	Description
SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the following parameters may be NULL: p_cfg, or p_ctrl or the callback.
SSP_ERR_INVALID_ARGUMENT	One of the following may be invalid: • p_cfg->channel: must be between 0 and KINT_MAX_NUM_CHANNELS • p_cfg->trigger not a valid value.
SSP_ERR_HW_LOCKED	The API has already been opened. It must be closed before it can be opened again.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet
- eventInfoGet

NOTE: This function is not reentrant.

9.28.3.3 Function steps

- Check to see that the arguments passed are not null pointers
- Grab the hardware lock. If successful this indicates that the open was not previously called.
- Disable interrupts in the KINT peripheral
- Clear any pending interrupt requests in the KINT peripheral
- Clear the Interrupt Request in the ICU
- Configure the trigger edge. The trigger edge can be modified in the TriggerSet function later if desired
- KEYMATRIX_TRIG_RISING condition
- Configure the usage of key interrupt flags
- Store the callback and context information

- If interrupts are to be enabled now, set it up for the selected channels. The channels can be changed later in the enable function but to modify the callback and context, the API has to be closed and reopened with the new callback and context.
  NOTE: The KINT hardware only supports a single interrupt for all channels

  Enable interrupt for the selected channels after casting since KRM is an 8 bit register
- Enable interrupt
- Store number of channels for use to the control block to use it later
- Mark KINT as initialized

9.28.4 R_KINT_KEYMATRIX_Close

```
ssp_err_t R_KINT_KEYMATRIX_Close ( keymatrix_ctrl_t  *const p_api_ctrl )
```

9.28.4.1 Brief description

Disable KINT. Implements close.

9.28.4.2 Detailed description

End of function R_KINT_KEYMATRIX_Open The Close function disables the interrupts in the peripheral and the NVIC and clears any pending interrupt requests. It also releases the hardware lock on the API.

Table 1656:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.

9.28.4.3 Function steps

- Disable interrupt in ICU
- Disable interrupts in the KINT peripheral
- Clear any pending interrupt requests in the KINT peripheral
- Clear the Interrupt Request bit
- Clear stored internal driver data
- Mark KINT as un-initialized
- Release the lock

9.28.5 R_KINT_KEYMATRIX_Enable

```
ssp_err_t R_KINT_KEYMATRIX_Enable ( keymatrix_ctrl_t  *const p_api_ctrl )
```

9.28.5.1 Brief description

Enable external irq for all the specified channel by R_KINT_KEYMATRIX_Open. Implements enable.

9.28.5.2 Detailed description

This function enables interrupts for the KINT peripheral both at the interrupt level and in the NVIC after clearing any pending requests in the KINT and ICU peripheral. All the channels specified by R_KINT_KEYMATRIX_Open are enabled.

Table 1657:Return values

Name	Description
SSP_SUCCESS	Interrupt disabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The peripheral is not opened.

9.28.5.3 Function steps

- Clear any pending interrupt requests in the KINT peripheral
- Clear the Interrupt Request Flag in the ICU
- Enable interrupt for the selected channels after casting since KRM is an 8 bit register
- Enable interrupt

9.28.6 R_KINT_KEYMATRIX_Disable

```
ssp_err_t R_KINT_KEYMATRIX_Disable ( keymatrix_ctrl_t  *const p_api_ctrl )
```

9.28.6.1 Brief description

Disable external irq for all the specified channel by R_KINT_KEYMATRIX_Open. Implements disable.

9.28.6.2 Detailed description

This function disables interrupts for the KINT peripheral both at the interrupt level and in the NVIC. All the channels specified by R_KINT_KEYMATRIX_Open are disabled.

Table 1658:Return values

Name	Description
SSP_SUCCESS	Interrupt disabled successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.28.6.3 Function steps

- Disable interrupts in the KINT peripheral
- Clear any pending interrupt requests in the KINT peripheral
- Disable interrupt in the ICU

9.28.7 R_KINT_KEYMATRIX_TriggerSet

```
ssp_err_t R_KINT_KEYMATRIX_TriggerSet ( keymatrix_ctrl_t  *const p_api_ctrl ,
 keymatrix_trigger_t  hw_trigger )
```

9.28.7.1 Brief description

Set trigger edge (falling or rising) provided. Implements triggerSet.

9.28.7.2 Detailed description

This function changes trigger sense at run-time. The change is applied to all the channels specified by R_KINT_KEYMATRIX_Open.

Table 1659:Return values

Name	Description
SSP_SUCCESS	Trigger value written successfully.
SSP_ERR_ASSERTION	The p_ctrl parameter was null.
SSP_ERR_INVALID_ARGUMENT	Trigger input invalid. hw_trigger must be one of the options from button_trigger_t.
SSP_ERR_NOT_OPEN	The channel is not opened.

NOTE: This function expects to be called when the driver is disabled (the driver state before R_KINT_KEYMATRIX-_Enable is called if the driver is opened in the non-auto start mode, or after R_KINT_KEYMATRIX_Disable is called if the driver is opened in the auto start mode). The driver does not restrict to call this API when the driver is enabled but users need to make sure the edge detection sense is instantly changed by this API call.

9.28.7.3 Function steps

- Configure the trigger edge
- KEYMATRIX_TRIG_RISING condition
- Configure the usage of key interrupt flags

9.28.8 R_KINT_VersionGet

```
ssp_err_t R_KINT_VersionGet ( ssp_version_t  *const p_version )
```

9.28.8.1 Brief description

Set driver version based on compile time macros.

9.28.8.2 Detailed description

Table 1660:Return values

Name	Description
SSP_SUCCESS	Successful return.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

9.28.9 Extensions

9.28.9.1 kint_instance_ctrl_t

kint_instance_ctrl_t

Detailed description

Channel instance control block. DO NOT INITIALIZE. Initialization occurs when open is called.

Variables

- R_KINT_Type * p_reg

  Pointer to register base address.

- keymatrix_channels_t channels

  Channel bitmask.

- IRQn_Type irq

  Interrupt priority number.

9.29 LPM

Driver for Low Power Modes.

The LPM (Low Power Mode) Driver provides several functions for controlling power consumption including stopping modules, selecting the operating mode, configuring low power modes, and transitioning to low power modes. The LPM driver supports configuration of MCU operating modes and MCU low power modes using the LPM hardware peripheral. The LPM driver supports operating modes low-voltage, low-speed, middle-speed, high-speed, and suboscillator mode.

The LPM driver supports low power modes deep standby, standby, sleep, and snooze. The LPM driver supports reducing power consumption when in deep standby mode via internal power supply control and reseting the states of IO ports. The LPM driver supports disabling and enabling of the MCU's other hardware peripherals.

NOTE: Not all operating modes are available on all MCU's. Not all low power modes are available on all MCU's.

9.29.1 Functions

- R_LPM_Init
- R_LPM_MSTPCRSet
- R_LPM_MSTPCRGet
- R_LPM_ModuleStop
- R_LPM_ModuleStart
- R_LPM_OperatingPowerModeSet
- R_LPM_SnoozeEnable
- R_LPM_SnoozeDisable
- R_LPM_LowPowerConfigure
- R_LPM_WUPENSet
- R_LPM_WUPENGet
- R_LPM_DeepStandbyCancelRequestEnable
- R_LPM_DeepStandbyCancelRequestDisable
- R_LPM_LowPowerModeEnter

- R_LPM_VersionGet

9.29.2 Defines

- #define LPM_CODE_VERSION_MAJOR
  Initial value: `(2U)`
- #define LPM_CODE_VERSION_MINOR
  Initial value: `(6U)`

9.29.3 R_LPM_Init

```
ssp_err_t R_LPM_Init ( lpm_cfg_t  const *const p_cfg )
```

9.29.3.1 Brief description

Initialize the mcu operating power mode.

9.29.3.2 Detailed description

Initialize the mcu operating power mode according to the passed in config structure.

Table 1661:Return values

Name	Description
SSP_SUCCESS	Configuration was successful
SSP_ERR_ASSERTION	p_cfg was NULL

9.29.4 R_LPM_MSTPCRSet

```
ssp_err_t R_LPM_MSTPCRSet (  uint32_t mstpcra_value ,   uint32_t mstpcrb_value ,
  uint32_t mstpcrc_value ,   uint32_t mstpcrd_value )
```

9.29.4.1 Brief description

Set the value of all the Module Stop Control Registers.

9.29.4.2 Detailed description

Table 1662:Return values

Name	Description
SSP_SUCCESS	Value was successfully set

9.29.5 R_LPM_MSTPCRGet

```
ssp_err_t R_LPM_MSTPCRGet ( uint32_t * mstpcra_value , uint32_t
* mstpcrb_value , uint32_t * mstpcrc_value , uint32_t * mstpcrd_value )
```

9.29.5.1 Brief description

Get the values of all the Module Stop Control Registers.

9.29.5.2 Detailed description

Table 1663:Return values

Name	Description
SSP_SUCCESS	Value was successfully retrieved
SSP_ERR_ASSERTION	One of the passed in pointers was NULL

9.29.6 R_LPM_ModuleStop

```
ssp_err_t R_LPM_ModuleStop ( lpm_mstp_t module )
```

9.29.6.1 Brief description

Stop a module.

9.29.6.2 Detailed description

Stop a module from running by setting the corresponding bit in the module stop register.

Table 1664:Return values

Name	Description
SSP_SUCCESS	Configuration was successful
SSP_ERR_ASSERTION	Module stop bit not available on this MCU.

9.29.7 R_LPM_ModuleStart

```
ssp_err_t R_LPM_ModuleStart ( lpm_mstp_t  module )
```

9.29.7.1 Brief description

Run a module.

9.29.7.2 Detailed description

Start a module running by clearing the corresponding bit in the module stop register.

Table 1665:Return values

Name	Description
SSP_SUCCESS	Configuration was successful
SSP_ERR_ASSERTION	Module stop bit not available on this MCU.

9.29.8 R_LPM_OperatingPowerModeSet

```
ssp_err_t R_LPM_OperatingPowerModeSet ( lpm_operating_power_t  power_mode ,
 lpm_subosc_t  subosc )
```

9.29.8.1 Brief description

Set the operating power mode.

9.29.8.2 Detailed description

Table 1666:Return values

Name	Description
SSP_SUCCESS	Operating power mode successfully set
SSP_ERR_ASSERTION	Operating power control mode not available on this MCU.

9.29.9 R_LPM_SnoozeEnable

```
ssp_err_t R_LPM_SnoozeEnable ( lpm_snooze_rxd0_t  rdx0_mode ,
 lpm_snooze_dtc_t  dtc_mode ,  lpm_snooze_request_t  requests ,
 lpm_snooze_end_t  triggers )
```

9.29.9.1 Brief description

Configure and enable snooze mode.

9.29.9.2 Detailed description

NOTE: this function must be called before entering Software Standby mode, otherwise snooze mode will not be entered

Table 1667:Return values

Name	Description
SSP_SUCCESS	Snooze mode successfully enabled
SSP_ERR_ASSERTION	One or more snooze settings are not available on this MCU.

9.29.10 R_LPM_SnoozeDisable

```
ssp_err_t R_LPM_SnoozeDisable (  void  )
```

9.29.10.1 Brief description

Disable snooze mode.

9.29.10.2 Detailed description

Table 1668:Return values

Name	Description
SSP_SUCCESS	Snooze mode successfully disabled

9.29.11 R_LPM_LowPowerConfigure

```
ssp_err_t R_LPM_LowPowerConfigure ( lpm_low_power_mode_t  power_mode ,
 lpm_output_port_enable_t  output_port_enable ,
 lpm_power_supply_t  power_supply ,  lpm_io_port_t  io_port_state )
```

9.29.11.1 Brief description

Configure a low power mode.

9.29.11.2 Detailed description

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

Table 1669:Return values

Name	Description
SSP_SUCCESS	Sleep mode entered successfully
SSP_ERR_ASSERTION	Low power mode are not available on this MCU.
SSP_ERR_ASSERTION	Invalid deep standby power supply configuration.

9.29.12 R_LPM_WUPENSet

```
ssp_err_t R_LPM_WUPENSet (  uint32_t wupen_value )
```

9.29.12.1 Brief description

Set the value of the Wake Up Interrupt Enable Register WUPEN.

9.29.12.2 Detailed description

Table 1670:Return values

Name	Description
SSP_SUCCESS	Value was successfully set

9.29.13 R_LPM_WUPENGet

```
ssp_err_t R_LPM_WUPENGet (  uint32_t * wupen_value )
```

9.29.13.1 Brief description

Get the value of the Wake Up Interrupt Enable Register WUPEN.

9.29.13.2 Detailed description

Table 1671:Return values

Name	Description
SSP_SUCCESS	Value was successfully retrieved
SSP_ERR_ASSERTION	One of the passed in pointers was NULL

9.29.14 R_LPM_DeepStandbyCancelRequestEnable

```
ssp_err_t R_LPM_DeepStandbyCancelRequestEnable ( lpm_deep_standby_t  pin_signal
,  lpm_cancel_request_edge_t  rising_falling )
```

9.29.14.1 Brief description

Enable a Deep Standby Cancel Request.

9.29.14.2 Detailed description

Enable a pin or signal that will cancel Deep Software Standby mode

Table 1672:Return values

Name	Description
SSP_SUCCESS	Deep Software Standby cancel request enabled

9.29.15 R_LPM_DeepStandbyCancelRequestDisable

```
ssp_err_t R_LPM_DeepStandbyCancelRequestDisable ( lpm_deep_standby_t  pin_signal
 )
```

9.29.15.1 Brief description

Disable a Deep Standby Cancel Request.

9.29.15.2 Detailed description

Disable a pin or signal that will cancel Deep Software Standby mode

Table 1673:Return values

Name	Description
SSP_SUCCESS	Deep Software Standby cancel request disabled

9.29.16 R_LPM_LowPowerModeEnter

```
ssp_err_t R_LPM_LowPowerModeEnter (  void  )
```

9.29.16.1 Brief description

Enter low power mode (sleep/standby/deep standby) using WFI macro.

9.29.16.2 Detailed description

Function will return after waking from low power mode.

Table 1674:Return values

Name	Description
SSP_SUCCESS	Successful.

9.29.16.3 Function steps

- DSB should be last instruction executed before WFI
  infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dai0321a/BIHICBGB.html

9.29.17 R_LPM_VersionGet

```
ssp_err_t R_LPM_VersionGet ( ssp_version_t  *const p_version )
```

9.29.17.1 Brief description

Get the driver version based on compile time macros.

9.29.17.2 Detailed description

Table 1675:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_version is NULL.

9.30 LPMV2 S124

Driver for Low Power Modes.

The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

NOTE: Not all low power modes are available on all MCU's.

9.30.1 Functions

- R_LPMV2_VersionGet
- R_LPMV2_Init
- R_LPMV2_LowPowerConfigure
- R_LPMV2_LowPowerModeEnter

9.30.2 Typedefs

- lpmv2_snooze_end_bits_t
- lpmv2_standby_wake_source_bits_t

9.30.3 Defines

- #define LPMV2_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define LPMV2_CODE_VERSION_MINOR

  Initial value: `(3U)`
- #define LPMV2_ERROR_RETURN

  Initial value:`#define SSP_ERROR_RETURN((a), (err), &(g_lpmv2_module_name[0]), &g_lpmv2_version)`

9.30.4 User-defined macros

- #define LPMV2_CFG_PARAM_CHECKING_ENABLE

  Initial value: `(BSP_CFG_PARAM_CHECKING_ENABLE)`

  Specify whether to include code for API parameter checking. Valid settings include:BSP_CFG_PARAM_-CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h1 : Includes parameter checking0 : Compiles out parameter checking

9.30.5 R_LPMV2_VersionGet

```
ssp_err_t R_LPMV2_VersionGet ( ssp_version_t  *const p_version )
```

9.30.5.1 Brief description

Get the driver version based on compile time macros.

9.30.5.2 Detailed description

Table 1676:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_INVALID_POINTER	p_version is NULL.

9.30.6 R_LPMV2_Init

```
ssp_err_t R_LPMV2_Init (  void  )
```

9.30.6.1 Brief description

Perform any necessary initialization.

9.30.6.2 Detailed description

Table 1677:Return values

Name	Description
SSP_SUCCESS	LPMV2 Software lock initialized

9.30.7 R_LPMV2_LowPowerConfigure

```
ssp_err_t R_LPMV2_LowPowerConfigure ( lpmv2_cfg_t  const *const p_cfg )
```

9.30.7.1 Brief description

Configure a low power mode.

9.30.7.2 Detailed description

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

Table 1678:Return values

Name	Description
SSP_SUCCESS	Low power mode successfuly applied
SSP_ERR_INVALID_POINTER	p_cfg is NULL
SSP_ERR_IN_USE	Lock was not acquired
SSP_ERR_INVALID_HW_CONDITION	Operating mode change transition was detected (OPCMTSF, SOPCMTSF bits)

9.30.8 R_LPMV2_LowPowerModeEnter

```
ssp_err_t R_LPMV2_LowPowerModeEnter ( void )
```

9.30.8.1 Brief description

Enter low power mode (sleep/standby/deep standby) using WFI macro.

9.30.8.2 Detailed description

Function will return after waking from low power mode.

Table 1679:Return values

Name	Description
SSP_SUCCESS	Successful.
SSP_ERR_INVALID_HW_CONDITION	HOCO was unstable during attempt to revert operating mode.

9.30.9 API Data

9.30.9.1 lpmv2_snooze_request_t

```
lpmv2_snooze_request_t
```

Detailed description

Snooze request sources

Enumerated values

Name	Description
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request.
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request.

Name	Description
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request.
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request.
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request.
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request.
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request.
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request.
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request.

9.30.9.2 lpmv2_snooze_end_t

`lpmv2_snooze_end_t`

Detailed description

Snooze end control

Enumerated values

Name	Description
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly.
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion.
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match.
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch.
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch.
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match.
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch.

9.30.9.3 lpmv2_snooze_dtc_t

`lpmv2_snooze_dtc_t`

Detailed description

DTC Enable in Snooze Mode

Enumerated values

Name	Description
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation

9.30.9.4 lpmv2_standby_wake_source_t

`lpmv2_standby_wake_source_t`

Detailed description

Wake from standby mode sources, does not apply to sleep or deep standby modes

Enumerated values

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0.
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1.
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2.
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3.
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4.
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5.
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6.
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7.
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt.
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt.
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8.
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9.
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10.

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11.
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12.
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13.
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14.
LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15.
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt.

9.30.9.5 lpmv2_snooze_end_bits_t

```
typedef uint8_t lpmv2_snooze_end_bits_t
```

9.30.9.6 lpmv2_standby_wake_source_bits_t

```
typedef uint32_t lpmv2_standby_wake_source_bits_t
```

9.30.10 Extensions

9.30.10.1 lpmv2_mcu_cfg_t

lpmv2_mcu_cfg_t

Detailed description

S124 specific configuration structure

S128 specific configuration structure

S3A3 specific configuration structure

S3A6 specific configuration structure

S3A7 specific configuration structure

S5D5 specific configuration structure

S5D9 specific configuration structure

S7G2 specific configuration structure

Variables

- lpmv2_standby_wake_source_bits_t standby_wake_sources
  Bitwise list of sources to wake from standby
- lpmv2_snooze_request_t snooze_request_source
  Snooze request source
- lpmv2_snooze_end_bits_t snooze_end_sources
  Bitwise list of snooze end sources
- lpmv2_snooze_dtc_t dtc_state_in_snooze
  State of DTC in snooze mode, enabled or disabled
- lpmv2_output_port_enable_t output_port_enable
  Output port enabled/disabled in standby and deep standby
- lpmv2_io_port_t io_port_state
  IO port state in deep standby (maintained or reset)
- lpmv2_power_supply_t power_supply_state
  Internal power supply state in standby and deep standby (deepcut)
- lpmv2_deep_standby_cancel_source_bits_t deep_standby_cancel_source
  Sources that can trigger exit from deep standby
- lpmv2_deep_standby_cancel_edge_bits_t deep_standby_cancel_edge
  Signal edges for the sources that can trigger exit from deep standby

9.31 LPMV2 S128

Driver for Low Power Modes.
The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

NOTE: Not all low power modes are available on all MCU's.

9.31.1 Typedefs

- lpmv2_snooze_end_bits_t
- lpmv2_standby_wake_source_bits_t

9.31.2 Defines

- #define LPMV2_CODE_VERSION_MAJOR

  Initial value: `(1U)`

- #define LPMV2_CODE_VERSION_MINOR

  Initial value: `(3U)`

- #define LPMV2_ERROR_RETURN

  Initial value:`#define SSP_ERROR_RETURN((a), (err), &(g_lpmv2_module_name[0]), &g_lpmv2_version)`

9.31.3 User-defined macros

- #define LPMV2_CFG_PARAM_CHECKING_ENABLE

  Initial value: `(BSP_CFG_PARAM_CHECKING_ENABLE)`

  Specify whether to include code for API parameter checking. Valid settings include:BSP_CFG_PARAM_-CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h1 : Includes parameter checking0 : Compiles out parameter checking

9.31.4 API Data

9.31.4.1 lpmv2_snooze_request_t

`lpmv2_snooze_request_t`

Detailed description

Snooze request sources

Enumerated values

Name	Description
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request.
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request.

Name	Description
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request.
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request.
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request.
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request.
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request.
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request.
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request.

9.31.4.2 lpmv2_snooze_end_t

`lpmv2_snooze_end_t`

Detailed description

Snooze end control

Enumerated values

Name	Description
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly.
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion.
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match.
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch.
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch.
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match.
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch.

9.31.4.3 lpmv2_snooze_dtc_t

`lpmv2_snooze_dtc_t`

Detailed description

DTC Enable in Snooze Mode

Enumerated values

Name	Description
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation

9.31.4.4 lpmv2_standby_wake_source_t

`lpmv2_standby_wake_source_t`

Detailed description

Wake from standby mode sources, does not apply to sleep or deep standby modes

Enumerated values

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0.
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1.
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2.
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3.
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4.
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5.
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6.
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7.
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt.
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt.
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8.
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9.
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10.

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11.
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12.
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13.
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14.
LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15.
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt.

9.31.4.5 lpmv2_snooze_end_bits_t

```
typedef uint8_t lpmv2_snooze_end_bits_t
```

9.31.4.6 lpmv2_standby_wake_source_bits_t

```
typedef uint32_t lpmv2_standby_wake_source_bits_t
```

9.31.5 Extensions

9.31.5.1 lpmv2_mcu_cfg_t

lpmv2_mcu_cfg_t

Detailed description

S124 specific configuration structure

S128 specific configuration structure

S3A3 specific configuration structure

S3A6 specific configuration structure

S3A7 specific configuration structure

S5D5 specific configuration structure

S5D9 specific configuration structure

S7G2 specific configuration structure

Variables

- lpmv2_standby_wake_source_bits_t standby_wake_sources
  Bitwise list of sources to wake from standby
- lpmv2_snooze_request_t snooze_request_source
  Snooze request source
- lpmv2_snooze_end_bits_t snooze_end_sources
  Bitwise list of snooze end sources
- lpmv2_snooze_dtc_t dtc_state_in_snooze
  State of DTC in snooze mode, enabled or disabled
- lpmv2_output_port_enable_t output_port_enable
  Output port enabled/disabled in standby and deep standby
- lpmv2_io_port_t io_port_state
  IO port state in deep standby (maintained or reset)
- lpmv2_power_supply_t power_supply_state
  Internal power supply state in standby and deep standby (deepcut)
- lpmv2_deep_standby_cancel_source_bits_t deep_standby_cancel_source
  Sources that can trigger exit from deep standby
- lpmv2_deep_standby_cancel_edge_bits_t deep_standby_cancel_edge
  Signal edges for the sources that can trigger exit from deep standby

9.32 LPMV2 S3A3

Driver for Low Power Modes.
The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

NOTE: Not all low power modes are available on all MCU's.

9.32.1 Typedefs

- lpmv2_snooze_end_bits_t
- lpmv2_standby_wake_source_bits_t

9.32.2 Defines

- #define LPMV2_CODE_VERSION_MAJOR

  Initial value: `(1U)`

- #define LPMV2_CODE_VERSION_MINOR

  Initial value: `(3U)`

- #define LPMV2_ERROR_RETURN

  Initial value:`#define SSP_ERROR_RETURN((a), (err), &(g_lpmv2_module_name[0]), &g_lpmv2_version)`

9.32.3 User-defined macros

- #define LPMV2_CFG_PARAM_CHECKING_ENABLE

  Initial value: `(BSP_CFG_PARAM_CHECKING_ENABLE)`

  Specify whether to include code for API parameter checking. Valid settings include:BSP_CFG_PARAM_-CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h1 : Includes parameter checking0 : Compiles out parameter checking

9.32.4 API Data

9.32.4.1 lpmv2_snooze_request_t

`lpmv2_snooze_request_t`

Detailed description

Snooze request sources

Enumerated values

Name	Description
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request.
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request.

Name	Description
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request.
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request.
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request.
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request.
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request.
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request.
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request.

9.32.4.2 lpmv2_snooze_end_t

`lpmv2_snooze_end_t`

Detailed description

Snooze end control

Enumerated values

Name	Description
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly.
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion.
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match.
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch.
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch.
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match.
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch.

9.32.4.3 lpmv2_snooze_dtc_t

`lpmv2_snooze_dtc_t`

Detailed description

DTC Enable in Snooze Mode

Enumerated values

Name	Description
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation

9.32.4.4 lpmv2_standby_wake_source_t

`lpmv2_standby_wake_source_t`

Detailed description

Wake from standby mode sources, does not apply to sleep or deep standby modes

Enumerated values

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0.
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1.
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2.
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3.
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4.
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5.
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6.
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7.
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt.
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt.
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8.
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9.
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10.

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11.
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12.
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13.
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14.
LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15.
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt.

9.32.4.5 lpmv2_output_port_enable_t

`lpmv2_output_port_enable_t`

Detailed description

Output port enable

Enumerated values

Name	Description
LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode.
LPMV2_OUTPUT_PORT_ENABLE_RETAIN	1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state.

9.32.4.6 lpmv2_snooze_request_t

`lpmv2_snooze_request_t`

Detailed description

Snooze request sources

Enumerated values

Name	Description
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request.
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request.
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request.
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request.
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request.
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request.
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request.

Name	Description
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request.
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request.

9.32.4.7 lpmv2_snooze_end_t

`lpmv2_snooze_end_t`

Detailed description

Snooze end control

Enumerated values

Name	Description
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly.
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion.
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match.
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch.
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch.
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match.
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch.

9.32.4.8 lpmv2_snooze_dtc_t

`lpmv2_snooze_dtc_t`

Detailed description

DTC Enable in Snooze Mode

Enumerated values

Name	Description
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation

9.32.4.9 lpmv2_standby_wake_source_t

`lpmv2_standby_wake_source_t`

Detailed description

Wake from standby mode sources, does not apply to sleep or deep standby modes

Enumerated values

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0.
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1.
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2.
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3.
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4.
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5.
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6.
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7.
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt.
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt.

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8.
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9.
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10.
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11.
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12.
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13.
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14.
LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15.
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt.

9.32.4.10 lpmv2_output_port_enable_t

`lpmv2_output_port_enable_t`

Detailed description

Output port enable

Enumerated values

Name	Description
LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode.
LPMV2_OUTPUT_PORT_ENABLE_RETAIN	1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state.

9.32.4.11 lpmv2_snooze_end_bits_t

```
typedef uint8_t lpmv2_snooze_end_bits_t
```

9.32.4.12 lpmv2_standby_wake_source_bits_t

```
typedef uint32_t lpmv2_standby_wake_source_bits_t
```

9.32.4.13 lpmv2_snooze_end_bits_t

```
typedef uint8_t lpmv2_snooze_end_bits_t
```

9.32.4.14 lpmv2_standby_wake_source_bits_t

```
typedef uint32_t lpmv2_standby_wake_source_bits_t
```

9.32.5 Extensions

9.32.5.1 lpmv2_mcu_cfg_t

lpmv2_mcu_cfg_t

Detailed description

S124 specific configuration structure

S128 specific configuration structure

S3A3 specific configuration structure

S3A6 specific configuration structure

S3A7 specific configuration structure

S5D5 specific configuration structure

S5D9 specific configuration structure

S7G2 specific configuration structure

Variables

- lpmv2_standby_wake_source_bits_t standby_wake_sources
  Bitwise list of sources to wake from standby
- lpmv2_snooze_request_t snooze_request_source
  Snooze request source
- lpmv2_snooze_end_bits_t snooze_end_sources
  Bitwise list of snooze end sources
- lpmv2_snooze_dtc_t dtc_state_in_snooze
  State of DTC in snooze mode, enabled or disabled
- lpmv2_output_port_enable_t output_port_enable
  Output port enabled/disabled in standby and deep standby
- lpmv2_io_port_t io_port_state
  IO port state in deep standby (maintained or reset)
- lpmv2_power_supply_t power_supply_state
  Internal power supply state in standby and deep standby (deepcut)
- lpmv2_deep_standby_cancel_source_bits_t deep_standby_cancel_source
  Sources that can trigger exit from deep standby
- lpmv2_deep_standby_cancel_edge_bits_t deep_standby_cancel_edge
  Signal edges for the sources that can trigger exit from deep standby

9.33 LPMV2 S3A6

Driver for Low Power Modes.
The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

NOTE: Not all low power modes are available on all MCU's.

9.33.1 Typedefs

- lpmv2_snooze_end_bits_t
- lpmv2_standby_wake_source_bits_t

9.33.2 Defines

- #define LPMV2_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define LPMV2_CODE_VERSION_MINOR

  Initial value: `(3U)`
- #define LPMV2_ERROR_RETURN

  Initial value:`#define SSP_ERROR_RETURN((a), (err), &(g_lpmv2_module_name[0]), &g_lpmv2_version)`

9.33.3 API Data

9.33.3.1 lpmv2_snooze_request_t

`lpmv2_snooze_request_t`

Detailed description

Snooze request sources

Enumerated values

Name	Description
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request.
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request.
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request.
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request.

Name	Description
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request.
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request.
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request.
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request.

9.33.3.2 lpmv2_snooze_end_t

`lpmv2_snooze_end_t`

Detailed description

Snooze end control

Enumerated values

Name	Description
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly.
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion.

Name	Description
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion.
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match.
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch.
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch.
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match.
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch.

9.33.3.3 lpmv2_snooze_dtc_t

`lpmv2_snooze_dtc_t`

Detailed description

DTC Enable in Snooze Mode

Enumerated values

Name	Description
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation

9.33.3.4 lpmv2_standby_wake_source_t

`lpmv2_standby_wake_source_t`

Detailed description

Wake from standby mode sources, does not apply to sleep or deep standby modes

Enumerated values

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0.
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1.
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2.

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3.
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4.
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5.
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6.
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7.
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt.
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt.
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8.
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9.
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10.
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11.
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12.
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13.
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14.

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15.
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt.

9.33.3.5 lpmv2_snooze_end_bits_t

```
typedef uint8_t lpmv2_snooze_end_bits_t
```

9.33.3.6 lpmv2_standby_wake_source_bits_t

```
typedef uint32_t lpmv2_standby_wake_source_bits_t
```

9.33.4 Extensions

9.33.4.1 lpmv2_mcu_cfg_t

lpmv2_mcu_cfg_t

Detailed description

S124 specific configuration structure

S128 specific configuration structure

S3A3 specific configuration structure

S3A6 specific configuration structure

S3A7 specific configuration structure

S5D5 specific configuration structure

S5D9 specific configuration structure

S7G2 specific configuration structure

Variables

- lpmv2_standby_wake_source_bits_t standby_wake_sources
  Bitwise list of sources to wake from standby
- lpmv2_snooze_request_t snooze_request_source
  Snooze request source
- lpmv2_snooze_end_bits_t snooze_end_sources
  Bitwise list of snooze end sources

- lpmv2_snooze_dtc_t dtc_state_in_snooze

  State of DTC in snooze mode, enabled or disabled
- lpmv2_output_port_enable_t output_port_enable

  Output port enabled/disabled in standby and deep standby
- lpmv2_io_port_t io_port_state

  IO port state in deep standby (maintained or reset)
- lpmv2_power_supply_t power_supply_state

  Internal power supply state in standby and deep standby (deepcut)
- lpmv2_deep_standby_cancel_source_bits_t deep_standby_cancel_source

  Sources that can trigger exit from deep standby
- lpmv2_deep_standby_cancel_edge_bits_t deep_standby_cancel_edge

  Signal edges for the sources that can trigger exit from deep standby

9.34 LPMV2 S1JA

Driver for Low Power Modes.

The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

NOTE: Not all low power modes are available on all MCU's.

9.34.1 Typedefs

- lpmv2_snooze_end_bits_t
- lpmv2_standby_wake_source_bits_t

9.34.2 Defines

- #define LPMV2_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define LPMV2_CODE_VERSION_MINOR

  Initial value: `(3U)`
- #define LPMV2_ERROR_RETURN

  Initial value:
```
#define SSP_ERROR_RETURN((a), (err), &(g_lpmv2_module_name[0]), &g_lpmv2_version)
```

9.34.3 User-defined macros

- #define LPMV2_CFG_PARAM_CHECKING_ENABLE

  Initial value: `(BSP_CFG_PARAM_CHECKING_ENABLE)`

  Specify whether to include code for API parameter checking. Valid settings include:BSP_CFG_PARAM_-CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h1 : Includes parameter checking0 : Compiles out parameter checking

9.34.4 API Data

9.34.4.1 lpmv2_snooze_request_t

`lpmv2_snooze_request_t`

Detailed description

Snooze request sources

Enumerated values

Name	Description
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request.
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request.
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request.
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request.
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request.
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request.

Name	Description
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request.
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request.
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request.

9.34.4.2 lpmv2_snooze_end_t

`lpmv2_snooze_end_t`

Detailed description

Snooze end control

Enumerated values

Name	Description
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly.
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion.
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match.

Name	Description
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch.
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch.
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match.
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch.

9.34.4.3 lpmv2_snooze_dtc_t

`lpmv2_snooze_dtc_t`

Detailed description

DTC Enable in Snooze Mode

Enumerated values

Name	Description
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation

9.34.4.4 lpmv2_standby_wake_source_t

`lpmv2_standby_wake_source_t`

Detailed description

Wake from standby mode sources, does not apply to sleep or deep standby modes

Enumerated values

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0.
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1.
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2.
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3.
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4.
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5.

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6.
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7.
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt.
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt.
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8.
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9.
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10.
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11.
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12.
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13.
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14.
LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15.
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt.

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt.

9.34.4.5 lpmv2_snooze_end_bits_t

```
typedef uint8_t lpmv2_snooze_end_bits_t
```

9.34.4.6 lpmv2_standby_wake_source_bits_t

```
typedef uint32_t lpmv2_standby_wake_source_bits_t
```

9.34.5 Extensions

9.34.5.1 lpmv2_mcu_cfg_t

lpmv2_mcu_cfg_t

Detailed description

S124 specific configuration structure

S128 specific configuration structure

S3A3 specific configuration structure

S3A6 specific configuration structure

S3A7 specific configuration structure

S5D5 specific configuration structure

S5D9 specific configuration structure

S7G2 specific configuration structure

Variables

- lpmv2_standby_wake_source_bits_t standby_wake_sources
  Bitwise list of sources to wake from standby
- lpmv2_snooze_request_t snooze_request_source
  Snooze request source
- lpmv2_snooze_end_bits_t snooze_end_sources
  Bitwise list of snooze end sources
- lpmv2_snooze_dtc_t dtc_state_in_snooze
  State of DTC in snooze mode, enabled or disabled
- lpmv2_output_port_enable_t output_port_enable
  Output port enabled/disabled in standby and deep standby

- lpmv2_io_port_t io_port_state
  IO port state in deep standby (maintained or reset)
- lpmv2_power_supply_t power_supply_state
  Internal power supply state in standby and deep standby (deepcut)
- lpmv2_deep_standby_cancel_source_bits_t deep_standby_cancel_source
  Sources that can trigger exit from deep standby
- lpmv2_deep_standby_cancel_edge_bits_t deep_standby_cancel_edge
  Signal edges for the sources that can trigger exit from deep standby

9.35 LPMV2 S3A7

Driver for Low Power Modes.
The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

NOTE: Not all low power modes are available on all MCU's.

9.35.1 Functions

- R_LPMV2_VersionGet
- R_LPMV2_Init
- R_LPMV2_LowPowerConfigure
- R_LPMV2_LowPowerModeEnter

9.35.2 Typedefs

- lpmv2_snooze_end_bits_t
- lpmv2_standby_wake_source_bits_t

9.35.3 Defines

- #define LPMV2_CODE_VERSION_MAJOR
  Initial value: `(1U)`
- #define LPMV2_CODE_VERSION_MINOR
  Initial value: `(3U)`

- #define LPMV2_ERROR_RETURN

  Initial value:`#define SSP_ERROR_RETURN((a), (err), &(g_lpmv2_module_name[0]), &g_lpmv2_version)`

9.35.4 User-defined macros

- #define LPMV2_CFG_PARAM_CHECKING_ENABLE

  Initial value: `(BSP_CFG_PARAM_CHECKING_ENABLE)`

  Specify whether to include code for API parameter checking. Valid settings include:BSP_CFG_PARAM_-CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h1 : Includes parameter checking0 : Compiles out parameter checking

9.35.5 R_LPMV2_VersionGet

```
ssp_err_t R_LPMV2_VersionGet ( ssp_version_t  *const p_version )
```

9.35.5.1 Brief description

Get the driver version based on compile time macros.

9.35.5.2 Detailed description

Table 1680:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_INVALID_POINTER	p_version is NULL.

9.35.6 R_LPMV2_Init

```
ssp_err_t R_LPMV2_Init (  void  )
```

9.35.6.1 Brief description

Perform any necessary initialization.

9.35.6.2 Detailed description

Table 1681:Return values

Name	Description
SSP_SUCCESS	LPMV2 Software lock initialized

9.35.7 R_LPMV2_LowPowerConfigure

```
ssp_err_t R_LPMV2_LowPowerConfigure ( lpmv2_cfg_t  const *const p_cfg )
```

9.35.7.1 Brief description

Configure a low power mode.

9.35.7.2 Detailed description

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

Table 1682:Return values

Name	Description
SSP_SUCCESS	Low power mode successfuly applied
SSP_ERR_INVALID_POINTER	p_cfg is NULL
SSP_ERR_IN_USE	Lock was not acquired
SSP_ERR_INVALID_HW_CONDITION	Operating mode change transition was detected (OPCMTSF, SOPCMTSF bits)

9.35.8 R_LPMV2_LowPowerModeEnter

```
ssp_err_t R_LPMV2_LowPowerModeEnter (  void  )
```

9.35.8.1 Brief description

Enter low power mode (sleep/standby/deep standby) using WFI macro.

9.35.8.2 Detailed description

Function will return after waking from low power mode.

Table 1683:Return values

Name	Description
SSP_SUCCESS	Successful.
SSP_ERR_INVALID_HW_CONDITION	HOCO was unstable during attempt to revert operating mode.

9.35.9 API Data

9.35.9.1 lpmv2_snooze_request_t

`lpmv2_snooze_request_t`

Detailed description

Snooze request sources

Enumerated values

Name	Description
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request.
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request.
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request.

Name	Description
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request.
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request.
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request.
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request.
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request.

9.35.9.2 lpmv2_snooze_end_t

`lpmv2_snooze_end_t`

Detailed description

Snooze end control

Enumerated values

Name	Description
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly.
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow.

Name	Description
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion.
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match.
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch.
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch.
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match.
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch.

9.35.9.3 lpmv2_snooze_dtc_t

`lpmv2_snooze_dtc_t`

Detailed description

DTC Enable in Snooze Mode

Enumerated values

Name	Description
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation

9.35.9.4 lpmv2_standby_wake_source_t

`lpmv2_standby_wake_source_t`

Detailed description

Wake from standby mode sources, does not apply to sleep or deep standby modes

Enumerated values

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0.

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1.
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2.
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3.
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4.
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5.
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6.
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7.
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt.
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt.
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8.
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9.
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10.
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11.
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12.

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13.
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14.
LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15.
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt.

9.35.9.5 lpmv2_output_port_enable_t

```
lpmv2_output_port_enable_t
```

Detailed description

Output port enable

Enumerated values

Name	Description
LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode.
LPMV2_OUTPUT_PORT_ENABLE_RETAIN	1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state.

9.35.9.6 lpmv2_snooze_end_bits_t

```
typedef uint8_t lpmv2_snooze_end_bits_t
```

9.35.9.7 lpmv2_standby_wake_source_bits_t

```
typedef uint32_t lpmv2_standby_wake_source_bits_t
```

9.35.10 Extensions

9.35.10.1 lpmv2_mcu_cfg_t

lpmv2_mcu_cfg_t

Detailed description

S124 specific configuration structure
S128 specific configuration structure
S3A3 specific configuration structure
S3A6 specific configuration structure
S3A7 specific configuration structure
S5D5 specific configuration structure
S5D9 specific configuration structure
S7G2 specific configuration structure

Variables

- lpmv2_standby_wake_source_bits_t standby_wake_sources
  Bitwise list of sources to wake from standby
- lpmv2_snooze_request_t snooze_request_source
  Snooze request source
- lpmv2_snooze_end_bits_t snooze_end_sources
  Bitwise list of snooze end sources
- lpmv2_snooze_dtc_t dtc_state_in_snooze
  State of DTC in snooze mode, enabled or disabled
- lpmv2_output_port_enable_t output_port_enable
  Output port enabled/disabled in standby and deep standby
- lpmv2_io_port_t io_port_state
  IO port state in deep standby (maintained or reset)
- lpmv2_power_supply_t power_supply_state
  Internal power supply state in standby and deep standby (deepcut)
- lpmv2_deep_standby_cancel_source_bits_t deep_standby_cancel_source
  Sources that can trigger exit from deep standby
- lpmv2_deep_standby_cancel_edge_bits_t deep_standby_cancel_edge
  Signal edges for the sources that can trigger exit from deep standby

9.36 LPMV2 S5D4

Driver for Low Power Modes.

The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

NOTE: Not all low power modes are available on all MCU's.

9.36.1 Typedefs

- lpmv2_snooze_end_bits_t
- lpmv2_standby_wake_source_bits_t
- lpmv2_deep_standby_cancel_edge_bits_t
- lpmv2_deep_standby_cancel_source_bits_t

9.36.2 Defines

- #define LPMV2_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define LPMV2_CODE_VERSION_MINOR

  Initial value: `(3U)`
- #define LPMV2_ERROR_RETURN

  Initial value:`#define SSP_ERROR_RETURN((a), (err), &(g_lpmv2_module_name[0]), &g_lpmv2_version)`

9.36.3 User-defined macros

- #define LPMV2_CFG_PARAM_CHECKING_ENABLE

  Initial value: `(BSP_CFG_PARAM_CHECKING_ENABLE)`

  Specify whether to include code for API parameter checking. Valid settings include:BSP_CFG_PARAM_-CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h1 : Includes parameter checking0 : Compiles out parameter checking

9.36.4 API Data

9.36.4.1 lpmv2_snooze_request_t

`lpmv2_snooze_request_t`

Detailed description

Snooze request sources

Enumerated values

Name	Description
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request.
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request.
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request.
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request.
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request.
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request.
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request.

Name	Description
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request.
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request.

9.36.4.2 lpmv2_snooze_end_t

`lpmv2_snooze_end_t`

Detailed description

Snooze end control

Enumerated values

Name	Description
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly.
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion.
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match.
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch.
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch.
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match.
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch.

9.36.4.3 lpmv2_snooze_dtc_t

`lpmv2_snooze_dtc_t`

Detailed description

DTC Enable in Snooze Mode

Enumerated values

Name	Description
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation

9.36.4.4 lpmv2_standby_wake_source_t

`lpmv2_standby_wake_source_t`

Detailed description

Wake from standby mode sources, does not apply to sleep or deep standby modes

Enumerated values

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0.
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1.
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2.
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3.
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4.
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5.
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6.
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7.
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt.
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt.

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8.
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9.
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10.
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11.
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12.
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13.
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14.
LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15.
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt.

9.36.4.5 lpmv2_io_port_t

`lpmv2_io_port_t`

Detailed description

I/O port state after Deep Software Standby mode

Enumerated values

Name	Description
LPMV2_IO_PORT_RESET	When the Deep Software Standby mode is canceled, the I/O ports are in the reset state

Name	Description
LPMV2_IO_PORT_NO_CHANGE	When the Deep Software Standby mode is canceled, the I/O ports are in the same state as in the Deep Software Standby mode

9.36.4.6 lpmv2_power_supply_t

`lpmv2_power_supply_t`

Detailed description

Power supply control

Enumerated values

Name	Description
LPMV2_POWER_SUPPLY_DEEPCUT0	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is supplied in deep software standby mode
LPMV2_POWER_SUPPLY_DEEPCUT1	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode
LPMV2_POWER_SUPPLY_DEEPCUT3	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode. In addition, LVD is disabled and the low power function in a poweron reset circuit is enabled

9.36.4.7 lpmv2_deep_standby_cancel_edge_t

`lpmv2_deep_standby_cancel_edge_t`

Detailed description

Deep Standby Interrupt Edge

Enumerated values

Name	Description
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_EDGE_NONE	No options for a deep standby cancel source.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_RISING	IRQ0-DS Pin Rising Edge.

Name	Description
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_FALLING	IRQ0-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_RISING	IRQ1-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_FALLING	IRQ1-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_RISING	IRQ2-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_FALLING	IRQ2-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_RISING	IRQ3-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_FALLING	IRQ3-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_RISING	IRQ4-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_FALLING	IRQ4-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_RISING	IRQ5-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_FALLING	IRQ5-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_RISING	IRQ6-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_FALLING	IRQ6-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_RISING	IRQ7-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_FALLING	IRQ7-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_RISING	IRQ8-DS Pin Rising Edge.

Name	Description
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_FALLING	IRQ8-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_RISING	IRQ9-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_FALLING	IRQ9-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_RISING	IRQ10-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_FALLING	IRQ10-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_RISING	IRQ11-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_FALLING	IRQ11-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_RISING	IRQ12-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_FALLING	IRQ12-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_RISING	IRQ13-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_FALLING	IRQ13-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_RISING	LVD1 Rising Slope.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_FALLING	LVD1 Falling Slope.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_RISING	LVD2 Rising Slope.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_FALLING	LVD2 Falling Slope.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_RISING	NMI Pin Rising Edge.

Name	Description
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_FALLING	NMI Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_RISING	IRQ14-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_FALLING	IRQ14-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_RISING	IRQ15-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_FALLING	IRQ15-DS Pin Falling Edge.

9.36.4.8 lpmv2_deep_standby_cancel_source_t

`lpmv2_deep_standby_cancel_source_t`

Detailed description

Deep Standby cancel sources

Enumerated values

Name	Description
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RESET_ONLY	Cancel deep standby only by reset.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0	IRQ0.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1	IRQ1.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2	IRQ2.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3	IRQ3.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4	IRQ4.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5	IRQ5.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6	IRQ6.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7	IRQ7.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8	IRQ8.

Name	Description
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9	IRQ9.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10	IRQ10.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11	IRQ11.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12	IRQ12.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13	IRQ13.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1	LVD1.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2	LVD2.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_INTERVAL	RTC Interval Interrupt.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_ALARM	RTC Alarm Interrupt.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI	NMI.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBFS	USBFS Suspend/Resume.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_AGT1	AGT1 Underflow.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14	IRQ14.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBHS	USBHS Suspend/Resume.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15	IRQ15.

9.36.4.9 lpmv2_output_port_enable_t

`lpmv2_output_port_enable_t`

Detailed description

Output port enable

Enumerated values

Name	Description
LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode.
LPMV2_OUTPUT_PORT_ENABLE_RETAIN	1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state.

9.36.4.10 lpmv2_snooze_end_bits_t

```
typedef uint8_t lpmv2_snooze_end_bits_t
```

9.36.4.11 lpmv2_standby_wake_source_bits_t

```
typedef uint32_t lpmv2_standby_wake_source_bits_t
```

9.36.4.12 lpmv2_deep_standby_cancel_edge_bits_t

```
typedef uint32_t lpmv2_deep_standby_cancel_edge_bits_t
```

9.36.4.13 lpmv2_deep_standby_cancel_source_bits_t

```
typedef uint32_t lpmv2_deep_standby_cancel_source_bits_t
```

9.36.5 Extensions

9.36.5.1 lpmv2_mcu_cfg_t

lpmv2_mcu_cfg_t

Detailed description

S124 specific configuration structure

S128 specific configuration structure

S3A3 specific configuration structure

S3A6 specific configuration structure

S3A7 specific configuration structure

S5D5 specific configuration structure

S5D9 specific configuration structure

S7G2 specific configuration structure

Variables

- lpmv2_standby_wake_source_bits_t standby_wake_sources
  Bitwise list of sources to wake from standby
- lpmv2_snooze_request_t snooze_request_source
  Snooze request source
- lpmv2_snooze_end_bits_t snooze_end_sources
  Bitwise list of snooze end sources
- lpmv2_snooze_dtc_t dtc_state_in_snooze
  State of DTC in snooze mode, enabled or disabled
- lpmv2_output_port_enable_t output_port_enable
  Output port enabled/disabled in standby and deep standby
- lpmv2_io_port_t io_port_state
  IO port state in deep standby (maintained or reset)
- lpmv2_power_supply_t power_supply_state
  Internal power supply state in standby and deep standby (deepcut)
- lpmv2_deep_standby_cancel_source_bits_t deep_standby_cancel_source
  Sources that can trigger exit from deep standby
- lpmv2_deep_standby_cancel_edge_bits_t deep_standby_cancel_edge
  Signal edges for the sources that can trigger exit from deep standby

9.37 LPMV2 S5D9

Driver for Low Power Modes.
The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

NOTE: Not all low power modes are available on all MCU's.

9.37.1 Typedefs

- lpmv2_snooze_end_bits_t
- lpmv2_standby_wake_source_bits_t
- lpmv2_deep_standby_cancel_edge_bits_t

- lpmv2_deep_standby_cancel_source_bits_t

9.37.2 Defines

- #define LPMV2_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define LPMV2_CODE_VERSION_MINOR

  Initial value: `(3U)`
- #define LPMV2_ERROR_RETURN

  Initial value:`#define SSP_ERROR_RETURN((a), (err), &(g_lpmv2_module_name[0]), &g_lpmv2_version)`

9.37.3 User-defined macros

- #define LPMV2_CFG_PARAM_CHECKING_ENABLE

  Initial value: `(BSP_CFG_PARAM_CHECKING_ENABLE)`

  Specify whether to include code for API parameter checking. Valid settings include:BSP_CFG_PARAM_-CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h1 : Includes parameter checking0 : Compiles out parameter checking

9.37.4 API Data

9.37.4.1 lpmv2_snooze_request_t

`lpmv2_snooze_request_t`

Detailed description

Snooze request sources

Enumerated values

Name	Description
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request.
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request.

Name	Description
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request.
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request.
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request.
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request.
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request.
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request.
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request.

9.37.4.2 lpmv2_snooze_end_t

`lpmv2_snooze_end_t`

Detailed description

Snooze end control

Enumerated values

Name	Description
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly.
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion.
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match.
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch.
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch.
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match.
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch.

9.37.4.3 lpmv2_snooze_dtc_t

`lpmv2_snooze_dtc_t`

Detailed description

DTC Enable in Snooze Mode

Enumerated values

Name	Description
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation

9.37.4.4 lpmv2_standby_wake_source_t

`lpmv2_standby_wake_source_t`

Detailed description

Wake from standby mode sources, does not apply to sleep or deep standby modes

Enumerated values

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0.
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1.
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2.
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3.
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4.
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5.
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6.
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7.
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt.
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt.
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8.
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9.
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10.

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11.
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12.
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13.
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14.
LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15.
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt.

9.37.4.5 lpmv2_io_port_t

`lpmv2_io_port_t`

Detailed description

I/O port state after Deep Software Standby mode

Enumerated values

Name	Description
LPMV2_IO_PORT_RESET	When the Deep Software Standby mode is canceled, the I/O ports are in the reset state
LPMV2_IO_PORT_NO_CHANGE	When the Deep Software Standby mode is canceled, the I/O ports are in the same state as in the Deep Software Standby mode

9.37.4.6 lpmv2_power_supply_t

`lpmv2_power_supply_t`

Detailed description

Power supply control

Enumerated values

Name	Description
LPMV2_POWER_SUPPLY_DEEPCUT0	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is supplied in deep software standby mode
LPMV2_POWER_SUPPLY_DEEPCUT1	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode
LPMV2_POWER_SUPPLY_DEEPCUT3	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode. In addition, LVD is disabled and the low power function in a poweron reset circuit is enabled

9.37.4.7 lpmv2_deep_standby_cancel_edge_t

`lpmv2_deep_standby_cancel_edge_t`

Detailed description

Deep Standby Interrupt Edge

Enumerated values

Name	Description
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_EDGE_NONE	No options for a deep standby cancel source.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_RISING	IRQ0-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_FALLING	IRQ0-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_RISING	IRQ1-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_FALLING	IRQ1-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_RISING	IRQ2-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_FALLING	IRQ2-DS Pin Falling Edge.

Name	Description
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_RISING	IRQ3-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_FALLING	IRQ3-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_RISING	IRQ4-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_FALLING	IRQ4-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_RISING	IRQ5-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_FALLING	IRQ5-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_RISING	IRQ6-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_FALLING	IRQ6-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_RISING	IRQ7-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_FALLING	IRQ7-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_RISING	IRQ8-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_FALLING	IRQ8-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_RISING	IRQ9-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_FALLING	IRQ9-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_RISING	IRQ10-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_FALLING	IRQ10-DS Pin Falling Edge.

Name	Description
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_RISING	IRQ11-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_FALLING	IRQ11-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_RISING	IRQ12-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_FALLING	IRQ12-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_RISING	IRQ13-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_FALLING	IRQ13-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_RISING	LVD1 Rising Slope.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_FALLING	LVD1 Falling Slope.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_RISING	LVD2 Rising Slope.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_FALLING	LVD2 Falling Slope.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_RISING	NMI Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_FALLING	NMI Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_RISING	IRQ14-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_FALLING	IRQ14-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_RISING	IRQ15-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_FALLING	IRQ15-DS Pin Falling Edge.

9.37.4.8 lpmv2_deep_standby_cancel_source_t

`lpmv2_deep_standby_cancel_source_t`

Detailed description

Deep Standby cancel sources

Enumerated values

Name	Description
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RESET_ONLY	Cancel deep standby only by reset.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0	IRQ0.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1	IRQ1.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2	IRQ2.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3	IRQ3.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4	IRQ4.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5	IRQ5.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6	IRQ6.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7	IRQ7.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8	IRQ8.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9	IRQ9.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10	IRQ10.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11	IRQ11.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12	IRQ12.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13	IRQ13.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1	LVD1.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2	LVD2.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_INTERVAL	RTC Interval Interrupt.

Name	Description
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_ALARM	RTC Alarm Interrupt.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI	NMI.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBFS	USBFS Suspend/Resume.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_AGT1	AGT1 Underflow.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14	IRQ14.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBHS	USBHS Suspend/Resume.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15	IRQ15.

9.37.4.9 lpmv2_output_port_enable_t

```
lpmv2_output_port_enable_t
```

Detailed description

Output port enable

Enumerated values

Name	Description
LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode.
LPMV2_OUTPUT_PORT_ENABLE_RETAIN	1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state.

9.37.4.10 lpmv2_snooze_end_bits_t

```
typedef uint8_t lpmv2_snooze_end_bits_t
```

9.37.4.11 lpmv2_standby_wake_source_bits_t

```
typedef uint32_t lpmv2_standby_wake_source_bits_t
```

9.37.4.12 lpmv2_deep_standby_cancel_edge_bits_t

```
typedef uint32_t lpmv2_deep_standby_cancel_edge_bits_t
```

9.37.4.13 lpmv2_deep_standby_cancel_source_bits_t

```
typedef uint32_t lpmv2_deep_standby_cancel_source_bits_t
```

9.37.5 Extensions

9.37.5.1 lpmv2_mcu_cfg_t

lpmv2_mcu_cfg_t

Detailed description

S124 specific configuration structure

S128 specific configuration structure

S3A3 specific configuration structure

S3A6 specific configuration structure

S3A7 specific configuration structure

S5D5 specific configuration structure

S5D9 specific configuration structure

S7G2 specific configuration structure

Variables

- lpmv2_standby_wake_source_bits_t standby_wake_sources
  Bitwise list of sources to wake from standby
- lpmv2_snooze_request_t snooze_request_source
  Snooze request source
- lpmv2_snooze_end_bits_t snooze_end_sources
  Bitwise list of snooze end sources
- lpmv2_snooze_dtc_t dtc_state_in_snooze
  State of DTC in snooze mode, enabled or disabled
- lpmv2_output_port_enable_t output_port_enable
  Output port enabled/disabled in standby and deep standby
- lpmv2_io_port_t io_port_state
  IO port state in deep standby (maintained or reset)
- lpmv2_power_supply_t power_supply_state
  Internal power supply state in standby and deep standby (deepcut)

- lpmv2_deep_standby_cancel_source_bits_t deep_standby_cancel_source
  Sources that can trigger exit from deep standby
- lpmv2_deep_standby_cancel_edge_bits_t deep_standby_cancel_edge
  Signal edges for the sources that can trigger exit from deep standby

9.38 LPMV2 S7G2

Driver for Low Power Modes.
The LPMV2 driver supports low power modes deep standby, standby, sleep, and snooze.

NOTE: Not all low power modes are available on all MCU's.

9.38.1 Functions

- R_LPMV2_VersionGet
- R_LPMV2_Init
- R_LPMV2_LowPowerConfigure
- R_LPMV2_LowPowerModeEnter

9.38.2 Typedefs

- lpmv2_snooze_end_bits_t
- lpmv2_standby_wake_source_bits_t
- lpmv2_deep_standby_cancel_edge_bits_t
- lpmv2_deep_standby_cancel_source_bits_t

9.38.3 Defines

- #define LPMV2_CODE_VERSION_MAJOR
  Initial value: `(1U)`
- #define LPMV2_CODE_VERSION_MINOR
  Initial value: `(3U)`
- #define LPMV2_ERROR_RETURN
  Initial value:
```
#define SSP_ERROR_RETURN((a), (err), &(g_lpmv2_module_name[0]), &g_lpmv2_version)
```

9.38.4 User-defined macros

- #define LPMV2_CFG_PARAM_CHECKING_ENABLE

  Initial value: `(BSP_CFG_PARAM_CHECKING_ENABLE)`

  Specify whether to include code for API parameter checking. Valid settings include:BSP_CFG_PARAM_-CHECKING_ENABLE : Utilizes the system default setting from bsp_cfg.h1 : Includes parameter checking0 : Compiles out parameter checking

9.38.5 R_LPMV2_VersionGet

```
ssp_err_t R_LPMV2_VersionGet ( ssp_version_t  *const p_version )
```

9.38.5.1 Brief description

Get the driver version based on compile time macros.

9.38.5.2 Detailed description

Table 1684:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_INVALID_POINTER	p_version is NULL.

9.38.6 R_LPMV2_Init

```
ssp_err_t R_LPMV2_Init (  void  )
```

9.38.6.1 Brief description

Perform any necessary initialization.

9.38.6.2 Detailed description

Table 1685:Return values

Name	Description
SSP_SUCCESS	LPMV2 Software lock initialized

9.38.7 R_LPMV2_LowPowerConfigure

```
ssp_err_t R_LPMV2_LowPowerConfigure ( lpmv2_cfg_t  const *const p_cfg )
```

9.38.7.1 Brief description

Configure a low power mode.

9.38.7.2 Detailed description

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

Table 1686:Return values

Name	Description
SSP_SUCCESS	Low power mode successfuly applied
SSP_ERR_INVALID_POINTER	p_cfg is NULL
SSP_ERR_IN_USE	Lock was not acquired
SSP_ERR_INVALID_HW_CONDITION	Operating mode change transition was detected (OPCMTSF, SOPCMTSF bits)

9.38.8 R_LPMV2_LowPowerModeEnter

```
ssp_err_t R_LPMV2_LowPowerModeEnter (  void  )
```

9.38.8.1 Brief description

Enter low power mode (sleep/standby/deep standby) using WFI macro.

9.38.8.2 Detailed description

Function will return after waking from low power mode.

Table 1687:Return values

Name	Description
SSP_SUCCESS	Successful.

Table 1687:Return values (Continued)

Name	Description
SSP_ERR_INVALID_HW_CONDITION	HOCO was unstable during attempt to revert operating mode.

9.38.9 API Data

9.38.9.1 lpmv2_snooze_request_t

`lpmv2_snooze_request_t`

Detailed description

Snooze request sources

Enumerated values

Name	Description
LPMV2_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request.
LPMV2_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request.
LPMV2_SNOOZE_REQUEST_KEY	Enable KR snooze request.
LPMV2_SNOOZE_REQUEST_ACMPLP	Enable Low-speed analog comparator snooze request.
LPMV2_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request.
LPMV2_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request.

Name	Description
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request.
LPMV2_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request.
LPMV2_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request.
LPMV2_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request.
LPMV2_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request.

9.38.9.2 lpmv2_snooze_end_t

`lpmv2_snooze_end_t`

Detailed description

Snooze end control

Enumerated values

Name	Description
LPMV2_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly.
LPMV2_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion.
LPMV2_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion.
LPMV2_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match.
LPMV2_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch.

Name	Description
LPMV2_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch.
LPMV2_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match.
LPMV2_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch.

9.38.9.3 lpmv2_snooze_dtc_t

`lpmv2_snooze_dtc_t`

Detailed description

DTC Enable in Snooze Mode

Enumerated values

Name	Description
LPMV2_SNOOZE_DTC_DISABLE	Disable DTC operation
LPMV2_SNOOZE_DTC_ENABLE	Enable DTC operation

9.38.9.4 lpmv2_standby_wake_source_t

`lpmv2_standby_wake_source_t`

Detailed description

Wake from standby mode sources, does not apply to sleep or deep standby modes

Enumerated values

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ0	IRQ0.
LPMV2_STANDBY_WAKE_SOURCE_IRQ1	IRQ1.
LPMV2_STANDBY_WAKE_SOURCE_IRQ2	IRQ2.
LPMV2_STANDBY_WAKE_SOURCE_IRQ3	IRQ3.
LPMV2_STANDBY_WAKE_SOURCE_IRQ4	IRQ4.
LPMV2_STANDBY_WAKE_SOURCE_IRQ5	IRQ5.
LPMV2_STANDBY_WAKE_SOURCE_IRQ6	IRQ6.

Name	Description
LPMV2_STANDBY_WAKE_SOURCE_IRQ7	IRQ7.
LPMV2_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt.
LPMV2_STANDBY_WAKE_SOURCE_KEY	Key interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt.
LPMV2_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt.
LPMV2_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt.
LPMV2_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_IRQ8	IRQ8.
LPMV2_STANDBY_WAKE_SOURCE_IRQ9	IRQ9.
LPMV2_STANDBY_WAKE_SOURCE_IRQ10	IRQ10.
LPMV2_STANDBY_WAKE_SOURCE_IRQ11	IRQ11.
LPMV2_STANDBY_WAKE_SOURCE_IRQ12	IRQ12.
LPMV2_STANDBY_WAKE_SOURCE_IRQ13	IRQ13.
LPMV2_STANDBY_WAKE_SOURCE_IRQ14	IRQ14.
LPMV2_STANDBY_WAKE_SOURCE_IRQ15	IRQ15.
LPMV2_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt.
LPMV2_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt.
LPMV2_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt.

9.38.9.5 lpmv2_io_port_t

`lpmv2_io_port_t`

Detailed description

I/O port state after Deep Software Standby mode

Enumerated values

Name	Description
LPMV2_IO_PORT_RESET	When the Deep Software Standby mode is canceled, the I/O ports are in the reset state
LPMV2_IO_PORT_NO_CHANGE	When the Deep Software Standby mode is canceled, the I/O ports are in the same state as in the Deep Software Standby mode

9.38.9.6 lpmv2_power_supply_t

`lpmv2_power_supply_t`

Detailed description

Power supply control

Enumerated values

Name	Description
LPMV2_POWER_SUPPLY_DEEPCUT0	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is supplied in deep software standby mode
LPMV2_POWER_SUPPLY_DEEPCUT1	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode
LPMV2_POWER_SUPPLY_DEEPCUT3	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode. In addition, LVD is disabled and the low power function in a poweron reset circuit is enabled

9.38.9.7 lpmv2_deep_standby_cancel_edge_t

`lpmv2_deep_standby_cancel_edge_t`

Detailed description

Deep Standby Interrupt Edge

Enumerated values

Name	Description
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_EDGE_NONE	No options for a deep standby cancel source.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_RISING	IRQ0-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_FALLING	IRQ0-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_RISING	IRQ1-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_FALLING	IRQ1-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_RISING	IRQ2-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_FALLING	IRQ2-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_RISING	IRQ3-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_FALLING	IRQ3-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_RISING	IRQ4-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_FALLING	IRQ4-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_RISING	IRQ5-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_FALLING	IRQ5-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_RISING	IRQ6-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_FALLING	IRQ6-DS Pin Falling Edge.

Name	Description
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_RISING	IRQ7-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_FALLING	IRQ7-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_RISING	IRQ8-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_FALLING	IRQ8-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_RISING	IRQ9-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_FALLING	IRQ9-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_RISING	IRQ10-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_FALLING	IRQ10-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_RISING	IRQ11-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_FALLING	IRQ11-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_RISING	IRQ12-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_FALLING	IRQ12-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_RISING	IRQ13-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_FALLING	IRQ13-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_RISING	LVD1 Rising Slope.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1_FALLING	LVD1 Falling Slope.

Name	Description
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_RISING	LVD2 Rising Slope.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2_FALLING	LVD2 Falling Slope.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_RISING	NMI Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI_FALLING	NMI Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_RISING	IRQ14-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_FALLING	IRQ14-DS Pin Falling Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_RISING	IRQ15-DS Pin Rising Edge.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_FALLING	IRQ15-DS Pin Falling Edge.

9.38.9.8 lpmv2_deep_standby_cancel_source_t

`lpmv2_deep_standby_cancel_source_t`

Detailed description

Deep Standby cancel sources

Enumerated values

Name	Description
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RESET_ONLY	Cancel deep standby only by reset.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ0	IRQ0.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ1	IRQ1.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ2	IRQ2.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ3	IRQ3.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ4	IRQ4.

Name	Description
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ5	IRQ5.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ6	IRQ6.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ7	IRQ7.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ8	IRQ8.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ9	IRQ9.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ10	IRQ10.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ11	IRQ11.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ12	IRQ12.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ13	IRQ13.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD1	LVD1.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_LVD2	LVD2.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_INTERVAL	RTC Interval Interrupt.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_RTC_ALARM	RTC Alarm Interrupt.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_NMI	NMI.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBFS	USBFS Suspend/Resume.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_AGT1	AGT1 Underflow.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ14	IRQ14.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_USBHS	USBHS Suspend/Resume.
LPMV2_DEEP_STANDBY_CANCEL_SOURCE_IRQ15	IRQ15.

9.38.9.9 lpmv2_output_port_enable_t

`lpmv2_output_port_enable_t`

Detailed description

Output port enable

Enumerated values

Name	Description
LPMV2_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode.
LPMV2_OUTPUT_PORT_ENABLE_RETAIN	1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state.

9.38.9.10 lpmv2_snooze_end_bits_t

```
typedef uint8_t lpmv2_snooze_end_bits_t
```

9.38.9.11 lpmv2_standby_wake_source_bits_t

```
typedef uint32_t lpmv2_standby_wake_source_bits_t
```

9.38.9.12 lpmv2_deep_standby_cancel_edge_bits_t

```
typedef uint32_t lpmv2_deep_standby_cancel_edge_bits_t
```

9.38.9.13 lpmv2_deep_standby_cancel_source_bits_t

```
typedef uint32_t lpmv2_deep_standby_cancel_source_bits_t
```

9.38.10 Extensions

9.38.10.1 lpmv2_mcu_cfg_t

lpmv2_mcu_cfg_t

Detailed description

S124 specific configuration structure

S128 specific configuration structure

S3A3 specific configuration structure

S3A6 specific configuration structure

S3A7 specific configuration structure

S5D5 specific configuration structure

S5D9 specific configuration structure

S7G2 specific configuration structure

Variables

- lpmv2_standby_wake_source_bits_t standby_wake_sources
  Bitwise list of sources to wake from standby
- lpmv2_snooze_request_t snooze_request_source
  Snooze request source
- lpmv2_snooze_end_bits_t snooze_end_sources
  Bitwise list of snooze end sources
- lpmv2_snooze_dtc_t dtc_state_in_snooze
  State of DTC in snooze mode, enabled or disabled
- lpmv2_output_port_enable_t output_port_enable
  Output port enabled/disabled in standby and deep standby
- lpmv2_io_port_t io_port_state
  IO port state in deep standby (maintained or reset)
- lpmv2_power_supply_t power_supply_state
  Internal power supply state in standby and deep standby (deepcut)
- lpmv2_deep_standby_cancel_source_bits_t deep_standby_cancel_source
  Sources that can trigger exit from deep standby
- lpmv2_deep_standby_cancel_edge_bits_t deep_standby_cancel_edge
  Signal edges for the sources that can trigger exit from deep standby

9.39 LVD

Driver for Low Voltage Detection.
Implementation of the LVD API.For a detailed description see the Low Voltage Detection Interface.

9.39.1 Functions

- R_LVD_Open
- R_LVD_Close
- R_LVD_StatusGet
- R_LVD_StatusClear
- R_LVD_VersionGet
- lvd_common_isr_handler

- lvd_lvd_isr
- lvd_nmi_handler
- detection_response_configure
- detection_response_check
- negation_delay_check
- interrupt_type_configure
- lvd_open_parameter_check

9.39.2 Defines

- #define LVD_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define LVD_CODE_VERSION_MINOR

  Initial value: `(4U)`

9.39.3 R_LVD_Open

```
ssp_err_t R_LVD_Open ( lvd_ctrl_t  *const p_api_ctrl ,  lvd_cfg_t  const
*const p_cfg )
```

9.39.3.1 Brief description

Initializes a low voltage detection driver according to the passed in configuration structure. Enables an LVD peripheral based on configuration structure.

9.39.3.2 Detailed description

Implements

- open.

NOTE: Digital filter is not to be used with standby modes

Table 1688:Return values

Name	Description
SSP_SUCCESS	Successful

Table 1688:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	One or more pointers passed as function argument point to NULL.
SSP_ERR_ASSERTION	Requested configuration of a voltage monitor was invalid.
SSP_ERR_ASSERTION	Invalid detection response.
SSP_ERR_ASSERTION	Invalid voltage threshold.
SSP_ERR_ASSERTION	Invalid monitor number.
SSP_ERR_ASSERTION	Invalid sample clock configuration.
SSP_ERR_IN_USE	Unable to acquire the hardware lock.
SSP_ERR_INVALID_MODE	MOCO must be running for LVD_NEGATION_DELAY_FROM_RESET negation delay option

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet
- eventInfoGet

9.39.3.3 Function steps

- Select the detection voltage by setting the LVDLVLR.LVDnLVL[m:0] bits.
- Enable voltage detection LVCMPCR.LVDnE = 1
- Enable output of the results of comparison by voltage monitor LVDnCR0.CMPE = 1
- Mark driver as opened by initializing it to "LVD" in its ASCII equivalent.

9.39.4 R_LVD_Close

```
ssp_err_t R_LVD_Close ( lvd_ctrl_t  *const p_api_ctrl )
```

9.39.4.1 Brief description

Disables the LVD peripheral. Closes the driver instance.

9.39.4.2 Detailed description

Implements

- close.

Table 1689:Return values

Name	Description
SSP_SUCCESS	Successful
SSP_ERR_ASSERTION	Pointers passed as function argument point to NULL.
SSP_ERR_ASSERTION	Invalid monitor number

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- eventInfoGet

9.39.4.3 Function steps

- Disable voltage monitor comparison result output
- Disable reset/interrupt event
- Clear low voltage detection flag LVDnSR.DET = 0
- Disable digital filtering
- Disable voltage monitor

9.39.5 R_LVD_StatusGet

```
ssp_err_t R_LVD_StatusGet ( lvd_ctrl_t  *const p_api_ctrl ,  lvd_status_t
* p_lvd_status )
```

9.39.5.1 Brief description

Get the current state of the monitor, (threshold crossing detected, voltage currently within range)

9.39.5.2 Detailed description

Implements

- statusGet.

Table 1690:Return values

Name	Description
SSP_SUCCESS	Successful

Table 1690:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	One or more pointers passed as function argument point to NULL, invalid LVD monitor number
SSP_ERR_NOT_OPEN	Driver is not open, status will not be valid

9.39.6 R_LVD_StatusClear

```
ssp_err_t R_LVD_StatusClear ( lvd_ctrl_t  *const p_api_ctrl )
```

9.39.6.1 Brief description

Clears the latched status of the monitor.

9.39.6.2 Detailed description

Implements

- statusClear.

Table 1691:Return values

Name	Description
SSP_SUCCESS	Successful
SSP_ERR_ASSERTION	Pointers passed as function argument point to NULL, invalid LVD monitor number
SSP_ERR_NOT_OPEN	Driver is not open, status will not be valid

9.39.7 R_LVD_VersionGet

```
ssp_err_t R_LVD_VersionGet ( ssp_version_t  *const p_version )
```

9.39.7.1 Brief description

Returns the LVD driver version based on compile time macros.

9.39.7.2 Detailed description

Implements

- versionGet.

Table 1692:Return values

Name	Description
SSP_SUCCESS	Successful
SSP_ERR_ASSERTION	p_version was NULL

9.39.8 lvd_common_isr_handler

```
lvd_common_isr_handler ( lvd_instance_ctrl_t  * p_ctrl ,
 uint32_t monitor_number )
```

9.39.8.1 Brief description

Common code needed for all LVD ISRs.

9.39.8.2 Detailed description

Table 1693:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block structure for the driver instance
monitor_number	in	Identifier for the monitor which generated the interrupt

NOTE: Do not call this function directly. To be used internally by LVD driver.

9.39.8.3 Function steps

- If a callback is specified, call it

9.39.9 lvd_lvd_isr

```
lvd_lvd_isr (  void  )
```

9.39.9.1 Brief description

ISR for LVD.

9.39.9.2 Detailed description

NOTE: Do not call this function directly. To be used internally by LVD driver.

9.39.9.3 Function steps

- Save context if RTOS is used
- Clear the Interrupt Request
- Restore context if RTOS is used

9.39.10 lvd_nmi_handler

```
lvd_nmi_handler (  bsp_grp_irq_t irq )
```

9.39.10.1 Brief description

NMI ISR handler for LVD 1 and 2.

9.39.10.2 Detailed description

Table 1694:Parameters

Name	Direction	Description
irq	in	BSP group IRQ identifier

NOTE: Do not call this function directly. To be used internally by LVD driver.

9.39.10.3 Function steps

- Save context if RTOS is used
- Restore context if RTOS is used

9.39.11 detection_response_configure

```
 detection_response_configure ( lvd_instance_ctrl_t  * p_ctrl ,  lvd_cfg_t  const
*const p_cfg ,   IRQn_Type irq )
```

9.39.11.1 Brief description

Configure LVD monitor detection response. Internal function, do not use directly.

9.39.11.2 Detailed description

Table 1695:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block structure for the driver instance
p_cfg	in	Pointer to the configuration structure for the driver instance
irq	in	IRQ associated with the monitor

NOTE: Do not call this function directly. To be used internally by LVD driver.

9.39.12 detection_response_check

```
ssp_err_t detection_response_check ( lvd_cfg_t  const *const p_cfg )
```

9.39.12.1 Brief description

Verify the detection response. Internal function, do not use directly.

9.39.12.2 Detailed description

Table 1696:Parameters

Name	Direction	Description
p_cfg	in	Pointer to the configuration structure for the driver instance

Table 1697:Return values

Name	Description
SSP_SUCCESS	detection response configuration is valid
SSP_ERR_ASSERTION	detection response configuration is invalid

9.39.13 negation_delay_check

```
ssp_err_t negation_delay_check ( lvd_cfg_t  const *const p_cfg )
```

9.39.13.1 Brief description

Verify the LVD signal negation delay. Internal function, do not use directly.

9.39.13.2 Detailed description

Table 1698:Parameters

Name	Direction	Description
p_cfg	in	Pointer to the configuration structure for the driver instance

Table 1699:Return values

Name	Description
SSP_SUCCESS	signal negation delay is valid
SSP_ERR_ASSERTION	signal negation delay is invalid

9.39.14 interrupt_type_configure

```
 interrupt_type_configure ( lvd_instance_ctrl_t  * p_ctrl ,  lvd_cfg_t  const
*const p_cfg )
```

9.39.14.1 Brief description

Configure the interrupt response type. Internal function, do not use directly.

9.39.14.2 Detailed description

Table 1700:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block structure for the driver instance
p_cfg	in	Pointer to the configuration structure for the driver instance

Table 1701:Return values

Name	Description
none	

9.39.15 lvd_open_parameter_check

```
ssp_err_t lvd_open_parameter_check ( lvd_instance_ctrl_t  * p_ctrl ,  lvd_cfg_t
const *const p_cfg )
```

9.39.15.1 Brief description

Helper function to do parameter checking for the Open function for LVD module.

9.39.15.2 Detailed description

Table 1702:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to the control block structure for the driver instance
p_cfg	in	Pointer to the configuration structure for the driver instance

Table 1703:Return values

Name	Description
SSP_SUCCESS	If all parameter checks are success
SSP_ERR_ASSERTION	If any of the parameter checks fails
SSP_ERR_INVALID_MODE	If the attempted mode is invalid for this configuration

9.39.16 API Data

9.39.16.1 lvd_sample_clock_t

`lvd_sample_clock_t`

Detailed description

Sample clock divider, use LVD_SAMPLE_CLOCK_DISABLED to disable digital filtering

Enumerated values

Name	Description
LVD_SAMPLE_CLOCK_LOCO_DIV_2	Digital filter sample clock is LOCO divided by 2.
LVD_SAMPLE_CLOCK_LOCO_DIV_4	Digital filter sample clock is LOCO divided by 4.
LVD_SAMPLE_CLOCK_LOCO_DIV_8	Digital filter sample clock is LOCO divided by 8.
LVD_SAMPLE_CLOCK_LOCO_DIV_16	Digital filter sample clock is LOCO divided by 16.
LVD_SAMPLE_CLOCK_DISABLED	Digital filter is disabled.

9.39.16.2 lvd_negation_delay_t

`lvd_negation_delay_t`

Detailed description

Negation of LVD signal follows reset or voltage in range

Enumerated values

Name	Description
LVD_NEGATION_DELAY_FROM_VOLTAGE	Negation follows a stabilization time (tLVDn) after VCC > Vdet1 is detected. If a transition to software standby or deep software standby is to be made, the only possible value for the RN bit is LVD_NEGATION_DELAY_FROM_VOLTAGE
LVD_NEGATION_DELAY_FROM_RESET	Negation follows a stabilization time (tLVDn) after assertion of the LVDn reset. If a transition to software standby or deep software standby is to be made, the only possible value for the RN bit is LVD_NEGATION_DELAY_FROM_VOLTAGE

9.39.17 Extensions

9.39.17.1 lvd_instance_ctrl_t

lvd_instance_ctrl_t

Detailed description

LVD instance control structure

Variables

- uint32_t monitor_number

  Monitor number, 1, 2, ...
- R_SYSTEM_Type * p_reg

  Pointer to LVD register base address.
- void(* p_callback)(*p_args)
- lvd_callback_args_t lvd_callback_args
- uint32_t opened

9.39.17.2 lvd_extend_t

lvd_extend_t

Detailed description

Hardware extend structure

Variables

- lvd_negation_delay_t negation_delay

  Negation of LVD signal follows reset or voltage in range
- lvd_sample_clock_t sample_clock_divisor

  Sample clock divider, use LVD_SAMPLE_CLOCK_DISABLED to disable digital filtering

9.40 Operational Amplifier (OPAMP)

Driver for the Operational Amplifier (OPAMP).

This module supports the OPAMP peripheral. It implements the following interfaces:

- OPAMP Interface

9.40.1 Functions

- R_OPAMP_Open
- R_OPAMP_InfoGet
- R_OPAMP_Start
- R_OPAMP_Stop
- R_OPAMP_StatusGet
- R_OPAMP_Trim
- R_OPAMP_Close
- R_OPAMP_VersionGet

9.40.2 Defines

- #define OPAMP_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file
- #define OPAMP_CODE_VERSION_MINOR

  Initial value: `(0U)`
- #define OPAMP_MASK_CHANNEL_0

  Initial value: `(1U << 0)`
- #define OPAMP_MASK_CHANNEL_1

  Initial value: `(1U << 1)`

- #define OPAMP_MASK_CHANNEL_2

  Initial value: `(1U << 2)`
- #define OPAMP_MASK_CHANNEL_3

  Initial value: `(1U << 3)`

9.40.3 R_OPAMP_Open

```
ssp_err_t R_OPAMP_Open ( opamp_ctrl_t  *const p_api_ctrl ,  opamp_cfg_t  const
*const p_cfg )
```

9.40.3.1 Detailed description

Applies power to the OPAMP and initializes the hardware based on the user configuration. Implements open.

The op-amp is not operational until the start is called. If the op-amp is configured to start after AGT compare match, the op-amp is not operational until start and the associated AGT compare match event occurs.

Some MCUs have switches that must be set before starting the op-amp. These switches must be set in the application code after open and before start.

Table 1704:Return values

Name	Description
SSP_SUCCESS	Configuration successful.
SSP_ERR_ASSERTION	An input pointer is NULL.
SSP_ERR_INVALID_ARGUMENT	An input argument is invalid.
SSP_ERR_IN_USE	Control block is already opened.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet

9.40.3.2 Function steps

- Verify the control block has not already been initialized.
- Confirm the OPAMP exists on this MCU.
- Lock the OPAMP
- Initialize the hardware based on the configuration.
- Stop operation of all op-amps.
- Initialize trim registers to factory trim values.

9.40.4 R_OPAMP_InfoGet

```
ssp_err_t R_OPAMP_InfoGet ( opamp_ctrl_t  *const p_api_ctrl ,  opamp_info_t
*const p_info )
```

9.40.4.1 Detailed description

Provides the minimum stabilization wait time in microseconds. Implements infoGet.

Table 1705:Return values

Name	Description
SSP_SUCCESS	Information stored in p_info.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.40.4.2 Function steps

- Perform parameter checking

9.40.5 R_OPAMP_Start

```
ssp_err_t R_OPAMP_Start ( opamp_ctrl_t  *const p_api_ctrl ,   uint32_t
const channel_mask )
```

9.40.5.1 Detailed description

If the OPAMP is configured for hardware triggers, enables hardware triggers. Otherwise, starts the op-amp. Implements start.

Some MCUs have switches that must be set before starting the op-amp. These switches must be set in the application code after open and before start.

Table 1706:Return values

Name	Description
SSP_SUCCESS	Op-amp started or hardware triggers enabled successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

Table 1706:Return values (Continued)

Name	Description
SSP_ERR_INVALID_ARGUMENT	channel_mask includes a channel that does not exist on this MCU.

9.40.5.2 Function steps

- Enable AGT start and ADC conversion end triggers or start the op-amp channel(s).
- Return the error code

9.40.6 R_OPAMP_Stop

```
ssp_err_t R_OPAMP_Stop ( opamp_ctrl_t  *const p_api_ctrl ,   uint32_t
const channel_mask )
```

9.40.6.1 Detailed description

Stops the op-amp. If the OPAMP is configured for hardware triggers, disables hardware triggers. Implements stop.

Table 1707:Return values

Name	Description
SSP_SUCCESS	Op-amp stopped or hardware triggers disabled successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.
SSP_ERR_INVALID_ARGUMENT	channel_mask includes a channel that does not exist on this MCU.

9.40.6.2 Function steps

- Disable AGT start and ADC conversion end triggers and stop the op-amp channel(s).
- Return the error code

9.40.7 R_OPAMP_StatusGet

```
ssp_err_t R_OPAMP_StatusGet ( opamp_ctrl_t  *const p_api_ctrl ,  opamp_status_t
*const p_status )
```

9.40.7.1 Detailed description

Provides the operating status for each op-amp in a bitmask. This bit is set when operation begins, before the stabilization wait time has elapsed. Implements statusGet.

Table 1708:Return values

Name	Description
SSP_SUCCESS	Operating status of each op-amp provided in p_status.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.40.7.2 Function steps

- Read the operating status of the op-amps.

9.40.8 R_OPAMP_Trim

```
ssp_err_t R_OPAMP_Trim ( opamp_ctrl_t  *const p_api_ctrl ,  opamp_trim_cmd_t
const cmd ,  opamp_trim_args_t  const *const p_args )
```

9.40.8.1 Detailed description

On MCUs that support trimming, the op-amp trim register is set to the factory default after open(). This function allows the application to trim the operational amplifier to a user setting, which overwrites the factory default factory trim values.

Not supported on all MCUs. See hardware manual for details. Not supported if configured for low power mode (OPAMP_MODE_LOW_POWER).

This function is not reentrant. Only one side of one op-amp can be trimmed at a time. Complete the procedure for one side of one channel before calling trim() with command OPAMP_TRIM_CMD_START again.

Implements trim.

The trim procedure works as follows: Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_START.

Connect a fixed voltage to the Pch (+) input.

Connect the Nch (-) input to the op-amp output to create a voltage follower.

Ensure the op-amp is operating and stabilized.

Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_START.

Measure the fixed voltage connected to the Pch (+) input using the SAR ADC and save the value (referred to as A later in this procedure). Iterate over the following loop 5 times:

Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_NEXT_STEP.

Measure the op-amp output using the SAR ADC (referred to as B in the next step).

If A <= B, call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_CLEAR_BIT.
Call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_START.
Measure the fixed voltage connected to the Pch (+) input using the SAR ADC and save the value (referred to as A later in this procedure). Iterate over the following loop 5 times:
Call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_NEXT_STEP.
Measure the op-amp output using the SAR ADC (referred to as B in the next step).
If A <= B, call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_CLEAR_BIT.

Table 1709:Return values

Name	Description
SSP_SUCCESS	Conversion result in p_data.
SSP_ERR_UNSUPPORTED	Trimming is not supported on this MCU.
SSP_ERR_INVALID_STATE	The command is not valid in the current state of the trim state machine.
SSP_ERR_INVALID_ARGUMENT	The requested channel is not operating or the trim procedure is not in progress for this channel/input combination.
SSP_ERR_INVALID_MODE	Trim is not allowed in low power mode.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.40.8.2 Function steps

- Initialize the trim register to 0 during OPAMP_TRIM_CMD_START.
- Set the next trim bit during OPAMP_TRIM_CMD_NEXT_STEP.
- Clear the current trim bit during OPAMP_TRIM_CMD_CLEAR_BIT.

9.40.9 R_OPAMP_Close

```
ssp_err_t R_OPAMP_Close ( opamp_ctrl_t  *const p_api_ctrl )
```

9.40.9.1 Detailed description

Stops the op-amps. Implements close.

Table 1710:Return values

Name	Description
SSP_SUCCESS	Instance control block closed successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.40.9.2 Function steps

- Perform parameter checking
- Set all OPAMP units and the reference current generator to be stopped.
- Mark driver as closed
- Enter the module-stop state.
- Release the hardware lock
- Return the error code

9.40.10 R_OPAMP_VersionGet

```
ssp_err_t R_OPAMP_VersionGet ( ssp_version_t  *const p_version )
```

9.40.10.1 Detailed description

Gets the API and code version. Implements versionGet.

Table 1711:Return values

Name	Description
SSP_SUCCESS	Version information available in p_version.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

9.40.10.2 Function steps

- Return the version number

9.40.11 API Data

9.40.11.1 opamp_trigger_t

`opamp_trigger_t`

Detailed description

Start and stop trigger for the op-amp.

Enumerated values

Name	Description
OPAMP_TRIGGER_SOFTWARE_START_SOFTWARE_STOP	Start and stop with APIs.
OPAMP_TRIGGER_AGT_START_SOFTWARE_STOP	Start by AGT compare match and stop with API.
OPAMP_TRIGGER_AGT_START_ADC_STOP	Start by AGT compare match and stop after ADC conversion.

9.40.11.2 opamp_agt_link_t

`opamp_agt_link_t`

Detailed description

Which AGT timer starts the op-amp. Only applies to channels if OPAMP_TRIGGER_AGT_START_SOFTWARE_STOP or OPAMP_TRIGGER_AGT_START_ADC_STOP is selected for the channel. If OPAMP_TRIGGER_SOFTWARE_START_SOFTWARE_STOP is selected for a channel, then no AGT compare match event will start that op-amp channel.

Enumerated values

Name	Description
OPAMP_AGT_LINK_AGT1_OPAMP_0_2_AGT0_OPAMP_1_3	OPAMP channel 0 and 2 are started by AGT1 compare match. OPAMP channel 1 and 3 are started by AGT0 compare match.
OPAMP_AGT_LINK_AGT1_OPAMP_0_1_AGT0_OPAMP_2_3	OPAMP channel 0 and 1 are started by AGT1 compare match. OPAMP channel 2 and 3 are started by AGT0 compare match.
OPAMP_AGT_LINK_AGT1_OPAMP_0_1_2_3	All OPAMP channels are started by AGT1 compare match.

9.40.11.3 opamp_mode_t

opamp_mode_t

Detailed description

Op-amp mode.

Enumerated values

Name	Description
OPAMP_MODE_LOW_POWER	Low power mode.
OPAMP_MODE_MIDDLE_SPEED	Middle speed mode (not supported on all MCUs)
OPAMP_MODE_HIGH_SPEED	High speed mode.

9.40.11.4 opamp_priv_trim_state_t

opamp_priv_trim_state_t

Enumerated values

Name	Description
OPAMP_PRIV_TRIM_STATE_INVALID	
OPAMP_PRIV_TRIM_STATE_END	
OPAMP_PRIV_TRIM_STATE_BIT_0	
OPAMP_PRIV_TRIM_STATE_BIT_1	
OPAMP_PRIV_TRIM_STATE_BIT_2	
OPAMP_PRIV_TRIM_STATE_BIT_3	
OPAMP_PRIV_TRIM_STATE_BIT_4	
OPAMP_PRIV_TRIM_STATE_BEGIN	

9.40.12 Extensions

9.40.12.1 opamp_on_opamp_cfg_t

opamp_on_opamp_cfg_t

Detailed description

OPAMP configuration extension. This extension is required and must be provided in p_extend.

Variables

- opamp_agt_link_t agt_link

  Configure which AGT links are paired to which channel. Only applies to channels if OPAMP_TRIGGER_AGT_START_SOFTWARE_STOP or OPAMP_TRIGGER_AGT_START_ADC_STOP is selected for the channel.

- opamp_mode_t mode

  Low power, middle speed, or high speed mode.

- opamp_trigger_t trigger_channel_0

  Start and stop triggers for channel 0.

- opamp_trigger_t trigger_channel_1

  Start and stop triggers for channel 1.

- opamp_trigger_t trigger_channel_2

  Start and stop triggers for channel 2.

- opamp_trigger_t trigger_channel_3

  Start and stop triggers for channel 3.

9.40.12.2 opamp_instance_ctrl_t

opamp_instance_ctrl_t

Detailed description

OPAMP instance control block. DO NOT INITIALIZE. Initialized in open.

Variables

- R_OPAMP_Type * p_reg
- uint32_t opened
- uint8_t trim_capable
- uint8_t switches
- uint32_t valid_opamps
- opamp_priv_trim_state_t trim_state
- uint8_t trim_channel
- opamp_trim_input_t trim_input

9.41 PDC

Driver for the Parallel Data Capture Unit (PDC).

9.41.1 Summary

extends PDC Interface The PDC interface allows the capturing of an image from a camera module.

9.41.2 Functions

- R_PDC_Open
- R_PDC_Close
- R_PDC_CaptureStart
- R_PDC_StateGet
- R_PDC_VersionGet

9.41.3 Defines

- #define PDC_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define PDC_CODE_VERSION_MINOR

  Initial value: `(4U)`

9.41.4 R_PDC_Open

```
ssp_err_t R_PDC_Open ( pdc_ctrl_t  *const p_api_ctrl ,  pdc_cfg_t  const
*const p_cfg )
```

9.41.4.1 Brief description

Powers on PDC, handles required initialization described in the hardware manual. Implements open.

9.41.4.2 Detailed description

The Open function provides initial configuration for the PDC module. It powers on the module and enables the PCLKO output and the PIXCLK input. Further initialization requires the PIXCLK input to be running in order to be able to reset the PDC as part of its initialization. This clock is input from a camera module and so the reset and further initialization is performed in captureStart. This function should be called once prior to calling any other PDC API functions. After the PDC is opened the Open function should not be called again without first calling the Close function.

Table 1712:Return values

Name	Description
SSP_SUCCESS	Initialization was successful.
SSP_ERR_ASSERTION	One of the following parameters is incorrect. Either • p_cfg is NULL, OR • p_api_ctrl is NULL, OR • The pointer to the transfer interface in the p_cfg parameter is NULL
SSP_ERR_INVALID_ARGUMENT	One of the following configuration parameters is incorrect. Either • bytes_per_pixel is zero, OR • x_capture_pixels is zero, OR • y_capture_pixels is zero, OR • x_capture_start_pixel + x_capture_pixels is greater than 4095, OR • y_capture_start_pixel + y_capture_pixels is greater than 4095
SSP_ERR_HW_LOCKED	Unable to reserve BSP hardware lock for this module.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet
- eventInfoGet

NOTE: This function is not reentrant.

9.41.4.3 Function steps

- Lock the PDC Hardware Resource
- Disable module stop mode for PDC
- Set PCLKB divider
- Enable PCLKO output
- Enable the PIXCLK input
- Mark driver as open by initializing it to "PDC" in its ASCII equivalent

9.41.5 R_PDC_Close

```
ssp_err_t R_PDC_Close ( pdc_ctrl_t  *const p_api_ctrl )
```

9.41.5.1 Brief description

Stops and closes the transfer interface, disables the PDC, powers off the PDC, clears internal driver data and disables interrupts. Implements close.

9.41.5.2 Detailed description

Table 1713:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	One of the following parameters is incorrect. Either • p_api_ctrl is NULL, OR • low level transfer is not assigned, OR • low level transfer APIs are not assigned
SSP_ERR_NOT_OPEN	Open has not been successfully called.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- disable
- close

NOTE: This API will close the PDC driver. If a capture is in progress it will be stopped. This function is reentrant.

9.41.5.3 Function steps

- Disable all interrupts.
- Enable module stop mode for PDC
- Unlock the PDC Hardware Resource

9.41.6 R_PDC_CaptureStart

```
ssp_err_t R_PDC_CaptureStart ( pdc_ctrl_t  *const p_api_ctrl ,   uint8_t
*const p_buffer )
```

9.41.6.1 Brief description

Starts a capture. Enables interrupts. Implements captureStart.

9.41.6.2 Detailed description

Sets up the transfer interface to transfer data from the PDC into the specified buffer. Configures the PDC settings as previously set by the open API. These settings are configured here as the PIXCLK input must be active for the PDC reset operation. When a capture is complete the callback registered during open API call will be called.

Table 1714:Return values

Name	Description
SSP_SUCCESS	Capture start successful.
SSP_ERR_ASSERTION	One of the following parameters is incorrect. Either • p_api_ctrl is NULL, OR • low level transfer is not assigned, OR • low level transfer APIs are not assigned • buffer is not assigned, assign buffer
SSP_ERR_NOT_OPEN	Open has not been successfully called.
SSP_ERR_IN_USE	Pdc transfer is already in progress, wait for transfer to complete.
SSP_ERR_TIMEOUT	Reset operation timed out.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- open
- enable

NOTE: If the PIXCLK is being generated by a camera module the camera must be configured after the call to open and before the call to captureStart. This function is not reentrant.

The user is responsible to ensuring that the memory pointed to by p_buffer is both valid and large enough to store a complete image. The amount of space required, in bytes can be calculated as shown:

size (bytes) = image width (pixels) * image height (lines) * number of bytes per pixel

9.41.6.3 Function steps

- Set up transfer interface

- Configure the transfer interface
- Open transfer interface
- Wait for reset to complete
- Set horizontal capture range
- Set horizontal capture size
- Set vertical capture range
- Set vertical capture size
- Set VSYNC polarity
- Set HSYNC polarity
- Set endianess of capture data
- Enable interrupts: Receive data ready interrupt, Underrun interrupt, Overrun interrupt, Frame end interrupt, Vertical line number setting error interrupt, Horizontal byte number setting error interrupt

9.41.7 R_PDC_StateGet

```
ssp_err_t R_PDC_StateGet ( pdc_ctrl_t  *const p_api_ctrl ,  pdc_state_t
* p_state )
```

9.41.7.1 Brief description

Returns the state of the VSYNC and HSYNC pins.Implements stateGet.

9.41.7.2 Detailed description

Table 1715:Return values

Name	Description
SSP_SUCCESS	State read successful.
SSP_ERR_ASSERTION	p_api_ctrl is NULL OR p_state is NULL
SSP_ERR_NOT_OPEN	Open has not been successfully called.

NOTE: This function is reentrant.

9.41.8 R_PDC_VersionGet

```
ssp_err_t R_PDC_VersionGet ( ssp_version_t  *const p_data )
```

9.41.8.1 Brief description

Return PDC HAL driver version. Implements versionGet.

9.41.8.2 Detailed description

Table 1716:Return values

Name	Description
SSP_SUCCESS	Version information successfully read.
SSP_ERR_ASSERTION	Null pointer passed as a parameter

NOTE: This function is reentrant.

9.41.9 Extensions

9.41.9.1 pdc_instance_ctrl_t

pdc_instance_ctrl_t

Detailed description

PDC instance control block. DO NOT INITIALIZE. Initialization occurs when open is called.

Variables

- R_PDC_Type * p_reg

  Pointer to PDC base address.
- uint32_t open

  Indicates whether or not the driver is open called.
- uint8_t bytes_per_pixel

  Number of bytes per pixel.
- uint16_t x_resolution_pixels

  Total number of horizontal pixels input to PDC.

- uint16_t y_resolution_pixels

  Total number of lines input to the PDC.
- uint16_t x_capture_start_pixel

  Horizontal position to start capture.
- uint16_t x_capture_pixels

  Number of horizontal pixels to capture.
- uint16_t y_capture_start_pixel

  Vertical position to start capture.
- uint16_t y_capture_pixels

  Number of vertical lines/pixels to capture.
- pdc_endian_t endian

  Endian of capture data.
- pdc_hsync_polarity_t hsync_polarity

  Polarity of HSYNC input.
- pdc_vsync_polarity_t vsync_polarity

  Polarity of VSYNC input.
- uint8_t * p_current_buffer

  Pointer to buffer currently in use.
- bool transfer_in_progress

  Indicates if a PDC transfer is already in progress.
- transfer_instance_t const * p_lower_lvl_transfer

  Pointer to the transfer instance the PDC should use.
- transfer_info_t info_transfer

  Transfer info structure for low level Transfer interface.
- void const * p_context

  Placeholder for user data. Passed to the user callback in pdc_callback_args_t.
- void(* p_callback)(*p_args)

  Callback provided when a PDC transfer ISR occurs.
- IRQn_Type frame_end_irq

  Frame end interrupt number.
- IRQn_Type irq

  PDC interrupt number.

9.42 QSPI

Driver for the Quad Serial Peripheral Interface (QSPI).

This is a driver for the Quad-SPI module (QSPI) which is a memory controller for connecting a serial ROM (non-volatile memory such as a serial flash memory, serial EEPROM, or serial FeRAM) that has an SPI-compatible interface.

9.42.1 Summary

Extends Quad SPI Flash Interface.

9.42.2 Functions

- R_QSPI_Open
- R_QSPI_Close
- R_QSPI_Read
- R_QSPI_PageProgram
- R_QSPI_Erase
- R_QSPI_InfoGet
- R_QSPI_SectorErase
- R_QSPI_StatusGet
- R_QSPI_BankSelect
- R_QSPI_VersionGet

9.42.3 Variables

- g_qspi_on_qspi

9.42.4 Defines

- #define QSPI_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define QSPI_CODE_VERSION_MINOR

  Initial value: `(6U)`

9.42.5 R_QSPI_Open

```
ssp_err_t R_QSPI_Open ( qspi_ctrl_t * p_api_ctrl , qspi_cfg_t const
*const p_cfg )
```

9.42.5.1 Brief description

Open the QSPI driver module.

9.42.5.2 Detailed description

Open the QSPI module driver in direct communication mode for the purposes of reading and writing flash memory via SPI protocols.

Table 1717:Return values

Name	Description
SSP_SUCCESS	Configuration was successful.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_cfg is NULL.
SSP_ERR_UNSUPPORTED	Driver not able to query the following information from the flash manufacturer id,memory capacity and memory type.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet

9.42.6 R_QSPI_Close

```
ssp_err_t R_QSPI_Close ( qspi_ctrl_t  * p_api_ctrl )
```

9.42.6.1 Brief description

Close the QSPI driver module.

9.42.6.2 Detailed description

Return the QSPI module back to ROM access mode.

Table 1718:Return values

Name	Description
SSP_SUCCESS	Configuration was successful.
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Driver is not opened.

9.42.7 R_QSPI_Read

```
ssp_err_t R_QSPI_Read ( qspi_ctrl_t  * p_api_ctrl ,   uint8_t
* p_device_address ,   uint8_t * p_memory_address ,   uint32_t byte_count )
```

9.42.7.1 Brief description

Read data from the flash.

9.42.7.2 Detailed description

Read a block of data from a particular address on the SPI flash device.

Table 1719:Return values

Name	Description
SSP_SUCCESS	The flash was programmed successfully.
SSP_ERR_ASSERTION	p_ctrl,p_device_address or p_memory_address is NULL.
SSP_ERR_NOT_OPEN	Driver is not opened.
SSP_ERR_TRANSFER_BUSY	Another serial communications transfer is in progress.

9.42.8 R_QSPI_PageProgram

```
ssp_err_t R_QSPI_PageProgram ( qspi_ctrl_t  * p_api_ctrl ,   uint8_t
* p_device_address ,   uint8_t * p_memory_address ,   uint32_t byte_count )
```

9.42.8.1 Brief description

Program a page of data to the flash.

9.42.8.2 Detailed description

Table 1720:Return values

Name	Description
SSP_SUCCESS	The flash was programmed successfully.
SSP_ERR_ASSERTION	p_ctrl,p_device_address or p_memory_address is NULL.

Table 1720:Return values (Continued)

Name	Description
SSP_ERR_INVALID_ARGUMENT	Invalid parameter is passed.
SSP_ERR_NOT_OPEN	Driver is not opened.

9.42.9 R_QSPI_Erase

```
ssp_err_t R_QSPI_Erase ( qspi_ctrl_t  * p_api_ctrl ,   uint8_t
* p_device_address ,   uint32_t byte_count )
```

9.42.9.1 Brief description

Erase a number of byte from the flash.

9.42.9.2 Detailed description

Table 1721:Return values

Name	Description
SSP_SUCCESS	The command to erase the flash was executed successfully.
SSP_ERR_ASSERTION	p_ctrl or p_device_address is NULL.
SSP_ERR_INVALID_ARGUMENT	Invalid byte_count entered.
SSP_ERR_NOT_OPEN	Driver is not opened.

9.42.9.3 Function steps

- Get the information of underline flash
- If requested byte_count is supported by underline flash, assign the value of size_index to cmd_index for searching the command
- Send command to enable writing
- Close the SPI bus cycle
- Get the erase command
- Send command to erase
- Send command to write data

- Close the SPI bus cycle
- Send command to disable writing
- Close the SPI bus cycle

9.42.10 R_QSPI_InfoGet

```
ssp_err_t R_QSPI_InfoGet ( qspi_ctrl_t  * p_api_ctrl ,  qspi_info_t
*const p_info )
```

9.42.10.1 Brief description

Returns the information about the flash.

9.42.10.2 Detailed description

Table 1722:Return values

Name	Description
SSP_SUCCESS	
SSP_ERR_ASSERTION	p_ctrl or p_info is NULL.
SSP_ERR_NOT_OPEN	Driver is not opened.

9.42.10.3 Function steps

- Get the information of underline flash

9.42.11 R_QSPI_SectorErase

```
ssp_err_t R_QSPI_SectorErase ( qspi_ctrl_t  * p_api_ctrl ,   uint8_t
* p_device_address )
```

9.42.11.1 Brief description

Erase a sector on the flash.

9.42.11.2 Detailed description

Erase one sector on the SPI flash device. Any passed in address within the sector to be erased is acceptable.

Table 1723:Return values

Name	Description
SSP_SUCCESS	The command to erase the sector of flash was executed successfully.
SSP_ERR_ASSERTION	p_ctrl or p_device_address is NULL.
SSP_ERR_NOT_OPEN	Driver is not opened.

9.42.12 R_QSPI_StatusGet

```
ssp_err_t R_QSPI_StatusGet ( qspi_ctrl_t  * p_api_ctrl ,   bool
* p_write_in_progress )
```

9.42.12.1 Brief description

Get the write or erase status of the flash.

9.42.12.2 Detailed description

Return the write status of the flash. This is most useful for determining if erases are complete.

Table 1724:Return values

Name	Description
SSP_SUCCESS	The write status is correct.
SSP_ERR_ASSERTION	p_ctrl or p_write_in_progress is NULL.
SSP_ERR_NOT_OPEN	Driver is not opened.

9.42.13 R_QSPI_BankSelect

```
ssp_err_t R_QSPI_BankSelect (  uint32_t bank )
```

9.42.13.1 Brief description

Select the bank to access.

9.42.13.2 Detailed description

A bank is a 64MB sliding access window into the flash memory space. This function sets the current bank.

Table 1725:Return values

Name	Description
SSP_SUCCESS	Bank successfully selected.

9.42.14 R_QSPI_VersionGet

```
ssp_err_t R_QSPI_VersionGet ( ssp_version_t  *const p_version )
```

9.42.14.1 Brief description

Get the driver version based on compile time macros.

9.42.14.2 Detailed description

Table 1726:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	p_version is NULL.

9.42.15 g_qspi_on_qspi

```
qspi_api_t::g_qspi_on_qspi
```

9.42.15.1 Initialized as

```
g_qspi_on_qspi=
{
    .open           = R_QSPI_Open,
    .close          = R_QSPI_Close,
    .read           = R_QSPI_Read,
    .pageProgram    = R_QSPI_PageProgram,
    .erase          = R_QSPI_Erase,
    .infoGet        = R_QSPI_InfoGet,
    .sectorErase    = R_QSPI_SectorErase,
    .statusGet      = R_QSPI_StatusGet,
```

```
    .bankSelect      = R_QSPI_BankSelect,
    .versionGet      = R_QSPI_VersionGet
}
```

9.42.16 Extensions

9.42.16.1 qspi_instance_ctrl_t

qspi_instance_ctrl_t

Detailed description

Instance control block. DO NOT INITIALIZE. Initialization occurs when open is called

Variables

- R_QSPI_Type * p_reg

  Pointer to QSPI base register.
- uint32_t max_eraseable_size

  Largest eraseable sector size in kbytes. Used to determine buffer size for.
- uint32_t num_address_bytes

  Number of bytes used to represent the address.
- uint32_t spi_mode

  SPI mode - 0 = Extended, 1 = Dual, 2 = Quad.
- uint32_t page_size

  Number of bytes in a programmable page.
- uint8_t manufacturer_id

  Manufacturer ID.
- uint8_t memory_type

  Memory type.
- uint8_t memory_capacity

  Memory capacity (in MByte)
- bool xip_mode

  0 = run in read mode, 1 = run in XIP mode

9.43 IIC

Driver for the I2C Bus Interface (IIC).

This module supports the Renesas Inter-Integrated Circuit (IIC) peripheral. It implements the following interfaces:

- I2C Master Interface r_i2c_api.h

9.43.1 Functions

- riic_notify
- riic_clock_settings
- riic_abort_seq_master
- riic_transfer_open
- r_riic_irq_cfg
- riic_param_check
- riic_open_hw_master
- riic_close_hw_master
- riic_configure_interrupts_master
- riic_abort_hw_master
- riic_enable_transfer_support_tx
- riic_enable_transfer_support_rx
- riic_run_hw_master
- riic_rxi_read_data
- riic_txi_send_address
- riic_set_valid_interrupts_priority
- riic_transfer_configure_rx
- riic_transfer_configure_tx
- riic_rxi_master
- riic_txi_master
- riic_tei_master
- riic_err_master
- R_RIIC_MasterVersionGet
- R_RIIC_MasterOpen
- R_RIIC_MasterClose
- R_RIIC_MasterRead
- R_RIIC_MasterWrite
- R_RIIC_MasterReset
- R_RIIC_MasterSlaveAddressSet

9.43.2 Variables

- g_riic_master_version
- g_i2c_master_on_riic

9.43.3 Defines

- #define RIIC_MASTER_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define RIIC_MASTER_CODE_VERSION_MINOR

  Initial value: `(9U)`
- #define RIIC_OPEN

  Initial value: `(0x52494943ULL)`

  "RIIC" in ASCII, used to determine if channel is open.
- #define VARIANT_FMPLUS_MASK

  Initial value:`0x04U`
- #define RIIC_ERROR_RETURN

  Initial value:`#define SSP_ERROR_RETURN((a), (err), &g_module_name[0], &g_riic_master_version)`

  Macro for error logger.
- #define I2C_CODE_READ

  Initial value: `(0x01U)`
- #define I2C_CODE_10BIT

  Initial value: `(0xF0U)`

9.43.4 riic_notify

```
riic_notify ( riic_instance_ctrl_t  *const p_ctrl ,  i2c_event_t  const event )
```

9.43.4.1 Brief description

Single point for managing the logic around notifying a transfer has finished.

9.43.4.2 Detailed description

Internal helper functions

(end addtogroup RIIC)

Table 1727:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to transfer that is ending.
event	in	The event code to pass to the callback.

9.43.4.3 Function steps

- Set the flag indicating that the transaction is completed

9.43.5 riic_clock_settings

```
ssp_err_t riic_clock_settings ( riic_instance_ctrl_t  *const p_ctrl )
```

9.43.5.1 Brief description

Configures the clock and filter settings.

9.43.5.2 Detailed description

Table 1728:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to RIIC software control block.

Table 1729:Return values

Name	Description
SSP_SUCCESS	Successfully configured the clock
SSP_ERR_INVALID_RATE	Failed to configure the clock settings for this rate

9.43.6 riic_abort_seq_master

```
ssp_err_t riic_abort_seq_master ( riic_instance_ctrl_t  *const p_ctrl )
```

9.43.6.1 Brief description

Single point for managing the logic around aborting a transfer when operating as a master.

9.43.6.2 Detailed description

Table 1730:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control struct of specific device

Table 1731:Return values

Name	Description
SSP_SUCCESS	Stopped any DTC assisted transfer.
SSP_ERR_ABORTED	If a transfer is in-progress.

9.43.7 riic_transfer_open

```
ssp_err_t riic_transfer_open ( riic_instance_ctrl_t * p_ctrl , i2c_cfg_t const
*const p_cfg )
```

9.43.7.1 Brief description

Configures RIIC related transfer drivers (if enabled).

9.43.7.2 Detailed description

Table 1732:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to IIC specific control structure
p_cfg	in	Pointer to IIC specific configuration structure

Table 1733:Return values

Name	Description
SSP_SUCCESS	Transfer interface initialized successfully.
SSP_ERR_ASSERTION	Pointer to transfer instance for I2C receive in p_cfg is NULL.
SSP_ERR_IRQ_BSP_DISABLED	Interrupt does not exist in the vector table.

See Common Error Codes for other possible return codes. This function internally calls

- productFeatureGet

9.43.8 r_riic_irq_cfg

```
ssp_err_t r_riic_irq_cfg (  ssp_feature_t * p_feature ,   ssp_signal_t signal ,
  uint8_t ipl ,   void * p_ctrl ,   IRQn_Type * p_irq )
```

9.43.8.1 Brief description

Sets interrupt priority and initializes vector info.

9.43.8.2 Detailed description

Table 1734:Parameters

Name	Direction	Description
p_feature	in	SSP feature
signal	in	SSP signal ID
ipl	in	Interrupt priority level
p_ctrl	in	Pointer to driver control block
p_irq	out	Pointer to IRQ for this signal, set here

Table 1735:Return values

Name	Description
SSP_SUCCESS	Interrupt enabled
SSP_ERR_IRQ_BSP_DISABLED	Interrupt does not exist in the vector table

See Common Error Codes for other possible return codes. This function calls

- eventInfoGet

9.43.9 riic_param_check

```
ssp_err_t riic_param_check ( riic_instance_ctrl_t  *const p_ctrl ,  i2c_cfg_t
const *const p_cfg )
```

9.43.9.1 Brief description

Parameter check.

9.43.9.2 Detailed description

Functions that manipulate hardware

Table 1736:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to IIC specific control structure
p_cfg	in	Pointer to IIC specific configuration structure

Table 1737:Return values

Name	Description
SSP_SUCCESS	Provided parameters not NULL.
SSP_ERR_ASSERTION	The parameter p_api_ctrl or p_cfg is NULL or clock rate is greater than 1MHz. or p_cfg->p_extend not equal to NULL.

9.43.9.3 Function steps

- p_extend not supported currently

9.43.10 riic_open_hw_master

```
ssp_err_t riic_open_hw_master ( riic_instance_ctrl_t  *const p_ctrl ,
  ssp_feature_t * p_feature )
```

9.43.10.1 Brief description

Performs the hardware initialization sequence when operating as a master.

9.43.10.2 Detailed description

Table 1738:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control struct of specific device
p_feature	in	SSP Feature

Table 1739:Return values

Name	Description
SSP_SUCCESS	Hardware initialized with proper configurations.
SSP_ERR_INVALID_RATE	The requested rate could not be set.

9.43.11 riic_close_hw_master

```
riic_close_hw_master ( riic_instance_ctrl_t  *const p_ctrl )
```

9.43.11.1 Brief description

Performs the hardware initialization sequence when operating as a master.

9.43.11.2 Detailed description

Table 1740:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control struct of specific device

9.43.12 riic_configure_interrupts_master

```
riic_configure_interrupts_master ( riic_instance_ctrl_t  *const p_ctrl )
```

9.43.12.1 Brief description

Enables and assigns the interrupts to be used in master mode.

9.43.12.2 Detailed description

Table 1741:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control struct of specific device

9.43.13 riic_abort_hw_master

```
riic_abort_hw_master ( riic_instance_ctrl_t  *const p_ctrl )
```

9.43.13.1 Brief description

Safely stops the current data transfer when operating as a master.

9.43.13.2 Detailed description

Table 1742:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control struct of specific device

9.43.14 riic_enable_transfer_support_tx

```
riic_enable_transfer_support_tx ( riic_instance_ctrl_t  *const p_ctrl )
```

9.43.14.1 Brief description

Enables the dtc transfer interface for the transmit operation.

9.43.14.2 Detailed description

Table 1743:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to transfer control block

9.43.15 riic_enable_transfer_support_rx

```
riic_enable_transfer_support_rx ( riic_instance_ctrl_t  *const p_ctrl )
```

9.43.15.1 Brief description

Enables the dtc transfer interface for the receive operation.

9.43.15.2 Detailed description

Table 1744:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to transfer control block

9.43.16 riic_run_hw_master

```
ssp_err_t riic_run_hw_master ( riic_instance_ctrl_t  *const p_ctrl )
```

9.43.16.1 Brief description

Performs the data transfer described by the parameters when operating as a master.

9.43.16.2 Detailed description

Table 1745:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control struct of specific device.

Table 1746:Return values

Name	Description
SSP_SUCCESS	Data transfer success.
SSP_ERR_IN_USE	If data transfer is in progress.
SSP_ERR_ABORTED	If an error occurred while data transfer.

9.43.17 riic_rxi_read_data

```
riic_rxi_read_data ( riic_instance_ctrl_t  *const p_ctrl )
```

9.43.17.1 Brief description

Check valid receive data and set WAIT, NACK and STOP/RESTART bit in RXI handler.

9.43.17.2 Detailed description

Table 1747:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to transfer control block

9.43.18 riic_txi_send_address

```
riic_txi_send_address ( riic_instance_ctrl_t  *const p_ctrl )
```

9.43.18.1 Brief description

Write the address byte to the riic bus.

9.43.18.2 Detailed description

Table 1748:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to transfer control block

9.43.19 riic_set_valid_interrupts_priority

```
ssp_err_t riic_set_valid_interrupts_priority ( riic_instance_ctrl_t  * p_ctrl ,
 i2c_cfg_t  const *const p_cfg )
```

9.43.19.1 Brief description

Set valid interrupts priority.

9.43.19.2 Detailed description

Table 1749:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control struct of specific device

Table 1749:Parameters (Continued)

Name	Direction	Description
p_cfg	in	Pointer to IIC specific configuration structure

Table 1750:Return values

Name	Description
SSP_SUCCESS	Interrupts enabled.
SSP_ERR_IRQ_BSP_DISABLED	Interrupt does not exist in the vector table.

See Common Error Codes for other possible return codes. This function calls

- productFeatureGet

9.43.20 riic_transfer_configure_rx

```
ssp_err_t riic_transfer_configure_rx ( riic_instance_ctrl_t  * p_ctrl ,
 i2c_cfg_t  const *const p_cfg )
```

9.43.20.1 Brief description

Configures RIIC RX related transfer.

9.43.20.2 Detailed description

Table 1751:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to IIC specific control structure
p_cfg	in	Pointer to IIC specific configuration structure

Table 1752:Return values

Name	Description
SSP_SUCCESS	rx transfer interface in configured with valid parameters.
SSP_ERR_ASSERTION	Pointer to transfer instance for I2C receive in p_cfg is NULL.
SSP_ERR_IRQ_BSP_DISABLED	Interrupt does not exist in the vector table.

See Common Error Codes for other possible return codes. This function calls

- productFeatureGet

9.43.20.3 Function steps

- Set default transfer info and open receive transfer module, if enabled.

9.43.21 riic_transfer_configure_tx

```
ssp_err_t riic_transfer_configure_tx ( riic_instance_ctrl_t  * p_ctrl ,
 i2c_cfg_t  const *const p_cfg )
```

9.43.21.1 Brief description

Configures RIIC TX related transfer.

9.43.21.2 Detailed description

Table 1753:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to IIC specific control structure
p_cfg	in	Pointer to IIC specific configuration structure

Table 1754:Return values

Name	Description
SSP_SUCCESS	tx transfer interface in configured with valid parameters.
SSP_ERR_ASSERTION	Pointer to transfer instance for I2C transmit in p_cfg is NULL.
SSP_ERR_IRQ_BSP_DISABLED	Interrupt does not exist in the vector table.

See Common Error Codes for other possible return codes. This function calls

- productFeatureGet

9.43.21.3 Function steps

- Set default transfer info and open transmit transfer module, if enabled.

9.43.22 riic_rxi_master

```
riic_rxi_master ( riic_instance_ctrl_t  * p_ctrl )
```

9.43.22.1 Brief description

Handles the receive data full interrupt when operating as a master.

9.43.22.2 Detailed description

Interrupt handlers

Table 1755:Parameters

Name	Direction	Description
p_ctrl	in	The target RIIC block's control block.

9.43.23 riic_txi_master

```
riic_txi_master ( riic_instance_ctrl_t  * p_ctrl )
```

9.43.23.1 Brief description

Handles the transmit data empty interrupt when operating as a master.

9.43.23.2 Detailed description

Table 1756:Parameters

Name	Direction	Description
p_ctrl	in	The target RIIC block's control block.

9.43.24 riic_tei_master

```
riic_tei_master ( riic_instance_ctrl_t  * p_ctrl )
```

9.43.24.1 Brief description

Handles the transmit end interrupt when operating as a master.

9.43.24.2 Detailed description

NOTE: This interrupt is configured to be generated at the end of last byte of the requested transfer.

Table 1757:Parameters

Name	Direction	Description
p_ctrl	in	The target RIIC block's control block.

9.43.25 riic_err_master

```
riic_err_master ( riic_instance_ctrl_t  * p_ctrl )
```

9.43.25.1 Brief description

Handles the error interrupts when operating as a master.

9.43.25.2 Detailed description

Table 1758:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to transfer control block

9.43.26 R_RIIC_MasterVersionGet

```
ssp_err_t R_RIIC_MasterVersionGet ( ssp_version_t  *const p_version )
```

9.43.26.1 Brief description

Gets version information and stores it in the provided version struct.

9.43.26.2 Detailed description

Table 1759:Return values

Name	Description
SSP_SUCCESS	Successful version get.
SSP_ERR_ASSERTION	p_version is NULL.

9.43.27 R_RIIC_MasterOpen

```
ssp_err_t R_RIIC_MasterOpen ( i2c_ctrl_t  *const p_api_ctrl ,  i2c_cfg_t  const
*const p_cfg )
```

9.43.27.1 Brief description

Opens the I2C device. May power on IIC peripheral and perform initialization described in hardware manual.

9.43.27.2 Detailed description

This function will reconfigure the clock settings of the peripheral when a device with a lower rate than previously configured is opened.

Table 1760:Return values

Name	Description
SSP_SUCCESS	Requested clock rate was set exactly.
SSP_ERR_ASSERTION	The parameter p_api_ctrl or p_cfg is NULL or clock rate is greater than 1MHz. or p_cfg->p_extend not equal to NULL.
SSP_ERR_IN_USE	Attempted to open an already open device instance.
SSP_ERR_INVALID_ARGUMENT	If fast mode plus is configured and the channel does not support it
SSP_ERR_INVALID_RATE	The requested rate cannot be set.

See Common Error Codes for other possible return codes. This function calls

- productFeatureGet
- g_cgc_on_cgc.systemClockFreqGet

9.43.28 R_RIIC_MasterClose

```
ssp_err_t R_RIIC_MasterClose ( i2c_ctrl_t  *const p_api_ctrl )
```

9.43.28.1 Brief description

Closes the I2C device. May power down IIC peripheral.

9.43.28.2 Detailed description

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

Table 1761:Return values

Name	Description
SSP_SUCCESS	Device closed without issue.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_ABORTED	Device was closed while a transfer was in progress.

9.43.29 R_RIIC_MasterRead

```
ssp_err_t R_RIIC_MasterRead ( i2c_ctrl_t  *const p_api_ctrl ,   uint8_t
*const p_dest ,   uint32_t const bytes ,   bool const restart )
```

9.43.29.1 Brief description

Performs a read from the I2C device.

9.43.29.2 Detailed description

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C read operation will begin. When no callback is provided by the user, this function performs a blocking read. Otherwise, the read operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_RX_COMPLETE in the callback.

Table 1762:Return values

Name	Description
SSP_SUCCESS	Function executed without issue, if no callback was provided, the process was kicked off.
SSP_ERR_ASSERTION	p_api_ctrl, p_dest or bytes is NULL.
SSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size(65535) while DTC is used for data transfer.
SSP_ERR_IN_USE	Another transfer was in progress.
SSP_ERR_ABORTED	The transfer failed.

9.43.30 R_RIIC_MasterWrite

```
ssp_err_t R_RIIC_MasterWrite ( i2c_ctrl_t  *const p_api_ctrl ,   uint8_t
*const p_src ,   uint32_t const bytes ,   bool const restart )
```

9.43.30.1 Brief description

Performs a write to the I2C device.

9.43.30.2 Detailed description

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C write operation will begin. When no callback is provided by the user, this function performs a blocking write. Otherwise, the write operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_TX_COMPLETE in the callback.

Table 1763:Return values

Name	Description
SSP_SUCCESS	Function executed without issue, if no callback was provided, the process was kicked off.
SSP_ERR_ASSERTION	p_api_ctrl or p_src is NULL.
SSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size(65535) while DTC is used for data transfer.
SSP_ERR_IN_USE	Another transfer was in progress.
SSP_ERR_ABORTED	The transfer failed.

9.43.31 R_RIIC_MasterReset

```
ssp_err_t R_RIIC_MasterReset ( i2c_ctrl_t  *const p_api_ctrl )
```

9.43.31.1 Brief description

Aborts any in-progress transfer and forces the IIC peripheral into a ready state.

9.43.31.2 Detailed description

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

Table 1764:Return values

Name	Description
SSP_SUCCESS	Channel was reset without issue.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_ABORTED	A transfer was aborted while resetting the hardware.

9.43.32 R_RIIC_MasterSlaveAddressSet

```
ssp_err_t R_RIIC_MasterSlaveAddressSet ( i2c_ctrl_t  *const p_api_ctrl ,
  uint16_t const slave ,  i2c_addr_mode_t  const addr_mode )
```

9.43.32.1 Brief description

Sets address and addressing mode of the slave device.

9.43.32.2 Detailed description

This function is used to set the device address and addressing mode of the slave without reconfiguring the entire bus.

Table 1765:Return values

Name	Description
SSP_SUCCESS	Address of the slave is set correctly.
SSP_ERR_ASSERTION	Pointer to control structure is NULL.
SSP_ERR_IN_USE	Another transfer was in-progress.
SSP_ERR_NOT_OPEN	Device was not even opened.

9.43.33 g_riic_master_version

ssp_version_t::g_riic_master_version

9.43.33.1 Detailed description

Name of module used by error logger macro Version data structure used by error logger macro.

9.43.34 g_i2c_master_on_riic

i2c_api_master_t::g_i2c_master_on_riic

9.43.34.1 Detailed description

RIIC Implementation of I2C device master interface

9.43.34.2 Initialized as

```
g_i2c_master_on_riic=
{
    .open            = R_RIIC_MasterOpen,
    .close           = R_RIIC_MasterClose,
    .read            = R_RIIC_MasterRead,
    .write           = R_RIIC_MasterWrite,
    .reset           = R_RIIC_MasterReset,
```

```
    .versionGet      = R_RIIC_MasterVersionGet,
    .slaveAddressSet = R_RIIC_MasterSlaveAddressSet
}
```

9.43.35 Extensions

9.43.35.1 riic_instance_ctrl_t

riic_instance_ctrl_t

Detailed description

I2C control structure. DO NOT INITIALIZE.

Variables

- i2c_cfg_t info

  Information describing I2C device.
- uint32_t open

  Flag to determine if the device is open.
- void * p_reg

  Base register for this channel.
- IRQn_Type rxi_irq

  Receive IRQ number.
- IRQn_Type txi_irq

  Transmit IRQ number.
- IRQn_Type tei_irq

  Transmit end IRQ number.
- IRQn_Type eri_irq

  Error IRQ number.
- uint8_t * p_buff

  Holds the data associated with the transfer
- uint32_t total

  Holds the total number of data bytes to transfer
- uint32_t remain

  Tracks the remaining data bytes to transfer
- uint32_t loaded

  Tracks the number of data bytes written to the register

- uint8_t addr_low

  Holds the last address byte to issue
- uint8_t addr_high

  Holds the first address byte to issue in 10-bit mode
- uint8_t addr_total

  Holds the total number of address bytes to transfer
- uint8_t addr_remain

  Tracks the remaining address bytes to transfer
- uint8_t addr_loaded

  Tracks the number of address bytes written to the register
- bool read

  Holds the direction of the data byte transfer
- bool restart

  Holds whether or not the restart should be issued when done
- bool err

  Tracks whether or not an error occurred during processing
- bool restarted

  Tracks whether or not a restart was issued during the previous transfer
- bool transaction_completed

  Tracks if the transaction started earlier was completed
- bool dummy_read_completed

  Tracks whether the dummy read is performed
- bool activation_on_rxi

  < Tracks whether the transfer is activated on RXI interrupt
- bool activation_on_txi

  < Tracks whether the transfer is activated on TXI interrupt
- bool address_restarted

  < Tracks whether the restart condition is send on 10 bit read

9.44 IIC Slave

Driver for the I2C Bus Slave Interface (IIC Slave).

This module supports the Renesas Inter-Integrated Circuit (IIC) peripheral. It implements the following interfaces:

- I2C Master Interface r_i2c_api.h

9.44.1 Functions

- riic_notify
- riic_slave_set_irq_parameters
- riic_open_hw_slave
- riic_close_hw_slave
- riic_run_hw_slave
- riic_slave_clock_settings
- riic_clear_all_pending_interrupts
- riic_slave_configure_interrupts
- riic_rxi_slave
- riic_txi_slave
- riic_err_slave
- R_RIIC_SlaveVersionGet
- R_RIIC_SlaveOpen
- R_RIIC_SlaveClose
- R_RIIC_MasterWriteSlaveRead
- R_RIIC_MasterReadSlaveWrite

9.44.2 Variables

- g_riic_slave_version
- g_i2c_slave_on_riic

9.44.3 Defines

- #define RIIC_SLAVE_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define RIIC_SLAVE_CODE_VERSION_MINOR

  Initial value: `(3U)`
- #define RIIC_SLAVE_ERROR_RETURN

  Initial value:`#define SSP_ERROR_RETURN((a), (err), &g_module_name[0], &g_riic_slave_version)`

  Macro for error logger.

- #define RIIC_SLAVE_OPEN

  Initial value: `(0x49324353ULL)`

  "I2CS" in ASCII, used to determine if channel is open.
- #define MAX_CKS_DIVISOR

  Initial value:`7U`
- #define MAX_BRCL_VALUE

  Initial value:`31U`

9.44.4 riic_notify

```
 riic_notify ( riic_slave_instance_ctrl_t  *const p_ctrl ,  i2c_event_t
const event )
```

9.44.4.1 Brief description

Single point for managing the logic around notifying a transfer has finished.

9.44.4.2 Detailed description

Internal helper functions

(end addtogroup RIIC_SLAVE)

Table 1766:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to transfer that is ending.
event	in	The event code to pass to the callback.

9.44.5 riic_slave_set_irq_parameters

```
ssp_err_t riic_slave_set_irq_parameters (  ssp_feature_t * p_feature ,
  ssp_signal_t signal ,   uint8_t ipl ,   void * p_ctrl ,   IRQn_Type * p_irq )
```

9.44.5.1 Detailed description

Sets interrupt priority and initializes vector info

Table 1767:Parameters

Name	Direction	Description
p_feature	in	SSP feature
signal	in	SSP signal ID
ipl	in	Interrupt priority level
p_ctrl	in	Pointer to driver control block
p_irq	out	Pointer to IRQ for this signal, set here

Table 1768:Return values

Name	Description
SSP_SUCCESS	Interrupt enabled
SSP_ERR_IRQ_BSP_DISABLED	Interrupt does not exist in the vector table

See Common Error Codes for other possible return codes. This function calls

- eventInfoGet

9.44.6 riic_open_hw_slave

```
riic_open_hw_slave ( riic_slave_instance_ctrl_t  *const p_ctrl ,
 ssp_feature_t * p_feature )
```

9.44.6.1 Brief description

Performs the hardware initialization sequence when operating as a slave.

9.44.6.2 Detailed description

Functions that manipulate hardware

Table 1769:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control struct of specific device
p_feature	in	

9.44.7 riic_close_hw_slave

```
riic_close_hw_slave ( riic_slave_instance_ctrl_t  *const p_ctrl )
```

9.44.7.1 Brief description

Performs the hardware initialization sequence when operating as a slave.

9.44.7.2 Detailed description

Table 1770:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control struct of specific device

9.44.8 riic_run_hw_slave

```
ssp_err_t riic_run_hw_slave ( riic_slave_instance_ctrl_t  *const p_ctrl )
```

9.44.8.1 Brief description

Performs the data transfer described by the parameters when operating as a slave.

9.44.8.2 Detailed description

Table 1771:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to transfer that needs to be done.

Table 1772:Return values

Name	Description
SSP_SUCCESS	Transaction completed successfully.
SSP_ERR_ABORTED	If transaction encounter an error.

9.44.9 riic_slave_clock_settings

```
riic_slave_clock_settings ( riic_slave_instance_ctrl_t  *const p_ctrl )
```

9.44.9.1 Brief description

Sets the I2C clock and BRCL counter to a value greater than the operation mode setup time.

9.44.9.2 Detailed description

Table 1773:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to driver control block

9.44.10 riic_clear_all_pending_interrupts

```
riic_clear_all_pending_interrupts ( riic_slave_instance_ctrl_t  *const p_ctrl )
```

9.44.10.1 Brief description

Enables and assigns the interrupts to be used in slave mode.

9.44.10.2 Detailed description

Table 1774:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control struct of specific device

9.44.11 riic_slave_configure_interrupts

```
ssp_err_t riic_slave_configure_interrupts (  ssp_feature_t * ssp_feature ,
 riic_slave_instance_ctrl_t  *const p_api_ctrl ,  i2c_cfg_t  const *const p_cfg )
```

9.44.11.1 Detailed description

Sets interrupt priority and initializes vector info

Table 1775:Parameters

Name	Direction	Description
ssp_feature	in	SSP feature
p_ctrl	in	Pointer to driver control block
p_cfg	out	Pointer to driver configuration block

Table 1776:Return values

Name	Description
SSP_SUCCESS	Interrupt enabled.
SSP_ERR_IRQ_BSP_DISABLED	Interrupt does not exist in the vector table.

9.44.12 riic_rxi_slave

```
riic_rxi_slave ( riic_slave_instance_ctrl_t  * p_ctrl )
```

9.44.12.1 Brief description

Handles the receive data full interrupt when operating as a slave.

9.44.12.2 Detailed description

Interrupt handlers

Table 1777:Parameters

Name	Direction	Description
p_ctrl	in	The target RIIC block's control block.

9.44.13 riic_txi_slave

```
riic_txi_slave ( riic_slave_instance_ctrl_t  * p_ctrl )
```

9.44.13.1 Brief description

Handles the transmit data empty interrupt when operating as a slave.

9.44.13.2 Detailed description

Table 1778:Parameters

Name	Direction	Description
p_ctrl	in	The target RIIC block's control block.

9.44.14 riic_err_slave

```
riic_err_slave ( riic_slave_instance_ctrl_t  * p_ctrl )
```

9.44.14.1 Brief description

Handles the error interrupts when operating as a slave.

9.44.14.2 Detailed description

Table 1779:Parameters

Name	Direction	Description
p_ctrl	in	The target RIIC block's control block.

9.44.15 R_RIIC_SlaveVersionGet

```
ssp_err_t R_RIIC_SlaveVersionGet ( ssp_version_t  *const p_version )
```

9.44.15.1 Brief description

Gets version information and stores it in the provided version struct.

9.44.15.2 Detailed description

Table 1780:Return values

Name	Description
SSP_SUCCESS	Successful version get.
SSP_ERR_ASSERTION	p_version is NULL.

9.44.16 R_RIIC_SlaveOpen

```
ssp_err_t R_RIIC_SlaveOpen ( i2c_ctrl_t  *const p_api_ctrl ,  i2c_cfg_t  const
*const p_cfg )
```

9.44.16.1 Brief description

Opens the I2C device. May power on IIC peripheral and perform initialization described in hardware manual.

9.44.16.2 Detailed description

Table 1781:Return values

Name	Description
SSP_SUCCESS	Opened identical configuration of already open instance.
SSP_ERR_ASSERTION	p_api_ctrl or p_cfg is NULL.
SSP_ERR_IN_USE	Attempted to open an already open device instance.
SSP_ERR_IRQ_BSP_DISABLED	Interrupt does not exist in the vector table.
SSP_ERR_INVALID_ARGUMENT	If fast mode plus is configured and the channel does not support it

See Common Error Codes for other possible return codes. This function calls

- productFeatureGet
- g_cgc_on_cgc.systemClockFreqGet

9.44.17 R_RIIC_SlaveClose

```
ssp_err_t R_RIIC_SlaveClose ( i2c_ctrl_t  *const p_api_ctrl )
```

9.44.17.1 Brief description

Closes the I2C device. Power down IIC peripheral.

9.44.17.2 Detailed description

Table 1782:Return values

Name	Description
SSP_SUCCESS	Device closed without issue.
SSP_ERR_NOT_OPEN	Device not opened.
SSP_ERR_ASSERTION	p_api_ctrl is NULL.
SSP_ERR_ABORTED	Device was closed while a transfer was in progress.

9.44.18 R_RIIC_MasterWriteSlaveRead

```
ssp_err_t R_RIIC_MasterWriteSlaveRead ( i2c_ctrl_t  *const p_api_ctrl ,
  uint8_t *const p_dest ,   uint32_t const bytes )
```

9.44.18.1 Brief description

Performs a read from the I2C Master device.

9.44.18.2 Detailed description

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C read operation will begin. When no callback is provided by the user, this function performs a blocking read. Otherwise, the read operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_RX_COMPLETE in the callback.

Table 1783:Return values

Name	Description
SSP_SUCCESS	Function executed without issue; if no callback was provided, the process was kicked off
SSP_ERR_ASSERTION	p_api_ctrl, bytes or p_dest is NULL.
SSP_ERR_IN_USE	Another transfer was in progress.
SSP_ERR_NOT_OPEN	device is not open.
SSP_ERR_ABORTED	If transaction encounter an error.

9.44.19 R_RIIC_MasterReadSlaveWrite

```
ssp_err_t R_RIIC_MasterReadSlaveWrite ( i2c_ctrl_t  *const p_api_ctrl ,
  uint8_t *const p_src ,   uint32_t const bytes )
```

9.44.19.1 Brief description

Performs a write to the I2C Master device.

9.44.19.2 Detailed description

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C write operation will begin. When no callback is provided by the user, this function performs a blocking write. Otherwise, the write operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_TX_COMPLETE in the callback.

Table 1784:Return values

Name	Description
SSP_SUCCESS	Function executed without issue; if no callback was provided, the process was kicked off
SSP_ERR_ASSERTION	p_api_ctrl or p_src is NULL.
SSP_ERR_IN_USE	Another transfer was in progress.
SSP_ERR_NOT_OPEN	device is not open.
SSP_ERR_ABORTED	If transaction encounter an error.

9.44.20 g_riic_slave_version

ssp_version_t::g_riic_slave_version

9.44.20.1 Detailed description

Name of module used by error logger macro Version data structure used by error logger macro.

9.44.21 g_i2c_slave_on_riic

i2c_api_slave_t::g_i2c_slave_on_riic

9.44.21.1 Detailed description

RIIC Implementation of I2C device slave interface

9.44.21.2 Initialized as

```
g_i2c_slave_on_riic=
{
    .open                       = R_RIIC_SlaveOpen,
    .close                      = R_RIIC_SlaveClose,
    .masterWriteSlaveRead       = R_RIIC_MasterWriteSlaveRead,
    .masterReadSlaveWrite       = R_RIIC_MasterReadSlaveWrite,
    .versionGet                 = R_RIIC_SlaveVersionGet
}
```

9.44.22 Extensions

9.44.22.1 riic_slave_instance_ctrl_t

riic_slave_instance_ctrl_t

Detailed description

I2C control structure. DO NOT INITIALIZE.

Variables

- i2c_cfg_t info

  Information describing I2C device.

- uint32_t open

  Flag to determine if the device is open.

- void * p_reg

  Base register for this channel.

- IRQn_Type rxi_irq

  Receive IRQ number.
- IRQn_Type txi_irq

  Transmit IRQ number.
- IRQn_Type eri_irq

  Error IRQ number.
- uint8_t * p_buff

  Holds the data associated with the transfer
- uint32_t total

  Holds the total number of data bytes to transfer
- uint32_t remain

  Tracks the remaining data bytes to transfer
- uint32_t loaded

  Tracks the number of data bytes written to the register
- uint32_t transaction_count

  Tracks the actual number of transactions
- bool read

  Holds the direction of the data byte transfer
- bool err

  Tracks whether or not an error occurred during processing
- bool slave_busy

  Tracks if the slave is busy performing a transaction
- bool do_dummy_read
- bool start_interrupt_enabled

  << Tracks whether a dummy read is issued on the first RX < Tracks whether the start interrupt is enabled
- bool restarted

9.45 SPI

Driver for the Serial Peripheral Interface (SPI).

This module supports SPI serial communication for the SPI module. The SPI Interface is defined in r_spi_api.h

9.45.1 Functions

- R_RSPI_Open
- R_RSPI_Read
- R_RSPI_Write
- R_RSPI_WriteRead
- rspi_write_read_common
- R_RSPI_Close
- rspi_baud_set
- R_RSPI_VersionGet

9.45.2 Defines

- #define RSPI_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define RSPI_CODE_VERSION_MINOR

  Initial value: `(8U)`

9.45.3 R_RSPI_Open

```
ssp_err_t R_RSPI_Open ( spi_ctrl_t  * p_api_ctrl ,  spi_cfg_t  const
*const p_cfg )
```

9.45.3.1 Brief description

This functions initializes a channel for SPI communication mode.

9.45.3.2 Detailed description

Implements open This function performs the following tasks: Performs parameter checking and processes error conditions. Applies power to the SPI channel. Disables interrupts. Initializes the associated registers with some default value and the user-configurable options. Provides the channel control for use with other API functions. Updates user-configurable file if necessary.

Table 1785:Return values

Name	Description
SSP_SUCCESS	Channel initialized successfully.

Table 1785:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	NULL pointer to following parameters p_ctrl, p_cfg, p_cfg::p_transfer_rx::p_api, p_cfg::p_transfer_rx::p_ctrl, p_cfg::p_transfer_rx::p_cfg, p_cfg::p_transfer_rx::p_cfg::p_info. or failed to set the baud rate,
SSP_ERR_INVALID_ARGUMENT	An element of the r_spi_cfg_t structure contains an invalid value. The parameters is out of range. Both transfer modules need to be present or absent.
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- open
- productFeatureGet
- eventInfoGet

NOTE: This function is reentrant.

9.45.4 R_RSPI_Read

```
ssp_err_t R_RSPI_Read ( spi_ctrl_t  *const p_api_ctrl ,   void const * p_dest ,
  uint32_t const length ,  spi_bit_width_t  const bit_width )
```

9.45.4.1 Brief description

This function receives data from a SPI device.

9.45.4.2 Detailed description

Implements read The function performs the following tasks: Performs parameter checking and processes error conditions. Disable Interrupts. Disable the SPI bus. Setup data bit width per user request. Enable the SPI bus. Enable interrupts. Start data transmission with dummy data via transmit buffer empty interrupt. Copy data from source buffer to the SPI data register for transmission. Receive data from receive buffer full interrupt occurs and copy data to the buffer of destination. Complete data reception via receive buffer full interrupt and transmitting dummy data.

Table 1786:Return values

Name	Description
SSP_SUCCESS	Read operation successfully completed.
SSP_ERR_ASSERTION	NULL pointer to control or destination parameters or transfer length is zero.
SSP_ERR_UNSUPPORTED	With DTC transfer mode, bit_width must match configured DTC transfer width
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.
SSP_ERR_NOT_OPEN	The channel has not been opened. Open channel first.
SSP_ERR_INVALID_HW_CONDITION	Failed to clear errors in the module

NOTE: This function is reentrant.

9.45.5 R_RSPI_Write

```
ssp_err_t R_RSPI_Write ( spi_ctrl_t  *const p_api_ctrl ,   void const * p_src ,
  uint32_t const length ,  spi_bit_width_t  const bit_width )
```

9.45.5.1 Brief description

This function transmits data to a SPI device using the TX Only Communications Operation Mode.

9.45.5.2 Detailed description

Implements write The function performs the following tasks: Performs parameter checking and processes error conditions. Disable Interrupts. Disable the SPI bus. Setup data bit width per user request. Enable the SPI bus. Enable interrupts. Start data transmission with dummy data via transmit buffer empty interrupt. Copy data from source buffer to the SPI data register for transmission. Receive data from receive buffer full interrupt occurs and do nothing with the received data. Complete data transmission via receive buffer full interrupt.

Table 1787:Return values

Name	Description
SSP_SUCCESS	Write operation successfully completed.

Table 1787:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	NULL pointer to control or source parameters or transfer length is zero.
SSP_ERR_UNSUPPORTED	With DTC transfer mode, bit_width must match configured DTC transfer width
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.
SSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
SSP_ERR_INVALID_HW_CONDITION	Failed to clear errors in the module

NOTE: This function is reentrant.

9.45.6 R_RSPI_WriteRead

```
ssp_err_t R_RSPI_WriteRead ( spi_ctrl_t  *const p_api_ctrl ,   void const
* p_src ,   void const * p_dest ,   uint32_t const length ,  spi_bit_width_t
const bit_width )
```

9.45.6.1 Brief description

This function simultaneously transmits data to a SPI device while receiving data from a SPI device (full duplex).

9.45.6.2 Detailed description

Implements spi_api_t::writeread The function performs the following tasks: Performs parameter checking and processes error conditions. Disable Interrupts. Disable the SPI bus. Setup data bit width per user request. Enable the SPI bus. Enable interrupts. Start data transmission using transmit buffer empty interrupt. Copy data from source buffer to the SPI data register for transmission. Receive data from receive buffer full interrupt occurs and copy data to the buffer of destination. Complete data transmission and reception via receive buffer full interrupt.

Table 1788:Return values

Name	Description
SSP_SUCCESS	Write operation successfully completed.
SSP_ERR_ASSERTION	NULL pointer to control, source or destination parameters or transfer length is zero.

Table 1788:Return values (Continued)

Name	Description
SSP_ERR_UNSUPPORTED	With DTC transfer mode, bit_width must match configured DTC transfer width
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.
SSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
SSP_ERR_INVALID_HW_CONDITION	Failed to clear errors in the module

NOTE: This function is reentrant.

9.45.7 rspi_write_read_common

```
ssp_err_t rspi_write_read_common ( rspi_instance_ctrl_t  *const p_ctrl ,   void
const * p_src ,   void const * p_dest ,   uint32_t const length ,
 spi_bit_width_t  const bit_width ,  spi_operation_t  tx_rx_mode )
```

9.45.8 R_RSPI_Close

```
ssp_err_t R_RSPI_Close ( spi_ctrl_t  *const p_api_ctrl )
```

9.45.8.1 Brief description

This function manages the closing of a channel by the following task.

9.45.8.2 Detailed description

Implements close Disables SPI operations by disabling the SPI bus. Power off the channel. Disables all the associated interrupts. Update channel status.

Table 1789:Return values

Name	Description
SSP_SUCCESS	Channel successfully closed.
SSP_ERR_ASSERTION	A required pointer argument is NULL.

Table 1789:Return values (Continued)

Name	Description
SSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.

NOTE: This function is reentrant.

9.45.9 rspi_baud_set

```
rspi_baud_set ( rspi_instance_ctrl_t  * p_ctrl ,  uint32_t baud_target )
```

9.45.10 R_RSPI_VersionGet

```
ssp_err_t R_RSPI_VersionGet ( ssp_version_t  * p_version )
```

9.45.10.1 Brief description

This function gets the version information of the underlying driver.

9.45.10.2 Detailed description

Implements spi_api_t::versionget

Table 1790:Return values

Name	Description
void	

NOTE: This function is reentrant.

9.45.11 API Data

9.45.11.1 rspi_spcmd_bit_length_t

```
rspi_spcmd_bit_length_t
```

Detailed description

Frame data length

Enumerated values

Name	Description
RSPI_SPCMD_BIT_LENGTH_8	0100 to 0111 = 8 bits data length
RSPI_SPCMD_BIT_LENGTH_16	1111 = 16 bits data length
RSPI_SPCMD_BIT_LENGTH_32	0011 = 32 bits data length

9.45.11.2 rspi_operation_t

`rspi_operation_t`

Detailed description

SPCR (RSPI Control register) SPMS (RSPI mode) select

Enumerated values

Name	Description
RSPI_OPERATION_SPI	SPI operation (4-wire method)
RSPI_OPERATION_CLK_SYN	Clock Synchronous operation (3-wire method)

9.45.11.3 rspi_communication_t

`rspi_communication_t`

Detailed description

SPCR (RSPI Control register) TXMD (communication operating mode) select

Enumerated values

Name	Description
RSPI_COMMUNICATION_FULL_DUPLEX	Full-Duplex synchronous serial communication
RSPI_COMMUNICATION_TRANSMIT_ONLY	Transit only serial communication

9.45.11.4 rspi_sslp_t

`rspi_sslp_t`

Detailed description

Definition for SSLP (RSPI Slave Select Polarity register) select

Enumerated values

Name	Description
RSPI_SSLP_LOW	SSLP signal polarity active low
RSPI_SSLP_HIGH	SSLP signal polarity active high

9.45.11.5 rspi_loopback1_t

`rspi_loopback1_t`

Detailed description

SPPCR (RSPI Pin Control Register) Loopback1 select

Enumerated values

Name	Description
RSPI_LOOPBACK1_NORMAL_DATA	Loopback1 normal mode
RSPI_LOOPBACK1_INVERTED_DATA	Loopback1 with inverted data

9.45.11.6 rspi_loopback2_t

`rspi_loopback2_t`

Detailed description

SPPCR (RSPI Pin Control Register) Loopback2 select

Enumerated values

Name	Description
RSPI_LOOPBACK2_NORMAL_DATA	Loopback2 normal mode
RSPI_LOOPBACK2_NOT_INVERTED_DATA	Loopback2 with not inverted data

9.45.11.7 rspi_mosi_idle_fixed_val_t

`rspi_mosi_idle_fixed_val_t`

Detailed description

SPPCR (RSPI Pin Control Register) MOIFV select

Enumerated values

Name	Description
RSPI_MOSI_IDLE_FIXED_VAL_LOW	MOSIn level low during MOSI idling
RSPI_MOSI_IDLE_FIXED_VAL_HIGH	MOSIn level high during MOSI idling

9.45.11.8 rspi_mosi_idle_val_fixing_t

`rspi_mosi_idle_val_fixing_t`

Detailed description

SPPCR (RSPI Pin Control Register) MOIFE (MOSI idle value fixing) select

Enumerated values

Name	Description
RSPI_MOSI_IDLE_VAL_FIXING_ENABLE	MOSI output value=final data from previous transfer
RSPI_MOSI_IDLE_VAL_FIXING_DISABLE	MOSI output value=value set in MOIFV bit

9.45.11.9 rspi_parity_state_t

`rspi_parity_state_t`

Detailed description

SPCR2 (RSPI Control Register 2) Parity Enable select

Enumerated values

Name	Description
RSPI_PARITY_STATE_DISABLE	Disable parity
RSPI_PARITY_STATE_ENABLE	Enable parity

9.45.11.10 rspi_parity_mode_t

`rspi_parity_mode_t`

Detailed description

SPCR2 (RSPI Control Register 2) Parity select

Enumerated values

Name	Description
RSPI_PARITY_MODE_ODD	Select even parity
RSPI_PARITY_MODE_EVEN	Select odd parity

9.45.11.11 rspi_ssl_select_t

`rspi_ssl_select_t`

Detailed description

SPCMD (RSPI Command) Register SSL Signal Assertion select

Enumerated values

Name	Description
RSPI_SSL_SELECT_SSL0	Select SSL0 as slave
RSPI_SSL_SELECT_SSL1	Select SSL1 as slave
RSPI_SSL_SELECT_SSL2	Select SSL2 as slave
RSPI_SSL_SELECT_SSL3	Select SSL3 as slave

9.45.11.12 rspi_ssl_level_keep_t

`rspi_ssl_level_keep_t`

Detailed description

SPCMD (RSPI Command) Register SSL Signal Level Keeping select

Enumerated values

Name	Description
RSPI_SSL_LEVEL_KEEP_NOT	Negates all SSL signals upon transfer completion
RSPI_SSL_LEVEL_KEEP	Keeps the SSL level upon transfer completion

9.45.11.13 rspi_clock_delay_count_t

`rspi_clock_delay_count_t`

Detailed description

SPCKD (RSPI Clock Delay) Register Clock Delay Count select

Enumerated values

Name	Description
RSPI_CLOCK_DELAY_COUNT_1	Set RSPCK Clock delay to 1 RSPCK
RSPI_CLOCK_DELAY_COUNT_2	Set RSPCK Clock delay to 2 RSPCK
RSPI_CLOCK_DELAY_COUNT_3	Set RSPCK Clock delay to 3 RSPCK
RSPI_CLOCK_DELAY_COUNT_4	Set RSPCK Clock delay to 4 RSPCK
RSPI_CLOCK_DELAY_COUNT_5	Set RSPCK Clock delay to 5 RSPCK
RSPI_CLOCK_DELAY_COUNT_6	Set RSPCK Clock delay to 6 RSPCK
RSPI_CLOCK_DELAY_COUNT_7	Set RSPCK Clock delay to 7 RSPCK
RSPI_CLOCK_DELAY_COUNT_8	Set RSPCK Clock delay to 8 RSPCK

9.45.11.14 rspi_clock_delay_state_t

`rspi_clock_delay_state_t`

Detailed description

SPCMD (RSPI Command) Register RSPCK Delay Enable/Disable select SCKDEN

Enumerated values

Name	Description
RSPI_CLOCK_DELAY_STATE_DISABLE	RSPCK delay=1 RSPCK
RSPI_CLOCK_DELAY_STATE_ENABLE	RSPCK delay=SPCKD register setting

9.45.11.15 rspi_ssl_negation_delay_count_t

`rspi_ssl_negation_delay_count_t`

Detailed description

SSLND (RSPI Slave Select Negation Delay) Register Slave Select Negation Delay Count select

Enumerated values

Name	Description
RSPI_SSL_NEGATION_DELAY_1	Set SSL negation delay to 1 RSPCK
RSPI_SSL_NEGATION_DELAY_2	Set SSL negation delay to 2 RSPCK
RSPI_SSL_NEGATION_DELAY_3	Set SSL negation delay to 3 RSPCK
RSPI_SSL_NEGATION_DELAY_4	Set SSL negation delay to 4 RSPCK
RSPI_SSL_NEGATION_DELAY_5	Set SSL negation delay to 5 RSPCK
RSPI_SSL_NEGATION_DELAY_6	Set SSL negation delay to 6 RSPCK
RSPI_SSL_NEGATION_DELAY_7	Set SSL negation delay to 7 RSPCK
RSPI_SSL_NEGATION_DELAY_8	Set SSL negation delay to 8 RSPCK

9.45.11.16 rspi_ssl_negation_delay_state_t

`rspi_ssl_negation_delay_state_t`

Detailed description

SPCMD (RSPI Command) Register SSL Negation Delay select SLNDEN

Enumerated values

Name	Description
RSPI_SSL_NEGATION_DELAY_DISABLE	SSL negation delay=1 RSPCK
RSPI_SSL_NEGATION_DELAY_ENABLE	SSL negation delay=SSLND register setting

9.45.11.17 rspi_next_access_delay_count_t

`rspi_next_access_delay_count_t`

Detailed description

SPND (RSPI Next-Access Delay) Register Next Access Delay Count select

Enumerated values

Name	Description
RSPI_NEXT_ACCESS_DELAY_COUNT_1	Set next access delay to 1 RSPCK+2PCLK

Name	Description
RSPI_NEXT_ACCESS_DELAY_COUNT_2	Set next access delay to 2 RSPCK+2PCLK
RSPI_NEXT_ACCESS_DELAY_COUNT_3	Set next access delay to 3 RSPCK+2PCLK
RSPI_NEXT_ACCESS_DELAY_COUNT_4	Set next access delay to 4 RSPCK+2PCLK
RSPI_NEXT_ACCESS_DELAY_COUNT_5	Set next access delay to 5 RSPCK+2PCLK
RSPI_NEXT_ACCESS_DELAY_COUNT_6	Set next access delay to 6 RSPCK+2PCLK
RSPI_NEXT_ACCESS_DELAY_COUNT_7	Set next access delay to 7 RSPCK+2PCLK
RSPI_NEXT_ACCESS_DELAY_COUNT_8	Set next access delay to 8 RSPCK+2PCLK

9.45.11.18 rspi_next_access_delay_state_t

`rspi_next_access_delay_state_t`

Detailed description

SPCMD (RSPI Command) Register Next Access Delay select SPNDEN

Enumerated values

Name	Description
RSPI_NEXT_ACCESS_DELAY_STATE_DISABLE	Next access delay=1 RSPCK + 2 PCLK
RSPI_NEXT_ACCESS_DELAY_STATE_ENABLE	Next access delay=SPND register setting

9.45.11.19 rspi_spcmd_br_div_t

`rspi_spcmd_br_div_t`

Enumerated values

Name	Description
RSPI_SPCMD_BR_DIV_1	
RSPI_SPCMD_BR_DIV_2	
RSPI_SPCMD_BR_DIV_4	
RSPI_SPCMD_BR_DIV_8	

9.45.11.20 rspi_spcmd_assert_ssl_t

`rspi_spcmd_assert_ssl_t`

Detailed description

Slave select to be asserted during transfer operation.

Enumerated values

Name	Description
RSPI_SPCMD_ASSERT_SSL0	
RSPI_SPCMD_ASSERT_SSL1	
RSPI_SPCMD_ASSERT_SSL2	
RSPI_SPCMD_ASSERT_SSL3	

9.45.12 Extensions

9.45.12.1 rspi_ssl_polarity_t

rspi_ssl_polarity_t

Detailed description

SSLP (RSPI Slave Select Polarity register) SSLnP select

Variables

- rspi_sslp_t rspi_ssl2
- rspi_sslp_t rspi_ssl3
- rspi_sslp_t rspi_ssl0
- rspi_sslp_t rspi_ssl1

9.45.12.2 rspi_loopback_t

rspi_loopback_t

Detailed description

SPPCR (RSPI Pin Control Register) Loopback select

Variables

- rspi_loopback1_t rspi_loopback1
- rspi_loopback2_t rspi_loopback2

9.45.12.3 rspi_mosi_idle_t

rspi_mosi_idle_t

Detailed description

SPPCR (RSPI Pin Control Register) MOIFV (mosi idle value) select

Variables

- rspi_mosi_idle_fixed_val_t rspi_mosi_idle_fixed_val
- rspi_mosi_idle_val_fixing_t rspi_mosi_idle_val_fixing

9.45.12.4 rspi_parity_t

rspi_parity_t

Detailed description

SPCR2 (RSPI Control Register 2) Parity select

Variables

- rspi_parity_state_t rspi_parity
- rspi_parity_mode_t rspi_parity_mode

9.45.12.5 rspi_clock_delay_t

rspi_clock_delay_t

Detailed description

Select RSPI Clock Delay Register (SPCKD) and SPCMD (RSPI Command) Register-Clock Delay state(SCKDEN)

Variables

- rspi_clock_delay_count_t rspi_clock_delay_count
- rspi_clock_delay_state_t rspi_clock_delay_state

9.45.12.6 rspi_ssl_negation_delay_t

rspi_ssl_negation_delay_t

Detailed description

Select SSL Negation Delay(SSLND) and SPCMD Register-SSL negation Delay state(SLNDEN)

Variables

- rspi_ssl_negation_delay_count_t rspi_ssl_neg_delay_count
- rspi_ssl_negation_delay_state_t rspi_ssl_neg_delay_state

9.45.12.7 rspi_access_delay_t

rspi_access_delay_t

Detailed description

Select Next Access Delay(SPND) and SPCMD Register-Next Access Delay state(SPNDEN)

Variables

- rspi_next_access_delay_count_t rspi_next_access_delay_count
- rspi_next_access_delay_state_t rspi_next_access_delay_state

9.45.12.8 spi_on_rspi_cfg_t

spi_on_rspi_cfg_t

Detailed description

Extended SPI interface configuration

Variables

- rspi_operation_t rspi_clksyn

  Select spi or clock syn mode operation
- rspi_communication_t rspi_comm

  Select full-duplex or transmit-only communication
- rspi_ssl_polarity_t ssl_polarity

  Select SSLn signal polarity
- rspi_loopback_t loopback

  Select loopback1 and loopback2
- rspi_mosi_idle_t mosi_idle

  Select mosi idle fixed value and selection
- rspi_parity_t parity

  Select parity and enable/disable parity
- rspi_ssl_select_t ssl_select

  Select which slave to use;0-SSL0;1-SSL1;2-SSL2;3-SSL3
- rspi_ssl_level_keep_t ssl_level_keep

  Select SSL level after transfer completion;0-negate;1-keeps
- rspi_clock_delay_t clock_delay

  Select clock delay from 0 to 7
- rspi_ssl_negation_delay_t ssl_neg_delay

  Select Slave elect negation delay from 0 to 7
- rspi_access_delay_t access_delay

  Select next access delay from 0 to 7

9.45.12.9 rspi_instance_ctrl_t

rspi_instance_ctrl_t

Detailed description

SPI instance control block. DO NOT INITIALIZE.

Variables

- uint8_t channel

  Channel number to be used.
- uint8_t current_slave

  Number of the currently assigned slave.
- uint32_t channel_opened

  Internal flag to indicate the peripheral was initialized.
- transfer_instance_t const * p_transfer_tx

  To use SPI DTC/DMA write transfer.
- transfer_instance_t const * p_transfer_rx

  To use SPI DTC/DMA read transfer.
- void(* p_callback)(*p_args)

  Pointer to user callback function.
- void const * p_context

  Pointer to the higher level device context.
- void * p_reg

  Base register for this channel.
- IRQn_Type rxi_irq

  Receive IRQ number.
- IRQn_Type txi_irq

  Transmit IRQ number.
- IRQn_Type tei_irq

  Transmit end IRQ number.
- IRQn_Type eri_irq

  Error IRQ number.
- void * p_src
- void * p_dest
- uint32_t tx_count
- uint32_t rx_count
- uint32_t xfr_length

- uint8_t bytes_per_transfer
- bool do_tx
- bool using_dtc
- transfer_addr_mode_t tx_dtc_addr_mode
- transfer_addr_mode_t rx_dtc_addr_mode
- spi_operation_t transfer_mode
- uint32_t rx_data
- bsp_lock_t resource_lock_tx_rx

  Resource lock for transmission/reception

9.46 RTC

Driver for the Realtime Clock (RTC).

This module supports the Real Time Clock (RTC). It implements the following interfaces:

- RTC Interface

9.46.1 Functions

- R_RTC_Open
- R_RTC_Close
- R_RTC_Configure
- R_RTC_CalendarTimeSet
- R_RTC_CalendarTimeGet
- R_RTC_CalendarAlarmSet
- R_RTC_CalendarAlarmGet
- R_RTC_CalendarCounterStart
- R_RTC_CalendarCounterStop
- R_RTC_IrqEnable
- R_RTC_IrqDisable
- R_RTC_PeriodicIrqRateSet
- R_RTC_InfoGet
- R_RTC_ErrorAdjustmentModeSet
- R_RTC_ErrorAdjustmentSet
- R_RTC_VersionGet

- r_rtc_start_bit_clear
- r_rtc_start_bit_set
- r_rtc_software_reset
- r_rtc_set_clock_source
- r_rtc_nvic_enable_irq
- r_rtc_config_rtc_interrupts
- r_rtc_enable_alarm_irq
- r_rtc_enable_irq
- r_rtc_disable_irq
- r_rtc_error_adjustment_mode_set_common
- r_rtc_error_adjustment_period_set
- rtc_dec_to_bcd
- rtc_bcd_to_dec

9.46.2 Defines

- #define RTC_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define RTC_CODE_VERSION_MINOR

  Initial value: `(7U)`

9.46.3 R_RTC_Open

```
ssp_err_t R_RTC_Open ( rtc_ctrl_t  *const p_api_ctrl ,  rtc_cfg_t  const
*const p_cfg )
```

9.46.3.1 Brief description

Open the RTC driver.

9.46.3.2 Detailed description

Implements open.

Opens and configures the RTC driver module. Configuration includes clock source, and interrupt callback function. If the sub-clock oscillator is the clock source it is started in this function.

Table 1791:Return values

Name	Description
SSP_SUCCESS	Initialization was successful and RTC has started.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_cfg pointer.
SSP_ERR_HW_LOCKED	Hardware in use
SSP_ERR_TIMEOUT	Status check for counter mode or reset timed out

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet

9.46.3.3 Function steps

- Mark driver as open by initializing it to "RTC" in its ASCII equivalent.

9.46.4 R_RTC_Close

```
ssp_err_t R_RTC_Close ( rtc_ctrl_t  *const p_api_ctrl )
```

9.46.4.1 Brief description

Close the RTC driver.

9.46.4.2 Detailed description

Implements close

Table 1792:Return values

Name	Description
SSP_SUCCESS	De-Initialization was successful and RTC driver closed.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for close.

9.46.5 R_RTC_Configure

```
ssp_err_t R_RTC_Configure ( rtc_ctrl_t  *const p_api_ctrl ,   void
*const p_extend )
```

9.46.5.1 Brief description

Configure the RTC driver.

9.46.5.2 Detailed description

Implements configure

Table 1793:Return values

Name	Description
SSP_SUCCESS	RTC was successful configured.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for operation.
SSP_ERR_TIMEOUT	Status check for counter mode or reset timed out

9.46.6 R_RTC_CalendarTimeSet

```
ssp_err_t R_RTC_CalendarTimeSet ( rtc_ctrl_t  *const p_api_ctrl ,  rtc_time_t
* p_time ,   bool clock_start )
```

9.46.6.1 Brief description

Set the calendar time.

9.46.6.2 Detailed description

Implements calendarTimeSet.

Table 1794:Return values

Name	Description
SSP_SUCCESS	Calendar time set operation was successful.

Table 1794:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	Invalid p_api_ctrl, p_time or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for operation.
SSP_ERR_INVALID_ARGUMENT	Invalid time parameter field.
SSP_ERR_TIMEOUT	Software reset status check failed.

9.46.7 R_RTC_CalendarTimeGet

```
ssp_err_t R_RTC_CalendarTimeGet ( rtc_ctrl_t  *const p_api_ctrl ,  rtc_time_t
* p_time )
```

9.46.7.1 Brief description

Get the calendar time.

9.46.7.2 Detailed description

Implements calendarTimeGet

Table 1795:Return values

Name	Description
SSP_SUCCESS	Calendar time get operation was successful.
SSP_ERR_ASSERTION	Invalid p_api_ctrl, p_time or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for operation.
SSP_ERR_TIMEOUT	IRQ enable operation timed out.

9.46.7.3 Function steps

- Store the carry IRQ status so it can be restored later.
- Restore the state of carry IRQ.

9.46.8 R_RTC_CalendarAlarmSet

```
ssp_err_t R_RTC_CalendarAlarmSet ( rtc_ctrl_t  *const p_api_ctrl ,
 rtc_alarm_time_t  * p_alarm ,   bool interrupt_enable_flag )
```

9.46.8.1 Brief description

Set the calendar alarm time.

9.46.8.2 Detailed description

Implements calendarAlarmSet.

NOTE: The calendar counter must be running before the alarm can be set.

Table 1796:Return values

Name	Description
SSP_SUCCESS	Calendar alarm time set operation was successful.
SSP_ERR_INVALID_ARGUMENT	Invalid time parameter field.
SSP_ERR_ASSERTION	Invalid p_api_ctrl, p_alarm or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for operation.

9.46.9 R_RTC_CalendarAlarmGet

```
ssp_err_t R_RTC_CalendarAlarmGet ( rtc_ctrl_t  *const p_api_ctrl ,
 rtc_alarm_time_t  * p_alarm )
```

9.46.9.1 Brief description

Get the calendar alarm time.

9.46.9.2 Detailed description

Implements calendarAlarmGet

Table 1797:Return values

Name	Description
SSP_SUCCESS	Calendar alarm time get operation was successful.
SSP_ERR_ASSERTION	Invalid p_api_ctrl, p_alarm or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for operation.

9.46.10 R_RTC_CalendarCounterStart

```
ssp_err_t R_RTC_CalendarCounterStart ( rtc_ctrl_t  *const p_api_ctrl )
```

9.46.10.1 Brief description

Start the calendar counter.

9.46.10.2 Detailed description

Implements calendarCounterStart.

Table 1798:Return values

Name	Description
SSP_SUCCESS	Calendar counter started.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for operation.

9.46.11 R_RTC_CalendarCounterStop

```
ssp_err_t R_RTC_CalendarCounterStop ( rtc_ctrl_t  *const p_api_ctrl )
```

9.46.11.1 Brief description

Stop the calendar counter.

9.46.11.2 Detailed description

Implements calendarCounterStop.

Table 1799:Return values

Name	Description
SSP_SUCCESS	Calendar counter stopped.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for operation.

9.46.12 R_RTC_IrqEnable

```
ssp_err_t R_RTC_IrqEnable ( rtc_ctrl_t  *const p_api_ctrl ,
 rtc_event_t  event )
```

9.46.12.1 Brief description

Enable the alarm interrupt.

9.46.12.2 Detailed description

Implements rtc_api_t::interruptEnable.

Table 1800:Return values

Name	Description
SSP_SUCCESS	Alarm interrupt enabled.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_IRQ_BSP_DISABLED	User IRQ parameter not valid.
SSP_ERR_INVALID_ARGUMENT	Invalid IRQ event.
SSP_ERR_TIMEOUT	IRQ enable operation timed out.
SSP_ERR_NOT_OPEN	Driver not open already for operation.

9.46.13 R_RTC_IrqDisable

```
ssp_err_t R_RTC_IrqDisable ( rtc_ctrl_t  *const p_api_ctrl ,
 rtc_event_t  event )
```

9.46.13.1 Brief description

Disable the alarm interrupt.

9.46.13.2 Detailed description

Implements rtc_api_t::interruptDisable.

Table 1801:Return values

Name	Description
SSP_SUCCESS	Alarm interrupt disabled.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_IRQ_BSP_DISABLED	User IRQ parameter not valid
SSP_ERR_INVALID_ARGUMENT	Invalid IRQ event
SSP_ERR_TIMEOUT	IRQ disable operation timed out.
SSP_ERR_NOT_OPEN	Driver not open already for operation.

9.46.14 R_RTC_PeriodicIrqRateSet

```
ssp_err_t R_RTC_PeriodicIrqRateSet ( rtc_ctrl_t  *const p_api_ctrl ,
 rtc_periodic_irq_select_t  rate )
```

9.46.14.1 Brief description

Set the periodic interrupt rate.

9.46.14.2 Detailed description

Implements rtc_api_t::periodicInterruptRateSet.

Table 1802:Return values

Name	Description
SSP_SUCCESS	The periodic interrupt rate was successfully set.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_TIMEOUT	Periodic interrupt rate get query timed out.
SSP_ERR_NOT_OPEN	Driver not open already for operation.

9.46.15 R_RTC_InfoGet

```
ssp_err_t R_RTC_InfoGet ( rtc_ctrl_t  * p_api_ctrl ,  rtc_info_t  * p_rtc_info )
```

9.46.15.1 Brief description

This function returns information about the driver clock source.

9.46.15.2 Detailed description

Implements infoGet

Table 1803:Return values

Name	Description
SSP_SUCCESS	Get information Successful.
SSP_ERR_ASSERTION	Invalid p_api_ctrl, p_rtc_info or p_ctrl->p_reg member pointed by p_api_ctrl pointer.
SSP_ERR_NOT_OPEN	Driver not open already for operation.

9.46.16 R_RTC_ErrorAdjustmentModeSet

```
ssp_err_t R_RTC_ErrorAdjustmentModeSet ( rtc_ctrl_t  * p_api_ctrl ,
 rtc_error_adjustment_mode_cfg_t  * p_error_adjustment_mode )
```

9.46.16.1 Brief description

This function sets time error adjustment mode.

9.46.16.2 Detailed description

Implements errorAdjustmentModeSet

Table 1804:Return values

Name	Description
SSP_SUCCESS	Time error adjustment mode set successful.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or error_adjustment_mode pointer.
SSP_ERR_NOT_OPEN	Driver not open for operation.
SSP_ERR_UNSUPPORTED	The clock source is not SubClock.
SSP_ERR_INVALID_ARGUMENT	Invalid error adjustment period.
SSP_ERR_TIMEOUT	Time error adjustment get query timed out.

9.46.17 R_RTC_ErrorAdjustmentSet

```
ssp_err_t R_RTC_ErrorAdjustmentSet ( rtc_ctrl_t  * p_api_ctrl ,
 rtc_error_adjustment_cfg_t  * p_error_adjustment_config )
```

9.46.17.1 Brief description

This function sets time error adjustment.

9.46.17.2 Detailed description

Implements errorAdjustmentSet

Table 1805:Return values

Name	Description
SSP_SUCCESS	Time error adjustment successful.
SSP_ERR_ASSERTION	Invalid p_api_ctrl or p_error_adjustment_config pointer.
SSP_ERR_NOT_OPEN	Driver not open for operation.
SSP_ERR_UNSUPPORTED	The clock source is not SubClock.
SSP_ERR_INVALID_ARGUMENT	Invalid error adjustment value.

Table 1805:Return values (Continued)

Name	Description
SSP_ERR_TIMEOUT	Time error adjustment get query timed out.

9.46.18 R_RTC_VersionGet

```
ssp_err_t R_RTC_VersionGet ( ssp_version_t  * p_version )
```

9.46.18.1 Brief description

Get driver version based on compile time macros.

9.46.18.2 Detailed description

Implements versionGet

Table 1806:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

9.46.19 r_rtc_start_bit_clear

```
ssp_err_t r_rtc_start_bit_clear (  R_RTC_Type * p_rtc_reg )
```

9.46.19.1 Brief description

Clear the start bit.

9.46.19.2 Detailed description

Table 1807:Return values

Name	Description
SSP_SUCCESS	start bit cleared
SSP_ERR_TIMEOUT	start bit not cleared

9.46.20 r_rtc_start_bit_set

```
ssp_err_t r_rtc_start_bit_set (  R_RTC_Type * p_rtc_reg )
```

9.46.20.1 Brief description

Set the start bit.

9.46.20.2 Detailed description

Table 1808:Return values

Name	Description
SSP_SUCCESS	start bit set
SSP_ERR_TIMEOUT	start bit not set

9.46.21 r_rtc_software_reset

```
ssp_err_t r_rtc_software_reset (  R_RTC_Type * p_rtc_reg )
```

9.46.21.1 Brief description

Perform a software reset.

9.46.21.2 Detailed description

Table 1809:Return values

Name	Description
SSP_SUCCESS	software reset complete
SSP_ERR_TIMEOUT	software reset not complete

9.46.22 r_rtc_set_clock_source

```
ssp_err_t r_rtc_set_clock_source (  R_RTC_Type * p_rtc_reg ,
 rtc_instance_ctrl_t  * p_ctrl )
```

9.46.22.1 Brief description

Set the RTC clock source.

9.46.22.2 Detailed description

Table 1810:Return values

Name	Description
SSP_SUCCESS	RTC clock source set
SSP_ERR_TIMEOUT	status check for counter mode or reset timed out

9.46.23 r_rtc_nvic_enable_irq

```
ssp_err_t r_rtc_nvic_enable_irq (  IRQn_Type nvic_interrupt ,
  uint32_t irq_en )
```

9.46.23.1 Brief description

check if timeout set else enable passed NVIC interrupt.

9.46.23.2 Detailed description

Implements a helper function

Table 1811:Return values

Name	Description
SSP_SUCCESS	Calendar counter stopped.

9.46.24 r_rtc_config_rtc_interrupts

```
ssp_err_t r_rtc_config_rtc_interrupts (  fmi_event_info_t * event_info ,
 rtc_instance_ctrl_t  * p_ctrl ,  rtc_cfg_t  const *const p_cfg ,  ssp_feature_t
* ssp_feature )
```

9.46.24.1 Brief description

get IRQ from event info to set IRQ priority and control info for IRQ handler .

9.46.24.2 Detailed description

Implements a helper function

Table 1812:Return values

Name	Description
SSP_SUCCESS	Successful configuration

9.46.25 r_rtc_enable_alarm_irq

```
ssp_err_t r_rtc_enable_alarm_irq ( rtc_instance_ctrl_t  * p_ctrl ,
  bool interrupt_enable_flag )
```

9.46.25.1 Brief description

enable alarm irq if valid

9.46.25.2 Detailed description

Implements a helper function

Table 1813:Return values

Name	Description
SSP_SUCCESS	Alarm IRQ enabled
SSP_ERR_TIMEOUT	check for Alarm IRQ enable bit timed out

9.46.26 r_rtc_enable_irq

```
ssp_err_t r_rtc_enable_irq (  R_RTC_Type * p_rtc_reg ,   IRQn_Type irq ,
 rtc_event_t  event ,   uint32_t timeout )
```

9.46.26.1 Brief description

enable alarm irq if valid

9.46.26.2 Detailed description

Implements a helper function

Table 1814:Return values

Name	Description
SSP_SUCCESS	enable IRQ successful
SSP_ERR_TIMEOUT	check for IRQ enable bit for the event timed out

9.46.27 r_rtc_disable_irq

```
ssp_err_t r_rtc_disable_irq (  R_RTC_Type * p_rtc_reg ,   IRQn_Type irq ,
 rtc_event_t  event ,   uint32_t timeout )
```

9.46.27.1 Brief description

enable alarm irq if valid

9.46.27.2 Detailed description

Implements a helper function

Table 1815:Return values

Name	Description
SSP_SUCCESS	disable IRQ successful
SSP_ERR_TIMEOUT	check for IRQ disable bit for the event timed out

9.46.28 r_rtc_error_adjustment_mode_set_common

```
ssp_err_t r_rtc_error_adjustment_mode_set_common (  R_RTC_Type * p_rtc_reg ,
 rtc_error_adjustment_mode_t  mode )
```

9.46.28.1 Brief description

Helper function to set time error adjustment mode.

9.46.28.2 Detailed description

Table 1816:Return values

Name	Description
SSP_SUCCESS	time error adjustment mode set successful
SSP_ERR_TIMEOUT	check for adjustment mode bit timed out

9.46.29 r_rtc_error_adjustment_period_set

```
ssp_err_t r_rtc_error_adjustment_period_set (  R_RTC_Type * p_rtc_reg ,
 rtc_error_adjustment_period_t  period )
```

9.46.29.1 Brief description

Helper function to set time error adjustment period.

9.46.29.2 Detailed description

Table 1817:Return values

Name	Description
SSP_SUCCESS	time error adjustment mode set successful
SSP_ERR_TIMEOUT	check for adjustment mode bit timed out

9.46.30 rtc_dec_to_bcd

```
rtc_dec_to_bcd (  uint8_t to_convert )
```

9.46.30.1 Brief description

RTC Alarm ISR.

9.46.30.2 Detailed description

Saves context if RTOS is used, stops the timer if one-shot mode, clears interrupts, calls callback if one was provided in the open function, and restores context if RTOS is used. Convert decimal to BCD

9.46.31 rtc_bcd_to_dec

```
rtc_bcd_to_dec (  uint8_t to_convert )
```

9.46.31.1 Brief description

Convert BCD to decimal.

9.46.32 API Data

9.46.32.1 rtc_count_mode_t

rtc_count_mode_t

Detailed description

Counting mode

Enumerated values

Name	Description
RTC_CALENDAR_MODE	
RTC_BINARY_MODE	

9.46.33 Extensions

9.46.33.1 rtc_instance_ctrl_t

rtc_instance_ctrl_t

Detailed description

Channel control block. DO NOT INITIALIZE. Initialization occurs when open is called

Variables

- void * p_reg

  Pointer to register base address.
- void(* p_callback)(*cb_data)

  Called from the ISR.
- void const * p_context

  Passed to the callback.
- uint32_t open

  Whether or not driver is open.

- IRQn_Type alarm_irq

  Alarm IRQ number.
- IRQn_Type periodic_irq

  Periodic IRQ number.
- IRQn_Type carry_irq

  Carry IRQ number.
- rtc_clock_source_t clock_source

  Clock source for the RTC block.

9.47 Simple I2C on SCI

Driver for the Simple IIC on SCI.

This module supports the SCI in I2C mode. It implements the following interfaces:

- I2C Master Interface r_i2c_api.h

9.47.1 Functions

- sci_siic_notify
- sci_siic_clock_settings
- sci_siic_sda_delay_settings
- sci_siic_abort_seq_master
- sci_siic_open_hw_master
- sci_siic_close_hw_master
- sci_siic_abort_hw_master
- sci_siic_run_hw_master
- sci_siic_set_irq_parameters
- sci_siic_configure_irqs
- sci_i2c_rxi_isr
- sci_i2c_txi_isr
- sci_i2c_tei_isr
- sci_siic_open_transfer_interface
- sci_siic_reconfigure_interrupts_for_transfer
- sci_siic_stop_transfer_interface
- sci_siic_transfer_configure_rx

- sci_siic_transfer_configure_tx
- sci_siic_enable_transfer_support_tx
- sci_siic_enable_transfer_support_rx
- sci_siic_txi_send_data
- sci_siic_tei_send_address
- sci_siic_open_parameter_check
- sci_siic_tei_handler
- sci_siic_txi_handler
- sci_siic_rxi_handler
- sci_siic_txi_process_nack
- sci_siic_dtc_max_length_check
- R_SCI_SIIC_MasterVersionGet
- R_SCI_SIIC_MasterOpen
- R_SCI_SIIC_MasterClose
- R_SCI_SIIC_MasterRead
- R_SCI_SIIC_MasterWrite
- R_SCI_SIIC_MasterReset
- R_SCI_SIIC_MasterSlaveAddressSet

9.47.2 Variables

- g_sci_siic_master_version
- g_dummy_write_data_for_read_op
- g_i2c_master_on_sci
- sync_baud

9.47.3 Defines

- #define SCI_SIIC_MASTER_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define SCI_SIIC_MASTER_CODE_VERSION_MINOR

  Initial value: `(7U)`
- #define SIIC_OPEN

  Initial value: `(0x83737343ULL)`

  "SIIC" in ASCII, used to determine if channel is open.

- #define MDDR_MIN

  Initial value:`128U`
- #define MDDR_MAX

  Initial value:`256U`
- #define SCI_SIIC_ERROR_RETURN

  Initial value:`#define SSP_ERROR_RETURN((a), (err), &g_module_name[0], NULL)`

  Macro for error logger.
- #define I2C_CODE_READ

  Initial value: `(0x01U)`
- #define I2C_CODE_10BIT

  Initial value: `(0xF0U)`
- #define SCI_SIIC_NUM_DIVISORS_SYNC

  Initial value: `(4U)`
- #define SCI_SIIC_BRR_MAX

  Initial value: `(255U)`
- #define SCI_SIIC_BRR_MIN

  Initial value: `(0U)`
- #define SCI_SIIC_MAX_PCLK

  Initial value: `(0xFFFFFFFFFFFFFFFFULL)`
- #define SCI_I2C_NANOSECONDS_PER_SECOND

  Initial value: `(1000000000U)`
- #define SCI_I2C_SIMR1_IICDL_MAXIMUM_VALUE

  Initial value: `(0x1FU)`

9.47.4 sci_siic_notify

```
 sci_siic_notify ( sci_i2c_instance_ctrl_t  *const p_ctrl ,  i2c_event_t
const event )
```

9.47.4.1 Brief description

Single point for managing the logic around notifying a transfer has finished.

9.47.4.2 Detailed description

(end defgroup SIIC)

Table 1818:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to transfer that is ending.
event	in	The event code to pass to the callback.

9.47.4.3 Function steps

- Set the flag indicating that the transaction is completed

9.47.5 sci_siic_clock_settings

```
sci_siic_clock_settings (  uint32_t *const p_rate ,   uint8_t *const p_brr ,
 uint8_t *const p_divisor ,   uint32_t *const p_mddr ,   uint16_t sda_delay ,
 uint32_t * cycles )
```

9.47.6 sci_siic_sda_delay_settings

```
sci_siic_sda_delay_settings (  uint32_t const clk_divisor ,
 uint16_t sda_delay ,   uint32_t *const p_cycles )
```

9.47.7 sci_siic_abort_seq_master

```
ssp_err_t sci_siic_abort_seq_master ( sci_i2c_instance_ctrl_t  *const p_ctrl )
```

9.47.7.1 Brief description

Single point for managing the logic around aborting a transfer when operating as a master.

9.47.7.2 Detailed description

Table 1819:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control struct of specific device

Table 1820:Return values

Name	Description
SSP_ERR_ABORTED	If there is an in-progress transfer

9.47.8 sci_siic_open_hw_master

```
ssp_err_t sci_siic_open_hw_master ( sci_i2c_instance_ctrl_t  *const p_ctrl )
```

9.47.8.1 Brief description

Performs the hardware initialization sequence when operating as a master.

9.47.8.2 Detailed description

Table 1821:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure of specific device

Table 1822:Return values

Name	Description
SSP_SUCCESS	Hardware initialized with proper configurations
SSP_ERR_INVALID_RATE	The requested rate cannot be set.

9.47.8.3 Function steps

- enables or disable the bitrate modulation function

9.47.9 sci_siic_close_hw_master

```
sci_siic_close_hw_master ( sci_i2c_instance_ctrl_t  *const p_ctrl )
```

9.47.9.1 Brief description

Performs the hardware initialization sequence when operating as a master.

9.47.9.2 Detailed description

Table 1823:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure of specific device

9.47.10 sci_siic_abort_hw_master

```
sci_siic_abort_hw_master ( sci_i2c_instance_ctrl_t  *const p_ctrl )
```

9.47.10.1 Brief description

Safely stops the current data transfer when operating as a master.

9.47.10.2 Detailed description

Table 1824:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to control structure of specific device

9.47.10.3 Function steps

- Disable channel interrupts

9.47.11 sci_siic_run_hw_master

```
ssp_err_t sci_siic_run_hw_master ( sci_i2c_instance_ctrl_t  *const p_ctrl )
```

9.47.11.1 Brief description

Performs the data transfer described by the parameters when operating as a master.

9.47.11.2 Detailed description

Table 1825:

Name	Direction	Description
p_ctrl	in	Pointer to Control structure of specific device.

Table 1826:

Name	Description
SSP_SUCCESS	Data transfered when operating as a master.
SSP_ERR_IN_USE	If a transfer is in progress.
SSP_ERR_ABORTED	If there is an in-progress transfer.

9.47.12 sci_siic_set_irq_parameters

```
 sci_siic_set_irq_parameters (  uint8_t ipl ,   void * p_ctrl ,   IRQn_Type
* p_irq )
```

9.47.12.1 Detailed description

Sets interrupt priority and initializes vector info for the callback

Table 1827:

Name	Direction	Description
ipl	in	Interrupt priority level
p_ctrl	in	Pointer to driver control block
p_irq	in	Pointer to IRQ for this signal

9.47.12.2 Function steps

- Set the interrupt priority for the specified interrupt number(*p_irq)
- Set the p_ctrl into the vector_info structure for the element. This information will be used by the ISR to determine the p_ctrl to be used in the ISR processing.

9.47.13 sci_siic_configure_irqs

```
ssp_err_t sci_siic_configure_irqs (  ssp_feature_t * p_feature ,  i2c_ctrl_t
*const p_api_ctrl ,  i2c_cfg_t  const *const p_cfg )
```

9.47.13.1 Detailed description

Sets interrupt priority and initializes vector info to be used in the callback. Then, Disables and clears interrupts in the NVIC and the peripheral.

Table 1828:

Name	Direction	Description
p_feature	in	SSP feature
p_api_ctrl	in	Pointer to driver control block
p_cfg	in	Pointer to driver configuration block

Table 1829:

Name	Description
SSP_SUCCESS	Interrupt enabled
SSP_ERR_IRQ_BSP_DISABLED	Interrupt does not exist in the vector table

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- eventInfoGet

9.47.13.2 Function steps

- Get the interrupt vector number for SSP_SIGNAL_SCI_RXI
- Save the interrupt vector number to the control block
- Get the interrupt vector number for SSP_SIGNAL_SCI_TXI
- Save the interrupt vector number to the control block
- Get the interrupt vector number for SSP_SIGNAL_SCI_TEI
- Save the interrupt vector number to the control block
- Disable the interrupt in the NVIC and the peripheral
- Clear the IR flag in the ICU
- Clear interrupt flag in the NVIC

- Disable the interrupt in the NVIC and the peripheral
- Clear the IR flag in the ICU
- Clear interrupt flag in the NVIC
- Disable the interrupt in the NVIC and the peripheral
- Clear the IR flag in the ICU
- Clear interrupt flag in the NVIC

9.47.14 sci_i2c_rxi_isr

```
sci_i2c_rxi_isr ( void )
```

9.47.14.1 Brief description

ISR for ACK/RXI interrupt.

9.47.15 sci_i2c_txi_isr

```
sci_i2c_txi_isr ( void )
```

9.47.15.1 Brief description

ISR for NACK/TXI interrupt.

9.47.16 sci_i2c_tei_isr

```
sci_i2c_tei_isr ( void )
```

9.47.16.1 Brief description

Handles the STI interrupt.

9.47.17 sci_siic_open_transfer_interface

```
ssp_err_t sci_siic_open_transfer_interface ( sci_i2c_instance_ctrl_t
*const p_ctrl , i2c_cfg_t const *const p_cfg )
```

9.47.17.1 Brief description

Configures SCI I2C related transfer drivers (if enabled).

9.47.17.2 Detailed description

Table 1830:

Name	Direction	Description
p_ctrl	in	Pointer to SCI I2C specific control structure
p_cfg	in	Pointer to SCI I2C specific configuration structure

Table 1831:

Name	Description
SSP_SUCCESS	If configures SCI I2C related transfer drivers
SSP_ERR_ASSERTION	Transfer configuration for tx/rx not proper.

9.47.18 sci_siic_reconfigure_interrupts_for_transfer

```
ssp_err_t sci_siic_reconfigure_interrupts_for_transfer ( sci_i2c_instance_ctrl_t
  *const p_ctrl )
```

9.47.18.1 Brief description

Reconfigure the address mode for transfer interface.

9.47.18.2 Detailed description

Table 1832:

Name	Direction	Description
p_ctrl	in	transfer control block

Table 1833:

Name	Description
SSP_SUCCESS	Address mode for transfer interface reconfigured.
SSP_ERR_IN_USE	If a transfer is in progress.

9.47.19 sci_siic_stop_transfer_interface

```
sci_siic_stop_transfer_interface ( sci_i2c_instance_ctrl_t  *const p_ctrl )
```

9.47.19.1 Brief description

Stop any running instance of the transfer interface.

9.47.19.2 Detailed description

Table 1834:

Name	Direction	Description
p_ctrl	in	transfer control block

9.47.20 sci_siic_transfer_configure_rx

```
ssp_err_t sci_siic_transfer_configure_rx ( sci_i2c_instance_ctrl_t
*const p_ctrl ,  i2c_cfg_t  const *const p_cfg ,   R_SCI0_Type * p_sci_reg )
```

9.47.20.1 Brief description

Configures SCI I2C RX related transfer (if enabled).

9.47.20.2 Detailed description

Table 1835:

Name	Direction	Description
p_ctrl	in	Pointer to SCI I2C specific control structure

Table 1835: (Continued)

Name	Direction	Description
p_cfg	in	Pointer to SCI I2C specific configuration structure
p_sci_reg	in	Pointer to hardware registers for SCI I2C module

Table 1836:

Name	Description
SSP_SUCCESS	Configurations proper for RX related transfer.
SSP_ERR_ASSERTION	Pointer to the transfer instance for I2C receive in p_cfg is NULL.

9.47.21 sci_siic_transfer_configure_tx

```
ssp_err_t sci_siic_transfer_configure_tx ( sci_i2c_instance_ctrl_t
*const p_ctrl ,  i2c_cfg_t  const *const p_cfg ,   R_SCI0_Type * p_sci_reg )
```

9.47.21.1 Brief description

Configures SCI I2C TX related transfer (if enabled).

9.47.21.2 Detailed description

Table 1837:

Name	Direction	Description
p_ctrl	in	Pointer to SCI I2C specific control structure
p_cfg	in	Pointer to SCI I2C specific configuration structure
p_sci_reg	in	Pointer to hardware registers for SCI I2C module

Table 1838:

Name	Description
SSP_SUCCESS	Configurations proper for TX related transfer.
SSP_ERR_ASSERTION	Pointer to the transfer instance for I2C transmit in p_cfg is NULL.

9.47.22 sci_siic_enable_transfer_support_tx

```
sci_siic_enable_transfer_support_tx ( sci_i2c_instance_ctrl_t  *const p_ctrl )
```

9.47.22.1 Brief description

Enables the dtc transfer interface for the transmit operation.

9.47.22.2 Detailed description

Table 1839:

Name	Direction	Description
p_ctrl	in	Pointer to transfer control block

9.47.23 sci_siic_enable_transfer_support_rx

```
sci_siic_enable_transfer_support_rx ( sci_i2c_instance_ctrl_t  *const p_ctrl )
```

9.47.23.1 Brief description

Enables the dtc transfer interface for the receive operation.

9.47.23.2 Detailed description

Table 1840:

Name	Direction	Description
p_ctrl	in	Pointer to transfer control block

9.47.24 sci_siic_txi_send_data

```
sci_siic_txi_send_data ( sci_i2c_instance_ctrl_t  *const p_ctrl ,  R_SCI0_Type
* p_sci_reg )
```

9.47.24.1 Brief description

Check for the receive condition.

9.47.24.2 Detailed description

Table 1841:

Name	Direction	Description
p_ctrl	in	Pointer to transfer control block
p_sci_reg	in	Pointer to hardware registers for SCI I2C module

9.47.25 sci_siic_tei_send_address

```
sci_siic_tei_send_address ( sci_i2c_instance_ctrl_t  *const p_ctrl ,
 R_SCI0_Type * p_sci_reg )
```

9.47.25.1 Brief description

Enables transfer support while handling the tei interrupt.

9.47.25.2 Detailed description

Table 1842:

Name	Direction	Description
p_ctrl	in	Pointer to transfer control block
p_sci_reg	in	Pointer to hardware registers for SCI I2C module

9.47.26 sci_siic_open_parameter_check

```
ssp_err_t sci_siic_open_parameter_check ( sci_i2c_instance_ctrl_t
*const p_ctrl ,  i2c_cfg_t  const *const p_cfg )
```

9.47.26.1 Brief description

Parameter check for control block and configuration block.

9.47.26.2 Detailed description

Table 1843:

Name	Direction	Description
p_ctrl	in	Pointer to transfer control block
p_cfg	in	Pointer to driver configuration block

Table 1844:

Name	Description
SSP_SUCCESS	If parameter p_ctrl, p_cfg or clock rate not NULL.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_cfg is NULL or if clock rate greater than 400KHz.

9.47.27 sci_siic_tei_handler

```
sci_siic_tei_handler ( sci_i2c_instance_ctrl_t  *const p_ctrl )
```

9.47.28 sci_siic_txi_handler

```
sci_siic_txi_handler ( sci_i2c_instance_ctrl_t  *const p_ctrl )
```

9.47.29 sci_siic_rxi_handler

```
sci_siic_rxi_handler ( sci_i2c_instance_ctrl_t  *const p_ctrl )
```

9.47.30 sci_siic_txi_process_nack

```
sci_siic_txi_process_nack ( sci_i2c_instance_ctrl_t  *const p_ctrl )
```

9.47.30.1 Brief description

Process NACK reception within TXI interrupt.

9.47.30.2 Detailed description

Table 1845:

Name	Direction	Description
p_ctrl	in	Pointer to transfer control block

9.47.31 sci_siic_dtc_max_length_check

```
ssp_err_t sci_siic_dtc_max_length_check ( sci_i2c_instance_ctrl_t
*const p_ctrl ,   const uint32_t bytes )
```

9.47.31.1 Brief description

Parameter check for Read and Write when DTC is used for transfer.

9.47.31.2 Detailed description

Table 1846:

Name	Direction	Description
p_ctrl	in	Pointer to transfer control block
bytes	in	number of bytes to be transferred

Table 1847:

Name	Description
SSP_SUCCESS	Provided length supported by DTC.

Table 1847: (Continued)

Name	Description
SSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size(65535) while DTC is used for data transfer.

9.47.32 R_SCI_SIIC_MasterVersionGet

```
ssp_err_t R_SCI_SIIC_MasterVersionGet ( ssp_version_t  *const p_version )
```

9.47.32.1 Brief description

Sets driver version based on compile time macros.

9.47.32.2 Detailed description

Table 1848:

Name	Description
SSP_SUCCESS	Successful version get.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

9.47.33 R_SCI_SIIC_MasterOpen

```
ssp_err_t R_SCI_SIIC_MasterOpen ( i2c_ctrl_t  *const p_api_ctrl ,  i2c_cfg_t
const *const p_cfg )
```

9.47.33.1 Brief description

Opens the I2C device. Power on I2C peripheral and perform initialization described in hardware manual.

9.47.33.2 Detailed description

This function will reconfigure the clock settings of the peripheral when a device with a lower rate than previously configured is opened.

Table 1849:

Name	Description
SSP_SUCCESS	Requested baud rate was valid.
SSP_ERR_ASSERTION	The parameter p_ctrl or p_cfg is NULL or if clock rate greater than 400KHz.
SSP_ERR_INVALID_RATE	The requested rate cannot be set.
SSP_ERR_IN_USE	Lock was not acquired
SSP_ERR_IRQ_BSP_DISABLED	Event information could not be found.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet
- eventInfoGet

9.47.33.3 Function steps

- Disable, clear and configure interrupts

9.47.34 R_SCI_SIIC_MasterClose

```
ssp_err_t R_SCI_SIIC_MasterClose ( i2c_ctrl_t  *const p_api_ctrl )
```

9.47.34.1 Brief description

Closes the I2C device. Power down I2C peripheral.

9.47.34.2 Detailed description

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

Table 1850:

Name	Description
SSP_SUCCESS	Device closed without issue.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.

Table 1850: (Continued)

Name	Description
SSP_ERR_ABORTED	Device was closed while a transfer was in-progress.
SSP_ERR_NOT_OPEN	Device was not even opened.

9.47.35 R_SCI_SIIC_MasterRead

```
ssp_err_t R_SCI_SIIC_MasterRead ( i2c_ctrl_t  *const p_api_ctrl ,   uint8_t
*const p_dest ,   uint32_t const bytes ,   bool const restart )
```

9.47.35.1 Brief description

Performs a read from the I2C device.

9.47.35.2 Detailed description

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C read operation will begin. When no callback is provided by the user, this function performs a blocking read. Otherwise, the read operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_RX_COMPLETE in the callback.

Table 1851:

Name	Description
SSP_SUCCESS	Function executed without issue.
SSP_ERR_ASSERTION	The parameter p_ctrl, p_dest is NULL, bytes is 0.
SSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size(65535) while DTC is used for data transfer.
SSP_ERR_IN_USE	Another transfer was in-progress.
SSP_ERR_NOT_OPEN	Device was not even opened.

9.47.36 R_SCI_SIIC_MasterWrite

```
ssp_err_t R_SCI_SIIC_MasterWrite ( i2c_ctrl_t  *const p_api_ctrl ,   uint8_t
*const p_src ,   uint32_t const bytes ,   bool const restart )
```

9.47.36.1 Brief description

Performs a write to the I2C device.

9.47.36.2 Detailed description

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C write operation will begin. When no callback is provided by the user, this function performs a blocking write. Otherwise, the write operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_TX_COMPLETE in the callback.

Table 1852:

Name	Description
SSP_SUCCESS	Function executed without issue.
SSP_ERR_ASSERTION	p_ctrl, p_src is NULL.
SSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size(65535) while DTC is used for data transfer.
SSP_ERR_IN_USE	Another transfer was in-progress.
SSP_ERR_NOT_OPEN	Device was not even opened.

9.47.37 R_SCI_SIIC_MasterReset

```
ssp_err_t R_SCI_SIIC_MasterReset ( i2c_ctrl_t  *const p_api_ctrl )
```

9.47.37.1 Brief description

Aborts any in-progress transfer and forces the I2C peripheral into a ready state.

9.47.37.2 Detailed description

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

Table 1853:

Name	Description
SSP_SUCCESS	Channel was reset without issue.

Table 1853: (Continued)

Name	Description
SSP_ERR_ASSERTION	p_ctrl is NULL.
SSP_ERR_ABORTED	A transfer was aborted while resetting the hardware.
SSP_ERR_NOT_OPEN	Device was not even opened.

9.47.38 R_SCI_SIIC_MasterSlaveAddressSet

```
ssp_err_t R_SCI_SIIC_MasterSlaveAddressSet ( i2c_ctrl_t  *const p_api_ctrl ,
  uint16_t const slave ,  i2c_addr_mode_t  const addr_mode )
```

9.47.38.1 Brief description

Sets address and addressing mode of the slave device.

9.47.38.2 Detailed description

This function is used to set the device address and addressing mode of the slave without reconfiguring the entire bus.

Table 1854:

Name	Description
SSP_SUCCESS	Address of the slave is set correctly.
SSP_ERR_ASSERTION	p_ctrl or address is NULL.
SSP_ERR_IN_USE	Another transfer was in-progress.
SSP_ERR_NOT_OPEN	Device was not even opened.

9.47.39 g_sci_siic_master_version

```
ssp_version_t::g_sci_siic_master_version
```

9.47.39.1 Detailed description

Name of module used by error logger macro Version data structure used by error logger macro.

9.47.40 g_dummy_write_data_for_read_op

```
const uint8_t g_dummy_write_data_for_read_op
```

9.47.40.1 Detailed description

constant used as the source location for the DTC dummy write

9.47.41 g_i2c_master_on_sci

i2c_api_master_t::g_i2c_master_on_sci

9.47.41.1 Detailed description

SCI SIIC Implementation of I2C device master interface

9.47.41.2 Initialized as

```
g_i2c_master_on_sci=
{
    .versionGet = R_SCI_SIIC_MasterVersionGet,
    .open       = R_SCI_SIIC_MasterOpen,
    .close      = R_SCI_SIIC_MasterClose,
    .read       = R_SCI_SIIC_MasterRead,
    .write      = R_SCI_SIIC_MasterWrite,
    .reset      = R_SCI_SIIC_MasterReset,
    .slaveAddressSet = R_SCI_SIIC_MasterSlaveAddressSet
}
```

9.47.42 Extensions

9.47.42.1 sci_i2c_instance_ctrl_t

sci_i2c_instance_ctrl_t

Detailed description

I2C control structure. DO NOT INITIALIZE.

Variables

- i2c_cfg_t info

  Information describing I2C device.

- uint32_t open

  Flag to determine if the device is open.

- void * p_reg

  Base register for this channel.
- transfer_instance_t const * p_transfer_tx

  DTC instance for I2C transmit.Set to NULL if unused.

  DTC/DMA support
- transfer_instance_t const * p_transfer_rx

  DTC instance for I2C receive. Set to NULL if unused.
- IRQn_Type rxi_irq

  Receive IRQ number.
- IRQn_Type txi_irq

  Transmit IRQ number.
- IRQn_Type tei_irq

  Transmit end IRQ number.
- uint8_t * p_buff

  Holds the data associated with the transfer
- uint32_t total

  Holds the total number of data bytes to transfer
- uint32_t remain

  Tracks the remaining data bytes to transfer
- uint32_t loaded

  Tracks the number of data bytes written to the register
- uint8_t addr_low

  Holds the last address byte to issue
- uint8_t addr_high

  Holds the first address byte to issue in 10-bit mode
- uint8_t addr_total

  Holds the total number of address bytes to transfer
- uint8_t addr_remain

  Tracks the remaining address bytes to transfer
- uint8_t addr_loaded

  Tracks the number of address bytes written to the register
- bool read

  Holds the direction of the data byte transfer

- bool restart

  Holds whether or not the restart should be issued when done

- bool err

  Tracks whether or not an error occurred during processing

- bool restarted

  Tracks whether or not a restart was issued during the previous transfer

- bool transaction_completed

  Tracks if the transaction started earlier was completed

- bool do_dummy_read
- bool activation_on_rxi

  << Tracks whether a dummy read is issued on the first RX < Tracks whether the transfer is activated on RXI interrupt

- bool activation_on_txi

  < Tracks whether the transfer is activated on TXI interrupt

9.47.42.2 sci_i2c_extended_cfg

sci_i2c_extended_cfg

Detailed description

SCI I2C extended configuration

Variables

- bool bitrate_modulation

  Bitrate Modulation Function enable or disable

9.48 Simple SPI on SCI

Driver for the Simple SPI on SCI.

This module supports simple SPI serial communication using the microcontroller's SCI peripheral. The Interface is defined in r_spi_api.h. This module implements SPI Interface.

9.48.1 Functions

- R_SCI_SPI_Open
- R_SCI_SPI_Read
- R_SCI_SPI_Write

- R_SCI_SPI_WriteRead
- r_sci_spi_write_read_common
- R_SCI_SPI_Close
- R_SCI_SPI_VersionGet

9.48.2 Variables

- g_spi_on_sci

9.48.3 Defines

- #define SCI_SPI_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define SCI_SPI_CODE_VERSION_MINOR

  Initial value: `(9U)`

9.48.4 R_SCI_SPI_Open

```
ssp_err_t R_SCI_SPI_Open ( spi_ctrl_t  * p_api_ctrl ,  spi_cfg_t  const
*const p_cfg )
```

9.48.4.1 Brief description

Initialize a channel for SPI communication mode. Implements open. This function performs the following tasks: Performs parameter checking and processes error conditions. Applies power to the SPI channel. Disables interrupts. Initializes the associated registers with default value and the user-configurable options. Provides the channel handle for use with other API functions. Updates user-configurable file if necessary.

9.48.4.2 Detailed description

Table 1855:

Name	Description
SSP_SUCCESS	Channel initialized successfully.
SSP_ERR_ASSERTION	One of the following invalid parameter passed. • Pointer p_api_ctrl is NULL • Pointer p_cfg is NULL
SSP_ERR_IN_USE	Channel currently in operation; Close channel first.

Table 1855: (Continued)

Name	Description
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.
See	Common Error Codes or functions called by this function for other possible return codes. This function calls: • productFeatureGet

NOTE: This function is reentrant.

NOTE: The bit-rate argument in p_cfg ranges from 2500 to 7.5m for Simple SPI at PCLK=120 MHz. For RSPI, BRDV is fixed at 0 to get the maximum bit rate. The range is 10.0 mbps to 30.0 mbps at PCLK=120.0 MHz.

9.48.5 R_SCI_SPI_Read

```
ssp_err_t R_SCI_SPI_Read ( spi_ctrl_t  *const p_api_ctrl ,   void const
* p_dest ,   uint32_t const length ,  spi_bit_width_t  const bit_width )
```

9.48.5.1 Brief description

Receive data from an SPI device. Implements read. The function performs the following tasks:

9.48.5.2 Detailed description

- Performs parameter checking and processes error conditions.
- Disable Interrupts.
- Set-up data bit width per user request.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission with dummy data via transmit buffer empty interrupt.
- Copy data from source buffer to the SPI data register for transmission.
- Receive data from receive buffer full interrupt occurs and copy data to the buffer of destination.
- Complete data reception via receive buffer full interrupt and transmitting dummy data.
- Disable transmitter.

- Disable receiver.
- Disable interrupts.

Table 1856:

Name	Description
SSP_SUCCESS	Read operation successfully completed.
SSP_ERR_ASSERTION	One of the following invalid parameters passed • Pointer p_api_ctrl is NULL • Bit width is not 8 bits • Length is equal to 0 • Pointer to destination is NULL
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.
SSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
See	Common Error Codes or functions called by this function for other possible return codes. This function calls: • reset

NOTE: This function is reentrant.

9.48.6 R_SCI_SPI_Write

```
ssp_err_t R_SCI_SPI_Write ( spi_ctrl_t  *const p_api_ctrl ,   void const
* p_src ,   uint32_t const length ,  spi_bit_width_t  const bit_width )
```

9.48.6.1 Brief description

Transmit data to a SPI device. Implements write.

9.48.6.2 Detailed description

- The function performs the following tasks:
- Performs parameter checking and processes error conditions.
- Disable Interrupts.

- Setup data bit width per user request.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission with data via transmit buffer empty interrupt.
- Copy data from source buffer to the SPI data register for transmission.
- Receive data from receive buffer full interrupt occurs and do nothing with the received data.
- Complete data transmission via receive buffer full interrupt.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

Table 1857:

Name	Description
SSP_SUCCESS	Write operation successfully completed.
SSP_ERR_ASSERTION	One of the following invalid parameters passed • Pointer p_api_ctrl is NULL • Pointer to source is NULL • Length is equal to 0 • Bit width is not equal to 8 bits
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.
SSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
See	Common Error Codes or functions called by this function for other possible return codes. This function calls: • reset

NOTE: This function is reentrant.

9.48.7 R_SCI_SPI_WriteRead

```
ssp_err_t R_SCI_SPI_WriteRead ( spi_ctrl_t  *const p_api_ctrl ,   void const
* p_src ,   void const * p_dest ,   uint32_t const length ,  spi_bit_width_t
const bit_width )
```

9.48.7.1 Brief description

Simultaneously transmit data to SPI device while receiving data from SPI device (full duplex). Implements writeRead. The function performs the following tasks:

9.48.7.2 Detailed description

- Performs parameter checking and processes error conditions.
- Disable Interrupts. Setup data bit width per user request. Enable transmitter. Enable receiver. Enable interrupts. Start data transmission using transmit buffer empty interrupt. Copy data from source buffer to the SPI data register for transmission. Receive data from receive buffer full interrupt occurs and copy data to the buffer of destination. Complete data transmission and reception via receive buffer full interrupt. Disable transmitter. Disable receiver. Disable interrupts.

Table 1858:

Name	Description
SSP_SUCCESS	Write operation successfully completed.
SSP_ERR_ASSERTION	One of the following invalid parameters passed • Pointer p_api_ctrl is NULL • Pointer to source is NULL • Pointer to destination is NULL • Length is equal to 0 • Bit width is not equal to 8 bits
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.
SSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
See	Common Error Codes or functions called by this function for other possible return codes. This function calls: • reset

NOTE: This function is reentrant.

9.48.8 r_sci_spi_write_read_common

```
ssp_err_t r_sci_spi_write_read_common ( sci_spi_instance_ctrl_t  *const p_ctrl ,
  void const * p_src ,   void const * p_dest ,   uint32_t const length ,
 spi_bit_width_t  const bit_width ,  spi_operation_t  tx_rx_mode )
```

9.48.8.1 Brief description

Initiates writ or read process. Common routine used by RSPI API write or read functions.

9.48.8.2 Detailed description

Table 1859:

Name	Direction	Description
p_ctrl	in	Pointer to the control block.
p_src	in	Pointer to data buffer which need to be sent.
p_dest	out	Pointer to buffer where received data will be stored.
length	in	Number of data transactions to be performed.
bit_width	in	Size of data for each transaction.
tx_rx_mode	in	Mode of the data transaction.

Table 1860:

Name	Description
SSP_SUCCESS	Operation successfully completed.
SSP_ERR_HW_LOCKED	The lock could not be acquired. The channel is busy.
SSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.

Table 1860: (Continued)

Name	Description
See	Common Error Codes or functions called by this function for other possible return codes. This function calls: • reset

NOTE: This function is reentrant.

9.48.9 R_SCI_SPI_Close

```
ssp_err_t R_SCI_SPI_Close ( spi_ctrl_t  *const p_api_ctrl )
```

9.48.9.1 Brief description

Handle the closing of a channel by the following task. Implements close Power off the channel. Disables all the associated interrupts. Update channel status.

9.48.9.2 Detailed description

Table 1861:

Name	Description
SSP_SUCCESS	Channel successfully closed.
SSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
SSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.

NOTE: This function is reentrant.

9.48.10 R_SCI_SPI_VersionGet

```
ssp_err_t R_SCI_SPI_VersionGet ( ssp_version_t  * p_version )
```

9.48.10.1 Brief description

Get the version information of the underlying driver. Implements versionGet.

9.48.10.2 Detailed description

Table 1862:

Name	Description
SSP_SUCCESS	Successful version get.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

NOTE: This function is reentrant.

9.48.11 g_spi_on_sci

`spi_api_t::g_spi_on_sci`

9.48.11.1 Detailed description

Filled in Interface API structure for this Instance.

9.48.12 Extensions

9.48.12.1 sci_spi_instance_ctrl_t

sci_spi_instance_ctrl_t

Detailed description

SPI instance control block. DO NOT INITIALIZE.

Variables

- uint8_t channel

  Channel number to be used.
- uint32_t channel_opened

  Internal flag to indicate the peripheral was initialized.
- transfer_instance_t const * p_transfer_tx

  To use SPI DTC/DMA write transfer.

- transfer_instance_t const * p_transfer_rx

  To use SPI DTC/DMA read transfer.
- void(* p_callback)(*p_args)

  Pointer to user callback function.
- void const * p_context

  Pointer to the higher level device context.
- void * p_reg

  Base register for this channel.
- IRQn_Type rxi_irq

  Receive IRQ number.
- IRQn_Type txi_irq

  Transmit IRQ number.
- IRQn_Type tei_irq

  Transmit end IRQ number.
- IRQn_Type eri_irq

  Error IRQ number.
- void * p_src
- void * p_dest
- uint32_t tx_count
- uint32_t rx_count
- uint32_t xfr_length
- transfer_addr_mode_t tx_dtc_addr_mode
- uint8_t bytes_per_transfer
- bool do_tx
- spi_operation_t transfer_mode
- bsp_lock_t resource_lock_tx_rx

  Resource lock for transmission/reception

9.48.12.2 sci_spi_extended_cfg

sci_spi_extended_cfg

Detailed description

SCI SPI extended configuration

Variables

- bool bitrate_modulation

  Bitrate Modulation Function enable or disable

9.49 UART on SCI

Driver for the UART on SCI.

9.49.1 Summary

This module supports the UART on SCI. It implements the UART interface and drives SCI as a full-duplex UART communication port. This module can drive all SCI channels as UART ports.

Extends UART Interface.

NOTE: This module can use either the 16-stage hardware FIFO or a DTC transfer implementation to write multiple bytes.

9.49.2 Functions

- R_SCI_UartOpen
- R_SCI_UartClose
- R_SCI_UartRead
- R_SCI_UartWrite
- R_SCI_UartBaudSet
- R_SCI_UartInfoGet
- R_SCI_UartVersionGet
- R_SCI_UartAbort

9.49.3 Defines

- #define SCI_UART_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define SCI_UART_CODE_VERSION_MINOR

  Initial value: `(6U)`

9.49.4 R_SCI_UartOpen

```
ssp_err_t R_SCI_UartOpen ( uart_ctrl_t  *const p_api_ctrl ,  uart_cfg_t  const
*const p_cfg )
```

9.49.4.1 Detailed description

Configures the UART driver based on the input configurations. If reception is enabled at compile time, reception is enabled at the end of this function.

Table 1863:

Name	Description
SSP_SUCCESS	Channel opened successfully.
SSP_ERR_ASSERTION	Pointer to UART control block or configuration structure is NULL.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter setting found in the configuration structure.
SSP_ERR_IN_USE	Control block has already been opened or channel is being used by another instance. Call close then open() to reconfigure.
SSP_ERR_IRQ_BSP_DISABLED	A required interrupt does not exist in the vector table

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet
- eventInfoGet
- systemClockFreqGet
- open

9.49.4.2 Function steps

- Make sure this channel exists.
- Reserve the hardware lock.
- Determine if this channel has a FIFO.
- Calculate the baud rate register settings.
- Configure the interrupts.
- Configure the transfer interface for transmission and reception if provided.
- Enable the SCI channel and reset the registers to their initial state.
- Set the default level of the TX pin to 1.
- Set the baud rate registers.
- Set the UART configuration settings provided in uart_cfg_t and uart_on_sci_cfg_t.
- If reception is enabled at build time, enable reception.

9.49.5 R_SCI_UartClose

```
ssp_err_t R_SCI_UartClose ( uart_ctrl_t  *const p_api_ctrl )
```

9.49.5.1 Detailed description

Disables interrupts, receiver, and transmitter. Closes lower level transfer drivers if used. Removes power and releases hardware lock.

Table 1864:

Name	Description
SSP_SUCCESS	Channel successfully closed.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL.
SSP_ERR_NOT_OPEN	The control block has not been opened

9.49.5.2 Function steps

- Mark the channel not open so other APIs cannot use it.
- Disable interrupts, receiver, and transmitter.
- If reception is enabled at build time, disable reception irqs.
- If transmission is enabled at build time, disable transmission irqs.
- Disable baud clock output.
- Close the lower level transfer instances.
- Clear control block parameters.
- Remove power to the channel.
- Unlock the SCI channel.

9.49.6 R_SCI_UartRead

```
ssp_err_t R_SCI_UartRead ( uart_ctrl_t  *const p_api_ctrl ,   uint8_t const
*const p_dest ,   uint32_t const bytes )
```

9.49.6.1 Detailed description

Receives user specified number of bytes into destination buffer pointer.

Table 1865:

Name	Description
SSP_SUCCESS	Data reception successfully ends.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used
SSP_ERR_INVALID_ARGUMENT	Destination address or data size is not valid for 9-bit mode.
SSP_ERR_NOT_OPEN	The control block has not been opened
SSP_ERR_UNSUPPORTED	SCI_UART_CFG_RX_ENABLE is set to 0

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- reset

NOTE: This API is only valid when SCI_UART_CFG_RX_ENABLE is enabled. If 9-bit data length is specified at R_SCI_UartOpen call, p_dest must be aligned 16-bit boundary.

9.49.6.2 Function steps

- Configure transfer instance to receive the requested number of bytes if transfer is used for reception.
- Do nothing in non-transfer case. Bytes will come in through the callback.

9.49.7 R_SCI_UartWrite

```
ssp_err_t R_SCI_UartWrite ( uart_ctrl_t  *const p_api_ctrl ,   uint8_t const
*const p_src ,   uint32_t const bytes )
```

9.49.7.1 Detailed description

Transmits user specified number of bytes from the source buffer pointer.

Table 1866:

Name	Description
SSP_SUCCESS	Data transmission finished successfully.

Table 1866: (Continued)

Name	Description
SSP_ERR_ASSERTION	Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used
SSP_ERR_INVALID_ARGUMENT	Source address or data size is not valid for 9-bit mode.
SSP_ERR_NOT_OPEN	The control block has not been opened
SSP_ERR_IN_USE	A UART transmission is in progress
SSP_ERR_UNSUPPORTED	SCI_UART_CFG_TX_ENABLE is set to 0

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- reset

NOTE: This API is only valid when SCI_UART_CFG_TX_ENABLE is enabled. If 9-bit data length is specified at R_SCI_UartOpen call, p_src must be aligned on a 16-bit boundary.

9.49.7.2 Function steps

- Transmit interrupts must be disabled to start with.
- Save data to transmit to the control block. It will be transmitted in the TXI ISR.
- If a transfer instance is used for transmission, reset the transfer instance to transmit the requested data.
- Trigger a TXI interrupt. This triggers the transfer instance or a TXI interrupt if the transfer instance is not used.

9.49.8 R_SCI_UartBaudSet

```
ssp_err_t R_SCI_UartBaudSet ( uart_ctrl_t  *const p_api_ctrl ,   uint32_t
const baudrate )
```

9.49.8.1 Detailed description

Updates the baud rate.

ATTENTION: This terminates any in-progress transmission.

Table 1867:

Name	Description
SSP_SUCCESS	Baud rate was successfully changed.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL or the UART is not configured to use the internal clock.
SSP_ERR_INVALID_ARGUMENT	Illegal baud rate value is specified.
SSP_ERR_NOT_OPEN	The control block has not been opened

9.49.8.2 Function steps

- Calculate new baud rate register settings.
- Disables transmitter and receiver. This terminates any in-progress transmission.
- Apply new baud rate register settings.
- Enable receiver.

9.49.9 R_SCI_UartInfoGet

```
ssp_err_t R_SCI_UartInfoGet ( uart_ctrl_t  *const p_api_ctrl ,  uart_info_t
*const p_info )
```

9.49.9.1 Detailed description

Provides the driver information, including the maximum number of bytes that can be received or transmitted at a time.

Table 1868:

Name	Description
SSP_SUCCESS	Information stored in provided p_info.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL.
SSP_ERR_NOT_OPEN	The control block has not been opened

9.49.9.2 Function steps

- Store number of bytes that can be read at a time.
- Store number of bytes that can be written at a time.

9.49.10 R_SCI_UartVersionGet

```
ssp_err_t R_SCI_UartVersionGet ( ssp_version_t  * p_version )
```

9.49.10.1 Detailed description

Provides API and code version in the user provided pointer.

Table 1869:

Name	Direction	Description
p_version	in	Version number set here

Table 1870:

Name	Description
SSP_SUCCESS	Version information stored in provided p_version.
SSP_ERR_ASSERTION	p_version is NULL.

9.49.11 R_SCI_UartAbort

```
ssp_err_t R_SCI_UartAbort ( uart_ctrl_t  *const p_api_ctrl ,
 uart_dir_t  communication_to_abort )
```

9.49.11.1 Detailed description

Provides API to abort ongoing transfer. Transmission is aborted after the current character is transmitted. Reception is still enabled after abort(). Any characters received after abort() and before the transfer is reset in the next call to read(), will arrive via the callback function with event UART_EVENT_RX_CHAR.

Table 1871:

Name	Description
SSP_SUCCESS	UART transaction aborted successfully.
SSP_ERR_ASSERTION	Pointer to UART control block is NULL.
SSP_ERR_NOT_OPEN	The control block has not been opened.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- disable

9.49.12 API Data

9.49.12.1 sci_clk_src_t

sci_clk_src_t

Detailed description

Enumeration for SCI clock source

Enumerated values

Name	Description
SCI_CLK_SRC_INT	Use internal clock for baud generation.
SCI_CLK_SRC_EXT	Use external clock for baud generation.
SCI_CLK_SRC_EXT8X	Use external clock 8x baud rate.
SCI_CLK_SRC_EXT16X	Use external clock 16x baud rate.

9.49.12.2 sci_uart_rx_fifo_trigger_t

sci_uart_rx_fifo_trigger_t

Detailed description

Receive FIFO trigger configuration.

Enumerated values

Name	Description
SCI_UART_RX_FIFO_TRIGGER_1	Callback after each byte is received without buffering.
SCI_UART_RX_FIFO_TRIGGER_MAX	Callback when FIFO is full or after 15 bit times with no data (fewer interrupts)

9.49.13 Extensions

9.49.13.1 sci_uart_instance_ctrl_t

sci_uart_instance_ctrl_t

Detailed description

UART instance control block.

Variables

- uint8_t channel

  Channel number.

- uint8_t fifo_depth

  FIFO depth of the UART channel.

- uint8_t rx_transfer_in_progress

  Set to 1 if a receive transfer is in progress, 0 otherwise.

- uint8_t data_bytes

  1 byte for 7 or 8 bit data, 2 bytes for 9 bit data

- uint8_t bitrate_modulation

  1 if bit rate modulation is enabled, 0 otherwise

- uint32_t open

  Used to determine if the channel is configured.

- transfer_instance_t const * p_transfer_rx

  Optional transfer instance used to send or receive multiple bytes without interrupts.

- transfer_instance_t const * p_transfer_tx

  Optional transfer instance used to send or receive multiple bytes without interrupts.

- uint8_t const * p_tx_src

  Source buffer pointer used to fill hardware FIFO from transmit ISR.

- uint32_t tx_src_bytes

  Size of source buffer pointer used to fill hardware FIFO from transmit ISR.

- IRQn_Type rxi_irq

  Receive IRQ number.

- IRQn_Type txi_irq

  Transmit IRQ number.

- IRQn_Type tei_irq

  Transmit end IRQ number.

- IRQn_Type eri_irq

  Error IRQ number.

- void(* p_callback)(*p_args)

  Pointer to callback function.

- void const * p_context

  Pointer to user interrupt context data.

- void * p_reg

  Base register for this channel.

- void(* p_extpin_ctrl)(uint32_t channel, uint32_t level)

  External pin control.

9.49.13.2 uart_on_sci_cfg_t

uart_on_sci_cfg_t

Detailed description

UART on SCI device Configuration

Variables

- sci_clk_src_t clk_src

  Use SCI_CLK_SRC_INT/EXT8X/EXT16X.

- bool baudclk_out

  Baud rate clock output enable.

- bool rx_edge_start

  Start reception on falling edge.

- bool noisecancel_en

  Noise cancel enable.

- sci_uart_rx_fifo_trigger_t rx_fifo_trigger

  Receive FIFO trigger level, unused if channel has no FIFO or if DTC is used.

- void(* p_extpin_ctrl)(uint32_t channel, uint32_t level)

  Pointer to the user callback function to control external GPIO pin control used as RTS signal.channelThe UART channel used. levelWhen level is 0, assert RTS. When level is 1, deassert RTS.

- bool bitrate_modulation

  Bitrate Modulation Function enable or disable.

9.50 Sigma Delta ADC (SDADC)

Driver for the 24-bit Sigma Delta A/D Converter (SDADC).

This module supports the SDADC peripheral. It implements the following interfaces:

- ADC Interface

9.50.1 Functions

- R_SDADC_Open
- R_SDADC_ScanConfigure
- R_SDADC_SampleStateCountSet
- R_SDADC_InfoGet
- R_SDADC_ScanStart
- R_SDADC_ScanStop
- R_SDADC_CheckScanDone
- R_SDADC_Read
- R_SDADC_Read32
- R_SDADC_OffsetSet
- R_SDADC_Calibrate
- R_SDADC_Close
- R_SDADC_VersionGet

9.50.2 Defines

- #define SDADC_CODE_VERSION_MAJOR

  Initial value: `(1U)`

  Version of code that implements the API defined in this file
- #define SDADC_CODE_VERSION_MINOR

  Initial value: `(0U)`
- #define SDADC_MAX_NUM_CHANNELS

  Initial value: `(5U)`

9.50.3 R_SDADC_Open

```
ssp_err_t R_SDADC_Open ( adc_ctrl_t  * p_api_ctrl ,  adc_cfg_t  const
*const p_cfg )
```

9.50.3.1 Detailed description

Applies power to the SDADC and initializes the hardware based on the user configuration. As part of this initialization, the SDADC clock is configured and enabled. If an interrupt priority is non-zero, enables an interrupt which will call a callback to notify the user when a conversion, scan, or calibration is complete. For all channels that are configured in differential mode, calibration is performed unless it is disabled in the user configuration. Implements open.

Table 1872:

Name	Description
SSP_SUCCESS	Configuration and calibration successful.
SSP_ERR_CALIBRATE_FAILED	Calibration failed.
SSP_ERR_ASSERTION	An input pointer is NULL.
SSP_ERR_INVALID_ARGUMENT	An input argument is invalid.
SSP_ERR_IN_USE	Control block is already open.
SSP_ERR_IRQ_BSP_DISABLED	A required interrupt is disabled

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet

9.50.3.2 Function steps

- Ensure the SDADC exists on the hardware, enable interrupts, and reserve the SDADC hardware lock.
- Initialize the hardware based on the configuration.
- Start the SDADC.
- Configure the SDADC clock source and divisor to the clock source selected in the clock configuration.
- Set the reference voltage for sensors (internal or external VREF mode).
- Set the A/D conversion operation mode (normal or low power mode).
- Supply the 24-bit sigma-delta A/D converter clock (SDADCCLK).
- Turn on the power of ADBGR, SBIAS, and ADREG.
- Stabilization wait time of 2 ms is required between power on of ADBGR/SBIAS/ADREG and VBIAS/channel/SDADC.
- Turn on the power of VBIAS, channel, and sigma-delta A/D converter.
- For each channel:
- Set the oversampling ratio.
- Set the gain.
- Select single-end input/differential input.
- If differential mode is used and calibration during open is not skipped, enable calibration on this channel.
- Select the polarity of single-end input (only for single-end mode).
- Set the A/D conversion count.

- Select the averaging operation. Select the average data count.
- Select whether to reverse the A/D conversion results of single-end input (only for negative-side single
- end mode).
- Enable conversion for the channel.
- Configure enabled channels and autoscan mode.
- If the A/D conversion trigger is ELC hardware events, the hardware events are enabled in scanStart.
- Mark driver as opened.
- If at least one channel is configured for differential mode and calibration is not disabled in the user configuration, calibrate all PGAs configured for differential mode.
- The calibration takes approximately 3.4 ms per channel in normal mode or 27 ms in low power mode. The open() API blocks waiting for calibration to complete unless calibration is skipped. Calibration during open can be skipped and handled in at the application level by setting skip_internal_calibration to true, then calling calibrate.

9.50.4 R_SDADC_ScanConfigure

```
ssp_err_t R_SDADC_ScanConfigure ( adc_ctrl_t  * p_api_ctrl ,  adc_channel_cfg_t
const *const p_channel_cfg )
```

9.50.4.1 Detailed description

Configures the enabled channels of the ADC. Only scan_mask is used. Implements scanCfg.

Table 1873:

Name	Description
SSP_SUCCESS	Information stored in p_adc_info.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.
SSP_ERR_INVALID_ARGUMENT	An input argument is invalid.

9.50.4.2 Function steps

- Update the enabled channels.

9.50.5 R_SDADC_SampleStateCountSet

```
ssp_err_t R_SDADC_SampleStateCountSet ( adc_ctrl_t  * p_api_ctrl ,
 adc_sample_state_t  * p_sample )
```

9.50.5.1 Detailed description

sampleStateCountSet is not supported on the SDADC.

Table 1874:

Name	Description
SSP_ERR_UNSUPPORTED	This API is not supported.

9.50.5.2 Function steps

- Return the unsupported error.

9.50.6 R_SDADC_InfoGet

```
ssp_err_t R_SDADC_InfoGet ( adc_ctrl_t  * p_api_ctrl ,  adc_info_t
* p_adc_info )
```

9.50.6.1 Detailed description

Returns the address of the lowest number configured channel, the total number of results to be read in order to read the results of all configured channels, the size of each result, and the ELC event enumerations. Implements infoGet.

Table 1875:

Name	Description
SSP_SUCCESS	Information stored in p_adc_info.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.50.6.2 Function steps

- Retrieve the scan mask of active channels from the control block
- Determine the lowest channel that is configured.
- Determine the highest channel that is configured.
- Determine the size of data that must be read to read all the channels between and including the highest and lowest channels.
- Determine the base address and transfer size.
- Specify the peripheral name in the ELC list

- Set sensor information to invalid value

9.50.7 R_SDADC_ScanStart

```
ssp_err_t R_SDADC_ScanStart ( adc_ctrl_t  * p_api_ctrl )
```

9.50.7.1 Detailed description

If the SDADC is configured for hardware triggers, enables hardware triggers. Otherwise, starts a scan. Implements scanStart.

Table 1876:

Name	Description
SSP_SUCCESS	Scan started or hardware triggers enabled successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.
SSP_ERR_IN_USE	A conversion or calibration is in progress.

9.50.7.2 Function steps

- Conversion cannot be performed if conversion or calibration is already in progress.
- If the unit is configured for software triggering, trigger a scan.
- Otherwise, enable hardware triggers.
- Return the error code

9.50.8 R_SDADC_ScanStop

```
ssp_err_t R_SDADC_ScanStop ( adc_ctrl_t  * p_api_ctrl )
```

9.50.8.1 Detailed description

If the SDADC is configured for hardware triggers, disables hardware triggers. Otherwise, stops any in-progress scan started by software. Implements scanStop.

Table 1877:

Name	Description
SSP_SUCCESS	Scan stopped or hardware triggers disabled successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.50.8.2 Function steps

- If the unit is configured for software triggering, stop the scan.
- Otherwise, disable hardware triggers.
- Return the error code

9.50.9 R_SDADC_CheckScanDone

```
ssp_err_t R_SDADC_CheckScanDone ( adc_ctrl_t  * p_api_ctrl )
```

9.50.9.1 Detailed description

Returns the status of a scan started by software, including calibration scans. It is not possible to determine the status of a scan started by a hardware trigger. Implements scanStatusGet.

Table 1878:

Name	Description
SSP_SUCCESS	No software scan or calibration is in progress.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_IN_USE	A conversion or calibration is in progress.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.50.9.2 Function steps

- If calibration is in progress, return an error.
- Return the scan status of a scan triggered by software.

9.50.10 R_SDADC_Read

```
ssp_err_t R_SDADC_Read ( adc_ctrl_t  * p_api_ctrl ,  adc_register_t
const reg_id ,   uint16_t *const p_data )
```

9.50.10.1 Detailed description

Reads the most recent conversion result from a channel. Truncates 24-bit results to the upper 16 bits. Implements read.

Table 1879:

Name	Description
SSP_SUCCESS	Conversion result in p_data.
SSP_ERR_INVALID_ARGUMENT	An input argument was outside the valid range.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.50.10.2 Function steps

- Read the most recent conversion value into a 16-bit result.

9.50.11 R_SDADC_Read32

```
ssp_err_t R_SDADC_Read32 ( adc_ctrl_t  * p_api_ctrl ,  adc_register_t
const reg_id ,   uint32_t *const p_data )
```

9.50.11.1 Detailed description

Reads the most recent conversion result from a channel. Implements read32.

Table 1880:

Name	Description
SSP_SUCCESS	Conversion result in p_data.
SSP_ERR_INVALID_ARGUMENT	An input argument was outside the valid range.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.50.11.2 Function steps

- Read the most recent conversion value into a 32-bit result.

9.50.12 R_SDADC_OffsetSet

```
ssp_err_t R_SDADC_OffsetSet ( adc_ctrl_t  *const p_api_ctrl ,  adc_register_t
const reg_id ,   int32_t const offset )
```

9.50.12.1 Detailed description

Sets the offset. Offset is applied after stage 1 of the input channel. Offset can only be applied when the channel is configured for differential input. Implements offsetSet.

Note: The offset is cleared if calibrate is called. The offset can be re-applied if necessary after the the callback with event ADC_EVENT_CALIBRATION_COMPLETE is called.

Table 1881:

Name	Direction	Description
p_api_ctrl	in	See p_ctrl in offsetSet.
reg_id	in	See reg_id in offsetSet.
offset	in	Must be between -15 and 15, offset (mV) = 10.9376 mV * offset_steps / stage 1 gain.

Table 1882:

Name	Description
SSP_SUCCESS	Offset updated successfully.
SSP_ERR_INVALID_ARGUMENT	An input argument was outside the valid range.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_IN_USE	A conversion or calibration is in progress.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.50.12.2 Function steps

- Offset cannot be updated if conversion or calibration is in progress.

- Set the offset.

9.50.13 R_SDADC_Calibrate

```
ssp_err_t R_SDADC_Calibrate ( adc_ctrl_t  *const p_api_ctrl ,   void
*const p_extend )
```

9.50.13.1 Detailed description

Requires sdadc_calibrate_args_t passed to p_extend. Calibrates the specified channel. Calibration is not required or supported for single-ended mode. Internal calibration is automatically done during open() unless it is disabled in the user configuration. This API is provided primarily for external calibration. A callback with the event ADC_EVENT_CALIBRATION_COMPLETE is called when calibration completes. The calibration status can also be monitored using adc_api_t::statusGet(). Implements calibrate.

During external offset calibration, apply a differential voltage of 0 to ANSDnP - ANSDnN, where n is the input channel and ANSDnP is OPAMP0 for channel 4 and ANSDnN is OPAMP1 for channel 4. Complete external offset calibration before external gain calibration.

During external gain calibration apply a voltage between 0.4 V / total_gain and 0.8 V / total_gain. The differential voltage applied during calibration is corrected to a conversion result of 0x7FFFFF, which is the maximum possible positive differential measurement.

This function clears the offset value. If offset is required after calibration, it must be reapplied after calibration is complete using offsetSet.

Table 1883:

Name	Description
SSP_SUCCESS	Calibration began successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_IN_USE	A conversion or calibration is in progress.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.50.13.2 Function steps

- Calibration cannot be performed if conversion or calibration is already in progress.
- Select software trigger.
- Select single scan mode.
- Enable calibration.
- Set the offset voltage to 0 mV.
- Set the calibration mode.
- Start calibration.

- Confirm that calibration started.
- Return the error code

9.50.14 R_SDADC_Close

```
ssp_err_t R_SDADC_Close ( adc_ctrl_t  * p_api_ctrl )
```

9.50.14.1 Detailed description

Stops any scan in progress, disables interrupts, and powers down the SDADC peripheral. Implements close.

Table 1884:

Name	Description
SSP_SUCCESS	Instance control block closed successfully.
SSP_ERR_ASSERTION	An input pointer was NULL.
SSP_ERR_NOT_OPEN	Instance control block is not open.

9.50.14.2 Function steps

- Perform parameter checking
- Mark driver as closed
- Disable hardware triggers.
- Stop A/D conversion.
- Disable interrupts.
- Wait 3 us in normal mode as required by the hardware manual.
- Turn off the power of VBIAS, channel, and sigma-delta A/D converter.
- Turn off the power of ADBGR, SBIAS, and ADREG.
- Stop the input clock for the 24-bit sigma-delta A/D converter (SDADCCLK).
- Enter the module-stop state.
- Release the hardware lock
- Return the error code

9.50.15 R_SDADC_VersionGet

```
ssp_err_t R_SDADC_VersionGet ( ssp_version_t  *const p_version )
```

9.50.15.1 Detailed description

Gets the API and code version. Implements versionGet.

Table 1885:

Name	Description
SSP_SUCCESS	Version information available in p_version.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

9.50.15.2 Function steps

- Return the version number

9.50.16 API Data

9.50.16.1 sdadc_vref_src_t

`sdadc_vref_src_t`

Detailed description

Source of Vref.

Enumerated values

Name	Description
SDADC_VREF_SRC_INTERNAL	Vref is internally sourced, can be output as SBIAS.
SDADC_VREF_SRC_EXTERNAL	Vref is externally sourced from the VREFI pin.

9.50.16.2 sdadc_vref_voltage_t

`sdadc_vref_voltage_t`

Detailed description

Voltage of Vref.

Enumerated values

Name	Description
SDADC_VREF_VOLTAGE_800_MV	Vref is 0.8 V.

Name	Description
SDADC_VREF_VOLTAGE_1000_MV	Vref is 1.0 V.
SDADC_VREF_VOLTAGE_1200_MV	Vref is 1.2 V.
SDADC_VREF_VOLTAGE_1400_MV	Vref is 1.4 V.
SDADC_VREF_VOLTAGE_1600_MV	Vref is 1.6 V.
SDADC_VREF_VOLTAGE_1800_MV	Vref is 1.8 V.
SDADC_VREF_VOLTAGE_2000_MV	Vref is 2.0 V.
SDADC_VREF_VOLTAGE_2200_MV	Vref is 2.2 V.
SDADC_VREF_VOLTAGE_2400_MV	Vref is 2.4 V (only valid for external Vref)

9.50.16.3 sdadc_channel_input_t

`sdadc_channel_input_t`

Detailed description

Per channel input mode.

Enumerated values

Name	Description
SDADC_CHANNEL_INPUT_DIFFERENTIAL	Differential input.
SDADC_CHANNEL_INPUT_SINGLE_ENDED	Single-ended input.

9.50.16.4 sdadc_channel_stage_1_gain_t

`sdadc_channel_stage_1_gain_t`

Detailed description

Per channel stage 1 gain options.

Enumerated values

Name	Description
SDADC_CHANNEL_STAGE_1_GAIN_1	Gain of 1.
SDADC_CHANNEL_STAGE_1_GAIN_2	Gain of 2.

Name	Description
SDADC_CHANNEL_STAGE_1_GAIN_3	Gain of 3 (only valid for stage 1)
SDADC_CHANNEL_STAGE_1_GAIN_4	Gain of 4.
SDADC_CHANNEL_STAGE_1_GAIN_8	Gain of 8.

9.50.16.5 sdadc_channel_stage_2_gain_t

`sdadc_channel_stage_2_gain_t`

Detailed description

Per channel stage 2 gain options.

Enumerated values

Name	Description
SDADC_CHANNEL_STAGE_2_GAIN_1	Gain of 1.
SDADC_CHANNEL_STAGE_2_GAIN_2	Gain of 2.
SDADC_CHANNEL_STAGE_2_GAIN_4	Gain of 4.
SDADC_CHANNEL_STAGE_2_GAIN_8	Gain of 8.

9.50.16.6 sdadc_channel_oversampling_t

`sdadc_channel_oversampling_t`

Detailed description

Per channel oversampling ratio.

Enumerated values

Name	Description
SDADC_CHANNEL_OVERSAMPLING_64	Oversampling ratio of 64.
SDADC_CHANNEL_OVERSAMPLING_128	Oversampling ratio of 128.
SDADC_CHANNEL_OVERSAMPLING_256	Oversampling ratio of 256.
SDADC_CHANNEL_OVERSAMPLING_512	Oversampling ratio of 512.
SDADC_CHANNEL_OVERSAMPLING_1024	Oversampling ratio of 1024.

Name	Description
SDADC_CHANNEL_OVERSAMPLING_2048	Oversampling ratio of 2048.

9.50.16.7 sdadc_channel_polarity_t

`sdadc_channel_polarity_t`

Detailed description

Per channel polarity, valid for single-ended input only.

Enumerated values

Name	Description
SDADC_CHANNEL_POLARITY_POSITIVE	Positive-side single-ended input.
SDADC_CHANNEL_POLARITY_NEGATIVE	Negative-side single-ended input.

9.50.16.8 sdadc_channel_average_t

`sdadc_channel_average_t`

Detailed description

Per channel number of conversions to average before conversion end callback.

Enumerated values

Name	Description
SDADC_CHANNEL_AVERAGE_NONE	Do not average (callback for each conversion)
SDADC_CHANNEL_AVERAGE_8	Average 8 samples for each conversion end callback.
SDADC_CHANNEL_AVERAGE_16	Average 16 samples for each conversion end callback.
SDADC_CHANNEL_AVERAGE_32	Average 32 samples for each conversion end callback.
SDADC_CHANNEL_AVERAGE_64	Average 64 samples for each conversion end callback.

9.50.16.9 sdadc_channel_inversion_t

`sdadc_channel_inversion_t`

Detailed description

Per channel polarity, valid for negative-side single-ended input only.

Enumerated values

Name	Description
SDADC_CHANNEL_INVERSION_OFF	Do not invert conversion result.
SDADC_CHANNEL_INVERSION_ON	Invert conversion result.

9.50.16.10 sdadc_channel_count_formula_t

`sdadc_channel_count_formula_t`

Detailed description

Select a formula to specify the number of conversions. The following symbols are used in the formulas:

- N: Number of conversions
- n: coefficient_n, do not set to 0 if m is 0
- m: coefficient_m, do not set to 0 if n is 0

Either m or n must be non-zero.

Enumerated values

Name	Description
SDADC_CHANNEL_COUNT_FORMULA_EXPONENTIAL	N = 32 * (2 ^ n - 1) + m * 2 ^ n.
SDADC_CHANNEL_COUNT_FORMULA_LINEAR	N = (32 * n) + m.

9.50.16.11 sdadc_calibration_t

`sdadc_calibration_t`

Detailed description

Calibration mode.

Enumerated values

Name	Description
SDADC_CALIBRATION_INTERNAL_GAIN_OFFSET	
SDADC_CALIBRATION_EXTERNAL_OFFSET	
SDADC_CALIBRATION_EXTERNAL_GAIN	

9.50.17 Extensions

9.50.17.1 sdadc_calibrate_args_t

sdadc_calibrate_args_t

Detailed description

Structure to pass to the calibrate p_extend argument.

Variables

- adc_register_t channel

  Which channel to calibrate.

- sdadc_calibration_t mode

  Calibration mode.

9.50.17.2 sdadc_channel_cfg_t

sdadc_channel_cfg_t

Detailed description

SDADC per channel configuration.

Variables

- sdadc_channel_stage_2_gain_t stage_2_gain

  Gain of PGA stage 2, must be 1 for single-ended input.

- sdadc_channel_stage_1_gain_t stage_1_gain

  Gain of PGA stage 1, must be 1 for single-ended input.

- sdadc_channel_oversampling_t oversampling

  Oversampling ratio, must be 256 in single-ended input.

- uint32_t __pad0__

- sdadc_channel_polarity_t polarity

  Polarity, valid for single-ended mode only.

- sdadc_channel_input_t input

  Single-ended or differential input.

- uint32_t coefficient_m

  See sdadc_channel_count_formula_t.

- uint32_t coefficient_n

  See sdadc_channel_count_formula_t.

- sdadc_channel_average_t average

  Number of samples to average for each conversion result.

- sdadc_channel_inversion_t invert

  Whether to invert negative single-ended input.
- uint32_t __pad1__
- sdadc_channel_count_formula_t count_formula

  Linear or exponential formula used for number of conversions.

9.50.17.3 adc_on_sdadc_cfg_t

adc_on_sdadc_cfg_t

Detailed description

SDADC configuration extension. This extension is required and must be provided in p_extend.

Variables

- uint8_t calibration_end_ipl

  Calibration end interrupt priority.
- uint8_t conversion_end_ipl

  Conversion end interrupt priority.
- bool skip_internal_calibration

  Whether to skip internal calibration of of the PGA during open.
- sdadc_vref_src_t vref_src

  Source of Vref (internal or external)
- sdadc_vref_voltage_t vref_voltage

  Voltage of Vref, required for both internal and external Vref. If Vref is from an external source, the voltage must match the specified voltage within 3%.
- sdadc_channel_cfg_t const * p_channel_cfgs[SDADC_MAX_NUM_CHANNELS]

  Configuration for each channel, set to NULL if unused.

9.50.17.4 sdadc_instance_ctrl_t

sdadc_instance_ctrl_t

Detailed description

ADC instance control block. DO NOT INITIALIZE. Initialized in open.

Variables

- adc_mode_t mode
- adc_resolution_t resolution
- adc_alignment_t alignment
- void const * p_context

- R_SDADC0_Type * p_reg
- void(* p_callback)(*p_args)
- adc_trigger_t trigger
- uint32_t trigger_enabled
- uint32_t opened
- uint32_t scan_mask
- uint32_t scan_cfg_mask
- uint16_t unit
- uint8_t calib_status
- IRQn_Type scan_end_irq
- IRQn_Type calib_end_irq
- IRQn_Type conv_end_irq
- uint16_t results_16[SDADC_MAX_NUM_CHANNELS]
- uint32_t results_32[SDADC_MAX_NUM_CHANNELS]
- union{} results

  See source code for definition of this member.

9.51 SDMMC

Driver for the SD/MMC Host Interface (SDHI).

SD/MMC driver to access SD, eMMC, and SDIO devices.

9.51.1 Functions

- R_SDMMC_Open
- R_SDMMC_Close
- R_SDMMC_Read
- R_SDMMC_Write
- R_SDMMC_Control
- R_SDMMC_ReadIo
- R_SDMMC_WriteIo
- R_SDMMC_ReadIoExt
- R_SDMMC_WriteIoExt
- R_SDMMC_IoIntEnable
- R_SDMMC_VersionGet

- R_SDMMC_InfoGet
- R_SDMMC_Erase

9.51.2 Defines

- #define SDMMC_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define SDMMC_CODE_VERSION_MINOR

  Initial value: `(7U)`

9.51.3 R_SDMMC_Open

```
ssp_err_t R_SDMMC_Open ( sdmmc_ctrl_t *const p_api_ctrl , sdmmc_cfg_t const
*const p_cfg )
```

9.51.3.1 Detailed description

Initializes the SDHI hardware and completes identification and configuration for the SD or eMMC device. This procedure requires several sequential commands. This API blocks until all identification and configuration commands are complete.

For SDIO, SDIO interrupts are enabled after card identification is complete. SDIO interrupts can be disabled using sdmmc_api_t::ioIntEnable().

Implements open.

Table 1886:

Name	Description
SSP_SUCCESS	Port is available and is now open for read/write/control access.
SSP_ERR_ASSERTION	Null Pointer or block size is not in the valid range of 1-512.
SSP_ERR_INVALID_ARGUMENT	Block size must be 512 bytes for SD cards and eMMC devices. It is configurable for SDIO only.
SSP_ERR_ALREADY_OPEN	Driver has already been opened with this instance of the control structure.
SSP_ERR_HW_LOCKED	The channel specified has already been opened.
SSP_ERR_CARD_INIT_FAILED	Hardware related failure occurred, with the MCU or with the card itself.
SSP_ERR_IRQ_BSP_DISABLED	Access interrupt is not enabled.

Table 1886: (Continued)

Name	Description
SSP_ERR_CARD_NOT_INSERTED	Card detection is enabled and no card is plugged in.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet
- eventInfoGet
- systemClockFreqGet

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

9.51.3.2 Function steps

- Verify the requested hardware channel exists on the MCU.
- Configure interrupts.
- Perform the identification procedure for SD card or eMMC device.
- Configure bus clock, block size, and bus width.
- Check to see if the card is write protected (SD cards only).

9.51.4 R_SDMMC_Close

```
ssp_err_t R_SDMMC_Close ( sdmmc_ctrl_t  *const p_api_ctrl )
```

9.51.4.1 Detailed description

Closes an open SD/MMC device. Implements close.

Table 1887:

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_NOT_OPEN	Driver has not been initialized.
SSP_ERR_TRANSFER_BUSY	Driver is busy with a previous operation.

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

9.51.4.2 Function steps

- Disable SDHI interrupts.
- Close the transfer driver.
- Release hardware lock.

9.51.5 R_SDMMC_Read

```
ssp_err_t R_SDMMC_Read ( sdmmc_ctrl_t  *const p_api_ctrl ,   uint8_t
*const p_dest ,   uint32_t const start_sector ,   uint32_t const sector_count )
```

9.51.5.1 Detailed description

Reads data from an SD or eMMC device. Up to 0x10000 sectors can be read at a time. Implements read.

This function blocks until the command is sent and the response is received. A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the read data is available.

Table 1888:

Name	Description
SSP_SUCCESS	Data read successfully.
SSP_ERR_ASSERTION	NULL pointer.
SSP_ERR_NOT_OPEN	Driver has not been initialized.
SSP_ERR_CARD_NOT_READY	Card was unplugged.
SSP_ERR_TRANSFER_BUSY	Driver is busy with a previous operation.
SSP_ERR_READ_FAILED	Read operation failed.

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

9.51.6 R_SDMMC_Write

```
ssp_err_t R_SDMMC_Write ( sdmmc_ctrl_t  *const p_api_ctrl ,   uint8_t const
*const p_source ,   uint32_t const start_sector ,   uint32_t const sector_count )
```

9.51.6.1 Detailed description

Writes data to an SD or eMMC device. Up to 0x10000 sectors can be written at a time. Implements write.

This function blocks until the command is sent and the response is received. A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the all data has been written.

Table 1889:

Name	Description
SSP_SUCCESS	Card write finished successfully.
SSP_ERR_ASSERTION	Handle or Source address is NULL.
SSP_ERR_NOT_OPEN	Driver has not been initialized.
SSP_ERR_CARD_NOT_READY	Card was unplugged.
SSP_ERR_TRANSFER_BUSY	Driver is busy with a previous operation.
SSP_ERR_WRITE_PROTECTED	SD card is Write Protected.
SSP_ERR_WRITE_FAILED	Write operation failed.

NOTE: This function is reentrant for different channels.

9.51.7 R_SDMMC_Control

```
ssp_err_t R_SDMMC_Control ( sdmmc_ctrl_t  *const p_api_ctrl ,  ssp_command_t
const command ,   void * p_data )
```

9.51.7.1 Detailed description

Sends control commands to and receives the status of the SD/MMC device. Implements control.

Table 1890:

Name	Description
SSP_SUCCESS	Command executed successfully.
SSP_ERR_ASSERTION	Null Pointer.
SSP_ERR_INVALID_ARGUMENT	Command is invalid.
SSP_ERR_INVALID_SIZE	Block size not in valid range of 1-512 for SDIO or 512 only for SD cards and eMMC.

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

9.51.8 R_SDMMC_ReadIo

```
ssp_err_t R_SDMMC_ReadIo ( sdmmc_ctrl_t  *const p_api_ctrl ,   uint8_t
*const p_data ,   uint32_t const function ,   uint32_t const address )
```

9.51.8.1 Detailed description

The Read function reads a one byte register from an SDIO card. Implements readIo.

This function blocks until the command is sent and the response is received. p_data contains the register value read when this function returns.

Table 1891:

Name	Description
SSP_SUCCESS	Data read successfully.
SSP_ERR_ASSERTION	NULL pointer.
SSP_ERR_NOT_OPEN	Driver has not been initialized.
SSP_ERR_CARD_NOT_READY	Card was unplugged.
SSP_ERR_TRANSFER_BUSY	Driver is busy with a previous operation.
SSP_ERR_READ_FAILED	Read operation failed.

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

9.51.9 R_SDMMC_WriteIo

```
ssp_err_t R_SDMMC_WriteIo ( sdmmc_ctrl_t  *const p_api_ctrl ,   uint8_t
*const p_data ,   uint32_t const function ,   uint32_t const address ,
  sdmmc_io_write_mode_t const read_after_write )
```

9.51.9.1 Detailed description

Writes a one byte register to an SDIO card. Implements writeIo.

This function blocks until the command is sent and the response is received. The register has been written when this function returns. If read_after_write is true, p_data contains the register value read when this function returns.

Table 1892:

Name	Description
SSP_SUCCESS	Card write finished successfully.
SSP_ERR_ASSERTION	Handle or Source address is NULL.
SSP_ERR_NOT_OPEN	Driver has not been initialized.
SSP_ERR_CARD_NOT_READY	Card was unplugged.
SSP_ERR_TRANSFER_BUSY	Driver is busy with a previous operation.
SSP_ERR_WRITE_FAILED	Write operation failed.

NOTE: This function is reentrant for different channels.

9.51.10 R_SDMMC_ReadIoExt

```
ssp_err_t R_SDMMC_ReadIoExt ( sdmmc_ctrl_t  *const p_api_ctrl ,   uint8_t
*const p_dest ,   uint32_t const function ,   uint32_t const address ,   uint32_t
*const count ,   sdmmc_io_transfer_mode_t transfer_mode ,
  sdmmc_io_address_mode_t address_mode )
```

9.51.10.1 Detailed description

Reads data from an SDIO card function. Implements readIoExt.

This function blocks until the command is sent and the response is received. A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the read data is available.

Table 1893:

Name	Description
SSP_SUCCESS	Data read successfully.
SSP_ERR_ASSERTION	NULL pointer, or count is not in the valid range of 1-512 for byte mode or 1-511 for block mode.
SSP_ERR_NOT_OPEN	Driver has not been initialized.
SSP_ERR_CARD_NOT_READY	Card was unplugged.
SSP_ERR_TRANSFER_BUSY	Driver is busy with a previous operation.
SSP_ERR_READ_FAILED	Read operation failed.

NOTE: This function is reentrant for different channels. It is not reentrant for the same channel.

9.51.11 R_SDMMC_WriteIoExt

```
ssp_err_t R_SDMMC_WriteIoExt ( sdmmc_ctrl_t  *const p_api_ctrl ,   uint8_t const
*const p_source ,   uint32_t const function ,   uint32_t const address ,
  uint32_t const count ,   sdmmc_io_transfer_mode_t transfer_mode ,
  sdmmc_io_address_mode_t address_mode )
```

9.51.11.1 Detailed description

Writes data to an SDIO card function. Implements writeIoExt.

This function blocks until the command is sent and the response is received. A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the all data has been written.

Table 1894:

Name	Description
SSP_SUCCESS	Card write finished successfully.

Table 1894: (Continued)

Name	Description
SSP_ERR_ASSERTION	NULL pointer, or count is not in the valid range of 1-512 for byte mode or 1-511 for block mode.
SSP_ERR_NOT_OPEN	Driver has not been initialized.
SSP_ERR_CARD_NOT_READY	Card was unplugged.
SSP_ERR_TRANSFER_BUSY	Driver is busy with a previous operation.
SSP_ERR_WRITE_FAILED	Write operation failed.

NOTE: This function is reentrant for different channels.

9.51.12 R_SDMMC_IoIntEnable

```
ssp_err_t R_SDMMC_IoIntEnable ( sdmmc_ctrl_t  *const p_api_ctrl ,
  bool enable )
```

9.51.12.1 Detailed description

Enables or disables the SDIO Interrupt. Implements sdmmc_api_t::ioIntEnable().

Table 1895:

Name	Description
SSP_SUCCESS	Card enabled or disabled SDIO interrupts successfully.
SSP_ERR_ASSERTION	NULL pointer.
SSP_ERR_TRANSFER_BUSY	Driver is busy with a previous operation.

NOTE: This function is reentrant for different channels.

9.51.12.2 Function steps

- Enable or disable interrupt.

9.51.13 R_SDMMC_VersionGet

```
ssp_err_t R_SDMMC_VersionGet ( ssp_version_t  *const p_version )
```

9.51.13.1 Detailed description

Returns the version of the firmware and API. Implements versionGet.

Table 1896:

Name	Description
SSP_SUCCESS	Function executed successfully.
SSP_ERR_ASSERTION	Null Pointer.

NOTE: This function is reentrant.

9.51.14 R_SDMMC_InfoGet

```
ssp_err_t R_SDMMC_InfoGet ( sdmmc_ctrl_t  *const p_api_ctrl ,  sdmmc_info_t
*const p_info )
```

9.51.14.1 Detailed description

Provides information about the connected device and driver status. Implements infoGet.

NOTE: This function is reentrant.

9.51.14.2 Function steps

- Copy information stored during open.
- Check if the card is busy.

9.51.15 R_SDMMC_Erase

```
ssp_err_t R_SDMMC_Erase ( sdmmc_ctrl_t  *const p_api_ctrl ,   uint32_t
const start_sector ,   uint32_t const sector_count )
```

9.51.15.1 Detailed description

Erases sectors of an SD card or eMMC device. Implements erase.

This function blocks until erase is complete.

Table 1897:

Name	Description
SSP_SUCCESS	Erase operation requested.
SSP_ERR_ASSERTION	A required pointer is NULL.
SSP_ERR_NOT_OPEN	Driver has not been initialized.
SSP_ERR_CARD_NOT_READY	Card was unplugged.
SSP_ERR_TRANSFER_BUSY	Driver is busy with a previous operation.
SSP_ERR_WRITE_PROTECTED	SD card is Write Protected.
SSP_ERR_ERASE_FAILED	Erase operation unsuccessful.

NOTE: This function is reentrant for different channels.

9.51.15.2 Function steps

- Send command to set start erase address (CMD35 for eMMC, CMD32 for SD).
- Send command to set end erase address (CMD36 for eMMC, CMD33 for SD).
- Send erase command (CMD38).

9.51.16 API Data

9.51.16.1 sdmmc_card_detect_t

```
sdmmc_card_detect_t
```

Detailed description

Card detection configuration options.

Enumerated values

Name	Description
SDMMC_CARD_DETECT_NONE	Card detection unused.
SDMMC_CARD_DETECT_CD	Card detection using the SDnCD pin.

9.51.16.2 sdmmc_transfer_dir_t

`sdmmc_transfer_dir_t`

Enumerated values

Name	Description
SDMMC_TRANSFER_DIR_NONE	
SDMMC_TRANSFER_DIR_READ	
SDMMC_TRANSFER_DIR_WRITE	

9.51.16.3 sdmmc_io_command_t

`sdmmc_io_command_t`

Enumerated values

Name	Description
SDMMC_IO_COMMAND_DIRECT_IO	
SDMMC_IO_COMMAND_DIRECT_IO_EXT	

9.51.17 Extensions

9.51.17.1 sdhi_event_t

Detailed description
Variables

word

bit

9.51.17.2 sdhi_sdio_event_t

Detailed description
Variables

word

bit

9.51.17.3 sdhi_int_status_t

sdhi_int_status_t

Detailed description
Variables

- uint32_t info2_mask

9.51.17.4 sdmmc_io_mode_t

sdmmc_io_mode_t

Detailed description
Variables

- sdmmc_io_command_t command
- sdmmc_io_transfer_mode_t transfer_mode
- sdmmc_io_address_mode_t address_mode
- sdmmc_io_write_mode_t write_mode

9.51.17.5 sdmmc_instance_ctrl_t

sdmmc_instance_ctrl_t

Detailed description

SDMMC instance control block. This is private to the SSP and should not be used or modified by the application.

Variables

- uint32_t open
- sdmmc_hw_t hw
- transfer_instance_t const * p_lower_lvl_transfer
- sdmmc_info_t status
- bool transfer_in_progress
- void(* p_callback)(*p_args)
- void const * p_context
- R_SDHI0_Type * p_reg
- sdhi_event_t sdhi_event

- IRQn_Type transfer_irq
- sdmmc_transfer_dir_t transfer_dir
- uint8_t * p_transfer_data
- uint32_t transfer_blocks_total
- uint32_t transfer_block_current
- uint32_t transfer_block_size
- uint32_t aligned_buff[SDMMC_MAX_BLOCK_SIZE/sizeof(uint32_t)]

9.51.17.6 sdmmc_extended_cfg_t

sdmmc_extended_cfg_t

Detailed description

Extended SDMMC configuration, to be pointed to p_extend.

Variables

- uint32_t block_size

  Block size in bytes. Block size must be 512 bytes for SD cards and eMMC devices. Block size can be 1-512 bytes for SDIO.

- sdmmc_card_detect_t card_detect

  Whether or not card detection is used.

9.52 SLCDC

Driver for the Segment LCD Controller (SLCDC).

9.52.1 Summary

Extends SLCDC Interface.

9.52.2 Functions

- R_SLCDC_Open
- R_SLCDC_Write
- R_SLCDC_Modify
- R_SLCDC_Start
- R_SLCDC_Stop
- R_SLCDC_ContrastIncrease

- R_SLCDC_ContrastDecrease
- R_SLCDC_SetDisplayArea
- R_SLCDC_Close
- R_SLCDC_VersionGet

9.52.3 R_SLCDC_Open

```
ssp_err_t R_SLCDC_Open ( slcdc_ctrl_t  *const p_api_ctrl ,  slcdc_cfg_t  const
*const p_cfg )
```

9.52.3.1 Detailed description

Table 1898:

Name	Description
SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
SSP_ERR_HW_LOCKED	SLCDC resource is locked.

See Common Error Codes for other possible return codes. This function calls

- productFeatureGet

9.52.4 R_SLCDC_Write

```
ssp_err_t R_SLCDC_Write ( slcdc_ctrl_t  *const p_api_ctrl ,  slcdc_size_t
const start_segment ,  slcdc_size_t  const *const p_data ,  slcdc_size_t
const segment_count )
```

9.52.4.1 Detailed description

Table 1899:

Name	Description
SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.

Table 1899: (Continued)

Name	Description
SSP_ERR_INVALID_ARGUMENT	Invalid parameter in the argument.
SSP_ERR_NOT_OPEN	Device is not opened or initialized

9.52.5 R_SLCDC_Modify

```
ssp_err_t R_SLCDC_Modify ( slcdc_ctrl_t  *const p_api_ctrl ,  slcdc_size_t
const segment_number ,  slcdc_size_t  const data ,  slcdc_size_t
const data_mask )
```

9.52.5.1 Detailed description

Table 1900:

Name	Description
SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter in the argument.
SSP_ERR_NOT_OPEN	Device is not opened or initialized

9.52.6 R_SLCDC_Start

```
ssp_err_t R_SLCDC_Start ( slcdc_ctrl_t  *const p_api_ctrl )
```

9.52.6.1 Detailed description

Table 1901:

Name	Description
SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.

Table 1901: (Continued)

Name	Description
SSP_ERR_INVALID_ARGUMENT	Invalid parameter in the argument.
SSP_ERR_NOT_OPEN	Device is not opened or initialized

9.52.7 R_SLCDC_Stop

```
ssp_err_t R_SLCDC_Stop ( slcdc_ctrl_t  *const p_api_ctrl )
```

9.52.7.1 Detailed description

Table 1902:

Name	Description
SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter in the argument.
SSP_ERR_NOT_OPEN	Device is not opened or initialized

9.52.8 R_SLCDC_ContrastIncrease

```
ssp_err_t R_SLCDC_ContrastIncrease ( slcdc_ctrl_t  *const p_api_ctrl )
```

9.52.8.1 Detailed description

Table 1903:

Name	Description
SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter in the argument.

Table 1903: (Continued)

Name	Description
SSP_ERR_NOT_OPEN	Device is not opened or initialized
SSP_ERR_UNSUPPORTED	Unsupported operation

9.52.9 R_SLCDC_ContrastDecrease

```
ssp_err_t R_SLCDC_ContrastDecrease ( slcdc_ctrl_t  *const p_api_ctrl )
```

9.52.9.1 Detailed description

Table 1904:

Name	Description
SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter in the argument.
SSP_ERR_NOT_OPEN	Device is not opened or initialized
SSP_ERR_UNSUPPORTED	Unsupported operation

9.52.10 R_SLCDC_SetDisplayArea

```
ssp_err_t R_SLCDC_SetDisplayArea ( slcdc_ctrl_t  *const p_api_ctrl ,
 slcdc_display_area_t  const display_area )
```

9.52.10.1 Detailed description

Table 1905:

Name	Description
SSP_SUCCESS	Device was opened successfully.

Table 1905: (Continued)

Name	Description
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
SSP_ERR_UNSUPPORTED	Unsupported operation
SSP_ERR_NOT_OPEN	Device is not opened or initialized
SSP_ERR_NOT_ENABLED	RTC not enabled for blink operation

See Common Error Codes for other possible return codes. This function calls

- productFeatureGet

9.52.11 R_SLCDC_Close

```
ssp_err_t R_SLCDC_Close ( slcdc_ctrl_t  *const p_api_ctrl )
```

9.52.11.1 Detailed description

Table 1906:

Name	Description
SSP_SUCCESS	Device was opened successfully.
SSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
SSP_ERR_INVALID_ARGUMENT	Invalid parameter in the argument.
SSP_ERR_NOT_OPEN	Device is not opened or initialized

9.52.12 R_SLCDC_VersionGet

```
ssp_err_t R_SLCDC_VersionGet ( ssp_version_t  *const p_version )
```

9.52.12.1 Brief description

Retrieve the API version number.

9.52.12.2 Detailed description

Table 1907:

Name	Description
SSP_SUCCESS	Successful return.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

9.52.13 Extensions

9.52.13.1 slcdc_instance_ctrl_t

slcdc_instance_ctrl_t

Detailed description

SLCDC control block. DO NOT INITIALIZE. Initialization occurs when open is called

Variables

- slcdc_display_state_t state
  Status of SLCD module.
- slcdc_cfg_t info
  SLCDC config info.
- void const * p_context
  Pointer to the higher level device context.
- R_LCD_Type * p_reg
  Pointer to register base address.

9.53 SSI

Driver for the Serial Sound Interface (SSI).

9.53.1 Summary

Extends I2S Interface.

9.53.2 Functions

- R_SSI_Open
- R_SSI_Stop
- R_SSI_Close
- R_SSI_Write
- R_SSI_Read
- R_SSI_WriteRead
- R_SSI_Mute
- R_SSI_InfoGet
- R_SSI_VersionGet

9.53.3 Defines

- #define SSI_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define SSI_CODE_VERSION_MINOR

  Initial value: `(5U)`

9.53.4 R_SSI_Open

```
ssp_err_t R_SSI_Open ( i2s_ctrl_t  *const p_api_ctrl ,  i2s_cfg_t  const
*const p_cfg )
```

9.53.4.1 Brief description

Opens the SSI. Implements open.

9.53.4.2 Detailed description

This function calculates the clock divisor based on the input audio clock frequency and the requested sampling frequency. It sets this clock divisor and the configurations specified in i2s_cfg_t. It also opens the timer and transfer interfaces if they are provided.

Table 1908:

Name	Description
SSP_SUCCESS	Ready for I2S communication.
SSP_ERR_ASSERTION	The pointer to p_ctrl or p_cfg is null.

Table 1908: (Continued)

Name	Description
SSP_ERR_IN_USE	The requested channel has already been opened or hardware has been locked.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet
- eventInfoGet
- open
- open

9.53.4.3 Function steps

- Configure dependent timer and transfer drivers.
- Configure interrupts.
- Configure the audio clock.
- Mark driver as open by initializing it to "SSI" in its ASCII equivalent.

9.53.5 R_SSI_Stop

```
ssp_err_t R_SSI_Stop ( i2s_ctrl_t  *const p_api_ctrl ,  i2s_dir_t  const dir )
```

9.53.5.1 Brief description

Stops SSI. Implements stop.

9.53.5.2 Detailed description

This function disables the transfer if the transfer interface is used, or sends a stop signal to stop writing data in the ISR if interrupt driven mode is used.

Table 1909:

Name	Description
SSP_SUCCESS	I2S communication stop request issued.
SSP_ERR_ASSERTION	The pointer to p_ctrl null.
SSP_ERR_NOT_OPEN	The channel is not opened.

See Common Error Codes or lower level drivers for other possible return codes.

9.53.5.3 Function steps

- Stop will occur when the peripheral is idle.

9.53.6 R_SSI_Close

```
ssp_err_t R_SSI_Close ( i2s_ctrl_t  *const p_api_ctrl )
```

9.53.6.1 Brief description

Closes SSI. Implements close.

9.53.6.2 Detailed description

This function powers down the SSI and closes the lower level timer and transfer drivers if they are used.

Table 1910:

Name	Description
SSP_SUCCESS	Device closed successfully.
SSP_ERR_ASSERTION	The pointer to p_ctrl null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.53.6.3 Function steps

- Stop feeding clock to SSI peripheral to deactivate it.
- If a timer instance is provided, close the timer instance.
- If a transfer instance is provided for write, close the transfer instance.
- If a transfer instance is provided for read, close the transfer instance.
- Release HW lock.

9.53.7 R_SSI_Write

```
ssp_err_t R_SSI_Write ( i2s_ctrl_t  *const p_api_ctrl ,   uint8_t const
*const p_src ,   uint16_t const bytes )
```

9.53.7.1 Brief description

Writes data buffer to SSI. Implements write.

9.53.7.2 Detailed description

This function resets the transfer if the transfer interface is used, or writes the length of data that fits in the FIFO then stores the remaining write buffer in the control block to be written in the ISR.

Write() cannot be called if another write(), read() or writeRead() operation is in progress. Write can be called when the SSI is idle, or after the I2S_EVENT_TX_EMPTY event.

Table 1911:

Name	Description
SSP_SUCCESS	Write initiated successfully.
SSP_ERR_ASSERTION	The pointer to p_ctrl or p_src was null, or bytes requested was 0.
SSP_ERR_IN_USE	Another transfer is in progress, data was not written.
SSP_ERR_NOT_OPEN	The channel is not opened.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- reset
- start

9.53.7.3 Function steps

- Make sure transmission is enabled.
- If a transfer instance is provided for write, reset the transfer.
- Otherwise, write to the FIFO directly.

9.53.8 R_SSI_Read

```
ssp_err_t R_SSI_Read ( i2s_ctrl_t  *const p_api_ctrl ,   uint8_t *const p_dest ,
  uint16_t const bytes )
```

9.53.8.1 Brief description

Reads data into provided buffer. Implements read.

9.53.8.2 Detailed description

This function resets the transfer if the transfer interface is used, or reads the length of data available in the FIFO then stores the remaining read buffer in the control block to be filled in the ISR.

Read() cannot be called if another write(), read() or writeRead() operation is in progress. Read can be called when the SSI is idle, or after the I2S_EVENT_RX_FULL event.

Table 1912:

Name	Description
SSP_SUCCESS	Read initiated successfully.
SSP_ERR_ASSERTION	The pointer to p_ctrl or p_dest was null, or bytes requested was 0.
SSP_ERR_NOT_OPEN	The channel is not opened.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- reset
- start

9.53.8.3 Function steps

- Make sure reception is enabled.
- If a transfer instance is provided for read, reset the transfer.
- Otherwise, read from the FIFO directly.

9.53.9 R_SSI_WriteRead

```
ssp_err_t R_SSI_WriteRead ( i2s_ctrl_t  *const p_api_ctrl ,   uint8_t const
*const p_src ,   uint8_t *const p_dest ,   uint16_t const bytes )
```

9.53.9.1 Brief description

Writes from source buffer and reads data into destination buffer. Implements writeRead.

9.53.9.2 Detailed description

This function calls R_SSI_Write and R_SSI_Read.

writeRead() cannot be called if another write(), read() or writeRead() operation is in progress. writeRead() can be called when the SSI is idle, or after the I2S_EVENT_RX_FULL event.

Table 1913:

Name	Description
SSP_SUCCESS	Write and read initiated successfully.
SSP_ERR_ASSERTION	The pointer to p_ctrl, p_src, or p_dest was null, or bytes requested was 0.
SSP_ERR_NOT_OPEN	The channel is not opened.

See Common Error Codes or lower level drivers for other possible return codes.

9.53.9.3 Function steps

- Call read, then write.

9.53.10 R_SSI_Mute

```
ssp_err_t R_SSI_Mute ( i2s_ctrl_t  *const p_api_ctrl ,  i2s_mute_t
const mute_enable )
```

9.53.10.1 Brief description

Mutes SSI. Implements mute.

9.53.10.2 Detailed description

Data is still written while mute is enabled, but the transmit line outputs zeros.

Table 1914:

Name	Description
SSP_SUCCESS	Transmission is muted.
SSP_ERR_ASSERTION	The pointer to p_ctrl was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.53.11 R_SSI_InfoGet

```
ssp_err_t R_SSI_InfoGet ( i2s_ctrl_t  *const p_api_ctrl ,  i2s_info_t
*const p_info )
```

9.53.11.1 Brief description

Get I2S information and store it in provided pointer p_info. Implements infoGet.

9.53.11.2 Detailed description

Table 1915:

Name	Description
SSP_SUCCESS	Information stored successfully.
SSP_ERR_ASSERTION	The p_ctrl or p_info parameter was null.
SSP_ERR_NOT_OPEN	The channel is not opened.

9.53.12 R_SSI_VersionGet

```
ssp_err_t R_SSI_VersionGet ( ssp_version_t  *const p_version )
```

9.53.12.1 Brief description

Sets driver version based on compile time macros.

9.53.12.2 Detailed description

Table 1916:

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

9.53.13 API Data

9.53.13.1 ssi_audio_clock_t

```
ssi_audio_clock_t
```

Detailed description

Audio clock source

Enumerated values

Name	Description
SSI_AUDIO_CLOCK_EXTERNAL	Audio clock source is the AUDIO_CLK input pin.
SSI_AUDIO_CLOCK_GTIOC1A	Audio clock source is internal connection to GPT channel 1 output.

9.53.14 Extensions

9.53.14.1 ssi_instance_ctrl_t

ssi_instance_ctrl_t

Detailed description

Channel instance control block. DO NOT INITIALIZE. Initialization occurs when open is called.

Variables

- void(* p_callback)(*p_args)

  Callback provided when an I2S ISR occurs. NULL indicates no CPU interrupt.
- void const * p_context

  Placeholder for user data. Passed to the user callback in i2s_callback_args_t.
- R_SSI0_Type * p_reg

  Pointer to SSI register base address.
- timer_instance_t const * p_timer

  Timer used to generate audio clock.
- transfer_instance_t const * p_transfer_tx

  Transfer used for hardware acceleration during write.
- transfer_instance_t const * p_transfer_rx

  Transfer used for hardware acceleration during read.
- uint32_t const * p_tx_src

  Source buffer pointer used to fill hardware FIFO from transmit ISR.
- uint32_t tx_src_bytes

  Size of source buffer used to fill hardware FIFO from transmit ISR.
- uint32_t * p_rx_dest

  Destination buffer pointer used to fill from hardware FIFO in receive ISR.

- uint32_t rx_dest_bytes

  Size of destination buffer used to fill from hardware FIFO in receive ISR.
- uint32_t sampling_freq_hz

  Sampling frequency in Hertz.
- uint8_t channel

  Channel number.
- uint8_t fifo_access_bytes

  Byte access to FIFO.
- uint8_t stop_requested_tx

  Stops I2S after transmit is complete.
- uint8_t stop_requested_rx

  Stops I2S after receive is complete.
- uint8_t tx_in_progress

  True if a transmit transfer is in progress.
- uint8_t tx_in_use

  True if a transmission is in progress.
- uint8_t rx_in_use

  True if a reception is in progress.
- uint8_t zeros_written

  True if zeros have been transmitted.
- IRQn_Type txi_irq

  Transmit IRQ number.
- IRQn_Type rxi_irq

  Receive IRQ number.
- IRQn_Type int_irq

  Idle/Error IRQ number.
- uint32_t open

  Whether or not this control block is initialized.

9.53.14.2 i2s_on_ssi_cfg_t

i2s_on_ssi_cfg_t

Detailed description

SSI configuration extension. This extension is optional.

Variables

- ssi_audio_clock_t audio_clock

  Audio clock source, default is SSI_AUDIO_CLOCK_EXTERNAL.

9.54 WDT

Driver for the Watchdog Timer (WDT).

9.54.1 Summary

This module supports the Watchdog Timer (WDT). It implements the WDT Interface. The WDT HAL APIs provide the ability to configure the operation of the WDT (when used in register start mode), refresh the watchdog, read the timer value and read and clear status flags.

9.54.2 Functions

- R_WDT_Open
- R_WDT_CfgGet
- R_WDT_TimeoutGet
- R_WDT_Refresh
- R_WDT_StatusGet
- R_WDT_StatusClear
- R_WDT_CounterGet
- R_WDT_VersionGet

9.54.3 Defines

- #define WDT_CODE_VERSION_MAJOR

  Initial value: `(1U)`
- #define WDT_CODE_VERSION_MINOR

  Initial value: `(4U)`

9.54.4 R_WDT_Open

```
ssp_err_t R_WDT_Open ( wdt_ctrl_t  *const p_api_ctrl ,  wdt_cfg_t  const
*const p_cfg )
```

9.54.4.1 Brief description

Configure the WDT in register start mode. In auto-start_mode the NMI callback can be registered. Implements open.

9.54.4.2 Detailed description

This function should only be called once as WDT configuration registers can only be written to once so subsequent calls will have no effect.

Table 1917:

Name	Description
SSP_SUCCESS	WDT successfully configured.
SSP_ERR_ASSERTION	Null Pointer(s).
SSP_ERR_INVALID_ARGUMENT	One or more configuration options is invalid.
SSP_ERR_INVALID_MODE	An attempt to open the WDT in register-start mode when the OFS0 register is configured for auto-start mode. Or to open the WDT in auto-start mode when the OSF0 is configured for register start mode.

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- productFeatureGet

NOTE: This function is reentrant. In auto-start mode the only valid configuration option is for registering the callback for the NMI ISR if NMI output has been selected.

9.54.4.3 Function steps

- g_wdt_version is accessed by the ASSERT macro only and so compiler toolchain can issue a warning that it is not accessed. The code below eliminates this warning and also ensures this data structure is not optimised away.
- Eliminate toolchain warning when NMI output is not being used.
- Check the expected start mode matches the OSF0 configuration.
- Lock the IWDT Hardware Resource
- Initialize global pointer to WDT for NMI callback use.
- Configuration only valid when WDT operating in register-start mode.
- Register-start mode.
- Register callback with BSP NMI ISR.

- Enable the WDT underflow/refresh error interrupt (will generate an NMI).
- Start the timer by performing a refresh.

9.54.5 R_WDT_CfgGet

```
ssp_err_t R_WDT_CfgGet ( wdt_ctrl_t  *const p_api_ctrl ,  wdt_cfg_t
*const p_cfg )
```

9.54.5.1 Brief description

Read the configuration of the WDT in both register-start and auto-start modes. Implements cfgGet.

9.54.5.2 Detailed description

Table 1918:

Name	Description
SSP_SUCCESS	WDT successfully configured.
SSP_ERR_ASSERTION	Null Pointer.

NOTE: This function is reentrant.

9.54.5.3 Function steps

- Register-start mode.
- Get timeout value from WDTCR register.

9.54.6 R_WDT_TimeoutGet

```
ssp_err_t R_WDT_TimeoutGet ( wdt_ctrl_t  *const p_api_ctrl ,
 wdt_timeout_values_t  *const p_timeout )
```

9.54.6.1 Brief description

Read timeout information for the watchdog timer. Implements timeoutGet.

9.54.6.2 Detailed description

Table 1919:

Name	Description
SSP_SUCCESS	WDT successfully refreshed.
SSP_ERR_ASSERTION	Null Pointer.
SSP_ERR_ABORTED	Invalid clock divider for this watchdog

NOTE: This function is reentrant. This function must not be called before calling R_WDT_Open.

9.54.7 R_WDT_Refresh

```
ssp_err_t R_WDT_Refresh ( wdt_ctrl_t  *const p_api_ctrl )
```

9.54.7.1 Brief description

Refresh the watchdog timer. Implements refresh.

9.54.7.2 Detailed description

In addition to refreshing the watchdog counter this function can be used, in register start mode to start the counter.

Table 1920:

Name	Description
SSP_SUCCESS	WDT successfully refreshed.

NOTE: This function is reentrant. This function only returns SSP_SUCCESS. If the refresh fails due to being performed outside of the permitted refresh period the device will either reset or trigger an NMI ISR to run. This function must not be called before calling R_WDT_Open.

9.54.8 R_WDT_StatusGet

```
ssp_err_t R_WDT_StatusGet ( wdt_ctrl_t  *const p_api_ctrl ,  wdt_status_t
*const p_status )
```

9.54.8.1 Brief description

Read the WDT status flags. Implements statusGet.

9.54.8.2 Detailed description

Indicates both status and error conditions.

Table 1921:

Name	Description
SSP_SUCCESS	WDT status successfully read.
SSP_ERR_ASSERTION	Null pointer as a parameter.

NOTE: This function is reentrant. When the WDT is configured to output a reset on underflow or refresh error reading the status and error flags serves no purpose as they will always indicate that no underflow has occurred and there is no refresh error. Reading the status and error flags is only valid when interrupt request output is enabled.

9.54.9 R_WDT_StatusClear

```
ssp_err_t R_WDT_StatusClear ( wdt_ctrl_t  *const p_api_ctrl ,
 wdt_status_t const  status )
```

9.54.9.1 Brief description

Clear the WDT status and error flags. Implements statusClear.

9.54.9.2 Detailed description

Table 1922:

Name	Description
SSP_SUCCESS	WDT flag(s) successfully cleared.

Table 1922: (Continued)

Name	Description
SSP_ERR_ASSERTION	Null pointer as a parameter.

NOTE: This function is reentrant.

9.54.9.3 Function steps

- Write zero to clear flags.

9.54.10 R_WDT_CounterGet

```
ssp_err_t R_WDT_CounterGet ( wdt_ctrl_t  *const p_api_ctrl ,   uint32_t
*const p_count )
```

9.54.10.1 Brief description

Read the current count value of the WDT. Implements counterGet.

9.54.10.2 Detailed description

Table 1923:

Name	Description
SSP_SUCCESS	WDT current count successfully read.
SSP_ERR_ASSERTION	Null pointer passed as a parameter.

NOTE: This function is reentrant.

9.54.11 R_WDT_VersionGet

```
ssp_err_t R_WDT_VersionGet ( ssp_version_t  *const p_data )
```

9.54.11.1 Brief description

Return WDT HAL driver version. Implements versionGet.

9.54.11.2 Detailed description

Table 1924:

Name	Description
SSP_SUCCESS	Version information successfully read.
SSP_ERR_ASSERTION	Null pointer passed as a parameter

NOTE: This function is reentrant.

9.54.12 Extensions

9.54.12.1 wdt_instance_ctrl_t

wdt_instance_ctrl_t

Detailed description

WDT control block. DO NOT INITIALIZE. Initialization occurs when open is called.

Variables

- bool wdt_open

  Indicates whether the open() API has been successfully called.
- void const * p_context

  Placeholder for user data. Passed to the user callback in wdt_callback_args_t.
- R_WDT_Type * p_reg

  Pointer to register base address.
- void(* p_callback)(*p_args)

  Callback provided when a WDT NMI ISR occurs.

9.55 SCE Module

Synergy Cryptographic Engine that provides primitive cryptographic functions.
Initializes the SCE module cryptographic operations.

9.55.1 Functions

- R_SCE_Open
- R_SCE_StatusGet
- R_SCE_Close
- R_SCE_VersionGet

9.55.2 Variables

- g_sce_crypto_api
- sceInitializationStatus
- s_sce_version
- crypto_bsp_lock

9.55.3 R_SCE_Open

```
R_SCE_Open (  crypto_ctrl_t *const p_ctrl ,   crypto_cfg_t const *const p_cfg )
```

9.55.3.1 Detailed description

SCE Initialization

Table 1925:

Name	Direction	Description
p_ctrl	inout	control structure for the SCE module
p_cfg	in	configuration structure for the SCE module

an uint32_t integer indicating

- PASS if the initialization is successful
- FAIL if the initialization failed
- RESOURCE_CONFLICT if the SCE hardware resource is busy

9.55.4 R_SCE_StatusGet

```
R_SCE_StatusGet (  uint32_t * p_status )
```

9.55.4.1 Brief description

This function indicates if the SCE has been initialized or not.

9.55.4.2 Detailed description

SCE Get status of SCE module initialization

Table 1926:

Name	Direction	Description
p_status	in	pointer to uint32_t. This memory location is updated with the SCE module initialization

Table 1927:

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

9.55.5 R_SCE_Close

```
R_SCE_Close (  crypto_ctrl_t *const p_ctrl )
```

9.55.5.1 Brief description

Close SCE driver. Does nothing. Always returns success.

9.55.5.2 Detailed description

Table 1928:

Name	Direction	Description
p_ctrl	in	control structure for SCE block

Table 1929:

Name	Description
SSP_SUCCESS	Successful close.

9.55.6 R_SCE_VersionGet

```
R_SCE_VersionGet ( ssp_version_t  *const p_version )
```

9.55.6.1 Brief description

Sets driver version based on compile time macros.

9.55.6.2 Detailed description

Table 1930:

Name	Direction	Description
p_version	out	version info for the SCE implementation

Table 1931:

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

9.55.7 g_sce_crypto_api

```
const g_sce_crypto_api
```

9.55.7.1 Initialized as

```
g_sce_crypto_api=
{
    .open           = R_SCE_Open,
    .close          = R_SCE_Close,
    .interfaceGet   = R_SCE_InterfaceGet,
    .statusGet      = R_SCE_StatusGet,
    .versionGet     = R_SCE_VersionGet
}
```

9.55.8 Modules

- SCE AES
- SCE HRK AES
- SCE ARC4
- SCE DSA
- SCE HASH
- SCE_ECC
- SCE_KEY_INSTALLATION
- SCE_RSA
- SCE_TDES
- SCE_TRNG

9.55.8.1 SCE AES

Primitive cryptographic functions.

AES key generation, encryption and decryption functions for plain-text / raw keys.

AES encryption and decryption functions

AES 256-bit CBC mode implementation for encryption and decryption functions

AES 256-bit CTR mode implementation for encryption and decryption functions

AES 256-bit ECB mode implementation for encryption and decryption functions

AES 256-bit XTS mode implementation for encryption and decryption functions

Functions

- R_SCE_AES_Open
- R_SCE_AES_VersionGet

- R_SCE_AES_Close
- R_SCE_AES_128CbcEncrypt
- R_SCE_AES_128CbcDecrypt
- R_SCE_AES_128CtrEncrypt
- R_SCE_AES_128EcbEncrypt
- R_SCE_AES_128EcbDecrypt
- R_SCE_AES_128GcmInitialize
- R_SCE_AES_128GcmTagCompute
- r_sce_aes_128gcmtag_compute_and_verify
- R_SCE_AES_128GcmOpen
- R_SCE_AES_128GcmEncrypt
- R_SCE_AES_128GcmGetGcmTag
- R_SCE_AES_128GcmSetGcmTag
- R_SCE_AES_128GcmDecrypt
- R_SCE_AES_128GcmZeroPaddingEncrypt
- R_SCE_AES_128GcmZeroPaddingDecrypt
- R_SCE_AES_128XtsEncrypt
- R_SCE_AES_128XtsDecrypt
- R_SCE_AES_192CbcEncrypt
- R_SCE_AES_192CbcDecrypt
- R_SCE_AES_192CtrEncrypt
- R_SCE_AES_192EcbEncrypt
- R_SCE_AES_192EcbDecrypt
- R_SCE_AES_192GcmInitialize
- R_SCE_AES_192GcmTagCompute
- r_sce_aes_192gcmtag_compute_and_verify
- R_SCE_AES_192GcmOpen
- R_SCE_AES_192GcmEncrypt
- R_SCE_AES_192GcmGetGcmTag
- R_SCE_AES_192GcmSetGcmTag
- R_SCE_AES_192GcmDecrypt
- R_SCE_AES_192GcmZeroPaddingEncrypt
- R_SCE_AES_192GcmZeroPaddingDecrypt
- R_SCE_AES_256CbcEncrypt

- R_SCE_AES_256CbcDecrypt
- R_SCE_AES_256CtrEncrypt
- R_SCE_AES_256EcbEncrypt
- R_SCE_AES_256EcbDecrypt
- R_SCE_AES_256GcmInitialize
- R_SCE_AES_256GcmTagCompute
- r_sce_aes_256gcmtag_compute_and_verify
- R_SCE_AES_256GcmOpen
- R_SCE_AES_256GcmEncrypt
- R_SCE_AES_256GcmGetGcmTag
- R_SCE_AES_256GcmSetGcmTag
- R_SCE_AES_256GcmDecrypt
- R_SCE_AES_256GcmZeroPaddingEncrypt
- R_SCE_AES_256GcmZeroPaddingDecrypt
- R_SCE_AES_256XtsEncrypt
- R_SCE_AES_256XtsDecrypt
- R_SCE_AES_EImpl_CreateKey
- R_SCE_AES_EImpl_EncryptFinal
- R_SCE_AES_EImpl_ZeroPaddingEncrypt
- R_SCE_AES_EImpl_ZeroPaddingDecrypt
- R_SCE_AES_EImpl_SetGcmTag
- R_SCE_AES_EImpl_GetGcmTag
- R_SCE_AES_EImpl_AddAdditionalAuthenticationData

Variables

- s_sce_aes_version
- g_aes128cbc_on_sce
- g_aes128ctr_on_sce
- g_aes128ecb_on_sce
- g_aes128gcm_on_sce
- g_aes128xts_on_sce
- g_aes192cbc_on_sce
- g_aes192ctr_on_sce
- g_aes192ecb_on_sce
- g_aes192gcm_on_sce

- g_aes256cbc_on_sce
- g_aes256ctr_on_sce
- g_aes256ecb_on_sce
- g_aes256gcm_on_sce
- g_aes256xts_on_sce

Defines

- #define SCE_AES_CODE_VERSION_MAJOR
  Initial value: `(01)`
- #define SCE_AES_CODE_VERSION_MINOR
  Initial value: `(00)`
- #define SIZE_GCMTAG_BITS
  Initial value: `(128)`
- #define SIZE_GCMTAG_BYTES
  Initial value: `( (SIZE_GCMTAG_BITS) / 8)`
- #define SIZE_GCMTAG_WORDS
  Initial value: `( (SIZE_GCMTAG_BITS) / 32)`
- #define SIZE_AES_BLOCK_BITS
  Initial value: `(128)`
- #define SIZE_AES_BLOCK_BYTES
  Initial value: `((SIZE_AES_BLOCK_BITS) / 8)`
- #define SIZE_AES_BLOCK_WORDS
  Initial value: `((SIZE_AES_BLOCK_BITS) / 32)`
- #define SIZE_AES_128BIT_KEYLEN_BITS
  Initial value: `(128)`
- #define SIZE_AES_128BIT_KEYLEN_BYTES
  Initial value: `((SIZE_AES_128BIT_KEYLEN_BITS) / 8)`
- #define SIZE_AES_128BIT_KEYLEN_WORDS
  Initial value: `((SIZE_AES_128BIT_KEYLEN_BITS) / 32)`
- #define SIZE_AES_192BIT_KEYLEN_BITS
  Initial value: `(192)`
- #define SIZE_AES_192BIT_KEYLEN_BYTES
  Initial value: `((SIZE_AES_192BIT_KEYLEN_BITS) / 8)`

- #define SIZE_AES_192BIT_KEYLEN_WORDS

  Initial value: `((SIZE_AES_192BIT_KEYLEN_BITS) / 32)`
- #define SIZE_AES_256BIT_KEYLEN_BITS

  Initial value: `(256)`
- #define SIZE_AES_256BIT_KEYLEN_BYTES

  Initial value: `((SIZE_AES_256BIT_KEYLEN_BITS) / 8)`
- #define SIZE_AES_256BIT_KEYLEN_WORDS

  Initial value: `((SIZE_AES_256BIT_KEYLEN_BITS) / 32)`

s_sce_aes_version

`ssp_version_t::s_sce_aes_version`

Detailed description

SCE/AES implementation of AES API. Version data structure used by error logger macro.

Initialized as

```
s_sce_aes_version=
{
    .api_version_major  = #define  AES_API_VERSION_MAJOR,
    .api_version_minor  = AES_API_VERSION_MINOR,
    .code_version_major = SCE_AES_CODE_VERSION_MAJOR,
    .code_version_minor = SCE_AES_CODE_VERSION_MINOR
}
```

g_aes128cbc_on_sce

`aes_api_t::g_aes128cbc_on_sce`

Detailed description

AES 128-bit CBC mode implementation

Initialized as

```
g_aes128cbc_on_sce=
{
    .open                              = R_SCE_AES_Open,
    .encrypt                           = R_SCE_AES_128CbcEncrypt,
    .decrypt                           = R_SCE_AES_128CbcDecrypt,
    .close                             = R_SCE_AES_Close,
    .versionGet                        = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes128ctr_on_sce

aes_api_t::g_aes128ctr_on_sce

Detailed description

AES 128-bit CTR mode implementation

Initialized as

```
g_aes128ctr_on_sce=
{
    .open                              = R_SCE_AES_Open,
    .encrypt                           = R_SCE_AES_128CtrEncrypt,
    .decrypt                           = R_SCE_AES_128CtrEncrypt,
    .close                             = R_SCE_AES_Close,
    .versionGet                        = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes128ecb_on_sce

aes_api_t::g_aes128ecb_on_sce

Detailed description

AES 128-bit ECB mode implementation

Initialized as

```
g_aes128ecb_on_sce=
{
    .open                              = R_SCE_AES_Open,
    .encrypt                           = R_SCE_AES_128EcbEncrypt,
    .decrypt                           = R_SCE_AES_128EcbDecrypt,
    .close                             = R_SCE_AES_Close,
    .versionGet                        = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes128gcm_on_sce

aes_api_t::g_aes128gcm_on_sce

Detailed description

AES 128-bit GCM mode implementation

Initialized as

```
g_aes128gcm_on_sce=
{
    .open                               = R_SCE_AES_128GcmOpen,
    .encrypt                            = R_SCE_AES_128GcmEncrypt,
    .decrypt                            = R_SCE_AES_128GcmDecrypt,
    .getGcmTag                          = R_SCE_AES_128GcmGetGcmTag,
    .setGcmTag                          = R_SCE_AES_128GcmSetGcmTag,
    .close                              = R_SCE_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .zeroPaddingEncrypt                 = R_SCE_AES_128GcmZeroPaddingEncrypt,
    .zeroPaddingDecrypt                 = R_SCE_AES_128GcmZeroPaddingDecrypt,
    .createKey                          = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                       = R_SCE_AES_EImpl_EncryptFinal,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData
}
```

g_aes128xts_on_sce

aes_api_t::g_aes128xts_on_sce

Detailed description

AES 128-bit CCM mode implementation

Initialized as

```
g_aes128xts_on_sce=
{
    .open                               = R_SCE_AES_Open,
    .encrypt                            = R_SCE_AES_128XtsEncrypt,
    .decrypt                            = R_SCE_AES_128XtsDecrypt,
    .close                              = R_SCE_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .createKey                          = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                       = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                 = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                 = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                          = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                          = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes192cbc_on_sce

aes_api_t::g_aes192cbc_on_sce

Detailed description

AES 192-bit CBC mode implementation

Initialized as

```
g_aes192cbc_on_sce=
{
    .open                              = R_SCE_AES_Open,
    .encrypt                           = R_SCE_AES_192CbcEncrypt,
    .decrypt                           = R_SCE_AES_192CbcDecrypt,
    .close                             = R_SCE_AES_Close,
    .versionGet                        = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes192ctr_on_sce

aes_api_t::g_aes192ctr_on_sce

Detailed description

AES 192-bit CTR mode implementation

Initialized as

```
g_aes192ctr_on_sce=
{
    .open                              = R_SCE_AES_Open,
    .encrypt                           = R_SCE_AES_192CtrEncrypt,
    .decrypt                           = R_SCE_AES_192CtrEncrypt,
    .close                             = R_SCE_AES_Close,
    .versionGet                        = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes192ecb_on_sce

aes_api_t::g_aes192ecb_on_sce

Detailed description

AES 192-bit ECB mode implementation

Initialized as

```
g_aes192ecb_on_sce=
{
    .open                              = R_SCE_AES_Open,
    .encrypt                           = R_SCE_AES_192EcbEncrypt,
    .decrypt                           = R_SCE_AES_192EcbDecrypt,
    .close                             = R_SCE_AES_Close,
    .versionGet                        = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes192gcm_on_sce

aes_api_t::g_aes192gcm_on_sce

Detailed description

AES 192-bit GCM mode implementation

Initialized as

```
g_aes192gcm_on_sce=
{
    .open                              = R_SCE_AES_192GcmOpen,
    .encrypt                           = R_SCE_AES_192GcmEncrypt,
    .decrypt                           = R_SCE_AES_192GcmDecrypt,
    .getGcmTag                         = R_SCE_AES_192GcmGetGcmTag,
    .setGcmTag                         = R_SCE_AES_192GcmSetGcmTag,
    .close                             = R_SCE_AES_Close,
    .versionGet                        = R_SCE_AES_VersionGet,
    .zeroPaddingEncrypt                = R_SCE_AES_192GcmZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_192GcmZeroPaddingDecrypt,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
}
```

g_aes256cbc_on_sce

aes_api_t::g_aes256cbc_on_sce

Detailed description

AES 256-bit CBC mode implementation

Initialized as

```
g_aes256cbc_on_sce=
{
    .open                              = R_SCE_AES_Open,
    .encrypt                           = R_SCE_AES_256CbcEncrypt,
    .decrypt                           = R_SCE_AES_256CbcDecrypt,
    .close                             = R_SCE_AES_Close,
    .versionGet                        = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes256ctr_on_sce

aes_api_t::g_aes256ctr_on_sce

Detailed description

AES 256-bit CTR mode implementation

Initialized as

```
g_aes256ctr_on_sce=
{
    .open                              = R_SCE_AES_Open,
    .encrypt                           = R_SCE_AES_256CtrEncrypt,
    .decrypt                           = R_SCE_AES_256CtrEncrypt,
    .close                             = R_SCE_AES_Close,
    .versionGet                        = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes256ecb_on_sce

aes_api_t::g_aes256ecb_on_sce

Detailed description

AES 256-bit ECB mode implementation

Initialized as

```
g_aes256ecb_on_sce=
{
    .open                              = R_SCE_AES_Open,
    .encrypt                           = R_SCE_AES_256EcbEncrypt,
    .decrypt                           = R_SCE_AES_256EcbDecrypt,
    .close                             = R_SCE_AES_Close,
    .versionGet                        = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes256gcm_on_sce

aes_api_t::g_aes256gcm_on_sce

Detailed description

AES 256-bit GCM mode implementation

Initialized as

```
g_aes256gcm_on_sce=
{
    .open                              = R_SCE_AES_256GcmOpen,
    .encrypt                           = R_SCE_AES_256GcmEncrypt,
    .decrypt                           = R_SCE_AES_256GcmDecrypt,
    .getGcmTag                         = R_SCE_AES_256GcmGetGcmTag,
    .setGcmTag                         = R_SCE_AES_256GcmSetGcmTag,
    .close                             = R_SCE_AES_Close,
    .versionGet                        = R_SCE_AES_VersionGet,
    .zeroPaddingEncrypt                = R_SCE_AES_256GcmZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_256GcmZeroPaddingDecrypt,
    .createKey                         = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
}
```

g_aes256xts_on_sce

aes_api_t::g_aes256xts_on_sce

Detailed description

AES 256-bit XTS mode implementation

Initialized as

```
g_aes256xts_on_sce=
{
    .open                                = R_SCE_AES_Open,
    .encrypt                             = R_SCE_AES_256XtsEncrypt,
    .decrypt                             = R_SCE_AES_256XtsDecrypt,
    .close                               = R_SCE_AES_Close,
    .versionGet                          = R_SCE_AES_VersionGet,
    .createKey                           = R_SCE_AES_EImpl_CreateKey,
    .encryptFinal                        = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                  = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                  = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                           = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                           = R_SCE_AES_EImpl_GetGcmTag
}
```

R_SCE_AES_Open

```
R_SCE_AES_Open ( aes_ctrl_t  *const p_ctrl ,  aes_cfg_t  const *const p_cfg )
```

Detailed description

AES Initialization

Table 1932:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE resource is busy
SSP_ERR_CRYPTO_SCE_FAIL	SCE internal I/O is not empty

R_SCE_AES_VersionGet

```
R_SCE_AES_VersionGet ( ssp_version_t  *const p_version )
```

Brief description

Sets driver version based on compile time macros.

Detailed description

Table 1933:

Name	Description
SSP_SUCCESS	Successful close.

Table 1933: (Continued)

Name	Description
SSP_ERR_ASSERTION	The parameter p_version is NULL.

R_SCE_AES_Close

```
R_SCE_AES_Close ( aes_ctrl_t  *const p_ctrl )
```

Detailed description

AES Close function

Table 1934:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE resource is busy
SSP_ERR_CRYPTO_SCE_FAIL	SCE internal I/O is not empty

R_SCE_AES_128CbcEncrypt

```
R_SCE_AES_128CbcEncrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t * p_key ,
  uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,   uint32_t
* p_dest )
```

Detailed description

AES 128-bit CBC mode implementation for encrypt interface API

Encrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1935:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_AES_128CbcDecrypt

```
R_SCE_AES_128CbcDecrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
 uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

Decrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1936:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_AES_128CtrEncrypt

```
R_SCE_AES_128CtrEncrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
  uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

AES 128-bit CTR mode implementation for encrypt interface API

Encrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1937:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_AES_128EcbEncrypt

```
R_SCE_AES_128EcbEncrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
  uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

AES 128-bit ECB mode implementation for encrypt interface API

Encrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1938:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: The contents of p_iv buffer are ignored in ECB chaining mode. NULL value is acceptable.

R_SCE_AES_128EcbDecrypt

```
R_SCE_AES_128EcbDecrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t * p_key ,
  uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,   uint32_t
* p_dest )
```

Detailed description

AES 128-bit ECB mode implementation for decrypt interface API

Decrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1939:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: The contents of p_iv buffer are ignored in ECB chaining mode. NULL value is acceptable.

R_SCE_AES_128GcmInitialize

```
 R_SCE_AES_128GcmInitialize ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv )
```

R_SCE_AES_128GcmTagCompute

```
 R_SCE_AES_128GcmTagCompute ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t * p_dest )
```

r_sce_aes_128gcmtag_compute_and_verify

```
 r_sce_aes_128gcmtag_compute_and_verify ( aes_ctrl_t  *const p_ctrl ,   const
uint32_t * p_key ,   uint32_t * p_iv ,   uint32_t * p_dest )
```

R_SCE_AES_128GcmOpen

```
 R_SCE_AES_128GcmOpen ( aes_ctrl_t  *const p_ctrl ,  aes_cfg_t  const
*const p_cfg )
```

R_SCE_AES_128GcmEncrypt

```
 R_SCE_AES_128GcmEncrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t * p_key ,
  uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,   uint32_t
* p_dest )
```

Detailed description

Encrypt num_words words of input data from buffer p_source using the 128-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1940:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_AES_128GcmGetGcmTag

```
R_SCE_AES_128GcmGetGcmTag ( aes_ctrl_t  *const p_ctrl ,   uint32_t num_words ,
 uint32_t * p_dest )
```

R_SCE_AES_128GcmSetGcmTag

```
R_SCE_AES_128GcmSetGcmTag ( aes_ctrl_t  *const p_ctrl ,   uint32_t num_words ,
 uint32_t * p_source )
```

R_SCE_AES_128GcmDecrypt

```
R_SCE_AES_128GcmDecrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t * p_key ,
 uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,   uint32_t
* p_dest )
```

Detailed description

Decrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1941:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

NOTE: For description of memcpy, memcmp and memset functions refer to C Standard library <string.h>

R_SCE_AES_128GcmZeroPaddingEncrypt

```
 R_SCE_AES_128GcmZeroPaddingEncrypt ( aes_ctrl_t  *const p_ctrl ,   const
uint32_t * p_key ,   uint32_t * p_iv ,   uint32_t num_bytes ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Detailed description

Encrypt num_bytes bytes of input data from buffer p_source using the 128-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data.

Table 1942:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least (num_bytes/16+1)*16 bytes of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

NOTE: this function is not thread safe.

R_SCE_AES_128GcmZeroPaddingDecrypt

```
 R_SCE_AES_128GcmZeroPaddingDecrypt ( aes_ctrl_t  *const p_ctrl ,   const
uint32_t * p_key ,   uint32_t * p_iv ,   uint32_t num_bytes ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Detailed description

Decrypt num_bytes bytes of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data.

Table 1943:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_bytes bytes of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

NOTE: For description of memcpy, memcmp and memset functions refer to C Standard library <string.h>

R_SCE_AES_128XtsEncrypt

```
R_SCE_AES_128XtsEncrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
  uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

AES 128-bit XTS mode implementation for encrypt interface API Encrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1944:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_AES_128XtsDecrypt

```
R_SCE_AES_128XtsDecrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
 uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

AES 128-bit XTS mode implementation for decrypt interface API Decrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1945:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_AES_192CbcEncrypt

```
R_SCE_AES_192CbcEncrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

AES 192-bit CBC mode implementation for encrypt interface API Encrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 1946:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 24 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_AES_192CbcDecrypt

```
R_SCE_AES_192CbcDecrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
 uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

AES 192-bit CBC mode implementation for decrypt interface API Decrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 1947:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 24 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_AES_192CtrEncrypt

```
R_SCE_AES_192CtrEncrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
 uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

AES 192-bit CTR mode implementation for encrypt and decrypt interface APIs Encrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1948:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 24 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_AES_192EcbEncrypt

```
R_SCE_AES_192EcbEncrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
 uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

AES 192-bit ECB mode implementation for encrypt interface API

Encrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1949:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 24 bytes of AES key data and

NOTE: The contents of p_iv buffer are ignored in ECB chaining mode. NULL value is acceptable.

R_SCE_AES_192EcbDecrypt

```
R_SCE_AES_192EcbDecrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
 uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

AES 192-bit ECB mode implementation for decrypt interface API Decrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1950:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 24 bytes of AES key data and

NOTE: The contents of p_iv buffer are ignored in ECB chaining mode. NULL value is acceptable.

R_SCE_AES_192GcmInitialize

```
 R_SCE_AES_192GcmInitialize ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv )
```

R_SCE_AES_192GcmTagCompute

```
 R_SCE_AES_192GcmTagCompute ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t * p_dest )
```

r_sce_aes_192gcmtag_compute_and_verify

```
 r_sce_aes_192gcmtag_compute_and_verify ( aes_ctrl_t  *const p_ctrl ,   const
uint32_t * p_key ,   uint32_t * p_iv ,   uint32_t * p_dest )
```

R_SCE_AES_192GcmOpen

```
 R_SCE_AES_192GcmOpen ( aes_ctrl_t  *const p_ctrl ,  aes_cfg_t  const
*const p_cfg )
```

R_SCE_AES_192GcmEncrypt

```
 R_SCE_AES_192GcmEncrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t * p_key ,
  uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,   uint32_t
* p_dest )
```

Detailed description

Encrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1951:

Name	Description
SF_CRYPTO_SUCCESS	Normal end

Table 1951: (Continued)

Name	Description
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_AES_192GcmGetGcmTag

```
R_SCE_AES_192GcmGetGcmTag ( aes_ctrl_t  *const p_ctrl ,   uint32_t num_words ,
 uint32_t * p_dest )
```

R_SCE_AES_192GcmSetGcmTag

```
R_SCE_AES_192GcmSetGcmTag ( aes_ctrl_t  *const p_ctrl ,   uint32_t num_words ,
 uint32_t * p_source )
```

R_SCE_AES_192GcmDecrypt

```
R_SCE_AES_192GcmDecrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t * p_key ,
 uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,   uint32_t
* p_dest )
```

Detailed description

Decrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1952:

Name	Description
SF_CRYPTO_SUCCESS	Normal end

Table 1952: (Continued)

Name	Description
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

NOTE: For description of memcpy, memcmp and memset functions refer to C Standard library <string.h>

R_SCE_AES_192GcmZeroPaddingEncrypt

```
 R_SCE_AES_192GcmZeroPaddingEncrypt ( aes_ctrl_t  *const p_ctrl ,   const
uint32_t * p_key ,   uint32_t * p_iv ,   uint32_t num_bytes ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Detailed description

Encrypt num_bytes bytes of input data from buffer p_source using the 128-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data.

Table 1953:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty

Table 1953: (Continued)

Name	Description
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_bytes bytes of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data

NOTE: this function is not thread safe.

R_SCE_AES_192GcmZeroPaddingDecrypt

```
 R_SCE_AES_192GcmZeroPaddingDecrypt ( aes_ctrl_t  *const p_ctrl ,   const
uint32_t * p_key ,   uint32_t * p_iv ,   uint32_t num_bytes ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Detailed description

Decrypt num_bytes bytes of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data.

Table 1954:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_bytes bytes of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

NOTE: For description of memcpy, memcmp and memset functions refer to C Standard library <string.h>

R_SCE_AES_256CbcEncrypt

```
R_SCE_AES_256CbcEncrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
  uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

AES 256-bit CBC mode implementation for encrypt interface API Encrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1955:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

R_SCE_AES_256CbcDecrypt

```
R_SCE_AES_256CbcDecrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
  uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

AES 256-bit CBC mode implementation for decrypt interface API Decrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1956:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

R_SCE_AES_256CtrEncrypt

```
R_SCE_AES_256CtrEncrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
  uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

AES 256-bit CTR mode implementation for encrypt and decrypt interface APIs Encrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1957:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

R_SCE_AES_256EcbEncrypt

```
R_SCE_AES_256EcbEncrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
  uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

AES 256-bit ECB mode implementation for encrypt interface API Encrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1958:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: The contents of p_iv buffer are ignored in ECB chaining mode. NULL value is acceptable.

R_SCE_AES_256EcbDecrypt

```
R_SCE_AES_256EcbDecrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
 uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

Decrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1959:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: The contents of p_iv buffer are ignored in ECB chaining mode. NULL value is acceptable.

R_SCE_AES_256GcmInitialize

```
R_SCE_AES_256GcmInitialize ( aes_ctrl_t *const p_ctrl , const uint32_t
* p_key , uint32_t * p_iv )
```

R_SCE_AES_256GcmTagCompute

```
R_SCE_AES_256GcmTagCompute ( aes_ctrl_t *const p_ctrl , const uint32_t
* p_key , uint32_t * p_iv , uint32_t * p_dest )
```

r_sce_aes_256gcmtag_compute_and_verify

```
r_sce_aes_256gcmtag_compute_and_verify ( aes_ctrl_t *const p_ctrl , const
uint32_t * p_key , uint32_t * p_iv , uint32_t * p_dest )
```

R_SCE_AES_256GcmOpen

```
R_SCE_AES_256GcmOpen ( aes_ctrl_t *const p_ctrl , aes_cfg_t const
*const p_cfg )
```

R_SCE_AES_256GcmEncrypt

```
R_SCE_AES_256GcmEncrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

Encrypt num_words words of input data from buffer p_source using the 256-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1960:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_AES_256GcmGetGcmTag

```
R_SCE_AES_256GcmGetGcmTag ( aes_ctrl_t *const p_ctrl , uint32_t num_words ,
uint32_t * p_dest )
```

R_SCE_AES_256GcmSetGcmTag

```
R_SCE_AES_256GcmSetGcmTag ( aes_ctrl_t  *const p_ctrl ,   uint32_t num_words ,
 uint32_t * p_source )
```

R_SCE_AES_256GcmDecrypt

```
R_SCE_AES_256GcmDecrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t * p_key ,
 uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,   uint32_t
* p_dest )
```

Detailed description

Decrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1961:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

NOTE: For description of memcpy, memcmp and memset functions refer to C Standard library <string.h>

R_SCE_AES_256GcmZeroPaddingEncrypt

```
R_SCE_AES_256GcmZeroPaddingEncrypt ( aes_ctrl_t  *const p_ctrl ,   const
uint32_t * p_key ,   uint32_t * p_iv ,   uint32_t num_bytes ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Detailed description

Encrypt num_bytes bytes of input data from buffer p_source using the 128-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data.

Table 1962:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_bytes bytes of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

NOTE: this function is not thread safe.

R_SCE_AES_256GcmZeroPaddingDecrypt

```
 R_SCE_AES_256GcmZeroPaddingDecrypt ( aes_ctrl_t  *const p_ctrl ,   const
uint32_t * p_key ,   uint32_t * p_iv ,   uint32_t num_bytes ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Detailed description

Decrypt num_bytes bytes of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_bytes bytes of data.

Table 1963:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_bytes bytes of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

NOTE: For description of memcpy, memcmp and memset functions refer to C Standard library <string.h>

R_SCE_AES_256XtsEncrypt

```
R_SCE_AES_256XtsEncrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
  uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

AES 256-bit XTS mode implementation for encrypt interface API Encrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1964:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_AES_256XtsDecrypt

```
R_SCE_AES_256XtsDecrypt ( aes_ctrl_t *const p_ctrl , const uint32_t * p_key ,
 uint32_t * p_iv , uint32_t num_words , uint32_t * p_source , uint32_t
* p_dest )
```

Detailed description

AES 256-bit XTS mode implementation for decrypt interface API Decrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1965:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_AES_EImpl_CreateKey

```
R_SCE_AES_EImpl_CreateKey ( aes_ctrl_t  *const p_ctrl ,   uint32_t num_words ,
 uint32_t * p_key )
```

Detailed description

SCE/AES implementation of AES API.

Table 1966:

Name	Description
SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.

R_SCE_AES_EImpl_EncryptFinal

```
R_SCE_AES_EImpl_EncryptFinal ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t input_num_words ,   uint32_t
* p_source ,   uint32_t output_num_words ,   uint32_t * p_dest )
```

Detailed description

Table 1967:

Name	Description
SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.

R_SCE_AES_EImpl_ZeroPaddingEncrypt

```
R_SCE_AES_EImpl_ZeroPaddingEncrypt ( aes_ctrl_t *const p_ctrl , const
uint32_t * p_key , uint32_t * p_iv , uint32_t num_bytes , uint32_t
* p_source , uint32_t * p_dest )
```

Detailed description

Table 1968:

Name	Description
SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.

R_SCE_AES_EImpl_ZeroPaddingDecrypt

```
R_SCE_AES_EImpl_ZeroPaddingDecrypt ( aes_ctrl_t *const p_ctrl , const
uint32_t * p_key , uint32_t * p_iv , uint32_t num_bytes , uint32_t
* p_source , uint32_t * p_dest )
```

Detailed description

Table 1969:

Name	Description
SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.

R_SCE_AES_EImpl_SetGcmTag

```
R_SCE_AES_EImpl_SetGcmTag ( aes_ctrl_t *const p_ctrl , uint32_t num_words ,
uint32_t * p_source )
```

Detailed description

Table 1970:

Name	Description
SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.

R_SCE_AES_EImpl_GetGcmTag

```
R_SCE_AES_EImpl_GetGcmTag ( aes_ctrl_t *const p_ctrl , uint32_t num_words ,
uint32_t * p_dest )
```

Detailed description

Table 1971:

Name	Description
SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.

R_SCE_AES_EImpl_AddAdditionalAuthenticationData

```
R_SCE_AES_EImpl_AddAdditionalAuthenticationData ( aes_ctrl_t  *const p_ctrl ,
  const uint32_t * p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t
* p_source )
```

Detailed description

Table 1972:

Name	Description
SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.

9.55.8.2 SCE HRK AES

Primitive cryptographic functions.

AES key generation, encryption and decryption functions for wrapped keys.

SCE HRK Encrypted Key AES Cipher implementation functions

Functions

- R_SCE_HRK_AES_Open
- R_SCE_HRK_AES_Close
- R_SCE_HRK_AES_VersionGet
- R_SCE_HRK_AES_128CreateEncryptedKey
- R_SCE_HRK_AES_192CreateEncryptedKey
- R_SCE_HRK_AES_256CreateEncryptedKey
- R_SCE_HRK_AES_128CbcEncrypt
- R_SCE_HRK_AES_128CbcDecrypt
- R_SCE_HRK_AES_128CtrEncrypt
- R_SCE_HRK_AES_128EcbEncrypt
- R_SCE_HRK_AES_128EcbDecrypt
- R_SCE_HRK_AES_128GcmOpen
- R_SCE_HRK_AES_128GcmEncrypt
- R_SCE_HRK_AES_128GcmZeroPaddingEncrypt

- R_SCE_HRK_AES_128GcmGetGcmTag
- R_SCE_HRK_AES_128GcmSetGcmTag
- R_SCE_HRK_AES_128GcmDecrypt
- R_SCE_HRK_AES_128GcmZeroPaddingDecrypt
- R_SCE_HRK_AES_128GcmTagCompute
- R_SCE_HRK_AES_128XtsCreateEncryptedKey
- R_SCE_HRK_AES_128XtsEncrypt
- R_SCE_HRK_AES_128XtsDecrypt
- R_SCE_HRK_AES_192CbcEncrypt
- R_SCE_HRK_AES_192CbcDecrypt
- R_SCE_HRK_AES_192CtrEncrypt
- R_SCE_HRK_AES_192EcbEncrypt
- R_SCE_HRK_AES_192EcbDecrypt
- R_SCE_HRK_AES_192GcmInitialize
- R_SCE_HRK_AES_192GcmTagCompute
- r_sce_hrk_aes_192gcmtag_compute_and_verify
- R_SCE_HRK_AES_192GcmOpen
- R_SCE_HRK_AES_192GcmEncrypt
- R_SCE_HRK_AES_192GcmGetGcmTag
- R_SCE_HRK_AES_192GcmSetGcmTag
- R_SCE_HRK_AES_192GcmDecrypt
- R_SCE_HRK_AES_192GcmZeroPaddingEncrypt
- R_SCE_HRK_AES_192GcmZeroPaddingDecrypt
- R_SCE_HRK_AES_256CbcEncrypt
- R_SCE_HRK_AES_256CbcDecrypt
- R_SCE_HRK_AES_256CtrEncrypt
- R_SCE_HRK_AES_256EcbEncrypt
- R_SCE_HRK_AES_256EcbDecrypt
- R_SCE_HRK_AES_256GcmInitialize
- R_SCE_HRK_AES_256GcmTagCompute
- r_sce_hrk_aes_256gcmtag_compute_and_verify
- R_SCE_HRK_AES_256GcmOpen
- R_SCE_HRK_AES_256GcmEncrypt
- R_SCE_HRK_AES_256GcmGetGcmTag

- R_SCE_HRK_AES_256GcmSetGcmTag
- R_SCE_HRK_AES_256GcmDecrypt
- R_SCE_HRK_AES_256GcmZeroPaddingEncrypt
- R_SCE_HRK_AES_256GcmZeroPaddingDecrypt
- R_SCE_HRK_AES_256XtsCreateEncryptedKey
- R_SCE_HRK_AES_256XtsEncrypt
- R_SCE_HRK_AES_256XtsDecrypt

Variables

- s_sce_hrk_aes_version
- g_aes128cbc_on_sceHrk
- g_aes128ctr_on_sceHrk
- g_aes128ecb_on_sceHrk
- g_aes128gcm_on_sceHrk
- g_aes128xts_on_sceHrk
- g_aes192cbc_on_sceHrk
- g_aes192ctr_on_sceHrk
- g_aes192ecb_on_sceHrk
- g_aes192gcm_on_sceHrk
- g_aes256cbc_on_sceHrk
- g_aes256ctr_on_sceHrk
- g_aes256ecb_on_sceHrk
- g_aes256gcm_on_sceHrk
- g_aes256xts_on_sceHrk

s_sce_hrk_aes_version

`ssp_version_t::s_sce_hrk_aes_version`

Detailed description

Version data structure used by error logger macro.

Initialized as

```
s_sce_hrk_aes_version=
{
    .api_version_major  = #define  AES_API_VERSION_MAJOR,
    .api_version_minor  = AES_API_VERSION_MINOR,
    .code_version_major = SCE_AES_CODE_VERSION_MAJOR,
    .code_version_minor = SCE_AES_CODE_VERSION_MINOR
}
```

g_aes128cbc_on_sceHrk

aes_api_t::g_aes128cbc_on_sceHrk

Initialized as

```
g_aes128cbc_on_sceHrk=
{
    .open                              = R_SCE_HRK_AES_Open,
    .encrypt                           = R_SCE_HRK_AES_128CbcEncrypt,
    .decrypt                           = R_SCE_HRK_AES_128CbcDecrypt,
    .close                             = R_SCE_HRK_AES_Close,
    .versionGet                        = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_HRK_AES_128CreateEncryptedKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes128ctr_on_sceHrk

aes_api_t::g_aes128ctr_on_sceHrk

Initialized as

```
g_aes128ctr_on_sceHrk=
{
    .open                              = R_SCE_HRK_AES_Open,
    .encrypt                           = R_SCE_HRK_AES_128CtrEncrypt,
    .decrypt                           = R_SCE_HRK_AES_128CtrEncrypt,
    .close                             = R_SCE_HRK_AES_Close,
    .versionGet                        = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_HRK_AES_128CreateEncryptedKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes128ecb_on_sceHrk

aes_api_t::g_aes128ecb_on_sceHrk

Detailed description

HRK Supported global structure definitions

Initialized as

```
g_aes128ecb_on_sceHrk=
{
```

```
    .open                                = R_SCE_HRK_AES_Open,
    .encrypt                             = R_SCE_HRK_AES_128EcbEncrypt,
    .decrypt                             = R_SCE_HRK_AES_128EcbDecrypt,
    .close                               = R_SCE_HRK_AES_Close,
    .versionGet                          = R_SCE_AES_VersionGet,
    .createKey                           = R_SCE_HRK_AES_128CreateEncryptedKey,
    .encryptFinal                        = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                  = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                  = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                           = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                           = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes128gcm_on_sceHrk

aes_api_t::g_aes128gcm_on_sceHrk

Initialized as

```
g_aes128gcm_on_sceHrk=
{
    .open       = R_SCE_HRK_AES_128GcmOpen,
    .createKey  = R_SCE_HRK_AES_128CreateEncryptedKey,
    .encrypt    = R_SCE_HRK_AES_128GcmEncrypt,
    .decrypt    = R_SCE_HRK_AES_128GcmDecrypt,
    .getGcmTag  = R_SCE_HRK_AES_128GcmGetGcmTag,
    .setGcmTag  = R_SCE_HRK_AES_128GcmSetGcmTag,
    .close      = R_SCE_HRK_AES_Close,
    .versionGet = R_SCE_HRK_AES_VersionGet,
    .zeroPaddingEncrypt = R_SCE_HRK_AES_128GcmZeroPaddingEncrypt,
    .zeroPaddingDecrypt = R_SCE_HRK_AES_128GcmZeroPaddingDecrypt
}
```

g_aes128xts_on_sceHrk

aes_api_t::g_aes128xts_on_sceHrk

Initialized as

```
g_aes128xts_on_sceHrk=
{
    .open                                = R_SCE_HRK_AES_Open,
    .encrypt                             = R_SCE_HRK_AES_128XtsEncrypt,
    .decrypt                             = R_SCE_HRK_AES_128XtsDecrypt,
    .close                               = R_SCE_HRK_AES_Close,
    .versionGet                          = R_SCE_AES_VersionGet,
    .createKey                           = R_SCE_HRK_AES_128XtsCreateEncryptedKey,
    .encryptFinal                        = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                  = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                  = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
```

```
    .setGcmTag                          = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                          = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes192cbc_on_sceHrk

aes_api_t::g_aes192cbc_on_sceHrk

Initialized as

```
g_aes192cbc_on_sceHrk=
{
    .open                               = R_SCE_HRK_AES_Open,
    .encrypt                            = R_SCE_HRK_AES_192CbcEncrypt,
    .decrypt                            = R_SCE_HRK_AES_192CbcDecrypt,
    .close                              = R_SCE_HRK_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .createKey                          = R_SCE_HRK_AES_192CreateEncryptedKey,
    .encryptFinal                       = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                 = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                 = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                          = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                          = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes192ctr_on_sceHrk

aes_api_t::g_aes192ctr_on_sceHrk

Initialized as

```
g_aes192ctr_on_sceHrk=
{
    .open                               = R_SCE_HRK_AES_Open,
    .encrypt                            = R_SCE_HRK_AES_192CtrEncrypt,
    .decrypt                            = R_SCE_HRK_AES_192CtrEncrypt,
    .close                              = R_SCE_HRK_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .createKey                          = R_SCE_HRK_AES_192CreateEncryptedKey,
    .encryptFinal                       = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                 = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                 = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                          = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                          = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes192ecb_on_sceHrk

aes_api_t::g_aes192ecb_on_sceHrk

Initialized as

```
g_aes192ecb_on_sceHrk=
{
    .open                              = R_SCE_HRK_AES_Open,
    .encrypt                           = R_SCE_HRK_AES_192EcbEncrypt,
    .decrypt                           = R_SCE_HRK_AES_192EcbDecrypt,
    .close                             = R_SCE_HRK_AES_Close,
    .versionGet                        = R_SCE_AES_VersionGet,
    .createKey                         = R_SCE_HRK_AES_192CreateEncryptedKey,
    .encryptFinal                      = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                         = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                         = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes192gcm_on_sceHrk

aes_api_t::g_aes192gcm_on_sceHrk

Detailed description

AES 192-bit GCM HRK mode implementation

Initialized as

```
g_aes192gcm_on_sceHrk=
{
    .open       = R_SCE_HRK_AES_192GcmOpen,
    .createKey  = R_SCE_HRK_AES_192CreateEncryptedKey,
    .encrypt    = R_SCE_HRK_AES_192GcmEncrypt,
    .decrypt    = R_SCE_HRK_AES_192GcmDecrypt,
    .getGcmTag  = R_SCE_HRK_AES_192GcmGetGcmTag,
    .setGcmTag  = R_SCE_HRK_AES_192GcmSetGcmTag,
    .close      = R_SCE_HRK_AES_Close,
    .versionGet = R_SCE_HRK_AES_VersionGet,
    .zeroPaddingEncrypt = R_SCE_HRK_AES_192GcmZeroPaddingEncrypt,
    .zeroPaddingDecrypt = R_SCE_HRK_AES_192GcmZeroPaddingDecrypt
}
```

g_aes256cbc_on_sceHrk

aes_api_t::g_aes256cbc_on_sceHrk

Initialized as

```
g_aes256cbc_on_sceHrk=
{
    .open                              = R_SCE_HRK_AES_Open,
    .encrypt                           = R_SCE_HRK_AES_256CbcEncrypt,
    .decrypt                           = R_SCE_HRK_AES_256CbcDecrypt,
    .close                             = R_SCE_HRK_AES_Close,
    .versionGet                        = R_SCE_AES_VersionGet,
```

```
    .createKey                          = R_SCE_HRK_AES_256CreateEncryptedKey,
    .encryptFinal                       = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                 = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                 = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                          = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                          = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes256ctr_on_sceHrk

aes_api_t::g_aes256ctr_on_sceHrk

Initialized as

```
g_aes256ctr_on_sceHrk=
{
    .open                               = R_SCE_HRK_AES_Open,
    .encrypt                            = R_SCE_HRK_AES_256CtrEncrypt,
    .decrypt                            = R_SCE_HRK_AES_256CtrEncrypt,
    .close                              = R_SCE_HRK_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .createKey                          = R_SCE_HRK_AES_256CreateEncryptedKey,
    .encryptFinal                       = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                 = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                 = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                          = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                          = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes256ecb_on_sceHrk

aes_api_t::g_aes256ecb_on_sceHrk

Initialized as

```
g_aes256ecb_on_sceHrk=
{
    .open                               = R_SCE_HRK_AES_Open,
    .encrypt                            = R_SCE_HRK_AES_256EcbEncrypt,
    .decrypt                            = R_SCE_HRK_AES_256EcbDecrypt,
    .close                              = R_SCE_HRK_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .createKey                          = R_SCE_HRK_AES_256CreateEncryptedKey,
    .encryptFinal                       = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                 = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                 = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                          = R_SCE_AES_EImpl_SetGcmTag,
```

```
    .getGcmTag                          = R_SCE_AES_EImpl_GetGcmTag
}
```

g_aes256gcm_on_sceHrk

aes_api_t::g_aes256gcm_on_sceHrk

Initialized as

```
g_aes256gcm_on_sceHrk=
{
    .open       = R_SCE_HRK_AES_256GcmOpen,
    .createKey  = R_SCE_HRK_AES_256CreateEncryptedKey,
    .encrypt    = R_SCE_HRK_AES_256GcmEncrypt,
    .decrypt    = R_SCE_HRK_AES_256GcmDecrypt,
    .getGcmTag  = R_SCE_HRK_AES_256GcmGetGcmTag,
    .setGcmTag  = R_SCE_HRK_AES_256GcmSetGcmTag,
    .close      = R_SCE_HRK_AES_Close,
    .versionGet = R_SCE_HRK_AES_VersionGet,
    .zeroPaddingEncrypt = R_SCE_HRK_AES_256GcmZeroPaddingEncrypt,
    .zeroPaddingDecrypt = R_SCE_HRK_AES_256GcmZeroPaddingDecrypt
}
```

g_aes256xts_on_sceHrk

aes_api_t::g_aes256xts_on_sceHrk

Initialized as

```
g_aes256xts_on_sceHrk=
{
    .open                               = R_SCE_HRK_AES_Open,
    .encrypt                            = R_SCE_HRK_AES_256XtsEncrypt,
    .decrypt                            = R_SCE_HRK_AES_256XtsDecrypt,
    .close                              = R_SCE_HRK_AES_Close,
    .versionGet                         = R_SCE_AES_VersionGet,
    .createKey                          = R_SCE_HRK_AES_256XtsCreateEncryptedKey,
    .encryptFinal                       = R_SCE_AES_EImpl_EncryptFinal,
    .zeroPaddingEncrypt                 = R_SCE_AES_EImpl_ZeroPaddingEncrypt,
    .zeroPaddingDecrypt                 = R_SCE_AES_EImpl_ZeroPaddingDecrypt,
    .addAdditionalAuthenticationData =
R_SCE_AES_EImpl_AddAdditionalAuthenticationData,
    .setGcmTag                          = R_SCE_AES_EImpl_SetGcmTag,
    .getGcmTag                          = R_SCE_AES_EImpl_GetGcmTag
}
```

R_SCE_HRK_AES_Open

```
 R_SCE_HRK_AES_Open ( aes_ctrl_t  *const p_ctrl ,  aes_cfg_t  const
*const p_cfg )
```

Detailed description

AES Initialization

Table 1973:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE resource is busy
SSP_ERR_CRYPTO_SCE_FAIL	SCE internal I/O is not empty

R_SCE_HRK_AES_Close

```
R_SCE_HRK_AES_Close ( aes_ctrl_t  *const p_ctrl )
```

Detailed description

AES Initialization

Table 1974:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE resource is busy
SSP_ERR_CRYPTO_SCE_FAIL	SCE internal I/O is not empty

R_SCE_HRK_AES_VersionGet

```
R_SCE_HRK_AES_VersionGet ( ssp_version_t  *const p_version )
```

R_SCE_HRK_AES_128CreateEncryptedKey

```
R_SCE_HRK_AES_128CreateEncryptedKey ( aes_ctrl_t  *const p_ctrl ,
 uint32_t num_words ,   uint32_t * p_key )
```

Brief description

Encrypted Wrapped key creation for AES 128-bit applies for CBC, ECB, CTR, GCM chaining mode implementations. Create a num_words words of wrapped key for AES XTS chaining mode. The result will be written to the output buffer p_key. The p_key array is assumed to have space for at least num_words words of data.

Detailed description

Table 1975:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty

Table 1975: (Continued)

Name	Description
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_INVALID_SIZE	Insufficient buffer size to accommodate created key

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

R_SCE_HRK_AES_192CreateEncryptedKey

```
R_SCE_HRK_AES_192CreateEncryptedKey ( aes_ctrl_t  *const p_ctrl ,
 uint32_t num_words ,   uint32_t * p_key )
```

Brief description

Encrypted Wrapped key creation for AES 192-bit applies for CBC, ECB, CTR, GCM chaining mode implementations. Create a num_words words of wrapped key for AES XTS chaining mode. The result will be written to the output buffer p_key. The p_key array is assumed to have space for at least num_words words of data.

Detailed description

Table 1976:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_INVALID_SIZE	Insufficient buffer size to accommodate created key

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

R_SCE_HRK_AES_256CreateEncryptedKey

```
R_SCE_HRK_AES_256CreateEncryptedKey ( aes_ctrl_t  *const p_ctrl ,
 uint32_t num_words ,   uint32_t * p_key )
```

Brief description

Encrypted Wrapped key creation for AES 256-bit applies for CBC, ECB, CTR, GCM chaining mode implementations. Create a num_words words of wrapped key for AES XTS chaining mode. The result will be written to the output buffer p_key. The p_key array is assumed to have space for at least num_words words of data.

Detailed description

Table 1977:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_INVALID_SIZE	Insufficient buffer size to accommodate created key

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

R_SCE_HRK_AES_128CbcEncrypt

```
 R_SCE_HRK_AES_128CbcEncrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted Key AES 128-bit CBC mode implementation for encrypt interface API

Encrypt num_words words of input data from buffer p_source using the wrapped 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 1978:

Name	Description
SF_CRYPTO_SUCCESS	Normal end

Table 1978: (Continued)

Name	Description
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 128-bit AES key generated using the R_SCE_HRK_AES_128CreateEncrypt-edKey.

R_SCE_HRK_AES_128CbcDecrypt

```
 R_SCE_HRK_AES_128CbcDecrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted Key AES 128-bit CBC mode implementation for decrypt interface API

Decrypt num_words words of input data from buffer p_source using the wrapped 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 1979:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty

Table 1979: (Continued)

Name	Description
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 128-bit AES key generated using the R_SCE_HRK_AES_128CreateEncrypt-edKey.

R_SCE_HRK_AES_128CtrEncrypt

```
 R_SCE_HRK_AES_128CtrEncrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted Key AES 128-bit CTR mode implementation for encrypt interface API

Encrypt num_words words of input data from buffer p_source using a wrapped 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1980:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

Table 1980: (Continued)

Name	Description
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 128-bit AES key generated using the R_SCE_HRK_AES_128CreateEncrypt-edKey.

R_SCE_HRK_AES_128EcbEncrypt

```
 R_SCE_HRK_AES_128EcbEncrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted Key AES 128-bit ECB mode implementation for encrypt interface API

Encrypt num_words words of input data from buffer p_source using the wrapped 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 1981:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 128-bit AES key generated using the R_SCE_HRK_AES_128CreateEncrypt-edKey.

NOTE: The contents of p_iv buffer are ignored and can be NULL in ECB chaining mode.

R_SCE_HRK_AES_128EcbDecrypt

```
 R_SCE_HRK_AES_128EcbDecrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted Key AES 128-bit ECB mode implementation for decrypt interface API

Decrypt num_words words of input data from buffer p_source using the wrapped 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1982:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 128-bit AES key generated using the R_SCE_HRK_AES_128CreateEncrypt-edKey.

NOTE: The contents of p_iv buffer are ignored and can be NULL in ECB chaining mode.

R_SCE_HRK_AES_128GcmOpen

```
 R_SCE_HRK_AES_128GcmOpen ( aes_ctrl_t  *const p_ctrl ,  aes_cfg_t  const
*const p_cfg )
```

R_SCE_HRK_AES_128GcmEncrypt

```
 R_SCE_HRK_AES_128GcmEncrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypt num_words words of input data from buffer p_source using the 128-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1983:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_HRK_AES_128GcmZeroPaddingEncrypt

```
 R_SCE_HRK_AES_128GcmZeroPaddingEncrypt ( aes_ctrl_t  *const p_ctrl ,   const
uint32_t * p_key ,   uint32_t * p_iv ,   uint32_t num_bytes ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

R_SCE_HRK_AES_128GcmGetGcmTag

```
 R_SCE_HRK_AES_128GcmGetGcmTag ( aes_ctrl_t  *const p_ctrl ,
  uint32_t num_words ,   uint32_t * p_dest )
```

R_SCE_HRK_AES_128GcmSetGcmTag

```
 R_SCE_HRK_AES_128GcmSetGcmTag ( aes_ctrl_t  *const p_ctrl ,
  uint32_t num_words ,   uint32_t * p_source )
```

R_SCE_HRK_AES_128GcmDecrypt

```
 R_SCE_HRK_AES_128GcmDecrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Decrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1984:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_HRK_AES_128GcmZeroPaddingDecrypt

```
 R_SCE_HRK_AES_128GcmZeroPaddingDecrypt ( aes_ctrl_t  *const p_ctrl ,   const
uint32_t * p_key ,   uint32_t * p_iv ,   uint32_t num_bytes ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

R_SCE_HRK_AES_128GcmTagCompute

```
 R_SCE_HRK_AES_128GcmTagCompute ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t * p_dest )
```

R_SCE_HRK_AES_128XtsCreateEncryptedKey

```
 R_SCE_HRK_AES_128XtsCreateEncryptedKey ( aes_ctrl_t  *const p_ctrl ,
  uint32_t num_words ,   uint32_t * p_key )
```

Detailed description

Encrypted Wrapped key creation for AES 1286-bit XTS mode implementation

Create a num_words words of wrapped key for AES XTS chaining mode. The result will be written to the output buffer p_key. The p_key array is assumed to have space for at least num_words words of data.

Table 1985:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_INVALID_SIZE	Insufficient buffer size to accommodate created key

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4 and must be less than 128M

R_SCE_HRK_AES_128XtsEncrypt

```
 R_SCE_HRK_AES_128XtsEncrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted Key AES 128-bit XTS mode implementation for encrypt interface API

Encrypt num_words words of input data from buffer p_source using the 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1986:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Invalid Key
SSP_ERR_INVALID_SIZE	Invalid size

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4 and must be less than 128M

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 128-bit AES key generated using the R_SCE_HRK_AES_128XtsCreateEncryptedKey.

Function steps

- Verify the upper-limit of num_words
- HW_SCE_AES_128XtsEncryptUsingEncryptedKey takes a pointer (InData_Len) whose value is in number of bits

R_SCE_HRK_AES_128XtsDecrypt

```
 R_SCE_HRK_AES_128XtsDecrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted Key AES 128-bit XTS mode implementation for decrypt interface API

Decrypt num_words words of input data from buffer p_source using a wrapped 128-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1987:

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Invalid Key
SSP_ERR_INVALID_SIZE	Invalid size

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4 and must be less than 128M

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 128-bit AES key generated using the R_SCE_HRK_AES_128CreateEncrypt-edKey.

Function steps

- Verify the upper-limit of num_words
- HW_SCE_AES_128XtsDecryptUsingEncryptedKey takes a pointer (InData_Len) whose value is in number of bits

R_SCE_HRK_AES_192CbcEncrypt

```
 R_SCE_HRK_AES_192CbcEncrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted key AES 192-bit CBC mode implementation for encrypt interface API

Encrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 1988:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 192-bit AES key generated using R_SCE_HRK_AES_192CreateEncrypt-

edKey.

R_SCE_HRK_AES_192CbcDecrypt

```
 R_SCE_HRK_AES_192CbcDecrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted key AES 192-bit CBC mode implementation for decrypt interface API

Decrypt num_words words of input data from buffer p_source using a wrapped 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 1989:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 192-bit AES key generated using the R_SCE_HRK_AES_192CreateEncrypt-edKey.

R_SCE_HRK_AES_192CtrEncrypt

```
 R_SCE_HRK_AES_192CtrEncrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted key AES 192-bit CTR mode implementation for encrypt interface API

Encrypt num_words words of input data from buffer p_source using a wrapped 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 1990:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 192-bit AES key generated using the R_SCE_HRK_AES_192CreateEncrypt-edKey.

R_SCE_HRK_AES_192EcbEncrypt

```
 R_SCE_HRK_AES_192EcbEncrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted key AES 192-bit ECB mode implementation for encrypt interface API

Encrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1991:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 192-bit AES key generated using the R_SCE_HRK_AES_192CreateEncrypt-edKey.

NOTE: The contents of p_iv buffer are ignored and can be NULL in ECB chaining mode.

R_SCE_HRK_AES_192EcbDecrypt

```
 R_SCE_HRK_AES_192EcbDecrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted key AES 192-bit ECB mode implementation for decrypt interface API

Decrypt num_words words of input data from buffer p_source using a wrapped 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 1992:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 192-bit AES key generated using the R_SCE_HRK_AES_192CreateEncrypt-edKey.

NOTE: The contents of p_iv buffer are ignored and can be NULL in ECB chaining mode.

R_SCE_HRK_AES_192GcmInitialize

```
 R_SCE_HRK_AES_192GcmInitialize ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv )
```

R_SCE_HRK_AES_192GcmTagCompute

```
 R_SCE_HRK_AES_192GcmTagCompute ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t * p_dest )
```

r_sce_hrk_aes_192gcmtag_compute_and_verify

```
r_sce_hrk_aes_192gcmtag_compute_and_verify ( aes_ctrl_t  *const p_ctrl ,
 const uint32_t * p_key ,   uint32_t * p_iv ,   uint32_t * p_dest )
```

R_SCE_HRK_AES_192GcmOpen

```
R_SCE_HRK_AES_192GcmOpen ( aes_ctrl_t  *const p_ctrl ,  aes_cfg_t  const
*const p_cfg )
```

R_SCE_HRK_AES_192GcmEncrypt

```
R_SCE_HRK_AES_192GcmEncrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypt num_words words of input data from buffer p_source using the 192-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1993:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_HRK_AES_192GcmGetGcmTag

```
R_SCE_HRK_AES_192GcmGetGcmTag ( aes_ctrl_t  *const p_ctrl ,
 uint32_t num_words ,   uint32_t * p_dest )
```

R_SCE_HRK_AES_192GcmSetGcmTag

```
R_SCE_HRK_AES_192GcmSetGcmTag ( aes_ctrl_t  *const p_ctrl ,
 uint32_t num_words ,   uint32_t * p_source )
```

R_SCE_HRK_AES_192GcmDecrypt

```
R_SCE_HRK_AES_192GcmDecrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
 uint32_t * p_dest )
```

Detailed description

Decrypt num_words words of input data from buffer p_source using the 192-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 1994:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_HRK_AES_192GcmZeroPaddingEncrypt

```
R_SCE_HRK_AES_192GcmZeroPaddingEncrypt ( aes_ctrl_t  *const p_ctrl ,   const
uint32_t * p_key ,   uint32_t * p_iv ,   uint32_t num_bytes ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

R_SCE_HRK_AES_192GcmZeroPaddingDecrypt

```
 R_SCE_HRK_AES_192GcmZeroPaddingDecrypt ( aes_ctrl_t  *const p_ctrl ,   const
uint32_t * p_key ,   uint32_t * p_iv ,   uint32_t num_bytes ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

R_SCE_HRK_AES_256CbcEncrypt

```
 R_SCE_HRK_AES_256CbcEncrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted AES 256-bit CBC mode implementation for encrypt interface API

Encrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 1995:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 256-bit AES key generated using the R_SCE_HRK_AES_256CreateEncrypt-edKey.

R_SCE_HRK_AES_256CbcDecrypt

```
 R_SCE_HRK_AES_256CbcDecrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted AES 256-bit ECB mode implementation for decrypt interface API

Decrypt num_words words of input data from buffer p_source using a wrapped 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 1996:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 256-bit AES key generated using the R_SCE_HRK_AES_256CreateEncrypt-edKey.

R_SCE_HRK_AES_256CtrEncrypt

```
 R_SCE_HRK_AES_256CtrEncrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted key AES 256-bit CTR mode implementation for encrypt interface API

Encrypt num_words words of input data from buffer p_source using a wrapped 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 1997:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 256-bit AES key generated using the R_SCE_HRK_AES_256CreateEncrypt-edKey.

R_SCE_HRK_AES_256EcbEncrypt

```
 R_SCE_HRK_AES_256EcbEncrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted key AES 256-bit ECB mode implementation for encrypt interface API

Encrypt num_words words of input data from buffer p_source using a wrapped 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 1998:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 256-bit AES key generated using the R_SCE_HRK_AES_256CreateEncrypt-edKey.

NOTE: The contents of p_iv buffer are ignored and can be NULL in ECB chaining mode.

R_SCE_HRK_AES_256EcbDecrypt

```
 R_SCE_HRK_AES_256EcbDecrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted key AES 256-bit ECB mode implementation for decrypt interface API

Decrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 1999:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 256-bit AES key generated using the R_SCE_HRK_AES_256CreateEncrypt-edKey.

NOTE: The contents of p_iv buffer are ignored and can be NULL in ECB chaining mode.

R_SCE_HRK_AES_256GcmInitialize

```
 R_SCE_HRK_AES_256GcmInitialize ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv )
```

R_SCE_HRK_AES_256GcmTagCompute

```
 R_SCE_HRK_AES_256GcmTagCompute ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t * p_dest )
```

r_sce_hrk_aes_256gcmtag_compute_and_verify

```
 r_sce_hrk_aes_256gcmtag_compute_and_verify ( aes_ctrl_t  *const p_ctrl ,
  const uint32_t * p_key ,   uint32_t * p_iv ,   uint32_t * p_dest )
```

R_SCE_HRK_AES_256GcmOpen

```
R_SCE_HRK_AES_256GcmOpen ( aes_ctrl_t  *const p_ctrl ,  aes_cfg_t  const
*const p_cfg )
```

R_SCE_HRK_AES_256GcmEncrypt

```
R_SCE_HRK_AES_256GcmEncrypt ( aes_ctrl_t  *const p_ctrl ,  const uint32_t
* p_key ,  uint32_t * p_iv ,  uint32_t num_words ,  uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypt num_words words of input data from buffer p_source using the 256-bit AES p_key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 2000:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_HRK_AES_256GcmGetGcmTag

```
R_SCE_HRK_AES_256GcmGetGcmTag ( aes_ctrl_t  *const p_ctrl ,
 uint32_t num_words ,  uint32_t * p_dest )
```

R_SCE_HRK_AES_256GcmSetGcmTag

```
R_SCE_HRK_AES_256GcmSetGcmTag ( aes_ctrl_t  *const p_ctrl ,
 uint32_t num_words ,  uint32_t * p_source )
```

R_SCE_HRK_AES_256GcmDecrypt

```
 R_SCE_HRK_AES_256GcmDecrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Decrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 2001:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Plain text key is passed

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 16 bytes of AES key data and

NOTE: the p_iv buffer must have at least 16 bytes of random data.

R_SCE_HRK_AES_256GcmZeroPaddingEncrypt

```
 R_SCE_HRK_AES_256GcmZeroPaddingEncrypt ( aes_ctrl_t  *const p_ctrl ,   const
uint32_t * p_key ,   uint32_t * p_iv ,   uint32_t num_bytes ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

R_SCE_HRK_AES_256GcmZeroPaddingDecrypt

```
 R_SCE_HRK_AES_256GcmZeroPaddingDecrypt ( aes_ctrl_t  *const p_ctrl ,   const
uint32_t * p_key ,   uint32_t * p_iv ,   uint32_t num_bytes ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

R_SCE_HRK_AES_256XtsCreateEncryptedKey

```
R_SCE_HRK_AES_256XtsCreateEncryptedKey ( aes_ctrl_t  *const p_ctrl ,
 uint32_t num_words ,   uint32_t * p_key )
```

Detailed description

Encrypted Wrapped key creation for AES 256-bit XTS mode implementation

Create a num_words words of wrapped key for AES XTS chaining mode. The result will be written to the output buffer p_key. The p_key array is assumed to have space for at least num_words words of data.

Table 2002:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_INVALID_SIZE	Insufficient buffer size to accommodate created key

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4

R_SCE_HRK_AES_256XtsEncrypt

```
 R_SCE_HRK_AES_256XtsEncrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted key AES 256-bit XTS mode implementation for encrypt interface API

Encrypt num_words words of input data from buffer p_source using the 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 2003:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end

Table 2003:Return values (Continued)

Name	Description
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Invalid Key
SSP_ERR_INVALID_SIZE	Invalid size

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4 and must be less than 128M

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 256-bit AES key generated using the R_SCE_HRK_AES_256XtsCreateEncryptedKey.

Function steps

- Verify the upper-limit of num_words
- HW_SCE_AES_256XtsEncryptUsingEncryptedKey takes a pointer (InData_Len) whose value is in number of bits

R_SCE_HRK_AES_256XtsDecrypt

```
 R_SCE_HRK_AES_256XtsDecrypt ( aes_ctrl_t  *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Detailed description

Encrypted key AES 256-bit XTS mode implementation for decrypt interface API

Decrypt num_words words of input data from buffer p_source using a wrapped 256-bit AES key from buffer p_key and initialization vector from buffer p_iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 2004:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred
SSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Invalid Key
SSP_ERR_INVALID_SIZE	Invalid size

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: num_words must be a multiple of 4 and must be less than 128M

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key is a wrapped/encrypted 256-bit AES key generated using the R_SCE_HRK_AES_256XtsCreateEncryptedKey.

Function steps

- Verify the upper-limit of num_words
- HW_SCE_AES_256XtsDecryptUsingEncryptedKey takes a pointer (InData_Len) whose value is in number of bits

9.55.8.3 SCE ARC4

Primitive cryptographic functions.

ARC4 encryption and decryption functions

Functions

- R_SCE_ARC4_Open
- R_SCE_ARC4_Close
- R_SCE_ARC4_Process

- R_SCE_ARC4_VersionGet
- R_SCE_ARC4_KeySet

R_SCE_ARC4_Open

```
R_SCE_ARC4_Open ( arc4_ctrl_t  *const p_ctrl ,  arc4_cfg_t  const *const p_cfg )
```

Detailed description

ARC4 Initialization

Table 2005:Return values

Name	Description
SF_CRYPTO_SUCCESS	successful.
SSP_ERR_CRYPTO_SCE_ALREADY_OPEN	ARC4 module is already in open state and is usable for the given p_ctrl parameter.
SSP_ERR_ASSERTION	An input parameter is invalid.

R_SCE_ARC4_Close

```
R_SCE_ARC4_Close ( arc4_ctrl_t  *const p_ctrl )
```

Detailed description

ARC4 Finalization

Table 2006:Return values

Name	Description
SF_CRYPTO_SUCCESS	successful

R_SCE_ARC4_Process

```
R_SCE_ARC4_Process ( arc4_ctrl_t  *const p_ctrl ,   uint32_t num_bytes ,
 uint8_t * p_source ,   uint8_t * p_dest )
```

Brief description

ARC4 Encrypt or decrypt source data and output result to destination buffer.

Detailed description

Table 2007:Return values

Name	Description
SSP_ERR_CRYPTO_INVALID_STATE	Invalid state, ensure that key data is configured either using the open() or using the keyset() function.
SSP_ERR_CRYPTO_INVALID_SIZE	invalid num_bytes passed. Should be a multiple of 16.
SF_CRYPTO_SUCCESS	successful.
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	An internal error has occurred.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Hardware is busy, unable to encrypt at this time.

Encrypt or decrypt input data using the previously configured key data

R_SCE_ARC4_VersionGet

```
R_SCE_ARC4_VersionGet ( ssp_version_t  *const p_version )
```

Brief description

Sets driver version based on compile time macros.

Detailed description

Table 2008:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

R_SCE_ARC4_KeySet

```
R_SCE_ARC4_KeySet ( arc4_ctrl_t  *const p_ctrl ,   uint32_t length ,   uint8_t
const * p_key )
```

Brief description

Sets user provided for use with subsequent encryptions.

Detailed description

Table 2009:Return values

Name	Description
SF_CRYPTO_SUCCESS	KeySet successful
SSP_ERR_NOT_OPEN	Module not opened.
SSP_ERR_ASSERTION	One of input parameter is NULL.

9.55.8.4 SCE DSA

Primitive cryptographic functions.

Primitive cryptographic functions (L=2048,N=256) DSA.

DSA signature generation and verification functions

DSA signature generation and verification functions (L=1024,N=160) DSA

DSA signature generation and verification functions (L=2048,N=224) DSA

Functions

- R_SCE_DSA_Open
- R_SCE_DSA_Close
- R_SCE_DSA_VersionGet
- R_SCE_DSA_1024_160SignatureVerify
- R_SCE_DSA_1024_160SignatureGenerate
- R_SCE_DSA_1024_160HashSignatureVerify
- R_SCE_DSA_1024_160HashSignatureGenerate
- R_SCE_DSA_2048_224SignatureVerify
- R_SCE_DSA_2048_224SignatureGenerate
- R_SCE_DSA_2048_224HashSignatureVerify
- R_SCE_DSA_2048_224HashSignatureGenerate
- R_SCE_DSA_2048_256SignatureVerify
- R_SCE_DSA_2048_256SignatureGenerate
- R_SCE_DSA_2048_256HashSignatureVerify
- R_SCE_DSA_2048_256HashSignatureGenerate

Variables

- s_sce_dsa_version
- g_dsa1024_160_on_sce

- g_dsa2048_224_on_sce
- g_dsa2048_256_on_sce

Defines

- #define DSA_PARAM_LENGTH_Q

  Initial value: `(160)`
- #define DSA_PARAM_LENGTH_P

  Initial value: `(1024)`
- #define DSA_PARAM_LENGTH_G

  Initial value: `(DSA_PARAM_LENGTH_P)`
- #define DSA_PRIVATE_KEYSIZE

  Initial value: `((DSA_PARAM_LENGTH_Q) / 32)`
- #define DSA_PUBLIC_KEYSIZE

  Initial value: `((DSA_PARAM_LENGTH_P) / 32)`
- #define DSA_DOMAIN_SIZE

  Initial value: `(DSA_PRIVATE_KEYSIZE + 2 * (DSA_PUBLIC_KEYSIZE))`
- #define DSA_SIGNATURE_SIZE

  Initial value: `(2 * (DSA_PRIVATE_KEYSIZE))`

dsa_domain_1024_160_t

dsa_domain_1024_160_t

Detailed description

Variables

- uint32_t q[(160/32)]

  DSA (1024,160) domain parameter Q.
- uint32_t p[(1024/32)]

  DSA (1024,160) domain parameter P.
- uint32_t g[(1024/32)]

  DSA (1024,160) domain parameter G.

dsa_signature_1024_160_t

dsa_signature_1024_160_t

Detailed description

Variables

- uint32_t r[(160/32)]

  DSA (1024,160) signature component R.

- uint32_t s[(160/32)]

  DSA (1024,160) signature component S.

dsa_domain_2048_224_t

dsa_domain_2048_224_t

Detailed description

Variables

- uint32_t q[(224/32)]

  DSA (2048,224) domain parameter Q.

- uint32_t p[(2048/32)]

  DSA (2048,224) domain parameter P.

- uint32_t g[(2048/32)]

  DSA (2048,224) domain parameter G.

dsa_signature_2048_224_t

dsa_signature_2048_224_t

Detailed description

Variables

- uint32_t r[(224/32)]

  DSA (2048,224) signature component R.

- uint32_t s[(224/32)]

  DSA (2048,224) signature component S.

dsa_domain_2048_256_t

dsa_domain_2048_256_t

Detailed description

Variables

- uint32_t q[(256/32)]

  DSA (2048,256) domain parameter Q.

- uint32_t p[(2048/32)]

  DSA (2048,256) domain parameter P.

- uint32_t g[(2048/32)]

  DSA (2048,256) domain parameter G.

dsa_signature_2048_256_t

dsa_signature_2048_256_t

Detailed description

Variables

- uint32_t r[(256/32)]

  DSA (2048,256) signature component R.

- uint32_t s[(256/32)]

  DSA (2048,256) signature component S.

s_sce_dsa_version

`ssp_version_t::s_sce_dsa_version`

Detailed description

Version data structure used by error logger macro.

g_dsa1024_160_on_sce

`const g_dsa1024_160_on_sce`

Detailed description

SCE/DSA implementation of DSA API.

Initialized as

```
g_dsa1024_160_on_sce=
{
    .open       = R_SCE_DSA_Open,
    .sign       = R_SCE_DSA_1024_160SignatureGenerate,
    .verify     = R_SCE_DSA_1024_160SignatureVerify,
    .close      = R_SCE_DSA_Close,
    .versionGet = R_SCE_DSA_VersionGet,
    .hashSign   = R_SCE_DSA_1024_160HashSignatureGenerate,
    .hashVerify = R_SCE_DSA_1024_160HashSignatureVerify
}
```

g_dsa2048_224_on_sce

`const g_dsa2048_224_on_sce`

Detailed description

SCE/DSA implementation of DSA API.

Initialized as

```
g_dsa2048_224_on_sce=
{
    .open       = R_SCE_DSA_Open,
    .sign       = R_SCE_DSA_2048_224SignatureGenerate,
    .verify     = R_SCE_DSA_2048_224SignatureVerify,
    .close      = R_SCE_DSA_Close,
    .versionGet = R_SCE_DSA_VersionGet,
    .hashSign   = R_SCE_DSA_2048_224HashSignatureGenerate,
    .hashVerify = R_SCE_DSA_2048_224HashSignatureVerify,
}
```

g_dsa2048_256_on_sce

```
const g_dsa2048_256_on_sce
```

Detailed description

SCE/DSA implementation of DSA API.

Initialized as

```
g_dsa2048_256_on_sce=
{
    .open       = R_SCE_DSA_Open,
    .sign       = R_SCE_DSA_2048_256SignatureGenerate,
    .verify     = R_SCE_DSA_2048_256SignatureVerify,
    .close      = R_SCE_DSA_Close,
    .versionGet = R_SCE_DSA_VersionGet,
    .hashSign   = R_SCE_DSA_2048_256HashSignatureGenerate,
    .hashVerify = R_SCE_DSA_2048_256HashSignatureVerify
}
```

R_SCE_DSA_Open

```
R_SCE_DSA_Open (  dsa_ctrl_t *const p_ctrl ,   dsa_cfg_t const *const p_cfg )
```

Detailed description

DSA Initialization

Table 2010:Return values

Name	Description
SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	The parameter p_ctrl or p_cfg is NULL.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE resource is busy
SSP_ERR_CRYPTO_SCE_FAIL	SCE internal I/O is not empty

R_SCE_DSA_Close

```
R_SCE_DSA_Close (  dsa_ctrl_t *const p_ctrl )
```

Detailed description

Close DSA driver

Table 2011:Return values

Name	Description
SF_CRYPTO_SUCCESS	Call successful

Table 2011:Return values (Continued)

Name	Description
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE resource is busy
SSP_ERR_CRYPTO_SCE_FAIL	SCE internal I/O is not empty

R_SCE_DSA_VersionGet

```
R_SCE_DSA_VersionGet ( ssp_version_t  *const p_version )
```

Brief description

Sets driver version based on compile time macros.

Detailed description

Table 2012:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

R_SCE_DSA_1024_160SignatureVerify

```
R_SCE_DSA_1024_160SignatureVerify (  const uint32_t * p_key ,   const uint32_t
* p_domain ,   uint32_t imaxcnt ,   uint32_t * p_signature ,   uint32_t
* p_paddedHash )
```

Brief description

Signature verification using (1024-bit,160-bit) DSA public key.

Detailed description

Verify DSA signature data from buffer p_signature of length 2*imaxcnt words using (L=1024,N=160) DSA public key from buffer p_key. The buffer p_paddedHash indicates the message buffer from which the DSA signature should have been generated.

Table 2013:Return values

Name	Description
SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Incorrect signature

Table 2013:Return values (Continued)

Name	Description
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the function R_SCE_Open API

NOTE: The p_key buffer must contain 32 words of DSA public key data

NOTE: The p_domain buffer must contain (20+128+128) bytes of data in the format (Q || P || G) where Q is 5 words, P is 32 words and G is 32 words.

R_SCE_DSA_1024_160SignatureGenerate

```
 R_SCE_DSA_1024_160SignatureGenerate (  const uint32_t * p_key ,   const uint32_t
* p_domain ,   uint32_t imaxcnt ,   uint32_t * p_source ,   uint32_t * p_dest )
```

Brief description

Signature generation using (1024-bit,160-bit) DSA private key.

Detailed description

Sign imaxcnt words of input data from buffer p_source using the (L=1024,N=160)-bit DSA private key from buffer p_key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast 2*imaxcnt words of data.

Table 2014:Return values

Name	Description
SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the function R_SCE_Open API

NOTE: p_dest must have space to hold at least 2*imaxcnt words of data.

NOTE: The p_key buffer must contain a valid DSA private key data and p_domain should contain DSA domain parameters in the order (Q || P || G) where Q is 5 words, P is 32 words and G is 32 words.

R_SCE_DSA_1024_160HashSignatureVerify

```
 R_SCE_DSA_1024_160HashSignatureVerify (  dsa_ctrl_t *const p_ctrl ,   const
uint32_t * p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_signature ,   uint32_t * p_paddedHash )
```

Brief description

Signature verification using (1024-bit,160-bit) DSA public key.

Detailed description

Verify DSA signature data from buffer p_signature of length 2*imaxcnt words using (L=1024,N=160) DSA public key from buffer p_key. The buffer p_paddedHash indicates the message buffer from which the DSA signature should have been generated.

Table 2015:Return values

Name	Description
SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Incorrect signature
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the function R_SCE_Open API

NOTE: The p_key buffer must contain 32 words of DSA public key data

NOTE: The p_domain buffer must contain (20+128+128) bytes of data in the format (Q || P || G) where Q is 5 words, P is 32 words and G is 32 words.

R_SCE_DSA_1024_160HashSignatureGenerate

```
 R_SCE_DSA_1024_160HashSignatureGenerate (  dsa_ctrl_t *const p_ctrl ,   const
uint32_t * p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Brief description

Signature generation using (1024-bit,160-bit) DSA private key.

Detailed description

Sign imaxcnt words of input data from buffer p_source using the (L=1024,N=160)-bit DSA private key from buffer p_key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast 2*imaxcnt words of data.

Table 2016:Return values

Name	Description
SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the function R_SCE_Open API

NOTE: p_dest must have space to hold at least 2*imaxcnt words of data.

NOTE: The p_key buffer must contain a valid DSA private key data and p_domain should contain DSA domain parameters in the order (Q || P || G) where Q is 5 words, P is 32 words and G is 32 words.

R_SCE_DSA_2048_224SignatureVerify

```
 R_SCE_DSA_2048_224SignatureVerify (  const uint32_t * p_key ,   const uint32_t
* p_domain ,   uint32_t imaxcnt ,   uint32_t * p_signature ,   uint32_t
* p_paddedHash )
```

Brief description

Signature verification using (2048-bit,224-bit) DSA public key.

Detailed description

Verify DSA signature data from buffer p_signature of length num_words using (L=2048,N=224) DSA public key. from buffer p_key. The buffer p_paddedHash indicates the message buffer from which the DSA signature should have been generated.

Table 2017:Return values

Name	Description
SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open API

NOTE: The p_key buffer must contain 128 bytes of DSA public key data in the format

NOTE: The p_domain buffer must contain (28+256+256) bytes of data in the format (Q || P || G).Verify DSA signature from buffer p_signature using the given DSA public key p_key with domain parameters from p_domain for the input message hash p_paddedHash

R_SCE_DSA_2048_224SignatureGenerate

```
 R_SCE_DSA_2048_224SignatureGenerate (  const uint32_t * p_key ,   const uint32_t
* p_domain ,   uint32_t imaxcnt ,   uint32_t * p_source ,   uint32_t * p_dest )
```

Brief description

Signature generation using (2048-bit,224-bit) DSA private key.

Detailed description

Sign imaxcnt words of input data from buffer p_source using the (L=2048,N=224)-bit DSA private key from buffer key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Table 2018:Return values

Name	Description
SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open API

NOTE: p_dest must have space to hold at least 2*imaxcnt words of data.

NOTE: The key buffer must contain 7 words of DSA private key data

NOTE: p_domain must contain valid DSA domain parameters

R_SCE_DSA_2048_224HashSignatureVerify

```
 R_SCE_DSA_2048_224HashSignatureVerify (  dsa_ctrl_t *const p_ctrl ,   const
uint32_t * p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_signature ,   uint32_t * p_paddedHash )
```

Brief description

Signature verification using (2048-bit,224-bit) DSA public key.

Detailed description

Verify DSA signature from buffer p_signature using the given DSA public key p_key with domain parameters from p_domain for the input message hash p_paddedHash

R_SCE_DSA_2048_224HashSignatureGenerate

```
 R_SCE_DSA_2048_224HashSignatureGenerate (  dsa_ctrl_t *const p_ctrl ,   const
uint32_t * p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Brief description

Signature generation using (2048-bit,224-bit) DSA private key.

Detailed description

Sign imaxcnt words of input data from buffer p_source using the (L=2048,N=224)-bit DSA private key from buffer key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Table 2019:Return values

Name	Description
SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open API

NOTE: p_dest must have space to hold at least 2*imaxcnt words of data.

NOTE: The key buffer must contain 7 words of DSA private key data

NOTE: p_domain must contain valid DSA domain parameters

R_SCE_DSA_2048_256SignatureVerify

```
 R_SCE_DSA_2048_256SignatureVerify (  const uint32_t * p_key ,   const uint32_t
* p_domain ,   uint32_t imaxcnt ,   uint32_t * p_signature ,   uint32_t
* p_paddedHash )
```

Brief description

Signature verification using (2048-bit,256-bit) DSA public key.

Detailed description

Verify DSA signature from buffer p_signature using the given DSA public key p_key with domain parameters from p_domain for the input message hash p_paddedHash

Verify DSA signature data from buffer p_signature of length num_words using (L=2048,N=256) DSA public key. from buffer p_key. The buffer p_paddedHash indicates the message buffer from which the DSA signature should have been generated.

Table 2020:Return values

Name	Description
SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: The p_key buffer must contain 256 bytes of DSA public key data

NOTE: The p_domain buffer must contain (32+256+256) bytes of data in the format (Q || P || G).

R_SCE_DSA_2048_256SignatureGenerate

```
 R_SCE_DSA_2048_256SignatureGenerate (  const uint32_t * p_key ,   const uint32_t
* p_domain ,   uint32_t imaxcnt ,   uint32_t * p_source ,   uint32_t * p_dest )
```

Brief description

Signature generation using (2048-bit,256-bit) DSA private key.

Detailed description

Generate signature for the buffer p_paddedHash with the given DSA private key p_key for the domain parameters p_domain. The result will be written to the buffer p_dest

Sign imaxcnt words of input data from buffer p_source using the (L=2048,N=256)-bit DSA private key from buffer key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Table 2021:Return values

Name	Description
SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.

Table 2021:Return values (Continued)

Name	Description
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least 2*imaxcnt words of data.

NOTE: The key buffer must contain 8 words of DSA private key data

R_SCE_DSA_2048_256HashSignatureVerify

```
 R_SCE_DSA_2048_256HashSignatureVerify (  dsa_ctrl_t *const p_ctrl ,   const
uint32_t * p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_signature ,   uint32_t * p_paddedHash )
```

Brief description

Signature verification using (2048-bit,256-bit) DSA public key.

Detailed description

Verify DSA signature from buffer p_signature using the given DSA public key p_key with domain parameters from p_domain for the input message hash p_paddedHash

Verify DSA signature data from buffer p_signature of length num_words using (L=2048,N=256) DSA public key. from buffer p_key. The buffer p_paddedHash indicates the message buffer from which the DSA signature should have been generated.

Table 2022:Return values

Name	Description
SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: The p_key buffer must contain 256 bytes of DSA public key data

NOTE: The p_domain buffer must contain (32+256+256) bytes of data in the format (Q || P || G).

R_SCE_DSA_2048_256HashSignatureGenerate

```
 R_SCE_DSA_2048_256HashSignatureGenerate (  dsa_ctrl_t *const p_ctrl ,   const
uint32_t * p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Brief description

Signature generation using (2048-bit,256-bit) DSA private key.

Detailed description

Generate signature for the buffer p_paddedHash with the given DSA private key p_key for the domain parameters p_domain. The result will be written to the buffer p_dest

Sign imaxcnt words of input data from buffer p_source using the (L=2048,N=256)-bit DSA private key from buffer key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Table 2023:Return values

Name	Description
SF_CRYPTO_SUCCESS	Call successful
SSP_ERR_ASSERTION	An input parameter is invalid.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least 2*imaxcnt words of data.

NOTE: The key buffer must contain 8 words of DSA private key data

9.55.8.5 SCE HASH

Primitive cryptographic functions.

HASH functions

message hasing/digest functions

Functions

- R_SCE_HASH_VersionGet
- R_SCE_MD5_Open
- R_SCE_MD5_Close
- R_SCE_MD5_UpdateHash
- R_SCE_MD5_HashUpdate
- R_SCE_SHA1_Open
- R_SCE_SHA1_Close
- R_SCE_SHA1_UpdateHash
- R_SCE_SHA1_HashUpdate
- R_SCE_SHA256_Open
- R_SCE_SHA256_Close
- R_SCE_SHA256_UpdateHash
- R_SCE_SHA256_HashUpdate

Variables

- s_sce_hash_version
- g_md5_hash_on_sce
- g_sha1_hash_on_sce
- g_sha256_hash_on_sce

s_sce_hash_version

`ssp_version_t::s_sce_hash_version`

Detailed description

SCE/AES implementation of HASH API. Version data structure used by error logger macro.

Initialized as

```
s_sce_hash_version=
{
    .api_version_major  = #define  HASH_API_VERSION_MAJOR,
    .api_version_minor  = HASH_API_VERSION_MINOR,
```

```
    .code_version_major = SCE_HASH_CODE_VERSION_MAJOR,
    .code_version_minor = SCE_HASH_CODE_VERSION_MINOR
}
```

g_md5_hash_on_sce

hash_api_t::g_md5_hash_on_sce

Detailed description

MD5 implementation of HASH API.

Initialized as

```
g_md5_hash_on_sce=
{
    .open       = R_SCE_MD5_Open,
    .updateHash = R_SCE_MD5_UpdateHash,
    .close      = R_SCE_MD5_Close,
    .versionGet = R_SCE_HASH_VersionGet,
    .hashUpdate = R_SCE_MD5_HashUpdate
}
```

g_sha1_hash_on_sce

hash_api_t::g_sha1_hash_on_sce

Detailed description

SHA1 implementation of HASH API.

Initialized as

```
g_sha1_hash_on_sce=
{
    .open       = R_SCE_SHA1_Open,
    .updateHash = R_SCE_SHA1_UpdateHash,
    .close      = R_SCE_SHA1_Close,
    .versionGet = R_SCE_HASH_VersionGet,
    .hashUpdate = R_SCE_SHA1_HashUpdate
}
```

g_sha256_hash_on_sce

hash_api_t::g_sha256_hash_on_sce

Detailed description

SHA256 implementation of HASH API.

Initialized as

```
g_sha256_hash_on_sce=
{
    .open       = R_SCE_SHA256_Open,
    .updateHash = R_SCE_SHA256_UpdateHash,
    .close      = R_SCE_SHA256_Close,
    .versionGet = R_SCE_HASH_VersionGet,
    .hashUpdate = R_SCE_SHA256_HashUpdate
}
```

R_SCE_HASH_VersionGet

```
R_SCE_HASH_VersionGet ( ssp_version_t  *const p_version )
```

Brief description

Sets driver version based on compile time macros.

Detailed description

SCE HASH Get Version

Table 2024:Parameters

Name	Direction	Description
p_version	in	pointer to ssp_version_t structure where the version info will be stored

Table 2025:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

R_SCE_MD5_Open

```
R_SCE_MD5_Open ( hash_ctrl_t  *const p_ctrl ,  hash_cfg_t  const *const p_cfg )
```

Detailed description

SCE MD5 open function using the SCE block

MD5 HASH open

Table 2026:Return values

Name	Description
SF_CRYPTO_SUCCESS	open successful
SSP_ERR_ASSERTION	if input parameters are NULL.

R_SCE_MD5_Close

```
R_SCE_MD5_Close ( hash_ctrl_t  *const p_ctrl )
```

Detailed description

HASH Initialization

Table 2027:Return values

Name	Description
SF_CRYPTO_SUCCESS	close successful SCE MD5 Close function
SF_CRYPTO_SUCCESS	- success.
SSP_ERR_ASSERTION	if input parameter is NULL.

R_SCE_MD5_UpdateHash

```
R_SCE_MD5_UpdateHash (  const uint32_t * p_msg ,   uint32_t num_words ,
 uint32_t * p_digest )
```

Detailed description

MD5 Update Hash function.

Compute the MD5 message digest for the given input message buffer p_msg of length num_words words. The length of the message buffer needs to be a multiple of 64 bytes. Generally the content of the message buffer are the padded value as given by Message||stopbit||zero padding||Message length.

The initial hash value given in buffer p_digest will be used and this buffer will be updated with the computed MD5 message digest value.

Table 2028:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

SSP_ERR_ASSERTION if input parameters are NULL

NOTE: : The final messsage digest has to be byte swapped / the order of the bytes have to be

NOTE: : reversed (the output of the intermediate updates do not have to be swapped).

NOTE: : The message length has to be byte swapped (the endianness has to be reversed).

R_SCE_MD5_HashUpdate

```
R_SCE_MD5_HashUpdate ( hash_ctrl_t  *const p_ctrl ,   const uint32_t * p_msg ,
 uint32_t num_words ,   uint32_t * p_digest )
```

Detailed description

MD5 HashUpdate Function

Compute the MD5 message digest for the given input message buffer p_msg of length num_words words. The length of the message buffer needs to be a multiple of 64 bytes. Generally the content of the message buffer are the padded value as given by Message||stopbit||zero padding||Message length.

The initial hash value given in buffer p_digest will be used and this buffer will be updated with the computed MD5 message digest value.

Table 2029:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured
SSP_ERR_ASSERTION	if input parameters are null.

NOTE: : The final messsage digest has to be byte swapped / the order of the bytes have to be

NOTE: : reversed (the output of the intermediate updates do not have to be swapped).

NOTE: : The message length has to be byte swapped (the endianness has to be reversed).

R_SCE_SHA1_Open

```
R_SCE_SHA1_Open ( hash_ctrl_t  *const p_ctrl ,  hash_cfg_t  const *const p_cfg )
```

Detailed description

SCE SHA1 open function using the SCE block

SHA1 HASH Initialization

Table 2030:Return values

Name	Description
SF_CRYPTO_SUCCESS	open successful

R_SCE_SHA1_Close

```
R_SCE_SHA1_Close ( hash_ctrl_t  *const p_ctrl )
```

Detailed description

HASH Initialization

Table 2031:Return values

Name	Description
SF_CRYPTO_SUCCESS	close successful SCE SHA1 Close function

Table 2032:Parameters

Name	Direction	Description
p_ctrl	in	control structure for SCE SHA1

R_SCE_SHA1_UpdateHash

```
R_SCE_SHA1_UpdateHash (  const uint32_t * p_msg ,   uint32_t num_words ,
 uint32_t * p_digest )
```

Detailed description

Compute the SHA1 message digest for the given input message buffer p_msg of length num_words words. The length of the message buffer needs to be a multiple of 64 bytes. Generally the content of the message buffer are the padded value as given by Message||stopbit||zero padding||Message length.

The initial hash value given in buffer p_digest will be used and this buffer will be updated with the computed SHA1 message digest value.

Table 2033:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

R_SCE_SHA1_HashUpdate

```
R_SCE_SHA1_HashUpdate ( hash_ctrl_t  *const p_ctrl ,   const uint32_t * p_msg ,
 uint32_t num_words ,   uint32_t * p_digest )
```

Detailed description

Compute the SHA1 message digest for the given input message buffer p_msg of length num_words words. The length of the message buffer needs to be a multiple of 64 bytes. Generally the content of the message buffer are the padded value as given by Message||stopbit||zero padding||Message length.

The initial hash value given in buffer p_digest will be used and this buffer will be updated with the computed SHA1 message digest value.

Table 2034:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

R_SCE_SHA256_Open

```
 R_SCE_SHA256_Open ( hash_ctrl_t  *const p_ctrl ,  hash_cfg_t  const
*const p_cfg )
```

Detailed description

SCE SHA256 HASH Initialization

Table 2035:Return values

Name	Description
SF_CRYPTO_SUCCESS	successful completion

R_SCE_SHA256_Close

```
 R_SCE_SHA256_Close ( hash_ctrl_t  *const p_ctrl )
```

Detailed description

SCE SHA256 Close function

Table 2036:Return values

Name	Description
SF_CRYPTO_SUCCESS	random numbergeneration successful

R_SCE_SHA256_UpdateHash

```
R_SCE_SHA256_UpdateHash (  const uint32_t * p_msg ,   uint32_t num_words ,
 uint32_t * p_digest )
```

Brief description

Update hash value using the given input message from buffer p_source of num_words words, write the result to p_digest

Detailed description

Compute the SHA256 message digest for the given input message buffer p_msg of length num_words words. The length of the message buffer needs to be a multiple of 64 bytes. Generally the contents of the message buffer are the padded value as given by Message||stopbit||zero padding||Message length.

The initial hash value as given in buffer p_digest will be used and this buffer will be updated with the computed SHA256 message digest value.

Table 2037:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

R_SCE_SHA256_HashUpdate

```
 R_SCE_SHA256_HashUpdate ( hash_ctrl_t  *const p_ctrl ,   const uint32_t
* p_msg ,   uint32_t num_words ,   uint32_t * p_digest )
```

Brief description

Update hash value using the given input message from buffer p_source of num_words words, write the result to p_digest

Detailed description

Compute the SHA256 message digest for the given input message buffer p_msg of length num_words words. The length of the message buffer needs to be a multiple of 64 bytes. Generally the contents of the message buffer are the padded value as given by Message||stopbit||zero padding||Message length.

The initial hash value as given in buffer p_digest will be used and this buffer will be updated with the computed SHA256 message digest value.

Table 2038:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

9.55.8.6 SCE_ECC

Primitive cryptographic functions.

ECC 192-bit implementation for scalar multiplication, key generation, signature generation and signature verification functions

ECC 256-bit implementation for scalar multiplication, key generation, signature generation and signature verification functions

Functions

- R_SCE_ECC_Open
- R_SCE_ECC_Close
- R_SCE_ECC_VersionGet
- r_sce_ecc_192_scalar_multiplication_buffer_validation
- r_sce_ecc_192_key_create_buffer_validation
- r_sce_ecc_192_private_key_sign_buffer_data_validation
- r_sce_ecc_192_private_key_sign_buffer_length_validation
- r_sce_ecc_192_public_key_verify_buffer_data_validation
- r_sce_ecc_192_public_key_verify_buffer_length_validation
- R_SCE_ECC_192ScalarMultiplication
- R_SCE_ECC_192KeyCreate
- R_SCE_ECC_192PrivateKeySign
- R_SCE_ECC_192PublicKeyVerify
- r_sce_hrk_ecc_192_scalar_multiplication_buffer_validation
- r_sce_hrk_ecc_192_key_create_buffer_validation
- r_sce_hrk_ecc_192_private_key_sign_buffer_data_validation
- r_sce_hrk_ecc_192_private_key_sign_buffer_length_validation
- R_SCE_HRK_ECC_192ScalarMultiplication
- R_SCE_HRK_ECC_192KeyCreate
- R_SCE_HRK_ECC_192PrivateKeySign
- r_sce_ecc_256_scalar_multiplication_buffer_validation
- r_sce_ecc_256_key_create_buffer_validation
- r_sce_ecc_256_private_key_sign_buffer_data_validation
- r_sce_ecc_256_private_key_sign_buffer_length_validation
- r_sce_ecc_256_public_key_verify_buffer_data_validation
- r_sce_ecc_256_public_key_verify_buffer_length_validation
- R_SCE_ECC_256ScalarMultiplication
- R_SCE_ECC_256KeyCreate
- R_SCE_ECC_256PrivateKeySign

- R_SCE_ECC_256PublicKeyVerify
- r_sce_hrk_ecc_256_scalar_multiplication_buffer_validation
- r_sce_hrk_ecc_256_key_create_buffer_validation
- r_sce_hrk_ecc_256_private_key_sign_buffer_data_validation
- r_sce_hrk_ecc_256_private_key_sign_buffer_length_validation
- R_SCE_HRK_ECC_256ScalarMultiplication
- R_SCE_HRK_ECC_256KeyCreate
- R_SCE_HRK_ECC_256PrivateKeySign

Variables

- g_ecc192_on_sce
- g_ecc192_on_sce_hrk
- g_ecc256_on_sce
- g_ecc256_on_sce_hrk

Defines

- #define SCE_ECC_CODE_VERSION_MAJOR

  Initial value: `(01U)`
- #define SCE_ECC_CODE_VERSION_MINOR

  Initial value: `(00U)`

g_ecc192_on_sce

```
const g_ecc192_on_sce
```

Detailed description

Exported global variablesSCE/RSA implementation of RSA API.

Initialized as

```
g_ecc192_on_sce=
{
    .open                   = R_SCE_ECC_Open,
    .close                  = R_SCE_ECC_Close,
    .scalarMultiplication   = R_SCE_ECC_192ScalarMultiplication,
    .keyCreate              = R_SCE_ECC_192KeyCreate,
    .sign                   = R_SCE_ECC_192PrivateKeySign,
    .verify                 = R_SCE_ECC_192PublicKeyVerify,
    .versionGet             = R_SCE_ECC_VersionGet
}
```

g_ecc192_on_sce_hrk

```
const g_ecc192_on_sce_hrk
```

Detailed description

Exported global variablesSCE/RSA implementation of RSA API.

Initialized as

```
g_ecc192_on_sce_hrk=
{
    .open                   = R_SCE_ECC_Open,
    .close                  = R_SCE_ECC_Close,
    .scalarMultiplication   = R_SCE_HRK_ECC_192ScalarMultiplication,
    .keyCreate              = R_SCE_HRK_ECC_192KeyCreate,
    .sign                   = R_SCE_HRK_ECC_192PrivateKeySign,
    .verify                 = R_SCE_ECC_192PublicKeyVerify,
    .versionGet             = R_SCE_ECC_VersionGet
}
```

g_ecc256_on_sce

```
const g_ecc256_on_sce
```

Detailed description

Exported global variablesSCE/RSA implementation of RSA API.

Initialized as

```
g_ecc256_on_sce=
{
    .open                   = R_SCE_ECC_Open,
    .close                  = R_SCE_ECC_Close,
    .scalarMultiplication   = R_SCE_ECC_256ScalarMultiplication,
    .keyCreate              = R_SCE_ECC_256KeyCreate,
    .sign                   = R_SCE_ECC_256PrivateKeySign,
    .verify                 = R_SCE_ECC_256PublicKeyVerify,
    .versionGet             = R_SCE_ECC_VersionGet
}
```

g_ecc256_on_sce_hrk

```
const g_ecc256_on_sce_hrk
```

Detailed description

Exported global variablesSCE/RSA implementation of RSA API.

Initialized as

```
g_ecc256_on_sce_hrk=
{
    .open                   = R_SCE_ECC_Open,
    .close                  = R_SCE_ECC_Close,
    .scalarMultiplication   = R_SCE_HRK_ECC_256ScalarMultiplication,
    .keyCreate              = R_SCE_HRK_ECC_256KeyCreate,
    .sign                   = R_SCE_HRK_ECC_256PrivateKeySign,
    .verify                 = R_SCE_ECC_256PublicKeyVerify,
    .versionGet             = R_SCE_ECC_VersionGet
}
```

R_SCE_ECC_Open

```
ssp_err_t R_SCE_ECC_Open ( ecc_ctrl_t *const p_ctrl , ecc_cfg_t const
*const p_cfg )
```

Brief description

ECC Initialization.

Detailed description

Table 2039:Return values

Name	Description
SSP_SUCCESS	Normal end
SSP_ERR_ASSERTION	Any of the input parameters are NULL.

R_SCE_ECC_Close

```
ssp_err_t R_SCE_ECC_Close ( ecc_ctrl_t *const p_ctrl )
```

Brief description

ECC Close function.

Detailed description

Table 2040:Return values

Name	Description
SSP_SUCCESS	Normal end
SSP_ERR_ASSERTION	The parameter p_ctrl is NULL.

R_SCE_ECC_VersionGet

```
ssp_err_t R_SCE_ECC_VersionGet ( ssp_version_t *const p_version )
```

Brief description

Sets driver version based on compile time macros.

Detailed description

Table 2041:Return values

Name	Description
SSP_SUCCESS	Normal end
SSP_ERR_ASSERTION	The parameter p_version is NULL.

r_sce_ecc_192_scalar_multiplication_buffer_validation

```
ssp_err_t r_sce_ecc_192_scalar_multiplication_buffer_validation (  r_crypto_data
_handle_t const * p_domain ,   r_crypto_data_handle_t const * p_k ,
  r_crypto_data_handle_t const * p_p ,   r_crypto_data_handle_t const * p_r )
```

Brief description

Validate scalar multiplication API buffers. Called by - R_SCE_ECC_192ScalarMultiplication.

Detailed description

Includes Private function prototypes

Table 2042:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.
p_k	in	Scalar.
p_p	in	Px||Py, where, Px and Py are x and y coordinates respectively.
p_r	in	Rx||Ry (R=kP), where, Rx and Ry are x and y coordinates respectively.

Table 2043:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).

r_sce_ecc_192_key_create_buffer_validation

```
ssp_err_t r_sce_ecc_192_key_create_buffer_validation (  r_crypto_data_handle_t
const * p_domain ,   r_crypto_data_handle_t const * p_generator_point ,
  r_crypto_data_handle_t const * p_key_private ,   r_crypto_data_handle_t const
* p_key_public )
```

Brief description

Validate key create API buffers. Called by - R_SCE_ECC_192KeyCreate.

Detailed description

Table 2044:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.
p_generator_point	in	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
p_key_private	in	Private Key generated.
p_key_public	in	Public Key generated.

Table 2045:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).

r_sce_ecc_192_private_key_sign_buffer_data_validation

```
ssp_err_t r_sce_ecc_192_private_key_sign_buffer_data_validation (  r_crypto_data
_handle_t const * p_domain ,   r_crypto_data_handle_t const * p_generator_point ,
  r_crypto_data_handle_t const * p_key_private ,   r_crypto_data_handle_t const
* msg_digest ,   r_crypto_data_handle_t const * signature_r ,
  r_crypto_data_handle_t const * signature_s )
```

Brief description

Validate private key sign API buffer data. Called by - R_SCE_ECC_192PrivateKeySign.

Detailed description

Table 2046:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.
p_generator_point	in	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
p_key_private	in	Private Key for signing.
msg_digest	in	Message Digest.
signature_r	in	Signature r generated.
signature_s	in	Signature s generated.

Table 2047:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).

r_sce_ecc_192_private_key_sign_buffer_length_validation

```
ssp_err_t r_sce_ecc_192_private_key_sign_buffer_length_validation (  r_crypto_da
ta_handle_t const * p_domain ,   r_crypto_data_handle_t const
* p_generator_point ,   r_crypto_data_handle_t const * p_key_private ,
  r_crypto_data_handle_t const * msg_digest ,   r_crypto_data_handle_t const
* signature_r ,   r_crypto_data_handle_t const * signature_s )
```

Brief description

Validate private key sign API buffer length. Called by - R_SCE_ECC_192PrivateKeySign.

Detailed description

Table 2048:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.
p_generator_point	in	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
p_key_private	in	Private Key for signing.
msg_digest	in	Message Digest.
signature_r	in	Signature r generated.
signature_s	in	Signature s generated.

Table 2049:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).

r_sce_ecc_192_public_key_verify_buffer_data_validation

```
ssp_err_t r_sce_ecc_192_public_key_verify_buffer_data_validation (  r_crypto_dat
a_handle_t const * p_domain ,   r_crypto_data_handle_t const
* p_generator_point ,   r_crypto_data_handle_t const * p_key_public ,
  r_crypto_data_handle_t const * msg_digest ,   r_crypto_data_handle_t const
* signature_r ,   r_crypto_data_handle_t const * signature_s )
```

Brief description

Validate public key verify API buffer data. Called by - R_SCE_ECC_192PublicKeyVerify.

Detailed description

Table 2050:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.
p_generator_point	in	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
p_key_public	in	Public Key for signature verification.
msg_digest	in	Message Digest.
signature_r	in	Signature r.
signature_s	in	Signature s.

Table 2051:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).

r_sce_ecc_192_public_key_verify_buffer_length_validation

```
ssp_err_t r_sce_ecc_192_public_key_verify_buffer_length_validation (  r_crypto_d
ata_handle_t const * p_domain ,   r_crypto_data_handle_t const
* p_generator_point ,   r_crypto_data_handle_t const * p_key_public ,
  r_crypto_data_handle_t const * msg_digest ,   r_crypto_data_handle_t const
* signature_r ,   r_crypto_data_handle_t const * signature_s )
```

Brief description

Validate public key verify API buffer length. Called by - R_SCE_ECC_192PublicKeyVerify.

Detailed description

Table 2052:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.
p_generator_point	in	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
p_key_public	in	Public Key for signature verification.
msg_digest	in	Message Digest.
signature_r	in	Signature r.
signature_s	in	Signature s.

Table 2053:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).

R_SCE_ECC_192ScalarMultiplication

```
ssp_err_t R_SCE_ECC_192ScalarMultiplication (  ecc_ctrl_t *const p_ctrl ,
  r_crypto_data_handle_t *const p_domain ,   r_crypto_data_handle_t *const p_k ,
  r_crypto_data_handle_t *const p_p ,   r_crypto_data_handle_t *const p_r )
```

Brief description

Perform scalar multiplication of a scalar pointed by p_k (k) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid.

Detailed description

Table 2054:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

NOTE: In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will be updated to the expected return buffer size irrespective of the return value of this API. The caller must check the return status before using the output data.

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_domain must be of size ECC_192_DOMAIN_PARAMETER_WITHOUT_ORDER_LENGTH_WORDS (18 words). (a||b||p - 6 words each)

NOTE: p_k must be of size ECC_192_PRIVATE_KEY_LENGTH_WORDS (6 words).

NOTE: p_p must be of size ECC_192_POINT_ON_CURVE_LENGTH_WORDS (12 words).

NOTE: p_r must have space to hold at least ECC_192_POINT_ON_CURVE_LENGTH_WORDS (12 words) of output data.

R_SCE_ECC_192KeyCreate

```
ssp_err_t R_SCE_ECC_192KeyCreate (  ecc_ctrl_t *const p_ctrl ,
  r_crypto_data_handle_t *const p_domain ,   r_crypto_data_handle_t
*const p_generator_point ,   r_crypto_data_handle_t *const p_key_private ,
  r_crypto_data_handle_t *const p_key_public )
```

Brief description

Generate public and private key pair for elliptic curve cryptography on 192 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Detailed description

Table 2055:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

NOTE: In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will be updated to the expected return buffer size irrespective of the return value of this API. The caller must check the return status before using the output data.

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_domain must be of size ECC_192_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (24 words). (a||b||p||n - 6 words each)

NOTE: p_generator_point must be of size ECC_192_GENERATOR_POINT_LENGTH_WORDS (12 words).

NOTE: p_key_private must have space to hold at least ECC_192_PRIVATE_KEY_LENGTH_WORDS (6 words) of output data.

NOTE: p_key_public must have space to hold at least ECC_192_PUBLIC_KEY_LENGTH_WORDS (12 words) of output data.

R_SCE_ECC_192PrivateKeySign

```
ssp_err_t R_SCE_ECC_192PrivateKeySign (  ecc_ctrl_t *const p_ctrl ,
  r_crypto_data_handle_t *const p_domain ,   r_crypto_data_handle_t
*const p_generator_point ,   r_crypto_data_handle_t *const p_key_private ,
  r_crypto_data_handle_t *const msg_digest ,   r_crypto_data_handle_t
*const signature_r ,   r_crypto_data_handle_t *const signature_s )
```

Brief description

Generate signature of ECDSA on 192 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Detailed description

Table 2056:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

NOTE: In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will be updated to the expected return buffer size irrespective of the return value of this API. The caller must check the return status before using the output data.

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_domain must be of size ECC_192_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (24 words). (a||b||p||n - 6 words each)

NOTE: p_generator_point must be of size ECC_192_GENERATOR_POINT_LENGTH_WORDS (12 words).

NOTE: p_key_private must be of size ECC_192_PRIVATE_KEY_LENGTH_WORDS (6 words).

NOTE: msg_digest must be of size ECC_192_MESSAGE_DIGEST_LENGTH_WORDS (6 words).

NOTE: signature_r must have space to hold at least ECC_192_SIGNATURE_R_LENGTH_WORDS (6 words) of output data.

NOTE: signature_s must have space to hold at least ECC_192_SIGNATURE_S_LENGTH_WORDS (6 words) of output data.

R_SCE_ECC_192PublicKeyVerify

```
ssp_err_t R_SCE_ECC_192PublicKeyVerify (  ecc_ctrl_t *const p_ctrl ,
  r_crypto_data_handle_t *const p_domain ,   r_crypto_data_handle_t
*const p_generator_point ,   r_crypto_data_handle_t *const p_key_public ,
  r_crypto_data_handle_t *const msg_digest ,   r_crypto_data_handle_t
*const signature_r ,   r_crypto_data_handle_t *const signature_s )
```

Brief description

Signature verification of ECDSA on 192 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Detailed description

Table 2057:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).

Table 2057:Return values (Continued)

Name	Description
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Verification failure.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_domain must be of size ECC_192_GENERATOR_POINT_LENGTH_WORDS (24 words). (a||b||p||n - 6 words each)

NOTE: p_generator_point must be of size ECC_192_GENERATOR_POINT_LENGTH_WORDS (12 words).

NOTE: p_key_public must be of size ECC_192_PUBLIC_KEY_LENGTH_WORDS (12 words).

NOTE: msg_digest must be of size ECC_192_MESSAGE_DIGEST_LENGTH_WORDS (6 words).

NOTE: signature_r must be of size ECC_192_SIGNATURE_R_LENGTH_WORDS (6 words).

NOTE: signature_s must be of size ECC_192_SIGNATURE_S_LENGTH_WORDS (6 words).

r_sce_hrk_ecc_192_scalar_multiplication_buffer_validation

```
ssp_err_t r_sce_hrk_ecc_192_scalar_multiplication_buffer_validation (  r_crypto_
data_handle_t const * p_domain ,   r_crypto_data_handle_t const * p_k ,
  r_crypto_data_handle_t const * p_p ,   r_crypto_data_handle_t const * p_r )
```

Brief description

Validate scalar multiplication API buffers. Called by - R_SCE_HRK_ECC_192ScalarMultiplication.

Detailed description

Includes Private function prototypes

Table 2058:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.
p_k	in	Scalar - Private key.
p_p	in	Px||Py, where, Px and Py are x and y coordinates respectively.
p_r	in	Rx||Ry (R=kP), where, Rx and Ry are x and y coordinates respectively.

Table 2059:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).

r_sce_hrk_ecc_192_key_create_buffer_validation

```
ssp_err_t r_sce_hrk_ecc_192_key_create_buffer_validation (  r_crypto_data_handle
_t const * p_domain ,   r_crypto_data_handle_t const * p_generator_point ,
  r_crypto_data_handle_t const * p_key_private ,   r_crypto_data_handle_t const
* p_key_public )
```

Brief description

Validate key create API buffers. Called by - R_SCE_HRK_ECC_192KeyCreate.

Detailed description

Table 2060:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.
p_generator_point	in	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
p_key_private	in	Private Key (wrapped) generated.
p_key_public	in	Public Key generated.

Table 2061:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).

r_sce_hrk_ecc_192_private_key_sign_buffer_data_validation

```
ssp_err_t r_sce_hrk_ecc_192_private_key_sign_buffer_data_validation (  r_crypto_
data_handle_t const * p_domain ,   r_crypto_data_handle_t const
* p_generator_point ,   r_crypto_data_handle_t const * p_key_private ,
  r_crypto_data_handle_t const * msg_digest ,   r_crypto_data_handle_t const
* signature_r ,   r_crypto_data_handle_t const * signature_s )
```

Brief description

Validate private key sign API data buffers. Called by - R_SCE_HRK_ECC_192PrivateKeySign.

Detailed description

Table 2062:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.
p_generator_point	in	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
p_key_private	in	Private Key (wrapped).
msg_digest	in	Message Digest.
signature_r	in	Signature r generated.
signature_s	in	Signature s generated.

Table 2063:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).

r_sce_hrk_ecc_192_private_key_sign_buffer_length_validation

```
ssp_err_t r_sce_hrk_ecc_192_private_key_sign_buffer_length_validation (  r_crypt
o_data_handle_t const * p_domain ,   r_crypto_data_handle_t const
* p_generator_point ,   r_crypto_data_handle_t const * p_key_private ,
  r_crypto_data_handle_t const * msg_digest ,   r_crypto_data_handle_t const
* signature_r ,   r_crypto_data_handle_t const * signature_s )
```

Brief description

Validate private key sign API buffer length. Called by - R_SCE_HRK_ECC_192PrivateKeySign.

Detailed description

Table 2064:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.
p_generator_point	in	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
p_key_private	in	Private Key (wrapped).
msg_digest	in	Message Digest.
signature_r	in	Signature r generated.
signature_s	in	Signature s generated.

Table 2065:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).

R_SCE_HRK_ECC_192ScalarMultiplication

```
ssp_err_t R_SCE_HRK_ECC_192ScalarMultiplication (  ecc_ctrl_t *const p_ctrl ,
  r_crypto_data_handle_t *const p_domain ,   r_crypto_data_handle_t
*const p_key_index ,   r_crypto_data_handle_t *const p_p ,
  r_crypto_data_handle_t *const p_r )
```

Brief description

Perform scalar multiplication of p_key_index (k) (wrapped key) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid.

Detailed description

Table 2066:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

NOTE: In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will be updated to the expected return buffer size irrespective of the return value of this API. The caller must check the return status before using the output data.

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_domain must be of size ECC_192_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (24 words). (a||b||p||n - 6 words each)

NOTE: p_key_index must be of size ECC_192_PRIVATE_KEY_HRK_LENGTH_WORDS (13 words).

NOTE: p_p must be of size ECC_192_GENERATOR_POINT_LENGTH_WORDS (12 words).

NOTE: p_r must have space to hold at least ECC_192_GENERATOR_POINT_LENGTH_WORDS (12 words) of output data.

NOTE: p_key_index is generated using generateKey API.

R_SCE_HRK_ECC_192KeyCreate

```
ssp_err_t R_SCE_HRK_ECC_192KeyCreate (  ecc_ctrl_t *const p_ctrl ,
  r_crypto_data_handle_t *const p_domain ,   r_crypto_data_handle_t
*const p_generator_point ,   r_crypto_data_handle_t *const p_key_index ,
  r_crypto_data_handle_t *const p_key_public )
```

Brief description

Generate public key and wrapped key for elliptic curve cryptography on 192 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key.

Detailed description

Table 2067:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

NOTE: In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will be updated to the expected return buffer size irrespective of the return value of this API. The caller must check the return status before using the output data.

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_domain must be of size ECC_192_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (24 words). (a||b||p||n - 6 words each)

NOTE: p_generator_point must be of size ECC_192_GENERATOR_POINT_LENGTH_WORDS (12 words).

NOTE: p_key_index must have space to hold at least ECC_192_PRIVATE_KEY_HRK_LENGTH_WORDS (13 words) of output data.

NOTE: p_key_public must have space to hold at least ECC_192_PUBLIC_KEY_LENGTH_WORDS (12 words) of output data.

R_SCE_HRK_ECC_192PrivateKeySign

```
ssp_err_t R_SCE_HRK_ECC_192PrivateKeySign (  ecc_ctrl_t *const p_ctrl ,
  r_crypto_data_handle_t *const p_domain ,   r_crypto_data_handle_t
*const p_generator_point ,   r_crypto_data_handle_t *const p_key_index ,
  r_crypto_data_handle_t *const msg_digest ,   r_crypto_data_handle_t
*const signature_r ,   r_crypto_data_handle_t *const signature_s )
```

Brief description

Generate signature of ECDSA on 192 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key.

Detailed description

Table 2068:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

NOTE: In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will be updated to the expected return buffer size irrespective of the return value of this API. The caller must check the return status before using the output data.

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_domain must be of size ECC_192_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (24 words). (a||b||p||n - 6 words each)

NOTE: p_generator_point must be of size ECC_192_GENERATOR_POINT_LENGTH_WORDS (12 words).

NOTE: p_key_index must be of size ECC_192_PRIVATE_KEY_HRK_LENGTH_WORDS (13 words).

NOTE: msg_digest must be of size ECC_192_MESSAGE_DIGEST_LENGTH_WORDS (6 words).

NOTE: signature_r must have space to hold at least ECC_192_SIGNATURE_R_LENGTH_WORDS (6 words) of output data.

NOTE: signature_s must have space to hold at least ECC_192_SIGNATURE_R_LENGTH_WORDS (6 words) of output data.

r_sce_ecc_256_scalar_multiplication_buffer_validation

```
ssp_err_t r_sce_ecc_256_scalar_multiplication_buffer_validation (  r_crypto_data
_handle_t const * p_domain ,   r_crypto_data_handle_t const * p_k ,
  r_crypto_data_handle_t const * p_p ,   r_crypto_data_handle_t const * p_r )
```

Brief description

Validate scalar multiplication API buffers. Called by - R_SCE_ECC_256ScalarMultiplication.

Detailed description

Includes Private function prototypes

Table 2069:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.
p_k	in	Scalar.
p_p	in	Px||Py, where, Px and Py are x and y coordinates respectively.
p_r	in	Rx||Ry (R=kP), where, Rx and Ry are x and y coordinates respectively.

Table 2070:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).

r_sce_ecc_256_key_create_buffer_validation

```
ssp_err_t r_sce_ecc_256_key_create_buffer_validation (  r_crypto_data_handle_t
const * p_domain ,   r_crypto_data_handle_t const * p_generator_point ,
  r_crypto_data_handle_t const * p_key_private ,   r_crypto_data_handle_t const
* p_key_public )
```

Brief description

Validate key create API buffers. Called by - R_SCE_ECC_256KeyCreate.

Detailed description

Table 2071:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.

Table 2071:Parameters (Continued)

Name	Direction	Description
p_generator_point	in	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
p_key_private	in	Private Key generated.
p_key_public	in	Public Key generated.

Table 2072:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).

r_sce_ecc_256_private_key_sign_buffer_data_validation

```
ssp_err_t r_sce_ecc_256_private_key_sign_buffer_data_validation (  r_crypto_data
_handle_t const * p_domain ,   r_crypto_data_handle_t const * p_generator_point ,
  r_crypto_data_handle_t const * p_key_private ,   r_crypto_data_handle_t const
* msg_digest ,   r_crypto_data_handle_t const * signature_r ,
  r_crypto_data_handle_t const * signature_s )
```

Brief description

Validate private key sign API buffer data. Called by - R_SCE_ECC_256PrivateKeySign.

Detailed description

Table 2073:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.

Table 2073:Parameters (Continued)

Name	Direction	Description
p_generator_point	in	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
p_key_private	in	Private Key for signing.
msg_digest	in	Message Digest.
signature_r	in	Signature r generated.
signature_s	in	Signature s generated.

Table 2074:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).

r_sce_ecc_256_private_key_sign_buffer_length_validation

```
ssp_err_t r_sce_ecc_256_private_key_sign_buffer_length_validation (  r_crypto_da
ta_handle_t const * p_domain ,   r_crypto_data_handle_t const
* p_generator_point ,   r_crypto_data_handle_t const * p_key_private ,
  r_crypto_data_handle_t const * msg_digest ,   r_crypto_data_handle_t const
* signature_r ,   r_crypto_data_handle_t const * signature_s )
```

Brief description

Validate private key sign API buffer length. Called by - R_SCE_ECC_256PrivateKeySign.

Detailed description

Table 2075:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.

Table 2075:Parameters (Continued)

Name	Direction	Description
p_generator_point	in	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
p_key_private	in	Private Key for signing.
msg_digest	in	Message Digest.
signature_r	in	Signature r generated.
signature_s	in	Signature s generated.

Table 2076:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).

r_sce_ecc_256_public_key_verify_buffer_data_validation

```
ssp_err_t r_sce_ecc_256_public_key_verify_buffer_data_validation (  r_crypto_dat
a_handle_t const * p_domain ,   r_crypto_data_handle_t const
* p_generator_point ,   r_crypto_data_handle_t const * p_key_public ,
  r_crypto_data_handle_t const * msg_digest ,   r_crypto_data_handle_t const
* signature_r ,   r_crypto_data_handle_t const * signature_s )
```

Brief description

Validate public key verify API buffer data. Called by - R_SCE_ECC_256PublicKeyVerify.

Detailed description

Table 2077:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.

Table 2077:Parameters (Continued)

Name	Direction	Description
p_generator_point	in	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
p_key_public	in	Public Key for signature verification.
msg_digest	in	Message Digest.
signature_r	in	Signature r.
signature_s	in	Signature s.

Table 2078:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).

r_sce_ecc_256_public_key_verify_buffer_length_validation

```
ssp_err_t r_sce_ecc_256_public_key_verify_buffer_length_validation (  r_crypto_d
ata_handle_t const * p_domain ,   r_crypto_data_handle_t const
* p_generator_point ,   r_crypto_data_handle_t const * p_key_public ,
  r_crypto_data_handle_t const * msg_digest ,   r_crypto_data_handle_t const
* signature_r ,   r_crypto_data_handle_t const * signature_s )
```

Brief description

Validate public key verify API buffer length. Called by - R_SCE_ECC_256PublicKeyVerify.

Detailed description

Table 2079:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.

Table 2079:Parameters (Continued)

Name	Direction	Description
p_generator_point	in	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
p_key_public	in	Public Key for signature verification.
msg_digest	in	Message Digest.
signature_r	in	Signature r.
signature_s	in	Signature s.

Table 2080:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).

R_SCE_ECC_256ScalarMultiplication

```
ssp_err_t R_SCE_ECC_256ScalarMultiplication (  ecc_ctrl_t *const p_ctrl ,
  r_crypto_data_handle_t *const p_domain ,   r_crypto_data_handle_t *const p_k ,
  r_crypto_data_handle_t *const p_p ,   r_crypto_data_handle_t *const p_r )
```

Brief description

Perform scalar multiplication of a scalar pointed by p_k (k) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid.

Detailed description

Table 2081:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.

Table 2081:Return values (Continued)

Name	Description
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

NOTE: In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will be updated to the expected return buffer size irrespective of the return value of this API. The caller must check the return status before using the output data.

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_domain must be of size ECC_256_DOMAIN_PARAMETER_WITHOUT_ORDER_LENGTH_WORDS (24 words). (a||b||p - 8 words each)

NOTE: p_k must be of size ECC_256_PRIVATE_KEY_LENGTH_WORDS (8 words).

NOTE: p_p must be of size ECC_256_POINT_ON_CURVE_LENGTH_WORDS (16 words).

NOTE: p_r must have space to hold at least ECC_256_POINT_ON_CURVE_LENGTH_WORDS (16 words) of output data.

R_SCE_ECC_256KeyCreate

```
ssp_err_t R_SCE_ECC_256KeyCreate (  ecc_ctrl_t *const p_ctrl ,
  r_crypto_data_handle_t *const p_domain ,   r_crypto_data_handle_t
*const p_generator_point ,   r_crypto_data_handle_t *const p_key_private ,
  r_crypto_data_handle_t *const p_key_public )
```

Brief description

Generate public and private key pair for elliptic curve cryptography on 256 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Detailed description

Table 2082:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

NOTE: In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will be updated to the expected return buffer size irrespective of the return value of this API. The caller must check the return status before using the output data.

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_domain must be of size ECC_256_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (32 words). (a||b||p||n - 8 words each)

NOTE: p_generator_point must be of size ECC_256_GENERATOR_POINT_LENGTH_WORDS (16 words).

NOTE: p_key_private must have space to hold at least ECC_256_PRIVATE_KEY_LENGTH_WORDS (8 words) of output data.

NOTE: p_key_public must have space to hold at least ECC_256_PUBLIC_KEY_LENGTH_WORDS (16 words) of output data.

R_SCE_ECC_256PrivateKeySign

```
ssp_err_t R_SCE_ECC_256PrivateKeySign (  ecc_ctrl_t *const p_ctrl ,
  r_crypto_data_handle_t *const p_domain ,   r_crypto_data_handle_t
*const p_generator_point ,   r_crypto_data_handle_t *const p_key_private ,
  r_crypto_data_handle_t *const msg_digest ,   r_crypto_data_handle_t
*const signature_r ,   r_crypto_data_handle_t *const signature_s )
```

Brief description

Generate signature of ECDSA on 256 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Detailed description

Table 2083:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

NOTE: In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will be updated to the expected return buffer size irrespective of the return value of this API. The caller must check the return status before using the output data.

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_domain must be of size ECC_256_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (32 words). (a||b||p||n - 8 words each)

NOTE: p_generator_point must be of size ECC_256_GENERATOR_POINT_LENGTH_WORDS (16 words).

NOTE: p_key_private must be of size ECC_256_PRIVATE_KEY_LENGTH_WORDS (8 words).

NOTE: msg_digest must be of size ECC_256_MESSAGE_DIGEST_LENGTH_WORDS (8 words).

NOTE: signature_r must have space to hold at least ECC_256_SIGNATURE_R_LENGTH_WORDS (8 words) of output data.

NOTE: signature_s must have space to hold at least ECC_256_SIGNATURE_S_LENGTH_WORDS (8 words) of output data.

R_SCE_ECC_256PublicKeyVerify

```
ssp_err_t R_SCE_ECC_256PublicKeyVerify (  ecc_ctrl_t *const p_ctrl ,
  r_crypto_data_handle_t *const p_domain ,   r_crypto_data_handle_t
*const p_generator_point ,   r_crypto_data_handle_t *const p_key_public ,
  r_crypto_data_handle_t *const msg_digest ,   r_crypto_data_handle_t
*const signature_r ,   r_crypto_data_handle_t *const signature_s )
```

Brief description

Signature verification of ECDSA on 256 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid.

Detailed description

Table 2084:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Verification failure.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_domain must be of size ECC_256_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (32 words). (a||b||p||n - 8 words each)

NOTE: p_generator_point must be of size ECC_256_GENERATOR_POINT_LENGTH_WORDS (16 words).

NOTE: p_key_public must be of size ECC_256_PUBLIC_KEY_LENGTH_WORDS (16 words).

NOTE: msg_digest must be of size ECC_256_MESSAGE_DIGEST_LENGTH_WORDS (8 words).

NOTE: signature_r must be of size ECC_256_SIGNATURE_R_LENGTH_WORDS (8 words).

NOTE: signature_s must be of size ECC_256_SIGNATURE_S_LENGTH_WORDS (8 words).

r_sce_hrk_ecc_256_scalar_multiplication_buffer_validation

```
ssp_err_t r_sce_hrk_ecc_256_scalar_multiplication_buffer_validation (  r_crypto_
data_handle_t const * p_domain ,   r_crypto_data_handle_t const * p_k ,
  r_crypto_data_handle_t const * p_p ,   r_crypto_data_handle_t const * p_r )
```

Brief description

Validate scalar multiplication API buffers. Called by - R_SCE_HRK_ECC_256ScalarMultiplication.

Detailed description

Includes Private function prototypes

Table 2085:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.
p_k	in	Scalar - Private key.
p_p	in	Px||Py, where, Px and Py are x and y coordinates respectively.
p_r	in	Rx||Ry (R=kP), where, Rx and Ry are x and y coordinates respectively.

Table 2086:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).

r_sce_hrk_ecc_256_key_create_buffer_validation

```
ssp_err_t r_sce_hrk_ecc_256_key_create_buffer_validation (  r_crypto_data_handle
_t const * p_domain ,   r_crypto_data_handle_t const * p_generator_point ,
  r_crypto_data_handle_t const * p_key_private ,   r_crypto_data_handle_t const
* p_key_public )
```

Brief description

Validate key create API buffers. Called by - R_SCE_HRK_ECC_256KeyCreate.

Detailed description

Table 2087:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.

Table 2087:Parameters (Continued)

Name	Direction	Description
p_generator_point	in	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
p_key_private	in	Private Key (wrapped) generated.
p_key_public	in	Public Key generated.

Table 2088:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).

r_sce_hrk_ecc_256_private_key_sign_buffer_data_validation

```
ssp_err_t r_sce_hrk_ecc_256_private_key_sign_buffer_data_validation (  r_crypto_
data_handle_t const * p_domain ,   r_crypto_data_handle_t const
* p_generator_point ,   r_crypto_data_handle_t const * p_key_private ,
  r_crypto_data_handle_t const * msg_digest ,   r_crypto_data_handle_t const
* signature_r ,   r_crypto_data_handle_t const * signature_s )
```

Brief description

Validate private key sign API data buffers. Called by - R_SCE_HRK_ECC_256PrivateKeySign.

Detailed description

Table 2089:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.

Table 2089:Parameters (Continued)

Name	Direction	Description
p_generator_point	in	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
p_key_private	in	Private Key (wrapped).
msg_digest	in	Message Digest.
signature_r	in	Signature r generated.
signature_s	in	Signature s generated.

Table 2090:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).

r_sce_hrk_ecc_256_private_key_sign_buffer_length_validation

```
ssp_err_t r_sce_hrk_ecc_256_private_key_sign_buffer_length_validation (  r_crypt
o_data_handle_t const * p_domain ,   r_crypto_data_handle_t const
* p_generator_point ,   r_crypto_data_handle_t const * p_key_private ,
  r_crypto_data_handle_t const * msg_digest ,   r_crypto_data_handle_t const
* signature_r ,   r_crypto_data_handle_t const * signature_s )
```

Brief description

Validate private key sign API buffer length. Called by - R_SCE_HRK_ECC_256PrivateKeySign.

Detailed description

Table 2091:Parameters

Name	Direction	Description
p_domain	in	a||b||p - These are domain parameters for ECC as defined in IEEE1363.

Table 2091:Parameters (Continued)

Name	Direction	Description
p_generator_point	in	Gx||Gy - Base point of the curve, where, Gx and Gy are x and y coordinates respectively. This parameter is one of the domain parameters.
p_key_private	in	Private Key (wrapped).
msg_digest	in	Message Digest.
signature_r	in	Signature r generated.
signature_s	in	Signature s generated.

Table 2092:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).

R_SCE_HRK_ECC_256ScalarMultiplication

```
ssp_err_t R_SCE_HRK_ECC_256ScalarMultiplication (  ecc_ctrl_t *const p_ctrl ,
  r_crypto_data_handle_t *const p_domain ,   r_crypto_data_handle_t
*const p_key_index ,   r_crypto_data_handle_t *const p_p ,
  r_crypto_data_handle_t *const p_r )
```

Brief description

Perform scalar multiplication of p_key_index (k) (wrapped key) with a point pointed by p_p (P) and return the result at location pointed by p_r (R=kP). Ensure the domain parameter, p_domain, exists and is valid.

Detailed description

Table 2093:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).

Table 2093:Return values (Continued)

Name	Description
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

NOTE: In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will be updated to the expected return buffer size irrespective of the return value of this API. The caller must check the return status before using the output data.

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_domain must be of size ECC_256_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (32 words). (a||b||p||n - 8 words each)

NOTE: p_key_index must be of size ECC_256_PRIVATE_KEY_HRK_LENGTH_WORDS (13 words).

NOTE: p_p must be of size ECC_256_POINT_ON_CURVE_LENGTH_WORDS (16 words).

NOTE: p_r must have space to hold at least ECC_256_POINT_ON_CURVE_LENGTH_WORDS (16 words) of output data.

NOTE: p_key_index is generated using generateKey API.

R_SCE_HRK_ECC_256KeyCreate

```
ssp_err_t R_SCE_HRK_ECC_256KeyCreate (  ecc_ctrl_t *const p_ctrl ,
  r_crypto_data_handle_t *const p_domain ,   r_crypto_data_handle_t
*const p_generator_point ,   r_crypto_data_handle_t *const p_key_index ,
  r_crypto_data_handle_t *const p_key_public )
```

Brief description

Generate public key and wrapped key for elliptic curve cryptography on 256 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key.

Detailed description

Table 2094:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

NOTE: In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will be updated to the expected return buffer size irrespective of the return value of this API. The caller must check the return status before using the output data.

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_domain must be of size ECC_256_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (32 words). (a||b||p||n - 8 words each)

NOTE: p_generator_point must be of size ECC_256_GENERATOR_POINT_LENGTH_WORDS (16 words).

NOTE: p_key_index must have space to hold at least ECC_256_PRIVATE_KEY_HRK_LENGTH_WORDS (13 words) of output data.

NOTE: p_key_public must have space to hold at least ECC_256_PUBLIC_KEY_LENGTH_WORDS (16 words) of output data.

R_SCE_HRK_ECC_256PrivateKeySign

```
ssp_err_t R_SCE_HRK_ECC_256PrivateKeySign (  ecc_ctrl_t *const p_ctrl ,
  r_crypto_data_handle_t *const p_domain ,   r_crypto_data_handle_t
*const p_generator_point ,   r_crypto_data_handle_t *const p_key_index ,
  r_crypto_data_handle_t *const msg_digest ,   r_crypto_data_handle_t
*const signature_r ,   r_crypto_data_handle_t *const signature_s )
```

Brief description

Generate signature of ECDSA on 256 bit of prime field. Ensure the domain parameter, p_domain, exists and is valid. Ensure the p_generator point, generator_point, exists and is valid. p_key_index is the wrapped key.

Detailed description

Table 2095:Return values

Name	Description
SSP_SUCCESS	Normal end.
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_INVALID_SIZE	invalid input buffer length(s).
SSP_ERR_CRYPTO_SCE_FAIL	Internal Error.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred.

NOTE: In case of failure, output buffer(s) may get partially updated. Data length field of output data handle(s) will be updated to the expected return buffer size irrespective of the return value of this API. The caller must check the return status before using the output data.

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_domain must be of size ECC_256_DOMAIN_PARAMETER_WITH_ORDER_LENGTH_WORDS (32 words). (a||b||p||n - 8 words each)

NOTE: p_generator_point must be of size ECC_256_GENERATOR_POINT_LENGTH_WORDS (16 words).

NOTE: p_key_index must be of size ECC_256_PRIVATE_KEY_HRK_LENGTH_WORDS (13 words).

NOTE: msg_digest must be of size ECC_256_MESSAGE_DIGEST_LENGTH_WORDS (8 words).

NOTE: signature_r must have space to hold at least ECC_256_SIGNATURE_S_LENGTH_WORDS (8 words) of output data.

NOTE: signature_s must have space to hold at least ECC_256_SIGNATURE_S_LENGTH_WORDS (8 words) of output data.

9.55.8.7 SCE_KEY_INSTALLATION

Primitive cryptographic functions.

key installation functions

Functions

- R_SCE_Key_Installation_Open
- R_SCE_Key_Installation_VersionGet
- R_SCE_Key_Installation_Close
- R_SCE_Key_Installation_KeyInstall

Variables

- s_sce_key_installation_version
- g_key_installation_on_sce

Defines

- #define SCE_KEY_INSTALLATION_CODE_VERSION_MAJOR

  Initial value: `(01U)`
- #define SCE_KEY_INSTALLATION_CODE_VERSION_MINOR

  Initial value: `(00U)`

key_installation_instance_ctrl_t

key_installation_instance_ctrl_t

Detailed description

Key Installation Interface control structure

Variables

- crypto_ctrl_t * p_crypto_ctrl

  pointer to crypto engine control structure

- crypto_api_t const * p_crypto_api

  pointer to crypto engine API

s_sce_key_installation_version

```
ssp_version_t::s_sce_key_installation_version
```

Detailed description

Version data structure used by error logger macro.

g_key_installation_on_sce

```
const g_key_installation_on_sce
```

Detailed description

SCE/Key_Installation API implementation.

Initialized as

```
g_key_installation_on_sce=
{
    .open            = R_SCE_Key_Installation_Open,
    .close           = R_SCE_Key_Installation_Close,
    .versionGet      = R_SCE_Key_Installation_VersionGet,
    .keyInstall      = R_SCE_Key_Installation_KeyInstall
}
```

R_SCE_Key_Installation_Open

```
ssp_err_t R_SCE_Key_Installation_Open (  key_installation_ctrl_t *const p_ctrl ,
  key_installation_cfg_t const *const p_cfg )
```

Brief description

Key Installation module Open API function.

Detailed description

Table 2096:Parameters

Name	Direction	Description
p_ctrl	inout	Pointer to Key Installation control block.

Table 2096:Parameters (Continued)

Name	Direction	Description
p_cfg	in	Pointer to Key Installation configuration structure.

Table 2097:Return values

Name	Description
SSP_SUCCESS	Normal end
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE resource is busy
SSP_ERR_CRYPTO_SCE_FAIL	SCE internal I/O is not empty

NOTE: This API must be called only after r_sce module open API call.

R_SCE_Key_Installation_VersionGet

```
ssp_err_t R_SCE_Key_Installation_VersionGet ( ssp_version_t  *const p_version )
```

Brief description

Gets Key Installation Driver API and Code version based on compile time macros.

Detailed description

Table 2098:Parameters

Name	Direction	Description
p_version	inout	Pointer to API version structure.

Table 2099:Return values

Name	Description
SSP_SUCCESS	Successful in getting the module version.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

R_SCE_Key_Installation_Close

```
ssp_err_t R_SCE_Key_Installation_Close (  key_installation_ctrl_t
*const p_ctrl )
```

Brief description

Key Installation module Close API function.

Detailed description

Table 2100:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to Key Installation control block.

Table 2101:Return values

Name	Description
SSP_SUCCESS	Normal end
SSP_ERR_ASSERTION	NULL input parameter(s).
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE resource is busy
SSP_ERR_CRYPTO_SCE_FAIL	SCE internal I/O is not empty

NOTE: This API must be called once Key Installation API call is over, then followed by r_sce close API call.

R_SCE_Key_Installation_KeyInstall

```
ssp_err_t R_SCE_Key_Installation_KeyInstall (  key_installation_ctrl_t
*const p_ctrl ,   key_installation_key_t const *const p_user_key ,
  key_installation_key_t const *const p_install_key ,   key_installation_key_t
*const p_key_data )
```

Brief description

Key Installation API function that converts the encrypted install key into the data available for Key operations.

Detailed description

Table 2102:Parameters

Name	Direction	Description
p_ctrl	in	Pointer to Key Installation control block.
p_user_key	in	Pointer to a user provided AES encrypted key.
p_install_key	in	Pointer to Renesas provided encrypted install key.
p_key_data	inout	Pointer to installed wrapped key.

Table 2103:Return values

Name	Description
SSP_SUCCESS	Successful Key install.
SSP_ERR_ASSERTION	The parameters p_ctrl or p_user_key or p_install_key or p_key_data is NULL.
SSP_ERR_UNSUPPORTED	Input key formats are not supported.
SSP_ERR_CRYPTO_INVALID_SIZE	Input buffer length (user provided encrypted key or Renesas install key or Output buffer lengths are invalid).

9.55.8.8 SCE_RSA

Primitive cryptographic functions.

RSA 1024-bit implementation for key generation, encryption, decryption, signature generation and signature verification functions for wrapped keys.

RSA 2048-bit implementation for key generation, encryption, decryption, signature generation and signature verification functions for wrapped keys.

RSA 1024-bit implementation for key generation, encryption, decryption, signature generation and signature verification functions for plain-text/ raw keys.

RSA 2048-bit implementation for key generation, encryption, decryption, signature generation and signature verification functions for plain-text/ raw keys.

RSA common functions

Functions

- R_SCE_HRK_RSA_1024PublicKeyEncrypt

- R_SCE_HRK_RSA_1024PublicKeyVerify
- R_SCE_HRK_RSA_1024PrivateKeyDecrypt
- R_SCE_HRK_RSA_1024PrivateKeySign
- R_SCE_HRK_RSA_1024PrivateCrtKeyDecrypt
- R_SCE_HRK_RSA_1024PrivateCrtKeySign
- R_SCE_HRK_RSA_1024KeyCreate
- R_SCE_HRK_RSA_2048PublicKeyEncrypt
- R_SCE_HRK_RSA_2048PublicKeyVerify
- R_SCE_HRK_RSA_2048PrivateKeyDecrypt
- R_SCE_HRK_RSA_2048PrivateKeySign
- R_SCE_HRK_RSA_2048PrivateCrtKeyDecrypt
- R_SCE_HRK_RSA_2048PrivateCrtKeySign
- R_SCE_HRK_RSA_2048KeyCreate
- R_SCE_RSA_Open
- R_SCE_RSA_Close
- R_SCE_RSA_VersionGet
- R_SCE_RSA_1024PublicKeyEncrypt
- R_SCE_RSA_1024PublicKeyVerify
- R_SCE_RSA_1024PrivateKeyDecrypt
- R_SCE_RSA_1024PrivateKeySign
- R_SCE_RSA_1024PrivateCrtKeyDecrypt
- R_SCE_RSA_1024PrivateCrtKeySign
- R_SCE_RSA_1024KeyCreate
- R_SCE_RSA_2048PublicKeyEncrypt
- R_SCE_RSA_2048PublicKeyVerify
- R_SCE_RSA_2048PrivateKeyDecrypt
- R_SCE_RSA_2048PrivateKeySign
- R_SCE_RSA_2048PrivateCrtKeyDecrypt
- R_SCE_RSA_2048PrivateCrtKeySign
- R_SCE_RSA_2048KeyCreate

Variables

- g_rsa1024_on_sce_hrk
- sce_hrk_rsa_table_1024
- g_rsa2048_on_sce_hrk

- sce_hrk_rsa_table_2048
- s_sce_rsa_version
- g_rsa1024_on_sce
- rsa_table_1024
- g_rsa2048_on_sce
- rsa_table_2048

Defines

- #define HASH_SIZE_BYTES

  Initial value: `((1024 / 8))`
- #define RSA_KEY_LENGTH

  Initial value:`1024U`
- #define INITIAL_STAGE_NUM

  Initial value: `(0)`
- #define RSA_NUM_TRIES

  Initial value: `(5120)`
- #define RSA_KEYLENGTH

  Initial value: `(1024U)`

  RSA Keylength in bits
- #define UINT32_BITS

  Initial value: `(32)`

  uint32_t word size in bits
- #define UINT32_BYTES

  Initial value: `(4)`

  uint32_t word size in bytes
- #define RSA_MODULUS_SIZE

  Initial value: `((#define RSA_KEYLENGTH)/(#define UINT32_BITS))`

  rsa modulus data size in words
- #define RSA_PRIVATE_EXPONENT_SIZE

  Initial value: `(#define RSA_MODULUS_SIZE)`

  rsa private exponent data size in words
- #define RSA_PUBLIC_EXPONENT_SIZE

  Initial value: `(1)`

  rsa public exponent data size in words

- #define RSA_PUBLIC_KEYSIZE

  Initial value: ((#define RSA_PUBLIC_EXPONENT_SIZE)+(#define RSA_MODULUS_SIZE))

  rsa public key size in words
- #define RSA_PRIVATE_KEYSIZE

  Initial value: ((#define RSA_PRIVATE_EXPONENT_SIZE)+(#define RSA_MODULUS_SIZE))

  rsa private key size in words
- #define RSA_PRIVATE_CRT_KEYSIZE

  Initial value: ((5*(#define RSA_MODULUS_SIZE))/2)

  RSA private key in CRT format size in words

g_rsa1024_on_sce_hrk

```
const g_rsa1024_on_sce_hrk
```

Detailed description

SCE/RSA implementation of RSA API.

Initialized as

```
g_rsa1024_on_sce_hrk=
{
    .open       = R_SCE_RSA_Open,
    .encrypt    = R_SCE_HRK_RSA_1024PublicKeyEncrypt,
    .decrypt    = R_SCE_HRK_RSA_1024PrivateKeyDecrypt,
    .decryptCrt = R_SCE_HRK_RSA_1024PrivateCrtKeyDecrypt,
    .sign       = R_SCE_HRK_RSA_1024PrivateKeySign,
    .signCrt    = R_SCE_HRK_RSA_1024PrivateCrtKeySign,
    .verify     = R_SCE_HRK_RSA_1024PublicKeyVerify,
    .close      = R_SCE_RSA_Close,
    .versionGet = R_SCE_RSA_VersionGet,
    .keyCreate  = R_SCE_HRK_RSA_1024KeyCreate
}
```

sce_hrk_rsa_table_1024

```
const sce_hrk_rsa_table_1024
```

g_rsa2048_on_sce_hrk

```
const g_rsa2048_on_sce_hrk
```

Detailed description

SCE/RSA implementation of RSA API.

Initialized as

```
g_rsa2048_on_sce_hrk=
{
    .open       = R_SCE_RSA_Open,
    .encrypt    = R_SCE_HRK_RSA_2048PublicKeyEncrypt,
    .decrypt    = R_SCE_HRK_RSA_2048PrivateKeyDecrypt,
    .decryptCrt = R_SCE_HRK_RSA_2048PrivateCrtKeyDecrypt,
```

```
    .sign       = R_SCE_HRK_RSA_2048PrivateKeySign,
    .signCrt    = R_SCE_HRK_RSA_2048PrivateCrtKeySign,
    .verify     = R_SCE_HRK_RSA_2048PublicKeyVerify,
    .close      = R_SCE_RSA_Close,
    .versionGet = R_SCE_RSA_VersionGet,
    .keyCreate  = R_SCE_HRK_RSA_2048KeyCreate
}
```

sce_hrk_rsa_table_2048

```
const sce_hrk_rsa_table_2048
```

s_sce_rsa_version

```
ssp_version_t::s_sce_rsa_version
```

Detailed description

Version data structure used by error logger macro.

g_rsa1024_on_sce

```
const g_rsa1024_on_sce
```

Detailed description

SCE/RSA implementation of RSA API.

Initialized as

```
g_rsa1024_on_sce=
{
    .open       = R_SCE_RSA_Open,
    .encrypt    = R_SCE_RSA_1024PublicKeyEncrypt,
    .decrypt    = R_SCE_RSA_1024PrivateKeyDecrypt,
    .decryptCrt = R_SCE_RSA_1024PrivateCrtKeyDecrypt,
    .sign       = R_SCE_RSA_1024PrivateKeySign,
    .signCrt    = R_SCE_RSA_1024PrivateCrtKeySign,
    .verify     = R_SCE_RSA_1024PublicKeyVerify,
    .close      = R_SCE_RSA_Close,
    .versionGet = R_SCE_RSA_VersionGet,
    .keyCreate  = R_SCE_RSA_1024KeyCreate

}
```

rsa_table_1024

```
const rsa_table_1024
```

Initialized as

```
rsa_table_1024=
{
    .keylength = #define  RSA_KEYLENGTH,
    .keygen    = HW_SCE_RSA_1024KeyGenerate
}
```

g_rsa2048_on_sce

```
const g_rsa2048_on_sce
```

Initialized as

```
g_rsa2048_on_sce=
{
    .open       = R_SCE_RSA_Open,
    .encrypt    = R_SCE_RSA_2048PublicKeyEncrypt,
    .decrypt    = R_SCE_RSA_2048PrivateKeyDecrypt,
    .decryptCrt = R_SCE_RSA_2048PrivateCrtKeyDecrypt,
    .sign       = R_SCE_RSA_2048PrivateKeySign,
    .signCrt    = R_SCE_RSA_2048PrivateCrtKeySign,
    .verify     = R_SCE_RSA_2048PublicKeyVerify,
    .close      = R_SCE_RSA_Close,
    .versionGet = R_SCE_RSA_VersionGet,
    .keyCreate  = R_SCE_RSA_2048KeyCreate
}
```

rsa_table_2048

```
const rsa_table_2048
```

Initialized as

```
rsa_table_2048=
{
    .keylength = #define  RSA_KEYLENGTH,
    .keygen    = HW_SCE_RSA_2048KeyGenerate
}
```

R_SCE_HRK_RSA_1024PublicKeyEncrypt

```
 R_SCE_HRK_RSA_1024PublicKeyEncrypt (  rsa_ctrl_t *const p_ctrl ,   const
uint32_t * p_key ,   const uint32_t * p_domain ,   uint32_t num_words ,
  uint32_t * p_source ,   uint32_t * p_dest )
```

R_SCE_HRK_RSA_1024PublicKeyVerify

```
 R_SCE_HRK_RSA_1024PublicKeyVerify (  rsa_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   const uint32_t * p_domain ,   uint32_t num_words ,   uint32_t
* p_signature ,   uint32_t * p_padded_hash )
```

R_SCE_HRK_RSA_1024PrivateKeyDecrypt

```
 R_SCE_HRK_RSA_1024PrivateKeyDecrypt (  rsa_ctrl_t *const p_ctrl ,   const
uint32_t * p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

R_SCE_HRK_RSA_1024PrivateKeySign

```
 R_SCE_HRK_RSA_1024PrivateKeySign (  rsa_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

R_SCE_HRK_RSA_1024PrivateCrtKeyDecrypt

```
 R_SCE_HRK_RSA_1024PrivateCrtKeyDecrypt (  rsa_ctrl_t *const p_ctrl ,   const
uint32_t * p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

R_SCE_HRK_RSA_1024PrivateCrtKeySign

```
 R_SCE_HRK_RSA_1024PrivateCrtKeySign (  rsa_ctrl_t *const p_ctrl ,   const
uint32_t * p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

R_SCE_HRK_RSA_1024KeyCreate

```
 R_SCE_HRK_RSA_1024KeyCreate (  rsa_ctrl_t *const p_ctrl ,   rsa_key_t
* p_private_key ,   rsa_key_t * p_public_key )
```

Brief description

Creation of 1024-bit RSA Key pair in with private key in wrapped format.

Detailed description

RSA private key is created based on the key_format specified by p_private_key.
RSA_KEY_FORMAT_WRAPPED_PRIVATE_KEY specifies a wrapped standard format key.

Table 2104:Return values

Name	Description
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Key creation was successful.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer may not empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.
SSP_ERR_CRYPTO_INVALID_SIZE	An input parameter of invalid size.
SSP_ERR_CRYPTO_UNKNOWN	An input parameter is of unknown type.

NOTE: The buffers to hold the keys must be adequate for the key formats requested.

R_SCE_HRK_RSA_2048PublicKeyEncrypt

```
 R_SCE_HRK_RSA_2048PublicKeyEncrypt (  rsa_ctrl_t *const p_ctrl ,   const
uint32_t * p_key ,   const uint32_t * p_domain ,   uint32_t num_words ,
  uint32_t * p_source ,   uint32_t * p_dest )
```

Brief description

Encrypt using 2048-bit RSA public key.

Detailed description

Encrypt num_words words of input data from buffer p_source using the 2048-bit RSA public key from buffer p_key and domain parameters from p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 2105:Return values

Name	Description
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_Open()

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 260 bytes of RSA public key data in the format (public_exponent || public_modulus), where public_exponent is 1 words of data and public_modulus is 64 words of data

R_SCE_HRK_RSA_2048PublicKeyVerify

```
 R_SCE_HRK_RSA_2048PublicKeyVerify (  rsa_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   const uint32_t * p_domain ,   uint32_t num_words ,   uint32_t
* p_signature ,   uint32_t * p_padded_hash )
```

Brief description

Signature Verification using 2048-bit RSA public key.

Detailed description

Verify RSA signature data from buffer p_signature of length num_words using 2048-bit RSA public key. The buffer p_padded_hash indicates the message buffer from which the RSA signature should have been generated.

Table 2106:Return values

Name	Description
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Normal end, valid signature
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	incorrect signature

Table 2106:Return values (Continued)

Name	Description
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: The p_key buffer must contain 260 bytes of RSA public key data in the format (public_exponent || public_modulus) where public_exponent is 1 word and public_modulus is 64 words.

NOTE: Length of the p_key, p_signature and p_padded_hash must each be equal to num_words. For RSA 2048-bit key, num_words should be equal to 64.

R_SCE_HRK_RSA_2048PrivateKeyDecrypt

```
 R_SCE_HRK_RSA_2048PrivateKeyDecrypt (  rsa_ctrl_t *const p_ctrl ,   const
uint32_t * p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Brief description

Decrypt using 2048-bit RSA private key in wrapped format.

Detailed description

Decrypt imaxcnt words of input data from buffer p_source using the 2048-bit RSA private key from buffer p_key. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Table 2107:Return values

Name	Description
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Decrypt operation was successful.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least imaxcnt words of data.

NOTE: The p_key buffer must contain 64 words of RSA private key data followed by 64 words of RSA key modulus data

R_SCE_HRK_RSA_2048PrivateKeySign

```
 R_SCE_HRK_RSA_2048PrivateKeySign (  rsa_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Brief description

Signature generation using 2048-bit RSA private key in wrapped format.

Detailed description

Sign imaxcnt words of input data from buffer p_source using the 2048-bit RSA private key from buffer p_key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Table 2108:Return values

Name	Description
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Signature generation was successful.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least imaxcnt words of data.

NOTE: The p_key buffer must contain 64 words of RSA private key data followed by 64 words of RSA key modulus data

R_SCE_HRK_RSA_2048PrivateCrtKeyDecrypt

```
R_SCE_HRK_RSA_2048PrivateCrtKeyDecrypt ( rsa_ctrl_t *const p_ctrl , const
uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t
* p_source , uint32_t * p_dest )
```

Brief description

Decrypt using 2048-bit RSA private key in wrapped CRT format.

Detailed description

Wrapped CRT format is not supported at present. Parameter check is not performed as the function only returns SSP_ERR_CRYPTO_NOT_IMPLEMENTED.

Table 2109:Return values

Name	Description
SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.

R_SCE_HRK_RSA_2048PrivateCrtKeySign

```
R_SCE_HRK_RSA_2048PrivateCrtKeySign ( rsa_ctrl_t *const p_ctrl , const
uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t
* p_source , uint32_t * p_dest )
```

Brief description

Signature generation using 2048-bit RSA private key represented in wrapped CRT format.

Detailed description

Wrapped CRT format is not supported at present. Parameter check is not performed as the function only returns SSP_ERR_CRYPTO_NOT_IMPLEMENTED.

Table 2110:Return values

Name	Description
SSP_ERR_CRYPTO_NOT_IMPLEMENTED	This function is not implemented.

R_SCE_HRK_RSA_2048KeyCreate

```
R_SCE_HRK_RSA_2048KeyCreate ( rsa_ctrl_t *const p_ctrl , rsa_key_t
* p_private_key , rsa_key_t * p_public_key )
```

Brief description

Creation of 2048-bit RSA Key pair in with private key in wrapped format.

Detailed description

RSA private key is created based on the key_format specified by p_private_key.
RSA_KEY_FORMAT_WRAPPED_PRIVATE_KEY specifies a wrapped standard format key.

Table 2111:Return values

Name	Description
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Key creation was successful.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer may not empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.
SSP_ERR_CRYPTO_INVALID_SIZE	An input parameter of invalid size.
SSP_ERR_CRYPTO_UNKNOWN	An input parameter is of unknown type.

NOTE: The buffers to hold the keys must be adequate for the key formats requested.

R_SCE_RSA_Open

```
R_SCE_RSA_Open (  rsa_ctrl_t *const p_ctrl ,   rsa_cfg_t const *const p_cfg )
```

Detailed description

RSA Initialization

Table 2112:Return values

Name	Description
SF_CRYPTO_SUCCESS	random number generation successful
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE resource is busy
SSP_ERR_CRYPTO_SCE_FAIL	SCE internal I/O is not empty

R_SCE_RSA_Close

```
R_SCE_RSA_Close (  rsa_ctrl_t *const p_ctrl )
```

Detailed description

Close RSA driver

Table 2113:Return values

Name	Description
SF_CRYPTO_SUCCESS	random numbergeneration successful
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	SCE resource is busy
SSP_ERR_CRYPTO_SCE_FAIL	SCE internal I/O is not empty

R_SCE_RSA_VersionGet

```
R_SCE_RSA_VersionGet ( ssp_version_t  *const p_version )
```

Brief description

Sets driver version based on compile time macros.

Detailed description

Table 2114:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

R_SCE_RSA_1024PublicKeyEncrypt

```
R_SCE_RSA_1024PublicKeyEncrypt (  rsa_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   const uint32_t * p_domain ,   uint32_t num_words ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Brief description

Encryption using 1024-bit RSA public key.

Detailed description

Encrypt num_words words of input data from buffer p_source using the 1024-bit RSA public key from buffer p_key and domain parameters from p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 2115:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty

Table 2115:Return values (Continued)

Name	Description
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_Open()

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The p_key buffer must contain 132 bytes of RSA public key data in the format (public_exponent || public_modulus), where public_exponent is 1 words of data and public_modulus is 32 words of data

NOTE: num_words must be equal to RSA MODULUS size (RSA Public Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit Public key = 1024/32 = 32 words.

R_SCE_RSA_1024PublicKeyVerify

```
 R_SCE_RSA_1024PublicKeyVerify (  rsa_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   const uint32_t * p_domain ,   uint32_t num_words ,   uint32_t
* p_signature ,   uint32_t * p_padded_hash )
```

Brief description

Signature Verification using 1024-bit RSA public key.

Detailed description

Verify RSA signature data from buffer p_signature of length num_words using 1024-bit RSA public key. The buffer p_padded_hash indicates the message buffer from which the RSA signature should have been generated.

Table 2116:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end, valid signature
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	incorrect signature
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty

Table 2116:Return values (Continued)

Name	Description
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: The p_key buffer must contain 132 bytes of RSA public key data in the format (public_exponent || public_modulus) where public_exponent is 1 word and public_modulus is 32 words.

NOTE: Length of the p_key, p_signature and p_padded_hash must each be equal to num_words. For RSA 1024-bit key, num_words should be equal to 32.

NOTE: num_words must be equal to RSA MODULUS size (RSA Public Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit Pubic key = 1024/32 = 32 words.

R_SCE_RSA_1024PrivateKeyDecrypt

```
 R_SCE_RSA_1024PrivateKeyDecrypt (  rsa_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Brief description

Decrypt using 1024-bit RSA private key (standard format)

Detailed description

Decrypt imaxcnt words of input data from buffer p_source using the 1024-bit RSA private key from buffer p_key. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Table 2117:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty

Table 2117:Return values (Continued)

Name	Description
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least imaxcnt words of data.

NOTE: The p_key buffer must contain 32 words of RSA private key data followed by 32 words of RSA key modulus data

NOTE: imaxcnt must be equal to RSA MODULUS size (RSA Private Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit private key = 1024/32 = 32 words.

R_SCE_RSA_1024PrivateKeySign

```
 R_SCE_RSA_1024PrivateKeySign (  rsa_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Brief description

Signature generation using 1024-bit RSA private key.

Detailed description

Sign imaxcnt words of input data from buffer p_source using the 1024-bit RSA private key from buffer p_key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Table 2118:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least imaxcnt words of data.

NOTE: The p_key buffer must contain 32 words of RSA private key data followed by 32 words of RSA key modulus data

NOTE: imaxcnt must be equal to RSA MODULUS size (RSA Private Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit private key = 1024/32 = 32 words.

R_SCE_RSA_1024PrivateCrtKeyDecrypt

```
R_SCE_RSA_1024PrivateCrtKeyDecrypt ( rsa_ctrl_t *const p_ctrl , const
uint32_t * p_key , const uint32_t * p_domain , uint32_t imaxcnt , uint32_t
* p_source , uint32_t * p_dest )
```

Brief description

Decrypt using 1024-bit RSA private key (CRT format)

Detailed description

Decrypt imaxcnt words of input data from buffer p_source using the 1024-bit RSA private key from buffer key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Table 2119:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized R_SCE_Open

NOTE: p_dest must have space to hold at least imaxcnt words of data.

NOTE: The p_key pointer to 80-word buffer with 1024-bit RSA key CRT parameters (d mod (q-1) || q || d mod (p-1) || p || q^-1 mod p)

NOTE: imaxcnt must be equal to RSA MODULUS size (RSA Private Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit private key = 1024/32 = 32 words.

R_SCE_RSA_1024PrivateCrtKeySign

```
 R_SCE_RSA_1024PrivateCrtKeySign (  rsa_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Brief description

Signature generation using 1024-bit RSA private key represented in CRT format.

Detailed description

Sign imaxcnt words of input data from buffer p_source using the 1024-bit RSA private key from buffer p_key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Table 2120:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least imaxcnt words of data.

NOTE: The p_key pointer to 80-word buffer with 1024-bit RSA key CRT parameters (d mod (q-1) || q || d mod (p-1) || p || q^-1 mod p)

NOTE: imaxcnt must be equal to RSA MODULUS size (RSA Private Key_size / number of bits per word). For example, RSA MODULUS size for RSA 1024-bit private key = 1024/32 = 32 words.

R_SCE_RSA_1024KeyCreate

```
 R_SCE_RSA_1024KeyCreate (  rsa_ctrl_t *const p_ctrl ,   rsa_key_t
* p_private_key ,   rsa_key_t * p_public_key )
```

Brief description

Creation of 1024-bit RSA Key pair in plain text format.

Detailed description

RSA private key is created based on the key_format specified by p_private_key.
RSA_KEY_FORMAT_PLAIN_TEXT_PRIVATE_KEY specifies a standard format private key.
RSA_KEY_FORMAT_PLAIN_TEXT_CRT_KEY specifies the CRT parameters to be created.

Table 2121:Return values

Name	Description
SSP_ERR_ASSERTION	An input parameter may be NULL or of invalid format.
SF_CRYPTO_SUCCESS	Key creation was successful.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer may not empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.
SSP_ERR_CRYPTO_INVALID_SIZE	An input parameter of invalid size.
SSP_ERR_CRYPTO_UNKNOWN	An input parameter is of unknown type.

NOTE: The buffers to hold the keys must be adequate for the key formats requested.

NOTE: The RSA CRT key consists of the exponent2 || prime2 || exponent1 || prime1 || coefficient, in that order.

R_SCE_RSA_2048PublicKeyEncrypt

```
 R_SCE_RSA_2048PublicKeyEncrypt (  rsa_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Brief description

Encrypt using 2048-bit RSA public key.

Detailed description

Encrypt imaxcnt words of input data from buffer p_source using the 2048-bit RSA public key from buffer p_key and domain parameters from p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Table 2122:Return values

Name	Description
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_Open()

NOTE: p_dest must have space to hold at least imaxcnt words of data.

NOTE: The p_key buffer must contain 260 bytes of RSA public key data in the format (public_exponent || public_modulus), where public_exponent is 1 words of data and public_modulus is 64 words of data

NOTE: imaxcnt must be equal to RSA MODULUS size (RSA Public Key_size / number of bits per word). For example, RSA MODULUS size for RSA 2048-bit Public key = 2048/32 = 64 words.

R_SCE_RSA_2048PublicKeyVerify

```
 R_SCE_RSA_2048PublicKeyVerify (  rsa_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_signature ,   uint32_t * p_paddedHash )
```

Brief description

Signature Verification using 2048-bit RSA public key.

Detailed description

Verify RSA signature data from buffer p_signature of length imaxcnt using 2048-bit RSA public key. The buffer p_padded_hash indicates the message buffer from which the RSA signature should have been generated.

Table 2123:Return values

Name	Description
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Normal end, valid signature
SSP_ERR_CRYPTO_SCE_VERIFY_FAIL	incorrect signature
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: The p_key buffer must contain 260 bytes of RSA public key data in the format (public_exponent || public_modulus) where public_exponent is 1 word and public_modulus is 64 words.

NOTE: Length of the p_key, p_signature and p_padded_hash must each be equal to num_words. For RSA 2048-bit key, num_words should be equal to 64.

NOTE: imaxcnt must be equal to RSA MODULUS size (RSA Public Key_size / number of bits per word). For example, RSA MODULUS size for RSA 2048-bit Public key = 2048/32 = 64 words.

R_SCE_RSA_2048PrivateKeyDecrypt

```
 R_SCE_RSA_2048PrivateKeyDecrypt (  rsa_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Brief description

Decrypt using 2048-bit RSA private key (standard format)

Detailed description

Decrypt imaxcnt words of input data from buffer p_source using the 2048-bit RSA private key from buffer p_key. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Table 2124:Return values

Name	Description
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least imaxcnt words of data.

NOTE: The p_key buffer must contain 64 words of RSA private key data followed by 64 words of RSA key modulus data

NOTE: imaxcnt must be equal to RSA MODULUS size (RSA Key private size / number of bits per word). For example, RSA MODULUS size for RSA 2048-bit private key = 2048/32 = 64 words.

R_SCE_RSA_2048PrivateKeySign

```
 R_SCE_RSA_2048PrivateKeySign (  rsa_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Brief description

Signature generation using 2048-bit RSA private key.

Detailed description

Sign imaxcnt words of input data from buffer p_source using the 2048-bit RSA private key from buffer p_key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Table 2125:Return values

Name	Description
SSP_ERR_ASSERTION	An input parameter is NULL or of invalid format.
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least imaxcnt words of data.

NOTE: The p_key buffer must contain 64 words of RSA private key data followed by 64 words of RSA key modulus data

NOTE: imaxcnt must be equal to RSA MODULUS size (RSA Key private size / number of bits per word). For example, RSA MODULUS size for RSA 2048-bit private key = 2048/32 = 64 words.

R_SCE_RSA_2048PrivateCrtKeyDecrypt

```
 R_SCE_RSA_2048PrivateCrtKeyDecrypt (  rsa_ctrl_t *const p_ctrl ,   const
uint32_t * p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Brief description

Decrypt using 2048-bit RSA private key (CRT format)

Detailed description

Decrypt imaxcnt words of input data from buffer p_source using the 2048-bit RSA private key from buffer p_key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Table 2126:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized R_SCE_Open

NOTE: p_dest must have space to hold at least imaxcnt words of data.

NOTE: The p_key buffer must contain pointer to 160-word buffer with 2048-bit RSA key CRT parameters (d mod (q-1) || q || d mod (p-1) || p || q^-1 mod p)

NOTE: imaxcnt must be equal to RSA MODULUS size (RSA Key private size / number of bits per word). For example, RSA MODULUS size for RSA 2048-bit private key = 2048/32 = 64 words.

R_SCE_RSA_2048PrivateCrtKeySign

```
 R_SCE_RSA_2048PrivateCrtKeySign (  rsa_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   const uint32_t * p_domain ,   uint32_t imaxcnt ,   uint32_t
* p_source ,   uint32_t * p_dest )
```

Brief description

Signature generation using 2048-bit RSA private key represented in CRT format.

Detailed description

Sign imaxcnt words of input data from buffer p_source using the 2048-bit RSA private key from buffer p_key and domain parameters from buffer p_domain. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast imaxcnt words of data.

Table 2127:Return values

Name	Description
SF_CRYPTO_SUCCESS	Normal end
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized by calling the functions R_SCE_Open

NOTE: p_dest must have space to hold at least imaxcnt words of data.

NOTE: The p_key buffer must contain pointer to 160-word buffer with 2048-bit RSA key CRT parameters (d mod (q-1) || q || d mod (p-1) || p || q^-1 mod p)

NOTE: imaxcnt must be equal to RSA MODULUS size (RSA Key private size / number of bits per word). For example, RSA MODULUS size for RSA 2048-bit private key = 2048/32 = 64 words.

R_SCE_RSA_2048KeyCreate

```
 R_SCE_RSA_2048KeyCreate (  rsa_ctrl_t *const p_ctrl ,   rsa_key_t
* p_private_key ,   rsa_key_t * p_public_key )
```

Brief description

Creation of 2048-bit RSA Key pair in plain text format.

Detailed description

RSA private key is created based on the key_format specified by p_private_key.
RSA_KEY_FORMAT_PLAIN_TEXT_PRIVATE_KEY specifies a standard format private key.
RSA_KEY_FORMAT_PLAIN_TEXT_CRT_KEY specifies the CRT parameters to be created.

Table 2128:Return values

Name	Description
SSP_ERR_ASSERTION	An input parameter may be NULL or of invalid format.

Table 2128:Return values (Continued)

Name	Description
SF_CRYPTO_SUCCESS	Key creation was successful.
SSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer may not empty.
SSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Resource conflict occurred.
SSP_ERR_CRYPTO_INVALID_SIZE	An input parameter of invalid size.
SSP_ERR_CRYPTO_UNKNOWN	An input parameter is of unknown type.

NOTE: The buffers to hold the keys must be adequate for the key formats requested.

NOTE: The RSA CRT key consists of the exponent2 || prime2 || exponent1 || prime1 || coefficient, in that order.

9.55.8.9 SCE_TDES

Primitive cryptographic functions.

Triple DES encryption and decryption functions

Functions

- R_SCE_TDES_Open
- R_SCE_TDES_Close
- R_SCE_TDES_VersionGet
- R_SCE_TDES_192CbcEncrypt
- R_SCE_TDES_192CbcDecrypt
- R_SCE_TDES_192CtrEncrypt
- R_SCE_TDES_192EcbEncrypt
- R_SCE_TDES_192EcbDecrypt

Variables

- g_tdes192ecb_on_sce
- g_tdes192cbc_on_sce
- g_tdes192ctr_on_sce
- s_sce_tdes_version

g_tdes192ecb_on_sce

```
const g_tdes192ecb_on_sce
```

Detailed description

SCE/TDES implementation of TDES API.

Initialized as

```
g_tdes192ecb_on_sce=
{
    .open       = R_SCE_TDES_Open,
    .encrypt    = R_SCE_TDES_192EcbEncrypt,
    .decrypt    = R_SCE_TDES_192EcbDecrypt,
    .close      = R_SCE_TDES_Close,
    .versionGet = R_SCE_TDES_VersionGet
}
```

g_tdes192cbc_on_sce

```
const g_tdes192cbc_on_sce
```

Initialized as

```
g_tdes192cbc_on_sce=
{
    .open       = R_SCE_TDES_Open,
    .encrypt    = R_SCE_TDES_192CbcEncrypt,
    .decrypt    = R_SCE_TDES_192CbcDecrypt,
    .close      = R_SCE_TDES_Close,
    .versionGet = R_SCE_TDES_VersionGet
}
```

g_tdes192ctr_on_sce

```
const g_tdes192ctr_on_sce
```

Initialized as

```
g_tdes192ctr_on_sce=
{
    .open       = R_SCE_TDES_Open,
    .encrypt    = R_SCE_TDES_192CtrEncrypt,
    .decrypt    = R_SCE_TDES_192CtrEncrypt,
    .close      = R_SCE_TDES_Close,
    .versionGet = R_SCE_TDES_VersionGet
}
```

s_sce_tdes_version

```
ssp_version_t::s_sce_tdes_version
```

Detailed description

Version data structure used by error logger macro.

R_SCE_TDES_Open

```
 R_SCE_TDES_Open (  tdes_ctrl_t *const p_ctrl ,   tdes_cfg_t const *const p_cfg )
```

Detailed description

TDES Initialization

Table 2129:Return values

Name	Description
SF_CRYPTO_SUCCESS	random number generation successful
SF_CRPYTO_ERR_CRYPTO_RESOURCE_CONFLICT	SCE resource is busy
SF_CRYPTO_ERR_CRYPTO_SCEFAIL	SCE internal I/O is not empty

R_SCE_TDES_Close

```
R_SCE_TDES_Close (  tdes_ctrl_t *const p_ctrl )
```

Detailed description

Close TDES

Table 2130:Return values

Name	Description
SF_CRYPTO_SUCCESS	random numbergeneration successful
SF_CRPYTO_ERR_CRYPTO_RESOURCE_CONFLICT	SCE resource is busy
SF_CRYPTO_ERR_CRYPTO_SCEFAIL	SCE internal I/O is not empty

R_SCE_TDES_VersionGet

```
R_SCE_TDES_VersionGet ( ssp_version_t  *const p_version )
```

Brief description

Sets driver version based on compile time macros.

Detailed description

Table 2131:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

R_SCE_TDES_192CbcEncrypt

```
 R_SCE_TDES_192CbcEncrypt (  tdes_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Brief description

Triple DES Encryption with CBC mode.

Detailed description

Encrypt num_words words of input data from buffer p_source using the 192-bit TDES key from buffer key and initialization vector from buffer iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 2132:Return values

Name	Description
PASS	Normal end
FAIL	Internal I/O buffer is not empty
RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized.

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The key buffer must contain 24 bytes of TDES key data and

NOTE: the iv buffer must have 8 bytes of random data.

R_SCE_TDES_192CbcDecrypt

```
 R_SCE_TDES_192CbcDecrypt (  tdes_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Brief description

Triple DES Decryption with CBC mode.

Detailed description

Decrypt num_words words of input data from buffer p_source using the 192-bit TDES key from buffer key and initialization vector from buffer iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 2133:Return values

Name	Description
PASS	Normal end
FAIL	Internal I/O buffer is not empty
RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized.

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The key buffer must contain 24 bytes of TDES key data and

NOTE: the iv buffer must have 8 bytes of random data.

R_SCE_TDES_192CtrEncrypt

```
 R_SCE_TDES_192CtrEncrypt (  tdes_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Brief description

Triple DES Encryption with CTR mode.

Detailed description

Encrypt num_words words of input data from buffer p_source using the 192-bit TDES key from buffer key and initialization vector from buffer iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for at least num_words words of data.

Table 2134:Return values

Name	Description
PASS	Normal end
FAIL	Internal I/O buffer is not empty
RESOURCE_CONFLICT	resource conflict occured
SSP_ERR_CRYPTO_NOT_IMPLEMENTED	API not yet implemented

NOTE: SCE module must have been initialized.

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The key buffer must contain 24 bytes of TDES key data and

NOTE: the iv buffer must have 8 bytes of random data.

R_SCE_TDES_192EcbEncrypt

```
 R_SCE_TDES_192EcbEncrypt (  tdes_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Brief description

Triple DES Encryption with ECB mode.

Detailed description

Encrypt num_words words of input data from buffer p_source using the 192-bit TDES key from buffer key and initialization vector from buffer iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Table 2135:Return values

Name	Description
PASS	Normal end
FAIL	Internal I/O buffer is not empty
RESOURCE_CONFLICT	resource conflict occurred

NOTE: SCE module must have been initialized and RNG quality check performed by calling the functions R_SCE_SoftReset(), R_SCE_Initialization1() and R_SCE_Initialization2()

NOTE: Block size for TDES is 64-bits (8 bytes or 2 words). Hence num_words must be a multiple of 2.

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The key buffer must contain 24 bytes of TDES key data.

NOTE: The contents of iv buffer are ignored in ECB chaining mode.

Function steps

- To prevent compiler warning for unused parameter.

R_SCE_TDES_192EcbDecrypt

```
 R_SCE_TDES_192EcbDecrypt (  tdes_ctrl_t *const p_ctrl ,   const uint32_t
* p_key ,   uint32_t * p_iv ,   uint32_t num_words ,   uint32_t * p_source ,
  uint32_t * p_dest )
```

Brief description

Triple DES Decryption with ECB mode. Decrypt num_words words of input data from buffer p_source using the 192-bit TDES key from buffer key and initialization vector from buffer iv. The result will be written to the output buffer from p_dest. The p_dest array is assumed to have space for atleast num_words words of data.

Detailed description

Table 2136:Return values

Name	Description
PASS	Normal end
FAIL	Internal I/O buffer is not empty
RESOURCE_CONFLICT	resource conflict occured

NOTE: SCE module must have been initialized.

NOTE: p_dest must have space to hold at least num_words words of data.

NOTE: The key buffer must contain 24 bytes of TDES key data and

NOTE: The contents of iv buffer are ignored in ECB chaining mode.

Function steps

- To prevent compiler warning for unused parameter.

9.55.8.10 SCE_TRNG

Primitive cryptographic functions.

random number generation functions

Functions

- s_generate_16byte_random_data
- R_SCE_TRNG_Open
- R_SCE_TRNG_Close
- R_SCE_TRNG_VersionGet
- R_SCE_TRNG_Read

Variables

- g_trng_on_sce
- s_sce_trng_version

g_trng_on_sce

`trng_api_t::g_trng_on_sce`

Detailed description

SCE/TRNG implementation of RNG API.

Initialized as

```
g_trng_on_sce=
{
    .open       = R_SCE_TRNG_Open,
    .read       = R_SCE_TRNG_Read,
    .close      = R_SCE_TRNG_Close,
    .versionGet = R_SCE_TRNG_VersionGet
}
```

s_sce_trng_version

`ssp_version_t::s_sce_trng_version`

Detailed description

Version data structure used by error logger macro.

Initialized as

```
s_sce_trng_version=
{
    .api_version_major  = #define  TRNG_API_VERSION_MAJOR,
    .api_version_minor  = TRNG_API_VERSION_MINOR,
    .code_version_major = SCE_TRNG_CODE_VERSION_MAJOR,
    .code_version_minor = SCE_TRNG_CODE_VERSION_MINOR
}
```

s_generate_16byte_random_data

```
s_generate_16byte_random_data ( trng_ctrl_t  * p_ctrl )
```

Brief description

Generate 16-bytes of random data using the SCE TRNG hardware block.

Detailed description

This function makes p_ctrl->nattempts to generate 16-bytes of random data using the SCE TRNG block. If successful, the buffer p_ctrl->currbuf[0..15] will be updated with newly generated 16-byte block. This data will be different from previously generated 16-byte block (p_ctrl->prevbuf[0..15]).

Table 2137:Return values

Name	Description
SF_CRYPTO_SUCCESS	successul generation of 16-byte random data

Table 2137:Return values (Continued)

Name	Description
SSP_ERR_CRYPTO_RNG_FATAL_ERROR	fatal error, unable to generate 16-byte random data that differs from the previously generated 16-byte block.

R_SCE_TRNG_Open

```
R_SCE_TRNG_Open ( trng_ctrl_t  *const p_ctrl ,  trng_cfg_t  const *const p_cfg )
```

Detailed description

Open the TRNG driver for reading random data from the hardware TRNG module TRNG Initialization

Table 2138:Return values

Name	Description
SF_CRYPTO_SUCCESS	random number generation successful
SF_CRPYTO_ERR_CRYPTO_RESOURCE_CONFLICT	SCE resource is busy
SF_CRYPTO_ERR_CRYPTO_SCEFAIL	SCE internal I/O is not empty

R_SCE_TRNG_Close

```
R_SCE_TRNG_Close ( trng_ctrl_t  *const p_ctrl )
```

Detailed description

Close the TRNG interface driver

R_SCE_TRNG_VersionGet

```
R_SCE_TRNG_VersionGet ( ssp_version_t  *const p_version )
```

Brief description

Sets driver version based on compile time macros.

Detailed description

Table 2139:Return values

Name	Description
SSP_SUCCESS	Successful close.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

R_SCE_TRNG_Read

```
R_SCE_TRNG_Read ( trng_ctrl_t  *const p_ctrl ,   uint32_t *const p_rngbuf ,
 uint32_t nwords )
```

Detailed description

Generate nwords of random number words (4-bytes each) and store them in p_rngbuf buffer SCE hardware TRNG module will be used for generating the random data.

Table 2140:Return values

Name	Description
SF_CRYPTO_SUCCESS	random number generation successful
SSP_ERR_CRYPTO_RNG_FATAL_ERROR	HW_SCE_RNG_Read failed to generate 128-bit (16-byte) random number in p_ctrl->nattempts number of attempts.
SF_CRPYTO_ERR_CRYPTO_RESOURCE_CONFLICT	SCE resource is busy
SF_CRYPTO_ERR_CRYPTO_SCEFAIL	SCE internal I/O is not empty

Chapter 10 API Reference: Board Support Package

10.1 Board Support Package

Common BSP includes.

The BSP is responsible for getting the MCU from reset to the user application (i.e. the main function). Before reaching the user application the BSP sets up the stacks, heap, clocks, interrupts, and C runtime environment. The BSP is configurable to allow users to modify the process to meet design requirements.

10.2 Supported MCUs

Supported MCUs in this version of the BSP.

The BSP has code specific to a MCU and a board. The code that is specific to a MCU can be shared amongst boards that use the MCU.

10.2.1 Functions

- R_SSP_VersionGet

10.2.2 R_SSP_VersionGet

```
ssp_err_t R_SSP_VersionGet ( ssp_pack_version_t  *const p_version )
```

10.2.2.1 Brief description

Set SSP version based on compile time macros.

10.2.2.2 Detailed description

Table 2141:Parameters

Name	Direction	Description
p_version	out	Memory address to return version information to.

Table 2142:Return values

Name	Description
SSP_SUCCESS	Version information stored.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

10.2.3 S124

Code that is common to S124 MCUs.

Implements functions that are common to S124 MCUs.

10.2.3.1 elc_peripheral_t

Possible peripherals to be linked to event signals

10.2.3.2 elc_event_t

Sources of event signals to be linked to other peripherals or the CPU1
NOTE: This list may change based on device. This list is for S124.

10.2.3.3 Cache Functions

This module implements cache functions.

Typedefs

- bsp_cache_frequency_t

bsp_cache_state_t

Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

bsp_cache_frequency_t

```
typedef uint32_t bsp_cache_frequency_t
```

10.2.3.4 Clock Initialization

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Functions

- bsp_clock_init
- bsp_cpu_clock_get

- bsp_clock_set_callback
- bsp_clocks_mem_wait_set
- bsp_clocks_mem_wait_get
- bsp_clock_set_prechange
- bsp_clock_set_postchange
- bsp_clocks_rom_wait_set
- bsp_clocks_rom_wait_get
- bsp_clocks_sram_wait_set
- bsp_clocks_hsram_wait_set

Defines

- #define CGC_SRAM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for SRAM.
- #define CGC_SRAM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for SRAM.
- #define CGC_ZERO_WAIT_CYCLES

  Initial value: `(0U)`
- #define CGC_TWO_WAIT_CYCLES

  Initial value: `(1U)`
- #define CGC_MAX_ZERO_WAIT_FREQ

  Initial value: `(32000000U)`
- #define CGC_PRCR_KEY

  Initial value: `(0xA500U)`
- #define CGC_PRCR_UNLOCK

  Initial value: `((CGC_PRCR_KEY) | 0x1U)`
- #define CGC_PRCR_LOCK

  Initial value: `((CGC_PRCR_KEY) | 0x0U)`
- #define CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS

  Initial value: `(120000000U)`
- #define CGC_SYS_CLOCK_FREQ_NO_HSRAM_WAITS

  Initial value: `(200000000U)`

- #define CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS

  Initial value: `(80000000U)`
- #define CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS

  Initial value: `(160000000U)`
- #define CGC_ROM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for ROM.
- #define CGC_ROM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for ROM.
- #define CGC_ROM_TWO_WAIT_CYCLES

  Initial value: `(2U)`

  Specify two wait states for ROM.
- #define CGC_SRAM_PRCR_KEY

  Initial value: `(0x78U)`
- #define CGC_SRAM_UNLOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x1U)`
- #define CGC_SRAM_LOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x0U)`

bsp_clock_init

```
bsp_clock_init (  void  )
```

Brief description

Sets up system clocks.

Function steps

- MOCO is default clock out of reset. Enable new clock if chosen. S124 has no PLL.
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- PLL Source clock is always the main oscillator.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- PLL Source clock is always the main oscillator.

- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- Set PLL Source clock.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- Set USB clock divisor.
- Configure BCLK
- Configure SDRAM Clock

bsp_cpu_clock_get

```
bsp_cpu_clock_get (  void  )
```

Brief description

Returns frequency of CPU clock in Hz.

Detailed description

Table 2143:Return values

Name	Description
Frequency	of the CPU in Hertz

bsp_clock_set_callback

```
ssp_err_t bsp_clock_set_callback (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings for a requested system clock change (PRE) or a just updated system clock change (POST)

Detailed description

Table 2144:Parameters

Name	Direction	Description
p_args	in	- Pre/Post request and clock frequency information

Table 2145:Return values

Name	Description
return	SSP_SUCCESS

Function steps

- The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0.
- < No MEMWAIT cycles

bsp_clocks_mem_wait_set

```
bsp_clocks_mem_wait_set (  uint32_t setting )
```

Brief description

This function sets the value of the MEMWAIT register which controls wait cycles for flash read access.

Detailed description

Table 2146:Parameters

Name	Direction	Description
setting	in	

Table 2147:Return values

Name	Description
none	

bsp_clocks_mem_wait_get

```
bsp_clocks_mem_wait_get (  void  )
```

Brief description

This function gets the value of the MEMWAIT register.

Detailed description

Table 2148:Return values

Name	Description
MEMWAIT	setting

bsp_clock_set_prechange

```
bsp_clock_set_prechange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the requested clock change to the CGC.

Detailed description

Table 2149:Return values

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK < 120 MHz
- 1 wait: ICLK >= 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- Calculate the Wait states for ROM
- If the requested_frequency <= 80 MHz 0 ROM wait states are required)
- If the 80 MHz < requested_frequency <= 160 MHz 1 ROM wait state is required)
- Anything above 160 MHz requires 2 ROM wait states
- Set the wait state BEFORE we change iclk

bsp_clock_set_postchange

```
bsp_clock_set_postchange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the system clock which has just been set by the CGC.

Detailed description

Table 2150:Return values

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK <= 120 MHz
- 1 wait: ICLK > 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- In this case we need to set the wait state AFTER we change iclk

bsp_clocks_rom_wait_set

```
bsp_clocks_rom_wait_set (  uint8_t setting )
```

Brief description

This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM.

Detailed description

Table 2151:Parameters

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2152:Return values

Name	Description
none	

bsp_clocks_rom_wait_get

```
bsp_clocks_rom_wait_get (  void  )
```

Brief description

This function gets the value of the ROMWT register.

Detailed description

Table 2153:Return values

Name	Description
MEMWAIT	setting

bsp_clocks_sram_wait_set

```
bsp_clocks_sram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for both the SRAM0 and SRAM1 RAM memory.

Detailed description

Table 2154:Parameters

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2155:Return values

Name	Description
none	

bsp_clocks_hsram_wait_set

```
bsp_clocks_hsram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for High Speed RAM memory.

Detailed description

Table 2156:Parameters

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2157:Return values

Name	Description
none	

10.2.3.5 Hardware Locks

This file allocates hardware locks used in Atomic Locking.

Functions

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 0U )
```

Detailed description

Used to allocated hardware locks. Parameters are as follows:

1) IP name (ssp_ip_t enum without the SSP_IP_ prefix).
2) Unit number (used for blocks with variations like USB, not to be confused with ADC unit).
3) Channel number

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AES , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( BSC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CTSU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DOC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ELC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( FCU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IWDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( KEY , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LPM , 1U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( RTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TRNG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TSN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( WDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAAD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 14U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 15U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IRDA , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MMF , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MPU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPAMP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( QSPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCE , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DRW , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( EPTPC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( JPEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( PDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 1U , 0U )
```

10.2.3.6 Module Start and Stop

Module start and stop functions are provided to enable or disable peripherals.

Functions

- r_bsp_module_start_stop
- R_BSP_ModuleStop
- R_BSP_ModuleStart

Data structures

- bsp_mstp_bit_t
- bsp_mstp_data_t

Variables

- g_bsp_module_stop
- gp_mstp

Defines

- #define BSP_MSTP_DATA_INVALID

  Initial value: `(0xFFU)`
- #define BSP_MSTP_EXCEPTION

  Initial value: `(1U << 5)`
- #define BSP_MSTP_REG

  Initial value: `((x) << 6)`
- #define BSP_MSTP_BIT

  Initial value: `(x)`
- #define BSP_MSTPCRA

  Initial value: `(0U)`
- #define BSP_MSTPCRB

  Initial value: `(1U)`
- #define BSP_MSTPCRC

  Initial value: `(2U)`
- #define BSP_MSTPCRD

  Initial value: `(3U)`
- #define BSP_COMPILE_TIME_ASSERT

  Initial value: `((void) sizeof(char[1 - 2 * !(e)]))`

  Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

g_bsp_module_stop

```
const uint8_t g_bsp_module_stop[]
```

Detailed description

Module stop bit definitions for each ssp_ip_t enum value.

gp_mstp

```
volatile uint32_t* const gp_mstp[]
```

Detailed description

Module stop register addresses.

r_bsp_module_start_stop

```
ssp_err_t r_bsp_module_start_stop (  ssp_feature_t const *const p_feature ,
  bool stop )
```

Brief description

Start module (cancel module stop) or stop module (enter module stop).

Detailed description

NOTE: Some module stop bits are shared between peripherals. Entering module stop is not permitted for these peripherals.

Table 2158:Parameters

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.
stop	in	true to stop module, false to start module

Table 2159:Return values

Name	Description
SSP_SUCCESS	Module is stopped or started based on stop parameter
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

Function steps

- The g_bsp_module_stop array must have entries for each ssp_ip_t enum value.
- Set MSTP bit if stop parameter is true and the bit is not shared with other channels.
- Clear MSTP bit if stop parameter is false.

R_BSP_ModuleStop

```
ssp_err_t R_BSP_ModuleStop (  ssp_feature_t const *const p_feature )
```

Brief description

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Detailed description

NOTE: Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Table 2160:Parameters

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2161:Return values

Name	Description
SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

R_BSP_ModuleStart

```
ssp_err_t R_BSP_ModuleStart (  ssp_feature_t const *const p_feature )
```

Brief description

Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Detailed description

Table 2162:Parameters

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2163:Return values

Name	Description
SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

bsp_mstp_bit_t

bsp_mstp_bit_t

Detailed description

Definition of BSP module stop bit.

Variables

- uint8_t bit

  Which bit in MSTP register.

- uint8_t reg

  < Special processing required

  Which MSTP register

bsp_mstp_data_t

Detailed description

Definition of BSP module stop bit.

Variables

byte

bit

10.2.3.7 ROM Registers

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Functions

- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2

Defines

- #define BSP_ROM_REG_OFS1_SETTING

  Initial value:
  ```
  (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) |
  ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))
  ```

  OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

- #define BSP_ROM_REG_MPU_CONTROL_SETTING

  Initial value:
  ```
  ((0xFFFFFCFEU) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABLE << 8) |
  \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABLE << 9) | \
  ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_ENABLE))
  ```

  Build up SECMPUAC register based on MPU settings.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ROM_REGISTERS )
```

Detailed description

ROM registers defined here. Some have masks to make sure reserved bits are set appropriately. S124 has no MPU.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ID_CODE_1 )
```

Detailed description

ID code definitions defined here.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ID_CODE_2 )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ID_CODE_3 )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ID_CODE_4 )
```

10.2.4 S128

Code that is common to S128 MCUs.

Implements functions that are common to S128 MCUs.

10.2.4.1 Cache Functions

This module implements cache functions.

Typedefs

- bsp_cache_frequency_t

bsp_cache_state_t

Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

bsp_cache_frequency_t

```
typedef uint32_t bsp_cache_frequency_t
```

10.2.4.2 Clock Initialization

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Functions

- bsp_clock_init
- bsp_cpu_clock_get
- bsp_clock_set_callback
- bsp_clocks_mem_wait_set
- bsp_clocks_mem_wait_get
- bsp_clock_set_prechange
- bsp_clock_set_postchange
- bsp_clocks_rom_wait_set
- bsp_clocks_rom_wait_get
- bsp_clocks_sram_wait_set
- bsp_clocks_hsram_wait_set

Defines

- #define CGC_SRAM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for SRAM.
- #define CGC_SRAM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for SRAM.

- #define CGC_ZERO_WAIT_CYCLES

  Initial value: `(0U)`
- #define CGC_TWO_WAIT_CYCLES

  Initial value: `(1U)`
- #define CGC_MAX_ZERO_WAIT_FREQ

  Initial value: `(32000000U)`
- #define CGC_PRCR_KEY

  Initial value: `(0xA500U)`
- #define CGC_PRCR_UNLOCK

  Initial value: `((CGC_PRCR_KEY) | 0x1U)`
- #define CGC_PRCR_LOCK

  Initial value: `((CGC_PRCR_KEY) | 0x0U)`
- #define CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS

  Initial value: `(120000000U)`
- #define CGC_SYS_CLOCK_FREQ_NO_HSRAM_WAITS

  Initial value: `(200000000U)`
- #define CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS

  Initial value: `(80000000U)`
- #define CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS

  Initial value: `(160000000U)`
- #define CGC_ROM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for ROM.
- #define CGC_ROM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for ROM.
- #define CGC_ROM_TWO_WAIT_CYCLES

  Initial value: `(2U)`

  Specify two wait states for ROM.
- #define CGC_SRAM_PRCR_KEY

  Initial value: `(0x78U)`
- #define CGC_SRAM_UNLOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x1U)`

- #define CGC_SRAM_LOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x0U)`

bsp_clock_init

```
bsp_clock_init (  void  )
```

Brief description

Sets up system clocks.

Function steps

- MOCO is default clock out of reset. Enable new clock if chosen. S124 has no PLL.
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- PLL Source clock is always the main oscillator.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- PLL Source clock is always the main oscillator.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- Set PLL Source clock.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- Set USB clock divisor.
- Configure BCLK
- Configure SDRAM Clock

bsp_cpu_clock_get

```
bsp_cpu_clock_get ( void )
```

Brief description

Returns frequency of CPU clock in Hz.

Detailed description

Table 2164:Return values

Name	Description
Frequency	of the CPU in Hertz

bsp_clock_set_callback

```
ssp_err_t bsp_clock_set_callback ( bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings for a requested system clock change (PRE) or a just updated system clock change (POST)

Detailed description

Table 2165:Parameters

Name	Direction	Description
p_args	in	- Pre/Post request and clock frequency information

Table 2166:Return values

Name	Description
return	SSP_SUCCESS

Function steps

- The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0.
- < No MEMWAIT cycles

bsp_clocks_mem_wait_set

```
bsp_clocks_mem_wait_set ( uint32_t setting )
```

Brief description

This function sets the value of the MEMWAIT register which controls wait cycles for flash read access.

Detailed description

Table 2167:Parameters

Name	Direction	Description
setting	in	

Table 2168:Return values

Name	Description
none	

bsp_clocks_mem_wait_get

```
bsp_clocks_mem_wait_get (  void  )
```

Brief description

This function gets the value of the MEMWAIT register.

Detailed description

Table 2169:Return values

Name	Description
MEMWAIT	setting

bsp_clock_set_prechange

```
bsp_clock_set_prechange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the requested clock change to the CGC.

Detailed description

Table 2170:Return values

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)

- No wait: ICLK < 120 MHz
- 1 wait: ICLK >= 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- Calculate the Wait states for ROM
- If the requested_frequency <= 80 MHz 0 ROM wait states are required)
- If the 80 MHz < requested_frequency <= 160 MHz 1 ROM wait state is required)
- Anything above 160 MHz requires 2 ROM wait states
- Set the wait state BEFORE we change iclk

bsp_clock_set_postchange

```
bsp_clock_set_postchange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the system clock which has just been set by the CGC.

Detailed description

Table 2171:Return values

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK <= 120 MHz
- 1 wait: ICLK > 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- In this case we need to set the wait state AFTER we change iclk

bsp_clocks_rom_wait_set

```
bsp_clocks_rom_wait_set (  uint8_t setting )
```

Brief description

This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM.

Detailed description

Table 2172:Parameters

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2173:Return values

Name	Description
none	

bsp_clocks_rom_wait_get

```
bsp_clocks_rom_wait_get (  void  )
```

Brief description

This function gets the value of the ROMWT register.

Detailed description

Table 2174:Return values

Name	Description
MEMWAIT	setting

bsp_clocks_sram_wait_set

```
bsp_clocks_sram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for both the SRAM0 and SRAM1 RAM memory.

Detailed description

Table 2175:Parameters

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2176:Return values

Name	Description
none	

bsp_clocks_hsram_wait_set

```
bsp_clocks_hsram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for High Speed RAM memory.

Detailed description

Table 2177:Parameters

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2178:Return values

Name	Description
none	

10.2.4.3 Hardware Locks

This file allocates hardware locks used in Atomic Locking.

Functions

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 0U )
```

Detailed description

Used to allocated hardware locks. Parameters are as follows:

1) IP name (ssp_ip_t enum without the SSP_IP_ prefix).
2) Unit number (used for blocks with variations like USB, not to be confused with ADC unit).
3) Channel number

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AES , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( BSC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CTSU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DOC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ELC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( FCU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IWDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( KEY , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LPM , 1U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( RTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TRNG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TSN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( WDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAAD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 14U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 15U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IRDA , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MMF , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MPU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPAMP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( QSPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCE , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DRW , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( EPTPC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( JPEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( PDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 1U , 0U )
```

10.2.4.4 Module Start and Stop

Module start and stop functions are provided to enable or disable peripherals.

Functions

- r_bsp_module_start_stop
- R_BSP_ModuleStop
- R_BSP_ModuleStart

Data structures

- bsp_mstp_bit_t
- bsp_mstp_data_t

Variables

- g_bsp_module_stop
- gp_mstp

Defines

- #define BSP_MCU_GROUP_S128

  Initial value:
- #define BSP_MSTP_DATA_INVALID

  Initial value: `(0xFFU)`
- #define BSP_MSTP_EXCEPTION

  Initial value: `(1U << 5)`

- #define BSP_MSTP_REG

  Initial value: `((x) << 6)`
- #define BSP_MSTP_BIT

  Initial value: `(x)`
- #define BSP_MSTPCRA

  Initial value: `(0U)`
- #define BSP_MSTPCRB

  Initial value: `(1U)`
- #define BSP_MSTPCRC

  Initial value: `(2U)`
- #define BSP_MSTPCRD

  Initial value: `(3U)`
- #define BSP_COMPILE_TIME_ASSERT

  Initial value: `((void) sizeof(char[1 - 2 * !(e)]))`

  Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

g_bsp_module_stop

```
const uint8_t g_bsp_module_stop[]
```

Detailed description

Module stop bit definitions for each ssp_ip_t enum value.

gp_mstp

```
volatile uint32_t* const gp_mstp[]
```

Detailed description

Module stop register addresses.

r_bsp_module_start_stop

```
ssp_err_t r_bsp_module_start_stop (  ssp_feature_t const *const p_feature ,
  bool stop )
```

Brief description

Start module (cancel module stop) or stop module (enter module stop).

Detailed description

NOTE: Some module stop bits are shared between peripherals. Entering module stop is not permitted for these peripherals.

Table 2179:Parameters

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.
stop	in	true to stop module, false to start module

Table 2180:Return values

Name	Description
SSP_SUCCESS	Module is stopped or started based on stop parameter
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

Function steps

- The g_bsp_module_stop array must have entries for each ssp_ip_t enum value.
- Set MSTP bit if stop parameter is true and the bit is not shared with other channels.
- Clear MSTP bit if stop parameter is false.

R_BSP_ModuleStop

```
ssp_err_t R_BSP_ModuleStop (  ssp_feature_t const *const p_feature )
```

Brief description

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Detailed description

NOTE: Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Table 2181:Parameters

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2182:Return values

Name	Description
SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

R_BSP_ModuleStart

```
ssp_err_t R_BSP_ModuleStart (  ssp_feature_t const *const p_feature )
```

Brief description

Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Detailed description

Table 2183:Parameters

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2184:Return values

Name	Description
SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

bsp_mstp_bit_t

bsp_mstp_bit_t

Detailed description

Definition of BSP module stop bit.

Variables

- uint8_t bit

  Which bit in MSTP register.

- uint8_t reg

  < Special processing required

  Which MSTP register

bsp_mstp_data_t

Detailed description

Definition of BSP module stop bit.

Variables

byte

bit

10.2.4.5 ROM Registers

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Functions

- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2

Defines

- #define BSP_ROM_REG_OFS1_SETTING

  Initial value:`(((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))`

  OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

- #define BSP_ROM_REG_MPU_CONTROL_SETTING

  Initial value:`((0xFFFFFCFEU) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABLE << 8) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABLE << 9) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_ENABLE))`

  Build up SECMPUAC register based on MPU settings.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ROM_REGISTERS  )
```

Detailed description

ROM registers defined here. Some have masks to make sure reserved bits are set appropriately. S124 has no MPU.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_1  )
```

Detailed description

ID code definitions defined here.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_2  )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_3  )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_4  )
```

10.2.5 S1JA

Code that is common to S1JA MCUs.

Implements functions that are common to S1JA MCUs.

10.2.5.1 Cache Functions

This module implements cache functions.

Typedefs

- bsp_cache_frequency_t

bsp_cache_state_t

Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

bsp_cache_frequency_t

```
typedef uint32_t bsp_cache_frequency_t
```

10.2.5.2 Clock Initialization

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Functions

- bsp_clock_init
- bsp_cpu_clock_get
- bsp_clock_set_callback
- bsp_clocks_mem_wait_set
- bsp_clocks_mem_wait_get
- bsp_clock_set_prechange
- bsp_clock_set_postchange
- bsp_clocks_rom_wait_set
- bsp_clocks_rom_wait_get
- bsp_clocks_sram_wait_set
- bsp_clocks_hsram_wait_set

Defines

- #define CGC_SRAM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for SRAM.
- #define CGC_SRAM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for SRAM.
- #define CGC_ZERO_WAIT_CYCLES

  Initial value: `(0U)`
- #define CGC_TWO_WAIT_CYCLES

  Initial value: `(1U)`
- #define CGC_MAX_ZERO_WAIT_FREQ

  Initial value: `(32000000U)`
- #define CGC_PRCR_KEY

  Initial value: `(0xA500U)`
- #define CGC_PRCR_UNLOCK

  Initial value: `((CGC_PRCR_KEY) | 0x1U)`
- #define CGC_PRCR_LOCK

  Initial value: `((CGC_PRCR_KEY) | 0x0U)`

- #define CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS

  Initial value: `(120000000U)`
- #define CGC_SYS_CLOCK_FREQ_NO_HSRAM_WAITS

  Initial value: `(200000000U)`
- #define CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS

  Initial value: `(80000000U)`
- #define CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS

  Initial value: `(160000000U)`
- #define CGC_ROM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for ROM.
- #define CGC_ROM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for ROM.
- #define CGC_ROM_TWO_WAIT_CYCLES

  Initial value: `(2U)`

  Specify two wait states for ROM.
- #define CGC_SRAM_PRCR_KEY

  Initial value: `(0x78U)`
- #define CGC_SRAM_UNLOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x1U)`
- #define CGC_SRAM_LOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x0U)`

bsp_clock_init

```
bsp_clock_init (  void  )
```

Brief description

Sets up system clocks.

Function steps

- MOCO is default clock out of reset. Enable new clock if chosen. S124 has no PLL.
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.

- PLL Source clock is always the main oscillator.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- PLL Source clock is always the main oscillator.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- Set PLL Source clock.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- Set USB clock divisor.
- Configure BCLK
- Configure SDRAM Clock

bsp_cpu_clock_get

```
bsp_cpu_clock_get (  void  )
```

Brief description

Returns frequency of CPU clock in Hz.

Detailed description

Table 2185:Return values

Name	Description
Frequency	of the CPU in Hertz

bsp_clock_set_callback

```
ssp_err_t bsp_clock_set_callback (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings for a requested system clock change (PRE) or a just updated system clock change (POST)

Detailed description

Table 2186:Parameters

Name	Direction	Description
p_args	in	- Pre/Post request and clock frequency information

Table 2187:Return values

Name	Description
return	SSP_SUCCESS

Function steps

- The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0.
- < No MEMWAIT cycles

bsp_clocks_mem_wait_set

```
bsp_clocks_mem_wait_set (  uint32_t setting )
```

Brief description

This function sets the value of the MEMWAIT register which controls wait cycles for flash read access.

Detailed description

Table 2188:Parameters

Name	Direction	Description
setting	in	

Table 2189:Return values

Name	Description
none	

bsp_clocks_mem_wait_get

```
bsp_clocks_mem_wait_get (  void  )
```

Brief description

This function gets the value of the MEMWAIT register.

Detailed description

Table 2190:Return values

Name	Description
MEMWAIT	setting

bsp_clock_set_prechange

```
bsp_clock_set_prechange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the requested clock change to the CGC.

Detailed description

Table 2191:Return values

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK < 120 MHz
- 1 wait: ICLK >= 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- Calculate the Wait states for ROM
- If the requested_frequency <= 80 MHz 0 ROM wait states are required)
- If the 80 MHz < requested_frequency <= 160 MHz 1 ROM wait state is required)
- Anything above 160 MHz requires 2 ROM wait states
- Set the wait state BEFORE we change iclk

bsp_clock_set_postchange

```
bsp_clock_set_postchange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the system clock which has just been set by the CGC.

Detailed description

Table 2192:Return values

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK <= 120 MHz
- 1 wait: ICLK > 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- In this case we need to set the wait state AFTER we change iclk

bsp_clocks_rom_wait_set

```
bsp_clocks_rom_wait_set (  uint8_t setting )
```

Brief description

This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM.

Detailed description

Table 2193:Parameters

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2194:Return values

Name	Description
none	

bsp_clocks_rom_wait_get

```
bsp_clocks_rom_wait_get (  void  )
```

Brief description

This function gets the value of the ROMWT register.

Detailed description

Table 2195:Return values

Name	Description
MEMWAIT	setting

bsp_clocks_sram_wait_set

```
bsp_clocks_sram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for both the SRAM0 and SRAM1 RAM memory.

Detailed description

Table 2196:Parameters

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2197:Return values

Name	Description
none	

bsp_clocks_hsram_wait_set

```
bsp_clocks_hsram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for High Speed RAM memory.

Detailed description

Table 2198:Parameters

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2199:Return values

Name	Description
none	

10.2.5.3 Hardware Locks

This file allocates hardware locks used in Atomic Locking.

Functions

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 0U )
```

Detailed description

Used to allocated hardware locks. Parameters are as follows:

1) IP name (ssp_ip_t enum without the SSP_IP_ prefix).
2) Unit number (used for blocks with variations like USB, not to be confused with ADC unit).
3) Channel number

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AES , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( BSC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CTSU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DOC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ELC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( FCU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IWDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( KEY , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LPM , 1U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( RTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TRNG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TSN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( WDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAAD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 14U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 15U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IRDA , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MMF , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MPU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPAMP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( QSPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCE , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DRW , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( EPTPC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( JPEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( PDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 1U , 0U )
```

10.2.5.4 Module Start and Stop

Module start and stop functions are provided to enable or disable peripherals.

Functions

- r_bsp_module_start_stop
- R_BSP_ModuleStop
- R_BSP_ModuleStart

Data structures

- bsp_mstp_bit_t
- bsp_mstp_data_t

Variables

- g_bsp_module_stop
- gp_mstp

Defines

- #define BSP_MCU_GROUP_S1JA

  Initial value:
- #define BSP_MSTP_DATA_INVALID

  Initial value: `(0xFFU)`
- #define BSP_MSTP_EXCEPTION

  Initial value: `(1U << 5)`
- #define BSP_MSTP_REG

  Initial value: `((x) << 6)`
- #define BSP_MSTP_BIT

  Initial value: `(x)`
- #define BSP_MSTPCRA

  Initial value: `(0U)`
- #define BSP_MSTPCRB

  Initial value: `(1U)`
- #define BSP_MSTPCRC

  Initial value: `(2U)`
- #define BSP_MSTPCRD

  Initial value: `(3U)`
- #define BSP_COMPILE_TIME_ASSERT

  Initial value: `((void) sizeof(char[1 - 2 * !(e)]))`

  Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

g_bsp_module_stop

```
const uint8_t g_bsp_module_stop[]
```

Detailed description

Module stop bit definitions for each ssp_ip_t enum value.

gp_mstp

```
volatile uint32_t* const gp_mstp[]
```

Detailed description

Module stop register addresses.

r_bsp_module_start_stop

```
ssp_err_t r_bsp_module_start_stop (  ssp_feature_t const *const p_feature ,
  bool stop )
```

Brief description

Start module (cancel module stop) or stop module (enter module stop).

Detailed description

NOTE: Some module stop bits are shared between peripherals. Entering module stop is not permitted for these peripherals.

Table 2200:Parameters

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.
stop	in	true to stop module, false to start module

Table 2201:Return values

Name	Description
SSP_SUCCESS	Module is stopped or started based on stop parameter
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

Function steps

- The g_bsp_module_stop array must have entries for each ssp_ip_t enum value.
- Set MSTP bit if stop parameter is true and the bit is not shared with other channels.
- Clear MSTP bit if stop parameter is false.

R_BSP_ModuleStop

```
ssp_err_t R_BSP_ModuleStop (  ssp_feature_t const *const p_feature )
```

Brief description

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Detailed description

NOTE: Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Table 2202:Parameters

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2203:Return values

Name	Description
SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

R_BSP_ModuleStart

```
ssp_err_t R_BSP_ModuleStart (  ssp_feature_t const *const p_feature )
```

Brief description

Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Detailed description

Table 2204:Parameters

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2205:Return values

Name	Description
SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

bsp_mstp_bit_t

bsp_mstp_bit_t

Detailed description

Definition of BSP module stop bit.

Variables

- uint8_t bit

  Which bit in MSTP register.

- uint8_t reg

  < Special processing required

  Which MSTP register

bsp_mstp_data_t

Detailed description

Definition of BSP module stop bit.

Variables

byte

bit

10.2.5.5 ROM Registers

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Functions

- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2

Defines

- #define BSP_ROM_REG_OFS1_SETTING

  Initial value:
  ```
  (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) |
  ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))
  ```

  OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

- #define BSP_ROM_REG_MPU_CONTROL_SETTING

  Initial value:
  ```
  ((0xFFFFFCFEU) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABLE << 8) |
  \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABLE << 9) | \
  ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_ENABLE))
  ```

  Build up SECMPUAC register based on MPU settings.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ROM_REGISTERS )
```

Detailed description

ROM registers defined here. Some have masks to make sure reserved bits are set appropriately. S124 has no MPU.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ID_CODE_1 )
```

Detailed description

ID code definitions defined here.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ID_CODE_2 )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ID_CODE_3 )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ID_CODE_4 )
```

10.2.6 S3A1

Code that is common to S3A1 MCUs.

Implements functions that are common to S3A1 MCUs.

10.2.6.1 elc_peripheral_t

Possible peripherals to be linked to event signals

10.2.6.2 elc_event_t

Sources of event signals to be linked to other peripherals or the CPU1
NOTE: This list may change based on device. This list is for S3A7.

10.2.6.3 Cache Functions

This module implements cache functions.

Typedefs

- bsp_cache_frequency_t

bsp_cache_state_t

Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

bsp_cache_frequency_t

```
typedef uint32_t bsp_cache_frequency_t
```

10.2.6.4 Clock Initialization

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Functions

- bsp_clock_init
- bsp_cpu_clock_get
- bsp_clock_set_callback
- bsp_clocks_mem_wait_set
- bsp_clocks_mem_wait_get
- bsp_clock_set_prechange
- bsp_clock_set_postchange
- bsp_clocks_rom_wait_set
- bsp_clocks_rom_wait_get
- bsp_clocks_sram_wait_set
- bsp_clocks_hsram_wait_set

Defines

- #define CGC_SRAM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for SRAM.
- #define CGC_SRAM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for SRAM.
- #define CGC_ZERO_WAIT_CYCLES

  Initial value: `(0U)`
- #define CGC_TWO_WAIT_CYCLES

  Initial value: `(1U)`
- #define CGC_MAX_ZERO_WAIT_FREQ

  Initial value: `(32000000U)`
- #define CGC_PRCR_KEY

  Initial value: `(0xA500U)`
- #define CGC_PRCR_UNLOCK

  Initial value: `((CGC_PRCR_KEY) | 0x1U)`
- #define CGC_PRCR_LOCK

  Initial value: `((CGC_PRCR_KEY) | 0x0U)`
- #define CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS

  Initial value: `(120000000U)`
- #define CGC_SYS_CLOCK_FREQ_NO_HSRAM_WAITS

  Initial value: `(200000000U)`
- #define CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS

  Initial value: `(80000000U)`
- #define CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS

  Initial value: `(160000000U)`
- #define CGC_ROM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for ROM.
- #define CGC_ROM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for ROM.

- #define CGC_ROM_TWO_WAIT_CYCLES

  Initial value: `(2U)`

  Specify two wait states for ROM.
- #define CGC_SRAM_PRCR_KEY

  Initial value: `(0x78U)`
- #define CGC_SRAM_UNLOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x1U)`
- #define CGC_SRAM_LOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x0U)`

bsp_clock_init

```
bsp_clock_init (  void  )
```

Brief description

Sets up system clocks.

Function steps

- MOCO is default clock out of reset. Enable new clock if chosen. S124 has no PLL.
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- PLL Source clock is always the main oscillator.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- PLL Source clock is always the main oscillator.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- Set PLL Source clock.
- Wait for PLL clock source to stabilize

- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- Set USB clock divisor.
- Configure BCLK
- Configure SDRAM Clock

bsp_cpu_clock_get

```
bsp_cpu_clock_get (  void  )
```

Brief description

Returns frequency of CPU clock in Hz.

Detailed description

Table 2206:Return values

Name	Description
Frequency	of the CPU in Hertz

bsp_clock_set_callback

```
ssp_err_t bsp_clock_set_callback (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings for a requested system clock change (PRE) or a just updated system clock change (POST)

Detailed description

Table 2207:Parameters

Name	Direction	Description
p_args	in	- Pre/Post request and clock frequency information

Table 2208:Return values

Name	Description
return	SSP_SUCCESS

Function steps

- The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0.
- < No MEMWAIT cycles

bsp_clocks_mem_wait_set

```
bsp_clocks_mem_wait_set (  uint32_t setting )
```

Brief description

This function sets the value of the MEMWAIT register which controls wait cycles for flash read access.

Detailed description

Table 2209:Parameters

Name	Direction	Description
setting	in	

Table 2210:Return values

Name	Description
none	

bsp_clocks_mem_wait_get

```
bsp_clocks_mem_wait_get (  void  )
```

Brief description

This function gets the value of the MEMWAIT register.

Detailed description

Table 2211:Return values

Name	Description
MEMWAIT	setting

bsp_clock_set_prechange

```
bsp_clock_set_prechange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the requested clock change to the CGC.

Detailed description

Table 2212:Return values

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK < 120 MHz
- 1 wait: ICLK >= 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- Calculate the Wait states for ROM
- If the requested_frequency <= 80 MHz 0 ROM wait states are required)
- If the 80 MHz < requested_frequency <= 160 MHz 1 ROM wait state is required)
- Anything above 160 MHz requires 2 ROM wait states
- Set the wait state BEFORE we change iclk

bsp_clock_set_postchange

```
bsp_clock_set_postchange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the system clock which has just been set by the CGC.

Detailed description

Table 2213:Return values

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK <= 120 MHz
- 1 wait: ICLK > 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz

- 1 wait: ICLK > 200 MHz
- In this case we need to set the wait state AFTER we change iclk

bsp_clocks_rom_wait_set

```
bsp_clocks_rom_wait_set (  uint8_t setting )
```

Brief description

This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM.

Detailed description

Table 2214:Parameters

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2215:Return values

Name	Description
none	

bsp_clocks_rom_wait_get

```
bsp_clocks_rom_wait_get (  void  )
```

Brief description

This function gets the value of the ROMWT register.

Detailed description

Table 2216:Return values

Name	Description
MEMWAIT	setting

bsp_clocks_sram_wait_set

```
bsp_clocks_sram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for both the SRAM0 and SRAM1 RAM memory.

Detailed description

Table 2217:Parameters

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2218:Return values

Name	Description
none	

bsp_clocks_hsram_wait_set

```
bsp_clocks_hsram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for High Speed RAM memory.

Detailed description

Table 2219:Parameters

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2220:Return values

Name	Description
none	

10.2.6.5 Hardware Locks

This file allocates hardware locks used in Atomic Locking.

Functions

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 0U )
```

Detailed description

Used to allocated hardware locks. Parameters are as follows:

1) IP name (ssp_ip_t enum without the SSP_IP_ prefix).
2) Unit number (used for blocks with variations like USB, not to be confused with ADC unit).
3) Channel number

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AES , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( BSC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CTSU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DOC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ELC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( FCU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IWDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( KEY , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LPM , 1U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( RTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TRNG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TSN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( WDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAAD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 14U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 15U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IRDA , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MMF , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MPU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPAMP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( QSPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCE , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DRW , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( EPTPC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( JPEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( PDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 1U , 0U )
```

10.2.6.6 Module Start and Stop

Module start and stop functions are provided to enable or disable peripherals.

Functions

- r_bsp_module_start_stop
- R_BSP_ModuleStop
- R_BSP_ModuleStart

Data structures

- bsp_mstp_bit_t
- bsp_mstp_data_t

Variables

- g_bsp_module_stop
- gp_mstp

Defines

- #define BSP_MCU_GROUP_S3A1

  Initial value:

- #define BSP_MSTP_DATA_INVALID

  Initial value: `(0xFFU)`
- #define BSP_MSTP_EXCEPTION

  Initial value: `(1U << 5)`
- #define BSP_MSTP_REG

  Initial value: `((x) << 6)`
- #define BSP_MSTP_BIT

  Initial value: `(x)`
- #define BSP_MSTPCRA

  Initial value: `(0U)`
- #define BSP_MSTPCRB

  Initial value: `(1U)`
- #define BSP_MSTPCRC

  Initial value: `(2U)`
- #define BSP_MSTPCRD

  Initial value: `(3U)`
- #define BSP_COMPILE_TIME_ASSERT

  Initial value: `((void) sizeof(char[1 - 2 * !(e)]))`

  Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

g_bsp_module_stop

```
const uint8_t g_bsp_module_stop[]
```

Detailed description

Module stop bit definitions for each ssp_ip_t enum value.

gp_mstp

```
volatile uint32_t* const gp_mstp[]
```

Detailed description

Module stop register addresses.

r_bsp_module_start_stop

```
ssp_err_t r_bsp_module_start_stop (  ssp_feature_t const *const p_feature ,
  bool stop )
```

Brief description

Start module (cancel module stop) or stop module (enter module stop).

Detailed description

NOTE: Some module stop bits are shared between peripherals. Entering module stop is not permitted for these peripherals.

Table 2221:Parameters

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.
stop	in	true to stop module, false to start module

Table 2222:Return values

Name	Description
SSP_SUCCESS	Module is stopped or started based on stop parameter
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

Function steps

- The g_bsp_module_stop array must have entries for each ssp_ip_t enum value.
- Set MSTP bit if stop parameter is true and the bit is not shared with other channels.
- Clear MSTP bit if stop parameter is false.

R_BSP_ModuleStop

```
ssp_err_t R_BSP_ModuleStop (  ssp_feature_t const *const p_feature )
```

Brief description

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Detailed description

NOTE: Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Table 2223:Parameters

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2224:Return values

Name	Description
SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

R_BSP_ModuleStart

```
ssp_err_t R_BSP_ModuleStart (  ssp_feature_t const *const p_feature )
```

Brief description

Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Detailed description

Table 2225:Parameters

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2226:Return values

Name	Description
SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

bsp_mstp_bit_t

bsp_mstp_bit_t

Detailed description

Definition of BSP module stop bit.

Variables

- uint8_t bit

  Which bit in MSTP register.

- uint8_t reg

  < Special processing required

  Which MSTP register

bsp_mstp_data_t

Detailed description

Definition of BSP module stop bit.

Variables

byte

bit

10.2.6.7 ROM Registers

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Functions

- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2

Defines

- #define BSP_ROM_REG_OFS1_SETTING

  Initial value:`(((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))`

  OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

- #define BSP_ROM_REG_MPU_CONTROL_SETTING

  Initial value:`((0xFFFFFCFEU) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABLE << 8) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABLE << 9) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_ENABLE))`

  Build up SECMPUAC register based on MPU settings.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ROM_REGISTERS  )
```

Detailed description

ROM registers defined here. Some have masks to make sure reserved bits are set appropriately. S124 has no MPU.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_1  )
```

Detailed description

ID code definitions defined here.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_2  )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_3  )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_4  )
```

10.2.7 S3A3

Code that is common to S3A3 MCUs.

Implements functions that are common to S3A3 MCUs.

10.2.7.1 Cache Functions

This module implements cache functions.

Typedefs

- bsp_cache_frequency_t

bsp_cache_state_t

Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

bsp_cache_frequency_t

```
typedef uint32_t bsp_cache_frequency_t
```

10.2.7.2 Clock Initialization

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Functions

- bsp_clock_init
- bsp_cpu_clock_get
- bsp_clock_set_callback
- bsp_clocks_mem_wait_set
- bsp_clocks_mem_wait_get
- bsp_clock_set_prechange
- bsp_clock_set_postchange
- bsp_clocks_rom_wait_set
- bsp_clocks_rom_wait_get
- bsp_clocks_sram_wait_set
- bsp_clocks_hsram_wait_set

Defines

- #define CGC_SRAM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for SRAM.
- #define CGC_SRAM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for SRAM.
- #define CGC_ZERO_WAIT_CYCLES

  Initial value: `(0U)`
- #define CGC_TWO_WAIT_CYCLES

  Initial value: `(1U)`
- #define CGC_MAX_ZERO_WAIT_FREQ

  Initial value: `(32000000U)`
- #define CGC_PRCR_KEY

  Initial value: `(0xA500U)`
- #define CGC_PRCR_UNLOCK

  Initial value: `((CGC_PRCR_KEY) | 0x1U)`
- #define CGC_PRCR_LOCK

  Initial value: `((CGC_PRCR_KEY) | 0x0U)`

- #define CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS

  Initial value: `(120000000U)`
- #define CGC_SYS_CLOCK_FREQ_NO_HSRAM_WAITS

  Initial value: `(200000000U)`
- #define CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS

  Initial value: `(80000000U)`
- #define CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS

  Initial value: `(160000000U)`
- #define CGC_ROM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for ROM.
- #define CGC_ROM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for ROM.
- #define CGC_ROM_TWO_WAIT_CYCLES

  Initial value: `(2U)`

  Specify two wait states for ROM.
- #define CGC_SRAM_PRCR_KEY

  Initial value: `(0x78U)`
- #define CGC_SRAM_UNLOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x1U)`
- #define CGC_SRAM_LOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x0U)`

bsp_clock_init

```
bsp_clock_init (  void  )
```

Brief description

Sets up system clocks.

Function steps

- MOCO is default clock out of reset. Enable new clock if chosen. S124 has no PLL.
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.

- PLL Source clock is always the main oscillator.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- PLL Source clock is always the main oscillator.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- Set PLL Source clock.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- Set USB clock divisor.
- Configure BCLK
- Configure SDRAM Clock

bsp_cpu_clock_get

```
bsp_cpu_clock_get (  void  )
```

Brief description

Returns frequency of CPU clock in Hz.

Detailed description

Table 2227:Return values

Name	Description
Frequency	of the CPU in Hertz

bsp_clock_set_callback

```
ssp_err_t bsp_clock_set_callback (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings for a requested system clock change (PRE) or a just updated system clock change (POST)

Detailed description

Table 2228:Parameters

Name	Direction	Description
p_args	in	- Pre/Post request and clock frequency information

Table 2229:Return values

Name	Description
return	SSP_SUCCESS

Function steps

- The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0.
- < No MEMWAIT cycles

bsp_clocks_mem_wait_set

```
bsp_clocks_mem_wait_set (  uint32_t setting )
```

Brief description

This function sets the value of the MEMWAIT register which controls wait cycles for flash read access.

Detailed description

Table 2230:Parameters

Name	Direction	Description
setting	in	

Table 2231:Return values

Name	Description
none	

bsp_clocks_mem_wait_get

```
bsp_clocks_mem_wait_get (  void  )
```

Brief description

This function gets the value of the MEMWAIT register.

Detailed description

Table 2232:Return values

Name	Description
MEMWAIT	setting

bsp_clock_set_prechange

```
bsp_clock_set_prechange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the requested clock change to the CGC.

Detailed description

Table 2233:Return values

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK < 120 MHz
- 1 wait: ICLK >= 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- Calculate the Wait states for ROM
- If the requested_frequency <= 80 MHz 0 ROM wait states are required)
- If the 80 MHz < requested_frequency <= 160 MHz 1 ROM wait state is required)
- Anything above 160 MHz requires 2 ROM wait states
- Set the wait state BEFORE we change iclk

bsp_clock_set_postchange

```
bsp_clock_set_postchange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the system clock which has just been set by the CGC.

Detailed description

Table 2234:Return values

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK <= 120 MHz
- 1 wait: ICLK > 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- In this case we need to set the wait state AFTER we change iclk

bsp_clocks_rom_wait_set

```
bsp_clocks_rom_wait_set (  uint8_t setting )
```

Brief description

This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM.

Detailed description

Table 2235:Parameters

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2236:Return values

Name	Description
none	

bsp_clocks_rom_wait_get

```
bsp_clocks_rom_wait_get (  void  )
```

Brief description

This function gets the value of the ROMWT register.

Detailed description

Table 2237:Return values

Name	Description
MEMWAIT	setting

bsp_clocks_sram_wait_set

```
bsp_clocks_sram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for both the SRAM0 and SRAM1 RAM memory.

Detailed description

Table 2238:Parameters

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2239:Return values

Name	Description
none	

bsp_clocks_hsram_wait_set

```
bsp_clocks_hsram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for High Speed RAM memory.

Detailed description

Table 2240:Parameters

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2241:Return values

Name	Description
none	

10.2.7.3 Hardware Locks

This file allocates hardware locks used in Atomic Locking.

Functions

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 0U )
```

Detailed description

Used to allocated hardware locks. Parameters are as follows:

1) IP name (ssp_ip_t enum without the SSP_IP_ prefix).
2) Unit number (used for blocks with variations like USB, not to be confused with ADC unit).
3) Channel number

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AES , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( BSC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CTSU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DOC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ELC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( FCU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IWDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( KEY , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LPM , 1U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( RTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TRNG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TSN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( WDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAAD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 14U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 15U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IRDA , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MMF , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MPU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPAMP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( QSPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCE , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DRW , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( EPTPC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( JPEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( PDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 1U , 0U )
```

10.2.7.4 Module Start and Stop

Module start and stop functions are provided to enable or disable peripherals.

Functions

- r_bsp_module_start_stop
- R_BSP_ModuleStop
- R_BSP_ModuleStart

Data structures

- bsp_mstp_bit_t
- bsp_mstp_data_t

Variables

- g_bsp_module_stop
- gp_mstp

Defines

- #define BSP_MCU_GROUP_S3A3

  Initial value:
- #define BSP_MSTP_DATA_INVALID

  Initial value: `(0xFFU)`
- #define BSP_MSTP_EXCEPTION

  Initial value: `(1U << 5)`
- #define BSP_MSTP_REG

  Initial value: `((x) << 6)`
- #define BSP_MSTP_BIT

  Initial value: `(x)`
- #define BSP_MSTPCRA

  Initial value: `(0U)`
- #define BSP_MSTPCRB

  Initial value: `(1U)`
- #define BSP_MSTPCRC

  Initial value: `(2U)`
- #define BSP_MSTPCRD

  Initial value: `(3U)`
- #define BSP_COMPILE_TIME_ASSERT

  Initial value: `((void) sizeof(char[1 - 2 * !(e)]))`

  Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

g_bsp_module_stop

```
const uint8_t g_bsp_module_stop[]
```

Detailed description

Module stop bit definitions for each ssp_ip_t enum value.

gp_mstp

```
volatile uint32_t* const gp_mstp[]
```

Detailed description

Module stop register addresses.

r_bsp_module_start_stop

```
ssp_err_t r_bsp_module_start_stop (  ssp_feature_t const *const p_feature ,
  bool stop )
```

Brief description

Start module (cancel module stop) or stop module (enter module stop).

Detailed description

NOTE: Some module stop bits are shared between peripherals. Entering module stop is not permitted for these peripherals.

Table 2242:Parameters

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.
stop	in	true to stop module, false to start module

Table 2243:Return values

Name	Description
SSP_SUCCESS	Module is stopped or started based on stop parameter
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

Function steps

- The g_bsp_module_stop array must have entries for each ssp_ip_t enum value.
- Set MSTP bit if stop parameter is true and the bit is not shared with other channels.
- Clear MSTP bit if stop parameter is false.

R_BSP_ModuleStop

```
ssp_err_t R_BSP_ModuleStop (  ssp_feature_t const *const p_feature )
```

Brief description

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Detailed description

NOTE: Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Table 2244:Parameters

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2245:Return values

Name	Description
SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

R_BSP_ModuleStart

```
ssp_err_t R_BSP_ModuleStart (  ssp_feature_t const *const p_feature )
```

Brief description

Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Detailed description

Table 2246:Parameters

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2247:Return values

Name	Description
SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

bsp_mstp_bit_t

bsp_mstp_bit_t

Detailed description

Definition of BSP module stop bit.

Variables

- uint8_t bit

  Which bit in MSTP register.

- uint8_t reg

  < Special processing required

  Which MSTP register

bsp_mstp_data_t

Detailed description

Definition of BSP module stop bit.

Variables

byte

bit

10.2.7.5 ROM Registers

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Functions

- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2

Defines

- #define BSP_ROM_REG_OFS1_SETTING

  Initial value:
  ```
  (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) |
  ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))
  ```

  OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

- #define BSP_ROM_REG_MPU_CONTROL_SETTING

  Initial value:
  ```
  ((0xFFFFFCFEU) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABLE << 8) |
  \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABLE << 9) | \
  ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_ENABLE))
  ```

  Build up SECMPUAC register based on MPU settings.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ROM_REGISTERS )
```

Detailed description

ROM registers defined here. Some have masks to make sure reserved bits are set appropriately. S124 has no MPU.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ID_CODE_1 )
```

Detailed description

ID code definitions defined here.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ID_CODE_2 )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ID_CODE_3 )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ID_CODE_4 )
```

10.2.8 S3A6

Code that is common to S3A6 MCUs.

Implements functions that are common to S3A6 MCUs.

10.2.8.1 Cache Functions

This module implements cache functions.

Typedefs

- bsp_cache_frequency_t

bsp_cache_state_t

Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

bsp_cache_frequency_t

```
typedef uint32_t bsp_cache_frequency_t
```

10.2.8.2 Clock Initialization

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Functions

- bsp_clock_init
- bsp_cpu_clock_get
- bsp_clock_set_callback
- bsp_clocks_mem_wait_set
- bsp_clocks_mem_wait_get
- bsp_clock_set_prechange
- bsp_clock_set_postchange
- bsp_clocks_rom_wait_set
- bsp_clocks_rom_wait_get
- bsp_clocks_sram_wait_set
- bsp_clocks_hsram_wait_set

Defines

- #define CGC_SRAM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for SRAM.
- #define CGC_SRAM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for SRAM.

- #define CGC_ZERO_WAIT_CYCLES

  Initial value: `(0U)`
- #define CGC_TWO_WAIT_CYCLES

  Initial value: `(1U)`
- #define CGC_MAX_ZERO_WAIT_FREQ

  Initial value: `(32000000U)`
- #define CGC_PRCR_KEY

  Initial value: `(0xA500U)`
- #define CGC_PRCR_UNLOCK

  Initial value: `((CGC_PRCR_KEY) | 0x1U)`
- #define CGC_PRCR_LOCK

  Initial value: `((CGC_PRCR_KEY) | 0x0U)`
- #define CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS

  Initial value: `(120000000U)`
- #define CGC_SYS_CLOCK_FREQ_NO_HSRAM_WAITS

  Initial value: `(200000000U)`
- #define CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS

  Initial value: `(80000000U)`
- #define CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS

  Initial value: `(160000000U)`
- #define CGC_ROM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for ROM.
- #define CGC_ROM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for ROM.
- #define CGC_ROM_TWO_WAIT_CYCLES

  Initial value: `(2U)`

  Specify two wait states for ROM.
- #define CGC_SRAM_PRCR_KEY

  Initial value: `(0x78U)`
- #define CGC_SRAM_UNLOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x1U)`

- #define CGC_SRAM_LOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x0U)`

bsp_clock_init

```
bsp_clock_init (  void  )
```

Brief description

Sets up system clocks.

Function steps

- MOCO is default clock out of reset. Enable new clock if chosen. S124 has no PLL.
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- PLL Source clock is always the main oscillator.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- PLL Source clock is always the main oscillator.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- Set PLL Source clock.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- Set USB clock divisor.
- Configure BCLK
- Configure SDRAM Clock

bsp_cpu_clock_get

```
bsp_cpu_clock_get ( void )
```

Brief description

Returns frequency of CPU clock in Hz.

Detailed description

Table 2248:Return values

Name	Description
Frequency	of the CPU in Hertz

bsp_clock_set_callback

```
ssp_err_t bsp_clock_set_callback ( bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings for a requested system clock change (PRE) or a just updated system clock change (POST)

Detailed description

Table 2249:Parameters

Name	Direction	Description
p_args	in	- Pre/Post request and clock frequency information

Table 2250:

Name	Description
return	SSP_SUCCESS

Function steps

- The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0.
- < No MEMWAIT cycles

bsp_clocks_mem_wait_set

```
bsp_clocks_mem_wait_set ( uint32_t setting )
```

Brief description

This function sets the value of the MEMWAIT register which controls wait cycles for flash read access.

Detailed description

Table 2251:

Name	Direction	Description
setting	in	

Table 2252:

Name	Description
none	

bsp_clocks_mem_wait_get

```
bsp_clocks_mem_wait_get (  void  )
```

Brief description

This function gets the value of the MEMWAIT register.

Detailed description

Table 2253:

Name	Description
MEMWAIT	setting

bsp_clock_set_prechange

```
bsp_clock_set_prechange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the requested clock change to the CGC.

Detailed description

Table 2254:

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)

- No wait: ICLK < 120 MHz
- 1 wait: ICLK >= 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- Calculate the Wait states for ROM
- If the requested_frequency <= 80 MHz 0 ROM wait states are required)
- If the 80 MHz < requested_frequency <= 160 MHz 1 ROM wait state is required)
- Anything above 160 MHz requires 2 ROM wait states
- Set the wait state BEFORE we change iclk

bsp_clock_set_postchange

```
bsp_clock_set_postchange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the system clock which has just been set by the CGC.

Detailed description

Table 2255:

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK <= 120 MHz
- 1 wait: ICLK > 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- In this case we need to set the wait state AFTER we change iclk

bsp_clocks_rom_wait_set

```
bsp_clocks_rom_wait_set (  uint8_t setting )
```

Brief description

This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM.

Detailed description

Table 2256:

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2257:

Name	Description
none	

bsp_clocks_rom_wait_get

```
bsp_clocks_rom_wait_get (  void  )
```

Brief description

This function gets the value of the ROMWT register.

Detailed description

Table 2258:

Name	Description
MEMWAIT	setting

bsp_clocks_sram_wait_set

```
bsp_clocks_sram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for both the SRAM0 and SRAM1 RAM memory.

Detailed description

Table 2259:

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2260:

Name	Description
none	

bsp_clocks_hsram_wait_set

```
bsp_clocks_hsram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for High Speed RAM memory.

Detailed description

Table 2261:

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2262:

Name	Description
none	

10.2.8.3 Hardware Locks

This file allocates hardware locks used in Atomic Locking.

Functions

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 0U )
```

Detailed description

Used to allocated hardware locks. Parameters are as follows:

1) IP name (ssp_ip_t enum without the SSP_IP_ prefix).
2) Unit number (used for blocks with variations like USB, not to be confused with ADC unit).
3) Channel number

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AES , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( BSC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CTSU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DOC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ELC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( FCU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IWDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( KEY , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LPM , 1U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( RTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TRNG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TSN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( WDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAAD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 14U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 15U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IRDA , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MMF , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MPU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPAMP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( QSPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCE , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DRW , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( EPTPC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( JPEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( PDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 1U , 0U )
```

10.2.8.4 Module Start and Stop

Module start and stop functions are provided to enable or disable peripherals.

Functions

- r_bsp_module_start_stop
- calc_mask
- R_BSP_ModuleStop
- R_BSP_ModuleStart

Data structures

- bsp_mstp_bit_t
- bsp_mstp_data_t

Variables

- g_bsp_module_stop
- gp_mstp

Defines

- #define BSP_MCU_GROUP_S3A6

  Initial value:
- #define BSP_MSTP_DATA_INVALID

  Initial value: `(0xFFU)`
- #define BSP_MSTP_EXCEPTION

  Initial value: `(1U << 5)`

- #define BSP_MSTP_REG

  Initial value: `((x) << 6)`
- #define BSP_MSTP_BIT

  Initial value: `(x)`
- #define BSP_MSTPCRA

  Initial value: `(0U)`
- #define BSP_MSTPCRB

  Initial value: `(1U)`
- #define BSP_MSTPCRC

  Initial value: `(2U)`
- #define BSP_MSTPCRD

  Initial value: `(3U)`
- #define BSP_COMPILE_TIME_ASSERT

  Initial value: `((void) sizeof(char[1 - 2 * !(e)]))`

  Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

g_bsp_module_stop

```
const uint8_t g_bsp_module_stop[]
```

Detailed description

Module stop bit definitions for each ssp_ip_t enum value.

gp_mstp

```
volatile uint32_t* const gp_mstp[]
```

Detailed description

Module stop register addresses.

r_bsp_module_start_stop

```
ssp_err_t r_bsp_module_start_stop (  ssp_feature_t const *const p_feature ,
  bool stop )
```

Brief description

Start module (cancel module stop) or stop module (enter module stop).

Detailed description

NOTE: Some module stop bits are shared between peripherals. Entering module stop is not permitted for these peripherals.

Table 2263:

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.
stop	in	true to stop module, false to start module

Table 2264:

Name	Description
SSP_SUCCESS	Module is stopped or started based on stop parameter
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

Function steps

- The g_bsp_module_stop array must have entries for each ssp_ip_t enum value.
- Set MSTP bit if stop parameter is true and the bit is not shared with other channels.
- Clear MSTP bit if stop parameter is false.

calc_mask

```
 calc_mask (  ssp_feature_t const *const p_feature ,   uint8_t data ,   uint32_t
* mask )
```

Brief description

Calculate the mask that will be used with the mstp base register used by the calling function.

R_BSP_ModuleStop

```
ssp_err_t R_BSP_ModuleStop (  ssp_feature_t const *const p_feature )
```

Brief description

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Detailed description

NOTE: Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Table 2265:

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2266:

Name	Description
SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

R_BSP_ModuleStart

```
ssp_err_t R_BSP_ModuleStart (  ssp_feature_t const *const p_feature )
```

Brief description

Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Detailed description

Table 2267:

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2268:

Name	Description
SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid

Table 2268: (Continued)

Name	Description
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

bsp_mstp_bit_t

bsp_mstp_bit_t

Detailed description

Definition of BSP module stop bit.

Variables

- uint8_t bit

  Which bit in MSTP register.

- uint8_t reg

  < Special processing required

  Which MSTP register

bsp_mstp_data_t

Detailed description

Definition of BSP module stop bit.

Variables

byte

bit

10.2.8.5 ROM Registers

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Functions

- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2

Defines

- #define BSP_ROM_REG_OFS1_SETTING

  Initial value: `(((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))`

  OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

- #define BSP_ROM_REG_MPU_CONTROL_SETTING

  Initial value: `((0xFFFFFCFEU) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABLE << 8) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABLE << 9) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_ENABLE))`

  Build up SECMPUAC register based on MPU settings.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ROM_REGISTERS  )
```

Detailed description

ROM registers defined here. Some have masks to make sure reserved bits are set appropriately. S124 has no MPU.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_1  )
```

Detailed description

ID code definitions defined here.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_2  )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_3  )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_4  )
```

10.2.9 S3A7

Code that is common to S3A7 MCUs.

Implements functions that are common to S3A7 MCUs.

10.2.9.1 elc_peripheral_t

Possible peripherals to be linked to event signals

10.2.9.2 elc_event_t

Sources of event signals to be linked to other peripherals or the CPU1
NOTE: This list may change based on device. This list is for S3A7.

10.2.9.3 Cache Functions

This module implements cache functions.

Typedefs

- bsp_cache_frequency_t

bsp_cache_state_t

Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

bsp_cache_frequency_t

```
typedef uint32_t bsp_cache_frequency_t
```

10.2.9.4 Clock Initialization

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Functions

- bsp_clock_init
- bsp_cpu_clock_get
- bsp_clock_set_callback
- bsp_clocks_mem_wait_set
- bsp_clocks_mem_wait_get
- bsp_clock_set_prechange
- bsp_clock_set_postchange
- bsp_clocks_rom_wait_set
- bsp_clocks_rom_wait_get
- bsp_clocks_sram_wait_set
- bsp_clocks_hsram_wait_set

Defines

- #define CGC_SRAM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for SRAM.

- #define CGC_SRAM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for SRAM.
- #define CGC_ZERO_WAIT_CYCLES

  Initial value: `(0U)`
- #define CGC_TWO_WAIT_CYCLES

  Initial value: `(1U)`
- #define CGC_MAX_ZERO_WAIT_FREQ

  Initial value: `(32000000U)`
- #define CGC_PRCR_KEY

  Initial value: `(0xA500U)`
- #define CGC_PRCR_UNLOCK

  Initial value: `((CGC_PRCR_KEY) | 0x1U)`
- #define CGC_PRCR_LOCK

  Initial value: `((CGC_PRCR_KEY) | 0x0U)`
- #define CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS

  Initial value: `(120000000U)`
- #define CGC_SYS_CLOCK_FREQ_NO_HSRAM_WAITS

  Initial value: `(200000000U)`
- #define CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS

  Initial value: `(80000000U)`
- #define CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS

  Initial value: `(160000000U)`
- #define CGC_ROM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for ROM.
- #define CGC_ROM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for ROM.
- #define CGC_ROM_TWO_WAIT_CYCLES

  Initial value: `(2U)`

  Specify two wait states for ROM.

- #define CGC_SRAM_PRCR_KEY

  Initial value: `(0x78U)`
- #define CGC_SRAM_UNLOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x1U)`
- #define CGC_SRAM_LOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x0U)`

bsp_clock_init

```
bsp_clock_init (  void  )
```

Brief description

Sets up system clocks.

Function steps

- MOCO is default clock out of reset. Enable new clock if chosen. S124 has no PLL.
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- PLL Source clock is always the main oscillator.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- PLL Source clock is always the main oscillator.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- Set PLL Source clock.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.

- Set USB clock divisor.
- Configure BCLK
- Configure SDRAM Clock

bsp_cpu_clock_get

```
bsp_cpu_clock_get (  void  )
```

Brief description

Returns frequency of CPU clock in Hz.

Detailed description

Table 2269:

Name	Description
Frequency	of the CPU in Hertz

bsp_clock_set_callback

```
ssp_err_t bsp_clock_set_callback (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings for a requested system clock change (PRE) or a just updated system clock change (POST)

Detailed description

Table 2270:

Name	Direction	Description
p_args	in	- Pre/Post request and clock frequency information

Table 2271:

Name	Description
return	SSP_SUCCESS

Function steps

- The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0.
- < No MEMWAIT cycles

bsp_clocks_mem_wait_set

```
bsp_clocks_mem_wait_set ( uint32_t setting )
```

Brief description

This function sets the value of the MEMWAIT register which controls wait cycles for flash read access.

Detailed description

Table 2272:

Name	Direction	Description
setting	in	

Table 2273:

Name	Description
none	

bsp_clocks_mem_wait_get

```
bsp_clocks_mem_wait_get ( void )
```

Brief description

This function gets the value of the MEMWAIT register.

Detailed description

Table 2274:

Name	Description
MEMWAIT	setting

bsp_clock_set_prechange

```
bsp_clock_set_prechange ( bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the requested clock change to the CGC.

Detailed description

Table 2275:

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK < 120 MHz
- 1 wait: ICLK >= 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- Calculate the Wait states for ROM
- If the requested_frequency <= 80 MHz 0 ROM wait states are required)
- If the 80 MHz < requested_frequency <= 160 MHz 1 ROM wait state is required)
- Anything above 160 MHz requires 2 ROM wait states
- Set the wait state BEFORE we change iclk

bsp_clock_set_postchange

```
bsp_clock_set_postchange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the system clock which has just been set by the CGC.

Detailed description

Table 2276:

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK <= 120 MHz
- 1 wait: ICLK > 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz

- 1 wait: ICLK > 200 MHz
- In this case we need to set the wait state AFTER we change iclk

bsp_clocks_rom_wait_set

```
bsp_clocks_rom_wait_set (  uint8_t setting )
```

Brief description

This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM.

Detailed description

Table 2277:

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2278:

Name	Description
none	

bsp_clocks_rom_wait_get

```
bsp_clocks_rom_wait_get (  void  )
```

Brief description

This function gets the value of the ROMWT register.

Detailed description

Table 2279:

Name	Description
MEMWAIT	setting

bsp_clocks_sram_wait_set

```
bsp_clocks_sram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for both the SRAM0 and SRAM1 RAM memory.

Detailed description

Table 2280:

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2281:

Name	Description
none	

bsp_clocks_hsram_wait_set

```
bsp_clocks_hsram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for High Speed RAM memory.

Detailed description

Table 2282:

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2283:

Name	Description
none	

10.2.9.5 Hardware Locks

This file allocates hardware locks used in Atomic Locking.

Functions

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 0U )
```

Detailed description

Used to allocated hardware locks. Parameters are as follows:

1) IP name (ssp_ip_t enum without the SSP_IP_ prefix).
2) Unit number (used for blocks with variations like USB, not to be confused with ADC unit).
3) Channel number

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AES , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( BSC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CTSU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DOC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ELC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( FCU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IWDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( KEY , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LPM , 1U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( RTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TRNG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TSN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( WDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAAD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 14U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 15U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IRDA , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MMF , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MPU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPAMP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( QSPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCE , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DRW , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( EPTPC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( JPEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( PDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 1U , 0U )
```

10.2.9.6 Module Start and Stop

Module start and stop functions are provided to enable or disable peripherals.

Functions

- r_bsp_module_start_stop
- R_BSP_ModuleStop
- R_BSP_ModuleStart

Data structures

- bsp_mstp_bit_t
- bsp_mstp_data_t

Variables

- g_bsp_module_stop
- gp_mstp

Defines

- #define BSP_MCU_GROUP_S3A7

  Initial value:

- #define BSP_MSTP_DATA_INVALID

  Initial value: `(0xFFU)`
- #define BSP_MSTP_EXCEPTION

  Initial value: `(1U << 5)`
- #define BSP_MSTP_REG

  Initial value: `((x) << 6)`
- #define BSP_MSTP_BIT

  Initial value: `(x)`
- #define BSP_MSTPCRA

  Initial value: `(0U)`
- #define BSP_MSTPCRB

  Initial value: `(1U)`
- #define BSP_MSTPCRC

  Initial value: `(2U)`
- #define BSP_MSTPCRD

  Initial value: `(3U)`
- #define BSP_COMPILE_TIME_ASSERT

  Initial value: `((void) sizeof(char[1 - 2 * !(e)]))`

  Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

g_bsp_module_stop

```
const uint8_t g_bsp_module_stop[]
```

Detailed description

Module stop bit definitions for each ssp_ip_t enum value.

gp_mstp

```
volatile uint32_t* const gp_mstp[]
```

Detailed description

Module stop register addresses.

r_bsp_module_start_stop

```
ssp_err_t r_bsp_module_start_stop (  ssp_feature_t const *const p_feature ,
  bool stop )
```

Brief description

Start module (cancel module stop) or stop module (enter module stop).

Detailed description

NOTE: Some module stop bits are shared between peripherals. Entering module stop is not permitted for these peripherals.

Table 2284:

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.
stop	in	true to stop module, false to start module

Table 2285:

Name	Description
SSP_SUCCESS	Module is stopped or started based on stop parameter
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

Function steps

- The g_bsp_module_stop array must have entries for each ssp_ip_t enum value.
- Set MSTP bit if stop parameter is true and the bit is not shared with other channels.
- Clear MSTP bit if stop parameter is false.

R_BSP_ModuleStop

```
ssp_err_t R_BSP_ModuleStop (  ssp_feature_t const *const p_feature )
```

Brief description

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Detailed description

NOTE: Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Table 2286:

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2287:

Name	Description
SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

R_BSP_ModuleStart

```
ssp_err_t R_BSP_ModuleStart (  ssp_feature_t const *const p_feature )
```

Brief description

Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Detailed description

Table 2288:

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2289:

Name	Description
SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

bsp_mstp_bit_t

bsp_mstp_bit_t

Detailed description

Definition of BSP module stop bit.

Variables

- uint8_t bit

  Which bit in MSTP register.

- uint8_t reg

  < Special processing required

  Which MSTP register

bsp_mstp_data_t

Detailed description

Definition of BSP module stop bit.

Variables

byte

bit

10.2.9.7 ROM Registers

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Functions

- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2

Defines

- #define BSP_ROM_REG_OFS1_SETTING

  Initial value:`(((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))`

  OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

- #define BSP_ROM_REG_MPU_CONTROL_SETTING

  Initial value:
  ```
  ((0xFFFFFCFEU) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABLE << 8) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABLE << 9) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_ENABLE))
  ```

  Build up SECMPUAC register based on MPU settings.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ROM_REGISTERS  )
```

Detailed description

ROM registers defined here. Some have masks to make sure reserved bits are set appropriately. S124 has no MPU.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_1  )
```

Detailed description

ID code definitions defined here.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_2  )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_3  )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_4  )
```

10.2.10 S5D5

Code that is common to S5D5 MCUs.

Implements functions that are common to S5D5 MCUs.

10.2.10.1 Cache Functions

This module implements cache functions.

bsp_cache_state_t

Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

10.2.10.2 Clock Initialization

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Functions

- bsp_clock_init
- bsp_cpu_clock_get
- bsp_clock_set_callback
- bsp_clocks_mem_wait_set
- bsp_clocks_mem_wait_get
- bsp_clock_set_prechange
- bsp_clock_set_postchange
- bsp_clocks_rom_wait_set
- bsp_clocks_rom_wait_get
- bsp_clocks_sram_wait_set
- bsp_clocks_hsram_wait_set

Defines

- #define CGC_SRAM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for SRAM.
- #define CGC_SRAM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for SRAM.
- #define CGC_ZERO_WAIT_CYCLES

  Initial value: `(0U)`
- #define CGC_TWO_WAIT_CYCLES

  Initial value: `(1U)`
- #define CGC_MAX_ZERO_WAIT_FREQ

  Initial value: `(32000000U)`
- #define CGC_PRCR_KEY

  Initial value: `(0xA500U)`
- #define CGC_PRCR_UNLOCK

  Initial value: `((CGC_PRCR_KEY) | 0x1U)`
- #define CGC_PRCR_LOCK

  Initial value: `((CGC_PRCR_KEY) | 0x0U)`
- #define CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS

  Initial value: `(120000000U)`

- #define CGC_SYS_CLOCK_FREQ_NO_HSRAM_WAITS

  Initial value: `(200000000U)`
- #define CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS

  Initial value: `(80000000U)`
- #define CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS

  Initial value: `(160000000U)`
- #define CGC_ROM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for ROM.
- #define CGC_ROM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for ROM.
- #define CGC_ROM_TWO_WAIT_CYCLES

  Initial value: `(2U)`

  Specify two wait states for ROM.
- #define CGC_SRAM_PRCR_KEY

  Initial value: `(0x78U)`
- #define CGC_SRAM_UNLOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x1U)`
- #define CGC_SRAM_LOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x0U)`

bsp_clock_init

```
bsp_clock_init (  void  )
```

Brief description

Sets up system clocks.

Function steps

- MOCO is default clock out of reset. Enable new clock if chosen. S124 has no PLL.
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- PLL Source clock is always the main oscillator.
- Need to start PLL source clock and let it stabilize before starting PLL

- Set PLL Divider.
- Set PLL Multiplier.
- PLL Source clock is always the main oscillator.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- Set PLL Source clock.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- Set USB clock divisor.
- Configure BCLK
- Configure SDRAM Clock

bsp_cpu_clock_get

```
bsp_cpu_clock_get (  void  )
```

Brief description

Returns frequency of CPU clock in Hz.

Detailed description

Table 2290:

Name	Description
Frequency	of the CPU in Hertz

bsp_clock_set_callback

```
ssp_err_t bsp_clock_set_callback (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings for a requested system clock change (PRE) or a just updated system clock change (POST)

Detailed description

Table 2291:

Name	Direction	Description
p_args	in	- Pre/Post request and clock frequency information

Table 2292:

Name	Description
return	SSP_SUCCESS

Function steps

- The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0.
- < No MEMWAIT cycles

bsp_clocks_mem_wait_set

```
bsp_clocks_mem_wait_set (  uint32_t setting )
```

Brief description

This function sets the value of the MEMWAIT register which controls wait cycles for flash read access.

Detailed description

Table 2293:

Name	Direction	Description
setting	in	

Table 2294:

Name	Description
none	

bsp_clocks_mem_wait_get

```
bsp_clocks_mem_wait_get (  void  )
```

Brief description

This function gets the value of the MEMWAIT register.

Detailed description

Table 2295:

Name	Description
MEMWAIT	setting

bsp_clock_set_prechange

```
bsp_clock_set_prechange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the requested clock change to the CGC.

Detailed description

Table 2296:

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK < 120 MHz
- 1 wait: ICLK >= 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- Calculate the Wait states for ROM
- If the requested_frequency <= 80 MHz 0 ROM wait states are required)
- If the 80 MHz < requested_frequency <= 160 MHz 1 ROM wait state is required)
- Anything above 160 MHz requires 2 ROM wait states
- Set the wait state BEFORE we change iclk

bsp_clock_set_postchange

```
bsp_clock_set_postchange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the system clock which has just been set by the CGC.

Detailed description

Table 2297:

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK <= 120 MHz
- 1 wait: ICLK > 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- In this case we need to set the wait state AFTER we change iclk

bsp_clocks_rom_wait_set

```
bsp_clocks_rom_wait_set (  uint8_t setting )
```

Brief description

This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM.

Detailed description

Table 2298:

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2299:

Name	Description
none	

bsp_clocks_rom_wait_get

```
bsp_clocks_rom_wait_get (  void  )
```

Brief description

This function gets the value of the ROMWT register.

Detailed description

Table 2300:

Name	Description
MEMWAIT	setting

bsp_clocks_sram_wait_set

```
bsp_clocks_sram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for both the SRAM0 and SRAM1 RAM memory.

Detailed description

Table 2301:

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2302:

Name	Description
none	

bsp_clocks_hsram_wait_set

```
bsp_clocks_hsram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for High Speed RAM memory.

Detailed description

Table 2303:

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2304:

Name	Description
none	

10.2.10.3 Hardware Locks

This file allocates hardware locks used in Atomic Locking.

Functions

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 0U )
```

Detailed description

Used to allocated hardware locks. Parameters are as follows:

1) IP name (ssp_ip_t enum without the SSP_IP_ prefix).
2) Unit number (used for blocks with variations like USB, not to be confused with ADC unit).
3) Channel number

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AES , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( BSC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CTSU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DOC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ELC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( FCU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IWDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( KEY , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LPM , 1U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( RTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TRNG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TSN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( WDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAAD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 14U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 15U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IRDA , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MMF , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MPU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPAMP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( QSPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCE , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DRW , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( EPTPC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( JPEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( PDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 1U , 0U )
```

10.2.10.4 Module Start and Stop

Module start and stop functions are provided to enable or disable peripherals.

Functions

- r_bsp_module_start_stop
- R_BSP_ModuleStop
- R_BSP_ModuleStart

Data structures

- bsp_mstp_bit_t
- bsp_mstp_data_t

Variables

- g_bsp_module_stop
- gp_mstp

Defines

- #define BSP_MCU_GROUP_S5D5

  Initial value:
- #define BSP_MSTP_DATA_INVALID

  Initial value: `(0xFFU)`
- #define BSP_MSTP_EXCEPTION

  Initial value: `(1U << 5)`
- #define BSP_MSTP_REG

  Initial value: `((x) << 6)`
- #define BSP_MSTP_BIT

  Initial value: `(x)`
- #define BSP_MSTPCRA

  Initial value: `(0U)`
- #define BSP_MSTPCRB

  Initial value: `(1U)`
- #define BSP_MSTPCRC

  Initial value: `(2U)`
- #define BSP_MSTPCRD

  Initial value: `(3U)`
- #define BSP_COMPILE_TIME_ASSERT

  Initial value: `((void) sizeof(char[1 - 2 * !(e)]))`

  Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

g_bsp_module_stop

```
const uint8_t g_bsp_module_stop[]
```

Detailed description

Module stop bit definitions for each ssp_ip_t enum value.

gp_mstp

```
volatile uint32_t* const gp_mstp[]
```

Detailed description

Module stop register addresses.

r_bsp_module_start_stop

```
ssp_err_t r_bsp_module_start_stop (  ssp_feature_t const *const p_feature ,
  bool stop )
```

Brief description

Start module (cancel module stop) or stop module (enter module stop).

Detailed description

NOTE: Some module stop bits are shared between peripherals. Entering module stop is not permitted for these peripherals.

Table 2305:

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.
stop	in	true to stop module, false to start module

Table 2306:

Name	Description
SSP_SUCCESS	Module is stopped or started based on stop parameter
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

Function steps

- The g_bsp_module_stop array must have entries for each ssp_ip_t enum value.
- Do nothing.
- Set MSTP bit if stop parameter is true and the bit is not shared with other channels.
- Clear MSTP bit if stop parameter is false.

R_BSP_ModuleStop

```
ssp_err_t R_BSP_ModuleStop (  ssp_feature_t const *const p_feature )
```

Brief description

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Detailed description

NOTE: Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Table 2307:

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2308:

Name	Description
SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

R_BSP_ModuleStart

```
ssp_err_t R_BSP_ModuleStart (  ssp_feature_t const *const p_feature )
```

Brief description

Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Detailed description

Table 2309:

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2310:

Name	Description
SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

bsp_mstp_bit_t

bsp_mstp_bit_t

Detailed description

Definition of BSP module stop bit.

Variables

- uint8_t bit

  Which bit in MSTP register.

- uint8_t reg

  < Special processing required

  Which MSTP register

bsp_mstp_data_t

Detailed description

Definition of BSP module stop bit.

Variables

byte

bit

10.2.10.5 ROM Registers

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Functions

- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2

Defines

- #define BSP_ROM_REG_OFS1_SETTING

  Initial value:
  ```
  (((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) |
  ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))
  ```

  OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

- #define BSP_ROM_REG_MPU_CONTROL_SETTING

  Initial value:
  ```
  ((0xFFFFFCFEU) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABLE << 8) |
  \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABLE << 9) | \
  ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_ENABLE))
  ```

  Build up SECMPUAC register based on MPU settings.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ROM_REGISTERS )
```

Detailed description

ROM registers defined here. Some have masks to make sure reserved bits are set appropriately. S124 has no MPU.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ID_CODE_1 )
```

Detailed description

ID code definitions defined here.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ID_CODE_2 )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ID_CODE_3 )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 ( BSP_SECTION_ID_CODE_4 )
```

10.2.11 S5D9

Code that is common to S5D9 MCUs.

Implements functions that are common to S5D9 MCUs.

10.2.11.1 Cache Functions

This module implements cache functions.

bsp_cache_state_t

Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

10.2.11.2 Clock Initialization

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Functions

- bsp_clock_init
- bsp_cpu_clock_get
- bsp_clock_set_callback
- bsp_clocks_mem_wait_set
- bsp_clocks_mem_wait_get
- bsp_clock_set_prechange
- bsp_clock_set_postchange
- bsp_clocks_rom_wait_set
- bsp_clocks_rom_wait_get
- bsp_clocks_sram_wait_set
- bsp_clocks_hsram_wait_set

Defines

- #define CGC_SRAM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for SRAM.
- #define CGC_SRAM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for SRAM.
- #define CGC_ZERO_WAIT_CYCLES

  Initial value: `(0U)`
- #define CGC_TWO_WAIT_CYCLES

  Initial value: `(1U)`

- #define CGC_MAX_ZERO_WAIT_FREQ

  Initial value: `(32000000U)`
- #define CGC_PRCR_KEY

  Initial value: `(0xA500U)`
- #define CGC_PRCR_UNLOCK

  Initial value: `((CGC_PRCR_KEY) | 0x1U)`
- #define CGC_PRCR_LOCK

  Initial value: `((CGC_PRCR_KEY) | 0x0U)`
- #define CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS

  Initial value: `(120000000U)`
- #define CGC_SYS_CLOCK_FREQ_NO_HSRAM_WAITS

  Initial value: `(200000000U)`
- #define CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS

  Initial value: `(80000000U)`
- #define CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS

  Initial value: `(160000000U)`
- #define CGC_ROM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for ROM.
- #define CGC_ROM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for ROM.
- #define CGC_ROM_TWO_WAIT_CYCLES

  Initial value: `(2U)`

  Specify two wait states for ROM.
- #define CGC_SRAM_PRCR_KEY

  Initial value: `(0x78U)`
- #define CGC_SRAM_UNLOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x1U)`
- #define CGC_SRAM_LOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x0U)`

bsp_clock_init

```
bsp_clock_init (  void  )
```

Brief description

Sets up system clocks.

Function steps

- MOCO is default clock out of reset. Enable new clock if chosen. S124 has no PLL.
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- PLL Source clock is always the main oscillator.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- PLL Source clock is always the main oscillator.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- Set PLL Source clock.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- Set USB clock divisor.
- Configure BCLK
- Configure SDRAM Clock

bsp_cpu_clock_get

```
bsp_cpu_clock_get (  void  )
```

Brief description

Returns frequency of CPU clock in Hz.

Detailed description

Table 2311:

Name	Description
Frequency	of the CPU in Hertz

bsp_clock_set_callback

```
ssp_err_t bsp_clock_set_callback (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings for a requested system clock change (PRE) or a just updated system clock change (POST)

Detailed description

Table 2312:

Name	Direction	Description
p_args	in	- Pre/Post request and clock frequency information

Table 2313:

Name	Description
return	SSP_SUCCESS

Function steps

- The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0.
- < No MEMWAIT cycles

bsp_clocks_mem_wait_set

```
bsp_clocks_mem_wait_set (  uint32_t setting )
```

Brief description

This function sets the value of the MEMWAIT register which controls wait cycles for flash read access.

Detailed description

Table 2314:

Name	Direction	Description
setting	in	

Table 2315:

Name	Description
none	

bsp_clocks_mem_wait_get

```
bsp_clocks_mem_wait_get (  void  )
```

Brief description

This function gets the value of the MEMWAIT register.

Detailed description

Table 2316:

Name	Description
MEMWAIT	setting

bsp_clock_set_prechange

```
bsp_clock_set_prechange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the requested clock change to the CGC.

Detailed description

Table 2317:

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK < 120 MHz

- 1 wait: ICLK >= 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- Calculate the Wait states for ROM
- If the requested_frequency <= 80 MHz 0 ROM wait states are required)
- If the 80 MHz < requested_frequency <= 160 MHz 1 ROM wait state is required)
- Anything above 160 MHz requires 2 ROM wait states
- Set the wait state BEFORE we change iclk

bsp_clock_set_postchange

```
bsp_clock_set_postchange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the system clock which has just been set by the CGC.

Detailed description

Table 2318:

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK <= 120 MHz
- 1 wait: ICLK > 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- In this case we need to set the wait state AFTER we change iclk

bsp_clocks_rom_wait_set

```
bsp_clocks_rom_wait_set (  uint8_t setting )
```

Brief description

This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM.

Detailed description

Table 2319:

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2320:

Name	Description
none	

bsp_clocks_rom_wait_get

```
bsp_clocks_rom_wait_get (  void  )
```

Brief description

This function gets the value of the ROMWT register.

Detailed description

Table 2321:

Name	Description
MEMWAIT	setting

bsp_clocks_sram_wait_set

```
bsp_clocks_sram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for both the SRAM0 and SRAM1 RAM memory.

Detailed description

Table 2322:

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2323:

Name	Description
none	

bsp_clocks_hsram_wait_set

```
bsp_clocks_hsram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for High Speed RAM memory.

Detailed description

Table 2324:

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2325:

Name	Description
none	

10.2.11.3 Hardware Locks

This file allocates hardware locks used in Atomic Locking.

Functions

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 0U )
```

Detailed description

Used to allocated hardware locks. Parameters are as follows:

1) IP name (ssp_ip_t enum without the SSP_IP_ prefix).
2) Unit number (used for blocks with variations like USB, not to be confused with ADC unit).
3) Channel number

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AES , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( BSC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CTSU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DOC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ELC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( FCU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IWDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( KEY , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LPM , 1U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( RTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TRNG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TSN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( WDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAAD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 14U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 15U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IRDA , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MMF , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MPU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPAMP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( QSPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCE , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DRW , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( EPTPC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( JPEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( PDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 1U , 0U )
```

10.2.11.4 Module Start and Stop

Module start and stop functions are provided to enable or disable peripherals.

Functions

- r_bsp_module_start_stop
- R_BSP_ModuleStop
- R_BSP_ModuleStart

Data structures

- bsp_mstp_bit_t
- bsp_mstp_data_t

Variables

- g_bsp_module_stop
- gp_mstp

Defines

- #define BSP_MCU_GROUP_S5D9

  Initial value:
- #define BSP_MSTP_DATA_INVALID

  Initial value: `(0xFFU)`
- #define BSP_MSTP_EXCEPTION

  Initial value: `(1U << 5)`

- #define BSP_MSTP_REG

  Initial value: `((x) << 6)`
- #define BSP_MSTP_BIT

  Initial value: `(x)`
- #define BSP_MSTPCRA

  Initial value: `(0U)`
- #define BSP_MSTPCRB

  Initial value: `(1U)`
- #define BSP_MSTPCRC

  Initial value: `(2U)`
- #define BSP_MSTPCRD

  Initial value: `(3U)`
- #define BSP_COMPILE_TIME_ASSERT

  Initial value: `((void) sizeof(char[1 - 2 * !(e)]))`

  Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

g_bsp_module_stop

```
const uint8_t g_bsp_module_stop[]
```

Detailed description

Module stop bit definitions for each ssp_ip_t enum value.

gp_mstp

```
volatile uint32_t* const gp_mstp[]
```

Detailed description

Module stop register addresses.

r_bsp_module_start_stop

```
ssp_err_t r_bsp_module_start_stop (  ssp_feature_t const *const p_feature ,
  bool stop )
```

Brief description

Start module (cancel module stop) or stop module (enter module stop).

Detailed description

NOTE: Some module stop bits are shared between peripherals. Entering module stop is not permitted for these peripherals.

Table 2326:

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.
stop	in	true to stop module, false to start module

Table 2327:

Name	Description
SSP_SUCCESS	Module is stopped or started based on stop parameter
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

Function steps

- The g_bsp_module_stop array must have entries for each ssp_ip_t enum value.
- Do nothing.
- Set MSTP bit if stop parameter is true and the bit is not shared with other channels.
- Clear MSTP bit if stop parameter is false.

R_BSP_ModuleStop

```
ssp_err_t R_BSP_ModuleStop (  ssp_feature_t const *const p_feature )
```

Brief description

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Detailed description

NOTE: Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Table 2328:

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2329:

Name	Description
SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

R_BSP_ModuleStart

```
ssp_err_t R_BSP_ModuleStart (  ssp_feature_t const *const p_feature )
```

Brief description

Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Detailed description

Table 2330:

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2331:

Name	Description
SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

bsp_mstp_bit_t

bsp_mstp_bit_t

Detailed description

Definition of BSP module stop bit.

Variables

- uint8_t bit

  Which bit in MSTP register.

- uint8_t reg

  < Special processing required

  Which MSTP register

bsp_mstp_data_t

Detailed description

Definition of BSP module stop bit.

Variables

byte

bit

10.2.11.5 ROM Registers

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Functions

- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2

Defines

- #define BSP_ROM_REG_OFS1_SETTING

  Initial value:`(((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))`

  OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

- #define BSP_ROM_REG_MPU_CONTROL_SETTING

  Initial value:`((0xFFFFFCFEU) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABLE << 8) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABLE << 9) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_ENABLE))`

  Build up SECMPUAC register based on MPU settings.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ROM_REGISTERS  )
```

Detailed description

ROM registers defined here. Some have masks to make sure reserved bits are set appropriately. S124 has no MPU.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_1  )
```

Detailed description

ID code definitions defined here.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_2  )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_3  )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_4  )
```

10.2.12 S7G2

Code that is common to S7G2 MCUs.

Implements functions that are common to S7G2 MCUs.

10.2.12.1 elc_peripheral_t

Possible peripherals to be linked to event signals

10.2.12.2 elc_event_t

Sources of event signals to be linked to other peripherals or the CPU1
NOTE: This list may change based on device. This list is for S7G2.

10.2.12.3 Cache Functions

This module implements cache functions.

bsp_cache_state_t

Cache enum. Passed into cache functions such as R_BSP_CacheOff() and R_BSP_CacheSet.

10.2.12.4 Clock Initialization

Functions in this file configure the system clocks based upon the macros in bsp_clock_cfg.h.

Functions

- bsp_clock_init
- bsp_cpu_clock_get
- bsp_clock_set_callback
- bsp_clocks_mem_wait_set
- bsp_clocks_mem_wait_get
- bsp_clock_set_prechange
- bsp_clock_set_postchange
- bsp_clocks_rom_wait_set
- bsp_clocks_rom_wait_get
- bsp_clocks_sram_wait_set
- bsp_clocks_hsram_wait_set

Defines

- #define CGC_SRAM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for SRAM.
- #define CGC_SRAM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for SRAM.
- #define CGC_ZERO_WAIT_CYCLES

  Initial value: `(0U)`
- #define CGC_TWO_WAIT_CYCLES

  Initial value: `(1U)`
- #define CGC_MAX_ZERO_WAIT_FREQ

  Initial value: `(32000000U)`

- #define CGC_PRCR_KEY

  Initial value: `(0xA500U)`
- #define CGC_PRCR_UNLOCK

  Initial value: `((CGC_PRCR_KEY) | 0x1U)`
- #define CGC_PRCR_LOCK

  Initial value: `((CGC_PRCR_KEY) | 0x0U)`
- #define CGC_SYS_CLOCK_FREQ_NO_RAM_WAITS

  Initial value: `(120000000U)`
- #define CGC_SYS_CLOCK_FREQ_NO_HSRAM_WAITS

  Initial value: `(200000000U)`
- #define CGC_SYS_CLOCK_FREQ_ONE_ROM_WAITS

  Initial value: `(80000000U)`
- #define CGC_SYS_CLOCK_FREQ_TWO_ROM_WAITS

  Initial value: `(160000000U)`
- #define CGC_ROM_ZERO_WAIT_CYCLES

  Initial value: `(0U)`

  Specify zero wait states for ROM.
- #define CGC_ROM_ONE_WAIT_CYCLES

  Initial value: `(1U)`

  Specify one wait states for ROM.
- #define CGC_ROM_TWO_WAIT_CYCLES

  Initial value: `(2U)`

  Specify two wait states for ROM.
- #define CGC_SRAM_PRCR_KEY

  Initial value: `(0x78U)`
- #define CGC_SRAM_UNLOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x1U)`
- #define CGC_SRAM_LOCK

  Initial value: `(((CGC_SRAM_PRCR_KEY) << 1) | 0x0U)`

bsp_clock_init

```
bsp_clock_init (  void  )
```

Brief description

Sets up system clocks.

Function steps

- MOCO is default clock out of reset. Enable new clock if chosen. S124 has no PLL.
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- PLL Source clock is always the main oscillator.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- PLL Source clock is always the main oscillator.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- MOCO is default clock out of reset. Enable new clock if chosen.
- Need to start PLL source clock and let it stabilize before starting PLL
- Set PLL Divider.
- Set PLL Multiplier.
- Set PLL Source clock.
- Wait for PLL clock source to stabilize
- MOCO, LOCO, and subclock do not have stabilization flags that can be checked.
- Wait for clock source to stabilize
- Set which clock to use for system clock and divisors for all system clocks.
- Set USB clock divisor.
- Configure BCLK
- Configure SDRAM Clock

bsp_cpu_clock_get

```
bsp_cpu_clock_get (  void  )
```

Brief description

Returns frequency of CPU clock in Hz.

Detailed description

Table 2332:

Name	Description
Frequency	of the CPU in Hertz

bsp_clock_set_callback

```
ssp_err_t bsp_clock_set_callback (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings for a requested system clock change (PRE) or a just updated system clock change (POST)

Detailed description

Table 2333:

Name	Direction	Description
p_args	in	- Pre/Post request and clock frequency information

Table 2334:

Name	Description
return	SSP_SUCCESS

Function steps

- The current frequency must be less than 32 MHz and the mcu must be in high speed mode, before changing wait cycles to 0.
- < No MEMWAIT cycles

bsp_clocks_mem_wait_set

```
bsp_clocks_mem_wait_set (  uint32_t setting )
```

Brief description

This function sets the value of the MEMWAIT register which controls wait cycles for flash read access.

Detailed description

Table 2335:

Name	Direction	Description
setting	in	

Table 2336:

Name	Description
none	

bsp_clocks_mem_wait_get

```
bsp_clocks_mem_wait_get (  void  )
```

Brief description

This function gets the value of the MEMWAIT register.

Detailed description

Table 2337:

Name	Description
MEMWAIT	setting

bsp_clock_set_prechange

```
bsp_clock_set_prechange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the requested clock change to the CGC.

Detailed description

Table 2338:

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK < 120 MHz

- 1 wait: ICLK >= 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- Calculate the Wait states for ROM
- If the requested_frequency <= 80 MHz 0 ROM wait states are required)
- If the 80 MHz < requested_frequency <= 160 MHz 1 ROM wait state is required)
- Anything above 160 MHz requires 2 ROM wait states
- Set the wait state BEFORE we change iclk

bsp_clock_set_postchange

```
bsp_clock_set_postchange (  bsp_clock_set_callback_args_t * p_args )
```

Brief description

This function sets the ROM and RAM wait state settings based on the system clock which has just been set by the CGC.

Detailed description

Table 2339:

Name	Description
none.	

Function steps

- Wait states for low speed RAM (SRAM0, SRAM1 and SRAM0 (DED)
- No wait: ICLK <= 120 MHz
- 1 wait: ICLK > 120 MHz
- Wait states for High speed RAM (SRAMHS)
- No wait: ICLK <= 200 MHz
- 1 wait: ICLK > 200 MHz
- In this case we need to set the wait state AFTER we change iclk

bsp_clocks_rom_wait_set

```
bsp_clocks_rom_wait_set (  uint8_t setting )
```

Brief description

This function sets the value of the ROMWT register which is used to specify wait states required when accessing Flash ROM.

Detailed description

Table 2340:

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2341:

Name	Description
none	

bsp_clocks_rom_wait_get

```
bsp_clocks_rom_wait_get (  void  )
```

Brief description

This function gets the value of the ROMWT register.

Detailed description

Table 2342:

Name	Description
MEMWAIT	setting

bsp_clocks_sram_wait_set

```
bsp_clocks_sram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for both the SRAM0 and SRAM1 RAM memory.

Detailed description

Table 2343:

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2344:

Name	Description
none	

bsp_clocks_hsram_wait_set

```
bsp_clocks_hsram_wait_set (  uint32_t setting )
```

Brief description

This function sets the RAM wait state settings for High Speed RAM memory.

Detailed description

Table 2345:

Name	Direction	Description
setting	in	The number of wait states to be used.

Table 2346:

Name	Description
none	

10.2.12.5 Hardware Locks

This file allocates hardware locks used in Atomic Locking.

Functions

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

- SSP_HW_LOCK_DEFINE
- SSP_HW_LOCK_DEFINE

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 0U )
```

Detailed description

Used to allocated hardware locks. Parameters are as follows:

1) IP name (ssp_ip_t enum without the SSP_IP_ prefix).
2) Unit number (used for blocks with variations like USB, not to be confused with ADC unit).
3) Channel number

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AES , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( AGT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( BSC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_LP , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CTSU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DOC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ELC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( FCU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IIC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IWDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( KEY , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LPM , 1U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( LVD , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPS , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SPI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( RTC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TRNG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( TSN , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( WDT , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAAD , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 9U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 14U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ICU , 0U , 15U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( IRDA , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MMF , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( MPU , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( OPAMP , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( POEG , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( QSPI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCE , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 2U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SSI , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ADC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( CAN , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 3U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( COMP_HS , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 4U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DMAC , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( DRW , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( EPTPC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( ETHER , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GLCDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 10U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 11U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 12U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( GPT , 0U , 13U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( JPEG , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( PDC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 5U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 6U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 7U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SCI , 0U , 8U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SRC , 0U , 0U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( SDHIMMC , 0U , 1U )
```

SSP_HW_LOCK_DEFINE

```
SSP_HW_LOCK_DEFINE ( USB , 1U , 0U )
```

10.2.12.6 Module Start and Stop

Module start and stop functions are provided to enable or disable peripherals.

Functions

- r_bsp_module_start_stop
- R_BSP_ModuleStop
- R_BSP_ModuleStart

Data structures

- bsp_mstp_bit_t
- bsp_mstp_data_t

Variables

- g_bsp_module_stop
- gp_mstp

Defines

- #define BSP_MCU_GROUP_S7G2

  Initial value:
- #define BSP_MSTP_DATA_INVALID

  Initial value: `(0xFFU)`
- #define BSP_MSTP_EXCEPTION

  Initial value: `(1U << 5)`

- #define BSP_MSTP_REG

  Initial value: `((x) << 6)`
- #define BSP_MSTP_BIT

  Initial value: `(x)`
- #define BSP_MSTPCRA

  Initial value: `(0U)`
- #define BSP_MSTPCRB

  Initial value: `(1U)`
- #define BSP_MSTPCRC

  Initial value: `(2U)`
- #define BSP_MSTPCRD

  Initial value: `(3U)`
- #define BSP_COMPILE_TIME_ASSERT

  Initial value: `((void) sizeof(char[1 - 2 * !(e)]))`

  Used to generate a compiler error (divided by 0 error) if the assertion fails. This is used in place of "#error" for expressions that cannot be evaluated by the preprocessor like sizeof().

g_bsp_module_stop

```
const uint8_t g_bsp_module_stop[]
```

Detailed description

Module stop bit definitions for each ssp_ip_t enum value.

gp_mstp

```
volatile uint32_t* const gp_mstp[]
```

Detailed description

Module stop register addresses.

r_bsp_module_start_stop

```
ssp_err_t r_bsp_module_start_stop (  ssp_feature_t const *const p_feature ,
  bool stop )
```

Brief description

Start module (cancel module stop) or stop module (enter module stop).

Detailed description

NOTE: Some module stop bits are shared between peripherals. Entering module stop is not permitted for these peripherals.

Table 2347:

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.
stop	in	true to stop module, false to start module

Table 2348:

Name	Description
SSP_SUCCESS	Module is stopped or started based on stop parameter
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

Function steps

- The g_bsp_module_stop array must have entries for each ssp_ip_t enum value.
- Do nothing.
- Set MSTP bit if stop parameter is true and the bit is not shared with other channels.
- Clear MSTP bit if stop parameter is false.

R_BSP_ModuleStop

```
ssp_err_t R_BSP_ModuleStop (  ssp_feature_t const *const p_feature )
```

Brief description

Stop module (enter module stop). Stopping a module disables clocks to the peripheral to save power.

Detailed description

NOTE: Some module stop bits are shared between peripherals. Modules with shared module stop bits cannot be stopped to prevent unintentionally stopping related modules.

Table 2349:

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2350:

Name	Description
SSP_SUCCESS	Module is stopped
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit, or module stop bit is shared and entering module stop is not supported because it could affect other modules.

R_BSP_ModuleStart

```
ssp_err_t R_BSP_ModuleStart (  ssp_feature_t const *const p_feature )
```

Brief description

Start module (cancel module stop). Starting a module enables clocks to the peripheral and allows registers to be set.

Detailed description

Table 2351:

Name	Direction	Description
p_feature	in	Pointer to definition of the feature, defined by ssp_feature_t.

Table 2352:

Name	Description
SSP_SUCCESS	Module is started
SSP_ERR_ASSERTION	p_feature::id is invalid
SSP_ERR_INVALID_ARGUMENT	Module has no module stop bit.

bsp_mstp_bit_t

bsp_mstp_bit_t

Detailed description

Definition of BSP module stop bit.

Variables

- uint8_t bit

  Which bit in MSTP register.

- uint8_t reg

  < Special processing required

  Which MSTP register

bsp_mstp_data_t

Detailed description

Definition of BSP module stop bit.

Variables

byte

bit

10.2.12.7 ROM Registers

Defines MCU registers that are in ROM (e.g. OFS) and must be set at compile-time. All registers can be set using bsp_cfg.h.

Functions

- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2
- BSP_PLACE_IN_SECTION_V2

Defines

- #define BSP_ROM_REG_OFS1_SETTING

  Initial value:`(((uint32_t)BSP_CFG_ROM_REG_OFS1 & 0xFFFF8FFFU) | ((uint32_t)BSP_CFG_HOCO_FREQUENCY << 12))`

  OR in the HOCO frequency setting from bsp_clock_cfg.h with the OFS1 setting from bsp_cfg.h.

- #define BSP_ROM_REG_MPU_CONTROL_SETTING

  Initial value:`((0xFFFFFCFEU) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC0_ENABLE << 8) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_PC1_ENABLE << 9) | \ ((uint32_t)BSP_CFG_ROM_REG_MPU_REGION0_ENABLE))`

  Build up SECMPUAC register based on MPU settings.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ROM_REGISTERS  )
```

Detailed description

ROM registers defined here. Some have masks to make sure reserved bits are set appropriately. S124 has no MPU.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_1  )
```

Detailed description

ID code definitions defined here.

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_2  )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_3  )
```

BSP_PLACE_IN_SECTION_V2

```
BSP_PLACE_IN_SECTION_V2 (  BSP_SECTION_ID_CODE_4  )
```

10.3 Common BSP Code

Code common to all BSPs.

Implements functions that are common to all BSPs.

10.3.1 Functions

- R_BSP_VersionGet
- R_SSP_VersionGet
- bsp_init_internal

10.3.2 Defines

- #define BSP_IRQ_DISABLED

  Initial value: `(0xFFU)`

  Used to signify that an ELC event is not able to be used as an interrupt.

- #define BSP_CODE_VERSION_MAJOR

  Initial value: `(1U)`

- #define BSP_CODE_VERSION_MINOR

  Initial value: `(10U)`

- #define BSP_API_VERSION_MAJOR

  Initial value: `(1U)`

- #define BSP_API_VERSION_MINOR

  Initial value: `(0U)`

- #define SF_CONTEXT_SAVE

  Initial value:

- #define SF_CONTEXT_RESTORE

  Initial value:

- #define SSP_ASSERT_FAIL

  Initial value:

  Function call to insert before returning assertion error.

- #define SSP_ERROR_LOG

  Initial value:

  This function is called before returning an error code. To stop on a runtime error, define ssp_error_log in user code and do required debugging (breakpoints, stack dump, etc) in this function.

- #define SSP_ERROR_RETURN

  Initial value:`{ \ if ((a)) \ { \ (void) 0; /* Do nothing */ \ } \ else \ { \ #define SSP_ERROR_LOG((err), (module), (version)); \ return (err); \ } \ }`

  All SSP error codes are returned using this macro. Calls #define SSP_ERROR_LOG function if condition "a" is false. Used to identify runtime errors in SSP functions.

- #define SSP_CRITICAL_SECTION_DEFINE

  Initial value:`uint32_t old_mask_level = 0U`

  This check is performed to select suitable ASM API with respect to core This macro defines a variable for saving previous mask value

- #define SSP_CRITICAL_SECTION_ENTER

  Initial value:`old_mask_level = __get_PRIMASK(); \ __set_PRIMASK(1U)`

  This macro defined to get the mask value
- #define SSP_CRITICAL_SECTION_EXIT

  Initial value:`__set_PRIMASK(old_mask_level)`

  This macro defined to restore the old mask value

10.3.3 R_BSP_VersionGet

```
ssp_err_t R_BSP_VersionGet ( ssp_version_t  * p_version )
```

10.3.3.1 Brief description

Set BSP version based on compile time macros.

10.3.3.2 Detailed description

Table 2353:

Name	Direction	Description
p_version	out	Memory address to return version information to.

Table 2354:

Name	Description
SSP_SUCCESS	Version information stored.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

10.3.4 R_SSP_VersionGet

```
ssp_err_t R_SSP_VersionGet ( ssp_pack_version_t  *const p_version )
```

10.3.4.1 Brief description

Set SSP version based on compile time macros.

10.3.4.2 Detailed description

Table 2355:

Name	Direction	Description
p_version	out	Memory address to return version information to.

Table 2356:

Name	Description
SSP_SUCCESS	Version information stored.
SSP_ERR_ASSERTION	The parameter p_version is NULL.

10.3.5 bsp_init_internal

```
bsp_init_internal (  void * p_args )
```

10.3.5.1 Brief description

Default initialization function, used only if bsp_init is not defined in the user application.

10.3.6 Common BSP LED Code and Types

Common support for board LEDs.

Contains types and functions that allow for common use of LEDs on boards

10.3.6.1 Data structures

- bsp_leds_t

10.3.6.2 Functions

- R_BSP_LedsGet

10.3.6.3 R_BSP_LedsGet

```
ssp_err_t R_BSP_LedsGet ( bsp_leds_t  * p_leds )
```

Brief description

Return information about the LEDs on the current board.

Detailed description

Structure with LED information.

Table 2357:

Name	Direction	Description
p_leds	out	Pointer to structure where LED info is stored.

10.3.6.4 bsp_leds_t

bsp_leds_t

Detailed description

Information on how many LEDs and what pins they are on.

Variables

- uint16_t led_count

  The number of LEDs on this board.

- ioport_port_pin_t const * p_leds

  Pointer to an array of IOPORT pins for controlling LEDs.

10.3.7 Compiler Support

The macros in this file are defined based upon which compiler is being used. The macros abstract common section names and gives a common way to place code in a particular section. Some macros have a version that ends in V2. These were created to unify the usage between compilers while retaining backwards compatibility in the existing macros and should be preferred in new code.

Description of macros:

- BSP_SECTION_STACK - Name of section where stack(s) are stored
- BSP_SECTION_HEAP - Name of section where heap(s) are stored
- BSP_SECTION_VECTOR - Name of section where vector table is stored
- BSP_SECTION_ROM_REGISTERS - Name of section where ROM registers are located
- BSP_PLACE_IN_SECTION - Macro for placing code in a particular section
- BSP_ALIGN_VARIABLE - Macro for specifying a minimum alignment in bytes
- BSP_PACKED - Macro for setting a 1 byte alignment to remove padding

- BSP_DONT_REMOVE - Keyword to tell linker/compiler to not optimize out a variable or function

NOTE: Currently supported compilers are GCC and IAR

10.3.7.1 Defines

- #define BSP_PLACE_IN_SECTION_V2
  Initial value:`__attribute__ ((section(x))) __attribute__ ((__used__))`
- #define BSP_ALIGN_VARIABLE_V2
  Initial value:`__attribute__ ((aligned (x)))`
- #define BSP_PACKED_V2
  Initial value:`__attribute__ ((aligned(1)))`

10.3.8 Software Delay

Common function to implement a software delay.
Implements a software delay function for all BSPs.

10.3.8.1 Functions

- R_BSP_SoftwareDelay
- software_delay_loop

10.3.8.2 R_BSP_SoftwareDelay

```
R_BSP_SoftwareDelay (  uint32_t delay ,   bsp_delay_units_t units )
```

Brief description

Delay the specified duration in units and return.

Detailed description

Table 2358:

Name	Direction	Description
delay	in	The number of 'units' to delay.

Table 2358: (Continued)

Name	Direction	Description
units	in	The 'base' (bsp_delay_units_t) for the units specified. Valid values are: BSP_DELAY_UNITS_SECONDS, BSP_DELAY_UNITS_MILLISECONDS, BSP_DELAY_UNITS_MICROSECONDS. For example: At 1 MHz one cycle takes 1 microsecond (.000001 seconds). At 12 MHz one cycle takes 1/12 microsecond or 83 nanoseconds. Therefore one run through software_delay_loop() takes: ~ (83 * DELAY_LOOP_CYCLES) or 332 ns. A delay of 2 us therefore requires 2000ns/332ns or 6 loops.

The 'theoretical' maximum delay that may be obtained is determined by a full 32 bit loop count and the system clock rate. @240MHz: ((0xFFFFFFFF loops * 4 cycles /loop) / 240000000) = 71 seconds. @32MHz: ((0xFFFFFFFF loops * 4 cycles /loop) / 32000000) = 536 seconds

Note that requests for very large delays will be affected by rounding in the calculations and the actual delay achieved may be slightly less. @32 MHz, for example, a request for 532 seconds will be closer to 536 seconds.

Note also that if the calculations result in a loop_cnt of zero, the software_delay_loop() function is not called at all. In this case the requested delay is too small (nanoseconds) to be carried out by the loop itself, and the overhead associated with executing the code to just get to this point has certainly satisfied the requested delay.

NOTE: This function calls bsp_cpu_clock_get which ultimately calls R_CGC_SystemClockFreqGet and therefore requires that the BSP has already initialized the CGC (which it does as part of the Sysinit). Care should be taken to ensure this remains the case if in the future this function were to be called as part of the BSP initialization.

Table 2359:

Name	Description
None.	

Function steps

- Convert the requested time to microseconds.
- Get the system clock frequency in Hz.
- Get the # of nanoseconds/cycle.
- Only delay if the supplied parameters constitute a delay.

10.3.8.3 software_delay_loop

```
software_delay_loop (  uint32_t loop_cnt )
```

Brief description

This assembly language routine takes roughly 4 cycles per loop. 2 additional cycles occur when the loop exits. The 'naked' attribute indicates that the specified function does not need prologue/epilogue sequences generated by the compiler.

Detailed description

Table 2360:

Name	Direction	Description
loop_cnt	in	The number of loops to iterate.

Table 2361:

Name	Description
None.	

Function steps

- < 1 cycle
- < 2 cycles
- < 2 cycles
- loop_cnt is used but since it is used in assembly an unused parameter warning can be generated.

10.3.9 Error Checking

This file performs build time error checking where possible.

10.3.9.1 Defines

- #define BSP_STACK_ALIGNMENT

  Initial value: `(8)`

  Stacks (and heap) must be sized and aligned to an integer multiple of this number.

10.3.10 Module specific feature overrides

This group contains lookup functions that provide MCU specific feature information that is not available in the factory flash.

10.3.10.1 Functions

- R_BSP_FeatureSciGet
- R_BSP_FeatureRspiGet
- R_BSP_FeatureLvdGet
- R_BSP_FeatureAcmphsGet
- R_BSP_FeatureAdcGet
- R_BSP_FeatureCtsuGet
- R_BSP_FeatureIoportGet
- R_BSP_FeatureCgcGet
- R_BSP_FeatureCanGet
- R_BSP_FeatureDacGet
- R_BSP_FeatureFlashLpGet
- R_BSP_FeatureOpampGet
- R_BSP_FeatureSdhiGet
- R_BSP_FeatureSsiGet

10.3.10.2 Data structures

- bsp_feature_sci_t
- bsp_feature_rspi_t
- bsp_feature_lvd_t
- bsp_feature_acmphs_t
- bsp_feature_adc_t

- bsp_feature_can_t
- bsp_feature_dac_t
- bsp_feature_flash_lp
- bsp_feature_flash_hp
- bsp_feature_ctsu_t
- bsp_feature_ioport_t
- bsp_feature_cgc_t
- bsp_feature_opamp_t
- bsp_feature_sdhi_t
- bsp_feature_ssi_t

10.3.10.3 R_BSP_FeatureSciGet

```
R_BSP_FeatureSciGet ( bsp_feature_sci_t  * p_sci_feature )
```

Detailed description

Get MCU specific SCI features.

Table 2362:

Name	Direction	Description
p_sci_feature	out	Pointer to structure to store SCI features.

10.3.10.4 R_BSP_FeatureRspiGet

```
R_BSP_FeatureRspiGet ( bsp_feature_rspi_t  * p_rspi_feature )
```

Detailed description

Get MCU specific RSPI features.

Table 2363:

Name	Direction	Description
p_rspi_feature	out	Pointer to structure to store RSPI features.

10.3.10.5 R_BSP_FeatureLvdGet

```
R_BSP_FeatureLvdGet ( bsp_feature_lvd_t  * p_lvd_feature )
```

Detailed description

Get MCU specific LVD features.

Table 2364:

Name	Direction	Description
p_lvd_feature	out	Pointer to structure to store LVD features.

10.3.10.6 R_BSP_FeatureAcmphsGet

```
R_BSP_FeatureAcmphsGet ( bsp_feature_acmphs_t  * p_acmphs_feature )
```

Detailed description

Get MCU specific ACMPHS features

Table 2365:

Name	Direction	Description
p_acmphs_feature	out	Pointer to structure to store ACMPHS features.

10.3.10.7 R_BSP_FeatureAdcGet

```
R_BSP_FeatureAdcGet ( bsp_feature_adc_t  * p_adc_feature )
```

Detailed description

Get MCU specific ADC features

Table 2366:

Name	Direction	Description
p_adc_feature	out	Pointer to structure to store ADC features.

10.3.10.8 R_BSP_FeatureCtsuGet

```
R_BSP_FeatureCtsuGet ( bsp_feature_ctsu_t  * p_ctsu_feature )
```

Detailed description

Get MCU specific CTSU features

Table 2367:

Name	Direction	Description
p_ctsu_feature	out	Pointer to structure to store CTSU features.

10.3.10.9 R_BSP_FeatureIoportGet

```
R_BSP_FeatureIoportGet ( bsp_feature_ioport_t  * p_ioport_feature )
```

Detailed description

Get MCU specific I/O port features

Table 2368:

Name	Direction	Description
p_ioport_feature	out	Pointer to structure to store I/O port features.

10.3.10.10 R_BSP_FeatureCgcGet

```
R_BSP_FeatureCgcGet ( bsp_feature_cgc_t  const ** pp_cgc_feature )
```

Detailed description

Get MCU specific CGC features

Table 2369:

Name	Direction	Description
pp_cgc_feature	out	Pointer to structure to store pointer to CGC features.

10.3.10.11 R_BSP_FeatureCanGet

```
R_BSP_FeatureCanGet ( bsp_feature_can_t  * p_can_feature )
```

Detailed description

Get MCU specific CAN features

Table 2370:

Name	Direction	Description
p_can_feature	out	Pointer to structure to store CAN features.

10.3.10.12 R_BSP_FeatureDacGet

```
R_BSP_FeatureDacGet ( bsp_feature_dac_t  * p_dac_feature )
```

Detailed description

Get MCU specific DAC features

Table 2371:

Name	Direction	Description
p_dac_feature	out	Pointer to structure to store DAC features.

10.3.10.13 R_BSP_FeatureFlashLpGet

```
R_BSP_FeatureFlashLpGet ( bsp_feature_flash_lp  * p_flash_lp_feature )
```

Detailed description

Get MCU specific FLASH LP features

Table 2372:

Name	Direction	Description
p_flash_lp_feature	out	Pointer to structure to store Flash LP features.

Function steps

- S124 uses 1 macro of 128K and single access for Code Flash. It can therefore access 128K as a single macro and it's Code Flash memory is effectively organized as a single macro of 128K, yielding a total of 128K Code Flash.
- S3A7 uses 4 macros of 256K and double access for Code Flash. It can therefore access 512K as a single macro and it's Code Flash memory is effectively organized as 2 macros of 512K each, yielding a total of 1MB Code Flash. This generates a macro boundary at 512K which is important for blank checking.

10.3.10.14 R_BSP_FeatureOpampGet

```
R_BSP_FeatureOpampGet ( bsp_feature_opamp_t  *const p_opamp_feature )
```

Detailed description

Get MCU specific OPAMP features

Table 2373:

Name	Direction	Description
p_opamp_feature	out	Pointer to structure to store OPAMP features.

10.3.10.15 R_BSP_FeatureSdhiGet

```
R_BSP_FeatureSdhiGet ( bsp_feature_sdhi_t  * p_sdhi_feature )
```

Detailed description

Get MCU specific SDHI features

Table 2374:

Name	Direction	Description
p_sdhi_feature	out	Pointer to structure to store SDHI features.

10.3.10.16 R_BSP_FeatureSsiGet

```
R_BSP_FeatureSsiGet ( bsp_feature_ssi_t  * p_ssi_feature )
```

Detailed description

Get MCU specific SSI features

Table 2375:

Name	Direction	Description
p_ssi_feature	out	Pointer to structure to store SSI features.

10.3.10.17 bsp_feature_sci_t

bsp_feature_sci_t

Detailed description

SCI MCU specific features.

Variables

- uint8_t clock

  Which clock the SCI is connected to.

10.3.10.18 bsp_feature_rspi_t

bsp_feature_rspi_t

Detailed description

RSPI MCU specific features.

Variables

- uint8_t clock

  Which clock the RSPI is connected to.
- uint8_t has_ssl_level_keep

10.3.10.19 bsp_feature_lvd_t

bsp_feature_lvd_t

Detailed description

LVD MCU specific features.

Variables

- uint8_t monitor_1_low_threshold

  Monitor 1 lowest valid voltage threshold.

- uint8_t monitor_1_hi_threshold

  Monitor 1 highest valid voltage threshold.
- uint8_t monitor_2_low_threshold

  Monitor 2 lowest valid voltage threshold.
- uint8_t monitor_2_hi_threshold

  Monitor 2 highest valid voltage threshold.
- uint32_t has_digital_filter

  Whether or not LVD has a digital filter.
- uint32_t negation_delay_clock

  Clock required for LVD signal negation delay after reset.

10.3.10.20 bsp_feature_acmphs_t

bsp_feature_acmphs_t

Detailed description

ACMPHS MCU specific features.

Variables

- uint32_t min_wait_time_us

  Minimum stabilization wait time in microseconds.

10.3.10.21 bsp_feature_adc_t

bsp_feature_adc_t

Detailed description

ADC MCU specific features.

Variables

- uint8_t has_sample_hold_reg

  Whether or not sample and hold registers are present.
- uint8_t group_b_sensors_allowed

  Whether or not sensors are allowed on group b.
- uint8_t sensors_exclusive

  Whether or not sensors can be used with other sensors/channels.
- uint8_t tsn_calibration_available

  Identify if the TSN calibration data is available.

- uint8_t tsn_control_available

  Identify if the TSN control register is available.
- int16_t tsn_slope

  TSN slope in micro-volts/°C.
- uint32_t sensor_min_sampling_time

  The minimum sampling time required by the on-chip temperature and voltage sensor in nsec.
- uint32_t clock_source

  The conversion clock used by the ADC peripheral.
- uint8_t addition_supported

  Whether addition is supported or not in this MCU.
- uint8_t calibration_reg_available

  Whether CALEXE register is available.

10.3.10.22 bsp_feature_can_t

bsp_feature_can_t

Detailed description

CAN MCU specific features.

Variables

- uint8_t mclock_only

  Whether or not MCLK is the only valid clock.
- uint8_t check_pclkb_ratio

  Whether clock:PCLKB must be 2:1.
- uint8_t clock

  Which clock to compare PCLKB to.

10.3.10.23 bsp_feature_dac_t

bsp_feature_dac_t

Detailed description

DAC MCU specific features.

Variables

- uint8_t has_davrefcr

  Whether or not DAC has DAVREFCR register.

10.3.10.24 bsp_feature_flash_lp

bsp_feature_flash_lp

Detailed description

FLASH LP MCU specific features.

Variables

- uint8_t flash_clock_src

  Source clock for Flash (ie. FCLK or ICLK)
- uint8_t flash_cf_macros

  Number of implemented code flash hardware macros.
- uint32_t cf_macro_size

  The size of the implemented Code Flash macro.

10.3.10.25 bsp_feature_flash_hp

bsp_feature_flash_hp

Detailed description

FLASH HP MCU specific features.

Variables

- uint8_t cf_block_size_write

  Code Flash Block size write.

10.3.10.26 bsp_feature_ctsu_t

bsp_feature_ctsu_t

Detailed description

CTSU MCU specific features.

Variables

- uint8_t ctsucr0_mask

  Mask of valid bits in CTSUCR0.
- uint8_t ctsucr1_mask

  Mask of valid bits in CTSUCR1.
- uint8_t ctsumch0_mask

  Mask of valid bits in CTSUMCH0.

- uint8_t ctsumch1_mask

  Mask of valid bits in CTSUMCH1.
- uint8_t ctsuchac_register_count

  Number of CTSUCHAC registers.
- uint8_t ctsuchtrc_register_count

  Number of CTSUCHTRC registers.

10.3.10.27 bsp_feature_ioport_t

bsp_feature_ioport_t

Detailed description

I/O port MCU specific features.

Variables

- uint8_t has_ethernet

  Whether or not MCU has Ethernet port configurations.
- uint8_t has_vbatt_pins

  Whether or not MCU has pins on vbatt domain.

10.3.10.28 bsp_feature_cgc_t

bsp_feature_cgc_t

Detailed description

CGC MCU specific features.

Variables

- uint32_t high_speed_freq_hz

  Frequency above which high speed mode must be used.
- uint32_t middle_speed_max_freq_hz

  Max frequency for middle speed, 0 indicates not available.
- uint32_t low_speed_max_freq_hz

  Max frequency for low speed, 0 indicates not available.
- uint32_t low_voltage_max_freq_hz

  Max frequency for low voltage, 0 indicates not available.
- uint32_t low_speed_pclk_div_min

  Minimum divisor for peripheral clocks when using oscillator stop detect.

- uint32_t low_voltage_pclk_div_min

  Minimum divisor for peripheral clocks when using oscillator stop detect.
- uint32_t hoco_freq_hz

  HOCO frequency.
- uint32_t main_osc_freq_hz

  Main oscillator frequency.
- uint8_t modrv_mask

  Mask for MODRV in MOMCR.
- uint8_t modrv_shift

  Offset of lowest bit of MODRV in MOMCR.
- uint8_t sodrv_mask

  Mask for SODRV in SOMCR.
- uint8_t sodrv_shift

  Offset of lowest bit of SODRV in SOMCR.
- uint8_t pll_div_max

  Maximum PLL divisor.
- uint8_t pll_mul_min

  Minimum PLL multiplier.
- uint8_t pll_mul_max

  Maximum PLL multiplier.
- uint8_t mainclock_drive

  Main clock drive capacity.
- uint32_t iclk_div

  ICLK divisor.
- uint32_t pllccr_type

  0: No PLL, 1: PLLCCR, 2: PLLCCR2
- uint32_t pll_src_configurable

  Whether or not PLL clock source is configurable.
- uint32_t has_subosc_speed

  Whether or not MCU has subosc speed mode.
- uint32_t has_lcd_clock

  Whether or not MCU has LCD clock.
- uint32_t has_sdram_clock

  Whether or not MCU has SDRAM clock.

- uint32_t has_usb_clock_div

  Whether or not MCU has USB clock divisor.
- uint32_t has_pclka

  Whether or not MCU has PCLKA clock.
- uint32_t has_pclkb

  Whether or not MCU has PCLKB clock.
- uint32_t has_pclkc

  Whether or not MCU has PCLKC clock.
- uint32_t has_pclkd

  Whether or not MCU has PCLKD clock.
- uint32_t has_fclk

  Whether or not MCU has FCLK clock.
- uint32_t has_bclk

  Whether or not MCU has BCLK clock.
- uint32_t has_sdadc_clock

  Whether or not MCU has SDADC clock.

10.3.10.29 bsp_feature_opamp_t

bsp_feature_opamp_t

Detailed description

OPAMP MCU specific features.

Variables

- uint16_t min_wait_time_lp_us

  Minimum wait time in low power mode.
- uint16_t min_wait_time_ms_us

  Minimum wait time in middle speed mode.
- uint16_t min_wait_time_hs_us

  Minimum wait time in high speed mode.

10.3.10.30 bsp_feature_sdhi_t

bsp_feature_sdhi_t

Detailed description

SDHI MCU specific features.

Variables

- uint8_t has_card_detection

  Whether or not MCU has card detection.

- uint32_t max_clock_frequency

  Maximum clock rate supported by the peripheral.

10.3.10.31 bsp_feature_ssi_t

bsp_feature_ssi_t

Detailed description

SSI MCU specific features.

Variables

- uint8_t fifo_num_stages

  Number of FIFO stages on this MCU.

10.3.11 Grouped Interrupt Support

Support for grouped interrupts. Grouped interrupts occur when multiple interrupt events trigger the same interrupt vector. When this common vector is triggered the activation source must be discovered. The functions in this file allow users to register a callback function for a single interrupt source in an interrupt group.

10.3.11.1 Functions

- R_BSP_GroupIrqWrite
- NMI_Handler

10.3.11.2 R_BSP_GroupIrqWrite

```
ssp_err_t R_BSP_GroupIrqWrite (  bsp_grp_irq_t irq ,   void(*)(bsp_grp_irq_t
irq) p_callback )
```

Brief description

Register a callback function for supported interrupts. If NULL is passed for the callback argument then any previously registered callbacks are unregistered.

Detailed description

Table 2376:

Name	Direction	Description
irq	in	Interrupt for which to register a callback.
p_callback	in	Pointer to function to call when interrupt occurs.

Table 2377:

Name	Description
SSP_SUCCESS	Callback registered

Function steps

- Check for valid address.
- Callback was NULL. De-register callback.
- Register callback.

10.3.11.3 NMI_Handler

```
NMI_Handler ( void )
```

Brief description

Non-maskable interrupt handler. This exception is defined by the BSP, unlike other system exceptions, because there are many sources that map to the NMI exception.

Function steps

- Determine what is the cause of this interrupt.
- IWDT underflow/refresh error interrupt is requested.
- Clear IWDT flag.
- WDT underflow/refresh error interrupt is requested.
- Clear WDT flag.
- Voltage monitoring 1 interrupt is requested.
- Clear LVD1 flag.
- Voltage monitoring 2 interrupt is requested.

- Clear LVD2 flag.
- Oscillation stop detection interrupt is requested.
- Clear oscillation stop detect flag.
- NMI pin interrupt is requested.
- Clear NMI pin interrupt flag.
- RAM Parity Error interrupt is requested.
- Clear RAM parity error flag.
- RAM ECC Error interrupt is requested.
- Clear RAM ECC error flag.
- Determine what is the cause of this interrupt.
- IWDT underflow/refresh error interrupt is requested.
- Clear IWDT flag.
- WDT underflow/refresh error interrupt is requested.
- Clear WDT flag.
- Voltage monitoring 1 interrupt is requested.
- Clear LVD1 flag.
- Voltage monitoring 2 interrupt is requested.
- Clear LVD2 flag.
- VBATT Monitor interrupt is requested.
- Clear VBATT flag.
- Oscillation stop detection interrupt is requested.
- Clear oscillation stop detect flag.
- NMI pin interrupt is requested.
- Clear NMI pin interrupt flag.
- RAM Parity Error interrupt is requested.
- Clear RAM parity error flag.
- RAM ECC Error interrupt is requested.
- Clear RAM ECC error flag.
- MPU Bus Slave Error interrupt is requested.
- Clear MPU Bus Slave error flag.
- MPU Bus Slave Error interrupt is requested.
- Clear MPU Bus Master error flag.
- MPU Stack Error interrupt is requested.
- Clear MPU Stack error flag.

- Determine what is the cause of this interrupt.
- IWDT underflow/refresh error interrupt is requested.
- Clear IWDT flag.
- WDT underflow/refresh error interrupt is requested.
- Clear WDT flag.
- Voltage monitoring 1 interrupt is requested.
- Clear LVD1 flag.
- Voltage monitoring 2 interrupt is requested.
- Clear LVD2 flag.
- VBATT Monitor interrupt is requested.
- Clear VBATT flag.
- Oscillation stop detection interrupt is requested.
- Clear oscillation stop detect flag.
- NMI pin interrupt is requested.
- Clear NMI pin interrupt flag.
- RAM Parity Error interrupt is requested.
- Clear RAM parity error flag.
- RAM ECC Error interrupt is requested.
- Clear RAM ECC error flag.
- MPU Bus Slave Error interrupt is requested.
- Clear MPU Bus Slave error flag.
- MPU Bus Master Error interrupt is requested.
- Clear MPU Bus Master error flag.
- MPU Stack Error interrupt is requested.
- Clear MPU Stack error flag.

10.3.12 Interrupt Initialization

This module configures certain ELC events so that they can trigger NVIC interrupts. Which events are used as interrupts depends on settings in bsp_irq_cfg.h.

10.3.12.1 Functions

- R_BSP_IrqStatusClear
- bsp_irq_cfg

10.3.12.2 R_BSP_IrqStatusClear

```
R_BSP_IrqStatusClear (  IRQn_Type irq )
```

Brief description

Clear the interrupt status flag (IR) for a given interrupt. When an interrupt is triggered the IR bit is set. If it is not cleared in the ISR then the interrupt will trigger again immediately.

Detailed description

Table 2378:

Name	Direction	Description
irq	in	Interrupt for which to clear the IR bit. Note that the enums listed for IRQn_Type are only those for the Cortex Processor Exceptions Numbers. In prior releases this list contained all of the interrupts enabled for the specfic MCU but enabled interrupts are now configured using the SSP_VECTOR_DEFINE macro.

NOTE: This does not work for system exceptions where the IRQn_Type value is < 0.

10.3.12.3 bsp_irq_cfg

```
bsp_irq_cfg (  void  )
```

Brief description

Using the vector table information section that has been built by the linker and placed into ROM in the .vector_info. section, this function will initialize the ICU so that configured ELC events will trigger interrupts in the NVIC.

10.3.13 Atomic Locking

This module implements atomic locking mechanisms.

10.3.13.1 Functions

- R_BSP_SoftwareLock
- r_bsp_software_lock_cm0plus
- R_BSP_SoftwareUnlock

- R_BSP_HardwareLock
- R_BSP_HardwareUnlock
- bsp_init_hardware_locks
- R_BSP_SoftwareLockInit

10.3.13.2 Data structures

- bsp_lock_t

10.3.13.3 R_BSP_SoftwareLock

```
ssp_err_t R_BSP_SoftwareLock ( bsp_lock_t  * p_lock )
```

Brief description

Attempt to acquire the lock that has been sent in.

Detailed description

Table 2379:

Name	Direction	Description
p_lock	in	Pointer to the structure which contains the lock to be acquired.

Table 2380:

Name	Description
SSP_SUCCESS	Lock was acquired
SSP_ERR_IN_USE	Lock was not acquired

10.3.13.4 r_bsp_software_lock_cm0plus

```
ssp_err_t r_bsp_software_lock_cm0plus ( bsp_lock_t  * p_lock )
```

Brief description

Attempt to acquire the lock that has been sent in (CM0+ implementation).

Detailed description

Table 2381:

Name	Direction	Description
p_lock	in	Pointer to the structure which contains the lock to be acquired.

Table 2382:

Name	Description
SSP_SUCCESS	Lock was acquired
SSP_ERR_IN_USE	Lock was not acquired

10.3.13.5 R_BSP_SoftwareUnlock

```
R_BSP_SoftwareUnlock ( bsp_lock_t  * p_lock )
```

Brief description

Release hold on lock.

Detailed description

Table 2383:

Name	Direction	Description
p_lock	in	Pointer to the structure which contains the lock to unlock.

10.3.13.6 R_BSP_HardwareLock

```
ssp_err_t R_BSP_HardwareLock (  ssp_feature_t const *const p_feature )
```

Brief description

Attempt to reserve a hardware resource lock.

Detailed description

Table 2384:

Name	Direction	Description
p_feature	in	Pointer to the module specific feature information.

Table 2385:

Name	Description
SSP_SUCCESS	Lock was acquired
SSP_ERR_IN_USE	Lock was not acquired

10.3.13.7 R_BSP_HardwareUnlock

```
R_BSP_HardwareUnlock (  ssp_feature_t const *const p_feature )
```

Brief description

Release hold on lock.

Detailed description

Table 2386:

Name	Direction	Description
p_feature	in	Pointer to the module specific feature information.

10.3.13.8 bsp_init_hardware_locks

```
bsp_init_hardware_locks (  void  )
```

Brief description

Initialize all of the hardware locks to BSP_LOCK_UNLOCKED.

10.3.13.9 R_BSP_SoftwareLockInit

```
R_BSP_SoftwareLockInit ( bsp_lock_t  * p_lock )
```

Brief description

Initialize lock value to be unlocked.

Detailed description

Table 2387:

Name	Direction	Description
p_lock	in	Pointer to the structure which contains the lock to initialize.

10.3.13.10 bsp_lock_t

bsp_lock_t

Detailed description

Lock structure. Passed into software locking functions such as R_BSP_SoftwareLock and R_BSP_SoftwareUnLock.

Variables

- uint8_t lock

  A uint8_t is used instead of a enum because the size must be 8-bits.

10.3.14 Register Protection

Important registers are write protected. This module provides APIs for configuring the protection of these registers. Reference counters are used to ensure proper operation.

10.3.14.1 Functions

- R_BSP_RegisterProtectEnable
- R_BSP_RegisterProtectDisable
- bsp_register_protect_open

10.3.14.2 R_BSP_RegisterProtectEnable

```
R_BSP_RegisterProtectEnable (  bsp_reg_protect_t regs_to_protect )
```

Brief description

Enable register protection. Registers that are protected cannot be written to. Register protection is enabled by using the Protect Register (PRCR) and the MPC's Write-Protect Register (PWPR).

Detailed description

Table 2388:

Name	Direction	Description
regs_to_protect	in	Registers which have write protection enabled.

Function steps

- Get/save the current state of interrupts
- Enable protection using PRCR register.
- When writing to the PRCR register the upper 8-bits must be the correct key. Set lower bits to 0 to disable writes.
- Restore the interrupt state

10.3.14.3 R_BSP_RegisterProtectDisable

```
R_BSP_RegisterProtectDisable (  bsp_reg_protect_t regs_to_unprotect )
```

Brief description

Disable register protection. Registers that are protected cannot be written to. Register protection is disabled by using the Protect Register (PRCR) and the MPC's Write-Protect Register (PWPR).

Detailed description

Table 2389:

Name	Direction	Description
regs_to_unprotect	in	Registers which have write protection disabled.

Function steps

- Get/save the current state of interrupts
- Disable protection using PRCR register.
- When writing to the PRCR register the upper 8-bits must be the correct key. Set lower bits to 0 to disable writes.
- Increment the protect counter
- Restore the interrupt state

10.3.14.4 bsp_register_protect_open

```
bsp_register_protect_open (  void  )
```

Brief description

Initializes variables needed for register protection functionality.

Function steps

- Initialize reference counters to 0.

Chapter 11 API Reference: Structure Index

11.1 Structures

11.1.1 acmphs_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    R_ACMPHS0_Type *  p_reg
    comparator_pin_output_t  pin_output
    void(*  p_callback)(comparator_callback_args_t *p_args)
    void const  *  p_context
    IRQn_Type  irq
    uint8_t  channel
} acmphs_instance_ctrl_t
```

11.1.1.1 open

```
uint32_t ::open
```

11.1.1.2 p_reg

```
R_ACMPHS0_Type* ::p_reg
```

11.1.1.3 pin_output

```
comparator_pin_output_t::pin_output
```

11.1.1.4 p_callback

```
void(* ::p_callback) ( *p_args)
```

11.1.1.5 p_context

```
void const* ::p_context
```

11.1.1.6 irq

```
IRQn_Type ::irq
```

11.1.1.7 channel

```
uint8_t ::channel
```

11.1.2 acmplp_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    R_ACMPLP_Type *  p_reg
    comparator_pin_output_t  pin_output
    void(*  p_callback)(comparator_callback_args_t *p_args)
    void const  *  p_context
    IRQn_Type  irq
    uint8_t  channel
    uint8_t  output_enabled
    comparator_polarity_invert_t  invert
} acmplp_instance_ctrl_t
```

11.1.2.1 open

```
uint32_t ::open
```

11.1.2.2 p_reg

```
R_ACMPLP_Type* ::p_reg
```

11.1.2.3 pin_output

```
comparator_pin_output_t::pin_output
```

11.1.2.4 p_callback

```
void(* ::p_callback) ( *p_args)
```

11.1.2.5 p_context

```
void const* ::p_context
```

11.1.2.6 irq

```
IRQn_Type ::irq
```

11.1.2.7 channel

uint8_t ::channel

11.1.2.8 output_enabled

uint8_t ::output_enabled

11.1.2.9 invert

comparator_polarity_invert_t::invert

11.1.3 adc_api_t

```
typedef struct{
    ssp_err_t(*  open)(adc_ctrl_t *const p_ctrl, adc_cfg_t const *const p_cfg)
    ssp_err_t(*  scanCfg)(adc_ctrl_t *const p_ctrl, adc_channel_cfg_t const
*const p_channel_cfg)
    ssp_err_t(*  scanStart)(adc_ctrl_t *const p_ctrl)
    ssp_err_t(*  scanStop)(adc_ctrl_t *const p_ctrl)
    ssp_err_t(*  scanStatusGet)(adc_ctrl_t *const p_ctrl)
    ssp_err_t(*  read)(adc_ctrl_t *const p_ctrl, adc_register_t const reg_id,
uint16_t *const p_data)
    ssp_err_t(*  read32)(adc_ctrl_t *const p_ctrl, adc_register_t const reg_id,
uint32_t *const p_data)
    ssp_err_t(*  sampleStateCountSet)(adc_ctrl_t *const p_ctrl,
adc_sample_state_t *p_sample)
    ssp_err_t(*  calibrate)(adc_ctrl_t *const p_ctrl, void *const p_extend)
    ssp_err_t(*  offsetSet)(adc_ctrl_t *const p_ctrl, adc_register_t const
reg_id, int32_t const offset)
    ssp_err_t(*  close)(adc_ctrl_t *const p_ctrl)
    ssp_err_t(*  infoGet)(adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} adc_api_t
```

11.1.4 adc_callback_args_t

```
typedef struct{
    uint16_t  unit
    adc_cb_event_t  event
    void const  *  p_context
    adc_register_t  channel
} adc_callback_args_t
```

11.1.4.1 unit

uint16_t adc_callback_args_t::unit

Brief description

ADC device in use.

11.1.4.2 event

adc_cb_event_t::event

Brief description

ADC callback event.

11.1.4.3 p_context

void const* adc_callback_args_t::p_context

Brief description

Placeholder for user data.

11.1.4.4 channel

adc_register_t::channel

Brief description

Channel of conversion result. Only valid for ADC_EVENT_CONVERSION_COMPLETE.

11.1.5 adc_cfg_t

```
typedef struct{
    uint16_t  unit
    adc_mode_t  mode
    adc_resolution_t  resolution
    adc_alignment_t  alignment
    adc_add_t  add_average_count
    adc_clear_t  clearing
    adc_trigger_t  trigger
    adc_trigger_t  trigger_group_b
    uint8_t  scan_end_ipl
    uint8_t  scan_end_b_ipl
    bool  calib_adc_skip
    void(*  p_callback)(adc_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} adc_cfg_t
```

11.1.5.1 unit

uint16_t adc_cfg_t::unit

Brief description

ADC Unit to be used.

11.1.5.2 mode

adc_mode_t::mode

Brief description

ADC operation mode.

11.1.5.3 resolution

adc_resolution_t::resolution

Brief description

ADC resolution 8, 10, or 12-bit.

11.1.5.4 alignment

adc_alignment_t::alignment

Brief description

Specify left or right alignment; ignored if addition used.

11.1.5.5 add_average_count

adc_add_t::add_average_count

Brief description

Add or average samples.

11.1.5.6 clearing

adc_clear_t::clearing

Brief description

Clear after read.

11.1.5.7 trigger

adc_trigger_t::trigger

Brief description

Default and Group A trigger source.

11.1.5.8 trigger_group_b

`adc_trigger_t::trigger_group_b`

Brief description

Group B trigger source; valid only for group mode.

11.1.5.9 scan_end_ipl

`uint8_t adc_cfg_t::scan_end_ipl`

Brief description

Scan end interrupt priority.

11.1.5.10 scan_end_b_ipl

`uint8_t adc_cfg_t::scan_end_b_ipl`

Brief description

Scan end group B interrupt priority.

11.1.5.11 calib_adc_skip

`bool adc_cfg_t::calib_adc_skip`

Brief description

Option to perform calibration when channels are configured.

11.1.5.12 p_callback

`void(* adc_cfg_t::p_callback) (adc_callback_args_t *p_args)`

Brief description

Callback function; set to NULL for none.

11.1.5.13 p_context

`void const* adc_cfg_t::p_context`

Brief description

Placeholder for user data. Passed to the user callback in adc_api_t::adc_callback_args_t.

11.1.5.14 p_extend

```
void const* adc_cfg_t::p_extend
```

Brief description

Extension parameter for hardware specific settings.

11.1.6 adc_channel_cfg_t

```
typedef struct{
    uint32_t  scan_mask
    uint32_t  scan_mask_group_b
    adc_group_a_t  priority_group_a
    uint32_t  add_mask
    uint8_t  sample_hold_mask
    uint8_t  sample_hold_states
} adc_channel_cfg_t
```

11.1.6.1 scan_mask

```
uint32_t adc_channel_cfg_t::scan_mask
```

Detailed description

Channels/bits: bit 0 is ch0; bit 15 is ch15. Use #define ADC_MASK_xxx from r_adc.h.

11.1.6.2 scan_mask_group_b

```
uint32_t adc_channel_cfg_t::scan_mask_group_b
```

Brief description

Valid for group modes. Use #define ADC_MASK_xxx from r_adc.h.

11.1.6.3 priority_group_a

```
adc_group_a_t::priority_group_a
```

Brief description

Valid for group modes.

11.1.6.4 add_mask

```
uint32_t adc_channel_cfg_t::add_mask
```

Brief description

Valid if add enabled in Open(). Use #define ADC_MASK_xxx from r_adc.h.

11.1.6.5 sample_hold_mask

```
uint8_t adc_channel_cfg_t::sample_hold_mask
```

Brief description

Channels/bits 0-2. Use #define ADC_MASK_xxx from r_adc.h.

11.1.6.6 sample_hold_states

```
uint8_t adc_channel_cfg_t::sample_hold_states
```

Brief description

Number of states to be used for sample and hold. Affects channels 0-2.

11.1.7 adc_info_t

```
typedef struct{
    __I uint16_t *  p_address
    uint32_t  length
    transfer_size_t  transfer_size
    elc_peripheral_t  elc_peripheral
    elc_event_t  elc_event
    uint32_t  calibration_data
    int16_t  slope_microvolts
    bool  calibration_ongoing
} adc_info_t
```

11.1.7.1 p_address

```
__I uint16_t* adc_info_t::p_address
```

Brief description

The address to start reading the data from.

11.1.7.2 length

```
uint32_t adc_info_t::length
```

Brief description

The total number of transfers to read.

11.1.7.3 transfer_size

```
transfer_size_t::transfer_size
```

Brief description

The size of each transfer.

11.1.7.4 elc_peripheral

`elc_peripheral_t::elc_peripheral`

Brief description

Name of the peripheral in the ELC list.

11.1.7.5 elc_event

`elc_event_t::elc_event`

Brief description

Name of the ELC event for the peripheral.

11.1.7.6 calibration_data

`uint32_t adc_info_t::calibration_data`

Detailed description

Temperature sensor calibration data (0xFFFFFFFF if unsupported). Refer to hardware manual for steps on using slope with calibration data to determine temperature

11.1.7.7 slope_microvolts

`int16_t adc_info_t::slope_microvolts`

Brief description

Temperature sensor slope in microvolts/°C.

11.1.7.8 calibration_ongoing

`bool adc_info_t::calibration_ongoing`

Brief description

Calibration is in progress.

11.1.8 adc_instance_ctrl_t

```
typedef struct{
    uint16_t  unit
    int16_t  slope_microvolts
```

```
    adc_mode_t  mode
    adc_alignment_t  alignment
    uint8_t  max_resolution
    uint8_t  pga_available
    uint8_t  tsn_ctrl_available
    uint8_t  tsn_calib_available
    uint8_t  adc_calib_available
    R_TSN_Control_Type *  p_tsn_ctrl_regs
    R_TSN_Calibration_Type *  p_tsn_calib_regs
    void const  *  p_context
    void *  p_reg
    void(*  callback)(adc_callback_args_t *p_args)
    adc_trigger_t  trigger
    uint32_t  opened
    uint32_t  scan_mask
    IRQn_Type  scan_end_irq
    IRQn_Type  scan_end_b_irq
} adc_instance_ctrl_t
```

11.1.8.1 unit

`uint16_t ::unit`

Brief description

ADC Unit in use.

11.1.8.2 slope_microvolts

`int16_t ::slope_microvolts`

Brief description

Temperature sensor slope in microvolts/°C.

11.1.8.3 mode

`adc_mode_t::mode`

Brief description

operational mode

11.1.8.4 alignment

`adc_alignment_t::alignment`

Brief description

alignment

11.1.8.5 max_resolution

```
uint8_t ::max_resolution
```

Brief description

ADC max resolution: 8, 10, 12, or 14-bit.

11.1.8.6 pga_available

```
uint8_t ::pga_available
```

Brief description

PGA available or not on MCU.

11.1.8.7 tsn_ctrl_available

```
uint8_t ::tsn_ctrl_available
```

Brief description

Availability of TSN control register.

11.1.8.8 tsn_calib_available

```
uint8_t ::tsn_calib_available
```

Brief description

Availability of TSn calibration register.

11.1.8.9 adc_calib_available

```
uint8_t ::adc_calib_available
```

Brief description

Availability of ADC calibration feature.

11.1.8.10 p_tsn_ctrl_regs

```
R_TSN_Control_Type* ::p_tsn_ctrl_regs
```

Brief description

Pointer to temperature control register.

11.1.8.11 p_tsn_calib_regs

```
R_TSN_Calibration_Type* ::p_tsn_calib_regs
```

Brief description

Pointer to temperature calibration register.

11.1.8.12 p_context

```
void const* ::p_context
```

Brief description

Placeholder for user data.

11.1.8.13 p_reg

```
void* ::p_reg
```

Brief description

Base register for this unit.

11.1.8.14 callback

```
void(* ::callback) ( *p_args)
```

Brief description

User callback pointer.

11.1.8.15 trigger

```
adc_trigger_t::trigger
```

Brief description

Trigger defined for normal mode.

11.1.8.16 opened

```
uint32_t ::opened
```

Brief description

Boolean to verify that the Unit has been initialized.

11.1.8.17 scan_mask

```
uint32_t ::scan_mask
```

Brief description

Scan mask used for Normal scan.

11.1.8.18 scan_end_irq

```
IRQn_Type ::scan_end_irq
```

Brief description

Scan end IRQ number.

11.1.8.19 scan_end_b_irq

```
IRQn_Type ::scan_end_b_irq
```

Brief description

Scan end group B IRQ number.

11.1.9 adc_instance_t

```
typedef struct{
    adc_ctrl_t *  p_ctrl
    adc_cfg_t const  *  p_cfg
    adc_channel_cfg_t const  *  p_channel_cfg
    adc_api_t const  *  p_api
} adc_instance_t
```

11.1.9.1 p_ctrl

```
adc_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.9.2 p_cfg

```
adc_cfg_t::p_cfg
```

Brief description

Pointer to the configuration structure for this instance.

11.1.9.3 p_channel_cfg

```
adc_channel_cfg_t::p_channel_cfg
```

Brief description

Pointer to the channel configuration structure for this instance.

11.1.9.4 p_api

adc_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.10 adc_on_sdadc_cfg_t

```
typedef struct{
    uint8_t  calibration_end_ipl
    uint8_t  conversion_end_ipl
    bool  skip_internal_calibration
    sdadc_vref_src_t  vref_src
    sdadc_vref_voltage_t  vref_voltage
    sdadc_channel_cfg_t const  *  p_channel_cfgs[SDADC_MAX_NUM_CHANNELS]
} adc_on_sdadc_cfg_t
```

11.1.10.1 calibration_end_ipl

uint8_t ::calibration_end_ipl

Brief description

Calibration end interrupt priority.

11.1.10.2 conversion_end_ipl

uint8_t ::conversion_end_ipl

Brief description

Conversion end interrupt priority.

11.1.10.3 skip_internal_calibration

bool ::skip_internal_calibration

Brief description

Whether to skip internal calibration of of the PGA during open.

11.1.10.4 vref_src

sdadc_vref_src_t::vref_src

Brief description

Source of Vref (internal or external)

11.1.10.5 vref_voltage

sdadc_vref_voltage_t::vref_voltage

Detailed description

Voltage of Vref, required for both internal and external Vref. If Vref is from an external source, the voltage must match the specified voltage within 3%.

11.1.10.6 p_channel_cfgs

sdadc_channel_cfg_t::p_channel_cfgs

Brief description

Configuration for each channel, set to NULL if unused.

11.1.11 adc_sample_state_t

```
typedef struct{
    adc_sample_state_reg_t  reg_id
    uint8_t  num_states
} adc_sample_state_t
```

11.1.11.1 reg_id

adc_sample_state_reg_t::reg_id

Brief description

Sample state register ID.

11.1.11.2 num_states

uint8_t adc_sample_state_t::num_states

Brief description

Number of sampling states for conversion. Ch16-20/21 use the same value.

11.1.12 aes_api_t

```
typedef struct{
    uint32_t(*  open)(aes_ctrl_t *const p_ctrl, aes_cfg_t const *const p_cfg)
    uint32_t(*  createKey)(aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t
*p_key)
    uint32_t(*  encrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key,
uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)
    uint32_t(*  addAdditionalAuthenticationData)(aes_ctrl_t *const p_ctrl, const
```

```
uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source)
    uint32_t(*  encryptFinal)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key,
uint32_t *p_iv, uint32_t input_num_words, uint32_t *p_source, uint32_t
output_num_words, uint32_t *p_dest)
    uint32_t(*  decrypt)(aes_ctrl_t *const p_ctrl, const uint32_t *p_key,
uint32_t *p_iv, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest)
    uint32_t(*  setGcmTag)(aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t
*p_source)
    uint32_t(*  getGcmTag)(aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t
*p_dest)
    uint32_t(*  close)(aes_ctrl_t *const p_ctrl)
    uint32_t(*  zeroPaddingEncrypt)(aes_ctrl_t *const p_ctrl, const uint32_t
*p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest)
    uint32_t(*  zeroPaddingDecrypt)(aes_ctrl_t *const p_ctrl, const uint32_t
*p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest)
    uint32_t(*  versionGet)(ssp_version_t *const p_version)
} aes_api_t
```

11.1.13 aes_cfg_t

```
typedef struct{
    crypto_api_t const  *  p_crypto_api
} aes_cfg_t
```

11.1.13.1 p_crypto_api

```
const* aes_cfg_t::p_crypto_api
```

Brief description

pointer to crypto engine api

11.1.14 aes_ctrl_t

```
typedef struct{
    crypto_ctrl_t *  p_crypto_ctrl
    crypto_api_t const  *  p_crypto_api
    uint32_t  work_buffer[DRV_AES_CONTEXT_BUFFER_SIZE]
} aes_ctrl_t
```

11.1.14.1 p_crypto_ctrl

```
* aes_ctrl_t::p_crypto_ctrl
```

Brief description

pointer to crypto engine control structure

11.1.14.2 p_crypto_api

```
const* aes_ctrl_t::p_crypto_api
```

Brief description

pointer to crypto engine API

11.1.14.3 work_buffer

```
uint32_t aes_ctrl_t::work_buffer[DRV_AES_CONTEXT_BUFFER_SIZE]
```

Brief description

Examples: AES-GCM mode uses this for storing authentication tag etc.

Detailed description

used for storing context/state of the Cipher

11.1.15 aes_instance_t

```
typedef struct{
    aes_ctrl_t *  p_ctrl
    aes_cfg_t const  *  p_cfg
    aes_api_t const  *  p_api
} aes_instance_t
```

11.1.15.1 p_ctrl

```
aes_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.15.2 p_cfg

```
aes_cfg_t::p_cfg
```

Brief description

Pointer to the configuration structure for this instance.

11.1.15.3 p_api

```
aes_api_t::p_api
```

Brief description

Pointer to the API structure for this instance.

11.1.16 agt_instance_ctrl_t

```
typedef struct{
    void(*  p_callback)(timer_callback_args_t *p_args)
    void const  *  p_context
    void *  p_reg
    uint32_t  open
    uint16_t  period
    uint8_t  channel
    IRQn_Type  irq
    timer_mode_t  mode
} agt_instance_ctrl_t
```

11.1.16.1 p_callback

```
void(* ::p_callback) ( *p_args)
```

Detailed description

Callback provided when a timer ISR occurs. NULL indicates no CPU interrupt.

11.1.16.2 p_context

```
void const* ::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in timer_callback_args_t.

11.1.16.3 p_reg

```
void* ::p_reg
```

Brief description

Base register for this channel.

11.1.16.4 open

```
uint32_t ::open
```

Brief description

Whether or not channel is open.

11.1.16.5 period

```
uint16_t ::period
```

Brief description

Current timer period (counts)

11.1.16.6 channel

`uint8_t ::channel`

Brief description

Channel number.

11.1.16.7 irq

`IRQn_Type ::irq`

Brief description

Counter overflow IRQ number.

11.1.16.8 mode

`timer_mode_t::mode`

Brief description

Timer mode.

11.1.17 arc4_api_t

```
typedef struct{
    uint32_t(*  open)(arc4_ctrl_t *const p_ctrl, arc4_cfg_t const *const p_cfg)
    uint32_t(*  keySet)(arc4_ctrl_t *const p_ctrl, uint32_t length, uint8_t const
*p_key)
    uint32_t(*  arc4Process)(arc4_ctrl_t *const p_ctrl, uint32_t num_bytes,
uint8_t *p_source, uint8_t *p_dest)
    uint32_t(*  close)(arc4_ctrl_t *const p_ctrl)
    uint32_t(*  versionGet)(ssp_version_t *const p_version)
} arc4_api_t
```

11.1.18 arc4_cfg_t

```
typedef struct{
    crypto_api_t const  *  p_crypto_api
    uint32_t  length
    uint8_t const  *  p_key
} arc4_cfg_t
```

11.1.18.1 p_crypto_api

```
const* arc4_cfg_t::p_crypto_api
```

Brief description

pointer to crypto engine api

11.1.18.2 length

```
uint32_t arc4_cfg_t::length
```

Brief description

Length of p_key in bytes.

11.1.18.3 p_key

```
uint8_t const* arc4_cfg_t::p_key
```

Brief description

ARC4 key to use for encrypt or decrypto operations.

11.1.19 arc4_ctrl_t

```
typedef struct{
    crypto_ctrl_t *  p_crypto_ctrl
    crypto_api_t const  *  p_crypto_api
    uint32_t  state
    uint32_t  work_buffer[DRV_ARC4_CONTEXT_BUFFER_SIZE]
    bsp_lock_t  open
} arc4_ctrl_t
```

11.1.19.1 p_crypto_ctrl

```
* arc4_ctrl_t::p_crypto_ctrl
```

Brief description

pointer to crypto engine control structure

11.1.19.2 p_crypto_api

```
const* arc4_ctrl_t::p_crypto_api
```

Brief description

pointer to crypto engine API

11.1.19.3 state

uint32_t arc4_ctrl_t::state

Brief description

used to identify state of the ARC4 control block

11.1.19.4 work_buffer

uint32_t arc4_ctrl_t::work_buffer[DRV_ARC4_CONTEXT_BUFFER_SIZE]

11.1.19.5 open

bsp_lock_t::open

Brief description

indicates whether driver is opened with this control block

Detailed description

used for storing context of the cipher < ARC4 uses this for storing the sbox results for the next encrypt/ decrypt operations

11.1.20 arc4_instance_t

```
typedef struct{
    arc4_ctrl_t *  p_ctrl
    arc4_cfg_t const  *  p_cfg
    arc4_api_t const  *  p_api
} arc4_instance_t
```

11.1.20.1 p_ctrl

arc4_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.20.2 p_cfg

arc4_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.20.3 p_api

```
arc4_api_t::p_api
```

Brief description

Pointer to the API structure for this instance.

11.1.21 bsp_feature_acmphs_t

```
typedef struct{
    uint32_t  min_wait_time_us
} bsp_feature_acmphs_t
```

11.1.21.1 min_wait_time_us

```
uint32_t ::min_wait_time_us
```

Brief description

Minimum stabilization wait time in microseconds.

11.1.22 bsp_feature_adc_t

```
typedef struct{
    uint8_t  has_sample_hold_reg
    uint8_t  group_b_sensors_allowed
    uint8_t  sensors_exclusive
    uint8_t  tsn_calibration_available
    uint8_t  tsn_control_available
    int16_t  tsn_slope
    uint32_t  sensor_min_sampling_time
    uint32_t  clock_source
    uint8_t  addition_supported
    uint8_t  calibration_reg_available
} bsp_feature_adc_t
```

11.1.22.1 has_sample_hold_reg

```
uint8_t ::has_sample_hold_reg
```

Brief description

Whether or not sample and hold registers are present.

11.1.22.2 group_b_sensors_allowed

```
uint8_t ::group_b_sensors_allowed
```

Brief description

Whether or not sensors are allowed on group b.

11.1.22.3 sensors_exclusive

```
uint8_t ::sensors_exclusive
```

Brief description

Whether or not sensors can be used with other sensors/channels.

11.1.22.4 tsn_calibration_available

```
uint8_t ::tsn_calibration_available
```

Brief description

Identify if the TSN calibration data is available.

11.1.22.5 tsn_control_available

```
uint8_t ::tsn_control_available
```

Brief description

Identify if the TSN control register is available.

11.1.22.6 tsn_slope

```
int16_t ::tsn_slope
```

Brief description

TSN slope in micro-volts/°C.

11.1.22.7 sensor_min_sampling_time

```
uint32_t ::sensor_min_sampling_time
```

Brief description

The minimum sampling time required by the on-chip temperature and voltage sensor in nsec.

11.1.22.8 clock_source

```
uint32_t ::clock_source
```

Brief description

The conversion clock used by the ADC peripheral.

11.1.22.9 addition_supported

```
uint8_t ::addition_supported
```

Brief description

Whether addition is supported or not in this MCU.

11.1.22.10 calibration_reg_available

```
uint8_t ::calibration_reg_available
```

Brief description

Whether CALEXE register is available.

11.1.23 bsp_feature_can_t

```
typedef struct{
    uint8_t  mclock_only
    uint8_t  check_pclkb_ratio
    uint8_t  clock
} bsp_feature_can_t
```

11.1.23.1 mclock_only

```
uint8_t ::mclock_only
```

Brief description

Whether or not MCLK is the only valid clock.

11.1.23.2 check_pclkb_ratio

```
uint8_t ::check_pclkb_ratio
```

Brief description

Whether clock:PCLKB must be 2:1.

11.1.23.3 clock

```
uint8_t ::clock
```

Brief description

Which clock to compare PCLKB to.

11.1.24 bsp_feature_cgc_t

```
typedef struct{
    uint32_t  high_speed_freq_hz
    uint32_t  middle_speed_max_freq_hz
    uint32_t  low_speed_max_freq_hz
    uint32_t  low_voltage_max_freq_hz
    uint32_t  low_speed_pclk_div_min
    uint32_t  low_voltage_pclk_div_min
    uint32_t  hoco_freq_hz
    uint32_t  main_osc_freq_hz
    uint8_t  modrv_mask
    uint8_t  modrv_shift
    uint8_t  sodrv_mask
    uint8_t  sodrv_shift
    uint8_t  pll_div_max
    uint8_t  pll_mul_min
    uint8_t  pll_mul_max
    uint8_t  mainclock_drive
    uint32_t  iclk_div
    uint32_t  pllccr_type
    uint32_t  pll_src_configurable
    uint32_t  has_subosc_speed
    uint32_t  has_lcd_clock
    uint32_t  has_sdram_clock
    uint32_t  has_usb_clock_div
    uint32_t  has_pclka
    uint32_t  has_pclkb
    uint32_t  has_pclkc
    uint32_t  has_pclkd
    uint32_t  has_fclk
    uint32_t  has_bclk
    uint32_t  has_sdadc_clock
} bsp_feature_cgc_t
```

11.1.24.1 high_speed_freq_hz

```
uint32_t ::high_speed_freq_hz
```

Brief description

Frequency above which high speed mode must be used.

11.1.24.2 middle_speed_max_freq_hz

```
uint32_t ::middle_speed_max_freq_hz
```

Brief description

Max frequency for middle speed, 0 indicates not available.

11.1.24.3 low_speed_max_freq_hz

```
uint32_t ::low_speed_max_freq_hz
```

Brief description

Max frequency for low speed, 0 indicates not available.

11.1.24.4 low_voltage_max_freq_hz

```
uint32_t ::low_voltage_max_freq_hz
```

Brief description

Max frequency for low voltage, 0 indicates not available.

11.1.24.5 low_speed_pclk_div_min

```
uint32_t ::low_speed_pclk_div_min
```

Brief description

Minimum divisor for peripheral clocks when using oscillator stop detect.

11.1.24.6 low_voltage_pclk_div_min

```
uint32_t ::low_voltage_pclk_div_min
```

Brief description

Minimum divisor for peripheral clocks when using oscillator stop detect.

11.1.24.7 hoco_freq_hz

```
uint32_t ::hoco_freq_hz
```

Brief description

HOCO frequency.

11.1.24.8 main_osc_freq_hz

```
uint32_t ::main_osc_freq_hz
```

Brief description

Main oscillator frequency.

11.1.24.9 modrv_mask

```
uint8_t ::modrv_mask
```

Brief description

Mask for MODRV in MOMCR.

11.1.24.10 modrv_shift

```
uint8_t ::modrv_shift
```

Brief description

Offset of lowest bit of MODRV in MOMCR.

11.1.24.11 sodrv_mask

```
uint8_t ::sodrv_mask
```

Brief description

Mask for SODRV in SOMCR.

11.1.24.12 sodrv_shift

```
uint8_t ::sodrv_shift
```

Brief description

Offset of lowest bit of SODRV in SOMCR.

11.1.24.13 pll_div_max

```
uint8_t ::pll_div_max
```

Brief description

Maximum PLL divisor.

11.1.24.14 pll_mul_min

```
uint8_t ::pll_mul_min
```

Brief description

Minimum PLL multiplier.

11.1.24.15 pll_mul_max

```
uint8_t ::pll_mul_max
```

Brief description

Maximum PLL multiplier.

11.1.24.16 mainclock_drive

```
uint8_t ::mainclock_drive
```

Brief description

Main clock drive capacity.

11.1.24.17 iclk_div

```
uint32_t ::iclk_div
```

Brief description

ICLK divisor.

11.1.24.18 pllccr_type

```
uint32_t ::pllccr_type
```

Brief description

0: No PLL, 1: PLLCCR, 2: PLLCCR2

11.1.24.19 pll_src_configurable

```
uint32_t ::pll_src_configurable
```

Brief description

Whether or not PLL clock source is configurable.

11.1.24.20 has_subosc_speed

```
uint32_t ::has_subosc_speed
```

Brief description

Whether or not MCU has subosc speed mode.

11.1.24.21 has_lcd_clock

```
uint32_t ::has_lcd_clock
```

Brief description

Whether or not MCU has LCD clock.

11.1.24.22 has_sdram_clock

```
uint32_t ::has_sdram_clock
```

Brief description

Whether or not MCU has SDRAM clock.

11.1.24.23 has_usb_clock_div

```
uint32_t ::has_usb_clock_div
```

Brief description

Whether or not MCU has USB clock divisor.

11.1.24.24 has_pclka

```
uint32_t ::has_pclka
```

Brief description

Whether or not MCU has PCLKA clock.

11.1.24.25 has_pclkb

```
uint32_t ::has_pclkb
```

Brief description

Whether or not MCU has PCLKB clock.

11.1.24.26 has_pclkc

```
uint32_t ::has_pclkc
```

Brief description

Whether or not MCU has PCLKC clock.

11.1.24.27 has_pclkd

```
uint32_t ::has_pclkd
```

Brief description

Whether or not MCU has PCLKD clock.

11.1.24.28 has_fclk

```
uint32_t ::has_fclk
```

Brief description

Whether or not MCU has FCLK clock.

11.1.24.29 has_bclk

```
uint32_t ::has_bclk
```

Brief description

Whether or not MCU has BCLK clock.

11.1.24.30 has_sdadc_clock

```
uint32_t ::has_sdadc_clock
```

Brief description

Whether or not MCU has SDADC clock.

11.1.25 bsp_feature_ctsu_t

```
typedef struct{
    uint8_t  ctsucr0_mask
    uint8_t  ctsucr1_mask
    uint8_t  ctsumch0_mask
    uint8_t  ctsumch1_mask
    uint8_t  ctsuchac_register_count
    uint8_t  ctsuchtrc_register_count
} bsp_feature_ctsu_t
```

11.1.25.1 ctsucr0_mask

```
uint8_t ::ctsucr0_mask
```

Brief description

Mask of valid bits in CTSUCR0.

11.1.25.2 ctsucr1_mask

```
uint8_t ::ctsucr1_mask
```

Brief description

Mask of valid bits in CTSUCR1.

11.1.25.3 ctsumch0_mask

```
uint8_t ::ctsumch0_mask
```

Brief description

Mask of valid bits in CTSUMCH0.

11.1.25.4 ctsumch1_mask

```
uint8_t ::ctsumch1_mask
```

Brief description

Mask of valid bits in CTSUMCH1.

11.1.25.5 ctsuchac_register_count

```
uint8_t ::ctsuchac_register_count
```

Brief description

Number of CTSUCHAC registers.

11.1.25.6 ctsuchtrc_register_count

```
uint8_t ::ctsuchtrc_register_count
```

Brief description

Number of CTSUCHTRC registers.

11.1.26 bsp_feature_dac_t

```
typedef struct{
    uint8_t  has_davrefcr
} bsp_feature_dac_t
```

11.1.26.1 has_davrefcr

```
uint8_t ::has_davrefcr
```

Brief description

Whether or not DAC has DAVREFCR register.

11.1.27 bsp_feature_flash_hp

```
typedef struct{
    uint8_t  cf_block_size_write
} bsp_feature_flash_hp
```

11.1.27.1 cf_block_size_write

```
uint8_t bsp_feature_flash_hp::cf_block_size_write
```

Brief description

Code Flash Block size write.

11.1.28 bsp_feature_flash_lp

```
typedef struct{
    uint8_t  flash_clock_src
    uint8_t  flash_cf_macros
    uint32_t  cf_macro_size
} bsp_feature_flash_lp
```

11.1.28.1 flash_clock_src

```
uint8_t bsp_feature_flash_lp::flash_clock_src
```

Brief description

Source clock for Flash (ie. FCLK or ICLK)

11.1.28.2 flash_cf_macros

```
uint8_t bsp_feature_flash_lp::flash_cf_macros
```

Brief description

Number of implemented code flash hardware macros.

11.1.28.3 cf_macro_size

```
uint32_t bsp_feature_flash_lp::cf_macro_size
```

Brief description

The size of the implemented Code Flash macro.

11.1.29 bsp_feature_ioport_t

```
typedef struct{
    uint8_t  has_ethernet
    uint8_t  has_vbatt_pins
} bsp_feature_ioport_t
```

11.1.29.1 has_ethernet

```
uint8_t ::has_ethernet
```

Brief description

Whether or not MCU has Ethernet port configurations.

11.1.29.2 has_vbatt_pins

```
uint8_t ::has_vbatt_pins
```

Brief description

Whether or not MCU has pins on vbatt domain.

11.1.30 bsp_feature_lvd_t

```
typedef struct{
    uint8_t  monitor_1_low_threshold
    uint8_t  monitor_1_hi_threshold
    uint8_t  monitor_2_low_threshold
    uint8_t  monitor_2_hi_threshold
    uint32_t  has_digital_filter
    uint32_t  negation_delay_clock
} bsp_feature_lvd_t
```

11.1.30.1 monitor_1_low_threshold

```
uint8_t ::monitor_1_low_threshold
```

Brief description

Monitor 1 lowest valid voltage threshold.

11.1.30.2 monitor_1_hi_threshold

```
uint8_t ::monitor_1_hi_threshold
```

Brief description

Monitor 1 highest valid voltage threshold.

11.1.30.3 monitor_2_low_threshold

```
uint8_t ::monitor_2_low_threshold
```

Brief description

Monitor 2 lowest valid voltage threshold.

11.1.30.4 monitor_2_hi_threshold

```
uint8_t ::monitor_2_hi_threshold
```

Brief description

Monitor 2 highest valid voltage threshold.

11.1.30.5 has_digital_filter

```
uint32_t ::has_digital_filter
```

Brief description

Whether or not LVD has a digital filter.

11.1.30.6 negation_delay_clock

```
uint32_t ::negation_delay_clock
```

Brief description

Clock required for LVD signal negation delay after reset.

11.1.31 bsp_feature_opamp_t

```
typedef struct{
    uint16_t  min_wait_time_lp_us
    uint16_t  min_wait_time_ms_us
    uint16_t  min_wait_time_hs_us
} bsp_feature_opamp_t
```

11.1.31.1 min_wait_time_lp_us

```
uint16_t ::min_wait_time_lp_us
```

Brief description

Minimum wait time in low power mode.

11.1.31.2 min_wait_time_ms_us

```
uint16_t ::min_wait_time_ms_us
```

Brief description

Minimum wait time in middle speed mode.

11.1.31.3 min_wait_time_hs_us

```
uint16_t ::min_wait_time_hs_us
```

Brief description

Minimum wait time in high speed mode.

11.1.32 bsp_feature_rspi_t

```
typedef struct{
    uint8_t  clock
    uint8_t  has_ssl_level_keep
} bsp_feature_rspi_t
```

11.1.32.1 clock

```
uint8_t ::clock
```

Brief description

Which clock the RSPI is connected to.

11.1.32.2 has_ssl_level_keep

```
uint8_t ::has_ssl_level_keep
```

11.1.33 bsp_feature_sci_t

```
typedef struct{
    uint8_t  clock
} bsp_feature_sci_t
```

11.1.33.1 clock

```
uint8_t ::clock
```

Brief description

Which clock the SCI is connected to.

11.1.34 bsp_feature_sdhi_t

```
typedef struct{
    uint8_t  has_card_detection
    uint32_t  max_clock_frequency
} bsp_feature_sdhi_t
```

11.1.34.1 has_card_detection

```
uint8_t ::has_card_detection
```

Brief description

Whether or not MCU has card detection.

11.1.34.2 max_clock_frequency

```
uint32_t ::max_clock_frequency
```

Brief description

Maximum clock rate supported by the peripheral.

11.1.35 bsp_feature_ssi_t

```
typedef struct{
    uint8_t  fifo_num_stages
} bsp_feature_ssi_t
```

11.1.35.1 fifo_num_stages

```
uint8_t ::fifo_num_stages
```

Brief description

Number of FIFO stages on this MCU.

11.1.36 bsp_leds_t

```
typedef struct{
    uint16_t  led_count
    ioport_port_pin_t const  *  p_leds
} bsp_leds_t
```

11.1.36.1 led_count

```
uint16_t ::led_count
```

Brief description

The number of LEDs on this board.

11.1.36.2 p_leds

`ioport_port_pin_t::p_leds`

Brief description

Pointer to an array of IOPORT pins for controlling LEDs.

11.1.37 bsp_lock_t

```
typedef struct{
    uint8_t  lock
} bsp_lock_t
```

11.1.37.1 lock

`uint8_t ::lock`

Brief description

A uint8_t is used instead of a enum because the size must be 8-bits.

11.1.38 bsp_mstp_bit_t

```
typedef struct{
    uint8_t  bit
    uint8_t  reg
} bsp_mstp_bit_t
```

11.1.38.1 bit

`uint8_t ::bit`

Brief description

Which bit in MSTP register.

11.1.38.2 reg

`uint8_t ::reg`

Brief description

< Special processing required

Detailed description

Which MSTP register

11.1.39 bsp_mstp_data_t

```
typedef struct{
    uint8_t  byte
    bsp_mstp_bit_t  bit
} bsp_mstp_data_t
```

11.1.39.1 byte

```
uint8_t ::byte
```

Brief description

Byte access for comparison.

11.1.39.2 bit

```
::bit
```

11.1.40 cac_api_t

```
typedef struct{
    ssp_err_t(*  open)(cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)
    ssp_err_t(*  read)(cac_ctrl_t *const p_ctrl, uint8_t *const p_status,
uint16_t *const p_counter)
    ssp_err_t(*  close)(cac_ctrl_t *const p_ctrl)
    ssp_err_t(*  stopMeasurement)(cac_ctrl_t *const p_ctrl)
    ssp_err_t(*  startMeasurement)(cac_ctrl_t *const p_ctrl)
    ssp_err_t(*  reset)(cac_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *p_version)
} cac_api_t
```

11.1.41 cac_callback_args_t

```
typedef struct{
    cac_event_t  event
    void const  *  p_context
} cac_callback_args_t
```

11.1.41.1 event

```
cac_event_t::event
```

Brief description

The event can be used to identify what caused the callback (cac ready or error).

11.1.41.2 p_context

```
void const* cac_callback_args_t::p_context
```

Brief description

Placeholder for user data. Set in open function in cac_cfg_t.

11.1.42 cac_cfg_t

```
typedef struct{
    cac_ref_clock_config_t  cac_ref_clock
    cac_meas_clock_config_t  cac_meas_clock
    uint16_t  cac_upper_limit
    uint16_t  cac_lower_limit
    bool  mei_interrupt_enabled
    bool  ovf_interrupt_enabled
    bool  ferr_interrupt_enabled
    bool  continuous_mode
    uint8_t  frequency_error_ipl
    uint8_t  measurement_end_ipl
    uint8_t  overflow_ipl
    void(*  p_callback)(cac_callback_args_t *p_args)
    void const  *  p_extend
    void const  *  p_context
} cac_cfg_t
```

11.1.42.1 cac_ref_clock

```
cac_ref_clock_config_t::cac_ref_clock
```

Brief description

reference clock specific settings

11.1.42.2 cac_meas_clock

```
cac_meas_clock_config_t::cac_meas_clock
```

Brief description

measurement clock specific settings

11.1.42.3 cac_upper_limit

```
uint16_t cac_cfg_t::cac_upper_limit
```

Brief description

the upper limit counter threshold

11.1.42.4 cac_lower_limit

```
uint16_t cac_cfg_t::cac_lower_limit
```

Brief description

the lower limit counter threshold

11.1.42.5 mei_interrupt_enabled

```
bool cac_cfg_t::mei_interrupt_enabled
```

Brief description

True if Measurement Complete interrupt is enabled.

11.1.42.6 ovf_interrupt_enabled

```
bool cac_cfg_t::ovf_interrupt_enabled
```

Brief description

True if Overflow interrupt is enabled.

11.1.42.7 ferr_interrupt_enabled

```
bool cac_cfg_t::ferr_interrupt_enabled
```

Brief description

True if Frequency Error interrupt is enabled.

11.1.42.8 continuous_mode

```
bool cac_cfg_t::continuous_mode
```

Brief description

True if measurement continuously restarts after completing.

11.1.42.9 frequency_error_ipl

```
uint8_t cac_cfg_t::frequency_error_ipl
```

Brief description

Frequency error interrupt priority.

11.1.42.10 measurement_end_ipl

```
uint8_t cac_cfg_t::measurement_end_ipl
```

Brief description

Measurement end interrupt priority.

11.1.42.11 overflow_ipl

```
uint8_t cac_cfg_t::overflow_ipl
```

Brief description

Overflow interrupt priority.

11.1.42.12 p_callback

```
void(* cac_cfg_t::p_callback) (cac_callback_args_t *p_args)
```

Brief description

Callback provided when a CAC interrupt ISR occurs.

11.1.42.13 p_extend

```
void const* cac_cfg_t::p_extend
```

Brief description

CAC hardware dependent configuration */.

11.1.42.14 p_context

```
void const* cac_cfg_t::p_context
```

Brief description

Placeholder for user data. Passed to user callback in cac_callback_args_t.

11.1.43 cac_instance_ctrl_t

```
typedef struct{
    void *  p_reg
    void(*  p_callback)(cac_callback_args_t *cb_data)
    void const  *  p_context
```

```
    IRQn_Type  frequency_error_irq
    IRQn_Type  measurement_end_irq
    IRQn_Type  overflow_irq
    uint32_t  cac_api_open
    bool  cac_continous_mode
    bsp_lock_t  cac_lock
    cac_clock_source_t  measurement_clock
    cac_clock_source_t  reference_clock
    void const  *  p_extend
} cac_instance_ctrl_t
```

11.1.43.1 p_reg

```
void* ::p_reg
```

Brief description

Pointer to register base address.

11.1.43.2 p_callback

```
void(* ::p_callback) ( *cb_data)
```

Brief description

Called from the ISR.

11.1.43.3 p_context

```
void const* ::p_context
```

Brief description

Passed to the callback.

11.1.43.4 frequency_error_irq

```
IRQn_Type ::frequency_error_irq
```

Brief description

Frequency error IRQ number.

11.1.43.5 measurement_end_irq

```
IRQn_Type ::measurement_end_irq
```

Brief description

Measurement end IRQ number.

11.1.43.6 overflow_irq

```
IRQn_Type ::overflow_irq
```

Brief description

Overflow IRQ number.

11.1.43.7 cac_api_open

```
uint32_t ::cac_api_open
```

Brief description

Set to "CAC" once API has been successfully opened.

11.1.43.8 cac_continous_mode

```
bool ::cac_continous_mode
```

Brief description

Set as a result of the Open() call.

11.1.43.9 cac_lock

```
bsp_lock_t::cac_lock
```

Brief description

CAC commands software lock.

11.1.43.10 measurement_clock

```
cac_clock_source_t::measurement_clock
```

Brief description

Clock specified in Open() as the measurement clock.

11.1.43.11 reference_clock

```
cac_clock_source_t::reference_clock
```

Brief description

Clock specified in Open() as the reference clock.

11.1.43.12 p_extend

```
void const* ::p_extend
```

11.1.44 cac_instance_t

```
typedef struct{
    cac_ctrl_t *  p_ctrl
    cac_cfg_t const  *  p_cfg
    cac_api_t const  *  p_api
} cac_instance_t
```

11.1.44.1 p_ctrl

cac_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.44.2 p_cfg

cac_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.44.3 p_api

cac_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.45 cac_meas_clock_config_t

```
typedef struct{
    cac_meas_divider_t  divider
    cac_clock_source_t  clock
} cac_meas_clock_config_t
```

11.1.45.1 divider

cac_meas_divider_t::divider

Brief description

Divider specification for the Measurement clock.

11.1.45.2 clock

cac_clock_source_t::clock

Brief description

Clock source for the Measurement clock.

11.1.46 cac_ref_clock_config_t

```
typedef struct{
    cac_ref_divider_t  divider
    cac_clock_source_t  clock
    cac_ref_digfilter_t  digfilter
    cac_ref_edge_t  edge
} cac_ref_clock_config_t
```

11.1.46.1 divider

cac_ref_divider_t::divider

Brief description

Divider specification for the Reference clock.

11.1.46.2 clock

cac_clock_source_t::clock

Brief description

Clock source for the Reference clock.

11.1.46.3 digfilter

cac_ref_digfilter_t::digfilter

Brief description

Digital filter selection for the CACREF ext clock.

11.1.46.4 edge

cac_ref_edge_t::edge

Brief description

Edge detection for the Reference clock.

11.1.47 can_api_t

```
typedef struct{
    ssp_err_t(*  open)(can_ctrl_t *const p_ctrl, can_cfg_t const *const p_cfg)
    ssp_err_t(*  read)(can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t
*const p_frame)
    ssp_err_t(*  write)(can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t
*const p_frame)
    ssp_err_t(*  close)(can_ctrl_t *const p_ctrl)
    ssp_err_t(*  control)(can_ctrl_t *const p_ctrl, can_command_t const command,
void *p_data)
    ssp_err_t(*  infoGet)(can_ctrl_t *const p_ctrl, can_info_t *const p_info)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} can_api_t
```

11.1.48 can_bit_timing_cfg_t

```
typedef struct{
    uint32_t  baud_rate_prescaler
    can_time_segment1_t  time_segment_1
    can_time_segment2_t  time_segment_2
    can_sync_jump_width_t  synchronization_jump_width
} can_bit_timing_cfg_t
```

11.1.48.1 baud_rate_prescaler

```
uint32_t can_bit_timing_cfg_t::baud_rate_prescaler
```

Brief description

Baud rate prescaler. Valid values: 1 - 1024.

11.1.48.2 time_segment_1

```
can_time_segment1_t::time_segment_1
```

Brief description

Time segment 1 control.

11.1.48.3 time_segment_2

```
can_time_segment2_t::time_segment_2
```

Brief description

Time segment 2 control.

11.1.48.4 synchronization_jump_width

can_sync_jump_width_t::synchronization_jump_width

Brief description

Synchronization jump width.

11.1.49 can_callback_args_t

```
typedef struct{
    uint32_t  channel
    can_event_t  event
    uint32_t  mailbox
    void const  *  p_context
} can_callback_args_t
```

11.1.49.1 channel

uint32_t can_callback_args_t::channel

Brief description

Device channel number.

11.1.49.2 event

can_event_t::event

Brief description

Event code.

11.1.49.3 mailbox

uint32_t can_callback_args_t::mailbox

Brief description

Mailbox number of interrupt source.

11.1.49.4 p_context

void const* can_callback_args_t::p_context

Brief description

Context provided to user during callback.

11.1.50 can_cfg_t

```
typedef struct{
    uint32_t  channel
    can_bit_timing_cfg_t *  p_bit_timing
    can_id_mode_t  id_mode
    uint32_t  mailbox_count
    can_mailbox_t *  p_mailbox
    can_message_mode_t  message_mode
    void(*  p_callback)(can_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
    uint8_t  error_ipl
    uint8_t  mailbox_rx_ipl
    uint8_t  mailbox_tx_ipl
} can_cfg_t
```

11.1.50.1 channel

uint32_t can_cfg_t::channel

Brief description

CAN channel.

11.1.50.2 p_bit_timing

can_bit_timing_cfg_t::p_bit_timing

Brief description

CAN bit timing.

11.1.50.3 id_mode

can_id_mode_t::id_mode

Brief description

Standard or Extended ID mode.

11.1.50.4 mailbox_count

uint32_t can_cfg_t::mailbox_count

Brief description

Number of mailboxes.

11.1.50.5 p_mailbox

```
can_mailbox_t::p_mailbox
```

Brief description

Pointer to mailboxes.

11.1.50.6 message_mode

```
can_message_mode_t::message_mode
```

Brief description

Overwrite message or overrun.

11.1.50.7 p_callback

```
void(* can_cfg_t::p_callback) (can_callback_args_t *p_args)
```

Brief description

Pointer to callback function.

11.1.50.8 p_context

```
void const* can_cfg_t::p_context
```

Brief description

User defined callback context.

11.1.50.9 p_extend

```
void const* can_cfg_t::p_extend
```

Brief description

CAN hardware dependent configuration.

11.1.50.10 error_ipl

```
uint8_t can_cfg_t::error_ipl
```

Brief description

Error interrupt priority.

11.1.50.11 mailbox_rx_ipl

```
uint8_t can_cfg_t::mailbox_rx_ipl
```

Brief description

Receive interrupt priority.

11.1.50.12 mailbox_tx_ipl

```
uint8_t can_cfg_t::mailbox_tx_ipl
```

Brief description

Transmit interrupt priority.

11.1.51 can_error_t

```
typedef struct{
    uint32_t  error
    struct can_error_t::st_error_b  error_b
} can_error_t
```

11.1.51.1 error

```
uint32_t can_error_t::error
```

11.1.51.2 error_b

```
struct can_error_t::st_error_b can_error_t::error_b
```

11.1.52 can_extended_cfg_t

```
typedef struct{
    can_clock_source_t  clock_source
    uint32_t *  p_mailbox_mask
} can_extended_cfg_t
```

11.1.52.1 clock_source

```
can_clock_source_t::clock_source
```

Brief description

Source of the CAN clock.

11.1.52.2 p_mailbox_mask

```
uint32_t* ::p_mailbox_mask
```

Brief description

Mailbox mask, one for every 4 mailboxes.

11.1.53 can_frame_t

```
typedef struct{
    can_id_t  id
    uint8_t  data_length_code
    uint8_t  data[8]
    can_frame_type_t  type
} can_frame_t
```

11.1.53.1 id

`can_id_t::id`

Brief description

CAN id.

11.1.53.2 data_length_code

`uint8_t can_frame_t::data_length_code`

Brief description

CAN Data Length code, number of bytes in the message.

11.1.53.3 data

`uint8_t can_frame_t::data[8]`

Brief description

CAN data, up to 8 bytes.

11.1.53.4 type

`can_frame_type_t::type`

Brief description

Frame type, data or remote frame.

11.1.54 can_info_t

```
typedef struct{
    can_mode_t  operation_mode
```

```
    can_status_t  status
    uint32_t  bit_rate
    uint8_t  error_count_transmit
    uint8_t  error_count_receive
    can_error_t  error_code
} can_info_t
```

11.1.54.1 operation_mode

can_mode_t::operation_mode

Brief description

Can operation mode.

11.1.54.2 status

can_status_t::status

Brief description

CAN status.

11.1.54.3 bit_rate

uint32_t can_info_t::bit_rate

Brief description

CAN bit rate.

11.1.54.4 error_count_transmit

uint8_t can_info_t::error_count_transmit

Brief description

Transmit error count.

11.1.54.5 error_count_receive

uint8_t can_info_t::error_count_receive

Brief description

Receive error count.

11.1.54.6 error_code

can_error_t::error_code

Brief description

Error code, cleared after reading.

11.1.55 can_instance_ctrl_t

```
typedef struct{
    uint32_t  channel
    uint32_t  open
    can_mode_t  operation_mode
    can_id_mode_t  id_mode
    uint32_t  mailbox_count
    can_mailbox_t *  p_mailbox
    can_message_mode_t  message_mode
    can_clock_source_t  clock_source
    void(*  p_callback)(can_callback_args_t *p_args)
    void const  *  p_context
    void *  p_reg
    IRQn_Type  error_irq
    IRQn_Type  mailbox_rx_irq
    IRQn_Type  mailbox_tx_irq
} can_instance_ctrl_t
```

11.1.55.1 channel

```
uint32_t ::channel
```

Brief description

Channel number.

Detailed description

Parameters to control CAN peripheral device

11.1.55.2 open

```
uint32_t ::open
```

Brief description

Open status of channel.

11.1.55.3 operation_mode

```
can_mode_t::operation_mode
```

Brief description

Can operation mode.

11.1.55.4 id_mode

can_id_mode_t::id_mode

Brief description

Standard or Extended ID mode.

11.1.55.5 mailbox_count

uint32_t ::mailbox_count

Brief description

Number of mailboxes.

11.1.55.6 p_mailbox

can_mailbox_t::p_mailbox

Brief description

Pointer to mailboxes.

11.1.55.7 message_mode

can_message_mode_t::message_mode

Brief description

Overwrite message or overrun.

11.1.55.8 clock_source

can_clock_source_t::clock_source

Brief description

Clock source. CANMCLK or PCLKB.

11.1.55.9 p_callback

void(* ::p_callback) (*p_args)

Brief description

Pointer to callback function.

Detailed description

Parameters to process CAN Event

11.1.55.10 p_context

```
void const* ::p_context
```

Brief description

Pointer to the higher level device context.

11.1.55.11 p_reg

```
void* ::p_reg
```

Brief description

Pointer to register base address.

11.1.55.12 error_irq

```
IRQn_Type ::error_irq
```

Brief description

Error IRQ number.

11.1.55.13 mailbox_rx_irq

```
IRQn_Type ::mailbox_rx_irq
```

Brief description

Receive mailbox IRQ number.

11.1.55.14 mailbox_tx_irq

```
IRQn_Type ::mailbox_tx_irq
```

Brief description

Transmit mailbox IRQ number.

11.1.56 can_instance_t

```
typedef struct{
    can_ctrl_t *  p_ctrl
    can_cfg_t const  *  p_cfg
    can_api_t const  *  p_api
} can_instance_t
```

11.1.56.1 p_ctrl

can_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.56.2 p_cfg

can_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.56.3 p_api

can_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.57 can_mailbox_t

```
typedef struct{
    can_id_t  mailbox_id
    can_mailbox_send_receive_t  mailbox_type
    can_frame_type_t  frame_type
} can_mailbox_t
```

11.1.57.1 mailbox_id

can_id_t::mailbox_id

Brief description

Mailbox ID.

11.1.57.2 mailbox_type

can_mailbox_send_receive_t::mailbox_type

Brief description

Receive or Transmit mailbox type.

11.1.57.3 frame_type

can_frame_type_t::frame_type

Brief description

Frame type for receive mailbox.

11.1.58 can_status_t

```
typedef struct{
    uint32_t  status
    struct can_status_t::st_status_b  status_b
} can_status_t
```

11.1.58.1 status

uint32_t can_status_t::status

11.1.58.2 status_b

struct can_status_t::st_status_b can_status_t::status_b

11.1.59 cgc_api_t

```
typedef struct{
    ssp_err_t(*  init)(void)
    ssp_err_t(*  clocksCfg)(cgc_clocks_cfg_t const *const p_clock_cfg)
    ssp_err_t(*  clockStart)(cgc_clock_t clock_source, cgc_clock_cfg_t
*p_clock_cfg)
    ssp_err_t(*  clockStop)(cgc_clock_t clock_source)
    ssp_err_t(*  systemClockSet)(cgc_clock_t clock_source,
cgc_system_clock_cfg_t const *const p_clock_cfg)
    ssp_err_t(*  systemClockGet)(cgc_clock_t *p_clock_source,
cgc_system_clock_cfg_t *p_set_clock_cfg)
    ssp_err_t(*  systemClockFreqGet)(cgc_system_clocks_t clock, uint32_t
*p_freq_hz)
    ssp_err_t(*  clockCheck)(cgc_clock_t clock_source)
    ssp_err_t(*  oscStopDetect)(void(*p_callback)(cgc_callback_args_t *p_args),
bool enable)
    ssp_err_t(*  oscStopStatusClear)(void)
    ssp_err_t(*  busClockOutCfg)(cgc_bclockout_dividers_t divider)
    ssp_err_t(*  busClockOutEnable)(void)
    ssp_err_t(*  busClockOutDisable)(void)
    ssp_err_t(*  clockOutCfg)(cgc_clock_t clock, cgc_clockout_dividers_t divider)
    ssp_err_t(*  clockOutEnable)(void)
    ssp_err_t(*  clockOutDisable)(void)
```

```
    ssp_err_t(*  lcdClockCfg)(cgc_clock_t clock)
    ssp_err_t(*  lcdClockEnable)(void)
    ssp_err_t(*  lcdClockDisable)(void)
    ssp_err_t(*  sdadcClockCfg)(cgc_clock_t clock)
    ssp_err_t(*  sdadcClockEnable)(void)
    ssp_err_t(*  sdadcClockDisable)(void)
    ssp_err_t(*  sdramClockOutEnable)(void)
    ssp_err_t(*  sdramClockOutDisable)(void)
    ssp_err_t(*  usbClockCfg)(cgc_usb_clock_div_t divider)
    ssp_err_t(*  systickUpdate)(uint32_t period_count,
cgc_systick_period_units_t units)
    ssp_err_t(*  versionGet)(ssp_version_t *p_version)
} cgc_api_t
```

11.1.60 cgc_callback_args_t

```
typedef struct{
    cgc_event_t  event
    void const  *  p_context
} cgc_callback_args_t
```

11.1.60.1 event

cgc_event_t::event

Brief description

The event can be used to identify what caused the callback.

11.1.60.2 p_context

void const* cgc_callback_args_t::p_context

Brief description

Placeholder for user data.

11.1.61 cgc_clock_cfg_t

```
typedef struct{
    cgc_clock_t  source_clock
    cgc_pll_div_t  divider
    float  multiplier
} cgc_clock_cfg_t
```

11.1.61.1 source_clock

cgc_clock_t::source_clock

Brief description

PLL source clock (S7G2 only).

11.1.61.2 divider

cgc_pll_div_t::divider

Brief description

PLL divider.

11.1.61.3 multiplier

float cgc_clock_cfg_t::multiplier

Brief description

PLL multiplier.

11.1.62 cgc_clocks_cfg_t

```
typedef struct{
    cgc_clock_t  system_clock
    cgc_clock_cfg_t  pll_cfg
    cgc_system_clock_cfg_t  sys_cfg
    cgc_clock_change_t  loco_state
    cgc_clock_change_t  moco_state
    cgc_clock_change_t  hoco_state
    cgc_clock_change_t  subosc_state
    cgc_clock_change_t  mainosc_state
    cgc_clock_change_t  pll_state
} cgc_clocks_cfg_t
```

11.1.62.1 system_clock

cgc_clock_t::system_clock

Brief description

System clock source enumeration.

11.1.62.2 pll_cfg

cgc_clock_cfg_t::pll_cfg

Brief description

PLL configuration structure.

11.1.62.3 sys_cfg

`cgc_system_clock_cfg_t::sys_cfg`

Brief description

Clock dividers structure.

11.1.62.4 loco_state

`cgc_clock_change_t::loco_state`

Brief description

State of LOCO.

11.1.62.5 moco_state

`cgc_clock_change_t::moco_state`

Brief description

State of MOCO.

11.1.62.6 hoco_state

`cgc_clock_change_t::hoco_state`

Brief description

State of HOCO.

11.1.62.7 subosc_state

`cgc_clock_change_t::subosc_state`

Brief description

State of Sub-oscillator.

11.1.62.8 mainosc_state

`cgc_clock_change_t::mainosc_state`

Brief description

State of Main oscillator.

11.1.62.9 pll_state

cgc_clock_change_t::pll_state

Brief description

State of PLL.

11.1.63 cgc_instance_t

```
typedef struct{
    cgc_clock_cfg_t const  *  p_cfg
    cgc_api_t const  *  p_api
} cgc_instance_t
```

11.1.63.1 p_cfg

cgc_clock_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.63.2 p_api

cgc_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.64 cgc_system_clock_cfg_t

```
typedef struct{
    cgc_sys_clock_div_t  pclka_div
    cgc_sys_clock_div_t  pclkb_div
    cgc_sys_clock_div_t  pclkc_div
    cgc_sys_clock_div_t  pclkd_div
    cgc_sys_clock_div_t  bclk_div
    cgc_sys_clock_div_t  fclk_div
    cgc_sys_clock_div_t  iclk_div
} cgc_system_clock_cfg_t
```

11.1.64.1 pclka_div

cgc_sys_clock_div_t::pclka_div

Brief description

Divider value for PCLKA.

11.1.64.2 pclkb_div

cgc_sys_clock_div_t::pclkb_div

Brief description

Divider value for PCLKB.

11.1.64.3 pclkc_div

cgc_sys_clock_div_t::pclkc_div

Brief description

Divider value for PCLKC.

11.1.64.4 pclkd_div

cgc_sys_clock_div_t::pclkd_div

Brief description

Divider value for PCLKD.

11.1.64.5 bclk_div

cgc_sys_clock_div_t::bclk_div

Brief description

Divider value for BCLK.

11.1.64.6 fclk_div

cgc_sys_clock_div_t::fclk_div

Brief description

Divider value for FCLK.

11.1.64.7 iclk_div

cgc_sys_clock_div_t::iclk_div

Brief description

Divider value for ICLK.

11.1.65 comparator_api_t

```
typedef struct{
    ssp_err_t(*  open)(comparator_ctrl_t *const p_ctrl, comparator_cfg_t const
*const p_cfg)
    ssp_err_t(*  outputEnable)(comparator_ctrl_t *const p_ctrl)
    ssp_err_t(*  infoGet)(comparator_ctrl_t *const p_ctrl, comparator_info_t
*const p_info)
    ssp_err_t(*  statusGet)(comparator_ctrl_t *const p_ctrl, comparator_status_t
*const p_status)
    ssp_err_t(*  close)(comparator_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} comparator_api_t
```

11.1.66 comparator_callback_args_t

```
typedef struct{
    void const  *  p_context
    uint32_t  channel
} comparator_callback_args_t
```

11.1.66.1 p_context

```
void const* comparator_callback_args_t::p_context
```

Detailed description

Placeholder for user data. Set in open function in comparator_cfg_t.

11.1.66.2 channel

```
uint32_t comparator_callback_args_t::channel
```

Brief description

The physical hardware channel that caused the interrupt.

11.1.67 comparator_cfg_t

```
typedef struct{
    uint8_t  channel
    uint8_t  irq_ipl
    comparator_mode_t  mode
    comparator_trigger_t  trigger
    comparator_filter_t  filter
    comparator_polarity_invert_t  invert
    comparator_pin_output_t  pin_output
```

```
    void(*  p_callback)(comparator_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} comparator_cfg_t
```

11.1.67.1 channel

```
uint8_t comparator_cfg_t::channel
```

Brief description

Hardware channel used.

11.1.67.2 irq_ipl

```
uint8_t comparator_cfg_t::irq_ipl
```

Brief description

Interrupt priority.

11.1.67.3 mode

```
comparator_mode_t::mode
```

Brief description

Normal or window mode.

11.1.67.4 trigger

```
comparator_trigger_t::trigger
```

Brief description

Trigger setting.

11.1.67.5 filter

```
comparator_filter_t::filter
```

Brief description

Digital filter clock divisor setting.

11.1.67.6 invert

```
comparator_polarity_invert_t::invert
```

Brief description

Whether to invert output.

11.1.67.7 pin_output

`comparator_pin_output_t::pin_output`

Brief description

Whether to include output on output pin.

11.1.67.8 p_callback

```
void(* comparator_cfg_t::p_callback) (comparator_callback_args_t *p_args)
```

Detailed description

Callback called when comparator event occurs.

11.1.67.9 p_context

```
void const* comparator_cfg_t::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in comparator_callback_args_t.

11.1.67.10 p_extend

```
void const* comparator_cfg_t::p_extend
```

Brief description

Comparator hardware dependent configuration.

11.1.68 comparator_info_t

```
typedef struct{
    uint32_t  min_stabilization_wait_us
} comparator_info_t
```

11.1.68.1 min_stabilization_wait_us

```
uint32_t comparator_info_t::min_stabilization_wait_us
```

Brief description

Minimum stabilization wait time in microseconds.

11.1.69 comparator_instance_t

```
typedef struct{
    comparator_ctrl_t *  p_ctrl
    comparator_cfg_t const  *  p_cfg
    comparator_api_t const  *  p_api
} comparator_instance_t
```

11.1.69.1 p_ctrl

comparator_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.69.2 p_cfg

comparator_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.69.3 p_api

comparator_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.70 comparator_status_t

```
typedef struct{
    comparator_state_t  state
} comparator_status_t
```

11.1.70.1 state

comparator_state_t::state

Brief description

Current comparator state.

11.1.71 crc_api_t

```
typedef struct{
    ssp_err_t(*  open)(crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)
    ssp_err_t(*  close)(crc_ctrl_t *const p_ctrl)
    ssp_err_t(*  crcResultGet)(crc_ctrl_t *const p_ctrl, uint32_t *crc_result)
    ssp_err_t(*  snoopEnable)(crc_ctrl_t *const p_ctrl, uint32_t crc_seed)
    ssp_err_t(*  snoopDisable)(crc_ctrl_t *const p_ctrl)
    ssp_err_t(*  snoopCfg)(crc_ctrl_t *const p_ctrl, crc_snoop_cfg_t *const
p_snoop_cfg)
    ssp_err_t(*  calculate)(crc_ctrl_t *const p_ctrl, void *p_input_buffer,
uint32_t num_bytes, uint32_t crc_seed, uint32_t *p_crc_result)
    ssp_err_t(*  versionGet)(ssp_version_t *version)
} crc_api_t
```

11.1.72 crc_cfg_t

```
typedef struct{
    crc_polynomial_t  polynomial
    crc_bit_order_t  bit_order
    void const  *  p_extend
} crc_cfg_t
```

11.1.72.1 polynomial

crc_polynomial_t::polynomial

Brief description

CRC Generating Polynomial Switching. (GPS)

11.1.72.2 bit_order

crc_bit_order_t::bit_order

Brief description

CRC Calculation Switching (LMS)

11.1.72.3 p_extend

void const* crc_cfg_t::p_extend

Brief description

CRC Hardware Dependent Configuration.

11.1.73 crc_instance_ctrl_t

```
typedef struct{
    R_CRC_Type *  p_reg
    uint32_t  open
    crc_polynomial_t  polynomial
    crc_bit_order_t  bit_order
} crc_instance_ctrl_t
```

11.1.73.1 p_reg

```
R_CRC_Type* ::p_reg
```

Brief description

Pointer to register base address.

11.1.73.2 open

```
uint32_t ::open
```

Brief description

Whether or not channel is open.

11.1.73.3 polynomial

```
crc_polynomial_t::polynomial
```

Brief description

CRC Generating Polynomial Switching (GPS).

11.1.73.4 bit_order

```
crc_bit_order_t::bit_order
```

Brief description

CRC Calculation Switching (LMS).

11.1.74 crc_instance_t

```
typedef struct{
    crc_ctrl_t *  p_ctrl
    crc_cfg_t const  *  p_cfg
    crc_api_t const  *  p_api
} crc_instance_t
```

11.1.74.1 p_ctrl

crc_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.74.2 p_cfg

crc_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.74.3 p_api

crc_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.75 crc_snoop_cfg_t

```
typedef struct{
    uint32_t  snoop_channel
    crc_snoop_direction_t  snoop_direction
} crc_snoop_cfg_t
```

11.1.75.1 snoop_channel

uint32_t crc_snoop_cfg_t::snoop_channel

Brief description

Register Snoop Address (CRCSA)

11.1.75.2 snoop_direction

crc_snoop_direction_t::snoop_direction

Brief description

Snoop-On-Write/Read Switch (CRCSWR)

11.1.76 ctsu_api_t

```
typedef struct{
    ssp_err_t(*  open)(ctsu_ctrl_t *p_ctrl, ctsu_cfg_t *p_cfg)
    ssp_err_t(*  close)(ctsu_ctrl_t *p_ctrl, ctsu_close_option_t opts)
    ssp_err_t(*  scan)(ctsu_ctrl_t *p_ctrl)
    ssp_err_t(*  update)(ctsu_ctrl_t *p_ctrl)
    ssp_err_t(*  read)(ctsu_ctrl_t *p_ctrl, void *p_dest, ctsu_read_t opts, const
ctsu_channel_pair_t *channels, const uint16_t count)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} ctsu_api_t
```

11.1.77 ctsu_callback_args_t

```
typedef struct{
    ctsu_event_t  event
    void const  *  p_context
} ctsu_callback_args_t
```

11.1.77.1 event

ctsu_event_t::event

Brief description

CTSU callback event.

11.1.77.2 p_context

void const* ctsu_callback_args_t::p_context

Brief description

Placeholder for user data.

11.1.78 ctsu_cfg_t

```
typedef struct{
    transfer_instance_t const  *const  p_lower_lvl_transfer_read
    transfer_instance_t const  *const  p_lower_lvl_transfer_write
    ctsu_hw_cfg_t *  p_ctsu_hw_cfg
    ctsu_functions_t *  p_ctsu_functions
    void(*  p_callback)(ctsu_callback_args_t *p_args)
    void *  p_context
    ctsu_process_option_t  ctsu_soft_option
    ctsu_close_option_t  ctsu_close_option
    uint8_t  write_ipl
```

```
    uint8_t  read_ipl
    uint8_t  end_ipl
} ctsu_cfg_t
```

11.1.78.1 p_lower_lvl_transfer_read

`transfer_instance_t::p_lower_lvl_transfer_read`

Brief description

Pointer to the Transfer instance to read results.

11.1.78.2 p_lower_lvl_transfer_write

`transfer_instance_t::p_lower_lvl_transfer_write`

Brief description

Pointer to the Transfer instance to write cfg.

11.1.78.3 p_ctsu_hw_cfg

`ctsu_hw_cfg_t::p_ctsu_hw_cfg`

Brief description

Pointer to a CTSU configuration.

11.1.78.4 p_ctsu_functions

`ctsu_functions_t::p_ctsu_functions`

Brief description

Pointer to a place holder for custom data functions.

11.1.78.5 p_callback

```
void(* ctsu_cfg_t::p_callback) (ctsu_callback_args_t *p_args)
```

Brief description

Callback to the function to use when scan is complete.

11.1.78.6 p_context

```
void* ctsu_cfg_t::p_context
```

Brief description

Pointer to data that should be passed to update_complete notification.

11.1.78.7 ctsu_soft_option

`ctsu_process_option_t::ctsu_soft_option`

Brief description

Software options to use when performing Open and Process.

11.1.78.8 ctsu_close_option

`ctsu_close_option_t::ctsu_close_option`

Brief description

Software options to use when closing touch operation.

11.1.78.9 write_ipl

`uint8_t ctsu_cfg_t::write_ipl`

Brief description

Write interrupt priority.

11.1.78.10 read_ipl

`uint8_t ctsu_cfg_t::read_ipl`

Brief description

Read interrupt priority.

11.1.78.11 end_ipl

`uint8_t ctsu_cfg_t::end_ipl`

Brief description

End interrupt priority.

11.1.79 ctsu_channel_data_mutual_t

```
typedef struct{
    uint16_t  sen_cnt_1
    uint16_t  ref_cnt_1
    uint16_t  sen_cnt_2
```

```
    uint16_t  ref_cnt_2
} ctsu_channel_data_mutual_t
```

11.1.79.1 sen_cnt_1

```
uint16_t ctsu_channel_data_mutual_t::sen_cnt_1
```

Brief description

Raw Sensor count primary reading.

11.1.79.2 ref_cnt_1

```
uint16_t ctsu_channel_data_mutual_t::ref_cnt_1
```

Brief description

Raw reference ICO count primary reading.

11.1.79.3 sen_cnt_2

```
uint16_t ctsu_channel_data_mutual_t::sen_cnt_2
```

Brief description

Raw sensor ICO count secondary reading.

11.1.79.4 ref_cnt_2

```
uint16_t ctsu_channel_data_mutual_t::ref_cnt_2
```

Brief description

Raw reference ICO count secondary reading.

11.1.80 ctsu_channel_data_self_t

```
typedef struct{
    uint16_t  sensor_count
    uint16_t  reference_count
} ctsu_channel_data_self_t
```

11.1.80.1 sensor_count

```
uint16_t ctsu_channel_data_self_t::sensor_count
```

Brief description

Raw sensor count.

11.1.80.2 reference_count

```
uint16_t ctsu_channel_data_self_t::reference_count
```

Brief description

Raw reference count.

11.1.81 ctsu_channel_pair_t

```
typedef struct{
    int8_t  rx
    int8_t  tx
} ctsu_channel_pair_t
```

11.1.81.1 rx

```
int8_t ctsu_channel_pair_t::rx
```

Brief description

Denotes the primary channel.

11.1.81.2 tx

```
int8_t ctsu_channel_pair_t::tx
```

Brief description

Denotes the secondary channel (used only for mutual capacitance mode)

11.1.82 ctsu_channel_setting_t

```
typedef struct{
    uint16_t  ctsussc
    uint16_t  ctsuso0
    uint16_t  ctsuso1
} ctsu_channel_setting_t
```

11.1.82.1 ctsussc

```
volatile uint16_t ctsu_channel_setting_t::ctsussc
```

Brief description

Holds value for the CTSUSSC register.

11.1.82.2 ctsuso0

```
volatile uint16_t ctsu_channel_setting_t::ctsuso0
```

Brief description

Holds value for the CTSUSO0 register.

11.1.82.3 ctsuso1

```
volatile uint16_t ctsu_channel_setting_t::ctsuso1
```

Brief description

Holds value for the CTSUSO1 register.

11.1.83 ctsu_functions_t

```
typedef struct{
    int32_t(*  preFilter)(void *p_args)
    int32_t(*  filter)(volatile uint16_t *output, volatile uint16_t *input)
    int32_t(*  postFilter)(void *p_args)
    int32_t(*  ctsuDecode)(void *p_args)
    int32_t(*  otDriftComp)(void *p_args)
    int32_t(*  rtDriftComp)(void *p_args)
    int32_t(*  otAutoTune)(void *p_args)
    int32_t(*  rtAutoTune)(void *p_args)
} ctsu_functions_t
```

11.1.83.1 preFilter

```
int32_t(* ctsu_functions_t::preFilter) (void *p_args)
```

Brief description

Used for calculating raw data SNR before data gets filtered.

11.1.83.2 filter

```
int32_t(* ctsu_functions_t::filter) (volatile uint16_t *output, volatile uint16_t
*input)
```

Brief description

Weighted averaging using CTSU_CFG_FILTER_DEPTH.

11.1.83.3 postFilter

```
int32_t(* ctsu_functions_t::postFilter) (void *p_args)
```

Brief description

Processing the filter results.

11.1.83.4 ctsuDecode

```
int32_t(* ctsu_functions_t::ctsuDecode) (void *p_args)
```

Brief description

Algorithm to decide if channel is touched or not.

11.1.83.5 otDriftComp

```
int32_t(* ctsu_functions_t::otDriftComp) (void *p_args)
```

Brief description

Algorithm to perform manipulations to baseline, envelope, and thresholds on initialization.

11.1.83.6 rtDriftComp

```
int32_t(* ctsu_functions_t::rtDriftComp) (void *p_args)
```

Brief description

Algorithm to perform run time manipulations to baseline, envelope, and thresholds.

11.1.83.7 otAutoTune

```
int32_t(* ctsu_functions_t::otAutoTune) (void *p_args)
```

Brief description

Function called once on initialization of system.

11.1.83.8 rtAutoTune

```
int32_t(* ctsu_functions_t::rtAutoTune) (void *p_args)
```

Brief description

Function called to auto tune sensor when system is running.

11.1.84 ctsu_hw_cfg_t

```
typedef struct{
    R_CTSU_Type  ctsu_settings
    ctsu_channel_setting_t *  write_settings
    uint16_t *  threshold
```

```
    uint16_t *  hysteresis
    uint16_t *  baseline
    void *  raw_result
    void *  filter_output
    void *  binary_result
    ctsu_channel_pair_t *  excluded
    int8_t  num_excluded
    const uint16_t *  series_resistance
} ctsu_hw_cfg_t
```

11.1.84.1 ctsu_settings

```
R_CTSU_Type ctsu_hw_cfg_t::ctsu_settings
```

Brief description

User defined SFR settings for CR0, CR1, SDPRS, SST, CHACn, CHTRCn, DCLKC.

11.1.84.2 write_settings

```
ctsu_channel_setting_t::write_settings
```

Brief description

User defined initial settings for thresholds for each active channel . Threshold is difference between runtime baseline and filtered output of sensor count.

Detailed description

User defined settings for SSC, SO0, SO1 for each active channel.

11.1.84.3 threshold

```
uint16_t* ctsu_hw_cfg_t::threshold
```

11.1.84.4 hysteresis

```
uint16_t* ctsu_hw_cfg_t::hysteresis
```

Brief description

User defined settings for tolerance in count values.

11.1.84.5 baseline

```
uint16_t* ctsu_hw_cfg_t::baseline
```

Brief description

A baseline of the expected count for each active channel when not touched.

11.1.84.6 raw_result

```
void* ctsu_hw_cfg_t::raw_result
```

Brief description

A pointer to a buffer which will hold raw results of CTSU measurement.

11.1.84.7 filter_output

```
void* ctsu_hw_cfg_t::filter_output
```

Brief description

A pointer to a buffer which will hold output after filtering raw results.

11.1.84.8 binary_result

```
void* ctsu_hw_cfg_t::binary_result
```

Brief description

A pointer to a location where binary data can be stored.

11.1.84.9 excluded

```
ctsu_channel_pair_t::excluded
```

Brief description

A pointer to an array which contains a list of channel pairs which need to be ignored in ascending order of rx and then tx.

11.1.84.10 num_excluded

```
int8_t ctsu_hw_cfg_t::num_excluded
```

Brief description

Number of elements in the array excluded.

11.1.84.11 series_resistance

```
const uint16_t* ctsu_hw_cfg_t::series_resistance
```

Brief description

Resistance of the channel (to determine RC constant when tuning).

11.1.85 ctsu_instance_ctrl_t

```
typedef struct{
    transfer_api_t const  *  p_api_transfer
    transfer_ctrl_t *  p_lowerl_lvl_transfer_read_ctrl
    transfer_ctrl_t *  p_lowerl_lvl_transfer_write_ctrl
    bool  ctsu_opened
    uint8_t  ctsu_unit
    ctsu_hw_cfg_t *  p_ctsu_hw_cfg
    ctsu_process_option_t  ctsu_open_option
    ctsu_process_option_t  ctsu_update_option
    ctsu_close_option_t  ctsu_close_option
    ctsu_action_t  ctsu_process_state
    void(*  p_callback)(ctsu_callback_args_t *p_args)
    void *  p_context
    R_CTSU_Type *  p_reg
} ctsu_instance_ctrl_t
```

11.1.85.1 p_api_transfer

transfer_api_t::p_api_transfer

Brief description

Pointer to lower level Transfer driver function pointers.

11.1.85.2 p_lowerl_lvl_transfer_read_ctrl

transfer_ctrl_t::p_lowerl_lvl_transfer_read_ctrl

Brief description

Pointer to the Transfer Read control.

11.1.85.3 p_lowerl_lvl_transfer_write_ctrl

transfer_ctrl_t::p_lowerl_lvl_transfer_write_ctrl

Brief description

Pointer to the Transfer Write control.

11.1.85.4 ctsu_opened

bool ::ctsu_opened

Brief description

Store initialization state.

11.1.85.5 ctsu_unit

```
uint8_t ::ctsu_unit
```

Brief description

CTSU Unit in use.

11.1.85.6 p_ctsu_hw_cfg

```
ctsu_hw_cfg_t::p_ctsu_hw_cfg
```

Brief description

Pointer to a CTSU configuration.

11.1.85.7 ctsu_open_option

```
ctsu_process_option_t::ctsu_open_option
```

Brief description

Software options to use when performing Open and Process.

11.1.85.8 ctsu_update_option

```
ctsu_process_option_t::ctsu_update_option
```

Brief description

Software options to use when performing parameter Update process.

11.1.85.9 ctsu_close_option

```
ctsu_close_option_t::ctsu_close_option
```

Brief description

Software options to use when closing touch operation.

11.1.85.10 ctsu_process_state

```
ctsu_action_t::ctsu_process_state
```

Brief description

Variable to observe CTSU processing state machine operation.

11.1.85.11 p_callback

```
void(* ::p_callback) ( *p_args)
```

Brief description

Callback to the function to use when updating dependent parameters is complete.

11.1.85.12 p_context

```
void* ::p_context
```

Brief description

Pointer to data that should be passed to update_complete notification.

11.1.85.13 p_reg

```
R_CTSU_Type* ::p_reg
```

Brief description

Pointer to base register address.

11.1.86 ctsu_instance_t

```
typedef struct{
    ctsu_ctrl_t *  p_ctrl
    ctsu_cfg_t *  p_cfg
    ctsu_api_t const  *  p_api
} ctsu_instance_t
```

11.1.86.1 p_ctrl

```
ctsu_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.86.2 p_cfg

```
ctsu_cfg_t::p_cfg
```

Brief description

Pointer to the configuration structure for this instance.

11.1.86.3 p_api

```
ctsu_api_t::p_api
```

Brief description

Pointer to the API structure for this instance.

11.1.87 dac8_extended_cfg_t

```
typedef struct{
    bool  enable_charge_pump
    dac8_mode_t  dac_mode
} dac8_extended_cfg_t
```

11.1.87.1 enable_charge_pump

```
bool ::enable_charge_pump
```

Brief description

Enable DAC charge pump.

11.1.87.2 dac_mode

```
dac8_mode_t::dac_mode
```

Brief description

DAC mode.

11.1.88 dac8_instance_ctrl_t

```
typedef struct{
    void *  p_reg
    uint8_t  channel
    uint8_t  channel_started
    uint32_t  channel_opened
    dac_data_format_t  data_format
} dac8_instance_ctrl_t
```

11.1.88.1 p_reg

```
void* ::p_reg
```

Brief description

Pointer to DAC base register.

11.1.88.2 channel

```
uint8_t ::channel
```

Brief description

ID associated with this DAC channel.

11.1.88.3 channel_started

`uint8_t ::channel_started`

Brief description

DAC operation on channel started.

11.1.88.4 channel_opened

`uint32_t ::channel_opened`

Brief description

DAC channel open.

11.1.88.5 data_format

`dac_data_format_t::data_format`

Brief description

DAC data format.

11.1.89 dac_api_t

```
typedef struct{
    ssp_err_t(*  open)(dac_ctrl_t *p_ctrl, dac_cfg_t const *const p_cfg)
    ssp_err_t(*  close)(dac_ctrl_t *p_ctrl)
    ssp_err_t(*  write)(dac_ctrl_t *p_ctrl, dac_size_t value)
    ssp_err_t(*  start)(dac_ctrl_t *p_ctrl)
    ssp_err_t(*  stop)(dac_ctrl_t *p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *p_version)
    ssp_err_t(*  infoGet)(dac_info_t *const p_info)
} dac_api_t
```

11.1.90 dac_cfg_t

```
typedef struct{
    uint8_t  channel
    bool  ad_da_synchronized
    dac_data_format_t  data_format
    bool  output_amplifier_enabled
    void const  *  p_extend
} dac_cfg_t
```

11.1.90.1 channel

```
uint8_t dac_cfg_t::channel
```

Brief description

ID associated with this DAC channel.

11.1.90.2 ad_da_synchronized

```
bool dac_cfg_t::ad_da_synchronized
```

Brief description

AD/DA synchronization.

11.1.90.3 data_format

```
dac_data_format_t::data_format
```

Brief description

Data format.

11.1.90.4 output_amplifier_enabled

```
bool dac_cfg_t::output_amplifier_enabled
```

Brief description

Output amplifier enable.

11.1.90.5 p_extend

```
void const* dac_cfg_t::p_extend
```

11.1.91 dac_info_t

```
typedef struct{
    uint8_t  bit_width
} dac_info_t
```

11.1.91.1 bit_width

```
uint8_t dac_info_t::bit_width
```

Brief description

Resolution of the DAC.

11.1.92 dac_instance_ctrl_t

```
typedef struct{
    void *  p_reg
    uint8_t  channel
    uint8_t  channel_started
    uint8_t  channel_opened
} dac_instance_ctrl_t
```

11.1.92.1 p_reg

```
void* ::p_reg
```

Brief description

Pointer to DAC base register.

11.1.92.2 channel

```
uint8_t ::channel
```

Brief description

ID associated with this DAC channel.

11.1.92.3 channel_started

```
uint8_t ::channel_started
```

Brief description

DAC operation on channel started.

11.1.92.4 channel_opened

```
uint8_t ::channel_opened
```

Brief description

DAC channel open.

11.1.93 dac_instance_t

```
typedef struct{
    dac_ctrl_t *  p_ctrl
    dac_cfg_t const  *  p_cfg
    dac_api_t const  *  p_api
} dac_instance_t
```

11.1.93.1 p_ctrl

dac_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.93.2 p_cfg

dac_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.93.3 p_api

dac_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.94 display_api_t

```
typedef struct{
    ssp_err_t(*  open)(display_ctrl_t *const p_ctrl, display_cfg_t const *const
p_cfg)
    ssp_err_t(*  close)(display_ctrl_t *const p_ctrl)
    ssp_err_t(*  start)(display_ctrl_t *const p_ctrl)
    ssp_err_t(*  stop)(display_ctrl_t *const p_ctrl)
    ssp_err_t(*  layerChange)(display_ctrl_t const *const p_ctrl,
display_runtime_cfg_t const *const p_cfg, display_frame_layer_t frame)
    ssp_err_t(*  correction)(display_ctrl_t const *const p_ctrl,
display_correction_t const *const p_param)
    ssp_err_t(*  clut)(display_ctrl_t const *const p_ctrl, display_clut_cfg_t
const *const p_clut_cfg, display_frame_layer_t frame)
    ssp_err_t(*  statusGet)(display_ctrl_t const *const p_ctrl, display_status_t
*const p_status)
    ssp_err_t(*  versionGet)(ssp_version_t *p_version)
} display_api_t
```

11.1.95 display_brightness_t

```
typedef struct{
    bool  enable
    uint16_t  r
    uint16_t  g
```

```
    uint16_t  b
} display_brightness_t
```

11.1.95.1 enable

```
bool display_brightness_t::enable
```

Brief description

Brightness Correction On/Off.

11.1.95.2 r

```
uint16_t display_brightness_t::r
```

Brief description

Brightness (DC) adjustment for R channel.

11.1.95.3 g

```
uint16_t display_brightness_t::g
```

Brief description

Brightness (DC) adjustment for G channel.

11.1.95.4 b

```
uint16_t display_brightness_t::b
```

Brief description

Brightness (DC) adjustment for B channel.

11.1.96 display_callback_args_t

```
typedef struct{
    display_event_t  event
    void const  *  p_context
} display_callback_args_t
```

11.1.96.1 event

```
display_event_t::event
```

Brief description

Event code.

11.1.96.2 p_context

```
void const* display_callback_args_t::p_context
```

Brief description

Context provided to user during callback.

11.1.97 display_cfg_t

```
typedef struct{
    display_input_cfg_t  input[DISPLAY_FRAME_LAYER_2+1]
    display_output_cfg_t  output
    display_layer_t  layer[DISPLAY_FRAME_LAYER_2+1]
    uint8_t  line_detect_ipl
    uint8_t  underflow_1_ipl
    uint8_t  underflow_2_ipl
    void(*  p_callback)(display_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} display_cfg_t
```

11.1.97.1 input

```
display_input_cfg_t::input
```

Brief description

Graphics input frame setting.

Detailed description

Generic configuration for display devices

11.1.97.2 output

```
display_output_cfg_t::output
```

Brief description

Graphics output frame setting.

11.1.97.3 layer

```
display_layer_t::layer
```

Brief description

Graphics layer blend setting.

11.1.97.4 line_detect_ipl

```
uint8_t display_cfg_t::line_detect_ipl
```

Brief description

Line detect interrupt priority.

11.1.97.5 underflow_1_ipl

```
uint8_t display_cfg_t::underflow_1_ipl
```

Brief description

Underflow 1 interrupt priority.

11.1.97.6 underflow_2_ipl

```
uint8_t display_cfg_t::underflow_2_ipl
```

Brief description

Underflow 1 interrupt priority.

11.1.97.7 p_callback

```
void(* display_cfg_t::p_callback) (display_callback_args_t *p_args)
```

Brief description

Pointer to callback function.

Detailed description

Configuration for display event processing

11.1.97.8 p_context

```
void const* display_cfg_t::p_context
```

Brief description

User defined context passed into callback function.

11.1.97.9 p_extend

```
void const* display_cfg_t::p_extend
```

Brief description

Display hardware dependent configuration.

Detailed description

Pointer to display peripheral specific configuration

11.1.98 display_clut_cfg_t

```
typedef struct{
    uint32_t *  p_base
    uint16_t  start
    uint16_t  size
} display_clut_cfg_t
```

11.1.98.1 p_base

```
uint32_t* display_clut_cfg_t::p_base
```

Brief description

Pointer to CLUT source data.

11.1.98.2 start

```
uint16_t display_clut_cfg_t::start
```

Brief description

Beginning of CLUT entry to be updated.

11.1.98.3 size

```
uint16_t display_clut_cfg_t::size
```

Brief description

Size of CLUT entry to be updated.

11.1.99 display_clut_t

```
typedef struct{
    uint32_t  color_num
    const uint32_t *  p_clut
} display_clut_t
```

11.1.99.1 color_num

```
uint32_t display_clut_t::color_num
```

Brief description

The number of colors in CLUT.

11.1.99.2 p_clut

```
const uint32_t* display_clut_t::p_clut
```

Brief description

Address of the area storing the CLUT data (in ARGB8888 format)

11.1.100 display_color_t

```
typedef struct{
    uint32_t  argb
    uint8_t  b
    uint8_t  g
    uint8_t  r
    uint8_t  a
    struct{}                byte
    union{}                 union{}
} display_color_t
```

11.1.100.1 argb

```
uint32_t display_color_t::argb
```

11.1.100.2 b

```
uint8_t display_color_t::b
```

Brief description

blue

11.1.100.3 g

```
uint8_t display_color_t::g
```

Brief description

green

11.1.100.4 r

```
uint8_t display_color_t::r
```

Brief description

red

11.1.100.5 a

```
uint8_t display_color_t::a
```

Brief description

a

11.1.100.6 byte

```
See source code for the definition of this member.
```

11.1.100.7 union{}

```
See source code for the definition of this member.
```

11.1.101 display_contrast_t

```
typedef struct{
    bool  enable
    uint8_t  r
    uint8_t  g
    uint8_t  b
} display_contrast_t
```

11.1.101.1 enable

```
bool display_contrast_t::enable
```

Brief description

Contrast Correction On/Off.

11.1.101.2 r

```
uint8_t display_contrast_t::r
```

Brief description

Contrast (gain) adjustment for R channel.

11.1.101.3 g

```
uint8_t display_contrast_t::g
```

Brief description

Contrast (gain) adjustment for G channel.

11.1.101.4 b

```
uint8_t display_contrast_t::b
```

Brief description

Contrast (gain) adjustment for B channel.

11.1.102 display_coordinate_t

```
typedef struct{
    int16_t  x
    int16_t  y
} display_coordinate_t
```

11.1.102.1 x

```
int16_t display_coordinate_t::x
```

Brief description

Coordinate X, this allows to set signed value.

11.1.102.2 y

```
int16_t display_coordinate_t::y
```

Brief description

Coordinate Y, this allows to set signed value.

11.1.103 display_correction_t

```
typedef struct{
    display_brightness_t  brightness
    display_contrast_t  contrast
} display_correction_t
```

11.1.103.1 brightness

```
display_brightness_t::brightness
```

Brief description

Brightness.

11.1.103.2 contrast

display_contrast_t::contrast

Brief description

Contrast.

11.1.104 display_gamma_correction_t

```
typedef struct{
    gamma_correction_t  r
    gamma_correction_t  g
    gamma_correction_t  b
} display_gamma_correction_t
```

11.1.104.1 r

gamma_correction_t::r

Brief description

Gamma correction for R channel.

11.1.104.2 g

gamma_correction_t::g

Brief description

Gamma correction for G channel.

11.1.104.3 b

gamma_correction_t::b

Brief description

Gamma correction for B channel.

11.1.105 display_input_cfg_t

```
typedef struct{
    uint32_t *  p_base
    uint16_t  hsize
    uint16_t  vsize
    uint32_t  hstride
    display_in_format_t  format
    bool  line_descending_enable
```

```
    bool  lines_repeat_enable
    uint16_t  lines_repeat_times
} display_input_cfg_t
```

11.1.105.1 p_base

```
uint32_t* display_input_cfg_t::p_base
```

Brief description

Base address to the frame buffer.

11.1.105.2 hsize

```
uint16_t display_input_cfg_t::hsize
```

Brief description

Horizontal pixel size in a line.

11.1.105.3 vsize

```
uint16_t display_input_cfg_t::vsize
```

Brief description

Vertical pixel size in a frame.

11.1.105.4 hstride

```
uint32_t display_input_cfg_t::hstride
```

Brief description

Memory stride (bytes) in a line.

11.1.105.5 format

```
display_in_format_t::format
```

Brief description

Input format setting.

11.1.105.6 line_descending_enable

```
bool display_input_cfg_t::line_descending_enable
```

Brief description

Line descending enable.

11.1.105.7 lines_repeat_enable

```
bool display_input_cfg_t::lines_repeat_enable
```

Brief description

Line repeat enable.

11.1.105.8 lines_repeat_times

```
uint16_t display_input_cfg_t::lines_repeat_times
```

Brief description

Expected number of line repeating.

11.1.106 display_instance_t

```
typedef struct{
    display_ctrl_t *  p_ctrl
    display_cfg_t const  *  p_cfg
    display_api_t const  *  p_api
} display_instance_t
```

11.1.106.1 p_ctrl

```
display_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.106.2 p_cfg

```
display_cfg_t::p_cfg
```

Brief description

Pointer to the configuration structure for this instance.

11.1.106.3 p_api

```
display_api_t::p_api
```

Brief description

Pointer to the API structure for this instance.

11.1.107 display_layer_t

```
typedef struct{
    display_coordinate_t  coordinate
    display_color_t  bg_color
    display_fade_control_t  fade_control
    uint8_t  fade_speed
} display_layer_t
```

11.1.107.1 coordinate

`display_coordinate_t::coordinate`

Brief description

Blending location (starting point of image)

11.1.107.2 bg_color

`display_color_t::bg_color`

Brief description

Color outside region.

11.1.107.3 fade_control

`display_fade_control_t::fade_control`

Brief description

Layer fade-in/out control on/off.

11.1.107.4 fade_speed

`uint8_t display_layer_t::fade_speed`

Brief description

Layer fade-in/out frame rate.

11.1.108 display_output_cfg_t

```
typedef struct{
    display_timing_t  htiming
```

```
    display_timing_t  vtiming
    display_out_format_t  format
    display_endian_t  endian
    display_color_order_t  color_order
    display_signal_polarity_t  data_enable_polarity
    display_sync_edge_t  sync_edge
    display_color_t  bg_color
    display_brightness_t  brightness
    display_contrast_t  contrast
    display_gamma_correction_t *  p_gamma_correction
    bool  dithering_on
} display_output_cfg_t
```

11.1.108.1 htiming

`display_timing_t::htiming`

Brief description

Horizontal display cycle setting.

11.1.108.2 vtiming

`display_timing_t::vtiming`

Brief description

Vertical display cycle setting.

11.1.108.3 format

`display_out_format_t::format`

Brief description

Output format setting.

11.1.108.4 endian

`display_endian_t::endian`

Brief description

Bit order of output data.

11.1.108.5 color_order

`display_color_order_t::color_order`

Brief description

Color order in pixel.

11.1.108.6 data_enable_polarity

`display_signal_polarity_t::data_enable_polarity`

Brief description

Data Enable signal polarity.

11.1.108.7 sync_edge

`display_sync_edge_t::sync_edge`

Brief description

Signal sync edge selection.

11.1.108.8 bg_color

`display_color_t::bg_color`

Brief description

Background color.

11.1.108.9 brightness

`display_brightness_t::brightness`

Brief description

Brightness setting.

11.1.108.10 contrast

`display_contrast_t::contrast`

Brief description

Contrast setting.

11.1.108.11 p_gamma_correction

`display_gamma_correction_t::p_gamma_correction`

Brief description

Pointer to gamma correction setting.

11.1.108.12 dithering_on

```
bool display_output_cfg_t::dithering_on
```

Brief description

Dithering on/off.

11.1.109 display_runtime_cfg_t

```
typedef struct{
    display_input_cfg_t  input
    display_layer_t  layer
} display_runtime_cfg_t
```

11.1.109.1 input

```
display_input_cfg_t::input
```

Brief description

Graphics input frame setting.

Detailed description

Generic configuration for display devices

11.1.109.2 layer

```
display_layer_t::layer
```

Brief description

Graphics layer alpha blending setting.

11.1.110 display_status_t

```
typedef struct{
    display_state_t  state
    display_fade_status_t  fade_status[DISPLAY_FRAME_LAYER_2+1]
} display_status_t
```

11.1.110.1 state

```
display_state_t::state
```

Brief description

Status of GLCD module.

11.1.110.2 fade_status

display_fade_status_t::fade_status

Brief description

Status of fade-in/fade-out status.

11.1.111 display_timing_t

```
typedef struct{
    uint16_t  total_cyc
    uint16_t  display_cyc
    uint16_t  back_porch
    uint16_t  sync_width
    display_signal_polarity_t  sync_polarity
} display_timing_t
```

11.1.111.1 total_cyc

uint16_t display_timing_t::total_cyc

Brief description

Total cycles in one line or total lines in one frame.

11.1.111.2 display_cyc

uint16_t display_timing_t::display_cyc

Brief description

Active video cycles or lines.

11.1.111.3 back_porch

uint16_t display_timing_t::back_porch

Brief description

Back poach cycles or lines.

11.1.111.4 sync_width

uint16_t display_timing_t::sync_width

Brief description

Sync signal asserting width.

11.1.111.5 sync_polarity

`display_signal_polarity_t::sync_polarity`

Brief description

Sync signal polarity.

11.1.112 dmac_instance_ctrl_t

```
typedef struct{
    uint32_t  id
    elc_event_t  trigger
    IRQn_Type  irq
    uint8_t  channel
    void(*  p_callback)(transfer_callback_args_t *cb_data)
    void const  *  p_context
    void *  p_reg
} dmac_instance_ctrl_t
```

11.1.112.1 id

`uint32_t ::id`

Brief description

Driver ID.

11.1.112.2 trigger

`elc_event_t::trigger`

Brief description

Transfer activation event. Matches event returned by infoGet.

11.1.112.3 irq

`IRQn_Type ::irq`

Brief description

Transfer activation IRQ.

11.1.112.4 channel

`uint8_t ::channel`

Brief description

Channel number.

11.1.112.5 p_callback

```
void(* ::p_callback) ( *cb_data)
```

Detailed description

Callback for transfer end interrupt.

11.1.112.6 p_context

```
void const* ::p_context
```

Detailed description

Placeholder for user data. Passed to the user p_callback in transfer_callback_args_t.

11.1.112.7 p_reg

```
void* ::p_reg
```

Detailed description

Pointer to base register.

11.1.113 doc_api_t

```
typedef struct{
    ssp_err_t(*  open)(doc_ctrl_t *const p_ctrl, doc_cfg_t const *const p_cfg)
    ssp_err_t(*  close)(doc_ctrl_t *const p_ctrl)
    ssp_err_t(*  statusGet)(doc_ctrl_t *const p_ctrl, doc_status_t *p_status)
    ssp_err_t(*  statusClear)(doc_ctrl_t *const p_ctrl)
    ssp_err_t(*  write)(doc_ctrl_t *const p_ctrl, doc_data_t *const p_data)
    ssp_err_t(*  inputRegisterWrite)(doc_ctrl_t *const p_ctrl, doc_size_t data)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} doc_api_t
```

11.1.114 doc_callback_args_t

```
typedef struct{
    doc_event_t  event
    void const  *  p_context
} doc_callback_args_t
```

11.1.114.1 event

doc_event_t::event

Brief description

The event is used to identify what caused the callback.

11.1.114.2 p_context

void const* doc_callback_args_t::p_context

Detailed description

Placeholder for user data. Set in open function in doc_cfg_t.

11.1.115 doc_cfg_t

```
typedef struct{
    doc_event_t  event
    uint8_t  irq_ipl
    void(*  p_callback)(doc_callback_args_t *p_args)
    void const  *  p_context
} doc_cfg_t
```

11.1.115.1 event

doc_event_t::event

Brief description

Select enumerated value from doc_event_t.

11.1.115.2 irq_ipl

uint8_t doc_cfg_t::irq_ipl

Brief description

DOC interrupt priority.

11.1.115.3 p_callback

void(* doc_cfg_t::p_callback) (doc_callback_args_t *p_args)

Detailed description

Callback provided when a DOC ISR occurs.

11.1.115.4 p_context

```
void const* doc_cfg_t::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in doc_callback_args_t.

11.1.116 doc_data_t

```
typedef struct{
    doc_size_t  dodir
    doc_size_t  dodsr
} doc_data_t
```

11.1.116.1 dodir

```
doc_size_t::dodir
```

Brief description

Value to be written to the DOC DODIR.

11.1.116.2 dodsr

```
doc_size_t::dodsr
```

Brief description

Value to be written to the DOC DODSR.

11.1.117 doc_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    void(*  p_callback)(doc_callback_args_t *p_args)
    doc_event_t  event
    void const  *  p_context
    void *  p_reg
} doc_instance_ctrl_t
```

11.1.117.1 open

```
uint32_t ::open
```

Brief description

Used by driver to check if the control structure is valid.

11.1.117.2 p_callback

```
void(* ::p_callback) ( *p_args)
```

Detailed description

Callback provided when a DOC ISR occurs. NULL indicates no CPU interrupt.

11.1.117.3 event

```
doc_event_t::event
```

Brief description

The event DOC is configured for. Passed in ISR callback.

11.1.117.4 p_context

```
void const* ::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in doc_callback_args_t.

11.1.117.5 p_reg

```
void* ::p_reg
```

Brief description

Base register.

11.1.118 doc_instance_t

```
typedef struct{
    doc_ctrl_t *  p_ctrl
    doc_cfg_t const  *  p_cfg
    doc_api_t const  *  p_api
} doc_instance_t
```

11.1.118.1 p_ctrl

```
doc_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.118.2 p_cfg

doc_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.118.3 p_api

doc_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.119 dsa_domain_1024_160_t

```
typedef struct{
    uint32_t  q[(160/32)]
    uint32_t  p[(1024/32)]
    uint32_t  g[(1024/32)]
} dsa_domain_1024_160_t
```

11.1.119.1 q

uint32_t ::q[(160/32)]

Brief description

DSA (1024,160) domain parameter Q.

11.1.119.2 p

uint32_t ::p[(1024/32)]

Brief description

DSA (1024,160) domain parameter P.

11.1.119.3 g

uint32_t ::g[(1024/32)]

Brief description

DSA (1024,160) domain parameter G.

11.1.120 dsa_domain_2048_224_t

```
typedef struct{
    uint32_t  q[(224/32)]
    uint32_t  p[(2048/32)]
    uint32_t  g[(2048/32)]
} dsa_domain_2048_224_t
```

11.1.120.1 q

```
uint32_t ::q[(224/32)]
```

Brief description

DSA (2048,224) domain parameter Q.

11.1.120.2 p

```
uint32_t ::p[(2048/32)]
```

Brief description

DSA (2048,224) domain parameter P.

11.1.120.3 g

```
uint32_t ::g[(2048/32)]
```

Brief description

DSA (2048,224) domain parameter G.

11.1.121 dsa_domain_2048_256_t

```
typedef struct{
    uint32_t  q[(256/32)]
    uint32_t  p[(2048/32)]
    uint32_t  g[(2048/32)]
} dsa_domain_2048_256_t
```

11.1.121.1 q

```
uint32_t ::q[(256/32)]
```

Brief description

DSA (2048,256) domain parameter Q.

11.1.121.2 p

```
uint32_t ::p[(2048/32)]
```

Brief description

DSA (2048,256) domain parameter P.

11.1.121.3 g

```
uint32_t ::g[(2048/32)]
```

Brief description

DSA (2048,256) domain parameter G.

11.1.122 dsa_signature_1024_160_t

```
typedef struct{
    uint32_t  r[(160/32)]
    uint32_t  s[(160/32)]
} dsa_signature_1024_160_t
```

11.1.122.1 r

```
uint32_t ::r[(160/32)]
```

Brief description

DSA (1024,160) signature component R.

11.1.122.2 s

```
uint32_t ::s[(160/32)]
```

Brief description

DSA (1024,160) signature component S.

11.1.123 dsa_signature_2048_224_t

```
typedef struct{
    uint32_t  r[(224/32)]
    uint32_t  s[(224/32)]
} dsa_signature_2048_224_t
```

11.1.123.1 r

```
uint32_t ::r[(224/32)]
```

Brief description

DSA (2048,224) signature component R.

11.1.123.2 s

```
uint32_t ::s[(224/32)]
```

Brief description

DSA (2048,224) signature component S.

11.1.124 dsa_signature_2048_256_t

```
typedef struct{
    uint32_t  r[(256/32)]
    uint32_t  s[(256/32)]
} dsa_signature_2048_256_t
```

11.1.124.1 r

```
uint32_t ::r[(256/32)]
```

Brief description

DSA (2048,256) signature component R.

11.1.124.2 s

```
uint32_t ::s[(256/32)]
```

Brief description

DSA (2048,256) signature component S.

11.1.125 dtc_instance_ctrl_t

```
typedef struct{
    uint32_t  id
    elc_event_t  trigger
    IRQn_Type  irq
    void(*  p_callback)(transfer_callback_args_t *cb_data)
    void const  *  p_context
} dtc_instance_ctrl_t
```

11.1.125.1 id

```
uint32_t ::id
```

Brief description

Driver ID.

11.1.125.2 trigger

```
elc_event_t::trigger
```

Brief description

Transfer activation event. Matches event returned by infoGet.

11.1.125.3 irq

```
IRQn_Type ::irq
```

Brief description

Transfer activation IRQ, does not apply to all HAL drivers.

11.1.125.4 p_callback

```
void(* ::p_callback) ( *cb_data)
```

Detailed description

Callback for transfer end interrupt used for ELC software trigger.

11.1.125.5 p_context

```
void const* ::p_context
```

Detailed description

Placeholder for user data. Passed to the user p_callback in transfer_callback_args_t.

11.1.126 dtc_reg_t

```
typedef struct{
    uint32_t  __pad0__
    uint8_t  MRB
    uint8_t  __pad0__
    uint8_t  DM
    uint8_t  DTS
    uint8_t  DISEL
    uint8_t  CHNS
```

```
    uint8_t  CHNE
    struct{}                MRB_b
    uint8_t  MRA
    uint8_t  SM
    uint8_t  SZ
    uint8_t  MD
    struct{}                MRA_b
    struct{}                struct{}
    void *volatile  SAR
    void *volatile  DAR
    uint16_t  CRB
    uint16_t  CRA
    uint8_t  CRAL
    uint8_t  CRAH
    struct{}                CRA_b
    struct{}                struct{}
} dtc_reg_t
```

11.1.126.1 __pad0__

```
uint32_t ::__pad0__
```

11.1.126.2 MRB

```
uint8_t ::MRB
```

Detailed description

Mode Register B

11.1.126.3 __pad0__

```
uint8_t ::__pad0__
```

11.1.126.4 DM

```
uint8_t ::DM
```

Brief description

Transfer Destination Address mode.

11.1.126.5 DTS

```
uint8_t ::DTS
```

Brief description

DTC Transfer Mode Select.

11.1.126.6 DISEL

```
uint8_t ::DISEL
```

Brief description

DTC Interrupt Select.

11.1.126.7 CHNS

```
uint8_t ::CHNS
```

Brief description

DTC Chain Transfer Select.

11.1.126.8 CHNE

```
uint8_t ::CHNE
```

Brief description

DTC CHain Transfer Enable.

11.1.126.9 MRB_b

```
See source code for the definition of this member.
```

Detailed description

- MRB bits */

11.1.126.10 MRA

```
uint8_t ::MRA
```

Detailed description

Mode Register A

11.1.126.11 SM

```
uint8_t ::SM
```

Brief description

Transfer Source Address mode.

11.1.126.12 SZ

```
uint8_t ::SZ
```

Brief description

DTC Data Transfer Size.

11.1.126.13 MD

```
uint8_t ::MD
```

Brief description

DTC Transfer Mode Select.

11.1.126.14 MRA_b

```
See source code for the definition of this member.
```

Detailed description

- MRA bits */

11.1.126.15 struct{}

```
See source code for the definition of this member.
```

Detailed description

- Mode registers */

11.1.126.16 SAR

```
void* volatile ::SAR
```

Brief description

Source address register.

11.1.126.17 DAR

```
void* volatile ::DAR
```

Detailed description

Destination address register

11.1.126.18 CRB

```
volatile uint16_t ::CRB
```

Detailed description

Transfer count register B

11.1.126.19 CRA

```
uint16_t ::CRA
```

Detailed description

Transfer count register A

11.1.126.20 CRAL

```
uint8_t ::CRAL
```

Brief description

Transfer counter A lower register.

11.1.126.21 CRAH

```
uint8_t ::CRAH
```

Brief description

Transfer counter B upper register.

11.1.126.22 CRA_b

```
See source code for the definition of this member.
```

Detailed description

- bits */

11.1.126.23 struct{}

```
See source code for the definition of this member.
```

Detailed description

- Transfer count registers */

11.1.127 elc_api_t

```
typedef struct{
    ssp_err_t(*  init)(elc_cfg_t const *const p_cfg)
    ssp_err_t(*  softwareEventGenerate)(elc_software_event_t event_num)
    ssp_err_t(*  linkSet)(elc_peripheral_t peripheral, elc_event_t signal)
    ssp_err_t(*  linkBreak)(elc_peripheral_t peripheral)
    ssp_err_t(*  enable)(void)
    ssp_err_t(*  disable)(void)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} elc_api_t
```

11.1.128 elc_cfg_t

```
typedef struct{
    bool  autostart
    uint32_t  link_count
    elc_link_t const  *  link_list
} elc_cfg_t
```

11.1.128.1 autostart

```
bool elc_cfg_t::autostart
```

Brief description

Start operation and enable interrupts during open().

11.1.128.2 link_count

```
uint32_t elc_cfg_t::link_count
```

Brief description

Number of event links.

11.1.128.3 link_list

```
elc_link_t::link_list
```

Brief description

Event links.

11.1.129 elc_instance_t

```
typedef struct{
    elc_cfg_t const  *  p_cfg
```

```
    elc_api_t const  *  p_api
} elc_instance_t
```

11.1.129.1 p_cfg

elc_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.129.2 p_api

elc_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.130 elc_link_t

```
typedef struct{
    elc_peripheral_t  peripheral
    elc_event_t  event
} elc_link_t
```

11.1.130.1 peripheral

elc_peripheral_t::peripheral

Brief description

Peripheral to receive the signal.

11.1.130.2 event

elc_event_t::event

Brief description

Signal that gets sent to the Peripheral.

11.1.131 external_irq_api_t

```
typedef struct{
    ssp_err_t(*  open)(external_irq_ctrl_t *const p_ctrl, external_irq_cfg_t
const *const p_cfg)
    ssp_err_t(*  enable)(external_irq_ctrl_t *const p_ctrl)
```

```
    ssp_err_t(*  disable)(external_irq_ctrl_t *const p_ctrl)
    ssp_err_t(*  triggerSet)(external_irq_ctrl_t *const p_ctrl,
external_irq_trigger_t const trigger)
    ssp_err_t(*  filterEnable)(external_irq_ctrl_t *const p_ctrl)
    ssp_err_t(*  filterDisable)(external_irq_ctrl_t *const p_ctrl)
    ssp_err_t(*  close)(external_irq_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} external_irq_api_t
```

11.1.132 external_irq_callback_args_t

```
typedef struct{
    void const  *  p_context
    uint32_t  channel
} external_irq_callback_args_t
```

11.1.132.1 p_context

```
void const* external_irq_callback_args_t::p_context
```

Detailed description

Placeholder for user data. Set in open function in external_irq_cfg_t.

11.1.132.2 channel

```
uint32_t external_irq_callback_args_t::channel
```

Brief description

The physical hardware channel that caused the interrupt.

11.1.133 external_irq_cfg_t

```
typedef struct{
    uint8_t  channel
    uint8_t  irq_ipl
    external_irq_trigger_t  trigger
    external_irq_pclk_div_t  pclk_div
    bool  autostart
    bool  filter_enable
    void(*  p_callback)(external_irq_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} external_irq_cfg_t
```

11.1.133.1 channel

```
uint8_t external_irq_cfg_t::channel
```

Brief description

Hardware channel used.

11.1.133.2 irq_ipl

```
uint8_t external_irq_cfg_t::irq_ipl
```

Brief description

Interrupt priority.

11.1.133.3 trigger

```
external_irq_trigger_t::trigger
```

Brief description

Trigger setting.

11.1.133.4 pclk_div

```
external_irq_pclk_div_t::pclk_div
```

Brief description

Digital filter clock divisor setting.

11.1.133.5 autostart

```
bool external_irq_cfg_t::autostart
```

Brief description

Start operation and enable interrupts during open().

11.1.133.6 filter_enable

```
bool external_irq_cfg_t::filter_enable
```

Brief description

Digital filter enable/disable setting.

11.1.133.7 p_callback

```
void(* external_irq_cfg_t::p_callback) (external_irq_callback_args_t *p_args)
```

Detailed description

Callback provided external input trigger occurs.

11.1.133.8 p_context

```
void const* external_irq_cfg_t::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in external_irq_callback_args_t.

11.1.133.9 p_extend

```
void const* external_irq_cfg_t::p_extend
```

Brief description

External IRQ hardware dependent configuration.

11.1.134 external_irq_instance_t

```
typedef struct{
    external_irq_ctrl_t *  p_ctrl
    external_irq_cfg_t const  *  p_cfg
    external_irq_api_t const  *  p_api
} external_irq_instance_t
```

11.1.134.1 p_ctrl

```
external_irq_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.134.2 p_cfg

```
external_irq_cfg_t::p_cfg
```

Brief description

Pointer to the configuration structure for this instance.

11.1.134.3 p_api

```
external_irq_api_t::p_api
```

Brief description

Pointer to the API structure for this instance.

11.1.135 flash_api_t

```
typedef struct{
    ssp_err_t(*  open)(flash_ctrl_t *const p_ctrl, flash_cfg_t const *const
p_cfg)
    ssp_err_t(*  write)(flash_ctrl_t *const p_ctrl, uint32_t const src_address,
uint32_t const flash_address, uint32_t const num_bytes)
    ssp_err_t(*  read)(flash_ctrl_t *const p_ctrl, uint8_t *const p_dest_address,
uint32_t const flash_address, uint32_t const num_bytes)
    ssp_err_t(*  erase)(flash_ctrl_t *const p_ctrl, uint32_t const address,
uint32_t const num_blocks)
    ssp_err_t(*  blankCheck)(flash_ctrl_t *const p_ctrl, uint32_t const address,
uint32_t const num_bytes, flash_result_t *const p_blank_check_result)
    ssp_err_t(*  infoGet)(flash_ctrl_t *const p_ctrl, flash_info_t *const p_info)
    ssp_err_t(*  close)(flash_ctrl_t *const p_ctrl)
    ssp_err_t(*  statusGet)(flash_ctrl_t *const p_ctrl)
    ssp_err_t(*  accessWindowSet)(flash_ctrl_t *const p_ctrl, uint32_t const
start_addr, uint32_t const end_addr)
    ssp_err_t(*  accessWindowClear)(flash_ctrl_t *const p_ctrl)
    ssp_err_t(*  reset)(flash_ctrl_t *const p_ctrl)
    ssp_err_t(*  updateFlashClockFreq)(flash_ctrl_t *const p_ctrl)
    ssp_err_t(*  startupAreaSelect)(flash_ctrl_t *const p_ctrl,
flash_startup_area_swap_t swap_type, bool is_temporary)
    ssp_err_t(*  versionGet)(ssp_version_t *p_version)
} flash_api_t
```

11.1.136 flash_callback_args_t

```
typedef struct{
    flash_event_t  event
    void const  *  p_context
} flash_callback_args_t
```

11.1.136.1 event

flash_event_t::event

Brief description

Event can be used to identify what caused the callback (flash ready or error).

11.1.136.2 p_context

```
void const* flash_callback_args_t::p_context
```

Brief description

Placeholder for user data. Set in open function in::flash_cfg_t.

11.1.137 flash_cfg_t

```
typedef struct{
    bool  data_flash_bgo
    void(*  p_callback)(flash_callback_args_t *p_args)
    void const  *  p_extend
    void const  *  p_context
    uint8_t  irq_ipl
    uint8_t  err_irq_ipl
} flash_cfg_t
```

11.1.137.1 data_flash_bgo

```
bool flash_cfg_t::data_flash_bgo
```

Brief description

True if BGO (Background Operation) is enabled for Data Flash.

11.1.137.2 p_callback

```
void(* flash_cfg_t::p_callback) (flash_callback_args_t *p_args)
```

Brief description

Callback provided when a Flash interrupt ISR occurs.

11.1.137.3 p_extend

```
void const* flash_cfg_t::p_extend
```

Brief description

FLASH hardware dependent configuration.

11.1.137.4 p_context

```
void const* flash_cfg_t::p_context
```

Brief description

Placeholder for user data. Passed to user callback in flash_callback_args_t.

11.1.137.5 irq_ipl

```
uint8_t flash_cfg_t::irq_ipl
```

Brief description

Flash ready interrupt priority.

11.1.137.6 err_irq_ipl

```
uint8_t flash_cfg_t::err_irq_ipl
```

Brief description

Flash error interrupt priority (unused in r_flash_lp)

11.1.138 flash_fmi_block_info_t

```
typedef struct{
    uint32_t  block_section_st_addr
    uint32_t  block_section_end_addr
    uint32_t  block_size
    uint32_t  block_size_write
} flash_fmi_block_info_t
```

11.1.138.1 block_section_st_addr

```
uint32_t flash_fmi_block_info_t::block_section_st_addr
```

Brief description

starting address for this block section (blocks of this size)

11.1.138.2 block_section_end_addr

```
uint32_t flash_fmi_block_info_t::block_section_end_addr
```

Brief description

ending address for this block section (blocks of this size)

11.1.138.3 block_size

```
uint32_t flash_fmi_block_info_t::block_size
```

Brief description

Flash erase block size.

11.1.138.4 block_size_write

```
uint32_t flash_fmi_block_info_t::block_size_write
```

Brief description

Flash write block size.

11.1.139 flash_fmi_regions_t

```
typedef struct{
    uint32_t  num_regions
    flash_fmi_block_info_t const  *  p_block_array
} flash_fmi_regions_t
```

11.1.139.1 num_regions

```
uint32_t flash_fmi_regions_t::num_regions
```

Brief description

Length of block info array.

11.1.139.2 p_block_array

```
flash_fmi_block_info_t::p_block_array
```

Brief description

Block info array base address.

11.1.140 flash_hp_instance_ctrl_t

```
typedef struct{
    uint32_t  opened
    R_FACI_Type *  p_reg
    void(*  p_callback)(flash_callback_args_t *p_args)
    bsp_cache_state_t  cache_state
    IRQn_Type  irq
    IRQn_Type  err_irq
} flash_hp_instance_ctrl_t
```

11.1.140.1 opened

```
uint32_t ::opened
```

Brief description

To check whether api has been opened or not.

11.1.140.2 p_reg

```
R_FACI_Type* ::p_reg
```

Brief description

Base address of flash registers.

11.1.140.3 p_callback

```
void(* ::p_callback) ( *p_args)
```

11.1.140.4 cache_state

```
bsp_cache_state_t::cache_state
```

Brief description

User Callback function.

Detailed description

Used to disable and then restore Flash Cache while API is open.

11.1.140.5 irq

```
IRQn_Type ::irq
```

Brief description

Flash ready interrupt number.

11.1.140.6 err_irq

```
IRQn_Type ::err_irq
```

Brief description

Flash error interrupt number.

11.1.141 flash_info_t

```
typedef struct{
    flash_fmi_regions_t  code_flash
    flash_fmi_regions_t  data_flash
} flash_info_t
```

11.1.141.1 code_flash

```
flash_fmi_regions_t::code_flash
```

Brief description

Information about the code flash regions.

11.1.141.2 data_flash

flash_fmi_regions_t::data_flash

Brief description

Information about the code flash regions.

11.1.142 flash_instance_t

```
typedef struct{
    flash_ctrl_t *  p_ctrl
    flash_cfg_t const  *  p_cfg
    flash_api_t const  *  p_api
} flash_instance_t
```

11.1.142.1 p_ctrl

flash_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.142.2 p_cfg

flash_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.142.3 p_api

flash_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.143 flash_lp_instance_ctrl_t

```
typedef struct{
    uint32_t  opened
    void *  p_reg
```

```
    void(*  p_callback)(flash_callback_args_t *p_args)
    bsp_cache_state_t  cache_state
    IRQn_Type  irq
} flash_lp_instance_ctrl_t
```

11.1.143.1 opened

```
uint32_t ::opened
```

Brief description

To check whether api has been opened or not.

11.1.143.2 p_reg

```
void* ::p_reg
```

Brief description

Base address of flash registers.

11.1.143.3 p_callback

```
void(* ::p_callback) ( *p_args)
```

11.1.143.4 cache_state

```
bsp_cache_state_t::cache_state
```

Brief description

Used to disable and then restore Flash Cache while API is open.

11.1.143.5 irq

```
IRQn_Type ::irq
```

Brief description

Flash ready interrupt number.

11.1.144 fmi_api_t

```
typedef struct{
    ssp_err_t(*  init)(void)
    ssp_err_t(*  productInfoGet)(fmi_product_info_t **pp_product_info)
    ssp_err_t(*  uniqueIdGet)(fmi_unique_id_t *p_unique_id)
    ssp_err_t(*  productFeatureGet)(ssp_feature_t const *const p_feature,
fmi_feature_info_t *const p_info)
```

```
    ssp_err_t(*  eventInfoGet)(ssp_feature_t const *const p_feature, ssp_signal_t
signal, fmi_event_info_t *const p_info)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} fmi_api_t
```

11.1.145 fmi_event_info_t

```
typedef struct{
    IRQn_Type  irq
    elc_event_t  event
} fmi_event_info_t
```

11.1.145.1 irq

```
IRQn_Type fmi_event_info_t::irq
```

11.1.145.2 event

```
elc_event_t::event
```

11.1.146 fmi_feature_info_t

```
typedef struct{
    void *  ptr
    uint32_t  channel_count
    uint32_t  variant_data
    uint32_t  extended_data_count
    uint32_t  version_major
    uint32_t  version_minor
    struct{}                struct{}
    void *  ptr_extended_data
} fmi_feature_info_t
```

11.1.146.1 ptr

```
void* fmi_feature_info_t::ptr
```

11.1.146.2 channel_count

```
uint32_t fmi_feature_info_t::channel_count
```

11.1.146.3 variant_data

```
uint32_t fmi_feature_info_t::variant_data
```

11.1.146.4 extended_data_count

uint32_t fmi_feature_info_t::extended_data_count

11.1.146.5 version_major

uint32_t fmi_feature_info_t::version_major

11.1.146.6 version_minor

uint32_t fmi_feature_info_t::version_minor

11.1.146.7 struct{}

See source code for the definition of this member.

11.1.146.8 ptr_extended_data

void* fmi_feature_info_t::ptr_extended_data

11.1.147 fmi_header_t

```
typedef struct{
    uint32_t  contents
    uint32_t  variant
    uint32_t  count
    uint32_t  minor
    uint32_t  major
} fmi_header_t
```

11.1.147.1 contents

uint32_t fmi_header_t::contents

11.1.147.2 variant

uint32_t fmi_header_t::variant

11.1.147.3 count

uint32_t fmi_header_t::count

11.1.147.4 minor

uint32_t fmi_header_t::minor

11.1.147.5 major

```
uint32_t fmi_header_t::major
```

11.1.148 fmi_instance_t

```
typedef struct{
    fmi_api_t const  *  p_api
} fmi_instance_t
```

11.1.148.1 p_api

```
fmi_api_t::p_api
```

Brief description

Pointer to the API structure for this instance.

11.1.149 fmi_product_info_t

```
typedef struct{
    fmi_header_t  header
    uint8_t  unique_id[16]
    uint8_t  product_name[16]
    uint8_t  product_marking[16]
    uint32_t  mask_revision
    uint32_t  pin_count
    uint32_t  pkg_type
    uint32_t  temp_range
    uint32_t  quality_code
    uint32_t  reserved
    struct{}                struct{}
    uint32_t  max_freq
    uint32_t  reserved1
    struct{}                struct{}
} fmi_product_info_t
```

11.1.149.1 header

```
fmi_header_t fmi_product_info_t::header
```

11.1.149.2 unique_id

```
uint8_t fmi_product_info_t::unique_id[16]
```

Brief description

DEPRECATED, use uniqueIdGet instead.

11.1.149.3 product_name

```
uint8_t fmi_product_info_t::product_name[16]
```

11.1.149.4 product_marking

```
uint8_t fmi_product_info_t::product_marking[16]
```

11.1.149.5 mask_revision

```
uint32_t fmi_product_info_t::mask_revision
```

11.1.149.6 pin_count

```
uint32_t fmi_product_info_t::pin_count
```

11.1.149.7 pkg_type

```
uint32_t fmi_product_info_t::pkg_type
```

11.1.149.8 temp_range

```
uint32_t fmi_product_info_t::temp_range
```

11.1.149.9 quality_code

```
uint32_t fmi_product_info_t::quality_code
```

11.1.149.10 reserved

```
uint32_t fmi_product_info_t::reserved
```

11.1.149.11 struct{}

```
See source code for the definition of this member.
```

11.1.149.12 max_freq

```
uint32_t fmi_product_info_t::max_freq
```

11.1.149.13 reserved1

```
uint32_t fmi_product_info_t::reserved1
```

11.1.149.14 struct{}

```
See source code for the definition of this member.
```

11.1.150 fmi_unique_id_t

```
typedef struct{
    uint32_t  unique_id[4]
} fmi_unique_id_t
```

11.1.150.1 unique_id

```
uint32_t fmi_unique_id_t::unique_id[4]
```

11.1.151 gamma_correction_t

```
typedef struct{
    bool  enable
    uint16_t  gain[DISPLAY_GAMMA_CURVE_ELEMENT_NUM]
    uint16_t  threshold[DISPLAY_GAMMA_CURVE_ELEMENT_NUM]
} gamma_correction_t
```

11.1.151.1 enable

```
bool gamma_correction_t::enable
```

Brief description

Gamma Correction On/Off.

11.1.151.2 gain

```
uint16_t gamma_correction_t::gain[DISPLAY_GAMMA_CURVE_ELEMENT_NUM]
```

Brief description

Gain adjustment.

11.1.151.3 threshold

```
uint16_t gamma_correction_t::threshold[DISPLAY_GAMMA_CURVE_ELEMENT_NUM]
```

Brief description

Start threshold.

11.1.152 glcd_cfg_t

```
typedef struct{
    glcd_tcon_pin_t  tcon_hsync
    glcd_tcon_pin_t  tcon_vsync
    glcd_tcon_pin_t  tcon_de
    glcd_correction_proc_order_t  correction_proc_order
    glcd_clk_src_t  clksrc
    glcd_panel_clk_div_t  clock_div_ratio
    glcd_dithering_mode_t  dithering_mode
    glcd_dithering_pattern_t  dithering_pattern_A
    glcd_dithering_pattern_t  dithering_pattern_B
    glcd_dithering_pattern_t  dithering_pattern_C
    glcd_dithering_pattern_t  dithering_pattern_D
} glcd_cfg_t
```

11.1.152.1 tcon_hsync

glcd_tcon_pin_t::tcon_hsync

Brief description

GLCD TCON output pin select.

11.1.152.2 tcon_vsync

glcd_tcon_pin_t::tcon_vsync

Brief description

GLCD TCON output pin select.

11.1.152.3 tcon_de

glcd_tcon_pin_t::tcon_de

Brief description

GLCD TCON output pin select.

11.1.152.4 correction_proc_order

glcd_correction_proc_order_t::correction_proc_order

Brief description

Correction control route select.

11.1.152.5 clksrc

glcd_clk_src_t::clksrc

Brief description

Clock Source selection.

11.1.152.6 clock_div_ratio

glcd_panel_clk_div_t::clock_div_ratio

Brief description

Clock divide ratio for dot clock.

11.1.152.7 dithering_mode

glcd_dithering_mode_t::dithering_mode

Brief description

Dithering mode.

11.1.152.8 dithering_pattern_A

glcd_dithering_pattern_t::dithering_pattern_A

Brief description

Dithering pattern A.

11.1.152.9 dithering_pattern_B

glcd_dithering_pattern_t::dithering_pattern_B

Brief description

Dithering pattern B.

11.1.152.10 dithering_pattern_C

glcd_dithering_pattern_t::dithering_pattern_C

Brief description

Dithering pattern C.

11.1.152.11 dithering_pattern_D

glcd_dithering_pattern_t::dithering_pattern_D

Brief description

Dithering pattern D.

11.1.153 glcd_ctrl_t

```
typedef struct{
    display_coordinate_t  back_porch
    uint16_t  hsize
    uint16_t  vsize
    bsp_lock_t  resource_lock
    void *  p_context
} glcd_ctrl_t
```

11.1.153.1 back_porch

display_coordinate_t::back_porch

Brief description

Zero coordinate for graphics plane(Bach porch End)

11.1.153.2 hsize

uint16_t ::hsize

Brief description

Horizontal pixel size in a line.

11.1.153.3 vsize

uint16_t ::vsize

Brief description

Vertical pixel size in a frame.

11.1.153.4 resource_lock

bsp_lock_t::resource_lock

Brief description

Resource lock.

11.1.153.5 p_context

```
void* ::p_context
```

Detailed description

Pointer to the function level device context (e.g. display_ctrl_t type data)

11.1.154 glcd_instance_ctrl_t

```
typedef struct{
    display_state_t  state
    void(*  p_callback)(display_callback_args_t *p_args)
    void const  *  p_context
    R_GLCDC_Type *  p_reg
} glcd_instance_ctrl_t
```

11.1.154.1 state

```
display_state_t::state
```

Brief description

Status of GLCD module.

11.1.154.2 p_callback

```
void(* ::p_callback) ( *p_args)
```

Brief description

Pointer to callback function.

11.1.154.3 p_context

```
void const* ::p_context
```

Brief description

Pointer to the higher level device context.

11.1.154.4 p_reg

```
R_GLCDC_Type* ::p_reg
```

Brief description

Base register address.

11.1.155 gpt_input_capture_extend_t

```
typedef struct{
    gpt_input_capture_signal_t  signal
    gpt_input_capture_signal_filter_t  signal_filter
    gpt_input_capture_clock_divider_t  clock_divider
    input_capture_signal_level_t  enable_level
    gpt_input_capture_signal_filter_t  enable_filter
} gpt_input_capture_extend_t
```

11.1.155.1 signal

gpt_input_capture_signal_t::signal

Brief description

One of gpt_input_capture_signal_t.

11.1.155.2 signal_filter

gpt_input_capture_signal_filter_t::signal_filter

Brief description

One of gpt_input_capture_signal_filter_t.

11.1.155.3 clock_divider

gpt_input_capture_clock_divider_t::clock_divider

Brief description

One of gpt_input_capture_clock_divider_t.

11.1.155.4 enable_level

input_capture_signal_level_t::enable_level

Detailed description

The unused GTIOCx pin can be used as an enable signal to enable captures. If the Input Capture Signal Pin is GTIOCA, then the enable pin is GTIOCB. The enable level is set here if used.

11.1.155.5 enable_filter

gpt_input_capture_signal_filter_t::enable_filter

Brief description

One of gpt_input_capture_signal_filter_t.

11.1.156 gpt_input_capture_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    uint8_t  channel
    input_capture_mode_t  mode
    input_capture_repetition_t  repetition
    uint32_t  capture_count
    uint32_t  overflows_last
    uint32_t  overflows_current
    void(*  p_callback)(input_capture_callback_args_t *p_args)
    void const  *  p_context
    void *  p_reg
    IRQn_Type  capture_irq
    IRQn_Type  overflow_irq
    input_capture_variant_t  variant
    uint32_t  start_bitmask
    uint32_t  stop_bitmask
} gpt_input_capture_instance_ctrl_t
```

11.1.156.1 open

```
uint32_t ::open
```

Brief description

Whether or not channel is open.

11.1.156.2 channel

```
uint8_t ::channel
```

Brief description

The channel in use.

11.1.156.3 mode

```
input_capture_mode_t::mode
```

Brief description

The mode of measurement being performed.

11.1.156.4 repetition

```
input_capture_repetition_t::repetition
```

Brief description

One-shot or periodic measurement.

11.1.156.5 capture_count

```
uint32_t ::capture_count
```

Brief description

The value of the timer captured at the time of interrupt.

11.1.156.6 overflows_last

```
uint32_t ::overflows_last
```

Brief description

Overflow count that occurred during last measurement.

11.1.156.7 overflows_current

```
uint32_t ::overflows_current
```

Brief description

Running count of overflows in current measurement.

11.1.156.8 p_callback

```
void(* ::p_callback) ( *p_args)
```

Brief description

Pointer to user callback.

11.1.156.9 p_context

```
void const* ::p_context
```

Brief description

Pointer to user's context data, to be passed to the callback function.

11.1.156.10 p_reg

```
void* ::p_reg
```

Brief description

GPT base register for this channel.

11.1.156.11 capture_irq

```
IRQn_Type ::capture_irq
```

Brief description

Capture IRQ number.

11.1.156.12 overflow_irq

```
IRQn_Type ::overflow_irq
```

Brief description

Overflow IRQ number.

11.1.156.13 variant

```
input_capture_variant_t::variant
```

Brief description

Timer variant.

11.1.156.14 start_bitmask

```
uint32_t ::start_bitmask
```

Brief description

Start and Clear bitmask for input capture.

11.1.156.15 stop_bitmask

```
uint32_t ::stop_bitmask
```

Brief description

Stop and capture bitmask for input capture.

11.1.157 gpt_instance_ctrl_t

```
typedef struct{
    void(*  p_callback)(timer_callback_args_t *p_args)
    void const  *  p_context
    void *  p_reg
    uint32_t  open
    uint8_t  channel
    bool  one_shot
    IRQn_Type  irq
```

```
    timer_variant_t  variant
} gpt_instance_ctrl_t
```

11.1.157.1 p_callback

```
void(* ::p_callback) ( *p_args)
```

Detailed description

Callback provided when a timer ISR occurs. NULL indicates no CPU interrupt.

11.1.157.2 p_context

```
void const* ::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in timer_callback_args_t.

11.1.157.3 p_reg

```
void* ::p_reg
```

Brief description

Base register for this channel.

11.1.157.4 open

```
uint32_t ::open
```

Brief description

Whether or not channel is open.

11.1.157.5 channel

```
uint8_t ::channel
```

Brief description

Channel number.

11.1.157.6 one_shot

```
bool ::one_shot
```

Brief description

Whether or not timer is in one shot mode.

11.1.157.7 irq

```
IRQn_Type ::irq
```

Brief description

Counter overflow IRQ number.

11.1.157.8 variant

```
timer_variant_t::variant
```

Brief description

Timer variant.

11.1.158 gpt_output_pin_t

```
typedef struct{
    bool  output_enabled
    gpt_pin_level_t  stop_level
} gpt_output_pin_t
```

11.1.158.1 output_enabled

```
bool ::output_enabled
```

Brief description

Set to true to enable output, false to disable output.

11.1.158.2 stop_level

```
gpt_pin_level_t::stop_level
```

Brief description

Select a stop level from gpt_pin_level_t.

11.1.159 hash_api_t

```
typedef struct{
    uint32_t(*  open)(hash_ctrl_t *const p_ctrl, hash_cfg_t const *const p_cfg)
    uint32_t(*  updateHash)(const uint32_t *p_source, uint32_t num_words,
uint32_t *p_dest)
    uint32_t(*  hashUpdate)(hash_ctrl_t *const p_ctrl, const uint32_t *p_source,
uint32_t num_words, uint32_t *p_dest)
    uint32_t(*  close)(hash_ctrl_t *const p_ctrl)
```

```
    uint32_t(*  versionGet)(ssp_version_t *const p_version)
} hash_api_t
```

11.1.160 hash_cfg_t

```
typedef struct{
    crypto_ctrl_t *  p_crypto_ctrl
    crypto_api_t const  *  p_crypto_api
} hash_cfg_t
```

11.1.160.1 p_crypto_ctrl

```
* hash_cfg_t::p_crypto_ctrl
```

Brief description

pointer to crypto engine control structure

11.1.160.2 p_crypto_api

```
const* hash_cfg_t::p_crypto_api
```

Brief description

pointer to crypto engine API structure

11.1.161 hash_ctrl_t

```
typedef struct{
    uint32_t  msgbuf[HASH_MESSAGE_BLOCK_SIZE_WORDS]
    uint32_t  hash[HASH_MAX_DIGEST_SIZE_WORDS]
    uint64_t  length
} hash_ctrl_t
```

11.1.161.1 msgbuf

```
uint32_t hash_ctrl_t::msgbuf[HASH_MESSAGE_BLOCK_SIZE_WORDS]
```

Brief description

message buffer to be hashed

11.1.161.2 hash

```
uint32_t hash_ctrl_t::hash[HASH_MAX_DIGEST_SIZE_WORDS]
```

Brief description

current hash value

11.1.161.3 length

`uint64_t hash_ctrl_t::length`

Brief description

64-bit message length (number of bits)

11.1.162 hash_instance_t

```
typedef struct{
    hash_ctrl_t *  p_ctrl
    hash_cfg_t const  *  p_cfg
    hash_api_t const  *  p_api
} hash_instance_t
```

11.1.162.1 p_ctrl

`hash_ctrl_t::p_ctrl`

Brief description

Pointer to the control structure for this instance.

11.1.162.2 p_cfg

`hash_cfg_t::p_cfg`

Brief description

Pointer to the configuration structure for this instance.

11.1.162.3 p_api

`hash_api_t::p_api`

Brief description

Pointer to the API structure for this instance.

11.1.163 i2c_api_master_t

```
typedef struct{
    ssp_err_t(*  open)(i2c_ctrl_t *const p_ctrl, i2c_cfg_t const *const p_cfg)
    ssp_err_t(*  close)(i2c_ctrl_t *const p_ctrl)
```

```
    ssp_err_t(*  read)(i2c_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t
const bytes, bool const restart)
    ssp_err_t(*  write)(i2c_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t
const bytes, bool const restart)
    ssp_err_t(*  reset)(i2c_ctrl_t *const p_ctrl)
    ssp_err_t(*  slaveAddressSet)(i2c_ctrl_t *const p_ctrl, uint16_t const slave,
i2c_addr_mode_t const addr_mode)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} i2c_api_master_t
```

11.1.164 i2c_api_slave_t

```
typedef struct{
    ssp_err_t(*  open)(i2c_ctrl_t *const p_ctrl, i2c_cfg_t const *const p_cfg)
    ssp_err_t(*  close)(i2c_ctrl_t *const p_ctrl)
    ssp_err_t(*  masterWriteSlaveRead)(i2c_ctrl_t *const p_ctrl, uint8_t *const
p_dest, uint32_t const bytes)
    ssp_err_t(*  masterReadSlaveWrite)(i2c_ctrl_t *const p_ctrl, uint8_t *const
p_src, uint32_t const bytes)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} i2c_api_slave_t
```

11.1.165 i2c_callback_args_t

```
typedef struct{
    void const  *const  p_context
    uint32_t const  bytes
    i2c_event_t const  event
} i2c_callback_args_t
```

11.1.165.1 p_context

```
void const* const i2c_callback_args_t::p_context
```

Brief description

Pointer to user-provided context.

11.1.165.2 bytes

```
uint32_t const i2c_callback_args_t::bytes
```

Brief description

Number of received/transmitted bytes in buff.

11.1.165.3 event

i2c_event_t::event

Brief description

Event code.

11.1.166 i2c_cfg_t

```
typedef struct{
    uint8_t  channel
    i2c_rate_t  rate
    uint16_t  slave
    i2c_addr_mode_t  addr_mode
    uint16_t  sda_delay
    uint8_t  rxi_ipl
    uint8_t  txi_ipl
    uint8_t  tei_ipl
    uint8_t  eri_ipl
    transfer_instance_t const  *  p_transfer_tx
    transfer_instance_t const  *  p_transfer_rx
    void(*  p_callback)(i2c_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} i2c_cfg_t
```

11.1.166.1 channel

uint8_t i2c_cfg_t::channel

Brief description

Identifier recognizable by implementation.

Detailed description

Generic configuration

11.1.166.2 rate

i2c_rate_t::rate

Brief description

Device's maximum clock rate from enum i2c_rate_t.

11.1.166.3 slave

uint16_t i2c_cfg_t::slave

Brief description

The address of the slave device.

11.1.166.4 addr_mode

`i2c_addr_mode_t::addr_mode`

Brief description

Indicates how slave fields should be interpreted.

11.1.166.5 sda_delay

`uint16_t i2c_cfg_t::sda_delay`

Brief description

The SDA output delay.

11.1.166.6 rxi_ipl

`uint8_t i2c_cfg_t::rxi_ipl`

Brief description

Receive interrupt priority.

11.1.166.7 txi_ipl

`uint8_t i2c_cfg_t::txi_ipl`

Brief description

Transmit interrupt priority.

11.1.166.8 tei_ipl

`uint8_t i2c_cfg_t::tei_ipl`

Brief description

Transmit end interrupt priority.

11.1.166.9 eri_ipl

`uint8_t i2c_cfg_t::eri_ipl`

Brief description

Error interrupt priority.

11.1.166.10 p_transfer_tx

`transfer_instance_t::p_transfer_tx`

Brief description

DTC instance for I2C transmit.Set to NULL if unused.

Detailed description

DTC/DMA support

11.1.166.11 p_transfer_rx

`transfer_instance_t::p_transfer_rx`

Brief description

DTC instance for I2C receive. Set to NULL if unused.

11.1.166.12 p_callback

```
void(* i2c_cfg_t::p_callback) (i2c_callback_args_t *p_args)
```

Brief description

Pointer to callback function.

Detailed description

Parameters to control software behavior

11.1.166.13 p_context

```
void const* i2c_cfg_t::p_context
```

Brief description

Pointer to the user-provided context.

11.1.166.14 p_extend

```
void const* i2c_cfg_t::p_extend
```

Brief description

Any configuration data needed by the hardware.

Detailed description

Implementation-specific configuration

11.1.167 i2c_master_instance_t

```
typedef struct{
    i2c_ctrl_t *  p_ctrl
    i2c_cfg_t const  *  p_cfg
    i2c_api_master_t const  *  p_api
} i2c_master_instance_t
```

11.1.167.1 p_ctrl

i2c_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.167.2 p_cfg

i2c_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.167.3 p_api

i2c_api_master_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.168 i2c_slave_instance_t

```
typedef struct{
    i2c_ctrl_t *  p_ctrl
    i2c_cfg_t const  *  p_cfg
    i2c_api_slave_t const  *  p_api
} i2c_slave_instance_t
```

11.1.168.1 p_ctrl

i2c_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.168.2 p_cfg

i2c_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.168.3 p_api

i2c_api_slave_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.169 i2s_api_t

```
typedef struct{
    ssp_err_t(*  open)(i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg)
    ssp_err_t(*  stop)(i2s_ctrl_t *const p_ctrl, i2s_dir_t const dir)
    ssp_err_t(*  mute)(i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)
    ssp_err_t(*  write)(i2s_ctrl_t *const p_ctrl, uint8_t const *const p_src,
uint16_t const bytes)
    ssp_err_t(*  read)(i2s_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint16_t
const bytes)
    ssp_err_t(*  writeRead)(i2s_ctrl_t *const p_ctrl, uint8_t const *const p_src,
uint8_t *const p_dest, uint16_t const bytes)
    ssp_err_t(*  infoGet)(i2s_ctrl_t *const p_ctrl, i2s_info_t *const p_info)
    ssp_err_t(*  close)(i2s_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} i2s_api_t
```

11.1.170 i2s_callback_args_t

```
typedef struct{
    void const  *  p_context
    i2s_event_t  event
} i2s_callback_args_t
```

11.1.170.1 p_context

void const* i2s_callback_args_t::p_context

Detailed description

Placeholder for user data. Set in open function in i2s_cfg_t.

11.1.170.2 event

i2s_event_t::event

Brief description

The event can be used to identify what caused the callback (overflow or error).

11.1.171 i2s_cfg_t

```
typedef struct{
    uint8_t  channel
    i2s_pcm_width_t  pcm_width
    i2s_word_length_t  word_length
    i2s_ws_continue_t  ws_continue
    uint32_t  sampling_freq_hz
    uint32_t  audio_clk_freq_hz
    timer_instance_t const  *  p_timer
    transfer_instance_t const  *  p_transfer_tx
    transfer_instance_t const  *  p_transfer_rx
    void(*  p_callback)(i2s_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
    uint8_t  rxi_ipl
    uint8_t  txi_ipl
    uint8_t  idle_err_ipl
} i2s_cfg_t
```

11.1.171.1 channel

uint8_t i2s_cfg_t::channel

Detailed description

Select a channel corresponding to the channel number of the hardware.

11.1.171.2 pcm_width

i2s_pcm_width_t::pcm_width

Brief description

Audio PCM data width.

11.1.171.3 word_length

i2s_word_length_t::word_length

Brief description

Audio word length, bits must be >= pcm_width bits.

11.1.171.4 ws_continue

`i2s_ws_continue_t::ws_continue`

Brief description

Whether to continue WS transmission during idle state.

11.1.171.5 sampling_freq_hz

`uint32_t i2s_cfg_t::sampling_freq_hz`

Brief description

Sampling frequency in Hertz.

11.1.171.6 audio_clk_freq_hz

`uint32_t i2s_cfg_t::audio_clk_freq_hz`

Detailed description

Audio clock frequency in Hertz. Must be a multiple between 1 and 128 of (16 * sampling_freq_hz * (word_length <enum_value> + 1)

11.1.171.7 p_timer

`timer_instance_t::p_timer`

Detailed description

To generate audio clock with GPT, link a timer instance here. Set to NULL if unused.

11.1.171.8 p_transfer_tx

`transfer_instance_t::p_transfer_tx`

Detailed description

To use DTC during write, link a DTC instance here. Set to NULL if unused.

11.1.171.9 p_transfer_rx

`transfer_instance_t::p_transfer_rx`

Detailed description

To use DTC during read, link a DTC instance here. Set to NULL if unused.

11.1.171.10 p_callback

```
void(* i2s_cfg_t::p_callback) (i2s_callback_args_t *p_args)
```

Detailed description

Callback provided when an I2S ISR occurs. Set to NULL for no CPU interrupt.

11.1.171.11 p_context

```
void const* i2s_cfg_t::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in i2s_callback_args_t.

11.1.171.12 p_extend

```
void const* i2s_cfg_t::p_extend
```

Brief description

Extension parameter for hardware specific settings.

11.1.171.13 rxi_ipl

```
uint8_t i2s_cfg_t::rxi_ipl
```

Brief description

Receive interrupt priority.

11.1.171.14 txi_ipl

```
uint8_t i2s_cfg_t::txi_ipl
```

Brief description

Transmit interrupt priority.

11.1.171.15 idle_err_ipl

```
uint8_t i2s_cfg_t::idle_err_ipl
```

Brief description

Idle/Error interrupt priority.

11.1.172 i2s_info_t

```
typedef struct{
    i2s_status_t  status
    uint32_t  sampling_freq_hz
} i2s_info_t
```

11.1.172.1 status

i2s_status_t::status

11.1.172.2 sampling_freq_hz

uint32_t i2s_info_t::sampling_freq_hz

Brief description

Sampling frequency in Hertz.

11.1.173 i2s_instance_t

```
typedef struct{
    i2s_ctrl_t *  p_ctrl
    i2s_cfg_t const  *  p_cfg
    i2s_api_t const  *  p_api
} i2s_instance_t
```

11.1.173.1 p_ctrl

i2s_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.173.2 p_cfg

i2s_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.173.3 p_api

i2s_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.174 i2s_on_ssi_cfg_t

```
typedef struct{
    ssi_audio_clock_t  audio_clock
} i2s_on_ssi_cfg_t
```

11.1.174.1 audio_clock

ssi_audio_clock_t::audio_clock

Brief description

Audio clock source, default is SSI_AUDIO_CLOCK_EXTERNAL.

11.1.175 icu_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    R_ICU_Type *  p_reg
    void(*  p_callback)(external_irq_callback_args_t *p_args)
    void const  *  p_context
    IRQn_Type  irq
    uint8_t  channel
} icu_instance_ctrl_t
```

11.1.175.1 open

uint32_t ::open

Brief description

Used to determine if channel control block is in use.

11.1.175.2 p_reg

R_ICU_Type* ::p_reg

Brief description

Pointer to register base address.

11.1.175.3 p_callback

void(* ::p_callback) (*p_args)

Detailed description

Callback provided when a external IRQ ISR occurs. Set to NULL for no CPU interrupt.

11.1.175.4 p_context

```
void const* ::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in external_irq_callback_args_t.

11.1.175.5 irq

```
IRQn_Type ::irq
```

Brief description

NVIC interrupt number.

11.1.175.6 channel

```
uint8_t ::channel
```

Brief description

Channel.

11.1.176 in_addr

```
typedef struct{
    unsigned long  s_addr
    ULONG  s_addr
} in_addr
```

11.1.176.1 s_addr

```
ULONG in_addr::s_addr
```

Brief description

load with inet_aton()

Detailed description

Load with inet_aton()

11.1.176.2 s_addr

```
ULONG in_addr::s_addr
```

11.1.177 input_capture_api_t

```
typedef struct{
    ssp_err_t(*  open)(input_capture_ctrl_t *const p_ctrl, input_capture_cfg_t
const *const p_cfg)
    ssp_err_t(*  disable)(input_capture_ctrl_t const *const p_ctrl)
    ssp_err_t(*  enable)(input_capture_ctrl_t const *const p_ctrl)
    ssp_err_t(*  infoGet)(input_capture_ctrl_t const *const p_ctrl,
input_capture_info_t *const p_info)
    ssp_err_t(*  lastCaptureGet)(input_capture_ctrl_t const *const p_ctrl,
input_capture_capture_t *const p_counter)
    ssp_err_t(*  close)(input_capture_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} input_capture_api_t
```

11.1.178 input_capture_callback_args_t

```
typedef struct{
    uint8_t  channel
    input_capture_event_t  event
    uint32_t  counter
    uint32_t  overflows
    void const  *  p_context
} input_capture_callback_args_t
```

11.1.178.1 channel

uint8_t input_capture_callback_args_t::channel

Brief description

The channel being used.

11.1.178.2 event

input_capture_event_t::event

Brief description

The event that caused the interrupt and callback.

11.1.178.3 counter

uint32_t input_capture_callback_args_t::counter

Brief description

The value of the timer captured at the time of interrupt.

11.1.178.4 overflows

```
uint32_t input_capture_callback_args_t::overflows
```

Brief description

The number of counter overflows that occurred during this measurement.

11.1.178.5 p_context

```
void const* input_capture_callback_args_t::p_context
```

Brief description

Placeholder for user data, set in p_context.

11.1.179 input_capture_capture_t

```
typedef struct{
    uint32_t  counter
    uint32_t  overflows
} input_capture_capture_t
```

11.1.179.1 counter

```
uint32_t input_capture_capture_t::counter
```

Brief description

The value of the timer captured at the time of interrupt.

11.1.179.2 overflows

```
uint32_t input_capture_capture_t::overflows
```

Brief description

The number of counter overflows that occurred during this measurement.

11.1.180 input_capture_cfg_t

```
typedef struct{
    uint8_t  channel
    uint8_t  capture_irq_ipl
    uint8_t  overflow_irq_ipl
    input_capture_mode_t  mode
    input_capture_signal_edge_t  edge
    input_capture_repetition_t  repetition
    bool  autostart
```

```
    void const  *  p_extend
    void(*  p_callback)(input_capture_callback_args_t *p_args)
    void const  *  p_context
} input_capture_cfg_t
```

11.1.180.1 channel

```
uint8_t input_capture_cfg_t::channel
```

Brief description

The channel in use.

11.1.180.2 capture_irq_ipl

```
uint8_t input_capture_cfg_t::capture_irq_ipl
```

Brief description

Capture interrupt priority.

11.1.180.3 overflow_irq_ipl

```
uint8_t input_capture_cfg_t::overflow_irq_ipl
```

Brief description

Overflow interrupt priority.

11.1.180.4 mode

```
input_capture_mode_t::mode
```

Brief description

The mode of measurement to be performed.

11.1.180.5 edge

```
input_capture_signal_edge_t::edge
```

Brief description

The triggering edge to start a measurement (rise or fall).

11.1.180.6 repetition

```
input_capture_repetition_t::repetition
```

Brief description

One-shot or periodic measurement.

11.1.180.7 autostart

```
bool input_capture_cfg_t::autostart
```

Brief description

Specifies whether interrupts are enabled or not after open.

11.1.180.8 p_extend

```
void const* input_capture_cfg_t::p_extend
```

Detailed description

REQUIRED. Pointer to peripheral-specific extension parameters. See gpt_input_capture_extend_t for GPT.

11.1.180.9 p_callback

```
void(* input_capture_cfg_t::p_callback) (input_capture_callback_args_t *p_args)
```

Detailed description

Pointer to user's callback function, or NULL if no interrupt desired.

11.1.180.10 p_context

```
void const* input_capture_cfg_t::p_context
```

Brief description

Pointer to user's context data, to be passed to the callback.

11.1.181 input_capture_info_t

```
typedef struct{
    input_capture_status_t  status
    input_capture_variant_t  variant
} input_capture_info_t
```

11.1.181.1 status

```
input_capture_status_t::status
```

Brief description

Whether or not a capture is in progress.

11.1.181.2 variant

input_capture_variant_t::variant

Brief description

Whether timer is 16 or 32 bits.

11.1.182 input_capture_instance_t

```
typedef struct{
    input_capture_ctrl_t *  p_ctrl
    input_capture_cfg_t const  *  p_cfg
    input_capture_api_t const  *  p_api
} input_capture_instance_t
```

11.1.182.1 p_ctrl

input_capture_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.182.2 p_cfg

input_capture_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.182.3 p_api

input_capture_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.183 ioport_api_t

```
typedef struct{
    ssp_err_t(*  init)(const ioport_cfg_t *p_cfg)
    ssp_err_t(*  pinsCfg)(const ioport_cfg_t *p_cfg)
    ssp_err_t(*  pinCfg)(ioport_port_pin_t pin, uint32_t cfg)
    ssp_err_t(*  pinDirectionSet)(ioport_port_pin_t pin, ioport_direction_t
direction)
    ssp_err_t(*  pinEventInputRead)(ioport_port_pin_t pin, ioport_level_t
```

```
*p_pin_event)
    ssp_err_t(*  pinEventOutputWrite)(ioport_port_pin_t pin, ioport_level_t
pin_value)
    ssp_err_t(*  pinEthernetModeCfg)(ioport_ethernet_channel_t channel,
ioport_ethernet_mode_t mode)
    ssp_err_t(*  pinRead)(ioport_port_pin_t pin, ioport_level_t *p_pin_value)
    ssp_err_t(*  pinWrite)(ioport_port_pin_t pin, ioport_level_t level)
    ssp_err_t(*  portDirectionSet)(ioport_port_t port, ioport_size_t
direction_values, ioport_size_t mask)
    ssp_err_t(*  portEventInputRead)(ioport_port_t port, ioport_size_t
*p_event_data)
    ssp_err_t(*  portEventOutputWrite)(ioport_port_t port, ioport_size_t
event_data, ioport_size_t mask_value)
    ssp_err_t(*  portRead)(ioport_port_t port, ioport_size_t *p_port_value)
    ssp_err_t(*  portWrite)(ioport_port_t port, ioport_size_t value,
ioport_size_t mask)
    ssp_err_t(*  versionGet)(ssp_version_t *p_data)
} ioport_api_t
```

11.1.184 ioport_cfg_t

```
typedef struct{
    uint16_t  number_of_pins
    ioport_pin_cfg_t const  *  p_pin_cfg_data
} ioport_cfg_t
```

11.1.184.1 number_of_pins

```
uint16_t ioport_cfg_t::number_of_pins
```

Brief description

Number of pins for which there is configuration data.

11.1.184.2 p_pin_cfg_data

```
ioport_pin_cfg_t::p_pin_cfg_data
```

Brief description

Pin configuration data.

11.1.185 ioport_instance_t

```
typedef struct{
    ioport_cfg_t const  *  p_cfg
    ioport_api_t const  *  p_api
} ioport_instance_t
```

11.1.185.1 p_cfg

ioport_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.185.2 p_api

ioport_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.186 ioport_pin_cfg_t

```
typedef struct{
    uint32_t  pin_cfg
    ioport_port_pin_t  pin
} ioport_pin_cfg_t
```

11.1.186.1 pin_cfg

uint32_t ioport_pin_cfg_t::pin_cfg

Brief description

Pin PFS configuration - Use ioport_cfg_options_t parameters to configure.

11.1.186.2 pin

ioport_port_pin_t::pin

Brief description

Pin identifier.

11.1.187 iwdt_instance_ctrl_t

```
typedef struct{
    bool  wdt_open
    void const  *  p_context
    R_IWDT_Type *  p_reg
    void(*  p_callback)(wdt_callback_args_t *p_args)
} iwdt_instance_ctrl_t
```

11.1.187.1 wdt_open

```
bool ::wdt_open
```

Detailed description

Indicates whether the open() API has been successfully called.

11.1.187.2 p_context

```
void const* ::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in wdt_callback_args_t.

11.1.187.3 p_reg

```
R_IWDT_Type* ::p_reg
```

Brief description

Pointer to register base address.

11.1.187.4 p_callback

```
void(* ::p_callback) ( *p_args)
```

Brief description

Callback provided when a WDT NMI ISR occurs.

11.1.188 jpeg_decode_api_t

```
typedef struct{
    ssp_err_t(*  open)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_cfg_t const
*const p_cfg)
    ssp_err_t(*  outputBufferSet)(jpeg_decode_ctrl_t *const p_ctrl, void
*p_buffer, uint32_t buffer_size)
    ssp_err_t(*  horizontalStrideSet)(jpeg_decode_ctrl_t *const p_ctrl, uint32_t
horizontal_stride)
    ssp_err_t(*  imageSubsampleSet)(jpeg_decode_ctrl_t *const p_ctrl,
jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t
vertical_subsample)
    ssp_err_t(*  inputBufferSet)(jpeg_decode_ctrl_t *const p_ctrl, void
*p_buffer, uint32_t buffer_size)
    ssp_err_t(*  linesDecodedGet)(jpeg_decode_ctrl_t *const p_ctrl, uint32_t
*const p_lines)
    ssp_err_t(*  imageSizeGet)(jpeg_decode_ctrl_t *const p_ctrl, uint16_t
*p_horizontal_size, uint16_t *p_vertical_size)
```

```
    ssp_err_t(*  statusGet)(jpeg_decode_ctrl_t *const p_ctrl,
jpeg_decode_status_t *const p_status)
    ssp_err_t(*  close)(jpeg_decode_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *p_version)
    ssp_err_t(*  pixelFormatGet)(jpeg_decode_ctrl_t *const p_ctrl,
jpeg_decode_color_space_t *const p_color_space)
} jpeg_decode_api_t
```

11.1.189 jpeg_decode_callback_args_t

```
typedef struct{
    jpeg_decode_status_t  status
    void const  *  p_context
} jpeg_decode_callback_args_t
```

11.1.189.1 status

`jpeg_decode_status_t::status`

Brief description

JPEG status.

11.1.189.2 p_context

`void const* jpeg_decode_callback_args_t::p_context`

Brief description

Pointer to user-provided context.

11.1.190 jpeg_decode_cfg_t

```
typedef struct{
    jpeg_decode_color_space_t  color_space
    jpeg_decode_data_format_t  input_data_format
    jpeg_decode_data_format_t  output_data_format
    jpeg_decode_pixel_format_t  pixel_format
    uint8_t  alpha_value
    uint8_t  jdti_ipl
    uint8_t  jedi_ipl
    void(*  p_callback)(jpeg_decode_callback_args_t *p_args)
    void const  *  p_context
} jpeg_decode_cfg_t
```

11.1.190.1 color_space

jpeg_decode_color_space_t::color_space

Brief description

Color space.

11.1.190.2 input_data_format

jpeg_decode_data_format_t::input_data_format

Brief description

Input data stream byte order.

11.1.190.3 output_data_format

jpeg_decode_data_format_t::output_data_format

Brief description

Output data stream byte order.

11.1.190.4 pixel_format

jpeg_decode_pixel_format_t::pixel_format

Brief description

Pixel format.

11.1.190.5 alpha_value

uint8_t jpeg_decode_cfg_t::alpha_value

Brief description

Alpha value to be applied to decoded pixel data. Only valid for ARGB888 format.

11.1.190.6 jdti_ipl

uint8_t jpeg_decode_cfg_t::jdti_ipl

Brief description

Data transfer interrupt priority.

11.1.190.7 jedi_ipl

uint8_t jpeg_decode_cfg_t::jedi_ipl

Brief description

Decompression interrupt priority.

11.1.190.8 p_callback

```
void(* jpeg_decode_cfg_t::p_callback) (jpeg_decode_callback_args_t *p_args)
```

Brief description

User-supplied callback functions.

11.1.190.9 p_context

```
void const* jpeg_decode_cfg_t::p_context
```

Brief description

Placeholder for user data. Passed to user callback in jpeg_decode_callback_args_t.

11.1.191 jpeg_decode_instance_ctrl_t

```
typedef struct{
    jpeg_decode_status_t  status
    ssp_err_t  error_code
    void(*  p_callback)(jpeg_decode_callback_args_t *p_args)
    void const  *  p_extend
    void const  *  p_context
    R_JPEG_Type *  p_reg
    jpeg_decode_pixel_format_t  pixel_format
    uint32_t  horizontal_stride
    uint32_t  outbuffer_size
    uint16_t  total_lines_decoded
    jpeg_decode_subsample_t  horizontal_subsample
} jpeg_decode_instance_ctrl_t
```

11.1.191.1 status

```
jpeg_decode_status_t::status
```

Brief description

JPEG Codec module status.

11.1.191.2 error_code

```
::error_code
```

Brief description

JPEG Codec error code (if any).

11.1.191.3 p_callback

```
void(* ::p_callback) ( *p_args)
```

Brief description

User-supplied callback functions.

11.1.191.4 p_extend

```
void const* ::p_extend
```

Brief description

JPEG Codec hardware dependent configuration */.

11.1.191.5 p_context

```
void const* ::p_context
```

Brief description

Placeholder for user data. Passed to user callback in jpeg_decode_callback_args_t.

11.1.191.6 p_reg

```
R_JPEG_Type* ::p_reg
```

Brief description

Pointer to register base address.

11.1.191.7 pixel_format

```
jpeg_decode_pixel_format_t::pixel_format
```

Brief description

Pixel format.

11.1.191.8 horizontal_stride

```
uint32_t ::horizontal_stride
```

Brief description

Horizontal Stride settings.

11.1.191.9 outbuffer_size

```
uint32_t ::outbuffer_size
```

Brief description

out buffer size

11.1.191.10 total_lines_decoded

```
uint16_t ::total_lines_decoded
```

Brief description

Track the number of lines decoded so far.

11.1.191.11 horizontal_subsample

```
jpeg_decode_subsample_t::horizontal_subsample
```

Brief description

Horizontal sub-sample setting.

11.1.192 jpeg_decode_instance_t

```
typedef struct{
    jpeg_decode_ctrl_t *  p_ctrl
    jpeg_decode_cfg_t const  *  p_cfg
    jpeg_decode_api_t const  *  p_api
} jpeg_decode_instance_t
```

11.1.192.1 p_ctrl

```
jpeg_decode_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.192.2 p_cfg

```
jpeg_decode_cfg_t::p_cfg
```

Brief description

Pointer to the configuration structure for this instance.

11.1.192.3 p_api

jpeg_decode_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.193 jpeg_encode_api_t

```
typedef struct{
    ssp_err_t(*  open)(jpeg_encode_ctrl_t *const p_ctrl, jpeg_encode_cfg_t const
*const p_cfg)
    ssp_err_t(*  imageParameterSet)(jpeg_encode_ctrl_t *const p_ctrl,
jpeg_encode_raw_image_parameters *p_raw_image_parameters)
    ssp_err_t(*  outputBufferSet)(jpeg_encode_ctrl_t *const p_ctrl, void
*p_buffer)
    ssp_err_t(*  inputBufferSet)(jpeg_encode_ctrl_t *const p_ctrl, void
*p_buffer, uint32_t buffer_size)
    ssp_err_t(*  statusGet)(jpeg_encode_ctrl_t *const p_ctrl, volatile
jpeg_encode_status_t *p_status)
    ssp_err_t(*  close)(jpeg_encode_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *p_version)
} jpeg_encode_api_t
```

11.1.194 jpeg_encode_callback_args_t

```
typedef struct{
    jpeg_encode_status_t  status
    uint32_t  image_size
    void const  *  p_context
} jpeg_encode_callback_args_t
```

11.1.194.1 status

jpeg_encode_status_t::status

Brief description

JPEG status.

11.1.194.2 image_size

uint32_t jpeg_encode_callback_args_t::image_size

Brief description

JPEG image size.

11.1.194.3 p_context

```
void const* jpeg_encode_callback_args_t::p_context
```

Brief description

Pointer to user-provided context.

11.1.195 jpeg_encode_cfg_t

```
typedef struct{
    jpeg_encode_data_format_t  input_data_format
    jpeg_encode_data_format_t  output_data_format
    uint16_t  dri_marker
    uint8_t  jdti_ipl
    uint8_t  jedi_ipl
    uint8_t  quality_factor
    uint16_t  vertical_resolution
    uint16_t  horizontal_resolution
    uint8_t const  *  p_quant_luma_table
    uint8_t const  *  p_quant_croma_table
    uint8_t const  *  p_huffman_luma_ac_table
    uint8_t const  *  p_huffman_luma_dc_table
    uint8_t const  *  p_huffman_croma_ac_table
    uint8_t const  *  p_huffman_croma_dc_table
    void(*  p_callback)(jpeg_encode_callback_args_t *p_args)
    void const  *  p_context
} jpeg_encode_cfg_t
```

11.1.195.1 input_data_format

```
jpeg_encode_data_format_t::input_data_format
```

Brief description

Input data stream byte order.

11.1.195.2 output_data_format

```
jpeg_encode_data_format_t::output_data_format
```

Brief description

Output data stream byte order.

11.1.195.3 dri_marker

```
uint16_t jpeg_encode_cfg_t::dri_marker
```

Brief description

DRI Marker setting 0 :- No DRI and RST marker.

11.1.195.4 jdti_ipl

```
uint8_t jpeg_encode_cfg_t::jdti_ipl
```

Brief description

Data transfer interrupt priority.

11.1.195.5 jedi_ipl

```
uint8_t jpeg_encode_cfg_t::jedi_ipl
```

Brief description

Decompression interrupt priority.

11.1.195.6 quality_factor

```
uint8_t jpeg_encode_cfg_t::quality_factor
```

Brief description

JPEG image quality.

11.1.195.7 vertical_resolution

```
uint16_t jpeg_encode_cfg_t::vertical_resolution
```

Brief description

vertical resolution of input image

11.1.195.8 horizontal_resolution

```
uint16_t jpeg_encode_cfg_t::horizontal_resolution
```

Brief description

horizontal resolution of input image

11.1.195.9 p_quant_luma_table

```
uint8_t const* jpeg_encode_cfg_t::p_quant_luma_table
```

Brief description

Luma table.

11.1.195.10 p_quant_croma_table

```
uint8_t const* jpeg_encode_cfg_t::p_quant_croma_table
```

Brief description

croma table

11.1.195.11 p_huffman_luma_ac_table

```
uint8_t const* jpeg_encode_cfg_t::p_huffman_luma_ac_table
```

Brief description

Huffman AC table for luma.

11.1.195.12 p_huffman_luma_dc_table

```
uint8_t const* jpeg_encode_cfg_t::p_huffman_luma_dc_table
```

Brief description

Huffman DC table for luma.

11.1.195.13 p_huffman_croma_ac_table

```
uint8_t const* jpeg_encode_cfg_t::p_huffman_croma_ac_table
```

Brief description

Huffman AC table for croma.

11.1.195.14 p_huffman_croma_dc_table

```
uint8_t const* jpeg_encode_cfg_t::p_huffman_croma_dc_table
```

Brief description

Huffman DC table for croma.

11.1.195.15 p_callback

```
void(* jpeg_encode_cfg_t::p_callback) (jpeg_encode_callback_args_t *p_args)
```

Brief description

User-supplied callback functions.

11.1.195.16 p_context

```
void const* jpeg_encode_cfg_t::p_context
```

Brief description

Placeholder for user data. Passed to user callback in jpeg_encode_callback_args_t.

11.1.196 jpeg_encode_instance_ctrl_t

```
typedef struct{
    jpeg_encode_status_t  status
    void(*  p_callback)(jpeg_encode_callback_args_t *p_args)
    void const  *  p_extend
    void const  *  p_context
    R_JPEG_Type *  p_reg
    uint32_t  horizontal_stride
    uint32_t  output_buffer_size
    uint16_t  lines_to_encoded
    uint16_t  vertical_resolution
} jpeg_encode_instance_ctrl_t
```

11.1.196.1 status

jpeg_encode_status_t::status

Brief description

JPEG Codec module status.

11.1.196.2 p_callback

```
void(* ::p_callback) ( *p_args)
```

Brief description

User-supplied callback functions.

11.1.196.3 p_extend

```
void const* ::p_extend
```

Brief description

JPEG Codec hardware dependent configuration */.

11.1.196.4 p_context

```
void const* ::p_context
```

Brief description

Placeholder for user data. Passed to user callback in jpeg_encode_callback_args_t.

11.1.196.5 p_reg

```
R_JPEG_Type* ::p_reg
```

Brief description

Pointer to register base address.

11.1.196.6 horizontal_stride

```
uint32_t ::horizontal_stride
```

Brief description

Horizontal Stride settings (Line offset).

11.1.196.7 output_buffer_size

```
uint32_t ::output_buffer_size
```

Brief description

out buffer size

11.1.196.8 lines_to_encoded

```
uint16_t ::lines_to_encoded
```

Brief description

Number of lines to encode.

11.1.196.9 vertical_resolution

```
uint16_t ::vertical_resolution
```

Brief description

vertical size

11.1.197 jpeg_encode_instance_t

```
typedef struct{
    jpeg_encode_ctrl_t *  p_ctrl
    jpeg_encode_cfg_t const  *  p_cfg
    jpeg_encode_api_t const  *  p_api
} jpeg_encode_instance_t
```

11.1.197.1 p_ctrl

jpeg_encode_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.197.2 p_cfg

jpeg_encode_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.197.3 p_api

jpeg_encode_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.198 jpeg_encode_raw_image_parameters

```
typedef struct{
    uint16_t  horizontal_stride
    uint16_t  horizontal_resolution
    uint16_t  vertical_resolution
} jpeg_encode_raw_image_parameters
```

11.1.198.1 horizontal_stride

uint16_t jpeg_encode_raw_image_parameters::horizontal_stride

Brief description

Horizontal stride.

11.1.198.2 horizontal_resolution

uint16_t jpeg_encode_raw_image_parameters::horizontal_resolution

Brief description

Horizontal Resolution in pixel.

11.1.198.3 vertical_resolution

```
uint16_t jpeg_encode_raw_image_parameters::vertical_resolution
```

Brief description

Vertical Resolution in pixel.

11.1.199 key_installation_instance_ctrl_t

```
typedef struct{
    crypto_ctrl_t *  p_crypto_ctrl
    crypto_api_t const  *  p_crypto_api
} key_installation_instance_ctrl_t
```

11.1.199.1 p_crypto_ctrl

```
* ::p_crypto_ctrl
```

Brief description

pointer to crypto engine control structure

11.1.199.2 p_crypto_api

```
const* ::p_crypto_api
```

Brief description

pointer to crypto engine API

11.1.200 keymatrix_api_t

```
typedef struct{
    ssp_err_t(*  open)(keymatrix_ctrl_t *const p_ctrl, keymatrix_cfg_t const
*const p_cfg)
    ssp_err_t(*  enable)(keymatrix_ctrl_t *const p_ctrl)
    ssp_err_t(*  disable)(keymatrix_ctrl_t *const p_ctrl)
    ssp_err_t(*  triggerSet)(keymatrix_ctrl_t *const p_ctrl, keymatrix_trigger_t
const trigger)
    ssp_err_t(*  close)(keymatrix_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} keymatrix_api_t
```

11.1.201 keymatrix_callback_args_t

```
typedef struct{
    void const  *  p_context
```

```
    keymatrix_channels_t  channels
} keymatrix_callback_args_t
```

11.1.201.1 p_context

```
void const* keymatrix_callback_args_t::p_context
```

Brief description

Holder for user data. Set in open function in keymatrix_cfg_t.

11.1.201.2 channels

```
keymatrix_channels_t::channels
```

Detailed description

Bit vector representing the physical hardware channel(s) that caused the interrupt. The bit vector is used for compatibility with matrix designs where more than one input will be active at once.
NOTE: Not all HAL drivers support matrix mode. See r_kint.h for details.

11.1.202 keymatrix_cfg_t

```
typedef struct{
    keymatrix_channels_t  channels
    keymatrix_trigger_t  trigger
    bool  autostart
    void(*  p_callback)(keymatrix_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
    uint8_t  irq_ipl
} keymatrix_cfg_t
```

11.1.202.1 channels

```
keymatrix_channels_t::channels
```

Brief description

Key Input channel(s). Bit mask of channels to open.

11.1.202.2 trigger

```
keymatrix_trigger_t::trigger
```

Brief description

Key Input trigger setting.

11.1.202.3 autostart

```
bool keymatrix_cfg_t::autostart
```

Brief description

Start operation and enable interrupts during open().

11.1.202.4 p_callback

```
void(* keymatrix_cfg_t::p_callback) (keymatrix_callback_args_t *p_args)
```

Brief description

Callback for key interrupt ISR.

11.1.202.5 p_context

```
void const* keymatrix_cfg_t::p_context
```

Brief description

Holder for user data. Passed to callback in keymatrix_user_cb_data_t.

11.1.202.6 p_extend

```
void const* keymatrix_cfg_t::p_extend
```

Brief description

Extension parameter for hardware specific settings.

11.1.202.7 irq_ipl

```
uint8_t keymatrix_cfg_t::irq_ipl
```

Brief description

Interrupt priority level.

11.1.203 keymatrix_instance_t

```
typedef struct{
    keymatrix_ctrl_t *  p_ctrl
    keymatrix_cfg_t const  *  p_cfg
    keymatrix_api_t const  *  p_api
} keymatrix_instance_t
```

11.1.203.1 p_ctrl

keymatrix_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.203.2 p_cfg

keymatrix_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.203.3 p_api

keymatrix_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.204 kint_instance_ctrl_t

```
typedef struct{
    R_KINT_Type *  p_reg
    keymatrix_channels_t  channels
    IRQn_Type  irq
} kint_instance_ctrl_t
```

11.1.204.1 p_reg

R_KINT_Type* ::p_reg

Brief description

Pointer to register base address.

11.1.204.2 channels

keymatrix_channels_t::channels

Brief description

Channel bitmask.

11.1.204.3 irq

```
IRQn_Type ::irq
```

Brief description

Interrupt priority number.

11.1.205 lpm_api_t

```
typedef struct{
    ssp_err_t(*  init)(lpm_cfg_t const *const p_cfg)
    ssp_err_t(*  mstpcrSet)(uint32_t mstpcra_value, uint32_t mstpcrb_value,
uint32_t mstpcrc_value, uint32_t mstpcrd_value)
    ssp_err_t(*  mstpcrGet)(uint32_t *mstpcra_value, uint32_t *mstpcrb_value,
uint32_t *mstpcrc_value, uint32_t *mstpcrd_value)
    ssp_err_t(*  moduleStop)(lpm_mstp_t module)
    ssp_err_t(*  moduleStart)(lpm_mstp_t module)
    ssp_err_t(*  operatingPowerModeSet)(lpm_operating_power_t power_mode,
lpm_subosc_t subosc)
    ssp_err_t(*  snoozeEnable)(lpm_snooze_rxd0_t rdx0_mode, lpm_snooze_dtc_t
dtc_mode, lpm_snooze_request_t requests, lpm_snooze_end_t triggers)
    ssp_err_t(*  snoozeDisable)(void)
    ssp_err_t(*  lowPowerCfg)(lpm_low_power_mode_t power_mode,
lpm_output_port_enable_t output_port_enable, lpm_power_supply_t power_supply,
lpm_io_port_t io_port_state)
    ssp_err_t(*  wupenSet)(uint32_t wupen_value)
    ssp_err_t(*  wupenGet)(uint32_t *wupen_value)
    ssp_err_t(*  deepStandbyCancelRequestEnable)(lpm_deep_standby_t pin_signal,
lpm_cancel_request_edge_t rising_falling)
    ssp_err_t(*  deepStandbyCancelRequestDisable)(lpm_deep_standby_t pin_signal)
    ssp_err_t(*  lowPowerModeEnter)(void)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} lpm_api_t
```

11.1.206 lpm_cfg_t

```
typedef struct{
    lpm_operating_power_t  operating_power
    lpm_subosc_t  sub_oscillator
    lpm_code_flash_t  code_flash
} lpm_cfg_t
```

11.1.206.1 operating_power

```
lpm_operating_power_t::operating_power
```

Brief description

Operating power mode.

11.1.206.2 sub_oscillator

lpm_subosc_t::sub_oscillator

Brief description

Sub oscillator.

11.1.206.3 code_flash

lpm_code_flash_t::code_flash

Brief description

Enable the code flash.

11.1.207 lpm_instance_t

```
typedef struct{
    lpm_cfg_t const  *  p_cfg
    lpm_api_t const  *  p_api
} lpm_instance_t
```

11.1.207.1 p_cfg

lpm_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.207.2 p_api

lpm_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.208 lpmv2_api_t

```
typedef struct{
    ssp_err_t(*  init)(void)
    ssp_err_t(*  lowPowerCfg)(lpmv2_cfg_t const *const p_cfg)
    ssp_err_t(*  lowPowerModeEnter)(void)
```

```
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} lpmv2_api_t
```

11.1.209 lpmv2_cfg_t

```
typedef struct{
    lpmv2_low_power_mode_t  low_power_mode
    void const  *  p_extend
} lpmv2_cfg_t
```

11.1.209.1 low_power_mode

lpmv2_low_power_mode_t::low_power_mode

Detailed description

Low Power Mode

11.1.209.2 p_extend

void const* lpmv2_cfg_t::p_extend

Detailed description

MCU Specific configuration

11.1.210 lpmv2_instance_t

```
typedef struct{
    lpmv2_cfg_t const  *const  p_cfg
    lpmv2_api_t const  *const  p_api
} lpmv2_instance_t
```

11.1.210.1 p_cfg

lpmv2_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.210.2 p_api

lpmv2_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.211 lpmv2_mcu_cfg_t

```
typedef struct{
    lpmv2_standby_wake_source_bits_t  standby_wake_sources
    lpmv2_snooze_request_t  snooze_request_source
    lpmv2_snooze_end_bits_t  snooze_end_sources
    lpmv2_snooze_dtc_t  dtc_state_in_snooze
    lpmv2_output_port_enable_t  output_port_enable
    lpmv2_io_port_t  io_port_state
    lpmv2_power_supply_t  power_supply_state
    lpmv2_deep_standby_cancel_source_bits_t  deep_standby_cancel_source
    lpmv2_deep_standby_cancel_edge_bits_t  deep_standby_cancel_edge
} lpmv2_mcu_cfg_t
```

11.1.211.1 standby_wake_sources

::standby_wake_sources

Detailed description

Bitwise list of sources to wake from standby

11.1.211.2 snooze_request_source

lpmv2_snooze_request_t::snooze_request_source

Detailed description

Snooze request source

11.1.211.3 snooze_end_sources

::snooze_end_sources

Detailed description

Bitwise list of snooze end sources

11.1.211.4 dtc_state_in_snooze

lpmv2_snooze_dtc_t::dtc_state_in_snooze

Detailed description

State of DTC in snooze mode, enabled or disabled

11.1.211.5 output_port_enable

lpmv2_output_port_enable_t::output_port_enable

Detailed description

Output port enabled/disabled in standby and deep standby

11.1.211.6 io_port_state

`lpmv2_io_port_t::io_port_state`

Detailed description

IO port state in deep standby (maintained or reset)

11.1.211.7 power_supply_state

`lpmv2_power_supply_t::power_supply_state`

Detailed description

Internal power supply state in standby and deep standby (deepcut)

11.1.211.8 deep_standby_cancel_source

`::deep_standby_cancel_source`

Detailed description

Sources that can trigger exit from deep standby

11.1.211.9 deep_standby_cancel_edge

`::deep_standby_cancel_edge`

Detailed description

Signal edges for the sources that can trigger exit from deep standby

11.1.212 lvd_api_t

```
typedef struct{
    ssp_err_t(*  open)(lvd_ctrl_t *const p_ctrl, lvd_cfg_t const *const p_cfg)
    ssp_err_t(*  statusGet)(lvd_ctrl_t *const p_ctrl, lvd_status_t *p_lvd_status)
    ssp_err_t(*  statusClear)(lvd_ctrl_t *const p_ctrl)
    ssp_err_t(*  close)(lvd_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} lvd_api_t
```

11.1.213 lvd_callback_args_t

```
typedef struct{
    uint32_t  monitor_number
    lvd_status_t  status
    void const  *  p_context
} lvd_callback_args_t
```

11.1.213.1 monitor_number

```
uint32_t lvd_callback_args_t::monitor_number
```

Brief description

Monitor number.

11.1.213.2 status

```
lvd_status_t::status
```

Brief description

Status of monitor.

11.1.213.3 p_context

```
void const* lvd_callback_args_t::p_context
```

Brief description

Placeholder for user data.

11.1.214 lvd_cfg_t

```
typedef struct{
    const uint32_t  monitor_number
    lvd_threshold_t  voltage_threshold
    lvd_response_t  detection_response
    lvd_voltage_slope_t  voltage_slope
    uint8_t  monitor_ipl
    void(*  p_callback)(lvd_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} lvd_cfg_t
```

11.1.214.1 monitor_number

```
const uint32_t lvd_cfg_t::monitor_number
```

Detailed description

Monitor number, 1, 2, ...

11.1.214.2 voltage_threshold

`lvd_threshold_t::voltage_threshold`

Detailed description

Threshold for out of range voltage detection

11.1.214.3 detection_response

`lvd_response_t::detection_response`

Detailed description

Response on detecting a threshold crossing

11.1.214.4 voltage_slope

`lvd_voltage_slope_t::voltage_slope`

Detailed description

Rising or falling voltage is to be detected

11.1.214.5 monitor_ipl

`uint8_t lvd_cfg_t::monitor_ipl`

Detailed description

Interrupt priority level.

11.1.214.6 p_callback

`void(* lvd_cfg_t::p_callback) (lvd_callback_args_t *p_args)`

Detailed description

User function to be called from interrupt

11.1.214.7 p_context

`void const* lvd_cfg_t::p_context`

Detailed description

Placeholder for user data. Passed to the user callback in

11.1.214.8 p_extend

```
void const* lvd_cfg_t::p_extend
```

Detailed description

Extension parameter for hardware specific settings

11.1.215 lvd_extend_t

```
typedef struct{
    lvd_negation_delay_t  negation_delay
    lvd_sample_clock_t  sample_clock_divisor
} lvd_extend_t
```

11.1.215.1 negation_delay

```
lvd_negation_delay_t::negation_delay
```

Detailed description

Negation of LVD signal follows reset or voltage in range

11.1.215.2 sample_clock_divisor

```
lvd_sample_clock_t::sample_clock_divisor
```

Detailed description

Sample clock divider, use LVD_SAMPLE_CLOCK_DISABLED to disable digital filtering

11.1.216 lvd_instance_ctrl_t

```
typedef struct{
    uint32_t  monitor_number
    R_SYSTEM_Type *  p_reg
    void(*  p_callback)(lvd_callback_args_t *p_args)
    lvd_callback_args_t  lvd_callback_args
    uint32_t  opened
} lvd_instance_ctrl_t
```

11.1.216.1 monitor_number

```
uint32_t ::monitor_number
```

Detailed description

Monitor number, 1, 2, ...

11.1.216.2 p_reg

```
R_SYSTEM_Type* ::p_reg
```

Brief description

Pointer to LVD register base address.

11.1.216.3 p_callback

```
void(* ::p_callback) ( *p_args)
```

11.1.216.4 lvd_callback_args

```
lvd_callback_args_t::lvd_callback_args
```

11.1.216.5 opened

```
uint32_t ::opened
```

11.1.217 lvd_instance_t

```
typedef struct{
    lvd_ctrl_t *  p_ctrl
    lvd_cfg_t const  *  p_cfg
    lvd_api_t const  *  p_api
} lvd_instance_t
```

11.1.217.1 p_ctrl

```
lvd_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.217.2 p_cfg

```
lvd_cfg_t::p_cfg
```

Brief description

Pointer to the configuration structure for this interface instance.

11.1.217.3 p_api

```
lvd_api_t::p_api
```

Brief description

Pointer to the API structure for this interface instance.

11.1.218 lvd_status_t

```
typedef struct{
    lvd_threshold_crossing_t  crossing_detected
    lvd_current_state_t  current_state
} lvd_status_t
```

11.1.218.1 crossing_detected

lvd_threshold_crossing_t::crossing_detected

Detailed description

Threshold crossing detection (latched)

11.1.218.2 current_state

lvd_current_state_t::current_state

Detailed description

Instantaneous status of monitored voltage (above or below threshold)

11.1.219 opamp_api_t

```
typedef struct{
    ssp_err_t(*  open)(opamp_ctrl_t *const p_ctrl, opamp_cfg_t const *const
p_cfg)
    ssp_err_t(*  start)(opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask)
    ssp_err_t(*  stop)(opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask)
    ssp_err_t(*  trim)(opamp_ctrl_t *const p_ctrl, opamp_trim_cmd_t const cmd,
opamp_trim_args_t const *const p_args)
    ssp_err_t(*  infoGet)(opamp_ctrl_t *const p_ctrl, opamp_info_t *const p_info)
    ssp_err_t(*  statusGet)(opamp_ctrl_t *const p_ctrl, opamp_status_t *const
p_status)
    ssp_err_t(*  close)(opamp_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} opamp_api_t
```

11.1.220 opamp_cfg_t

```
typedef struct{
    void const  *  p_extend
} opamp_cfg_t
```

11.1.220.1 p_extend

```
void const* opamp_cfg_t::p_extend
```

Brief description

Extension parameter for hardware specific settings.

11.1.221 opamp_info_t

```
typedef struct{
    uint32_t  min_stabilization_wait_us
} opamp_info_t
```

11.1.221.1 min_stabilization_wait_us

```
uint32_t opamp_info_t::min_stabilization_wait_us
```

Brief description

Minimum stabilization wait time in microseconds.

11.1.222 opamp_instance_ctrl_t

```
typedef struct{
    R_OPAMP_Type *  p_reg
    uint32_t  opened
    uint8_t  trim_capable
    uint8_t  switches
    uint32_t  valid_opamps
    opamp_priv_trim_state_t  trim_state
    uint8_t  trim_channel
    opamp_trim_input_t  trim_input
} opamp_instance_ctrl_t
```

11.1.222.1 p_reg

```
R_OPAMP_Type* ::p_reg
```

11.1.222.2 opened

```
uint32_t ::opened
```

11.1.222.3 trim_capable

```
uint8_t ::trim_capable
```

11.1.222.4 switches

```
uint8_t ::switches
```

11.1.222.5 valid_opamps

```
uint32_t ::valid_opamps
```

11.1.222.6 trim_state

```
::trim_state
```

11.1.222.7 trim_channel

```
uint8_t ::trim_channel
```

11.1.222.8 trim_input

```
opamp_trim_input_t::trim_input
```

11.1.223 opamp_instance_t

```
typedef struct{
    opamp_ctrl_t *  p_ctrl
    opamp_cfg_t const  *  p_cfg
    opamp_api_t const  *  p_api
} opamp_instance_t
```

11.1.223.1 p_ctrl

```
opamp_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.223.2 p_cfg

opamp_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.223.3 p_api

opamp_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.224 opamp_on_opamp_cfg_t

```
typedef struct{
    opamp_agt_link_t  agt_link
    opamp_mode_t  mode
    opamp_trigger_t  trigger_channel_0
    opamp_trigger_t  trigger_channel_1
    opamp_trigger_t  trigger_channel_2
    opamp_trigger_t  trigger_channel_3
} opamp_on_opamp_cfg_t
```

11.1.224.1 agt_link

opamp_agt_link_t::agt_link

Detailed description

Configure which AGT links are paired to which channel. Only applies to channels if OPAMP_TRIGGER_AGT_START_SOFTWARE_STOP or OPAMP_TRIGGER_AGT_START_ADC_STOP is selected for the channel.

11.1.224.2 mode

opamp_mode_t::mode

Brief description

Low power, middle speed, or high speed mode.

11.1.224.3 trigger_channel_0

opamp_trigger_t::trigger_channel_0

Brief description

Start and stop triggers for channel 0.

11.1.224.4 trigger_channel_1

`opamp_trigger_t::trigger_channel_1`

Brief description

Start and stop triggers for channel 1.

11.1.224.5 trigger_channel_2

`opamp_trigger_t::trigger_channel_2`

Brief description

Start and stop triggers for channel 2.

11.1.224.6 trigger_channel_3

`opamp_trigger_t::trigger_channel_3`

Brief description

Start and stop triggers for channel 3.

11.1.225 opamp_status_t

```
typedef struct{
    uint32_t  operating_channel_mask
} opamp_status_t
```

11.1.225.1 operating_channel_mask

`uint32_t opamp_status_t::operating_channel_mask`

Brief description

Bitmask of channels currently operating.

11.1.226 opamp_trim_args_t

```
typedef struct{
    uint8_t  channel
    opamp_trim_input_t  input
} opamp_trim_args_t
```

11.1.226.1 channel

```
uint8_t opamp_trim_args_t::channel
```

Brief description

Channel.

11.1.226.2 input

```
opamp_trim_input_t::input
```

Brief description

Which input of the channel above.

11.1.227 pdc_api_t

```
typedef struct{
    ssp_err_t(*  open)(pdc_ctrl_t *const p_ctrl, pdc_cfg_t const *const p_cfg)
    ssp_err_t(*  close)(pdc_ctrl_t *const p_ctrl)
    ssp_err_t(*  captureStart)(pdc_ctrl_t *const p_ctrl, uint8_t *const p_buffer)
    ssp_err_t(*  stateGet)(pdc_ctrl_t *const p_ctrl, pdc_state_t *p_state)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_data)
} pdc_api_t
```

11.1.228 pdc_callback_args_t

```
typedef struct{
    pdc_event_t  event
    uint8_t *  p_buffer
    void const  *  p_context
} pdc_callback_args_t
```

11.1.228.1 event

```
pdc_event_t::event
```

Brief description

Event causing the callback.

11.1.228.2 p_buffer

```
uint8_t* pdc_callback_args_t::p_buffer
```

Brief description

Pointer to buffer containing the captured data.

11.1.228.3 p_context

```
void const* pdc_callback_args_t::p_context
```

Brief description

Placeholder for user data. Set in open function in pdc_cfg_t.

11.1.229 pdc_cfg_t

```
typedef struct{
    uint16_t  x_capture_start_pixel
    uint16_t  x_capture_pixels
    uint16_t  y_capture_start_pixel
    uint16_t  y_capture_pixels
    pdc_clock_division_t  clock_division
    pdc_endian_t  endian
    pdc_hsync_polarity_t  hsync_polarity
    pdc_vsync_polarity_t  vsync_polarity
    uint8_t *  p_buffer
    uint8_t  bytes_per_pixel
    uint8_t  frame_end_ipl
    uint8_t  irq_ipl
    transfer_instance_t const  *const  p_lower_lvl_transfer
    void(*  p_callback)(pdc_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} pdc_cfg_t
```

11.1.229.1 x_capture_start_pixel

```
uint16_t pdc_cfg_t::x_capture_start_pixel
```

Brief description

Horizontal position to start capture.

11.1.229.2 x_capture_pixels

```
uint16_t pdc_cfg_t::x_capture_pixels
```

Brief description

Number of horizontal pixels to capture.

11.1.229.3 y_capture_start_pixel

```
uint16_t pdc_cfg_t::y_capture_start_pixel
```

Brief description

Vertical position to start capture.

11.1.229.4 y_capture_pixels

`uint16_t pdc_cfg_t::y_capture_pixels`

Brief description

Number of vertical lines/pixels to capture.

11.1.229.5 clock_division

`pdc_clock_division_t::clock_division`

Brief description

Clock divider.

11.1.229.6 endian

`pdc_endian_t::endian`

Brief description

Endian of capture data.

11.1.229.7 hsync_polarity

`pdc_hsync_polarity_t::hsync_polarity`

Brief description

Polarity of HSYNC input.

11.1.229.8 vsync_polarity

`pdc_vsync_polarity_t::vsync_polarity`

Brief description

Polarity of VSYNC input.

11.1.229.9 p_buffer

`uint8_t* pdc_cfg_t::p_buffer`

Brief description

Pointer to buffer to write image into.

11.1.229.10 bytes_per_pixel

```
uint8_t pdc_cfg_t::bytes_per_pixel
```

Brief description

Number of bytes per pixel.

11.1.229.11 frame_end_ipl

```
uint8_t pdc_cfg_t::frame_end_ipl
```

Brief description

Frame end interrupt priority.

11.1.229.12 irq_ipl

```
uint8_t pdc_cfg_t::irq_ipl
```

Brief description

PDC interrupt priority.

11.1.229.13 p_lower_lvl_transfer

```
transfer_instance_t::p_lower_lvl_transfer
```

Brief description

Pointer to the transfer instance the PDC should use.

11.1.229.14 p_callback

```
void(* pdc_cfg_t::p_callback) (pdc_callback_args_t *p_args)
```

Brief description

Callback provided when a PDC transfer ISR occurs.

11.1.229.15 p_context

```
void const* pdc_cfg_t::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in pdc_callback_args_t.

11.1.229.16 p_extend

```
void const* pdc_cfg_t::p_extend
```

11.1.230 pdc_instance_ctrl_t

```
typedef struct{
    R_PDC_Type *  p_reg
    uint32_t  open
    uint8_t  bytes_per_pixel
    uint16_t  x_resolution_pixels
    uint16_t  y_resolution_pixels
    uint16_t  x_capture_start_pixel
    uint16_t  x_capture_pixels
    uint16_t  y_capture_start_pixel
    uint16_t  y_capture_pixels
    pdc_endian_t  endian
    pdc_hsync_polarity_t  hsync_polarity
    pdc_vsync_polarity_t  vsync_polarity
    uint8_t *  p_current_buffer
    bool  transfer_in_progress
    transfer_instance_t const  *  p_lower_lvl_transfer
    transfer_info_t  info_transfer
    void const  *  p_context
    void(*  p_callback)(pdc_callback_args_t *p_args)
    IRQn_Type  frame_end_irq
    IRQn_Type  irq
} pdc_instance_ctrl_t
```

11.1.230.1 p_reg

```
R_PDC_Type* ::p_reg
```

Brief description

Pointer to PDC base address.

11.1.230.2 open

```
uint32_t ::open
```

Detailed description

Indicates whether or not the driver is open called.

11.1.230.3 bytes_per_pixel

```
uint8_t ::bytes_per_pixel
```

Brief description

Number of bytes per pixel.

11.1.230.4 x_resolution_pixels

```
uint16_t ::x_resolution_pixels
```

Brief description

Total number of horizontal pixels input to PDC.

11.1.230.5 y_resolution_pixels

```
uint16_t ::y_resolution_pixels
```

Brief description

Total number of lines input to the PDC.

11.1.230.6 x_capture_start_pixel

```
uint16_t ::x_capture_start_pixel
```

Brief description

Horizontal position to start capture.

11.1.230.7 x_capture_pixels

```
uint16_t ::x_capture_pixels
```

Brief description

Number of horizontal pixels to capture.

11.1.230.8 y_capture_start_pixel

```
uint16_t ::y_capture_start_pixel
```

Brief description

Vertical position to start capture.

11.1.230.9 y_capture_pixels

```
uint16_t ::y_capture_pixels
```

Brief description

Number of vertical lines/pixels to capture.

11.1.230.10 endian

```
pdc_endian_t::endian
```

Brief description

Endian of capture data.

11.1.230.11 hsync_polarity

`pdc_hsync_polarity_t::hsync_polarity`

Brief description

Polarity of HSYNC input.

11.1.230.12 vsync_polarity

`pdc_vsync_polarity_t::vsync_polarity`

Brief description

Polarity of VSYNC input.

11.1.230.13 p_current_buffer

`uint8_t* ::p_current_buffer`

Brief description

Pointer to buffer currently in use.

11.1.230.14 transfer_in_progress

`bool ::transfer_in_progress`

Brief description

Indicates if a PDC transfer is already in progress.

11.1.230.15 p_lower_lvl_transfer

`transfer_instance_t::p_lower_lvl_transfer`

Brief description

Pointer to the transfer instance the PDC should use.

11.1.230.16 info_transfer

`transfer_info_t::info_transfer`

Brief description

Transfer info structure for low level Transfer interface.

11.1.230.17 p_context

```
void const* ::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in pdc_callback_args_t.

11.1.230.18 p_callback

```
void(* ::p_callback) ( *p_args)
```

Brief description

Callback provided when a PDC transfer ISR occurs.

11.1.230.19 frame_end_irq

```
IRQn_Type ::frame_end_irq
```

Brief description

Frame end interrupt number.

11.1.230.20 irq

```
IRQn_Type ::irq
```

Brief description

PDC interrupt number.

11.1.231 pdc_instance_t

```
typedef struct{
    pdc_ctrl_t *  p_ctrl
    pdc_cfg_t const  *  p_cfg
    pdc_api_t const  *  p_api
} pdc_instance_t
```

11.1.231.1 p_ctrl

```
pdc_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.231.2 p_cfg

pdc_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.231.3 p_api

pdc_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.232 pdc_state_t

```
typedef struct{
    pdc_vsync_state_t  vsync
    pdc_hsync_state_t  hsync
} pdc_state_t
```

11.1.232.1 vsync

pdc_vsync_state_t::vsync

Brief description

VSYNC signal state.

11.1.232.2 hsync

pdc_hsync_state_t::hsync

Brief description

HSYNC signal state.

11.1.233 qspi_api_t

```
typedef struct{
    ssp_err_t(*  open)(qspi_ctrl_t *p_ctrl, qspi_cfg_t const *const p_cfg)
    ssp_err_t(*  close)(qspi_ctrl_t *p_ctrl)
    ssp_err_t(*  read)(qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint8_t
*p_memory_address, uint32_t byte_count)
    ssp_err_t(*  pageProgram)(qspi_ctrl_t *p_ctrl, uint8_t *p_device_address,
uint8_t *p_memory_address, uint32_t byte_count)
    ssp_err_t(*  erase)(qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint32_t
```

```
byte_count)
    ssp_err_t(*  infoGet)(qspi_ctrl_t *const p_ctrl, qspi_info_t *const p_info)
    ssp_err_t(*  sectorErase)(qspi_ctrl_t *p_ctrl, uint8_t *p_device_address)
    ssp_err_t(*  statusGet)(qspi_ctrl_t *p_ctrl, bool *p_write_in_progress)
    ssp_err_t(*  bankSelect)(uint32_t bank)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} qspi_api_t
```

11.1.234 qspi_cfg_t

```
typedef struct{
    void *  p_extend
} qspi_cfg_t
```

11.1.234.1 p_extend

```
void* qspi_cfg_t::p_extend
```

Brief description

place holder for future development

11.1.235 qspi_info_t

```
typedef struct{
    uint32_t  total_size_bytes
    uint32_t  min_program_size_bytes
    uint32_t *  p_erase_sizes_bytes
    uint8_t  num_erase_sizes
} qspi_info_t
```

11.1.235.1 total_size_bytes

```
uint32_t qspi_info_t::total_size_bytes
```

Brief description

Size of this QSPI in bytes.

11.1.235.2 min_program_size_bytes

```
uint32_t qspi_info_t::min_program_size_bytes
```

Brief description

Minimum program size in bytes.

11.1.235.3 p_erase_sizes_bytes

```
uint32_t* qspi_info_t::p_erase_sizes_bytes
```

Brief description

Array of available erase sizes. Each entry is in bytes.

11.1.235.4 num_erase_sizes

```
uint8_t qspi_info_t::num_erase_sizes
```

Brief description

Number of available erase sizes.

11.1.236 qspi_instance_ctrl_t

```
typedef struct{
    R_QSPI_Type *  p_reg
    uint32_t  max_eraseable_size
    uint32_t  num_address_bytes
    uint32_t  spi_mode
    uint32_t  page_size
    uint8_t  manufacturer_id
    uint8_t  memory_type
    uint8_t  memory_capacity
    bool  xip_mode
} qspi_instance_ctrl_t
```

11.1.236.1 p_reg

```
R_QSPI_Type* ::p_reg
```

Brief description

Pointer to QSPI base register.

11.1.236.2 max_eraseable_size

```
uint32_t ::max_eraseable_size
```

Brief description

Largest eraseable sector size in kbytes. Used to determine buffer size for.

11.1.236.3 num_address_bytes

```
uint32_t ::num_address_bytes
```

Brief description

Number of bytes used to represent the address.

11.1.236.4 spi_mode

```
uint32_t ::spi_mode
```

Brief description

SPI mode - 0 = Extended, 1 = Dual, 2 = Quad.

11.1.236.5 page_size

```
uint32_t ::page_size
```

Brief description

Number of bytes in a programmable page.

11.1.236.6 manufacturer_id

```
uint8_t ::manufacturer_id
```

Brief description

Manufacturer ID.

11.1.236.7 memory_type

```
uint8_t ::memory_type
```

Brief description

Memory type.

11.1.236.8 memory_capacity

```
uint8_t ::memory_capacity
```

Brief description

Memory capacity (in MByte)

11.1.236.9 xip_mode

```
bool ::xip_mode
```

Brief description

0 = run in read mode, 1 = run in XIP mode

11.1.237 qspi_instance_t

```
typedef struct{
    qspi_ctrl_t *  p_ctrl
    qspi_cfg_t const  *  p_cfg
    qspi_api_t const  *  p_api
} qspi_instance_t
```

11.1.237.1 p_ctrl

qspi_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.237.2 p_cfg

qspi_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.237.3 p_api

qspi_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.238 recalculated_param_t

```
typedef struct{
    uint16_t  hpix_size
    uint16_t  vpix_size
    int16_t  hpix_offset
    int16_t  vpix_offset
    uint32_t  hread_size
    uint32_t  base_address
} recalculated_param_t
```

11.1.238.1 hpix_size

uint16_t ::hpix_size

11.1.238.2 vpix_size

```
uint16_t ::vpix_size
```

11.1.238.3 hpix_offset

```
int16_t ::hpix_offset
```

11.1.238.4 vpix_offset

```
int16_t ::vpix_offset
```

11.1.238.5 hread_size

```
uint32_t ::hread_size
```

11.1.238.6 base_address

```
uint32_t ::base_address
```

11.1.239 riic_instance_ctrl_t

```
typedef struct{
    i2c_cfg_t  info
    uint32_t  open
    void *  p_reg
    IRQn_Type  rxi_irq
    IRQn_Type  txi_irq
    IRQn_Type  tei_irq
    IRQn_Type  eri_irq
    uint8_t *  p_buff
    uint32_t  total
    uint32_t  remain
    uint32_t  loaded
    uint8_t  addr_low
    uint8_t  addr_high
    uint8_t  addr_total
    uint8_t  addr_remain
    uint8_t  addr_loaded
    bool  read
    bool  restart
    bool  err
    bool  restarted
    bool  transaction_completed
    bool  dummy_read_completed
    bool  activation_on_rxi
    bool  activation_on_txi
```

```
    bool  address_restarted
} riic_instance_ctrl_t
```

11.1.239.1 info

i2c_cfg_t::info

Brief description

Information describing I2C device.

11.1.239.2 open

uint32_t ::open

Brief description

Flag to determine if the device is open.

11.1.239.3 p_reg

void* ::p_reg

Brief description

Base register for this channel.

11.1.239.4 rxi_irq

IRQn_Type ::rxi_irq

Brief description

Receive IRQ number.

11.1.239.5 txi_irq

IRQn_Type ::txi_irq

Brief description

Transmit IRQ number.

11.1.239.6 tei_irq

IRQn_Type ::tei_irq

Brief description

Transmit end IRQ number.

11.1.239.7 eri_irq

```
IRQn_Type ::eri_irq
```

Brief description

Error IRQ number.

11.1.239.8 p_buff

```
uint8_t* ::p_buff
```

Detailed description

Holds the data associated with the transfer

11.1.239.9 total

```
uint32_t ::total
```

Detailed description

Holds the total number of data bytes to transfer

11.1.239.10 remain

```
uint32_t ::remain
```

Detailed description

Tracks the remaining data bytes to transfer

11.1.239.11 loaded

```
uint32_t ::loaded
```

Detailed description

Tracks the number of data bytes written to the register

11.1.239.12 addr_low

```
uint8_t ::addr_low
```

Detailed description

Holds the last address byte to issue

11.1.239.13 addr_high

```
uint8_t ::addr_high
```

Detailed description

Holds the first address byte to issue in 10-bit mode

11.1.239.14 addr_total

```
uint8_t ::addr_total
```

Detailed description

Holds the total number of address bytes to transfer

11.1.239.15 addr_remain

```
uint8_t ::addr_remain
```

Detailed description

Tracks the remaining address bytes to transfer

11.1.239.16 addr_loaded

```
uint8_t ::addr_loaded
```

Detailed description

Tracks the number of address bytes written to the register

11.1.239.17 read

```
volatile bool ::read
```

Detailed description

Holds the direction of the data byte transfer

11.1.239.18 restart

```
volatile bool ::restart
```

Detailed description

Holds whether or not the restart should be issued when done

11.1.239.19 err

```
volatile bool ::err
```

Detailed description

Tracks whether or not an error occurred during processing

11.1.239.20 restarted

```
volatile bool ::restarted
```

Detailed description

Tracks whether or not a restart was issued during the previous transfer

11.1.239.21 transaction_completed

```
volatile bool ::transaction_completed
```

Detailed description

Tracks if the transaction started earlier was completed

11.1.239.22 dummy_read_completed

```
volatile bool ::dummy_read_completed
```

Detailed description

Tracks whether the dummy read is performed

11.1.239.23 activation_on_rxi

```
volatile bool ::activation_on_rxi
```

Detailed description

< Tracks whether the transfer is activated on RXI interrupt

11.1.239.24 activation_on_txi

```
volatile bool ::activation_on_txi
```

Detailed description

< Tracks whether the transfer is activated on TXI interrupt

11.1.239.25 address_restarted

```
volatile bool ::address_restarted
```

Detailed description

< Tracks whether the restart condition is send on 10 bit read

11.1.240 riic_slave_instance_ctrl_t

```
typedef struct{
    i2c_cfg_t  info
    uint32_t  open
    void *  p_reg
    IRQn_Type  rxi_irq
    IRQn_Type  txi_irq
    IRQn_Type  eri_irq
    uint8_t *  p_buff
    uint32_t  total
    uint32_t  remain
    uint32_t  loaded
    uint32_t  transaction_count
    bool  read
    bool  err
    bool  slave_busy
    bool  do_dummy_read
    bool  start_interrupt_enabled
    bool  restarted
} riic_slave_instance_ctrl_t
```

11.1.240.1 info

i2c_cfg_t::info

Brief description

Information describing I2C device.

11.1.240.2 open

uint32_t ::open

Brief description

Flag to determine if the device is open.

11.1.240.3 p_reg

void* ::p_reg

Brief description

Base register for this channel.

11.1.240.4 rxi_irq

IRQn_Type ::rxi_irq

Brief description

Receive IRQ number.

11.1.240.5 txi_irq

```
IRQn_Type ::txi_irq
```

Brief description

Transmit IRQ number.

11.1.240.6 eri_irq

```
IRQn_Type ::eri_irq
```

Brief description

Error IRQ number.

11.1.240.7 p_buff

```
uint8_t* ::p_buff
```

Detailed description

Holds the data associated with the transfer

11.1.240.8 total

```
uint32_t ::total
```

Detailed description

Holds the total number of data bytes to transfer

11.1.240.9 remain

```
uint32_t ::remain
```

Detailed description

Tracks the remaining data bytes to transfer

11.1.240.10 loaded

```
uint32_t ::loaded
```

Detailed description

Tracks the number of data bytes written to the register

11.1.240.11 transaction_count

```
uint32_t ::transaction_count
```

Detailed description

Tracks the actual number of transactions

11.1.240.12 read

```
volatile bool ::read
```

Detailed description

Holds the direction of the data byte transfer

11.1.240.13 err

```
volatile bool ::err
```

Detailed description

Tracks whether or not an error occurred during processing

11.1.240.14 slave_busy

```
volatile bool ::slave_busy
```

Detailed description

Tracks if the slave is busy performing a transaction

11.1.240.15 do_dummy_read

```
volatile bool ::do_dummy_read
```

11.1.240.16 start_interrupt_enabled

```
volatile bool ::start_interrupt_enabled
```

Detailed description

<< Tracks whether a dummy read is issued on the first RX < Tracks whether the start interrupt is enabled

11.1.240.17 restarted

```
volatile bool ::restarted
```

11.1.241 rspi_access_delay_t

```
typedef struct{
    rspi_next_access_delay_count_t  rspi_next_access_delay_count
    rspi_next_access_delay_state_t  rspi_next_access_delay_state
} rspi_access_delay_t
```

11.1.241.1 rspi_next_access_delay_count

rspi_next_access_delay_count_t::rspi_next_access_delay_count

11.1.241.2 rspi_next_access_delay_state

rspi_next_access_delay_state_t::rspi_next_access_delay_state

11.1.242 rspi_clock_delay_t

```
typedef struct{
    rspi_clock_delay_count_t  rspi_clock_delay_count
    rspi_clock_delay_state_t  rspi_clock_delay_state
} rspi_clock_delay_t
```

11.1.242.1 rspi_clock_delay_count

rspi_clock_delay_count_t::rspi_clock_delay_count

11.1.242.2 rspi_clock_delay_state

rspi_clock_delay_state_t::rspi_clock_delay_state

11.1.243 rspi_instance_ctrl_t

```
typedef struct{
    uint8_t  channel
    uint8_t  current_slave
    uint32_t  channel_opened
    transfer_instance_t const  *  p_transfer_tx
    transfer_instance_t const  *  p_transfer_rx
    void(*  p_callback)(spi_callback_args_t *p_args)
    void const  *  p_context
    void *  p_reg
    IRQn_Type  rxi_irq
    IRQn_Type  txi_irq
    IRQn_Type  tei_irq
    IRQn_Type  eri_irq
```

```
    void *  p_src
    void *  p_dest
    uint32_t  tx_count
    uint32_t  rx_count
    uint32_t  xfr_length
    uint8_t  bytes_per_transfer
    bool  do_tx
    bool  using_dtc
    transfer_addr_mode_t  tx_dtc_addr_mode
    transfer_addr_mode_t  rx_dtc_addr_mode
    spi_operation_t  transfer_mode
    uint32_t  rx_data
    bsp_lock_t  resource_lock_tx_rx
} rspi_instance_ctrl_t
```

11.1.243.1 channel

```
uint8_t ::channel
```

Brief description

Channel number to be used.

11.1.243.2 current_slave

```
uint8_t ::current_slave
```

Brief description

Number of the currently assigned slave.

11.1.243.3 channel_opened

```
uint32_t ::channel_opened
```

Brief description

Internal flag to indicate the peripheral was initialized.

11.1.243.4 p_transfer_tx

```
transfer_instance_t::p_transfer_tx
```

Brief description

To use SPI DTC/DMA write transfer.

11.1.243.5 p_transfer_rx

`transfer_instance_t::p_transfer_rx`

Brief description

To use SPI DTC/DMA read transfer.

11.1.243.6 p_callback

```
void(* ::p_callback) ( *p_args)
```

Brief description

Pointer to user callback function.

11.1.243.7 p_context

```
void const* ::p_context
```

Brief description

Pointer to the higher level device context.

11.1.243.8 p_reg

```
void* ::p_reg
```

Brief description

Base register for this channel.

11.1.243.9 rxi_irq

```
IRQn_Type ::rxi_irq
```

Brief description

Receive IRQ number.

11.1.243.10 txi_irq

```
IRQn_Type ::txi_irq
```

Brief description

Transmit IRQ number.

11.1.243.11 tei_irq

```
IRQn_Type ::tei_irq
```

Brief description

Transmit end IRQ number.

11.1.243.12 eri_irq

```
IRQn_Type ::eri_irq
```

Brief description

Error IRQ number.

11.1.243.13 p_src

```
void* ::p_src
```

11.1.243.14 p_dest

```
void* ::p_dest
```

11.1.243.15 tx_count

```
uint32_t ::tx_count
```

11.1.243.16 rx_count

```
uint32_t ::rx_count
```

11.1.243.17 xfr_length

```
uint32_t ::xfr_length
```

11.1.243.18 bytes_per_transfer

```
uint8_t ::bytes_per_transfer
```

11.1.243.19 do_tx

```
bool ::do_tx
```

11.1.243.20 using_dtc

```
bool ::using_dtc
```

11.1.243.21 tx_dtc_addr_mode

transfer_addr_mode_t::tx_dtc_addr_mode

11.1.243.22 rx_dtc_addr_mode

transfer_addr_mode_t::rx_dtc_addr_mode

11.1.243.23 transfer_mode

spi_operation_t::transfer_mode

11.1.243.24 rx_data

uint32_t ::rx_data

11.1.243.25 resource_lock_tx_rx

bsp_lock_t::resource_lock_tx_rx

Detailed description

Resource lock for transmission/reception

11.1.244 rspi_loopback_t

```
typedef struct{
    rspi_loopback1_t  rspi_loopback1
    rspi_loopback2_t  rspi_loopback2
} rspi_loopback_t
```

11.1.244.1 rspi_loopback1

rspi_loopback1_t::rspi_loopback1

11.1.244.2 rspi_loopback2

rspi_loopback2_t::rspi_loopback2

11.1.245 rspi_mosi_idle_t

```
typedef struct{
    rspi_mosi_idle_fixed_val_t  rspi_mosi_idle_fixed_val
    rspi_mosi_idle_val_fixing_t  rspi_mosi_idle_val_fixing
} rspi_mosi_idle_t
```

11.1.245.1 rspi_mosi_idle_fixed_val

rspi_mosi_idle_fixed_val_t::rspi_mosi_idle_fixed_val

11.1.245.2 rspi_mosi_idle_val_fixing

rspi_mosi_idle_val_fixing_t::rspi_mosi_idle_val_fixing

11.1.246 rspi_parity_t

```
typedef struct{
    rspi_parity_state_t  rspi_parity
    rspi_parity_mode_t  rspi_parity_mode
} rspi_parity_t
```

11.1.246.1 rspi_parity

rspi_parity_state_t::rspi_parity

11.1.246.2 rspi_parity_mode

rspi_parity_mode_t::rspi_parity_mode

11.1.247 rspi_ssl_negation_delay_t

```
typedef struct{
    rspi_ssl_negation_delay_count_t  rspi_ssl_neg_delay_count
    rspi_ssl_negation_delay_state_t  rspi_ssl_neg_delay_state
} rspi_ssl_negation_delay_t
```

11.1.247.1 rspi_ssl_neg_delay_count

rspi_ssl_negation_delay_count_t::rspi_ssl_neg_delay_count

11.1.247.2 rspi_ssl_neg_delay_state

rspi_ssl_negation_delay_state_t::rspi_ssl_neg_delay_state

11.1.248 rspi_ssl_polarity_t

```
typedef struct{
    rspi_sslp_t  rspi_ssl2
    rspi_sslp_t  rspi_ssl3
    rspi_sslp_t  rspi_ssl0
```

```
    rspi_sslp_t  rspi_ssl1
} rspi_ssl_polarity_t
```

11.1.248.1 rspi_ssl2

rspi_sslp_t::rspi_ssl2

11.1.248.2 rspi_ssl3

rspi_sslp_t::rspi_ssl3

11.1.248.3 rspi_ssl0

rspi_sslp_t::rspi_ssl0

11.1.248.4 rspi_ssl1

rspi_sslp_t::rspi_ssl1

11.1.249 rtc_alarm_time_t

```
typedef struct{
    rtc_time_t  time
    bool  sec_match
    bool  min_match
    bool  hour_match
    bool  mday_match
    bool  mon_match
    bool  year_match
    bool  dayofweek_match
} rtc_alarm_time_t
```

11.1.249.1 time

rtc_time_t::time

Brief description

Time structure.

11.1.249.2 sec_match

bool rtc_alarm_time_t::sec_match

Brief description

Enable the alarm based on a match of the seconds field.

11.1.249.3 min_match

```
bool rtc_alarm_time_t::min_match
```

Brief description

Enable the alarm based on a match of the minutes field.

11.1.249.4 hour_match

```
bool rtc_alarm_time_t::hour_match
```

Brief description

Enable the alarm based on a match of the hours field.

11.1.249.5 mday_match

```
bool rtc_alarm_time_t::mday_match
```

Brief description

Enable the alarm based on a match of the days field.

11.1.249.6 mon_match

```
bool rtc_alarm_time_t::mon_match
```

Brief description

Enable the alarm based on a match of the months field.

11.1.249.7 year_match

```
bool rtc_alarm_time_t::year_match
```

Brief description

Enable the alarm based on a match of the years field.

11.1.249.8 dayofweek_match

```
bool rtc_alarm_time_t::dayofweek_match
```

Brief description

Enable the alarm based on a match of the dayofweek field.

11.1.250 rtc_api_t

```
typedef struct{
    ssp_err_t(*  open)(rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg)
    ssp_err_t(*  close)(rtc_ctrl_t *const p_ctrl)
    ssp_err_t(*  configure)(rtc_ctrl_t *const p_ctrl, void *const p_extend)
    ssp_err_t(*  calendarTimeSet)(rtc_ctrl_t *const p_ctrl, rtc_time_t *p_time,
bool clock_start)
    ssp_err_t(*  calendarTimeGet)(rtc_ctrl_t *const p_ctrl, rtc_time_t *p_time)
    ssp_err_t(*  calendarAlarmSet)(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t
*p_alarm, bool irq_enable_flag)
    ssp_err_t(*  calendarAlarmGet)(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t
*p_alarm)
    ssp_err_t(*  calendarCounterStart)(rtc_ctrl_t *const p_ctrl)
    ssp_err_t(*  calendarCounterStop)(rtc_ctrl_t *const p_ctrl)
    ssp_err_t(*  irqEnable)(rtc_ctrl_t *const p_ctrl, rtc_event_t irq)
    ssp_err_t(*  irqDisable)(rtc_ctrl_t *const p_ctrl, rtc_event_t irq)
    ssp_err_t(*  periodicIrqRateSet)(rtc_ctrl_t *const p_ctrl,
rtc_periodic_irq_select_t rate)
    ssp_err_t(*  infoGet)(rtc_ctrl_t *p_ctrl, rtc_info_t *p_rtc_info)
    ssp_err_t(*  errorAdjustmentModeSet)(rtc_ctrl_t *p_ctrl,
rtc_error_adjustment_mode_cfg_t *p_error_adjustment_mode)
    ssp_err_t(*  errorAdjustmentSet)(rtc_ctrl_t *p_ctrl,
rtc_error_adjustment_cfg_t *p_error_adjustment_config)
    ssp_err_t(*  versionGet)(ssp_version_t *version)
} rtc_api_t
```

11.1.251 rtc_callback_args_t

```
typedef struct{
    rtc_event_t  event
    void const  *  p_context
} rtc_callback_args_t
```

11.1.251.1 event

rtc_event_t::event

Brief description

The event can be used to identify what caused the callback (compare match or error).

11.1.251.2 p_context

void const* rtc_callback_args_t::p_context

Brief description

Placeholder for user data. Set in r_timer_t::open function in timer_cfg_t.

11.1.252 rtc_cfg_t

```
typedef struct{
    rtc_clock_source_t  clock_source
    bool  hw_cfg
    uint32_t  error_adjustment_value
    rtc_error_adjustment_t  error_adjustment_type
    uint8_t  alarm_ipl
    uint8_t  periodic_ipl
    uint8_t  carry_ipl
    void(*  p_callback)(rtc_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} rtc_cfg_t
```

11.1.252.1 clock_source

`rtc_clock_source_t::clock_source`

Brief description

Clock source for the RTC block.

11.1.252.2 hw_cfg

`bool rtc_cfg_t::hw_cfg`

Brief description

Initialize RTC in Open()

11.1.252.3 error_adjustment_value

`uint32_t rtc_cfg_t::error_adjustment_value`

Brief description

Value of the prescaler for error adjustment.

11.1.252.4 error_adjustment_type

`rtc_error_adjustment_t::error_adjustment_type`

Brief description

How the prescaler value is applied.

11.1.252.5 alarm_ipl

```
uint8_t rtc_cfg_t::alarm_ipl
```

Brief description

Alarm interrupt priority.

11.1.252.6 periodic_ipl

```
uint8_t rtc_cfg_t::periodic_ipl
```

Brief description

Periodic interrupt priority.

11.1.252.7 carry_ipl

```
uint8_t rtc_cfg_t::carry_ipl
```

Brief description

Carry interrupt priority.

11.1.252.8 p_callback

```
void(* rtc_cfg_t::p_callback) (rtc_callback_args_t *p_args)
```

Brief description

Called from the ISR.

11.1.252.9 p_context

```
void const* rtc_cfg_t::p_context
```

Brief description

Passed to the callback.

11.1.252.10 p_extend

```
void const* rtc_cfg_t::p_extend
```

Brief description

RTC hardware dependant configuration.

11.1.253 rtc_error_adjustment_cfg_t

```
typedef struct{
    rtc_error_adjustment_t  adjustment_type
    uint8_t  adjustment_value
} rtc_error_adjustment_cfg_t
```

11.1.253.1 adjustment_type

rtc_error_adjustment_t::adjustment_type

Brief description

Time error adjustment type setting.

11.1.253.2 adjustment_value

uint8_t rtc_error_adjustment_cfg_t::adjustment_value

Brief description

Time error adjustment value.

11.1.254 rtc_error_adjustment_mode_cfg_t

```
typedef struct{
    rtc_error_adjustment_mode_t  adjustment_mode
    rtc_error_adjustment_period_t  adjustment_period
} rtc_error_adjustment_mode_cfg_t
```

11.1.254.1 adjustment_mode

rtc_error_adjustment_mode_t::adjustment_mode

Brief description

Time error adjustment mode setting.

11.1.254.2 adjustment_period

rtc_error_adjustment_period_t::adjustment_period

Brief description

Time error adjustment period setting.

11.1.255 rtc_info_t

```
typedef struct{
    rtc_clock_source_t  clock_source
    rtc_status_t  status
} rtc_info_t
```

11.1.255.1 clock_source

rtc_clock_source_t::clock_source

Brief description

Clock source for the RTC block.

11.1.255.2 status

rtc_status_t::status

Brief description

RTC run status.

11.1.256 rtc_instance_ctrl_t

```
typedef struct{
    void *  p_reg
    void(*  p_callback)(rtc_callback_args_t *cb_data)
    void const  *  p_context
    uint32_t  open
    IRQn_Type  alarm_irq
    IRQn_Type  periodic_irq
    IRQn_Type  carry_irq
    rtc_clock_source_t  clock_source
} rtc_instance_ctrl_t
```

11.1.256.1 p_reg

void* ::p_reg

Brief description

Pointer to register base address.

11.1.256.2 p_callback

void(* ::p_callback) (*cb_data)

Brief description

Called from the ISR.

11.1.256.3 p_context

```
void const* ::p_context
```

Brief description

Passed to the callback.

11.1.256.4 open

```
uint32_t ::open
```

Brief description

Whether or not driver is open.

11.1.256.5 alarm_irq

```
IRQn_Type ::alarm_irq
```

Brief description

Alarm IRQ number.

11.1.256.6 periodic_irq

```
IRQn_Type ::periodic_irq
```

Brief description

Periodic IRQ number.

11.1.256.7 carry_irq

```
IRQn_Type ::carry_irq
```

Brief description

Carry IRQ number.

11.1.256.8 clock_source

```
rtc_clock_source_t::clock_source
```

Brief description

Clock source for the RTC block.

11.1.257 rtc_instance_t

```
typedef struct{
    rtc_ctrl_t *  p_ctrl
    rtc_cfg_t const  *  p_cfg
    rtc_api_t const  *  p_api
} rtc_instance_t
```

11.1.257.1 p_ctrl

rtc_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.257.2 p_cfg

rtc_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.257.3 p_api

rtc_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.258 sb_ble_prf_ias_set_alert_t

```
typedef struct{
    sf_ble_prf_ias_svc_code_t  svc_code
    sf_ble_prf_ias_alert_type_t  lvl
} sb_ble_prf_ias_set_alert_t
```

11.1.258.1 svc_code

sf_ble_prf_ias_svc_code_t::svc_code

Brief description

SVC code.

11.1.258.2 lvl

sf_ble_prf_ias_alert_type_t::lvl

Brief description

Alert level.

11.1.259 sci_i2c_extended_cfg

```
typedef struct{
    bool  bitrate_modulation
} sci_i2c_extended_cfg
```

11.1.259.1 bitrate_modulation

bool sci_i2c_extended_cfg::bitrate_modulation

Detailed description

Bitrate Modulation Function enable or disable

11.1.260 sci_i2c_instance_ctrl_t

```
typedef struct{
    i2c_cfg_t  info
    uint32_t  open
    void *  p_reg
    transfer_instance_t const  *  p_transfer_tx
    transfer_instance_t const  *  p_transfer_rx
    IRQn_Type  rxi_irq
    IRQn_Type  txi_irq
    IRQn_Type  tei_irq
    uint8_t *  p_buff
    uint32_t  total
    uint32_t  remain
    uint32_t  loaded
    uint8_t  addr_low
    uint8_t  addr_high
    uint8_t  addr_total
    uint8_t  addr_remain
    uint8_t  addr_loaded
    bool  read
    bool  restart
    bool  err
    bool  restarted
    bool  transaction_completed
    bool  do_dummy_read
```

```
    bool  activation_on_rxi
    bool  activation_on_txi
} sci_i2c_instance_ctrl_t
```

11.1.260.1 info

i2c_cfg_t::info

Brief description

Information describing I2C device.

11.1.260.2 open

uint32_t ::open

Brief description

Flag to determine if the device is open.

11.1.260.3 p_reg

void* ::p_reg

Brief description

Base register for this channel.

11.1.260.4 p_transfer_tx

transfer_instance_t::p_transfer_tx

Brief description

DTC instance for I2C transmit.Set to NULL if unused.

Detailed description

DTC/DMA support

11.1.260.5 p_transfer_rx

transfer_instance_t::p_transfer_rx

Brief description

DTC instance for I2C receive. Set to NULL if unused.

11.1.260.6 rxi_irq

```
IRQn_Type ::rxi_irq
```

Brief description

Receive IRQ number.

11.1.260.7 txi_irq

```
IRQn_Type ::txi_irq
```

Brief description

Transmit IRQ number.

11.1.260.8 tei_irq

```
IRQn_Type ::tei_irq
```

Brief description

Transmit end IRQ number.

11.1.260.9 p_buff

```
uint8_t* ::p_buff
```

Detailed description

Holds the data associated with the transfer

11.1.260.10 total

```
uint32_t ::total
```

Detailed description

Holds the total number of data bytes to transfer

11.1.260.11 remain

```
uint32_t ::remain
```

Detailed description

Tracks the remaining data bytes to transfer

11.1.260.12 loaded

```
uint32_t ::loaded
```

Detailed description

Tracks the number of data bytes written to the register

11.1.260.13 addr_low

```
uint8_t ::addr_low
```

Detailed description

Holds the last address byte to issue

11.1.260.14 addr_high

```
uint8_t ::addr_high
```

Detailed description

Holds the first address byte to issue in 10-bit mode

11.1.260.15 addr_total

```
uint8_t ::addr_total
```

Detailed description

Holds the total number of address bytes to transfer

11.1.260.16 addr_remain

```
uint8_t ::addr_remain
```

Detailed description

Tracks the remaining address bytes to transfer

11.1.260.17 addr_loaded

```
uint8_t ::addr_loaded
```

Detailed description

Tracks the number of address bytes written to the register

11.1.260.18 read

```
volatile bool ::read
```

Detailed description

Holds the direction of the data byte transfer

11.1.260.19 restart

```
volatile bool ::restart
```

Detailed description

Holds whether or not the restart should be issued when done

11.1.260.20 err

```
volatile bool ::err
```

Detailed description

Tracks whether or not an error occurred during processing

11.1.260.21 restarted

```
volatile bool ::restarted
```

Detailed description

Tracks whether or not a restart was issued during the previous transfer

11.1.260.22 transaction_completed

```
volatile bool ::transaction_completed
```

Detailed description

Tracks if the transaction started earlier was completed

11.1.260.23 do_dummy_read

```
volatile bool ::do_dummy_read
```

11.1.260.24 activation_on_rxi

```
volatile bool ::activation_on_rxi
```

Detailed description

<< Tracks whether a dummy read is issued on the first RX < Tracks whether the transfer is activated on RXI interrupt

11.1.260.25 activation_on_txi

```
volatile bool ::activation_on_txi
```

Detailed description

< Tracks whether the transfer is activated on TXI interrupt

11.1.261 sci_spi_extended_cfg

```
typedef struct{
    bool  bitrate_modulation
} sci_spi_extended_cfg
```

11.1.261.1 bitrate_modulation

```
bool sci_spi_extended_cfg::bitrate_modulation
```

Detailed description

Bitrate Modulation Function enable or disable

11.1.262 sci_spi_instance_ctrl_t

```
typedef struct{
    uint8_t  channel
    uint32_t  channel_opened
    transfer_instance_t const  *  p_transfer_tx
    transfer_instance_t const  *  p_transfer_rx
    void(*  p_callback)(spi_callback_args_t *p_args)
    void const  *  p_context
    void *  p_reg
    IRQn_Type  rxi_irq
    IRQn_Type  txi_irq
    IRQn_Type  tei_irq
    IRQn_Type  eri_irq
    void *  p_src
    void *  p_dest
    uint32_t  tx_count
    uint32_t  rx_count
    uint32_t  xfr_length
    transfer_addr_mode_t  tx_dtc_addr_mode
    uint8_t  bytes_per_transfer
    bool  do_tx
    spi_operation_t  transfer_mode
    bsp_lock_t  resource_lock_tx_rx
} sci_spi_instance_ctrl_t
```

11.1.262.1 channel

```
uint8_t ::channel
```

Brief description

Channel number to be used.

11.1.262.2 channel_opened

```
uint32_t ::channel_opened
```

Brief description

Internal flag to indicate the peripheral was initialized.

11.1.262.3 p_transfer_tx

```
transfer_instance_t::p_transfer_tx
```

Brief description

To use SPI DTC/DMA write transfer.

11.1.262.4 p_transfer_rx

```
transfer_instance_t::p_transfer_rx
```

Brief description

To use SPI DTC/DMA read transfer.

11.1.262.5 p_callback

```
void(* ::p_callback) ( *p_args)
```

Brief description

Pointer to user callback function.

11.1.262.6 p_context

```
void const* ::p_context
```

Brief description

Pointer to the higher level device context.

11.1.262.7 p_reg

```
void* ::p_reg
```

Brief description

Base register for this channel.

11.1.262.8 rxi_irq

```
IRQn_Type ::rxi_irq
```

Brief description

Receive IRQ number.

11.1.262.9 txi_irq

```
IRQn_Type ::txi_irq
```

Brief description

Transmit IRQ number.

11.1.262.10 tei_irq

```
IRQn_Type ::tei_irq
```

Brief description

Transmit end IRQ number.

11.1.262.11 eri_irq

```
IRQn_Type ::eri_irq
```

Brief description

Error IRQ number.

11.1.262.12 p_src

```
void* ::p_src
```

11.1.262.13 p_dest

```
void* ::p_dest
```

11.1.262.14 tx_count

```
uint32_t ::tx_count
```

11.1.262.15 rx_count

```
uint32_t ::rx_count
```

11.1.262.16 xfr_length

```
uint32_t ::xfr_length
```

11.1.262.17 tx_dtc_addr_mode

transfer_addr_mode_t::tx_dtc_addr_mode

11.1.262.18 bytes_per_transfer

uint8_t ::bytes_per_transfer

11.1.262.19 do_tx

bool ::do_tx

11.1.262.20 transfer_mode

spi_operation_t::transfer_mode

11.1.262.21 resource_lock_tx_rx

bsp_lock_t::resource_lock_tx_rx

Detailed description

Resource lock for transmission/reception

11.1.263 sci_uart_instance_ctrl_t

```
typedef struct{
    uint8_t  channel
    uint8_t  fifo_depth
    uint8_t  rx_transfer_in_progress
    uint8_t  data_bytes
    uint8_t  bitrate_modulation
    uint32_t  open
    transfer_instance_t const  *  p_transfer_rx
    transfer_instance_t const  *  p_transfer_tx
    uint8_t const  *  p_tx_src
    uint32_t  tx_src_bytes
    IRQn_Type  rxi_irq
    IRQn_Type  txi_irq
    IRQn_Type  tei_irq
    IRQn_Type  eri_irq
    void(*  p_callback)(uart_callback_args_t *p_args)
    void const  *  p_context
    void *  p_reg
    void(*  p_extpin_ctrl)(uint32_t channel, uint32_t level)
} sci_uart_instance_ctrl_t
```

11.1.263.1 channel

```
uint8_t ::channel
```

Brief description

Channel number.

11.1.263.2 fifo_depth

```
uint8_t ::fifo_depth
```

Brief description

FIFO depth of the UART channel.

11.1.263.3 rx_transfer_in_progress

```
uint8_t ::rx_transfer_in_progress
```

Brief description

Set to 1 if a receive transfer is in progress, 0 otherwise.

11.1.263.4 data_bytes

```
uint8_t ::data_bytes
```

Brief description

1 byte for 7 or 8 bit data, 2 bytes for 9 bit data

11.1.263.5 bitrate_modulation

```
uint8_t ::bitrate_modulation
```

Brief description

1 if bit rate modulation is enabled, 0 otherwise

11.1.263.6 open

```
uint32_t ::open
```

Brief description

Used to determine if the channel is configured.

11.1.263.7 p_transfer_rx

```
transfer_instance_t::p_transfer_rx
```

Detailed description

Optional transfer instance used to send or receive multiple bytes without interrupts.

11.1.263.8 p_transfer_tx

`transfer_instance_t::p_transfer_tx`

Detailed description

Optional transfer instance used to send or receive multiple bytes without interrupts.

11.1.263.9 p_tx_src

```
uint8_t const* ::p_tx_src
```

Detailed description

Source buffer pointer used to fill hardware FIFO from transmit ISR.

11.1.263.10 tx_src_bytes

```
uint32_t ::tx_src_bytes
```

Detailed description

Size of source buffer pointer used to fill hardware FIFO from transmit ISR.

11.1.263.11 rxi_irq

```
IRQn_Type ::rxi_irq
```

Brief description

Receive IRQ number.

11.1.263.12 txi_irq

```
IRQn_Type ::txi_irq
```

Brief description

Transmit IRQ number.

11.1.263.13 tei_irq

```
IRQn_Type ::tei_irq
```

Brief description

Transmit end IRQ number.

11.1.263.14 eri_irq

```
IRQn_Type ::eri_irq
```

Brief description

Error IRQ number.

11.1.263.15 p_callback

```
void(* ::p_callback) ( *p_args)
```

Brief description

Pointer to callback function.

11.1.263.16 p_context

```
void const* ::p_context
```

Brief description

Pointer to user interrupt context data.

11.1.263.17 p_reg

```
void* ::p_reg
```

Brief description

Base register for this channel.

11.1.263.18 p_extpin_ctrl

```
void(* ::p_extpin_ctrl) (uint32_t channel, uint32_t level)
```

Brief description

External pin control.

11.1.264 sdadc_calibrate_args_t

```
typedef struct{
    adc_register_t  channel
    sdadc_calibration_t  mode
} sdadc_calibrate_args_t
```

11.1.264.1 channel

```
adc_register_t::channel
```

Brief description

Which channel to calibrate.

11.1.264.2 mode

sdadc_calibration_t::mode

Brief description

Calibration mode.

11.1.265 sdadc_channel_cfg_t

```
typedef struct{
    sdadc_channel_stage_2_gain_t  stage_2_gain
    sdadc_channel_stage_1_gain_t  stage_1_gain
    sdadc_channel_oversampling_t  oversampling
    uint32_t  __pad0__
    sdadc_channel_polarity_t  polarity
    sdadc_channel_input_t  input
    uint32_t  coefficient_m
    uint32_t  coefficient_n
    sdadc_channel_average_t  average
    sdadc_channel_inversion_t  invert
    uint32_t  __pad1__
    sdadc_channel_count_formula_t  count_formula
} sdadc_channel_cfg_t
```

11.1.265.1 stage_2_gain

sdadc_channel_stage_2_gain_t::stage_2_gain

Brief description

Gain of PGA stage 2, must be 1 for single-ended input.

11.1.265.2 stage_1_gain

sdadc_channel_stage_1_gain_t::stage_1_gain

Brief description

Gain of PGA stage 1, must be 1 for single-ended input.

11.1.265.3 oversampling

sdadc_channel_oversampling_t::oversampling

Brief description

Oversampling ratio, must be 256 in single-ended input.

11.1.265.4 __pad0__

`uint32_t ::__pad0__`

11.1.265.5 polarity

`sdadc_channel_polarity_t::polarity`

Brief description

Polarity, valid for single-ended mode only.

11.1.265.6 input

`sdadc_channel_input_t::input`

Brief description

Single-ended or differential input.

11.1.265.7 coefficient_m

`uint32_t ::coefficient_m`

Brief description

See sdadc_channel_count_formula_t.

11.1.265.8 coefficient_n

`uint32_t ::coefficient_n`

Brief description

See sdadc_channel_count_formula_t.

11.1.265.9 average

`sdadc_channel_average_t::average`

Brief description

Number of samples to average for each conversion result.

11.1.265.10 invert

sdadc_channel_inversion_t::invert

Brief description

Whether to invert negative single-ended input.

11.1.265.11 __pad1__

uint32_t ::__pad1__

11.1.265.12 count_formula

sdadc_channel_count_formula_t::count_formula

Brief description

Linear or exponential formula used for number of conversions.

11.1.266 sdadc_instance_ctrl_t

```
typedef struct{
    adc_mode_t  mode
    adc_resolution_t  resolution
    adc_alignment_t  alignment
    void const  *  p_context
    R_SDADC0_Type *  p_reg
    void(*  p_callback)(adc_callback_args_t *p_args)
    adc_trigger_t  trigger
    uint32_t  trigger_enabled
    uint32_t  opened
    uint32_t  scan_mask
    uint32_t  scan_cfg_mask
    uint16_t  unit
    uint8_t  calib_status
    IRQn_Type  scan_end_irq
    IRQn_Type  calib_end_irq
    IRQn_Type  conv_end_irq
    uint16_t  results_16[SDADC_MAX_NUM_CHANNELS]
    uint32_t  results_32[SDADC_MAX_NUM_CHANNELS]
    union{}                 results
} sdadc_instance_ctrl_t
```

11.1.266.1 mode

adc_mode_t::mode

11.1.266.2 resolution

adc_resolution_t::resolution

11.1.266.3 alignment

adc_alignment_t::alignment

11.1.266.4 p_context

void const* ::p_context

11.1.266.5 p_reg

R_SDADC0_Type* ::p_reg

11.1.266.6 p_callback

void(* ::p_callback) (*p_args)

11.1.266.7 trigger

adc_trigger_t::trigger

11.1.266.8 trigger_enabled

uint32_t ::trigger_enabled

11.1.266.9 opened

uint32_t ::opened

11.1.266.10 scan_mask

uint32_t ::scan_mask

11.1.266.11 scan_cfg_mask

uint32_t ::scan_cfg_mask

11.1.266.12 unit

uint16_t ::unit

11.1.266.13 calib_status

```
volatile uint8_t ::calib_status
```

11.1.266.14 scan_end_irq

```
IRQn_Type ::scan_end_irq
```

11.1.266.15 calib_end_irq

```
IRQn_Type ::calib_end_irq
```

11.1.266.16 conv_end_irq

```
IRQn_Type ::conv_end_irq
```

11.1.266.17 results_16

```
uint16_t ::results_16[SDADC_MAX_NUM_CHANNELS]
```

11.1.266.18 results_32

```
uint32_t ::results_32[SDADC_MAX_NUM_CHANNELS]
```

11.1.266.19 results

```
See source code for the definition of this member.
```

11.1.267 sdhi_event_t

```
typedef struct{
    uint32_t  word
    struct sdhi_event_t::s_sdhi_event_type  bit
} sdhi_event_t
```

11.1.267.1 word

```
uint32_t ::word
```

11.1.267.2 bit

```
struct ::s_sdhi_event_type ::bit
```

11.1.268 sdhi_int_status_t

```
typedef struct{
    uint32_t  info2_mask
} sdhi_int_status_t
```

11.1.268.1 info2_mask

```
uint32_t ::info2_mask
```

11.1.269 sdhi_sdio_event_t

```
typedef struct{
    uint32_t  word
    struct sdhi_sdio_event_t::s_sdhi_sdio_event_type  bit
} sdhi_sdio_event_t
```

11.1.269.1 word

```
uint32_t ::word
```

11.1.269.2 bit

```
struct ::s_sdhi_sdio_event_type ::bit
```

11.1.270 sdmmc_api_t

```
typedef struct{
    ssp_err_t(*  open)(sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const *const
p_cfg)
    ssp_err_t(*  close)(sdmmc_ctrl_t *const p_ctrl)
    ssp_err_t(*  read)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest,
uint32_t const start_sector, uint32_t const sector_count)
    ssp_err_t(*  write)(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const
p_source, uint32_t const start_sector, uint32_t const sector_count)
    ssp_err_t(*  control)(sdmmc_ctrl_t *const p_ctrl, ssp_command_t const
command, void *p_data)
    ssp_err_t(*  readIo)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data,
uint32_t const function, uint32_t const address)
    ssp_err_t(*  writeIo)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data,
uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const
read_after_write)
    ssp_err_t(*  readIoExt)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest,
uint32_t const function, uint32_t const address, uint32_t *const count,
sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
    ssp_err_t(*  writeIoExt)(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const
```

```
p_source, uint32_t const function, uint32_t const address, uint32_t const count,
sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
    ssp_err_t(*  IoIntEnable)(sdmmc_ctrl_t *const p_ctrl, bool enable)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
    ssp_err_t(*  infoGet)(sdmmc_ctrl_t *const p_ctrl, sdmmc_info_t *const p_info)
    ssp_err_t(*  erase)(sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector,
uint32_t const sector_count)
} sdmmc_api_t
```

11.1.271 sdmmc_callback_args_t

```
typedef struct{
    sdmmc_event_t  event
    void const  *  p_context
} sdmmc_callback_args_t
```

11.1.271.1 event

sdmmc_event_t::event

Brief description

The event can be used to identify what caused the callback.

11.1.271.2 p_context

void const* sdmmc_callback_args_t::p_context

Brief description

Placeholder for user data.

11.1.272 sdmmc_cfg_t

```
typedef struct{
    sdmmc_hw_t  hw
    transfer_instance_t const  *  p_lower_lvl_transfer
    void(*  p_callback)(sdmmc_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
    uint8_t  access_ipl
    uint8_t  sdio_ipl
    uint8_t  card_ipl
    uint8_t  dma_req_ipl
} sdmmc_cfg_t
```

11.1.272.1 hw

sdmmc_hw_t::hw

Brief description

Channel, media type, bus width defined by the hardware.

11.1.272.2 p_lower_lvl_transfer

transfer_instance_t::p_lower_lvl_transfer

Brief description

Transfer instance used to move data with DMA or DTC.

11.1.272.3 p_callback

```
void(* sdmmc_cfg_t::p_callback) (sdmmc_callback_args_t *p_args)
```

Brief description

Pointer to callback function.

11.1.272.4 p_context

```
void const* sdmmc_cfg_t::p_context
```

Brief description

User defined context passed into callback function.

11.1.272.5 p_extend

```
void const* sdmmc_cfg_t::p_extend
```

Brief description

SD/MMC hardware dependent configuration.

11.1.272.6 access_ipl

```
uint8_t sdmmc_cfg_t::access_ipl
```

Brief description

Access interrupt priority.

11.1.272.7 sdio_ipl

```
uint8_t sdmmc_cfg_t::sdio_ipl
```

Brief description

SDIO interrupt priority.

11.1.272.8 card_ipl

```
uint8_t sdmmc_cfg_t::card_ipl
```

Brief description

Card interrupt priority.

11.1.272.9 dma_req_ipl

```
uint8_t sdmmc_cfg_t::dma_req_ipl
```

Brief description

DMA request interrupt priority.

11.1.273 sdmmc_extended_cfg_t

```
typedef struct{
    uint32_t  block_size
    sdmmc_card_detect_t  card_detect
} sdmmc_extended_cfg_t
```

11.1.273.1 block_size

```
uint32_t ::block_size
```

Detailed description

Block size in bytes. Block size must be 512 bytes for SD cards and eMMC devices. Block size can be 1-512 bytes for SDIO.

11.1.273.2 card_detect

```
sdmmc_card_detect_t::card_detect
```

Detailed description

Whether or not card detection is used.

11.1.274 sdmmc_hw_t

```
typedef struct{
    uint8_t  channel
    sdmmc_media_type_t  media_type
```

```
    sdmmc_bus_width_t  bus_width
} sdmmc_hw_t
```

11.1.274.1 channel

```
uint8_t sdmmc_hw_t::channel
```

Brief description

Channel of SD/MMC host interface.

11.1.274.2 media_type

```
sdmmc_media_type_t::media_type
```

Brief description

Embedded or pluggable card.

11.1.274.3 bus_width

```
sdmmc_bus_width_t::bus_width
```

Brief description

Device bus width is 1, 4 or 8 bits wide.

11.1.275 sdmmc_info_t

```
typedef struct{
    sdmmc_card_type_t  card_type
    bool  ready
    bool  hc
    bool  sdio
    bool  write_protected
    bool  transfer_in_progress
    uint8_t  csd_version
    uint8_t  device_type
    sdmmc_bus_width_t  bus_width
    uint8_t  hs_timing
    uint32_t  sdhi_rca
    uint32_t  max_clock_rate
    uint32_t  clock_rate
    uint32_t  sector_size
    uint32_t  sector_count
    uint32_t  erase_sector_count
} sdmmc_info_t
```

11.1.275.1 card_type

sdmmc_card_type_t::card_type

Brief description

SD or eMMC.

11.1.275.2 ready

bool sdmmc_info_t::ready

Detailed description

False if card was removed (only applies if MCU supports card detection and SDnCD pin is connected).

True otherwise.

If ready is false, the driver must be closed, then reopened with a card inserted.

11.1.275.3 hc

bool sdmmc_info_t::hc

Brief description

true = Card is High Capacity card

11.1.275.4 sdio

bool sdmmc_info_t::sdio

Brief description

true = SDIO present

11.1.275.5 write_protected

bool sdmmc_info_t::write_protected

Brief description

true = Card is write protected

11.1.275.6 transfer_in_progress

bool sdmmc_info_t::transfer_in_progress

Brief description

true = Card is busy

11.1.275.7 csd_version

```
uint8_t sdmmc_info_t::csd_version
```

Brief description

CSD version.

11.1.275.8 device_type

```
uint8_t sdmmc_info_t::device_type
```

Brief description

Speed and data rate (eMMC)

11.1.275.9 bus_width

```
sdmmc_bus_width_t::bus_width
```

Brief description

Current media bus width.

11.1.275.10 hs_timing

```
uint8_t sdmmc_info_t::hs_timing
```

Brief description

High Speed status.

11.1.275.11 sdhi_rca

```
uint32_t sdmmc_info_t::sdhi_rca
```

Brief description

Relative Card Address.

11.1.275.12 max_clock_rate

```
uint32_t sdmmc_info_t::max_clock_rate
```

Brief description

Maximum clock rate for media card.

11.1.275.13 clock_rate

```
uint32_t sdmmc_info_t::clock_rate
```

Brief description

Current clock rate.

11.1.275.14 sector_size

```
uint32_t sdmmc_info_t::sector_size
```

Brief description

Sector size.

11.1.275.15 sector_count

```
uint32_t sdmmc_info_t::sector_count
```

Brief description

Sector count.

11.1.275.16 erase_sector_count

```
uint32_t sdmmc_info_t::erase_sector_count
```

Brief description

Minimum erasable unit (in 512 byte sectors)

11.1.276 sdmmc_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    sdmmc_hw_t  hw
    transfer_instance_t const  *  p_lower_lvl_transfer
    sdmmc_info_t  status
    bool  transfer_in_progress
    void(*  p_callback)(sdmmc_callback_args_t *p_args)
    void const  *  p_context
    R_SDHI0_Type *  p_reg
    sdhi_event_t  sdhi_event
    IRQn_Type  transfer_irq
    sdmmc_transfer_dir_t  transfer_dir
    uint8_t *  p_transfer_data
    uint32_t  transfer_blocks_total
    uint32_t  transfer_block_current
    uint32_t  transfer_block_size
    uint32_t  aligned_buff[SDMMC_MAX_BLOCK_SIZE/sizeof(uint32_t)]
} sdmmc_instance_ctrl_t
```

11.1.276.1 open

```
uint32_t ::open
```

11.1.276.2 hw

```
sdmmc_hw_t::hw
```

11.1.276.3 p_lower_lvl_transfer

```
transfer_instance_t::p_lower_lvl_transfer
```

11.1.276.4 status

```
sdmmc_info_t::status
```

11.1.276.5 transfer_in_progress

```
bool ::transfer_in_progress
```

11.1.276.6 p_callback

```
void(* ::p_callback) ( *p_args)
```

11.1.276.7 p_context

```
void const* ::p_context
```

11.1.276.8 p_reg

```
R_SDHI0_Type* ::p_reg
```

11.1.276.9 sdhi_event

```
volatile ::sdhi_event
```

11.1.276.10 transfer_irq

```
IRQn_Type ::transfer_irq
```

11.1.276.11 transfer_dir

```
::transfer_dir
```

11.1.276.12 p_transfer_data

```
uint8_t* ::p_transfer_data
```

11.1.276.13 transfer_blocks_total

```
uint32_t ::transfer_blocks_total
```

11.1.276.14 transfer_block_current

```
uint32_t ::transfer_block_current
```

11.1.276.15 transfer_block_size

```
uint32_t ::transfer_block_size
```

11.1.276.16 aligned_buff

```
uint32_t ::aligned_buff[SDMMC_MAX_BLOCK_SIZE/sizeof(uint32_t)]
```

11.1.277 sdmmc_instance_t

```
typedef struct{
    sdmmc_ctrl_t *  p_ctrl
    sdmmc_cfg_t const  *  p_cfg
    sdmmc_api_t const  *  p_api
} sdmmc_instance_t
```

11.1.277.1 p_ctrl

sdmmc_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.277.2 p_cfg

sdmmc_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.277.3 p_api

sdmmc_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.278 sdmmc_io_mode_t

```
typedef struct{
    sdmmc_io_command_t  command
    sdmmc_io_transfer_mode_t  transfer_mode
    sdmmc_io_address_mode_t  address_mode
    sdmmc_io_write_mode_t  write_mode
} sdmmc_io_mode_t
```

11.1.278.1 command

::command

11.1.278.2 transfer_mode

::transfer_mode

11.1.278.3 address_mode

::address_mode

11.1.278.4 write_mode

::write_mode

11.1.279 sf_adc_periodic_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_adc_periodic_ctrl_t *const p_ctrl,
sf_adc_periodic_cfg_t const *const p_cfg)
    ssp_err_t(*  start)(sf_adc_periodic_ctrl_t *const p_ctrl)
    ssp_err_t(*  stop)(sf_adc_periodic_ctrl_t *const p_ctrl)
    ssp_err_t(*  close)(sf_adc_periodic_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_adc_periodic_api_t
```

11.1.280 sf_adc_periodic_callback_args_t

```
typedef struct{
    sf_adc_periodic_event_t  event
    uint32_t  buffer_index
```

```
    void const  *  p_context
    adc_data_size_t *  p_data_buffer
    uint32_t  num_new_samples
} sf_adc_periodic_callback_args_t
```

11.1.280.1 event

sf_adc_periodic_event_t::event

Brief description

Periodic ADC callback event.

11.1.280.2 buffer_index

uint32_t sf_adc_periodic_callback_args_t::buffer_index

Brief description

Index to the buffer where the new data is stored.

11.1.280.3 p_context

void const* sf_adc_periodic_callback_args_t::p_context

Brief description

Placeholder for user data.

11.1.280.4 p_data_buffer

adc_data_size_t* sf_adc_periodic_callback_args_t::p_data_buffer

Brief description

Pointer to the buffer that will store the samples.

11.1.280.5 num_new_samples

uint32_t sf_adc_periodic_callback_args_t::num_new_samples

Brief description

Number of new samples in the data buffer.

11.1.281 sf_adc_periodic_cfg_t

```
typedef struct{
    adc_instance_t const  *const  p_lower_lvl_adc
```

```
    timer_instance_t const  *const  p_lower_lvl_timer
    transfer_instance_t const  *const  p_lower_lvl_transfer
    adc_data_size_t *  p_data_buffer
    uint32_t  data_buffer_length
    uint32_t  sample_count
    elc_event_t  scan_trigger
    void(*  p_callback)(sf_adc_periodic_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} sf_adc_periodic_cfg_t
```

11.1.281.1 p_lower_lvl_adc

`adc_instance_t::p_lower_lvl_adc`

Brief description

Pointer to the ADC instance.

11.1.281.2 p_lower_lvl_timer

`timer_instance_t::p_lower_lvl_timer`

Brief description

Pointer to the Timer instance.

11.1.281.3 p_lower_lvl_transfer

`transfer_instance_t::p_lower_lvl_transfer`

Brief description

Pointer to the Transfer instance.

11.1.281.4 p_data_buffer

`adc_data_size_t* sf_adc_periodic_cfg_t::p_data_buffer`

Brief description

Pointer to the buffer that will store the samples.

11.1.281.5 data_buffer_length

`uint32_t sf_adc_periodic_cfg_t::data_buffer_length`

Brief description

Length of the data buffer that will store the samples.

11.1.281.6 sample_count

```
uint32_t sf_adc_periodic_cfg_t::sample_count
```

Brief description

Samples per channel to be buffered before notifying the app.

11.1.281.7 scan_trigger

```
elc_event_t::scan_trigger
```

Brief description

The hardware trigger that starts the ADC scan.

11.1.281.8 p_callback

```
void(* sf_adc_periodic_cfg_t::p_callback) (sf_adc_periodic_callback_args_t
*p_args)
```

Brief description

Callback function.

11.1.281.9 p_context

```
void const* sf_adc_periodic_cfg_t::p_context
```

Brief description

Placeholder for user data.

11.1.281.10 p_extend

```
void const* sf_adc_periodic_cfg_t::p_extend
```

Brief description

Extension parameter for hardware specific settings.

11.1.282 sf_adc_periodic_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    TX_MUTEX  mutex
    adc_instance_t const  *  p_lower_lvl_adc
    timer_instance_t const  *  p_lower_lvl_timer
    transfer_instance_t const  *  p_lower_lvl_transfer
    void const  *volatile  p_src_transfer
```

```
    adc_data_size_t *  p_data_buffer
    uint32_t  data_buffer_length
    uint32_t  data_buffer_index
    uint32_t  sample_count
    uint32_t  dtc_transfer_length
    void(*  p_callback)(sf_adc_periodic_callback_args_t *p_args)
    void const  *  p_context
} sf_adc_periodic_instance_ctrl_t
```

11.1.282.1 open

`uint32_t ::open`

Brief description

Used by driver to check if pointer to control block is valid.

11.1.282.2 mutex

`TX_MUTEX ::mutex`

Brief description

Mutex used to protect access to lower level driver hardware registers.

11.1.282.3 p_lower_lvl_adc

`adc_instance_t::p_lower_lvl_adc`

Brief description

Pointer to the ADC instance.

11.1.282.4 p_lower_lvl_timer

`timer_instance_t::p_lower_lvl_timer`

Brief description

Pointer to the Timer instance.

11.1.282.5 p_lower_lvl_transfer

`transfer_instance_t::p_lower_lvl_transfer`

Brief description

Pointer to the Transfer instance.

11.1.282.6 p_src_transfer

```
void const* volatile ::p_src_transfer
```

Brief description

Source pointer for the low level transfer method.

11.1.282.7 p_data_buffer

```
* ::p_data_buffer
```

Brief description

Pointer to the buffer that will store the samples.

11.1.282.8 data_buffer_length

```
uint32_t ::data_buffer_length
```

Brief description

Length of the data buffer that will store the samples.

11.1.282.9 data_buffer_index

```
uint32_t ::data_buffer_index
```

Brief description

Index of the data buffer where data is to be written to next.

11.1.282.10 sample_count

```
uint32_t ::sample_count
```

Brief description

Samples per channel to be buffered before notifying the app.

11.1.282.11 dtc_transfer_length

```
uint32_t ::dtc_transfer_length
```

Brief description

Total Length of DTC transfer for requested number of samples.

11.1.282.12 p_callback

```
void(* ::p_callback) ( *p_args)
```

Brief description

Callback function.

11.1.282.13 p_context

```
void const* ::p_context
```

Brief description

Placeholder for user data.

11.1.283 sf_adc_periodic_instance_t

```
typedef struct{
    sf_adc_periodic_ctrl_t *  p_ctrl
    sf_adc_periodic_cfg_t const  *  p_cfg
    sf_adc_periodic_api_t const  *  p_api
} sf_adc_periodic_instance_t
```

11.1.283.1 p_ctrl

```
sf_adc_periodic_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.283.2 p_cfg

```
sf_adc_periodic_cfg_t::p_cfg
```

Brief description

Pointer to the configuration structure for this instance.

11.1.283.3 p_api

```
sf_adc_periodic_api_t::p_api
```

Brief description

Pointer to the API structure for this instance.

11.1.284 sf_audio_playback_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_audio_playback_ctrl_t *const p_ctrl,
sf_audio_playback_cfg_t const *const p_cfg)
```

```
    ssp_err_t(*  close)(sf_audio_playback_ctrl_t *const p_ctrl)
    ssp_err_t(*  start)(sf_audio_playback_ctrl_t *const p_ctrl,
sf_audio_playback_data_t *const p_data, UINT const timeout)
    ssp_err_t(*  pause)(sf_audio_playback_ctrl_t *const p_ctrl)
    ssp_err_t(*  stop)(sf_audio_playback_ctrl_t *const p_ctrl)
    ssp_err_t(*  resume)(sf_audio_playback_ctrl_t *const p_ctrl)
    ssp_err_t(*  volumeSet)(sf_audio_playback_ctrl_t *const p_ctrl, uint8_t const
volume)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_audio_playback_api_t
```

11.1.285 sf_audio_playback_cfg_t

```
typedef struct{
    void(*  p_callback)(sf_message_callback_args_t *p_args)
    sf_audio_playback_common_ctrl_t *  p_common_ctrl
    sf_audio_playback_common_cfg_t const  *  p_common_cfg
    uint8_t  class_instance
} sf_audio_playback_cfg_t
```

11.1.285.1 p_callback

```
void(* sf_audio_playback_cfg_t::p_callback) (sf_message_callback_args_t *p_args)
```

Detailed description

Callback called when playback of a buffer passed to start is complete. Set to NULL for no callback.

11.1.285.2 p_common_ctrl

```
sf_audio_playback_common_ctrl_t::p_common_ctrl
```

Detailed description

Pointer to the hardware control block used by this stream.

11.1.285.3 p_common_cfg

```
sf_audio_playback_common_cfg_t::p_common_cfg
```

Detailed description

Pointer to common configurations shared by all streams using the same hardware.

11.1.285.4 class_instance

```
uint8_t sf_audio_playback_cfg_t::class_instance
```

Brief description

Class instance used to identify the stream to the messaging framework.

11.1.286 sf_audio_playback_common_cfg_t

```
typedef struct{
    UINT  priority
    sf_audio_playback_hw_instance_t const  *  p_lower_lvl_hw
    sf_message_instance_t const  *  p_message
    TX_QUEUE *  p_queue
    void const  *  p_extend
} sf_audio_playback_common_cfg_t
```

11.1.286.1 priority

```
UINT sf_audio_playback_common_cfg_t::priority
```

Brief description

Priority of the audio playback thread.

11.1.286.2 p_lower_lvl_hw

```
sf_audio_playback_hw_instance_t::p_lower_lvl_hw
```

Brief description

Hardware instance.

11.1.286.3 p_message

```
sf_message_instance_t::p_message
```

Detailed description

Pointer to messaging framework instance used to post audio messages.

11.1.286.4 p_queue

```
TX_QUEUE* sf_audio_playback_common_cfg_t::p_queue
```

Detailed description

Pointer to the messaging framework queue specified for this audio stream. Must be subscribed to the SF_MESSAGE_EVENT_CLASS_AUDIO event class.

11.1.286.5 p_extend

```
void const* sf_audio_playback_common_cfg_t::p_extend
```

Detailed description

Implementation specific extension configuration.

11.1.287 sf_audio_playback_common_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    void const  *  p_next_buffer
    uint32_t  next_length
    sf_message_instance_t const  *  p_message
    TX_QUEUE *  p_queue
    sf_audio_playback_hw_instance_t const  *  p_lower_lvl_hw
    sf_audio_playback_instance_ctrl_t
*  p_stream[SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS]
    TX_THREAD  thread
    TX_EVENT_FLAGS_GROUP  flags
    sf_audio_playback_data_type_t  data_type
    uint8_t  volume
    uint8_t  buffer_index
    uint8_t  stack[SF_AUDIO_PLAYBACK_STACK_SIZE]
    int16_t  samples[2][SF_AUDIO_PLAYBACK_CFG_BUFFER_SIZE_BYTES/sizeof(int16_t)]
    bool  playing
} sf_audio_playback_common_instance_ctrl_t
```

11.1.287.1 open

```
uint32_t ::open
```

Brief description

Used to determine if driver is initialized.

11.1.287.2 p_next_buffer

```
void const* ::p_next_buffer
```

Brief description

Pointer to next buffer (to be played when the current buffer completes).

11.1.287.3 next_length

```
uint32_t ::next_length
```

Brief description

Length of next buffer (to be played when the current buffer completes).

11.1.287.4 p_message

sf_message_instance_t::p_message

Brief description

Pointer to message control block.

11.1.287.5 p_queue

TX_QUEUE* ::p_queue

Brief description

Queue subscribed to SF_MESSAGE_EVENT_CLASS_AUDIO events.

11.1.287.6 p_lower_lvl_hw

sf_audio_playback_hw_instance_t::p_lower_lvl_hw

Brief description

Hardware API's used.

11.1.287.7 p_stream

* ::p_stream[SF_AUDIO_PLAYBACK_CFG_MAX_STREAMS]

Brief description

Stream specific data.

11.1.287.8 thread

TX_THREAD ::thread

Brief description

Main audio thread.

11.1.287.9 flags

TX_EVENT_FLAGS_GROUP ::flags

Brief description

Event flags used to end wait in audio thread.

11.1.287.10 data_type

sf_audio_playback_data_type_t::data_type

Brief description

Sample format required by the hardware.

11.1.287.11 volume

uint8_t ::volume

Brief description

Volume from 0 (muted) to 255 (maximum, default on open).

11.1.287.12 buffer_index

uint8_t ::buffer_index

Brief description

Which ping pong buffer to use.

11.1.287.13 stack

uint8_t ::stack[SF_AUDIO_PLAYBACK_STACK_SIZE]

Brief description

Stack for audio thread.

11.1.287.14 samples

int16_t ::samples[2][SF_AUDIO_PLAYBACK_CFG_BUFFER_SIZE_BYTES/sizeof(int16_t)]

Detailed description

Ping pong buffers, used to store converted data during transfer.

11.1.287.15 playing

bool ::playing

Brief description

State of audio instance (currently playing if true)

11.1.288 sf_audio_playback_data_t

```
typedef struct{
    sf_message_header_t  header
    sf_audio_playback_data_type_t  type
    uint32_t  size_bytes
    void const  *  p_data
    UINT  loop_timeout
    bool  stream_end
} sf_audio_playback_data_t
```

11.1.288.1 header

sf_message_header_t::header

Brief description

Required common members of messaging framework payloads.

11.1.288.2 type

sf_audio_playback_data_type_t::type

Brief description

Data type. Must be uncompressed.

11.1.288.3 size_bytes

uint32_t sf_audio_playback_data_t::size_bytes

Brief description

Size of data in bytes.

11.1.288.4 p_data

void const* sf_audio_playback_data_t::p_data

Brief description

Pointer to data. Data start address must be 4-byte aligned.

11.1.288.5 loop_timeout

UINT sf_audio_playback_data_t::loop_timeout

Detailed description

ThreadX timeout, select TX_NO_WAIT to play the entire buffer once, TX_WAIT_FOREVER to loop until SF_AUDIO_PLAYBACK_Pause is called from another thread, or any timeout value from (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts to loop until the tick counts expire.

11.1.288.6 stream_end

```
bool sf_audio_playback_data_t::stream_end
```

Detailed description

This releases ownership of the stream and allows other threads to post data. Set to true if not more data will be sent as a part of this logical bitstream. Set to false if more packets are being prepared.

11.1.289 sf_audio_playback_data_type_t

```
typedef struct{
    uint8_t  scale_bits_max
    bool  is_signed
} sf_audio_playback_data_type_t
```

11.1.289.1 scale_bits_max

```
uint8_t sf_audio_playback_data_type_t::scale_bits_max
```

Brief description

Maximum data resolution in bits.

11.1.289.2 is_signed

```
bool sf_audio_playback_data_type_t::is_signed
```

Brief description

Set to 1 for signed samples, or 0 for unsigned samples.

11.1.290 sf_audio_playback_hw_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_audio_playback_hw_ctrl_t *const p_ctrl,
sf_audio_playback_hw_cfg_t const *const p_cfg)
    ssp_err_t(*  start)(sf_audio_playback_hw_ctrl_t *const p_ctrl)
    ssp_err_t(*  stop)(sf_audio_playback_hw_ctrl_t *const p_ctrl)
    ssp_err_t(*  play)(sf_audio_playback_hw_ctrl_t *const p_ctrl, int16_t const
*const p_buffer, uint32_t length)
    ssp_err_t(*  dataTypeGet)(sf_audio_playback_hw_ctrl_t *const p_ctrl,
sf_audio_playback_data_type_t *const p_data_type)
```

```
    ssp_err_t(*  close)(sf_audio_playback_hw_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_audio_playback_hw_api_t
```

11.1.291 sf_audio_playback_hw_callback_args_t

```
typedef struct{
    void *  p_context
} sf_audio_playback_hw_callback_args_t
```

11.1.291.1 p_context

```
void* sf_audio_playback_hw_callback_args_t::p_context
```

Detailed description

Placeholder for user data. Set in open function in sf_audio_playback_hw_cfg_t.

11.1.292 sf_audio_playback_hw_cfg_t

```
typedef struct{
    void(*  p_callback)(sf_audio_playback_hw_callback_args_t *p_args)
    void *  p_context
    void const  *  p_extend
} sf_audio_playback_hw_cfg_t
```

11.1.292.1 p_callback

```
void(* sf_audio_playback_hw_cfg_t::p_callback)
(sf_audio_playback_hw_callback_args_t *p_args)
```

Detailed description

Callback called when play is complete. Set to NULL for no callback.

11.1.292.2 p_context

```
void* sf_audio_playback_hw_cfg_t::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in sf_audio_playback_hw_callback_args_t.

11.1.292.3 p_extend

```
void const* sf_audio_playback_hw_cfg_t::p_extend
```

Brief description

Hardware dependent configuration.

11.1.293 sf_audio_playback_hw_dac_cfg_t

```
typedef struct{
    dac_instance_t const  *  p_lower_lvl_dac
    timer_instance_t const  *  p_lower_lvl_timer
    transfer_instance_t const  *  p_lower_lvl_transfer
} sf_audio_playback_hw_dac_cfg_t
```

11.1.293.1 p_lower_lvl_dac

dac_instance_t::p_lower_lvl_dac

Brief description

DAC API used to access DAC hardware.

11.1.293.2 p_lower_lvl_timer

timer_instance_t::p_lower_lvl_timer

Brief description

Timer API used to generate sampling frequency.

11.1.293.3 p_lower_lvl_transfer

transfer_instance_t::p_lower_lvl_transfer

Brief description

Transfer API used to transfer data each sampling frequency.

11.1.294 sf_audio_playback_hw_dac_instance_ctrl_t

```
typedef struct{
    void(*  p_callback)(sf_audio_playback_hw_callback_args_t *p_args)
    void *  p_context
    dac_instance_t const  *  p_lower_lvl_dac
    timer_instance_t const  *  p_lower_lvl_timer
    transfer_instance_t const  *  p_lower_lvl_transfer
} sf_audio_playback_hw_dac_instance_ctrl_t
```

11.1.294.1 p_callback

```
void(* sf_::p_callback) ( *p_args)
```

Detailed description

Callback called when play is complete.

11.1.294.2 p_context

```
void* sf_::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in sf_audio_playback_hw_callback_args_t.

11.1.294.3 p_lower_lvl_dac

```
dac_instance_t::p_lower_lvl_dac
```

Brief description

DAC API used to access DAC hardware.

11.1.294.4 p_lower_lvl_timer

```
timer_instance_t::p_lower_lvl_timer
```

Brief description

Timer API used to generate sampling frequency.

11.1.294.5 p_lower_lvl_transfer

```
transfer_instance_t::p_lower_lvl_transfer
```

Brief description

Transfer API used to transfer data each sampling frequency.

11.1.295 sf_audio_playback_hw_i2s_cfg_t

```
typedef struct{
    i2s_instance_t const  *  p_lower_lvl_i2s
} sf_audio_playback_hw_i2s_cfg_t
```

11.1.295.1 p_lower_lvl_i2s

```
i2s_instance_t::p_lower_lvl_i2s
```

Brief description

I2S API used to access I2S hardware.

11.1.296 sf_audio_playback_hw_i2s_instance_ctrl_t

```
typedef struct{
    void(*  p_callback)(sf_audio_playback_hw_callback_args_t *p_args)
    void *  p_context
    i2s_instance_t const  *  p_lower_lvl_i2s
} sf_audio_playback_hw_i2s_instance_ctrl_t
```

11.1.296.1 p_callback

```
void(* sf_::p_callback) ( *p_args)
```

Detailed description

Callback called when play is complete.

11.1.296.2 p_context

```
void* sf_::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in sf_audio_playback_hw_callback_args_t.

11.1.296.3 p_lower_lvl_i2s

```
i2s_instance_t::p_lower_lvl_i2s
```

Brief description

I2S API used to access I2S hardware.

11.1.297 sf_audio_playback_hw_instance_t

```
typedef struct{
    sf_audio_playback_hw_ctrl_t *  p_ctrl
    sf_audio_playback_hw_cfg_t const  *  p_cfg
    sf_audio_playback_hw_api_t const  *  p_api
} sf_audio_playback_hw_instance_t
```

11.1.297.1 p_ctrl

```
sf_audio_playback_hw_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.297.2 p_cfg

sf_audio_playback_hw_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.297.3 p_api

sf_audio_playback_hw_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.298 sf_audio_playback_instance_t

```
typedef struct{
    sf_audio_playback_ctrl_t *  p_ctrl
    sf_audio_playback_cfg_t const  *  p_cfg
    sf_audio_playback_api_t const  *  p_api
} sf_audio_playback_instance_t
```

11.1.298.1 p_ctrl

sf_audio_playback_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.298.2 p_cfg

sf_audio_playback_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.298.3 p_api

sf_audio_playback_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.299 sf_audio_record_adc_hw_cfg_t

```
typedef struct{
    sf_adc_periodic_instance_t const  *  p_lower_lvl_adc_periodic
} sf_audio_record_adc_hw_cfg_t
```

11.1.299.1 p_lower_lvl_adc_periodic

sf_adc_periodic_instance_t::p_lower_lvl_adc_periodic

11.1.300 sf_audio_record_adc_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    TX_MUTEX  mutex
    void *  p_capture_data_buffer
    uint32_t  sample_count
    void(*  p_callback)(sf_audio_record_callback_args_t *p_args)
    void const  *  p_context
    sf_adc_periodic_instance_t const  *  p_lower_lvl_adc_periodic
} sf_audio_record_adc_instance_ctrl_t
```

11.1.300.1 open

uint32_t ::open

Detailed description

Used by driver to check if pointer to control block is valid

11.1.300.2 mutex

TX_MUTEX ::mutex

Detailed description

Mutex used to protect access to lower level driver hardware registers

11.1.300.3 p_capture_data_buffer

void* ::p_capture_data_buffer

Detailed description

Pointer to the buffer that will store the samples

11.1.300.4 sample_count

```
uint32_t ::sample_count
```

Detailed description

Samples per channel to be buffered before notifying the app

11.1.300.5 p_callback

```
void(* ::p_callback) ( *p_args)
```

Brief description

Callback function.

11.1.300.6 p_context

```
void const* ::p_context
```

Brief description

Placeholder for user data.

11.1.300.7 p_lower_lvl_adc_periodic

```
sf_adc_periodic_instance_t::p_lower_lvl_adc_periodic
```

Brief description

Lower level ADC periodic instance.

11.1.301 sf_audio_record_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_audio_record_ctrl_t *const p_ctrl,
sf_audio_record_cfg_t const *const p_cfg)
    ssp_err_t(*  start)(sf_audio_record_ctrl_t *const p_ctrl)
    ssp_err_t(*  stop)(sf_audio_record_ctrl_t *const p_ctrl)
    ssp_err_t(*  infoGet)(sf_audio_record_ctrl_t *const p_ctrl,
sf_audio_record_info_t *p_info)
    ssp_err_t(*  close)(sf_audio_record_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_audio_record_api_t
```

11.1.302 sf_audio_record_callback_args_t

```
typedef struct{
    sf_audio_record_event_t  event
    uint32_t  buffer_index
    void const  *  p_context
} sf_audio_record_callback_args_t
```

11.1.302.1 event

sf_audio_record_event_t::event

Brief description

Audio callback event.

11.1.302.2 buffer_index

uint32_t sf_audio_record_callback_args_t::buffer_index

Brief description

Index to the buffer where the new data is stored.

11.1.302.3 p_context

void const* sf_audio_record_callback_args_t::p_context

Brief description

Placeholder for user data.

11.1.303 sf_audio_record_cfg_t

```
typedef struct{
    sf_audio_record_data_size_t  capture_data_size
    uint32_t  sampling_rate_hz
    void *  p_capture_data_buffer
    uint32_t  capture_data_buffer_size
    uint32_t  sample_count
    void(*  p_callback)(sf_audio_record_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} sf_audio_record_cfg_t
```

11.1.303.1 capture_data_size

sf_audio_record_data_size_t::capture_data_size

Brief description

Size of data in the sample 8 or 16 bit.

11.1.303.2 sampling_rate_hz

```
uint32_t sf_audio_record_cfg_t::sampling_rate_hz
```

Brief description

Sampling rate for audio capture.

11.1.303.3 p_capture_data_buffer

```
void* sf_audio_record_cfg_t::p_capture_data_buffer
```

Detailed description

Pointer to the buffer that will store the samples

11.1.303.4 capture_data_buffer_size

```
uint32_t sf_audio_record_cfg_t::capture_data_buffer_size
```

Detailed description

total size of buffer configured by user to store samples

11.1.303.5 sample_count

```
uint32_t sf_audio_record_cfg_t::sample_count
```

Detailed description

Samples per channel to be buffered before notifying the user via callback

11.1.303.6 p_callback

```
void(* sf_audio_record_cfg_t::p_callback) (sf_audio_record_callback_args_t
*p_args)
```

Brief description

Callback function.

11.1.303.7 p_context

```
void const* sf_audio_record_cfg_t::p_context
```

Brief description

Placeholder for user data.

11.1.303.8 p_extend

```
void const* sf_audio_record_cfg_t::p_extend
```

Detailed description

Extension parameter for hardware specific settings.

11.1.304 sf_audio_record_i2s_hw_cfg_t

```
typedef struct{
    i2s_instance_t const  *  p_lower_lvl_i2s
} sf_audio_record_i2s_hw_cfg_t
```

11.1.304.1 p_lower_lvl_i2s

```
i2s_instance_t::p_lower_lvl_i2s
```

11.1.305 sf_audio_record_i2s_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    TX_MUTEX  mutex
    void *  p_capture_data_buffer
    uint32_t  capture_data_size
    uint32_t  data_size
    uint32_t  buffer_size
    uint32_t  current_buffer_index
    void(*  p_callback)(sf_audio_record_callback_args_t *p_args)
    void const  *  p_context
    i2s_instance_t const  *  p_lower_lvl_i2s
} sf_audio_record_i2s_instance_ctrl_t
```

11.1.305.1 open

```
uint32_t ::open
```

Brief description

Used by driver to check if pointer to control.

11.1.305.2 mutex

```
TX_MUTEX ::mutex
```

Brief description

Mutex used to protect access to lower level driver hardware registers.

11.1.305.3 p_capture_data_buffer

```
void* ::p_capture_data_buffer
```

Brief description

Pointer to the buffer record buffer */.

11.1.305.4 capture_data_size

```
uint32_t ::capture_data_size
```

Brief description

capture data type

11.1.305.5 data_size

```
uint32_t ::data_size
```

Brief description

Number of bytes captured for each iteration.

11.1.305.6 buffer_size

```
uint32_t ::buffer_size
```

Brief description

size of the current record buffer */

11.1.305.7 current_buffer_index

```
uint32_t ::current_buffer_index
```

Brief description

Index into current buffer */.

11.1.305.8 p_callback

```
void(* ::p_callback) ( *p_args)
```

Brief description

Callback function.

11.1.305.9 p_context

```
void const* ::p_context
```

Brief description

Placeholder for user data.

11.1.305.10 p_lower_lvl_i2s

```
i2s_instance_t::p_lower_lvl_i2s
```

Brief description

Lower level I2S instance.

11.1.306 sf_audio_record_info_t

```
typedef struct{
    sf_audio_record_channel_t  channel_info
} sf_audio_record_info_t
```

11.1.306.1 channel_info

```
sf_audio_record_channel_t::channel_info
```

11.1.307 sf_audio_record_instance_t

```
typedef struct{
    sf_audio_record_ctrl_t *  p_ctrl
    sf_audio_record_cfg_t const  *  p_cfg
    sf_audio_record_api_t const  *  p_api
} sf_audio_record_instance_t
```

11.1.307.1 p_ctrl

```
sf_audio_record_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.307.2 p_cfg

sf_audio_record_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.307.3 p_api

sf_audio_record_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.308 sf_ble_addr_t

```
typedef struct{
    uint8_t  addr[SF_BLE_ADDR_LEN]
} sf_ble_addr_t
```

11.1.308.1 addr

uint8_t sf_ble_addr_t::addr[SF_BLE_ADDR_LEN]

Brief description

6-byte array address value

11.1.309 sf_ble_addr_verify_ind_t

```
typedef struct{
    uint8_t  bd_addr[SF_BLE_ADDR_LEN]
    sf_ble_addr_type_t  addr_type
    uint8_t  accept_addr
} sf_ble_addr_verify_ind_t
```

11.1.309.1 bd_addr

uint8_t sf_ble_addr_verify_ind_t::bd_addr[SF_BLE_ADDR_LEN]

Brief description

input parameter specifying bluetooth address of remote device

11.1.309.2 addr_type

sf_ble_addr_type_t::addr_type

Brief description

input parameter specifying Address type of remote BLE device

11.1.309.3 accept_addr

uint8_t sf_ble_addr_verify_ind_t::accept_addr

Brief description

output parameter: application has to set this parameter in callback if it accepts this address

11.1.310 sf_ble_adv_data_t

```
typedef struct{
    uint8_t  data[SF_BLE_ADV_DATA_LEN]
    uint8_t  adv_data_length
} sf_ble_adv_data_t
```

11.1.310.1 data

uint8_t sf_ble_adv_data_t::data[SF_BLE_ADV_DATA_LEN]

Brief description

Advertising data bytes array.

11.1.310.2 adv_data_length

uint8_t sf_ble_adv_data_t::adv_data_length

Brief description

Advertising data length.

11.1.311 sf_ble_adv_info_t

```
typedef struct{
    sf_ble_disc_type_t  disc_mode
    sf_ble_conn_type_t  conn_mode
    sf_ble_adv_type_t  adv_type
    uint16_t  adv_intv_min
    uint16_t  adv_intv_max
    sf_ble_addr_type_t  own_addr_type
    sf_ble_addr_type_t  direct_addr_type
```

```
    uint8_t  direct_bd_addr[SF_BLE_ADDR_LEN]
    sf_ble_adv_chnl_map_t  adv_chnl_map
    sf_ble_adv_filt_type_t  adv_filt_policy
    sf_ble_scan_response_data_t  scan_response_data
    sf_ble_adv_data_t  adv_data
} sf_ble_adv_info_t
```

11.1.311.1 disc_mode

sf_ble_disc_type_t::disc_mode

Brief description

Discovery mode.

11.1.311.2 conn_mode

sf_ble_conn_type_t::conn_mode

Brief description

Connection mode.

11.1.311.3 adv_type

sf_ble_adv_type_t::adv_type

Brief description

Advertisement type.

11.1.311.4 adv_intv_min

uint16_t sf_ble_adv_info_t::adv_intv_min

Brief description

Minimum interval for advertising.

11.1.311.5 adv_intv_max

uint16_t sf_ble_adv_info_t::adv_intv_max

Brief description

Maximum interval for advertising.

11.1.311.6 own_addr_type

sf_ble_addr_type_t::own_addr_type

Brief description

Own address type.

11.1.311.7 direct_addr_type

`sf_ble_addr_type_t::direct_addr_type`

Brief description

Direct connection address type.

11.1.311.8 direct_bd_addr

`uint8_t sf_ble_adv_info_t::direct_bd_addr[SF_BLE_ADDR_LEN]`

Brief description

Direct connection BLE address.

11.1.311.9 adv_chnl_map

`sf_ble_adv_chnl_map_t::adv_chnl_map`

Brief description

Advertising channel map.

11.1.311.10 adv_filt_policy

`sf_ble_adv_filt_type_t::adv_filt_policy`

Brief description

Advertising filter policy.

11.1.311.11 scan_response_data

`sf_ble_scan_response_data_t::scan_response_data`

Brief description

Scan Response information, will be advertised only for active scan.

11.1.311.12 adv_data

`sf_ble_adv_data_t::adv_data`

Brief description

Advertising data.

11.1.312 sf_ble_anp_ancp_change_t

```
typedef struct{
    sf_ble_conn_handle_t  conhdl
    sf_ble_anp_ancp_t  control_point_value
} sf_ble_anp_ancp_change_t
```

11.1.312.1 conhdl

sf_ble_conn_handle_t::conhdl

Brief description

Connection handle.

11.1.312.2 control_point_value

sf_ble_anp_ancp_t::control_point_value

Brief description

Control point value.

11.1.313 sf_ble_anp_ancp_t

```
typedef struct{
    sf_ble_prf_anp_cmd_id_t  command_id
    sf_ble_prf_anp_category_id  category_id
} sf_ble_anp_ancp_t
```

11.1.313.1 command_id

sf_ble_prf_anp_cmd_id_t::command_id

Brief description

Command type.

11.1.313.2 category_id

sf_ble_prf_anp_category_id::category_id

Brief description

Category on which to act.

11.1.314 sf_ble_anp_api_new_alert_ntf_t

```
typedef struct{
    sf_ble_conn_handle_t  conhdl
    sf_ble_anp_api_new_alert_t  new_alert
} sf_ble_anp_api_new_alert_ntf_t
```

11.1.315 sf_ble_anp_api_new_alert_t

```
typedef struct{
    sf_ble_prf_anp_category_id  category_id
    uint8_t  alert_num
    uint8_t  text_size
    uint8_t  text[SF_BLE_ANP_ALT_TEXT_MAX]
} sf_ble_anp_api_new_alert_t
```

11.1.316 sf_ble_anp_api_unread_alert_ntf_t

```
typedef struct{
    sf_ble_conn_handle_t  conhdl
    sf_ble_anp_api_unread_alert_t  alert
} sf_ble_anp_api_unread_alert_ntf_t
```

11.1.317 sf_ble_anp_api_unread_alert_t

```
typedef struct{
    sf_ble_prf_anp_category_id  category_id
    uint8_t  unread_count
} sf_ble_anp_api_unread_alert_t
```

11.1.318 sf_ble_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_ble_ctrl_t *const p_ctrl, const sf_ble_cfg_t *p_cfg)
    ssp_err_t(*  close)(sf_ble_ctrl_t *const p_ctrl)
    ssp_err_t(*  infoGet)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t
*p_handle, sf_ble_info_t *p_ble_info)
    ssp_err_t(*  provisionGet)(sf_ble_ctrl_t *const p_ctrl, sf_ble_provisioning_t
*p_ble_provisioning)
    ssp_err_t(*  provisionSet)(sf_ble_ctrl_t *const p_ctrl, const
sf_ble_provisioning_t *p_ble_provisioning)
    ssp_err_t(*  scan)(sf_ble_ctrl_t *const p_ctrl, sf_ble_scan_t *p_scan,
uint8_t *p_cnt, sf_ble_scan_info_t *p_scan_info)
    ssp_err_t(*  advertisementStart)(sf_ble_ctrl_t *const p_ctrl,
sf_ble_adv_info_t *const p_advt_info)
```

```
    ssp_err_t(*  advertisementStop)(sf_ble_ctrl_t *const p_ctrl)
    ssp_err_t(*  whitelistAdd)(sf_ble_ctrl_t *const p_ctrl, const uint8_t
*p_bd_addr)
    ssp_err_t(*  whitelistDel)(sf_ble_ctrl_t *const p_ctrl, const uint8_t
*p_bd_addr)
    ssp_err_t(*  bondingStart)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t
*p_handle, const uint8_t *p_bd_addr, sf_ble_bonding_start_t *p_bonding_start)
    ssp_err_t(*  bondingResponse)(sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, const uint8_t *p_bd_addr,
sf_ble_bonding_response_t *p_bonding_resp)
    ssp_err_t(*  startEncryption)(sf_ble_ctrl_t *const p_ctrl,
sf_ble_sm_enc_info_t const *p_enc_info)
    ssp_err_t(*  connect)(sf_ble_ctrl_t *const p_ctrl, sf_ble_connection_t const
*const p_conn, sf_ble_conn_handle_t *p_handle)
    ssp_err_t(*  listen)(sf_ble_ctrl_t *const p_ctrl)
    ssp_err_t(*  authorization)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t
*p_handle)
    ssp_err_t(*  disconnect)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t
*p_handle)
    ssp_err_t(*  setTxPower)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t
*const p_handle, sf_ble_set_tx_pwr_info_t *p_tx_power_info)
    ssp_err_t(*  gattAddCustomProfiles)(sf_ble_ctrl_t *const p_ctrl,
sf_ble_svc_attribute_t *p_svc_attr, uint32_t svc_attr_len,
sf_ble_char_attribute_t *p_char_attr, uint32_t char_attr_len)
    ssp_err_t(*  gattServiceDiscovery)(sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, sf_ble_service_discovery_req_t const *const
p_sf_ble_svc_dscv_req, sf_ble_service_discovery_rsp_t *const
p_sf_ble_svc_dscv_rsp, uint32_t *const p_rsp_cnt)
    ssp_err_t(*  gattCharDiscovery)(sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, sf_ble_char_discovery_req_t const *const
p_sf_ble_char_dscv_req, sf_ble_char_discovery_rsp_t *const
p_sf_ble_char_dscv_rsp, uint32_t *const p_rsp_cnt)
    ssp_err_t(*  gattCharDescDiscovery)(sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, uint16_t start_handle, uint16_t end_handle,
sf_ble_char_desc_discovery_rsp_t *const p_sf_ble_chardesc_dscv_rsp, uint32_t
*const p_rsp_cnt)
    ssp_err_t(*  gattCharRead)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t
*p_handle, sf_ble_char_read_req_t const *const p_char_read_req,
sf_ble_char_read_rsp_t *const p_char_read_rsp)
    ssp_err_t(*  gattCharWrite)(sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t
*p_handle, sf_ble_char_write_req_t const *const p_char_write_req)
    ssp_err_t(*  gattCharExecuteWrite)(sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, sf_ble_execute_write_t execute_flag)
    ssp_err_t(*  gattCharWriteLocal)(sf_ble_ctrl_t *const p_ctrl, uint16_t
char_handle, uint16_t data_length, uint8_t *const p_data)
    ssp_err_t(*  gattSendNotify)(sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, uint16_t char_handle)
    ssp_err_t(*  gattSendIndicate)(sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, uint16_t char_handle)
```

```
    ssp_err_t(*  gattWriteResponse)(sf_ble_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, uint16_t handle, sf_ble_attribute_error_code_t
error_code)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_ble_api_t
```

11.1.319 sf_ble_attr_info_t

```
typedef struct{
    uint8_t  len
    uint8_t  data[SF_BLE_ATTM_MAX_VALUE]
} sf_ble_attr_info_t
```

11.1.319.1 len

```
uint8_t sf_ble_attr_info_t::len
```

Brief description

data length

11.1.319.2 data

```
uint8_t sf_ble_attr_info_t::data[SF_BLE_ATTM_MAX_VALUE]
```

Brief description

data

11.1.320 sf_ble_bas_battery_lvl_ntf_t

```
typedef struct{
    sf_ble_conn_handle_t  conhdl
    sf_ble_prf_bas_battery_lvl_t  batt_lvl
} sf_ble_bas_battery_lvl_ntf_t
```

11.1.320.1 conhdl

```
sf_ble_conn_handle_t::conhdl
```

Brief description

Connection handle.

11.1.320.2 batt_lvl

```
sf_ble_prf_bas_battery_lvl_t::batt_lvl
```

Brief description

Battery Level.

11.1.321 sf_ble_blp_meas_info_t

```
typedef struct{
    uint8_t  flag_stable_meas
    uint8_t  flags
    int16_t  press_val1
    int16_t  press_val2
    int16_t  press_val3
    sf_ble_prf_cts_date_time_t  stamp
    int16_t  rate
    uint8_t  id
    uint8_t  reserved
    uint16_t  meas_sts
} sf_ble_blp_meas_info_t
```

11.1.321.1 flag_stable_meas

```
uint8_t sf_ble_blp_meas_info_t::flag_stable_meas
```

Brief description

Stable or intermediary type of measurements. Set to 0 if intermediate measurement.

11.1.321.2 flags

```
uint8_t sf_ble_blp_meas_info_t::flags
```

Brief description

flags

11.1.321.3 press_val1

```
int16_t sf_ble_blp_meas_info_t::press_val1
```

Brief description

blood pressure value - Systolic or cuff pressure. Cuff pressure has to be filled here

11.1.321.4 press_val2

```
int16_t sf_ble_blp_meas_info_t::press_val2
```

Brief description

blood pressure value - Diastolic or subfield1

11.1.321.5 press_val3

```
int16_t sf_ble_blp_meas_info_t::press_val3
```

Brief description

blood pressure value - MAP or subfield2

11.1.321.6 stamp

```
sf_ble_prf_cts_date_time_t::stamp
```

Brief description

time stamp

11.1.321.7 rate

```
int16_t sf_ble_blp_meas_info_t::rate
```

Brief description

pulse rate

11.1.321.8 id

```
uint8_t sf_ble_blp_meas_info_t::id
```

Brief description

user ID

11.1.321.9 reserved

```
uint8_t sf_ble_blp_meas_info_t::reserved
```

Brief description

Reserved.

11.1.321.10 meas_sts

```
uint16_t sf_ble_blp_meas_info_t::meas_sts
```

Brief description

measurement status

11.1.322 sf_ble_blp_meas_recv_data_t

```
typedef struct{
    uint16_t  conhdl
    sf_ble_onbp_char_t  char_code
    sf_ble_blp_meas_info_t  meas_info
} sf_ble_blp_meas_recv_data_t
```

11.1.322.1 conhdl

uint16_t sf_ble_blp_meas_recv_data_t::conhdl

Brief description

Connection handle.

11.1.322.2 char_code

sf_ble_onbp_char_t::char_code

Brief description

Characteristic code.

11.1.322.3 meas_info

sf_ble_blp_meas_info_t::meas_info

Brief description

BLP measurement data.

11.1.323 sf_ble_bonding_req_ind_t

```
typedef struct{
    sf_ble_addr_t  bd_addr
    uint8_t  index
    uint8_t  auth_req
    uint8_t  io_cap
    uint8_t  oob_data_flg
    uint8_t  max_enc_size
    uint8_t  ikey_dist
    uint8_t  rkey_dist
} sf_ble_bonding_req_ind_t
```

11.1.323.1 bd_addr

sf_ble_addr_t::bd_addr

Brief description

BLE address.

11.1.323.2 index

`uint8_t sf_ble_bonding_req_ind_t::index`

Brief description

Connection index.

11.1.323.3 auth_req

`uint8_t sf_ble_bonding_req_ind_t::auth_req`

Brief description

Authentication request type.

11.1.323.4 io_cap

`uint8_t sf_ble_bonding_req_ind_t::io_cap`

Brief description

IO capability.

11.1.323.5 oob_data_flg

`uint8_t sf_ble_bonding_req_ind_t::oob_data_flg`

Brief description

Indicating if OOB data is present.

11.1.323.6 max_enc_size

`uint8_t sf_ble_bonding_req_ind_t::max_enc_size`

Brief description

Maximum encryption key size.

11.1.323.7 ikey_dist

`uint8_t sf_ble_bonding_req_ind_t::ikey_dist`

Brief description

Type of key distributed by the initiator.

11.1.323.8 rkey_dist

uint8_t sf_ble_bonding_req_ind_t::rkey_dist

Brief description

Type of key distributed by the responder.

11.1.324 sf_ble_bonding_response_t

```
typedef struct{
    uint8_t  accept
    sf_ble_iocap_t  io_cap
    uint8_t  max_key_size
    sf_ble_key_dist_t  ikeys
    sf_ble_key_dist_t  rkeys
} sf_ble_bonding_response_t
```

11.1.324.1 accept

uint8_t sf_ble_bonding_response_t::accept

Brief description

accept or reject bonding

11.1.324.2 io_cap

sf_ble_iocap_t::io_cap

Brief description

IO capabilities.

11.1.324.3 max_key_size

uint8_t sf_ble_bonding_response_t::max_key_size

Brief description

Max key size.

11.1.324.4 ikeys

sf_ble_key_dist_t::ikeys

Brief description

Initiator key distribution.

11.1.324.5 rkeys

sf_ble_key_dist_t::rkeys

Brief description

Responder key distribution.

11.1.325 sf_ble_bonding_start_t

```
typedef struct{
    sf_ble_iocap_t  iocap
    uint8_t  key_size
    sf_ble_key_dist_t  ikey_dist
    sf_ble_key_dist_t  rkey_dist
} sf_ble_bonding_start_t
```

11.1.325.1 iocap

sf_ble_iocap_t::iocap

Brief description

IO capabilities.

11.1.325.2 key_size

uint8_t sf_ble_bonding_start_t::key_size

Brief description

Encryption key size.

11.1.325.3 ikey_dist

sf_ble_key_dist_t::ikey_dist

Brief description

Initiator key distribution.

11.1.325.4 rkey_dist

sf_ble_key_dist_t::rkey_dist

Brief description

Responder key distribution.

11.1.326 sf_ble_cfg_t

```
typedef struct{
    uint8_t  bd_addr[SF_BLE_ADDR_LEN]
    sf_ble_addr_type_t  own_addr_type
    uint8_t  max_slaves
    uint8_t  update_bd_addr
    uint16_t  scan_interval
    uint16_t  scan_window
    uint16_t  disc_time
    uint16_t  con_interval
    uint16_t  slave_latency
    uint16_t  sup_timeout
    void const  *  p_extend
} sf_ble_cfg_t
```

11.1.326.1 bd_addr

```
uint8_t sf_ble_cfg_t::bd_addr[SF_BLE_ADDR_LEN]
```

Brief description

BLE address.

11.1.326.2 own_addr_type

```
sf_ble_addr_type_t::own_addr_type
```

Brief description

self address type

11.1.326.3 max_slaves

```
uint8_t sf_ble_cfg_t::max_slaves
```

Brief description

Maximum slaves allowed to be connected.

11.1.326.4 update_bd_addr

```
uint8_t sf_ble_cfg_t::update_bd_addr
```

Brief description

Set this to true to update bluetooth address during SF_BLE_Open.

11.1.326.5 scan_interval

```
uint16_t sf_ble_cfg_t::scan_interval
```

Brief description

BLE scan interval for receiving advertisement.

11.1.326.6 scan_window

```
uint16_t sf_ble_cfg_t::scan_window
```

Brief description

Period of time during which advertising data is received at the scan interval.

11.1.326.7 disc_time

```
uint16_t sf_ble_cfg_t::disc_time
```

Brief description

Duration for which the device remain discoverable.

11.1.326.8 con_interval

```
uint16_t sf_ble_cfg_t::con_interval
```

Brief description

Interval for transmitting and receiving data periodically after connection establishment.

11.1.326.9 slave_latency

```
uint16_t sf_ble_cfg_t::slave_latency
```

Brief description

Period of time during which data is transmitted and received at the connection interval.

11.1.326.10 sup_timeout

```
uint16_t sf_ble_cfg_t::sup_timeout
```

Brief description

Link loss time-out.

11.1.326.11 p_extend

```
void const* sf_ble_cfg_t::p_extend
```

Brief description

Instance specific configuration.

11.1.327 sf_ble_char_attribute_t

```
typedef struct{
    sf_ble_svc_attribute_t *  p_service
    sf_ble_uuid_t  attr_uuid
    uint16_t  attr_declare_handle
    uint16_t  attr_declare_type
    uint16_t  attr_value_handle
    sf_ble_char_attr_permissions_t  attr_perm
    sf_ble_char_property_t  attr_properties
    uint8_t *  p_attr_value
    uint8_t  attr_value_len
} sf_ble_char_attribute_t
```

11.1.327.1 p_service

sf_ble_svc_attribute_t::p_service

Brief description

Service to which this characteristics belongs.

11.1.327.2 attr_uuid

sf_ble_uuid_t::attr_uuid

Brief description

UUID of characteristics value.

11.1.327.3 attr_declare_handle

uint16_t sf_ble_char_attribute_t::attr_declare_handle

Brief description

Characteristics handle.

11.1.327.4 attr_declare_type

uint16_t sf_ble_char_attribute_t::attr_declare_type

Brief description

Characteristics declare type SF_BLE_GATT_CHAR_DECLARE.

11.1.327.5 attr_value_handle

uint16_t sf_ble_char_attribute_t::attr_value_handle

Brief description

Characteristics value handle.

11.1.327.6 attr_perm

sf_ble_char_attr_permissions_t::attr_perm

Brief description

Characteristics permission.

11.1.327.7 attr_properties

sf_ble_char_property_t::attr_properties

Brief description

Characteristics properties.

11.1.327.8 p_attr_value

uint8_t* sf_ble_char_attribute_t::p_attr_value

Brief description

Characteristics value data.

11.1.327.9 attr_value_len

uint8_t sf_ble_char_attribute_t::attr_value_len

Brief description

Characteristics value data length.

11.1.328 sf_ble_char_desc_discovery_rsp_t

```
typedef struct{
    uint16_t  desc_handle
    sf_ble_uuid_t  uuid
} sf_ble_char_desc_discovery_rsp_t
```

11.1.328.1 desc_handle

uint16_t sf_ble_char_desc_discovery_rsp_t::desc_handle

Brief description

Characteristic descriptor handle.

11.1.328.2 uuid

sf_ble_uuid_t::uuid

Brief description

Characteristic descriptor UUID.

11.1.329 sf_ble_char_discovery_req_t

```
typedef struct{
    sf_ble_uuid_t  uuid
    uint16_t  start_handle
    uint16_t  end_handle
    sf_ble_char_discovery_t  discovery_type
} sf_ble_char_discovery_req_t
```

11.1.329.1 uuid

sf_ble_uuid_t::uuid

Brief description

Characteristic UUID.

11.1.329.2 start_handle

uint16_t sf_ble_char_discovery_req_t::start_handle

Brief description

Discovery start handle.

11.1.329.3 end_handle

uint16_t sf_ble_char_discovery_req_t::end_handle

Brief description

Discovery end handle.

11.1.329.4 discovery_type

sf_ble_char_discovery_t::discovery_type

Brief description

Characteristic discovery type.

11.1.330 sf_ble_char_discovery_rsp_t

```
typedef struct{
    uint16_t  char_handle
    sf_ble_uuid_t  uuid
    uint16_t  value_handle
    sf_ble_char_property_t  property
} sf_ble_char_discovery_rsp_t
```

11.1.330.1 char_handle

```
uint16_t sf_ble_char_discovery_rsp_t::char_handle
```

Brief description

Characteristic handle.

11.1.330.2 uuid

```
sf_ble_uuid_t::uuid
```

Brief description

Characteristic UUID.

11.1.330.3 value_handle

```
uint16_t sf_ble_char_discovery_rsp_t::value_handle
```

Brief description

Characteristic value handle.

11.1.330.4 property

```
sf_ble_char_property_t::property
```

Brief description

Characteristic property.

11.1.331 sf_ble_char_multiple_read_req_t

```
typedef struct{
    uint16_t  handles[SF_BLE_MAX_MULTI_CHAR_READ_CNT]
```

```
    uint8_t  expected_result_size[SF_BLE_MAX_MULTI_CHAR_READ_CNT]
} sf_ble_char_multiple_read_req_t
```

11.1.331.1 handles

```
uint16_t
sf_ble_char_multiple_read_req_t::handles[SF_BLE_MAX_MULTI_CHAR_READ_CNT]
```

Brief description

Characteristic handles for multiple read.

11.1.331.2 expected_result_size

```
uint8_t
sf_ble_char_multiple_read_req_t::expected_result_size[SF_BLE_MAX_MULTI_CHAR_READ
_CNT]
```

Brief description

Expected result length in bytes.

11.1.332 sf_ble_char_multiple_read_rsp_t

```
typedef struct{
    uint8_t *  p_data[SF_BLE_MAX_MULTI_CHAR_READ_CNT]
    uint16_t  data_len[SF_BLE_MAX_MULTI_CHAR_READ_CNT]
} sf_ble_char_multiple_read_rsp_t
```

11.1.332.1 p_data

```
uint8_t* sf_ble_char_multiple_read_rsp_t::p_data[SF_BLE_MAX_MULTI_CHAR_READ_CNT]
```

Brief description

Pointer to Characteristic value.

11.1.332.2 data_len

```
uint16_t
sf_ble_char_multiple_read_rsp_t::data_len[SF_BLE_MAX_MULTI_CHAR_READ_CNT]
```

Brief description

Characteristic value length.

11.1.333 sf_ble_char_read_by_handle_rsp_t

```
typedef struct{
    uint8_t *  p_data
    uint16_t  data_len
} sf_ble_char_read_by_handle_rsp_t
```

11.1.333.1 p_data

```
uint8_t* sf_ble_char_read_by_handle_rsp_t::p_data
```

Brief description

Pointer to Characteristic value.

11.1.333.2 data_len

```
uint16_t sf_ble_char_read_by_handle_rsp_t::data_len
```

Brief description

Characteristic value length.

11.1.334 sf_ble_char_read_by_uuid_rsp_t

```
typedef struct{
    uint16_t  handle[SF_BLE_MAX_CHAR_UUID_READ_CNT]
    uint8_t *  p_data[SF_BLE_MAX_CHAR_UUID_READ_CNT]
    uint16_t  data_len[SF_BLE_MAX_CHAR_UUID_READ_CNT]
} sf_ble_char_read_by_uuid_rsp_t
```

11.1.334.1 handle

```
uint16_t sf_ble_char_read_by_uuid_rsp_t::handle[SF_BLE_MAX_CHAR_UUID_READ_CNT]
```

Brief description

Characteristic handles.

11.1.334.2 p_data

```
uint8_t* sf_ble_char_read_by_uuid_rsp_t::p_data[SF_BLE_MAX_CHAR_UUID_READ_CNT]
```

Brief description

Pointer to Characteristic value.

11.1.334.3 data_len

```
uint16_t sf_ble_char_read_by_uuid_rsp_t::data_len[SF_BLE_MAX_CHAR_UUID_READ_CNT]
```

Brief description

Characteristic value length.

11.1.335 sf_ble_char_read_req_t

```
typedef struct{
    sf_ble_char_read_t  char_read_type
    uint16_t  handle
    sf_ble_uuid_t  uuid
    uint16_t  offset
    sf_ble_char_multiple_read_req_t *  p_mul_read_req
    uint16_t  handles_cnt
} sf_ble_char_read_req_t
```

11.1.335.1 char_read_type

```
sf_ble_char_read_t::char_read_type
```

Brief description

Characteristic read type.

11.1.335.2 handle

```
uint16_t sf_ble_char_read_req_t::handle
```

Brief description

Characteristic value or descriptor handle.

11.1.335.3 uuid

```
sf_ble_uuid_t::uuid
```

Brief description

Characteristic UUID.

11.1.335.4 offset

```
uint16_t sf_ble_char_read_req_t::offset
```

Brief description

Offset for long Characteristic value.

11.1.335.5 p_mul_read_req

sf_ble_char_multiple_read_req_t::p_mul_read_req

Brief description

Pointer to multiple read Characteristic request.

11.1.335.6 handles_cnt

uint16_t sf_ble_char_read_req_t::handles_cnt

Brief description

Characteristic handles count for multiple read.

11.1.336 sf_ble_char_read_rsp_t

```
typedef struct{
    sf_ble_char_read_by_handle_rsp_t *  p_char_read_by_handle_rsp
    sf_ble_char_read_by_uuid_rsp_t *  p_char_read_by_uuid_rsp
    sf_ble_char_multiple_read_rsp_t *  p_char_multiple_read_rsp
} sf_ble_char_read_rsp_t
```

11.1.336.1 p_char_read_by_handle_rsp

sf_ble_char_read_by_handle_rsp_t::p_char_read_by_handle_rsp

Brief description

Response for read Characteristic by handle.

11.1.336.2 p_char_read_by_uuid_rsp

sf_ble_char_read_by_uuid_rsp_t::p_char_read_by_uuid_rsp

Brief description

Response for read Characteristic by UUID.

11.1.336.3 p_char_multiple_read_rsp

sf_ble_char_multiple_read_rsp_t::p_char_multiple_read_rsp

Brief description

Response for read multiple Characteristics.

11.1.337 sf_ble_char_write_req_t

```
typedef struct{
    sf_ble_char_write_t  char_write_type
    uint16_t  handle
    uint8_t *  p_data
    uint16_t  offset
    uint16_t  data_length
    sf_ble_execute_write_t  auto_execute
} sf_ble_char_write_req_t
```

11.1.337.1 char_write_type

sf_ble_char_write_t::char_write_type

Brief description

Characteristic write type.

11.1.337.2 handle

uint16_t sf_ble_char_write_req_t::handle

Brief description

Characteristic value or descriptor handle.

11.1.337.3 p_data

uint8_t* sf_ble_char_write_req_t::p_data

Brief description

Pointer to data.

11.1.337.4 offset

uint16_t sf_ble_char_write_req_t::offset

Brief description

Offset for long Characteristic value.

11.1.337.5 data_length

uint16_t sf_ble_char_write_req_t::data_length

Brief description

Data length.

11.1.337.6 auto_execute

sf_ble_execute_write_t::auto_execute

Brief description

Automatic execute write flag.

11.1.338 sf_ble_chipset_info_t

```
typedef struct{
    uint32_t  version
    uint8_t  bd_addr[SF_BLE_ADDR_LEN]
} sf_ble_chipset_info_t
```

11.1.338.1 version

uint32_t sf_ble_chipset_info_t::version

Brief description

chipset version

11.1.338.2 bd_addr

uint8_t sf_ble_chipset_info_t::bd_addr[SF_BLE_ADDR_LEN]

Brief description

BLE address.

11.1.339 sf_ble_connect_info_t

```
typedef struct{
    uint8_t  status
    uint8_t  reserved
    sf_ble_conn_handle_t  conhdl
    uint8_t  peer_addr_type
    sf_ble_addr_t  peer_addr
    uint8_t  reserved2
    uint16_t  con_interval
    uint16_t  con_latency
    uint16_t  sup_to
    uint8_t  clk_accuracy
    uint8_t  reserved3
} sf_ble_connect_info_t
```

11.1.339.1 status

uint8_t sf_ble_connect_info_t::status

Brief description

Confirmation status.

11.1.339.2 reserved

uint8_t sf_ble_connect_info_t::reserved

Brief description

Reserved.

11.1.339.3 conhdl

sf_ble_conn_handle_t::conhdl

Brief description

Connection handle.

11.1.339.4 peer_addr_type

uint8_t sf_ble_connect_info_t::peer_addr_type

Brief description

Peer address type.

11.1.339.5 peer_addr

sf_ble_addr_t::peer_addr

Brief description

Peer BT address.

11.1.339.6 reserved2

uint8_t sf_ble_connect_info_t::reserved2

Brief description

Reserved.

11.1.339.7 con_interval

uint16_t sf_ble_connect_info_t::con_interval

Brief description

Connection interval.

11.1.339.8 con_latency

`uint16_t sf_ble_connect_info_t::con_latency`

Brief description

Connection latency.

11.1.339.9 sup_to

`uint16_t sf_ble_connect_info_t::sup_to`

Brief description

Link supervision time-out.

11.1.339.10 clk_accuracy

`uint8_t sf_ble_connect_info_t::clk_accuracy`

Brief description

Clock accuracy.

11.1.339.11 reserved3

`uint8_t sf_ble_connect_info_t::reserved3`

Brief description

Reserved.

11.1.340 sf_ble_connection_t

```
typedef struct{
    sf_ble_init_filt_type_t  init_filt_type
    sf_ble_addr_type_t  addr_type
    uint8_t  bd_addr[SF_BLE_ADDR_LEN]
} sf_ble_connection_t
```

11.1.340.1 init_filt_type

`sf_ble_init_filt_type_t::init_filt_type`

Brief description

Connection filter type.

11.1.340.2 addr_type

`sf_ble_addr_type_t::addr_type`

Brief description

BLE address type.

11.1.340.3 bd_addr

`uint8_t sf_ble_connection_t::bd_addr[SF_BLE_ADDR_LEN]`

Brief description

BLE address.

11.1.341 sf_ble_ctrl_t

```
typedef struct{
    void *  p_driver_handle
} sf_ble_ctrl_t
```

11.1.341.1 p_driver_handle

`void* sf_ble_ctrl_t::p_driver_handle`

Brief description

Storage for information needed for each BLE device driver in the system.

11.1.342 sf_ble_cts_curr_time_ntf_t

```
typedef struct{
    sf_ble_conn_handle_t  conhdl
    sf_ble_prf_cts_curr_time_t  current_time
} sf_ble_cts_curr_time_ntf_t
```

11.1.342.1 conhdl

`sf_ble_conn_handle_t::conhdl`

Brief description

Connection handle.

11.1.342.2 current_time

sf_ble_prf_cts_curr_time_t::current_time

Brief description

heart rate measurement data

11.1.343 sf_ble_cts_local_time_t

```
typedef struct{
    int8_t  time_zone
    uint8_t  dst_offset
} sf_ble_cts_local_time_t
```

11.1.343.1 time_zone

int8_t sf_ble_cts_local_time_t::time_zone

Brief description

Time Zone.

11.1.343.2 dst_offset

uint8_t sf_ble_cts_local_time_t::dst_offset

Brief description

DST Offset.

11.1.344 sf_ble_cts_ref_time_t

```
typedef struct{
    uint8_t  time_source
    uint8_t  accuracy
    uint8_t  days_since_update
    uint8_t  hours_since_update
} sf_ble_cts_ref_time_t
```

11.1.344.1 time_source

uint8_t sf_ble_cts_ref_time_t::time_source

Brief description

Source of time.

11.1.344.2 accuracy

`uint8_t sf_ble_cts_ref_time_t::accuracy`

Brief description

Estimated accuracy of time compared to original time source.

11.1.344.3 days_since_update

`uint8_t sf_ble_cts_ref_time_t::days_since_update`

Brief description

Days that passed since time was updated successfully from time source.

11.1.344.4 hours_since_update

`uint8_t sf_ble_cts_ref_time_t::hours_since_update`

Brief description

Time that passed since time was updated successfully from time source.

11.1.345 sf_ble_disconnect_t

```
typedef struct{
    sf_ble_disconnect_reason_t  reason
    uint8_t  status
    sf_ble_conn_handle_t  conhdl
} sf_ble_disconnect_t
```

11.1.345.1 reason

`sf_ble_disconnect_reason_t::reason`

Brief description

Disconnection reason.

11.1.345.2 status

`uint8_t sf_ble_disconnect_t::status`

Brief description

Disconnect status if initiated by local host.

11.1.345.3 conhdl

sf_ble_conn_handle_t::conhdl

Brief description

Connection handle of the remote device.

11.1.346 sf_ble_event_info_t

```
typedef struct{
    void *  p_data
    uint16_t  event
} sf_ble_event_info_t
```

11.1.346.1 p_data

void* sf_ble_event_info_t::p_data

Brief description

Data for BLE event.

11.1.346.2 event

uint16_t sf_ble_event_info_t::event

Brief description

Event type.

11.1.347 sf_ble_gatt_attr_event_t

```
typedef struct{
    uint16_t  attr_handle
    uint16_t  offset
    uint8_t const  *  p_value
    uint16_t  length
    ssp_err_t  response
} sf_ble_gatt_attr_event_t
```

11.1.347.1 attr_handle

uint16_t sf_ble_gatt_attr_event_t::attr_handle

Brief description

Attribute handle for which event is generated.

11.1.347.2 offset

```
uint16_t sf_ble_gatt_attr_event_t::offset
```

Brief description

The offset at which the client is willing to write.

11.1.347.3 p_value

```
uint8_t const* sf_ble_gatt_attr_event_t::p_value
```

Brief description

The value at which the client is willing to write.

11.1.347.4 length

```
uint16_t sf_ble_gatt_attr_event_t::length
```

Brief description

The amount of bytes that the client is willing to write as from this offset.

11.1.347.5 response

```
ssp_err_t sf_ble_gatt_attr_event_t::response
```

Brief description

User will update this variable, Pass SSP_SUCCESS if user accepts the remote request.

11.1.348 sf_ble_gatt_notif_ind_event_data_t

```
typedef struct{
    uint16_t  handle
    uint8_t *  p_data
    uint16_t  data_length
} sf_ble_gatt_notif_ind_event_data_t
```

11.1.348.1 handle

```
uint16_t sf_ble_gatt_notif_ind_event_data_t::handle
```

Brief description

Characteristic value or descriptor handle.

11.1.348.2 p_data

```
uint8_t* sf_ble_gatt_notif_ind_event_data_t::p_data
```

Brief description

Pointer to data.

11.1.348.3 data_length

```
uint16_t sf_ble_gatt_notif_ind_event_data_t::data_length
```

Brief description

Data length.

11.1.349 sf_ble_hrp_api_hrmeas_t

```
typedef struct{
    uint8_t  flags
    uint8_t  rr_interval_num
    uint16_t  heart_rate_measure
    uint16_t  energy_expended
    uint16_t  rr_interval[SF_BLE_PRF_HRP_API_RR_INTERVAL_BUFF_LEN]
} sf_ble_hrp_api_hrmeas_t
```

11.1.350 sf_ble_hrp_api_meas_ntf_t

```
typedef struct{
    sf_ble_conn_handle_t  conhdl
    sf_ble_hrp_api_hrmeas_t  measurements_info
} sf_ble_hrp_api_meas_ntf_t
```

11.1.351 sf_ble_hrp_cp_change_t

```
typedef struct{
    sf_ble_conn_handle_t  conhdl
    sf_ble_prf_hrp_api_hrcp_t  control_point_value
} sf_ble_hrp_cp_change_t
```

11.1.351.1 conhdl

```
sf_ble_conn_handle_t::conhdl
```

Brief description

Connection handle.

11.1.351.2 control_point_value

sf_ble_prf_hrp_api_hrcp_t::control_point_value

Brief description

Control point value.

11.1.352 sf_ble_info_t

```
typedef struct{
    sf_ble_chipset_info_t  chipset
    uint16_t  rssi
} sf_ble_info_t
```

11.1.352.1 chipset

sf_ble_chipset_info_t::chipset

Brief description

Chipset information.

11.1.352.2 rssi

uint16_t sf_ble_info_t::rssi

Brief description

RSSI value.

11.1.353 sf_ble_instance_t

```
typedef struct{
    sf_ble_ctrl_t *  p_ctrl
    sf_ble_cfg_t const  *  p_cfg
    sf_ble_api_t const  *  p_api
} sf_ble_instance_t
```

11.1.353.1 p_ctrl

sf_ble_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.353.2 p_cfg

sf_ble_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.353.3 p_api

sf_ble_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.354 sf_ble_long_attr_info_t

```
typedef struct{
    uint8_t  val_len
    uint8_t  reserved
    uint16_t  attr_hdl
    uint8_t  value[SF_BLE_ATTM_MAX_VALUE]
} sf_ble_long_attr_info_t
```

11.1.354.1 val_len

uint8_t sf_ble_long_attr_info_t::val_len

Brief description

size of the value data

11.1.354.2 reserved

uint8_t sf_ble_long_attr_info_t::reserved

Brief description

Reserved.

11.1.354.3 attr_hdl

uint16_t sf_ble_long_attr_info_t::attr_hdl

Brief description

Attribute handle.

11.1.354.4 value

uint8_t sf_ble_long_attr_info_t::value[SF_BLE_ATTM_MAX_VALUE]

Brief description

actual value pairs

11.1.355 sf_ble_onboard_profile_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_ble_onboard_profile_ctrl_t *const p_ctrl, const
sf_ble_onboard_profile_cfg_t *p_cfg)
    ssp_err_t(*  close)(sf_ble_onboard_profile_ctrl_t *const p_ctrl)
    ssp_err_t(*  onbpEnable)(sf_ble_onboard_profile_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_profile_callback_t
p_prf_cb, sf_ble_prf_sec_t sec)
    ssp_err_t(*  onbpDisable)(sf_ble_onboard_profile_ctrl_t *const p_ctrl,
sf_ble_conn_handle_t *p_handle, sf_onbp_t profile)
    ssp_err_t(*  onbpServerWriteData)(sf_ble_onboard_profile_ctrl_t *const
p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t
characteristics, const void *p_data)
    ssp_err_t(*  onbpServerSendNotification)(sf_ble_onboard_profile_ctrl_t
*const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile,
sf_ble_onbp_char_t characteristics, const void *p_data)
    ssp_err_t(*  onbpServerSendIndication)(sf_ble_onboard_profile_ctrl_t *const
p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t
characteristics, const void *p_data)
    ssp_err_t(*  onbpClientWriteCCCD)(sf_ble_onboard_profile_ctrl_t *const
p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t
cccd_char, sf_ble_cccd_val_t cccd_val)
    ssp_err_t(*  onbpClientWriteChar)(sf_ble_onboard_profile_ctrl_t *const
p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t
characteristics, const void *p_data)
    ssp_err_t(*  onbpClientReadChar)(sf_ble_onboard_profile_ctrl_t *const
p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t
characteristics)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_ble_onboard_profile_api_t
```

11.1.356 sf_ble_onboard_profile_cccd_changed_t

```
typedef struct{
    sf_ble_conn_handle_t  conn_handle
    sf_ble_onbp_char_t  char_code
    sf_ble_cccd_val_t  cccd_val
    uint8_t  inst_idx
} sf_ble_onboard_profile_cccd_changed_t
```

11.1.356.1 conn_handle

sf_ble_conn_handle_t::conn_handle

Brief description

Connection handle.

11.1.356.2 char_code

sf_ble_onbp_char_t::char_code

Brief description

CCCD type that has been changed by the remote node.

11.1.356.3 cccd_val

sf_ble_cccd_val_t::cccd_val

Brief description

CCCD value.

11.1.356.4 inst_idx

uint8_t sf_ble_onboard_profile_cccd_changed_t::inst_idx

Brief description

Instance index, Applicable for HOGP.

11.1.357 sf_ble_onboard_profile_cfg_t

```
typedef struct{
    sf_ble_instance_t const  *  p_low_lvl_ble
    void *  p_extend
} sf_ble_onboard_profile_cfg_t
```

11.1.357.1 p_low_lvl_ble

sf_ble_instance_t::p_low_lvl_ble

Brief description

Low level BLE Interface.

11.1.357.2 p_extend

void* sf_ble_onboard_profile_cfg_t::p_extend

Brief description

Extended configuration.

11.1.358 sf_ble_onboard_profile_ctrl_t

```
typedef struct{
    sf_ble_instance_t *  p_low_lvl_ble
} sf_ble_onboard_profile_ctrl_t
```

11.1.358.1 p_low_lvl_ble

sf_ble_instance_t::p_low_lvl_ble

Brief description

Low level BLE Framework information needed by On-Board Profile.

11.1.359 sf_ble_onboard_profile_instance_t

```
typedef struct{
    sf_ble_onboard_profile_ctrl_t *  p_ctrl
    sf_ble_onboard_profile_cfg_t const  *  p_cfg
    sf_ble_onboard_profile_api_t const  *  p_api
} sf_ble_onboard_profile_instance_t
```

11.1.359.1 p_ctrl

sf_ble_onboard_profile_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.359.2 p_cfg

sf_ble_onboard_profile_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.359.3 p_api

sf_ble_onboard_profile_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.360 sf_ble_prf_alert_status_ntf_t

```
typedef struct{
    sf_ble_conn_handle_t  conhdl
    sf_ble_prf_alert_status  alert_status
} sf_ble_prf_alert_status_ntf_t
```

11.1.360.1 conhdl

sf_ble_conn_handle_t::conhdl

Brief description

Connection handle.

11.1.360.2 alert_status

sf_ble_prf_alert_status::alert_status

Brief description

Ringer control point value.

11.1.361 sf_ble_prf_cts_curr_time_t

```
typedef struct{
    sf_ble_prf_cts_date_time_t  stamp
    uint8_t  day_of_week
    uint8_t  fractions256
    uint8_t  adjust_reason
    uint8_t  reserved
} sf_ble_prf_cts_curr_time_t
```

11.1.361.1 stamp

sf_ble_prf_cts_date_time_t::stamp

Brief description

Date time info.

11.1.361.2 day_of_week

uint8_t sf_ble_prf_cts_curr_time_t::day_of_week

Brief description

Day of week.

11.1.361.3 fractions256

uint8_t sf_ble_prf_cts_curr_time_t::fractions256

Brief description

Fraction value.

11.1.361.4 adjust_reason

uint8_t sf_ble_prf_cts_curr_time_t::adjust_reason

Brief description

Adjust reason.

11.1.361.5 reserved

uint8_t sf_ble_prf_cts_curr_time_t::reserved

Brief description

Reserved.

11.1.362 sf_ble_prf_cts_date_time_t

```
typedef struct{
    uint16_t  year
    uint8_t  month
    uint8_t  day
    uint8_t  hour
    uint8_t  min
    uint8_t  sec
    uint8_t  reserved
} sf_ble_prf_cts_date_time_t
```

11.1.362.1 year

uint16_t sf_ble_prf_cts_date_time_t::year

Brief description

Year value.

11.1.362.2 month

uint8_t sf_ble_prf_cts_date_time_t::month

Brief description

Month value.

11.1.362.3 day

`uint8_t sf_ble_prf_cts_date_time_t::day`

Brief description

Day value.

11.1.362.4 hour

`uint8_t sf_ble_prf_cts_date_time_t::hour`

Brief description

Hour value.

11.1.362.5 min

`uint8_t sf_ble_prf_cts_date_time_t::min`

Brief description

Minute value.

11.1.362.6 sec

`uint8_t sf_ble_prf_cts_date_time_t::sec`

Brief description

Second value.

11.1.362.7 reserved

`uint8_t sf_ble_prf_cts_date_time_t::reserved`

Brief description

Reserved.

11.1.363 sf_ble_prf_dis_pnpid_t

```
typedef struct{
    uint8_t  vendorIdSource
    uint16_t  vendorId
    uint16_t  productId
```

```
    uint16_t  productVersion
} sf_ble_prf_dis_pnpid_t
```

11.1.363.1 vendorIdSource

```
uint8_t sf_ble_prf_dis_pnpid_t::vendorIdSource
```

Brief description

Vendor ID source.

11.1.363.2 vendorId

```
uint16_t sf_ble_prf_dis_pnpid_t::vendorId
```

Brief description

Vendor ID.

11.1.363.3 productId

```
uint16_t sf_ble_prf_dis_pnpid_t::productId
```

Brief description

Product ID.

11.1.363.4 productVersion

```
uint16_t sf_ble_prf_dis_pnpid_t::productVersion
```

Brief description

Version of Product.

11.1.364 sf_ble_prf_hid_change_event_t

```
typedef struct{
    uint16_t  conhdl
    uint8_t  inst_idx
    sf_ble_prf_value_t  ble_prf_value
} sf_ble_prf_hid_change_event_t
```

11.1.364.1 conhdl

```
uint16_t sf_ble_prf_hid_change_event_t::conhdl
```

Brief description

Connection handle.

11.1.364.2 inst_idx

uint8_t sf_ble_prf_hid_change_event_t::inst_idx

Brief description

Instance Index.

11.1.364.3 ble_prf_value

sf_ble_prf_value_t::ble_prf_value

Brief description

HID Profile value changed by the client.

11.1.365 sf_ble_prf_hid_report_desc_t

```
typedef struct{
    sf_ble_hidd_device_type_t  device_type
    uint8_t  report_type
    uint8_t  value[SF_BLE_PRF_HIDS_REPORT_MAX]
    uint16_t  value_size
} sf_ble_prf_hid_report_desc_t
```

11.1.365.1 device_type

sf_ble_hidd_device_type_t::device_type

Brief description

Device type.

11.1.365.2 report_type

uint8_t sf_ble_prf_hid_report_desc_t::report_type

Brief description

Report type.

11.1.365.3 value

uint8_t sf_ble_prf_hid_report_desc_t::value[SF_BLE_PRF_HIDS_REPORT_MAX]

Brief description

Report values.

11.1.365.4 value_size

```
uint16_t sf_ble_prf_hid_report_desc_t::value_size
```

Brief description

Report size.

11.1.366 sf_ble_prf_hid_report_ind_t

```
typedef struct{
    uint16_t  conhdl
    uint8_t  inst_idx
    uint8_t  reserved
    sf_ble_prf_hid_report_desc_t  report
} sf_ble_prf_hid_report_ind_t
```

11.1.366.1 conhdl

```
uint16_t sf_ble_prf_hid_report_ind_t::conhdl
```

Brief description

Connection handle.

11.1.366.2 inst_idx

```
uint8_t sf_ble_prf_hid_report_ind_t::inst_idx
```

Brief description

Instance Index.

11.1.366.3 reserved

```
uint8_t sf_ble_prf_hid_report_ind_t::reserved
```

Brief description

Reserved.

11.1.366.4 report

```
sf_ble_prf_hid_report_desc_t::report
```

Brief description

Report received from either BHOST or RHOST.

11.1.367 sf_ble_prf_htp_temp_info_ind_t

```
typedef struct{
    uint16_t  conhdl
    sf_ble_prf_htp_temp_info_t  temp_info
} sf_ble_prf_htp_temp_info_ind_t
```

11.1.367.1 conhdl

```
uint16_t sf_ble_prf_htp_temp_info_ind_t::conhdl
```

Brief description

Connection handle.

11.1.367.2 temp_info

```
sf_ble_prf_htp_temp_info_t::temp_info
```

Brief description

Thermometer info.

11.1.368 sf_ble_prf_htp_temp_info_t

```
typedef struct{
    uint8_t  flag_stable_meas
    uint8_t  flags
    int32_t  temp_val
    sf_ble_prf_cts_date_time_t  stamp
    sf_ble_prf_htp_temp_type_t  type
    uint8_t  reserved
} sf_ble_prf_htp_temp_info_t
```

11.1.368.1 flag_stable_meas

```
uint8_t sf_ble_prf_htp_temp_info_t::flag_stable_meas
```

Brief description

Stable or intermediary type of temperature.

11.1.368.2 flags

`uint8_t sf_ble_prf_htp_temp_info_t::flags`

Brief description

flags

11.1.368.3 temp_val

`int32_t sf_ble_prf_htp_temp_info_t::temp_val`

Brief description

temp value

11.1.368.4 stamp

`sf_ble_prf_cts_date_time_t::stamp`

Brief description

time stamp

11.1.368.5 type

`sf_ble_prf_htp_temp_type_t::type`

Brief description

type

11.1.368.6 reserved

`uint8_t sf_ble_prf_htp_temp_info_t::reserved`

Brief description

Reserved.

11.1.369 sf_ble_prf_ias_alert_lvl_change_t

```
typedef struct{
    sf_ble_conn_handle_t  conhdl
    sf_ble_prf_ias_alert_type_t  alert_lvl
} sf_ble_prf_ias_alert_lvl_change_t
```

11.1.369.1 conhdl

`sf_ble_conn_handle_t::conhdl`

Brief description

Connection handle.

11.1.369.2 alert_lvl

`sf_ble_prf_ias_alert_type_t::alert_lvl`

Brief description

Control point value.

11.1.370 sf_ble_prf_ndcs_time_dst_t

```
typedef struct{
    sf_ble_prf_cts_date_time_t  stamp
    uint8_t  dst_offset
    uint8_t  reserved
} sf_ble_prf_ndcs_time_dst_t
```

11.1.370.1 stamp

`sf_ble_prf_cts_date_time_t::stamp`

Brief description

Current time stamp.

11.1.370.2 dst_offset

`uint8_t sf_ble_prf_ndcs_time_dst_t::dst_offset`

Brief description

DST Offset.

11.1.370.3 reserved

`uint8_t sf_ble_prf_ndcs_time_dst_t::reserved`

Brief description

Reserved.

11.1.371 sf_ble_prf_ringer_cp_change_t

```
typedef struct{
    sf_ble_conn_handle_t  conhdl
```

```
    sf_ble_prf_ringer_cp_t  ringer_cp
} sf_ble_prf_ringer_cp_change_t
```

11.1.371.1 conhdl

sf_ble_conn_handle_t::conhdl

Brief description

Connection handle.

11.1.371.2 ringer_cp

sf_ble_prf_ringer_cp_t::ringer_cp

Brief description

Ringer control point value.

11.1.372 sf_ble_prf_ringer_setting_ntf_t

```
typedef struct{
    sf_ble_conn_handle_t  conhdl
    sf_ble_prf_ringer_setting_t  ringer_setting
} sf_ble_prf_ringer_setting_ntf_t
```

11.1.372.1 conhdl

sf_ble_conn_handle_t::conhdl

Brief description

Connection handle.

11.1.372.2 ringer_setting

sf_ble_prf_ringer_setting_t::ringer_setting

Brief description

Ringer control point value.

11.1.373 sf_ble_prf_rtus_time_updt_state_t

```
typedef struct{
    uint8_t  current_state
    uint8_t  update_result
} sf_ble_prf_rtus_time_updt_state_t
```

11.1.373.1 current_state

```
uint8_t sf_ble_prf_rtus_time_::current_state
```

Brief description

Current state of Reference time.

11.1.373.2 update_result

```
uint8_t sf_ble_prf_rtus_time_::update_result
```

Brief description

Result of update.

11.1.374 sf_ble_prf_scps_scan_intv_t

```
typedef struct{
    uint16_t  le_scan_interval
    uint16_t  le_scan_window
} sf_ble_prf_scps_scan_intv_t
```

11.1.374.1 le_scan_interval

```
uint16_t sf_ble_prf_scps_scan_intv_t::le_scan_interval
```

Brief description

scan interval value

11.1.374.2 le_scan_window

```
uint16_t sf_ble_prf_scps_scan_intv_t::le_scan_window
```

Brief description

scan window value

11.1.375 sf_ble_prf_tip_write_data_t

```
typedef struct{
    sf_ble_prf_cts_curr_time_t  current_time
    sf_ble_cts_local_time_t  local_time
    sf_ble_cts_ref_time_t  ref_time
    sf_ble_prf_ndcs_time_dst_t  next_dst
    sf_ble_prf_rtus_time_updt_state_t  update_state
} sf_ble_prf_tip_write_data_t
```

11.1.375.1 current_time

sf_ble_prf_cts_curr_time_t::current_time

Brief description

Current Time Information.

11.1.375.2 local_time

sf_ble_cts_local_time_t::local_time

Brief description

Local Time Information.

11.1.375.3 ref_time

sf_ble_cts_ref_time_t::ref_time

Brief description

Reference Time Information.

11.1.375.4 next_dst

sf_ble_prf_ndcs_time_dst_t::next_dst

Brief description

Next DST Information.

11.1.375.5 update_state

sf_ble_prf_rtus_time_updt_state_t::update_state

Brief description

Update Status Information.

11.1.376 sf_ble_prf_value_t

```
typedef struct{
    sf_ble_prf_hid_protocol_mode_t  protocol_mode_val
    sf_ble_prf_hid_ctrl_point_val_t  control_point_val
} sf_ble_prf_value_t
```

11.1.376.1 protocol_mode_val

sf_ble_prf_hid_protocol_mode_t::protocol_mode_val

Brief description

Protocol Mode.

11.1.376.2 control_point_val

`sf_ble_prf_hid_ctrl_point_val_t::control_point_val`

Brief description

HID Control Point.

11.1.377 sf_ble_provisioning_t

```
typedef struct{
    uint8_t  gap_name[SF_BLE_MAX_GAP_NAME_LEN]
    uint8_t  bcast_mode
    sf_ble_bonding_mode_t  bonding_mode
    sf_ble_sec_info_t  sec_info
    sf_ble_gap_role_t  gap_role
    sf_ble_callback_t  p_ble_callback
} sf_ble_provisioning_t
```

11.1.377.1 gap_name

`uint8_t sf_ble_provisioning_t::gap_name[SF_BLE_MAX_GAP_NAME_LEN]`

Brief description

GAP name.

11.1.377.2 bcast_mode

`uint8_t sf_ble_provisioning_t::bcast_mode`

Brief description

broadcast mode

11.1.377.3 bonding_mode

`sf_ble_bonding_mode_t::bonding_mode`

Brief description

bonding mode

11.1.377.4 sec_info

sf_ble_sec_info_t::sec_info

Brief description

Security information.

11.1.377.5 gap_role

sf_ble_gap_role_t::gap_role

Brief description

GAP role (master/slave)

11.1.377.6 p_ble_callback

sf_ble_callback_t::p_ble_callback

Detailed description

GAP user event callback that runs in driver thread context. Application should make sure callback logic is as minimal as possible without any blocking calls

11.1.378 sf_ble_scan_info_t

```
typedef struct{
    uint16_t  total_scan_duration
    sf_ble_scan_mode_t  scan_mode
    sf_ble_disc_type_t  discovery_type
    sf_ble_addr_type_t  address_type
    sf_ble_adv_filt_type_t  filt_policy
    sf_ble_duplicate_filter_t  duplicate_filt
} sf_ble_scan_info_t
```

11.1.378.1 total_scan_duration

uint16_t sf_ble_scan_info_t::total_scan_duration

Brief description

BLE total scan duration in milliseconds.

11.1.378.2 scan_mode

sf_ble_scan_mode_t::scan_mode

Brief description

BLE scan mode.

11.1.378.3 discovery_type

`sf_ble_disc_type_t::discovery_type`

Brief description

Specifies discovery type, Used only in active scan.

11.1.378.4 address_type

`sf_ble_addr_type_t::address_type`

Brief description

BLE address type.

11.1.378.5 filt_policy

`sf_ble_adv_filt_type_t::filt_policy`

Brief description

Scan Filter policy.

11.1.378.6 duplicate_filt

`sf_ble_duplicate_filter_t::duplicate_filt`

Brief description

Duplicate filter configuration.

11.1.379 sf_ble_scan_response_data_t

```
typedef struct{
    uint8_t  scan_response_data[SF_BLE_ADV_DATA_LEN]
    uint8_t  scan_response_data_length
} sf_ble_scan_response_data_t
```

11.1.379.1 scan_response_data

`uint8_t sf_ble_scan_response_data_t::scan_response_data[SF_BLE_ADV_DATA_LEN]`

Brief description

Advertising data bytes array.

11.1.379.2 scan_response_data_length

```
uint8_t sf_ble_scan_response_data_t::scan_response_data_length
```

Brief description

Advertising data length.

11.1.380 sf_ble_scan_t

```
typedef struct{
    sf_ble_addr_type_t  addr_type
    uint8_t  bd_addr[SF_BLE_ADDR_LEN]
    uint16_t  data_len
    uint8_t  data[SF_BLE_MAX_BLE_ADV_DATA_LEN]
    int16_t  rssi
    sf_ble_adv_event_type_t  event_type
} sf_ble_scan_t
```

11.1.380.1 addr_type

```
sf_ble_addr_type_t::addr_type
```

Brief description

Address type of remote BLE device.

11.1.380.2 bd_addr

```
uint8_t sf_ble_scan_t::bd_addr[SF_BLE_ADDR_LEN]
```

Brief description

Remote BLE address.

11.1.380.3 data_len

```
uint16_t sf_ble_scan_t::data_len
```

Brief description

Scan data length.

11.1.380.4 data

```
uint8_t sf_ble_scan_t::data[SF_BLE_MAX_BLE_ADV_DATA_LEN]
```

Brief description

Scan data.

11.1.380.5 rssi

```
int16_t sf_ble_scan_t::rssi
```

Brief description

RSSI value.

11.1.380.6 event_type

```
sf_ble_adv_event_type_t::event_type
```

Brief description

Advertising event type.

11.1.381 sf_ble_scps_scan_intv_change_t

```
typedef struct{
    sf_ble_conn_handle_t  conhdl
    sf_ble_prf_scps_scan_intv_t  scan_interval_window_val
} sf_ble_scps_scan_intv_change_t
```

11.1.381.1 conhdl

```
sf_ble_conn_handle_t::conhdl
```

Brief description

Connection handle.

11.1.381.2 scan_interval_window_val

```
sf_ble_prf_scps_scan_intv_t::scan_interval_window_val
```

Brief description

Scan Interval window.

11.1.382 sf_ble_sec_enc_start_ind_t

```
typedef struct{
    sf_ble_conn_handle_t  conhdl
    uint8_t  status
    uint8_t  key_size
    sf_ble_auth_type_t  auth_type
    uint8_t  bonding_status
} sf_ble_sec_enc_start_ind_t
```

11.1.382.1 conhdl

sf_ble_conn_handle_t::conhdl

Brief description

Connection handle of the remote device.

11.1.382.2 status

uint8_t sf_ble_sec_enc_start_ind_t::status

Brief description

Result of encryption start, 0 = SUCCESS else error.

11.1.382.3 key_size

uint8_t sf_ble_sec_enc_start_ind_t::key_size

Brief description

Key Size.

11.1.382.4 auth_type

sf_ble_auth_type_t::auth_type

Brief description

Security properties.

11.1.382.5 bonding_status

uint8_t sf_ble_sec_enc_start_ind_t::bonding_status

Brief description

Bonding Status, 0 = Unbonded, 1 = Bonded.

11.1.383 sf_ble_sec_info_t

```
typedef struct{
    sf_ble_sec_mode_t  sec_mode
    uint8_t  id_key[SF_BLE_SEC_KEY_LEN]
    uint8_t  csrk_key[SF_BLE_SEC_KEY_LEN]
    uint8_t  ltk_key[SF_BLE_SEC_KEY_LEN]
    uint8_t  rand_num[SF_BLE_RAND_NUM_LENGTH]
    uint16_t  ediv
} sf_ble_sec_info_t
```

11.1.383.1 sec_mode

`sf_ble_sec_mode_t::sec_mode`

Brief description

security mode

11.1.383.2 id_key

`uint8_t sf_ble_sec_info_t::id_key[SF_BLE_SEC_KEY_LEN]`

Brief description

GAP Identity Resolving Security key.

11.1.383.3 csrk_key

`uint8_t sf_ble_sec_info_t::csrk_key[SF_BLE_SEC_KEY_LEN]`

Brief description

GAP Connection Signature Resolving Security key.

11.1.383.4 ltk_key

`uint8_t sf_ble_sec_info_t::ltk_key[SF_BLE_SEC_KEY_LEN]`

Brief description

GAP Connection Long term Security key.

11.1.383.5 rand_num

`uint8_t sf_ble_sec_info_t::rand_num[SF_BLE_RAND_NUM_LENGTH]`

Brief description

GAP Connection Random number for Long term Security key.

11.1.383.6 ediv

`uint16_t sf_ble_sec_info_t::ediv`

Brief description

GAP Connection Encrypted Diversifier for Long term Security key.

11.1.384 sf_ble_service_discovery_req_t

```
typedef struct{
    sf_ble_uuid_t  uuid
    uint16_t  start_handle
    uint16_t  end_handle
    sf_ble_service_discovery_t  discovery_type
} sf_ble_service_discovery_req_t
```

11.1.384.1 uuid

sf_ble_uuid_t::uuid

Brief description

Service UUID.

11.1.384.2 start_handle

uint16_t sf_ble_service_discovery_req_t::start_handle

Brief description

Discovery start handle.

11.1.384.3 end_handle

uint16_t sf_ble_service_discovery_req_t::end_handle

Brief description

Discovery end handle.

11.1.384.4 discovery_type

sf_ble_service_discovery_t::discovery_type

Brief description

Service discovery type.

11.1.385 sf_ble_service_discovery_rsp_t

```
typedef struct{
    uint16_t  service_handle
    sf_ble_uuid_t  uuid
    uint16_t  start_handle
    uint16_t  end_handle
} sf_ble_service_discovery_rsp_t
```

11.1.385.1 service_handle

```
uint16_t sf_ble_service_discovery_rsp_t::service_handle
```

Brief description

Service handle.

11.1.385.2 uuid

```
sf_ble_uuid_t::uuid
```

Brief description

Service UUID.

11.1.385.3 start_handle

```
uint16_t sf_ble_service_discovery_rsp_t::start_handle
```

Brief description

Start handle of sub-attributes handle range.

11.1.385.4 end_handle

```
uint16_t sf_ble_service_discovery_rsp_t::end_handle
```

Brief description

End handle of sub-attributes handle range.

11.1.386 sf_ble_set_tx_pwr_info_t

```
typedef struct{
    int8_t  power_lvl
    sf_ble_tx_pwr_state_t  state
} sf_ble_set_tx_pwr_info_t
```

11.1.386.1 power_lvl

```
int8_t sf_ble_set_tx_pwr_info_t::power_lvl
```

Brief description

TX power level in dBm.

11.1.386.2 state

```
sf_ble_tx_pwr_state_t::state
```

Brief description

Operating state to set the transmit power level.

11.1.387 sf_ble_sm_enc_info_t

```
typedef struct{
    sf_ble_conn_handle_t  conn_idx
    uint16_t  ediv
    uint8_t  rand_num[SF_BLE_RAND_NUM_LENGTH]
    uint8_t  ltk_key[SF_BLE_SEC_KEY_LEN]
} sf_ble_sm_enc_info_t
```

11.1.387.1 conn_idx

sf_ble_conn_handle_t::conn_idx

Brief description

connection index

11.1.387.2 ediv

uint16_t sf_ble_sm_enc_info_t::ediv

Brief description

BLE Security EDIV.

11.1.387.3 rand_num

uint8_t sf_ble_sm_enc_info_t::rand_num[SF_BLE_RAND_NUM_LENGTH]

Brief description

Random number for security.

11.1.387.4 ltk_key

uint8_t sf_ble_sm_enc_info_t::ltk_key[SF_BLE_SEC_KEY_LEN]

Brief description

BLE long term security key.

11.1.388 sf_ble_sm_key_ind_t

```
typedef struct{
    uint8_t  conn_idx
```

```
    sf_ble_key_dist_t  key_code
    uint16_t  ediv
    uint8_t  rand_num[SF_BLE_RAND_NUM_LENGTH]
    uint8_t  ltk_key[SF_BLE_SEC_KEY_LEN]
} sf_ble_sm_key_ind_t
```

11.1.388.1 conn_idx

`uint8_t sf_ble_sm_key_ind_t::conn_idx`

Brief description

connection index

11.1.388.2 key_code

`sf_ble_key_dist_t::key_code`

Brief description

type of security key

11.1.388.3 ediv

`uint16_t sf_ble_sm_key_ind_t::ediv`

Brief description

BLE Security EDIV.

11.1.388.4 rand_num

`uint8_t sf_ble_sm_key_ind_t::rand_num[SF_BLE_RAND_NUM_LENGTH]`

Brief description

Random number for security.

11.1.388.5 ltk_key

`uint8_t sf_ble_sm_key_ind_t::ltk_key[SF_BLE_SEC_KEY_LEN]`

Brief description

BLE long term security key.

11.1.389 sf_ble_sm_tk_ind_t

```
typedef struct{
    sf_ble_sm_tk_info_t  tk_info
    uint8_t  tk_req_status
    uint8_t  tk_key[SF_BLE_SEC_KEY_LEN]
} sf_ble_sm_tk_ind_t
```

11.1.389.1 tk_info

`sf_ble_sm_tk_info_t::tk_info`

Brief description

input parameter,information to app about temporary key request

11.1.389.2 tk_req_status

`uint8_t sf_ble_sm_tk_ind_t::tk_req_status`

Brief description

output parameter: request status, application has to set this in callback event whether request is valid or not

11.1.389.3 tk_key

`uint8_t sf_ble_sm_tk_ind_t::tk_key[SF_BLE_SEC_KEY_LEN]`

Brief description

application has to set this in callback event temporary key for encryption

11.1.390 sf_ble_sm_tk_info_t

```
typedef struct{
    sf_ble_conn_handle_t  conhdl
    uint8_t  disp_en
} sf_ble_sm_tk_info_t
```

11.1.390.1 conhdl

`sf_ble_conn_handle_t::conhdl`

Brief description

Connection handle.

11.1.390.2 disp_en

```
uint8_t sf_ble_sm_tk_info_t::disp_en
```

Brief description

Whether to enable display.

11.1.391 sf_ble_svc_attribute_t

```
typedef struct{
    uint16_t  attr_handle
    uint16_t  attr_type
    uint16_t  parent_svc_handle
    uint8_t *  p_attr_value
    sf_ble_uuid_length_t  attr_value_len
} sf_ble_svc_attribute_t
```

11.1.391.1 attr_handle

```
uint16_t sf_ble_svc_attribute_t::attr_handle
```

Brief description

Service Handle.

11.1.391.2 attr_type

```
uint16_t sf_ble_svc_attribute_t::attr_type
```

Brief description

Attribute type such as SF_BLE_GATT_PRI_SERVICE, SF_BLE_GATT_INCLUDE_SERVICE.

11.1.391.3 parent_svc_handle

```
uint16_t sf_ble_svc_attribute_t::parent_svc_handle
```

Brief description

parent_svc_handle is service handle of parent service which is already registered

Detailed description

If service is included service,

11.1.391.4 p_attr_value

```
uint8_t* sf_ble_svc_attribute_t::p_attr_value
```

Brief description

Service UUID Pointer.

11.1.391.5 attr_value_len

`sf_ble_uuid_length_t::attr_value_len`

Brief description

UUID Length.

11.1.392 sf_ble_tip_cp_change_t

```
typedef struct{
    sf_ble_conn_handle_t  conhdl
    sf_ble_prf_tip_time_ctrl_point_t  control_point_value
} sf_ble_tip_cp_change_t
```

11.1.392.1 conhdl

`sf_ble_conn_handle_t::conhdl`

Brief description

Connection handle.

11.1.392.2 control_point_value

`sf_ble_prf_tip_time_ctrl_point_t::control_point_value`

Brief description

Time Update Control point value.

11.1.393 sf_ble_uuid_t

```
typedef struct{
    uint16_t  uuid16
    uint32_t  uuid32
    uint8_t  uuid128[SF_BLE_128BITS_UUID_LENGTH]
    union{}                 union{}
    sf_ble_uuid_length_t  uuid_length
} sf_ble_uuid_t
```

11.1.393.1 uuid16

`uint16_t sf_ble_uuid_t::uuid16`

Brief description

16 bit UUID

11.1.393.2 uuid32

```
uint32_t sf_ble_uuid_t::uuid32
```

Brief description

32 bit UUID

11.1.393.3 uuid128

```
uint8_t sf_ble_uuid_t::uuid128[SF_BLE_128BITS_UUID_LENGTH]
```

Brief description

128 bit UUID

11.1.393.4 union{}

```
See source code for the definition of this member.
```

11.1.393.5 uuid_length

```
sf_ble_uuid_length_t::uuid_length
```

Brief description

Service UUID length.

11.1.394 sf_ble_write_cmd_event_data_t

```
typedef struct{
    uint16_t  handle
    uint16_t  offset
    sf_ble_write_response_t  response_required
    uint8_t *  p_data
    uint16_t  data_length
} sf_ble_write_cmd_event_data_t
```

11.1.394.1 handle

```
uint16_t sf_ble_write_cmd_event_data_t::handle
```

Brief description

Characteristic value or descriptor handle.

11.1.394.2 offset

`uint16_t sf_ble_write_cmd_event_data_t::offset`

Brief description

Offset for long Characteristic value.

11.1.394.3 response_required

`sf_ble_write_response_t::response_required`

Brief description

Write response required or not.

11.1.394.4 p_data

`uint8_t* sf_ble_write_cmd_event_data_t::p_data`

Brief description

Pointer to data.

11.1.394.5 data_length

`uint16_t sf_ble_write_cmd_event_data_t::data_length`

Brief description

Data length.

11.1.395 sf_block_media_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_block_media_ctrl_t *p_ctrl, sf_block_media_cfg_t const
*const p_cfg)
    ssp_err_t(*  read)(sf_block_media_ctrl_t *p_ctrl, uint8_t *const p_dest,
uint32_t const start_sector, uint32_t const sector_count)
    ssp_err_t(*  write)(sf_block_media_ctrl_t *p_ctrl, uint8_t const *const
p_src, uint32_t const start_sector, uint32_t const sector_count)
    ssp_err_t(*  ioctl)(sf_block_media_ctrl_t *p_ctrl, ssp_command_t const
command, void *p_data)
    ssp_err_t(*  close)(sf_block_media_ctrl_t *p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_block_media_api_t
```

11.1.396 sf_block_media_cfg_t

```
typedef struct{
    uint32_t  block_size
    void const  *  p_extend
} sf_block_media_cfg_t
```

11.1.396.1 block_size

```
uint32_t sf_block_media_cfg_t::block_size
```

Brief description

Block size in bytes.

11.1.396.2 p_extend

```
void const* sf_block_media_cfg_t::p_extend
```

Brief description

Instance dependent configuration.

11.1.397 sf_block_media_instance_t

```
typedef struct{
    sf_block_media_ctrl_t *  p_ctrl
    sf_block_media_cfg_t const  *  p_cfg
    sf_block_media_api_t const  *  p_api
} sf_block_media_instance_t
```

11.1.397.1 p_ctrl

```
sf_block_media_ctrl_t::p_ctrl
```

Brief description

Block media pointer to device driver control structure.

11.1.397.2 p_cfg

```
sf_block_media_cfg_t::p_cfg
```

Brief description

Block media pointer to device driver configuration structure.

11.1.397.3 p_api

sf_block_media_api_t::p_api

Brief description

Block media pointer to device driver api structure.

11.1.398 sf_block_media_on_qspi_cfg_t

```
typedef struct{
    qspi_instance_t const  *const  p_lower_lvl_qspi
} sf_block_media_on_qspi_cfg_t
```

11.1.398.1 p_lower_lvl_qspi

qspi_instance_t::p_lower_lvl_qspi

Brief description

Pointer to QSPI instance structure.

11.1.399 sf_block_media_on_ram_cfg_t

```
typedef struct{
    uint8_t *  p_ram_buffer
    uint32_t  ram_buffer_size
} sf_block_media_on_ram_cfg_t
```

11.1.399.1 p_ram_buffer

uint8_t* ::p_ram_buffer

Brief description

Pointer to RAM buffer address.

11.1.399.2 ram_buffer_size

uint32_t ::ram_buffer_size

Brief description

Size of RAM buffer.

11.1.400 sf_block_media_on_sdmmc_cfg_t

```
typedef struct{
    sdmmc_instance_t const  *const  p_lower_lvl_sdmmc
} sf_block_media_on_sdmmc_cfg_t
```

11.1.400.1 p_lower_lvl_sdmmc

sdmmc_instance_t::p_lower_lvl_sdmmc

Brief description

Pointer to SDMMC instance structure.

11.1.401 sf_block_media_qspi_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    uint32_t  block_size
    qspi_instance_t *  p_lower_lvl_qspi
    TX_MUTEX  mutex
} sf_block_media_qspi_instance_ctrl_t
```

11.1.401.1 open

uint32_t ::open

Brief description

Used to determine if framework is initialized.

11.1.401.2 block_size

uint32_t ::block_size

Brief description

Block size in bytes.

11.1.401.3 p_lower_lvl_qspi

qspi_instance_t::p_lower_lvl_qspi

Brief description

Pointer to QSPI instance structure.

11.1.401.4 mutex

```
TX_MUTEX ::mutex
```

Brief description

Mutex used to protect access to lower level driver hardware registers.

11.1.402 sf_block_media_ram_instance_ctrl_t

```
typedef struct{
    uint8_t *  p_ram_buffer
    uint32_t  block_size
    uint32_t  open
    uint32_t  ram_buffer_size
} sf_block_media_ram_instance_ctrl_t
```

11.1.402.1 p_ram_buffer

```
uint8_t* ::p_ram_buffer
```

Brief description

pointer to RAM buffer address

11.1.402.2 block_size

```
uint32_t ::block_size
```

Brief description

Block size in bytes.

11.1.402.3 open

```
uint32_t ::open
```

Brief description

Used to determine if framework is initialized.

11.1.402.4 ram_buffer_size

```
uint32_t ::ram_buffer_size
```

Brief description

Size of RAM buffer.

11.1.403 sf_block_media_sdmmc_instance_ctrl_t

```
typedef struct{
    uint32_t  block_size
    sdmmc_instance_t *  p_lower_lvl_sdmmc
    TX_EVENT_FLAGS_GROUP  eventflag
    uint32_t  open
} sf_block_media_sdmmc_instance_ctrl_t
```

11.1.403.1 block_size

```
uint32_t ::block_size
```

Brief description

Block size in bytes.

11.1.403.2 p_lower_lvl_sdmmc

```
sdmmc_instance_t::p_lower_lvl_sdmmc
```

Brief description

Pointer to SDMMC instance structure.

11.1.403.3 eventflag

```
TX_EVENT_FLAGS_GROUP ::eventflag
```

Brief description

Pointer to the event flag object for SDMMC data transfer.

11.1.403.4 open

```
uint32_t ::open
```

Brief description

Used to determine if framework is initialized.

11.1.404 sf_cellular_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_cellular_ctrl_t *p_ctrl, sf_cellular_cfg_t const *const
p_cfg)
    ssp_err_t(*  close)(sf_cellular_ctrl_t *const p_ctrl)
    ssp_err_t(*  provisioningGet)(sf_cellular_ctrl_t *const p_ctrl,
sf_cellular_provisioning_t *const p_cellular_provisioning)
```

```
    ssp_err_t(*  provisioningSet)(sf_cellular_ctrl_t *const p_ctrl,
sf_cellular_provisioning_t const *const p_cellular_provisioning)
    ssp_err_t(*  infoGet)(sf_cellular_ctrl_t *const p_ctrl, sf_cellular_info_t
*const p_cellular_info)
    ssp_err_t(*  statisticsGet)(sf_cellular_ctrl_t *const p_ctrl,
sf_cellular_stats_t *const p_cellular_device_stats)
    ssp_err_t(*  transmit)(sf_cellular_ctrl_t *const p_ctrl, uint8_t *const
p_buf, uint32_t length)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
    ssp_err_t(*  networkConnect)(sf_cellular_ctrl_t *const p_ctrl)
    ssp_err_t(*  networkDisconnect)(sf_cellular_ctrl_t *const p_ctrl)
    ssp_err_t(*  networkStatusGet)(sf_cellular_ctrl_t *const p_ctrl,
sf_cellular_network_status_t *p_network_status)
    ssp_err_t(*  simPinSet)(sf_cellular_ctrl_t *const p_ctrl, uint8_t *const
p_old_pin, uint8_t *const p_new_pin)
    ssp_err_t(*  simLock)(sf_cellular_ctrl_t *const p_ctrl, uint8_t *const p_pin)
    ssp_err_t(*  simUnlock)(sf_cellular_ctrl_t *const p_ctrl, uint8_t *const
p_pin)
    ssp_err_t(*  simIDGet)(sf_cellular_ctrl_t *const p_ctrl, uint8_t *p_sim_id)
    ssp_err_t(*  fotaCheck)(sf_cellular_ctrl_t *const p_ctrl, void *p_fotacheck)
    ssp_err_t(*  fotaStart)(sf_cellular_ctrl_t *const p_ctrl, void *p_fotastart)
    ssp_err_t(*  fotaStop)(sf_cellular_ctrl_t *const p_ctrl, void *p_fotastop)
    ssp_err_t(*  reset)(sf_cellular_ctrl_t *const p_ctrl,
sf_cellular_reset_type_t reset_type)
} sf_cellular_api_t
```

11.1.405 sf_cellular_at_cmd_set_t

```
typedef struct{
    uint8_t *  p_cmd
    uint8_t *  p_success_resp
    uint16_t  max_resp_length
    uint8_t  retry
    uint16_t  retry_delay
} sf_cellular_at_cmd_set_t
```

11.1.405.1 p_cmd

```
uint8_t* sf_cellular_at_cmd_set_t::p_cmd
```

Brief description

AT Command.

11.1.405.2 p_success_resp

```
uint8_t* sf_cellular_at_cmd_set_t::p_success_resp
```

Brief description

Success response string.

11.1.405.3 max_resp_length

```
uint16_t sf_cellular_at_cmd_set_t::max_resp_length
```

Brief description

Maximum length of expected response.

11.1.405.4 retry

```
uint8_t sf_cellular_at_cmd_set_t::retry
```

Brief description

Retry count.

11.1.405.5 retry_delay

```
uint16_t sf_cellular_at_cmd_set_t::retry_delay
```

Brief description

Delay between AT command retry.

11.1.406 sf_cellular_callback_args_t

```
typedef struct{
    sf_cellular_event_t  event
    uint8_t *  p_data
    uint32_t  length
    void const  *  p_context
} sf_cellular_callback_args_t
```

11.1.406.1 event

```
sf_cellular_event_t::event
```

Brief description

Event Code.

11.1.406.2 p_data

```
uint8_t* sf_cellular_callback_args_t::p_data
```

Brief description

Pointer to received data.

11.1.406.3 length

```
uint32_t sf_cellular_callback_args_t::length
```

Brief description

Receive Data Length.

11.1.406.4 p_context

```
void const* sf_cellular_callback_args_t::p_context
```

Brief description

Context Provided to user during callback.

11.1.407 sf_cellular_cfg_t

```
typedef struct{
    sf_cellular_op_select_mode_t  op_select_mode
    sf_cellular_op_t  op
    uint16_t  num_pref_ops
    sf_cellular_op_t  pref_ops[SF_CELLULAR_MAX_PREFFERED_OPERATOR_COUNT]
    sf_cellular_timezone_update_mode_t  tz_upd_mode
    uint8_t *  p_sim_pin
    uint8_t *  p_puk_pin
    ssp_err_t(*  p_prov_callback)(sf_cellular_callback_args_t *p_args)
    void(*  p_recv_callback)(sf_cellular_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
    sf_cellular_at_cmd_set_t const  *  p_cmd_set
} sf_cellular_cfg_t
```

11.1.407.1 op_select_mode

```
sf_cellular_op_select_mode_t::op_select_mode
```

Brief description

Cellular Operator selection mode.

11.1.407.2 op

```
sf_cellular_op_t::op
```

Brief description

Cellular operator. Valid when operator selection mode is manual.

11.1.407.3 num_pref_ops

```
uint16_t sf_cellular_cfg_t::num_pref_ops
```

Brief description

Number of preferred operators in the pref_ops array.

11.1.407.4 pref_ops

```
sf_cellular_op_t::pref_ops
```

Brief description

Array of structures describing preferred operators.

11.1.407.5 tz_upd_mode

```
sf_cellular_timezone_update_mode_t::tz_upd_mode
```

Brief description

TimeZone update mode policy.

11.1.407.6 p_sim_pin

```
uint8_t* sf_cellular_cfg_t::p_sim_pin
```

Brief description

SIM Pin.

11.1.407.7 p_puk_pin

```
uint8_t* sf_cellular_cfg_t::p_puk_pin
```

Brief description

PUK Pin.

11.1.407.8 p_prov_callback

```
ssp_err_t(* sf_cellular_cfg_t::p_prov_callback) (sf_cellular_callback_args_t
*p_args)
```

Detailed description

Pointer to provisioning callback function, used in NSAL

11.1.407.9 p_recv_callback

```
void(* sf_cellular_cfg_t::p_recv_callback) (sf_cellular_callback_args_t *p_args)
```

Detailed description

This is the receive callback function used by NetX which will take a data packet from the Cellular module and hand it over to NetX for further processing.

11.1.407.10 p_context

```
void const* sf_cellular_cfg_t::p_context
```

Brief description

User defined context passed into callback function.

11.1.407.11 p_extend

```
void const* sf_cellular_cfg_t::p_extend
```

Brief description

Instance specific configuration.

11.1.407.12 p_cmd_set

```
sf_cellular_at_cmd_set_t::p_cmd_set
```

Brief description

Instance specific command set.

11.1.408 sf_cellular_ctrl_t

```
typedef struct{
    void *  p_driver_handle
} sf_cellular_ctrl_t
```

11.1.408.1 p_driver_handle

```
void* sf_cellular_ctrl_t::p_driver_handle
```

Brief description

Stores information required by underlying Cellular device driver.

11.1.409 sf_cellular_info_t

```
typedef struct{
    uint8_t  mfg_name[SF_CELLULAR_MFG_NAME_LEN]
    uint8_t  chipset[SF_CELLULAR_CHIPSET_LEN]
    uint8_t  fw_version[SF_CELLULAR_FWVERSION_LEN]
    uint8_t  imei[SF_CELLULAR_IMEI_LEN]
    uint16_t  rssi
    uint16_t  ber
    uint8_t  ip_addr[SF_CELLULAR_IP_ADDR_LEN]
} sf_cellular_info_t
```

11.1.409.1 mfg_name

```
uint8_t sf_cellular_info_t::mfg_name[SF_CELLULAR_MFG_NAME_LEN]
```

Brief description

Manufacturer name.

11.1.409.2 chipset

```
uint8_t sf_cellular_info_t::chipset[SF_CELLULAR_CHIPSET_LEN]
```

Brief description

Pointer to string showing Cellular chipset/driver information.

11.1.409.3 fw_version

```
uint8_t sf_cellular_info_t::fw_version[SF_CELLULAR_FWVERSION_LEN]
```

Brief description

Cellular firmware version.

11.1.409.4 imei

```
uint8_t sf_cellular_info_t::imei[SF_CELLULAR_IMEI_LEN]
```

Brief description

IMEI number.

11.1.409.5 rssi

```
uint16_t sf_cellular_info_t::rssi
```

Brief description

Received signal strength indication.

11.1.409.6 ber

`uint16_t sf_cellular_info_t::ber`

Brief description

Bit rate error.

11.1.409.7 ip_addr

`uint8_t sf_cellular_info_t::ip_addr[SF_CELLULAR_IP_ADDR_LEN]`

Brief description

IP address.

11.1.410 sf_cellular_instance_t

```
typedef struct{
    sf_cellular_ctrl_t *  p_ctrl
    sf_cellular_cfg_t const  *  p_cfg
    sf_cellular_api_t const  *  p_api
} sf_cellular_instance_t
```

11.1.410.1 p_ctrl

`sf_cellular_ctrl_t::p_ctrl`

Brief description

Pointer to the control structure for this instance.

11.1.410.2 p_cfg

`sf_cellular_cfg_t::p_cfg`

Brief description

Pointer to the configuration structure for this instance.

11.1.410.3 p_api

`sf_cellular_api_t::p_api`

Brief description

Pointer to the API structure for this instance.

11.1.411 sf_cellular_network_status_t

```
typedef struct{
    uint16_t  country_code
    uint16_t  operator_code
    int16_t  rssi
    uint8_t  cid[SF_CELLULAR_CID_LEN]
    uint8_t  imsi[SF_CELLULAR_IMSI_LEN]
    uint8_t  op_name[SF_CELLULAR_MAX_OPERATOR_NAME_LEN]
    uint8_t  service_domain
    uint8_t  active_band
} sf_cellular_network_status_t
```

11.1.411.1 country_code

```
uint16_t sf_cellular_network_status_t::country_code
```

Brief description

Country code.

11.1.411.2 operator_code

```
uint16_t sf_cellular_network_status_t::operator_code
```

Brief description

Operator code.

11.1.411.3 rssi

```
int16_t sf_cellular_network_status_t::rssi
```

Brief description

RSSI.

11.1.411.4 cid

```
uint8_t sf_cellular_network_status_t::cid[SF_CELLULAR_CID_LEN]
```

Brief description

Cell ID.

11.1.411.5 imsi

```
uint8_t sf_cellular_network_status_t::imsi[SF_CELLULAR_IMSI_LEN]
```

Brief description

IMSI.

11.1.411.6 op_name

```
uint8_t sf_cellular_network_status_t::op_name[SF_CELLULAR_MAX_OPERATOR_NAME_LEN]
```

Brief description

Operator name.

11.1.411.7 service_domain

```
uint8_t sf_cellular_network_status_t::service_domain
```

Brief description

Service Domain.

11.1.411.8 active_band

```
uint8_t sf_cellular_network_status_t::active_band
```

Brief description

Active Band.

11.1.412 sf_cellular_nsal_cfg_t

```
typedef struct{
    uint8_t *  p_ppp_stack
    uint32_t  ppp_stack_size
    UINT  priority
    NX_PPP *  p_ppp
    NX_IP *  p_ip
    NX_PACKET_POOL *  p_ppp_packet_pool
    void(*  p_ppp_invalid_packet_cb)(NX_PACKET *p_packet_ptr)
    void(*  p_ppp_send_byte)(UCHAR byte)
    void(*  p_link_down_cb)(NX_PPP *p_ppp_ptr)
    void(*  p_link_up_cb)(NX_PPP *p_ppp_ptr)
    sf_cellular_auth_type_t  auth_type
    UINT(*  p_chap_get_challenge_cb)(CHAR *p_rand_value, CHAR *p_id, CHAR
*p_name)
    UINT(*  p_chap_get_responder_cb)(CHAR *p_system, CHAR *p_name, CHAR
*p_secret)
```

```
    UINT(*  p_chap_get_verify_cb)(CHAR *p_system, CHAR *p_name, CHAR *p_secret)
    UINT(*  p_pap_generate_login)(CHAR *p_name, CHAR *p_password)
    UINT(*  p_pap_verify_login)(CHAR *p_name, CHAR *p_password)
    uint32_t  local_ip
    uint32_t  peer_ip
    void const  *  p_extend
} sf_cellular_nsal_cfg_t
```

11.1.412.1 p_ppp_stack

```
uint8_t* sf_cellular_nsal_cfg_t::p_ppp_stack
```

Brief description

PPP Stack.

11.1.412.2 ppp_stack_size

```
uint32_t sf_cellular_nsal_cfg_t::ppp_stack_size
```

Brief description

PPP Stack size.

11.1.412.3 priority

```
UINT sf_cellular_nsal_cfg_t::priority
```

Brief description

PPP Thread Priority.

11.1.412.4 p_ppp

```
NX_PPP* sf_cellular_nsal_cfg_t::p_ppp
```

Brief description

NetX PPP Interface.

11.1.412.5 p_ip

```
NX_IP* sf_cellular_nsal_cfg_t::p_ip
```

Brief description

NetX IP Interface.

11.1.412.6 p_ppp_packet_pool

```
NX_PACKET_POOL* sf_cellular_nsal_cfg_t::p_ppp_packet_pool
```

Brief description

NetX Packet pool for internal.

11.1.412.7 p_ppp_invalid_packet_cb

```
void(* sf_cellular_nsal_cfg_t::p_ppp_invalid_packet_cb) (NX_PACKET *p_packet_ptr)
```

Brief description

Callback handler for Invalid packet.

11.1.412.8 p_ppp_send_byte

```
void(* sf_cellular_nsal_cfg_t::p_ppp_send_byte) (UCHAR byte)
```

Brief description

PPP Send byte callback function.

11.1.412.9 p_link_down_cb

```
void(* sf_cellular_nsal_cfg_t::p_link_down_cb) (NX_PPP *p_ppp_ptr)
```

Brief description

PPP Link down notification callback.

Detailed description

Link Notification callback function

11.1.412.10 p_link_up_cb

```
void(* sf_cellular_nsal_cfg_t::p_link_up_cb) (NX_PPP *p_ppp_ptr)
```

Brief description

PPP Link up notification callback.

11.1.412.11 auth_type

```
sf_cellular_auth_type_t::auth_type
```

Brief description

Authentication Type.

11.1.412.12 p_chap_get_challenge_cb

```
UINT(* sf_cellular_nsal_cfg_t::p_chap_get_challenge_cb) (CHAR *p_rand_value, CHAR
*p_id, CHAR *p_name)
```

Brief description

Get challenge notification callback.

Detailed description

CHAP Callback Function

11.1.412.13 p_chap_get_responder_cb

```
UINT(* sf_cellular_nsal_cfg_t::p_chap_get_responder_cb) (CHAR *p_system, CHAR
*p_name, CHAR *p_secret)
```

Brief description

Get Responder notification callback.

11.1.412.14 p_chap_get_verify_cb

```
UINT(* sf_cellular_nsal_cfg_t::p_chap_get_verify_cb) (CHAR *p_system, CHAR
*p_name, CHAR *p_secret)
```

Brief description

Get Chap verification callback.

11.1.412.15 p_pap_generate_login

```
UINT(* sf_cellular_nsal_cfg_t::p_pap_generate_login) (CHAR *p_name, CHAR
*p_password)
```

Brief description

PAP Authentication generate login callback.

Detailed description

PAP Callback Function

11.1.412.16 p_pap_verify_login

```
UINT(* sf_cellular_nsal_cfg_t::p_pap_verify_login) (CHAR *p_name, CHAR
*p_password)
```

Brief description

PAP authentication verification callback.

11.1.412.17 local_ip

```
uint32_t sf_cellular_nsal_cfg_t::local_ip
```

Brief description

Local IP Address.

11.1.412.18 peer_ip

```
uint32_t sf_cellular_nsal_cfg_t::peer_ip
```

Brief description

Peer IP Address.

11.1.412.19 p_extend

```
void const* sf_cellular_nsal_cfg_t::p_extend
```

Brief description

Instance specific configuration.

11.1.413 sf_cellular_op_t

```
typedef struct{
    sf_cellular_op_name_format_t  op_name_format
    uint8_t  op_name[SF_CELLULAR_MAX_OPERATOR_NAME_LEN]
} sf_cellular_op_t
```

11.1.413.1 op_name_format

```
sf_cellular_op_name_format_t::op_name_format
```

Brief description

Cellular operator name format.

11.1.413.2 op_name

```
uint8_t sf_cellular_op_t::op_name[SF_CELLULAR_MAX_OPERATOR_NAME_LEN]
```

Brief description

Cellular operator name.

11.1.414 sf_cellular_provisioning_t

```
typedef struct{
    uint8_t  apn[SF_CELLULAR_MAX_STRING_LEN]
    sf_cellular_auth_type_t  auth_type
    uint8_t  username[SF_CELLULAR_MAX_STRING_LEN]
    uint8_t  password[SF_CELLULAR_MAX_STRING_LEN]
    sf_cellular_airplane_mode_t  airplane_mode
    uint8_t  context_id
    sf_cellular_pdp_type_t  pdp_type
} sf_cellular_provisioning_t
```

11.1.414.1 apn

```
uint8_t sf_cellular_provisioning_t::apn[SF_CELLULAR_MAX_STRING_LEN]
```

Brief description

Access Point Name.

11.1.414.2 auth_type

```
sf_cellular_auth_type_t::auth_type
```

Brief description

Authentication type: PAP/CHAP.

11.1.414.3 username

```
uint8_t sf_cellular_provisioning_t::username[SF_CELLULAR_MAX_STRING_LEN]
```

Brief description

User name used for authentication.

11.1.414.4 password

```
uint8_t sf_cellular_provisioning_t::password[SF_CELLULAR_MAX_STRING_LEN]
```

Brief description

Password used for authentication.

11.1.414.5 airplane_mode

```
sf_cellular_airplane_mode_t::airplane_mode
```

Brief description

Airplane mode.

11.1.414.6 context_id

`uint8_t sf_cellular_provisioning_t::context_id`

Brief description

Context ID to be used for connection.

11.1.414.7 pdp_type

`sf_cellular_pdp_type_t::pdp_type`

Brief description

PDP Type for Context.

11.1.415 sf_cellular_socket_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_cellular_socket_ctrl_t *p_ctrl,
sf_cellular_socket_cfg_t const *const p_cfg)
    ssp_err_t(*  close)(sf_cellular_socket_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_cellular_socket_api_t
```

11.1.416 sf_cellular_socket_cfg_t

```
typedef struct{
    sf_cellular_instance_t *  p_lower_lvl_cellular
    void *  p_extend
} sf_cellular_socket_cfg_t
```

11.1.416.1 p_lower_lvl_cellular

`sf_cellular_instance_t::p_lower_lvl_cellular`

Brief description

Pointer to SF on-chip stack instance.

11.1.416.2 p_extend

`void* sf_cellular_socket_cfg_t::p_extend`

Brief description

Extended configuration.

11.1.417 sf_cellular_socket_ctrl_t

```
typedef struct{
    sf_cellular_instance_t *  p_lower_lvl_cellular
} sf_cellular_socket_ctrl_t
```

11.1.417.1 p_lower_lvl_cellular

sf_cellular_instance_t::p_lower_lvl_cellular

Brief description

low level cellular interface

11.1.418 sf_cellular_socket_instance_t

```
typedef struct{
    sf_cellular_socket_ctrl_t *  p_ctrl
    sf_cellular_socket_cfg_t const  *  p_cfg
    sf_cellular_socket_api_t const  *  p_api
} sf_cellular_socket_instance_t
```

11.1.418.1 p_ctrl

sf_cellular_socket_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.418.2 p_cfg

sf_cellular_socket_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.418.3 p_api

sf_cellular_socket_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.419 sf_cellular_stats_t

```
typedef struct{
    uint32_t  rx_bytes
    uint32_t  tx_bytes
    uint32_t  rx_err
    uint32_t  tx_err
} sf_cellular_stats_t
```

11.1.419.1 rx_bytes

```
uint32_t sf_cellular_stats_t::rx_bytes
```

Brief description

Bytes received successfully.

11.1.419.2 tx_bytes

```
uint32_t sf_cellular_stats_t::tx_bytes
```

Brief description

Bytes transmitted successfully.

11.1.419.3 rx_err

```
uint32_t sf_cellular_stats_t::rx_err
```

Brief description

Bytes receive errors.

11.1.419.4 tx_err

```
uint32_t sf_cellular_stats_t::tx_err
```

Brief description

Bytes transmit errors.

11.1.420 sf_comms_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_comms_ctrl_t *const p_ctrl, sf_comms_cfg_t const *const
p_cfg)
    ssp_err_t(*  close)(sf_comms_ctrl_t *const p_ctrl)
    ssp_err_t(*  read)(sf_comms_ctrl_t *const p_ctrl, uint8_t *const p_dest,
uint32_t const bytes, UINT const timeout)
```

```
    ssp_err_t(*  write)(sf_comms_ctrl_t *const p_ctrl, uint8_t const *const
p_src, uint32_t const bytes, UINT const timeout)
    ssp_err_t(*  lock)(sf_comms_ctrl_t *const p_ctrl, sf_comms_lock_t lock_type,
UINT timeout)
    ssp_err_t(*  unlock)(sf_comms_ctrl_t *const p_ctrl, sf_comms_lock_t
lock_type)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_comms_api_t
```

11.1.421 sf_comms_cfg_t

```
typedef struct{
    void const  *  p_extend
} sf_comms_cfg_t
```

11.1.421.1 p_extend

```
void const* sf_comms_cfg_t::p_extend
```

Brief description

Pointer to lower level communications control structure.

11.1.422 sf_comms_instance_t

```
typedef struct{
    sf_comms_ctrl_t *  p_ctrl
    sf_comms_cfg_t const  *  p_cfg
    sf_comms_api_t const  *  p_api
} sf_comms_instance_t
```

11.1.422.1 p_ctrl

```
sf_comms_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.422.2 p_cfg

```
sf_comms_cfg_t::p_cfg
```

Brief description

Pointer to the configuration structure for this instance.

11.1.422.3 p_api

sf_comms_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.423 sf_comms_telnet_cfg_t

```
typedef struct{
    NX_TELNET_SERVER *  p_telnet_server
    CHAR *  p_telnet_server_name
    NX_IP *  p_ip
    VOID *  p_stack
    ULONG  stack_size
} sf_comms_telnet_cfg_t
```

11.1.423.1 p_telnet_server

NX_TELNET_SERVER* ::p_telnet_server

11.1.423.2 p_telnet_server_name

CHAR* ::p_telnet_server_name

11.1.423.3 p_ip

NX_IP* ::p_ip

11.1.423.4 p_stack

VOID* ::p_stack

11.1.423.5 stack_size

ULONG ::stack_size

11.1.424 sf_comms_telnet_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    NX_PACKET *  p_current_packet
    uint32_t  packet_index
    TX_MUTEX  mutex[2]
    NX_PACKET_POOL  pool
```

```
    uint8_t  pool_memory[SF_COMMS_TELNET_PACKET_POOL_MEMORY_SIZE_BYTES]
    NX_TELNET_SERVER *  p_telnet_server
    UINT  logical_connection
    TX_EVENT_FLAGS_GROUP  available
    TX_QUEUE  queue
    uint8_t  queue_memory[SF_COMMS_TELNET_QUEUE_MEMORY_SIZE_BYTES]
} sf_comms_telnet_instance_ctrl_t
```

11.1.424.1 open

```
uint32_t ::open
```

11.1.424.2 p_current_packet

```
NX_PACKET* ::p_current_packet
```

11.1.424.3 packet_index

```
uint32_t ::packet_index
```

11.1.424.4 mutex

```
TX_MUTEX ::mutex[2]
```

11.1.424.5 pool

```
NX_PACKET_POOL ::pool
```

11.1.424.6 pool_memory

```
uint8_t ::pool_memory[SF_COMMS_TELNET_PACKET_POOL_MEMORY_SIZE_BYTES]
```

11.1.424.7 p_telnet_server

```
NX_TELNET_SERVER* ::p_telnet_server
```

11.1.424.8 logical_connection

```
UINT ::logical_connection
```

11.1.424.9 available

```
TX_EVENT_FLAGS_GROUP ::available
```

Brief description

Flag to tell if this connection is available or not.

11.1.424.10 queue

```
TX_QUEUE ::queue
```

Brief description

Queue of received bytes.

11.1.424.11 queue_memory

```
uint8_t ::queue_memory[SF_COMMS_TELNET_QUEUE_MEMORY_SIZE_BYTES]
```

11.1.425 sf_console_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_console_ctrl_t *const p_ctrl, sf_console_cfg_t const
*const p_cfg)
    ssp_err_t(*  close)(sf_console_ctrl_t *const p_ctrl)
    ssp_err_t(*  prompt)(sf_console_ctrl_t *const p_ctrl, sf_console_menu_t const
*const p_menu, UINT const timeout)
    ssp_err_t(*  parse)(sf_console_ctrl_t *const p_ctrl, sf_console_menu_t const
*const p_cmd_list, uint8_t const *const p_input, uint32_t const bytes)
    ssp_err_t(*  read)(sf_console_ctrl_t *const p_ctrl, uint8_t *const p_dest,
uint32_t const bytes, uint32_t const timeout)
    ssp_err_t(*  write)(sf_console_ctrl_t *const p_ctrl, uint8_t const *const
p_src, uint32_t const timeout)
    ssp_err_t(*  argumentFind)(uint8_t const *const p_arg, uint8_t const *const
p_str, int32_t *const p_index, int32_t *const p_data)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_console_api_t
```

11.1.426 sf_console_callback_args_t

```
typedef struct{
    sf_console_ctrl_t *  p_ctrl
    uint8_t const  *  p_remaining_string
    uint8_t const  *  context
    uint32_t  bytes
} sf_console_callback_args_t
```

11.1.426.1 p_ctrl

```
sf_console_ctrl_t::p_ctrl
```

Brief description

Pointer to console that received the command that caused this callback.

11.1.426.2 p_remaining_string

```
uint8_t const* sf_console_callback_args_t::p_remaining_string
```

Brief description

String remaining after parsing command.

11.1.426.3 context

```
uint8_t const* sf_console_callback_args_t::context
```

Brief description

Pointer to user provided data.

11.1.426.4 bytes

```
uint32_t sf_console_callback_args_t::bytes
```

Brief description

The number of bytes remaining in the input string.

11.1.427 sf_console_cfg_t

```
typedef struct{
    sf_comms_instance_t const  *  p_comms
    sf_console_menu_t const  *  p_initial_menu
    bool  echo
    bool  autostart
} sf_console_cfg_t
```

11.1.427.1 p_comms

```
sf_comms_instance_t::p_comms
```

Brief description

Pointer to communications driver instance.

11.1.427.2 p_initial_menu

```
sf_console_menu_t::p_initial_menu
```

Brief description

First menu to print during Open.

11.1.427.3 echo

```
bool sf_console_cfg_t::echo
```

Brief description

Whether to echo input commands to transmitter.

11.1.427.4 autostart

```
bool sf_console_cfg_t::autostart
```

Brief description

If true, prompt will occur with p_initial_menu after initialization.

11.1.428 sf_console_command_t

```
typedef struct{
    uint8_t *  command
    uint8_t *  help
    void(*  callback)(sf_console_callback_args_t *p_args)
    void const  *  context
} sf_console_command_t
```

11.1.428.1 command

```
uint8_t* sf_console_command_t::command
```

Brief description

Command string.

11.1.428.2 help

```
uint8_t* sf_console_command_t::help
```

Brief description

Description of command.

11.1.428.3 callback

```
void(* sf_console_command_t::callback) (sf_console_callback_args_t *p_args)
```

Brief description

Callback to call when command is selected.

11.1.428.4 context

```
void const* sf_console_command_t::context
```

Brief description

User provided context passed into callback.

11.1.429 sf_console_instance_ctrl_t

```
typedef struct{
    sf_console_menu_t const  *  p_current_menu
    sf_comms_instance_t const  *  p_comms
    uint8_t  new_line
    bool  echo
    uint8_t  input[SF_CONSOLE_MAX_INPUT_LENGTH]
} sf_console_instance_ctrl_t
```

11.1.429.1 p_current_menu

```
sf_console_menu_t::p_current_menu
```

Brief description

Current menu is stored here.

11.1.429.2 p_comms

```
sf_comms_instance_t::p_comms
```

Brief description

Pointer to communications driver instance.

11.1.429.3 new_line

```
uint8_t ::new_line
```

Brief description

Whether to echo input commands to transmitter.

11.1.429.4 echo

```
bool ::echo
```

Brief description

Whether to echo input commands to transmitter.

11.1.429.5 input

```
uint8_t ::input[SF_CONSOLE_MAX_INPUT_LENGTH]
```

Brief description

Input buffer used to store user input.

11.1.430 sf_console_instance_t

```
typedef struct{
    sf_console_ctrl_t *  p_ctrl
    sf_console_cfg_t const  *  p_cfg
    sf_console_api_t const  *  p_api
} sf_console_instance_t
```

11.1.430.1 p_ctrl

```
sf_console_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.430.2 p_cfg

```
sf_console_cfg_t::p_cfg
```

Brief description

Pointer to the configuration structure for this instance.

11.1.430.3 p_api

```
sf_console_api_t::p_api
```

Brief description

Pointer to the API structure for this instance.

11.1.431 sf_console_menu_t

```
typedef struct{
    struct st_sf_console_menu const  *  menu_prev
    uint8_t const  *  menu_name
```

```
    uint32_t  num_commands
    sf_console_command_t const  *  command_list
} sf_console_menu_t
```

11.1.431.1 menu_prev

```
struct st_sf_console_menu const* sf_console_menu_t::menu_prev
```

Brief description

Previous menu.

11.1.431.2 menu_name

```
uint8_t const* sf_console_menu_t::menu_name
```

Brief description

Menu name, used as a prompt.

11.1.431.3 num_commands

```
uint32_t sf_console_menu_t::num_commands
```

Brief description

Number of commands in this menu.

11.1.431.4 command_list

```
sf_console_command_t::command_list
```

Brief description

Pointer to an array of commands of length num_commands.

11.1.432 sf_crypto_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_crypto_ctrl_t *const p_ctrl, sf_crypto_cfg_t const
*const p_cfg)
    ssp_err_t(*  close)(sf_crypto_ctrl_t *const p_ctrl)
    ssp_err_t(*  lock)(sf_crypto_ctrl_t *const p_ctrl)
    ssp_err_t(*  unlock)(sf_crypto_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
    ssp_err_t(*  statusGet)(sf_crypto_ctrl_t *const p_ctrl, sf_crypto_state_t
*p_status)
} sf_crypto_api_t
```

11.1.433 sf_crypto_callback_args_t

```
typedef struct{
    sf_crypto_event_t  event
    ssp_err_t  error
} sf_crypto_callback_args_t
```

11.1.433.1 event

sf_crypto_event_t::event

Brief description

Event code of the low level hardware.

11.1.433.2 error

::error

Brief description

Error code if SF_CRYPTO_EVENT_ERROR.

11.1.434 sf_crypto_cfg_t

```
typedef struct{
    uint32_t  wait_option
    crypto_instance_t *  p_lower_lvl_crypto
    void const  *  p_extend
    void const  *  p_context
    void *  p_memory_pool
    uint32_t  memory_pool_size
    sf_crypto_close_option_t  close_option
} sf_crypto_cfg_t
```

11.1.434.1 wait_option

uint32_t ::wait_option

Brief description

Wait option for RTOS service calls.

11.1.434.2 p_lower_lvl_crypto

* ::p_lower_lvl_crypto

Brief description

Pointer to a low-level Crypto engine HAL driver instance.

11.1.434.3 p_extend

```
void const* ::p_extend
```

Brief description

Extension parameter for hardware specific settings.

11.1.434.4 p_context

```
void const* ::p_context
```

Brief description

Placeholder for user data.

11.1.434.5 p_memory_pool

```
void* ::p_memory_pool
```

Brief description

Byte pool address.

11.1.434.6 memory_pool_size

```
uint32_t ::memory_pool_size
```

Brief description

Byte pool size.

11.1.434.7 close_option

```
sf_crypto_close_option_t::close_option
```

Brief description

Close option.

11.1.435 sf_crypto_cipher_aes_init_params_t

```
typedef struct{
    sf_crypto_cipher_padding_scheme_t  padding_scheme
    sf_crypto_data_handle_t *  p_iv
```

```
    sf_crypto_data_handle_t *  p_auth_tag
} sf_crypto_cipher_aes_init_params_t
```

11.1.435.1 padding_scheme

sf_crypto_cipher_padding_scheme_t::padding_scheme

11.1.435.2 p_iv

sf_crypto_data_handle_t::p_iv

Brief description

pointer to IV for the AES operation.

11.1.435.3 p_auth_tag

sf_crypto_data_handle_t::p_auth_tag

Detailed description

Pointer to the GCM Authentication Tag. Only tag length of SF_CRYPTO_CIPHER_AES_GCM_TAG_LENGTH_16_BYTES is supported.

11.1.436 sf_crypto_cipher_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_crypto_cipher_ctrl_t *const p_ctrl,
sf_crypto_cipher_cfg_t const *const p_cfg)
    ssp_err_t(*  cipherInit)(sf_crypto_cipher_ctrl_t *const p_ctrl,
sf_crypto_cipher_op_mode_t cipher_operation_mode, sf_crypto_key_t const *const
p_key, sf_crypto_cipher_algorithm_init_params_t *const
p_algorithm_specific_params)
    ssp_err_t(*  cipherUpdate)(sf_crypto_cipher_ctrl_t *const p_ctrl,
sf_crypto_data_handle_t const *const p_data_in, sf_crypto_data_handle_t *const
p_data_out)
    ssp_err_t(*  cipherFinal)(sf_crypto_cipher_ctrl_t *const p_ctrl,
sf_crypto_data_handle_t const *const p_data_in, sf_crypto_data_handle_t *const
p_data_out)
    ssp_err_t(*  cipherAadUpdate)(sf_crypto_cipher_ctrl_t *const p_ctrl,
sf_crypto_data_handle_t const *const p_aad)
    ssp_err_t(*  close)(sf_crypto_cipher_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_crypto_cipher_api_t
```

11.1.437 sf_crypto_cipher_cfg_t

```
typedef struct{
    sf_crypto_key_type_t  key_type
    sf_crypto_key_size_t  key_size
    sf_crypto_cipher_mode_t  cipher_chaining_mode
    sf_crypto_instance_t *  p_lower_lvl_crypto_common
    sf_crypto_trng_instance_t *  p_lower_lvl_crypto_trng
    void const  *  p_extend
} sf_crypto_cipher_cfg_t
```

11.1.437.1 key_type

sf_crypto_key_type_t::key_type

Brief description

Key type for cipher operation.

11.1.437.2 key_size

sf_crypto_key_size_t::key_size

Brief description

Key size for cipher operation.

11.1.437.3 cipher_chaining_mode

sf_crypto_cipher_mode_t::cipher_chaining_mode

Brief description

Chaining mode specified for the cipher operation.

11.1.437.4 p_lower_lvl_crypto_common

sf_crypto_instance_t::p_lower_lvl_crypto_common

Brief description

Pointer to a Crypto Framework common instance.

11.1.437.5 p_lower_lvl_crypto_trng

sf_crypto_trng_instance_t::p_lower_lvl_crypto_trng

Brief description

Pointer to a Crypto Framework TRNG instance.

11.1.437.6 p_extend

```
void const* ::p_extend
```

Brief description

Future extension for hardware specific settings.

11.1.438 sf_crypto_cipher_instance_ctrl_t

```
typedef struct{
    sf_crypto_key_type_t  key_type
    sf_crypto_key_size_t  key_size
    sf_crypto_cipher_mode_t  cipher_chaining_mode
    sf_crypto_cipher_state_t  status
    crypto_algorithm_type_t  cipher_algorithm_type
    sf_crypto_instance_ctrl_t *  p_lower_lvl_fwk_common_ctrl
    sf_crypto_api_t *  p_lower_lvl_fwk_common_api
    sf_crypto_trng_instance_ctrl_t *  p_lower_lvl_sf_crypto_trng_ctrl
    sf_crypto_trng_api_t *  p_sf_crypto_trng_api
    void *  p_hal_ctrl
    void *  p_hal_api
    void *  p_cipher_context_buffer
} sf_crypto_cipher_instance_ctrl_t
```

11.1.438.1 key_type

```
sf_crypto_key_type_t::key_type
```

Brief description

Key type.

11.1.438.2 key_size

```
sf_crypto_key_size_t::key_size
```

Brief description

Key size.

11.1.438.3 cipher_chaining_mode

```
sf_crypto_cipher_mode_t::cipher_chaining_mode
```

Brief description

Chaining mode specified for the cipher operation.

11.1.438.4 status

sf_crypto_cipher_state_t::status

Brief description

Module status.

11.1.438.5 cipher_algorithm_type

::cipher_algorithm_type

Brief description

Cipher algorithm for the keys selected.

11.1.438.6 p_lower_lvl_fwk_common_ctrl

sf_crypto_instance_ctrl_t::p_lower_lvl_fwk_common_ctrl

Brief description

Pointer to the Crypto Framework Common instance.

11.1.438.7 p_lower_lvl_fwk_common_api

sf_crypto_api_t::p_lower_lvl_fwk_common_api

Brief description

Pointer to the Crypto Framework Common API.

11.1.438.8 p_lower_lvl_sf_crypto_trng_ctrl

* ::p_lower_lvl_sf_crypto_trng_ctrl

Brief description

Pointer to the Crypto TRNG API.

11.1.438.9 p_sf_crypto_trng_api

sf_crypto_trng_api_t::p_sf_crypto_trng_api

Brief description

Pointer to the Crypto TRNG API.

11.1.438.10 p_hal_ctrl

void* ::p_hal_ctrl

Brief description

Pointer to HAL control structure for the Cipher operation.

11.1.438.11 p_hal_api

```
void* ::p_hal_api
```

Brief description

Pointer to HAL API structure for the cipher algorithm.

11.1.438.12 p_cipher_context_buffer

```
void* ::p_cipher_context_buffer
```

Brief description

Cipher context buffer after DWORD alignment.

11.1.439 sf_crypto_cipher_instance_t

```
typedef struct{
    sf_crypto_cipher_ctrl_t *  p_ctrl
    sf_crypto_cipher_cfg_t const  *  p_cfg
    sf_crypto_cipher_api_t const  *  p_api
} sf_crypto_cipher_instance_t
```

11.1.439.1 p_ctrl

```
sf_crypto_cipher_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.439.2 p_cfg

```
sf_crypto_cipher_cfg_t::p_cfg
```

Brief description

Pointer to the configuration structure for this instance.

11.1.439.3 p_api

```
sf_crypto_cipher_api_t::p_api
```

Brief description

Pointer to the API structure for this instance.

11.1.440 sf_crypto_cipher_rsa_init_params_t

```
typedef struct{
    sf_crypto_cipher_padding_scheme_t  padding_scheme
} sf_crypto_cipher_rsa_init_params_t
```

11.1.440.1 padding_scheme

sf_crypto_cipher_padding_scheme_t::padding_scheme

11.1.441 sf_crypto_data_handle_t

```
typedef struct{
    uint8_t *  p_data
    uint32_t  data_length
} sf_crypto_data_handle_t
```

11.1.441.1 p_data

uint8_t* ::p_data

Brief description

Pointer to data.

11.1.441.2 data_length

uint32_t ::data_length

Brief description

The length of data pointed by p_data.

11.1.442 sf_crypto_hash_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_crypto_hash_ctrl_t *const p_ctrl, sf_crypto_hash_cfg_t
const *const p_cfg)
    ssp_err_t(*  close)(sf_crypto_hash_ctrl_t *const p_ctrl)
    ssp_err_t(*  hashInit)(sf_crypto_hash_ctrl_t *const p_ctrl)
    ssp_err_t(*  hashUpdate)(sf_crypto_hash_ctrl_t *const p_ctrl,
sf_crypto_data_handle_t const *const p_data_in)
    ssp_err_t(*  hashFinal)(sf_crypto_hash_ctrl_t *const p_ctrl,
```

```
sf_crypto_data_handle_t *const p_msg_digest, uint32_t *p_size)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_crypto_hash_api_t
```

11.1.443 sf_crypto_hash_callback_args_t

```
typedef struct{
    ssp_err_t  error
} sf_crypto_hash_callback_args_t
```

11.1.443.1 error

::error

Brief description

Error code if SF_CRYPTO_EVENT_ERROR.

11.1.444 sf_crypto_hash_cfg_t

```
typedef struct{
    sf_crypto_hash_type_t  hash_type
    sf_crypto_instance_t *  p_lower_lvl_crypto_common
    hash_instance_t *  p_lower_lvl_instance
    void *  p_extend
} sf_crypto_hash_cfg_t
```

11.1.444.1 hash_type

sf_crypto_hash_type_t::hash_type

Brief description

HASH algorithm type.

11.1.444.2 p_lower_lvl_crypto_common

sf_crypto_instance_t::p_lower_lvl_crypto_common

Brief description

Pointer to a Crypto Framework common instance.

11.1.444.3 p_lower_lvl_instance

hash_instance_t::p_lower_lvl_instance

Brief description

pointer to HASH lower-level module instance

11.1.444.4 p_extend

```
void* ::p_extend
```

Brief description

Pointer to an optional configuration for Crypto HAL module.

11.1.445 sf_crypto_hash_context_t

```
typedef struct{
    uint8_t *  p_message_digest
    uint8_t *  p_message_digest_org
    uint8_t *  p_message_buffer
    uint8_t *  p_message_buffer_org
    uint64_t  message_bytes
    uint32_t  message_bytes_buffered
} sf_crypto_hash_context_t
```

11.1.445.1 p_message_digest

```
uint8_t* ::p_message_digest
```

Brief description

Intermediate digest stored buffer - WORD aligned.

11.1.445.2 p_message_digest_org

```
uint8_t* ::p_message_digest_org
```

Brief description

Originally allocated buffer - may not be WORD aligned.

11.1.445.3 p_message_buffer

```
uint8_t* ::p_message_buffer
```

Brief description

Intermediate message data stored buffer - - WORD aligned.

11.1.445.4 p_message_buffer_org

```
uint8_t* ::p_message_buffer_org
```

Brief description

IOriginally allocated buffer - may not be WORD aligned.

11.1.445.5 message_bytes

```
uint64_t ::message_bytes
```

Brief description

Number of bytes from user data processed.

11.1.445.6 message_bytes_buffered

```
uint32_t ::message_bytes_buffered
```

Brief description

Number of bytes buffered in the message data stored buffer.

11.1.446 sf_crypto_hash_instance_t

```
typedef struct{
    sf_crypto_hash_ctrl_t *  p_ctrl
    sf_crypto_hash_cfg_t *  p_cfg
    sf_crypto_hash_api_t const  *  p_api
} sf_crypto_hash_instance_t
```

11.1.446.1 p_ctrl

```
sf_crypto_hash_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.446.2 p_cfg

```
sf_crypto_hash_cfg_t::p_cfg
```

Brief description

Pointer to the configuration structure for this instance.

11.1.446.3 p_api

sf_crypto_hash_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.447 sf_crypto_instance_ctrl_t

```
typedef struct{
    sf_crypto_state_t  status
    TX_MUTEX  mutex
    TX_SEMAPHORE  semaphore
    TX_BYTE_POOL  byte_pool
    uint32_t  wait_option
    uint32_t  open_counter
    void *  p_lower_lvl_crypto
    void(*  p_callback)(sf_crypto_callback_args_t *p_args)
    void *  p_context
    sf_crypto_close_option_t  close_option
} sf_crypto_instance_ctrl_t
```

11.1.447.1 status

sf_crypto_state_t::status

Brief description

Module status.

11.1.447.2 mutex

TX_MUTEX ::mutex

Brief description

Mutex used in the Crypto Framework.

11.1.447.3 semaphore

TX_SEMAPHORE ::semaphore

Brief description

Semaphore used in the Crypto Framework (Reserve)

11.1.447.4 byte_pool

```
TX_BYTE_POOL ::byte_pool
```

Brief description

Byte pool used in the Crypto Framework.

11.1.447.5 wait_option

```
uint32_t ::wait_option
```

Brief description

Wait time option used for RTOS service calls.

11.1.447.6 open_counter

```
uint32_t ::open_counter
```

Brief description

Counter to keep the number of SF_CRYPTO_XXX opened.

11.1.447.7 p_lower_lvl_crypto

```
void* ::p_lower_lvl_crypto
```

Brief description

Pointer to a low-level Crypto engine HAL driver instance.

11.1.447.8 p_callback

```
void(* ::p_callback) ( *p_args)
```

Brief description

Pointer to callback function.

11.1.447.9 p_context

```
void* ::p_context
```

Brief description

Pointer to a context.

11.1.447.10 close_option

```
sf_crypto_close_option_t::close_option
```

Brief description

Close option.

11.1.448 sf_crypto_instance_t

```
typedef struct{
    sf_crypto_ctrl_t *  p_ctrl
    sf_crypto_cfg_t const  *  p_cfg
    sf_crypto_api_t const  *  p_api
} sf_crypto_instance_t
```

11.1.448.1 p_ctrl

sf_crypto_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.448.2 p_cfg

sf_crypto_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.448.3 p_api

sf_crypto_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.449 sf_crypto_key_installation_instance_ctrl_t

```
typedef struct{
    sf_crypto_key_installation_state_t  status
    sf_crypto_key_type_t  key_type
    sf_crypto_key_size_t  key_size
    sf_crypto_instance_ctrl_t *  p_lower_lvl_common_ctrl
    sf_crypto_api_t *  p_lower_lvl_common_api
    void *  p_lower_lvl_instance
} sf_crypto_key_installation_instance_ctrl_t
```

11.1.449.1 status

::status

Brief description

Module status.

11.1.449.2 key_type

sf_crypto_key_type_t::key_type

Brief description

Type of key to be installed.

11.1.449.3 key_size

sf_crypto_key_size_t::key_size

Brief description

Size of key to be installed.

11.1.449.4 p_lower_lvl_common_ctrl

sf_crypto_instance_ctrl_t::p_lower_lvl_common_ctrl

Brief description

Pointer to the Crypto Framework Common instance.

11.1.449.5 p_lower_lvl_common_api

sf_crypto_api_t::p_lower_lvl_common_api

Brief description

Pointer to the Crypto Framework Common instance.

11.1.449.6 p_lower_lvl_instance

void* ::p_lower_lvl_instance

Brief description

Pointer to HAL KeyInstall Crypto module instance structure.

11.1.450 sf_crypto_key_instance_ctrl_t

```
typedef struct{
    sf_crypto_key_state_t  status
    sf_crypto_key_type_t  key_type
    sf_crypto_key_size_t  key_size
    sf_crypto_data_handle_t  domain_params
    sf_crypto_data_handle_t  generator_point
    sf_crypto_instance_ctrl_t *  p_fwk_common_ctrl
    sf_crypto_api_t *  p_fwk_common_api
    void *  p_hal_ctrl
    void *  p_hal_api
} sf_crypto_key_instance_ctrl_t
```

11.1.450.1 status

::status

Brief description

Module status.

11.1.450.2 key_type

sf_crypto_key_type_t::key_type

Brief description

Key type.

11.1.450.3 key_size

sf_crypto_key_size_t::key_size

Brief description

Key size.

11.1.450.4 domain_params

sf_crypto_data_handle_t::domain_params

Brief description

Domain parameters structure with pointer to data and length.

11.1.450.5 generator_point

sf_crypto_data_handle_t::generator_point

Brief description

Generator Point structure with pointer to data and length.

11.1.450.6 p_fwk_common_ctrl

`sf_crypto_instance_ctrl_t::p_fwk_common_ctrl`

Brief description

Pointer to the Crypto Framework Common instance.

11.1.450.7 p_fwk_common_api

`sf_crypto_api_t::p_fwk_common_api`

Brief description

Pointer to the Crypto Framework Common instance.

11.1.450.8 p_hal_ctrl

`void* ::p_hal_ctrl`

Brief description

pointer to Crypto module control structure

11.1.450.9 p_hal_api

`void* ::p_hal_api`

Brief description

pointer to Crypto module API structure

11.1.451 sf_crypto_signature_context_t

```
typedef struct{
    sf_crypto_signature_mode_t  operation_mode
    sf_crypto_signature_algorithm_init_params_t *  p_aglorithm_specific_params
    sf_crypto_data_handle_t  buffer
    uint8_t *  p_key_data
    uint32_t  key_data_length
} sf_crypto_signature_context_t
```

11.1.451.1 operation_mode

`::operation_mode`

Detailed description

Operating mode. (Sign / Verify operation)

11.1.451.2 p_aglorithm_specific_params

```
* ::p_aglorithm_specific_params
```

Detailed description

Algorithm specific parameters. OR hold formatted input data.

11.1.451.3 buffer

```
sf_crypto_data_handle_t::buffer
```

Detailed description

Internal buffer to format input data

11.1.451.4 p_key_data

```
uint8_t* ::p_key_data
```

Detailed description

Buffer to hold private key in case of Sign Operations. OR. Buffer to hold public key in case of Verify Operations.

11.1.451.5 key_data_length

```
uint32_t ::key_data_length
```

Detailed description

Length of key data.

11.1.452 sf_crypto_signature_instance_ctrl_t

```
typedef struct{
    sf_crypto_signature_state_t  status
    sf_crypto_key_type_t  key_type
    sf_crypto_key_size_t  key_size
    sf_crypto_signature_operation_state_t  operation_state
    sf_crypto_signature_context_t  operation_context
    sf_crypto_instance_ctrl_t *  p_lower_lvl_common_ctrl
    sf_crypto_api_t *  p_lower_lvl_common_api
    void *  p_hal_ctrl
    void *  p_hal_api
    sf_crypto_hash_instance_t *  p_lower_lvl_sf_crypto_hash
} sf_crypto_signature_instance_ctrl_t
```

11.1.452.1 status

sf_crypto_signature_state_t::status

Brief description

Module status.

11.1.452.2 key_type

sf_crypto_key_type_t::key_type

Brief description

Key type.

11.1.452.3 key_size

sf_crypto_key_size_t::key_size

Brief description

Key size.

11.1.452.4 operation_state

sf_crypto_signature_operation_state_t::operation_state

Brief description

Internal Operation state.

11.1.452.5 operation_context

sf_crypto_signature_context_t::operation_context

Detailed description

Context for sign / verify operations.

11.1.452.6 p_lower_lvl_common_ctrl

sf_crypto_instance_ctrl_t::p_lower_lvl_common_ctrl

Detailed description

Pointer to the Crypto Framework Common instance

11.1.452.7 p_lower_lvl_common_api

sf_crypto_api_t::p_lower_lvl_common_api

Detailed description

Pointer to the Crypto Framework API instance

11.1.452.8 p_hal_ctrl

```
void* ::p_hal_ctrl
```

Detailed description

pointer to Crypto module control structure

11.1.452.9 p_hal_api

```
void* ::p_hal_api
```

Detailed description

pointer to Crypto module API structure

11.1.452.10 p_lower_lvl_sf_crypto_hash

```
sf_crypto_hash_instance_t::p_lower_lvl_sf_crypto_hash
```

Detailed description

pointer to Crypto Framework Hash instance

11.1.453 sf_crypto_trng_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_crypto_trng_ctrl_t *const p_ctrl, sf_crypto_trng_cfg_t
const *const p_cfg)
    ssp_err_t(*  close)(sf_crypto_trng_ctrl_t *const p_ctrl)
    ssp_err_t(*  randomNumberGenerate)(sf_crypto_trng_ctrl_t *const p_ctrl,
sf_crypto_data_handle_t *const p_random_number_buff)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_crypto_trng_api_t
```

11.1.454 sf_crypto_trng_cfg_t

```
typedef struct{
    sf_crypto_instance_t *  p_lower_lvl_common
    trng_instance_t *  p_lower_lvl_instance
    void *  p_extend
} sf_crypto_trng_cfg_t
```

11.1.454.1 p_lower_lvl_common

sf_crypto_instance_t::p_lower_lvl_common

Brief description

Pointer to a Crypto Framework common instance.

11.1.454.2 p_lower_lvl_instance

trng_instance_t::p_lower_lvl_instance

Brief description

Pointer to Crypto TRNG HAL instance.

11.1.454.3 p_extend

void* ::p_extend

Brief description

Pointer to an optional configuration for HW specific settings.

11.1.455 sf_crypto_trng_instance_t

```
typedef struct{
    sf_crypto_trng_ctrl_t *  p_ctrl
    sf_crypto_trng_cfg_t *  p_cfg
    sf_crypto_trng_api_t const  *  p_api
} sf_crypto_trng_instance_t
```

11.1.455.1 p_ctrl

sf_crypto_trng_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.455.2 p_cfg

sf_crypto_trng_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.455.3 p_api

sf_crypto_trng_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.456 sf_el_fx_t

```
typedef struct{
    sf_block_media_instance_t *  p_lower_lvl_block_media
    void *  p_extend
} sf_el_fx_t
```

11.1.456.1 p_lower_lvl_block_media

sf_block_media_instance_t::p_lower_lvl_block_media

Brief description

Lower level block media pointer.

11.1.456.2 p_extend

void* ::p_extend

11.1.457 sf_el_gx_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_el_gx_ctrl_t *const p_ctrl, sf_el_gx_cfg_t const *const
p_cfg)
    ssp_err_t(*  close)(sf_el_gx_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *p_version)
    UINT(*  setup)(GX_DISPLAY *p_display)
    ssp_err_t(*  canvasInit)(sf_el_gx_ctrl_t *const p_ctrl, GX_WINDOW_ROOT
*p_window_root)
} sf_el_gx_api_t
```

11.1.458 sf_el_gx_callback_args_t

```
typedef struct{
    sf_el_gx_device_t  device
    sf_el_gx_event_t  event
    uint32_t  error
} sf_el_gx_callback_args_t
```

11.1.458.1 device

sf_el_gx_device_t::device

Brief description

Device code.

11.1.458.2 event

sf_el_gx_event_t::event

Brief description

Event code of the low level hardware.

11.1.458.3 error

uint32_t sf_el_gx_callback_args_t::error

Brief description

Error code if SF_EL_GX_EVENT_ERROR.

11.1.459 sf_el_gx_cfg_t

```
typedef struct{
    display_instance_t *  p_display_instance
    display_runtime_cfg_t *  p_display_runtime_cfg
    void *  p_canvas
    void *  p_framebuffer_a
    void *  p_framebuffer_b
    void(*  p_callback)(sf_el_gx_callback_args_t *p_args)
    void *  p_context
    void *  p_jpegbuffer
    uint32_t  jpegbuffer_size
    uint16_t  rotation_angle
    void *  p_sf_jpeg_decode_instance
    _Bool  dave2d_buffer_cache_enabled
} sf_el_gx_cfg_t
```

11.1.459.1 p_display_instance

display_instance_t::p_display_instance

Brief description

Pointer to a display instance.

11.1.459.2 p_display_runtime_cfg

```
display_runtime_cfg_t::p_display_runtime_cfg
```

Brief description

Pointer to a runtime display configuration.

11.1.459.3 p_canvas

```
void* sf_el_gx_cfg_t::p_canvas
```

Brief description

Pointer to a canvas(reserved)

11.1.459.4 p_framebuffer_a

```
void* sf_el_gx_cfg_t::p_framebuffer_a
```

Brief description

Pointer to a frame buffer(A)

11.1.459.5 p_framebuffer_b

```
void* sf_el_gx_cfg_t::p_framebuffer_b
```

Brief description

Pointer to a frame buffer(B)

11.1.459.6 p_callback

```
void(* sf_el_gx_cfg_t::p_callback) (sf_el_gx_callback_args_t *p_args)
```

Brief description

Pointer to callback function.

11.1.459.7 p_context

```
void* sf_el_gx_cfg_t::p_context
```

Brief description

Pointer to a context.

11.1.459.8 p_jpegbuffer

```
void* sf_el_gx_cfg_t::p_jpegbuffer
```

Brief description

Pointer to a JPEG work buffer.

11.1.459.9 jpegbuffer_size

```
uint32_t sf_el_gx_cfg_t::jpegbuffer_size
```

Brief description

Size of a JPEG work buffer.

11.1.459.10 rotation_angle

```
uint16_t sf_el_gx_cfg_t::rotation_angle
```

Brief description

Screen rotation angle(0/90/270)

11.1.459.11 p_sf_jpeg_decode_instance

```
void* sf_el_gx_cfg_t::p_sf_jpeg_decode_instance
```

Brief description

Pointer to a JPEG framework instance.

11.1.459.12 dave2d_buffer_cache_enabled

```
_Bool sf_el_gx_cfg_t::dave2d_buffer_cache_enabled
```

Brief description

D/AVE 2D buffer cache enabled/disabled.

11.1.460 sf_el_gx_instance_ctrl_t

```
typedef struct{
    GX_DISPLAY *  p_display
    display_instance_t *  p_display_instance
    display_runtime_cfg_t *  p_display_runtime_cfg
    void *  p_canvas
    void *  p_framebuffer_read
    void *  p_framebuffer_write
    void(*  p_callback)(sf_el_gx_callback_args_t *p_args)
    void *  p_context
    TX_SEMAPHORE  semaphore
    bool  rendering_enable
    bool  display_list_flushed
```

```
    sf_el_gx_state_t  state
    void *  p_jpegbuffer
    uint32_t  jpegbuffer_size
    uint16_t  rotation_angle
    void *  p_sf_jpeg_decode_instance
    _Bool  dave2d_buffer_cache_enabled
} sf_el_gx_instance_ctrl_t
```

11.1.460.1 p_display

```
GX_DISPLAY* ::p_display
```

Brief description

Pointer to the GUIX display context.

11.1.460.2 p_display_instance

```
display_instance_t::p_display_instance
```

Brief description

Pointer to a display instance.

11.1.460.3 p_display_runtime_cfg

```
display_runtime_cfg_t::p_display_runtime_cfg
```

Brief description

Pointer to a runtime display configuration.

11.1.460.4 p_canvas

```
void* ::p_canvas
```

Brief description

Pointer to a canvas(reserved)

11.1.460.5 p_framebuffer_read

```
void* ::p_framebuffer_read
```

Brief description

Pointer to a frame buffer (for displaying)

11.1.460.6 p_framebuffer_write

```
void* ::p_framebuffer_write
```

Brief description

Pointer to a frame buffer (for rendering)

11.1.460.7 p_callback

```
void(* ::p_callback) ( *p_args)
```

Brief description

Pointer to callback function.

11.1.460.8 p_context

```
void* ::p_context
```

Brief description

Pointer to a context.

11.1.460.9 semaphore

```
TX_SEMAPHORE ::semaphore
```

Brief description

Semaphore for the frame buffer flip sync.

11.1.460.10 rendering_enable

```
bool ::rendering_enable
```

Brief description

Sync flag between Rendering and displaying.

11.1.460.11 display_list_flushed

```
bool ::display_list_flushed
```

Brief description

Flag to show the display list is flushed.

11.1.460.12 state

```
sf_el_gx_state_t::state
```

Brief description

State of this module.

11.1.460.13 p_jpegbuffer

```
void* ::p_jpegbuffer
```

Brief description

Pointer to a JPEG work buffer.

11.1.460.14 jpegbuffer_size

```
uint32_t ::jpegbuffer_size
```

Brief description

Size of a JPEG work buffer.

11.1.460.15 rotation_angle

```
uint16_t ::rotation_angle
```

Brief description

Screen rotation angle(0/90/270)

11.1.460.16 p_sf_jpeg_decode_instance

```
void* ::p_sf_jpeg_decode_instance
```

Brief description

Pointer to a JPEG framework instance.

11.1.460.17 dave2d_buffer_cache_enabled

```
_Bool ::dave2d_buffer_cache_enabled
```

Brief description

D/AVE 2D buffer cache enabled/disabled.

11.1.461 sf_el_gx_instance_t

```
typedef struct{
    sf_el_gx_ctrl_t *  p_ctrl
    sf_el_gx_cfg_t const  *  p_cfg
```

```
    sf_el_gx_api_t const  *  p_api
} sf_el_gx_instance_t
```

11.1.461.1 p_ctrl

sf_el_gx_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.461.2 p_cfg

sf_el_gx_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.461.3 p_api

sf_el_gx_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.462 sf_el_nx_comms_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    NX_PACKET *  p_current_packet
    uint32_t  packet_index
    TX_MUTEX  mutex[2]
    NX_PACKET_POOL  pool
    uint8_t  pool_memory[SF_EL_NX_COMMS_PACKET_POOL_MEMORY_SIZE_BYTES]
    NX_IP  ip
    uint8_t  ip_memory[SF_EL_NX_COMMS_IP_MEMORY_SIZE_BYTES]
    uint8_t  arp_memory[SF_EL_NX_COMMS_ARP_MEMORY_SIZE_BYTES]
    NX_TELNET_SERVER  telnet_server
    uint8_t  telnet_server_memory[SF_EL_NX_COMMS_TELNET_SERVER_MEMORY_SIZE_BYTES
]
    UINT  logical_connection
    TX_EVENT_FLAGS_GROUP  available
    TX_QUEUE  queue
    uint8_t  queue_memory[SF_EL_NX_COMMS_QUEUE_MEMORY_SIZE_BYTES]
} sf_el_nx_comms_instance_ctrl_t
```

11.1.462.1 open

```
uint32_t ::open
```

11.1.462.2 p_current_packet

```
NX_PACKET* ::p_current_packet
```

11.1.462.3 packet_index

```
uint32_t ::packet_index
```

11.1.462.4 mutex

```
TX_MUTEX ::mutex[2]
```

11.1.462.5 pool

```
NX_PACKET_POOL ::pool
```

11.1.462.6 pool_memory

```
uint8_t ::pool_memory[SF_EL_NX_COMMS_PACKET_POOL_MEMORY_SIZE_BYTES]
```

11.1.462.7 ip

```
NX_IP ::ip
```

11.1.462.8 ip_memory

```
uint8_t ::ip_memory[SF_EL_NX_COMMS_IP_MEMORY_SIZE_BYTES]
```

11.1.462.9 arp_memory

```
uint8_t ::arp_memory[SF_EL_NX_COMMS_ARP_MEMORY_SIZE_BYTES]
```

11.1.462.10 telnet_server

```
NX_TELNET_SERVER ::telnet_server
```

11.1.462.11 telnet_server_memory

```
uint8_t ::telnet_server_memory[SF_EL_NX_COMMS_TELNET_SERVER_MEMORY_SIZE_BYTES]
```

11.1.462.12 logical_connection

```
UINT ::logical_connection
```

11.1.462.13 available

```
TX_EVENT_FLAGS_GROUP ::available
```

Brief description

Flag to tell if this connection is available or not.

11.1.462.14 queue

```
TX_QUEUE ::queue
```

Brief description

Queue of received bytes.

11.1.462.15 queue_memory

```
uint8_t ::queue_memory[SF_EL_NX_COMMS_QUEUE_MEMORY_SIZE_BYTES]
```

11.1.463 sf_el_nx_comms_on_comms_cfg_t

```
typedef struct{
    uint32_t  ip_address
    uint32_t  subnet_mask
    VOID(*  driver)(NX_IP_DRIVER *driver_req_ptr)
} sf_el_nx_comms_on_comms_cfg_t
```

11.1.463.1 ip_address

```
uint32_t sf_::ip_address
```

11.1.463.2 subnet_mask

```
uint32_t sf_::subnet_mask
```

11.1.463.3 driver

```
VOID(* sf_::driver) (NX_IP_DRIVER *driver_req_ptr)
```

11.1.464 sf_el_ux_comms_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    TX_MUTEX  mutex[2]
    UX_SLAVE_CLASS_CDC_ACM *  p_cdc
    uint32_t  leftover_length
    uint32_t  index
    uint8_t  memory[SF_EL_UX_COMMS_CFG_USB_MEMORY_SIZE_BYTES]
    uint8_t  rx_memory[SF_EL_UX_COMMS_CFG_BUFFER_MAX_LENGTH]
    TX_SEMAPHORE  semaphore
} sf_el_ux_comms_instance_ctrl_t
```

11.1.464.1 open

```
uint32_t ::open
```

11.1.464.2 mutex

```
TX_MUTEX ::mutex
```

11.1.464.3 p_cdc

```
UX_SLAVE_CLASS_CDC_ACM * ::p_cdc
```

11.1.464.4 leftover_length

```
uint32_t ::leftover_length
```

11.1.464.5 index

```
uint32_t ::index
```

11.1.464.6 memory

```
uint8_t ::memory[SF_EL_UX_COMMS_CFG_USB_MEMORY_SIZE_BYTES]
```

11.1.464.7 rx_memory

```
uint8_t ::rx_memory
```

11.1.464.8 semaphore

```
TX_SEMAPHORE ::semaphore
```

11.1.465 sf_external_irq_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_external_irq_ctrl_t *const p_ctrl,
sf_external_irq_cfg_t const *const p_cfg)
    ssp_err_t(*  wait)(sf_external_irq_ctrl_t *const p_ctrl, ULONG const timeout)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
    ssp_err_t(*  close)(sf_external_irq_ctrl_t *const p_ctrl)
} sf_external_irq_api_t
```

11.1.466 sf_external_irq_cfg_t

```
typedef struct{
    external_irq_instance_t const  *  p_lower_lvl_irq
    sf_external_irq_event_t  event
} sf_external_irq_cfg_t
```

11.1.466.1 p_lower_lvl_irq

external_irq_instance_t::p_lower_lvl_irq

Detailed description

All info needed to work with lower layer

11.1.466.2 event

sf_external_irq_event_t::event

Brief description

Select what happens when the external IRQ is triggered.

11.1.467 sf_external_irq_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    TX_MUTEX  mutex
    TX_SEMAPHORE  semaphore
    external_irq_instance_t const  *  p_lower_lvl_irq
    bool  callback_used
} sf_external_irq_instance_ctrl_t
```

11.1.467.1 open

uint32_t ::open

Brief description

Used by driver to check if control block is valid.

11.1.467.2 mutex

```
TX_MUTEX ::mutex
```

Brief description

Mutex used to protect access to lower level driver hardware registers.

11.1.467.3 semaphore

```
TX_SEMAPHORE ::semaphore
```

Brief description

Semaphore used for SF_EXTERNAL_IRQ_Wait.

11.1.467.4 p_lower_lvl_irq

```
external_irq_instance_t::p_lower_lvl_irq
```

Brief description

Pointer to lower level driver instance.

11.1.467.5 callback_used

```
bool ::callback_used
```

Brief description

Used by driver to check if wait can be used.

11.1.468 sf_external_irq_instance_t

```
typedef struct{
    sf_external_irq_ctrl_t *  p_ctrl
    sf_external_irq_cfg_t const  *  p_cfg
    sf_external_irq_api_t const  *  p_api
} sf_external_irq_instance_t
```

11.1.468.1 p_ctrl

```
sf_external_irq_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.468.2 p_cfg

sf_external_irq_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.468.3 p_api

sf_external_irq_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.469 sf_i2c_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_i2c_ctrl_t *p_ctrl, sf_i2c_cfg_t const *const p_cfg)
    ssp_err_t(*  read)(sf_i2c_ctrl_t *const p_ctrl, uint8_t *const p_dest,
uint32_t const bytes, bool const restart, uint32_t const timeout)
    ssp_err_t(*  write)(sf_i2c_ctrl_t *const p_ctrl, uint8_t *const p_src,
uint32_t const bytes, bool const restart, uint32_t const timeout)
    ssp_err_t(*  reset)(sf_i2c_ctrl_t *const p_ctrl, uint32_t const timeout)
    ssp_err_t(*  close)(sf_i2c_ctrl_t *const p_ctrl)
    ssp_err_t(*  lock)(sf_i2c_ctrl_t *const p_ctrl)
    ssp_err_t(*  unlock)(sf_i2c_ctrl_t *const p_ctrl)
    ssp_err_t(*  version)(ssp_version_t *const p_version)
} sf_i2c_api_t
```

11.1.470 sf_i2c_bus_t

```
typedef struct{
    uint8_t  channel
    TX_MUTEX *  p_lock_mutex
    TX_EVENT_FLAGS_GROUP *  p_sync_eventflag
    sf_i2c_ctrl_t **  pp_curr_ctrl
    uint8_t *  p_bus_name
    i2c_api_master_t const  *  p_lower_lvl_api
    uint8_t  device_count
    sf_i2c_ctrl_t **  pp_curr_bus_ctrl
} sf_i2c_bus_t
```

11.1.470.1 channel

```
uint8_t sf_i2c_bus_t::channel
```

Brief description

Channel.

11.1.470.2 p_lock_mutex

```
TX_MUTEX* sf_i2c_bus_t::p_lock_mutex
```

Brief description

Lock mutex handle for this channel.

11.1.470.3 p_sync_eventflag

```
TX_EVENT_FLAGS_GROUP* sf_i2c_bus_t::p_sync_eventflag
```

Brief description

Pointer to the event flag object for I2C data transfer.

11.1.470.4 pp_curr_ctrl

```
sf_i2c_ctrl_t::pp_curr_ctrl
```

Brief description

Current device using the bus (by switching the address)

11.1.470.5 p_bus_name

```
uint8_t* sf_i2c_bus_t::p_bus_name
```

Brief description

User-supplied name to identify the bus. Useful for debugging.

11.1.470.6 p_lower_lvl_api

```
i2c_api_master_t::p_lower_lvl_api
```

Brief description

Pointer to I2C HAL interface to be used in the framework.

11.1.470.7 device_count

```
uint8_t sf_i2c_bus_t::device_count
```

Brief description

Number of devices on the bus; initialize to 0.

11.1.470.8 pp_curr_bus_ctrl

sf_i2c_ctrl_t::pp_curr_bus_ctrl

Brief description

Device configured on the bus (low level configuration)

11.1.471 sf_i2c_cfg_t

```
typedef struct{
    sf_i2c_bus_t *  p_bus
    i2c_cfg_t const  *  p_lower_lvl_cfg
} sf_i2c_cfg_t
```

11.1.471.1 p_bus

sf_i2c_bus_t::p_bus

Brief description

Bus used by the device.

11.1.471.2 p_lower_lvl_cfg

i2c_cfg_t::p_lower_lvl_cfg

Brief description

Pointer to I2C HAL configuration.

11.1.472 sf_i2c_instance_ctrl_t

```
typedef struct{
    sf_i2c_bus_t *  p_bus
    i2c_master_instance_t const  *  p_lower_lvl_i2c
    i2c_cfg_t  lower_lvl_cfg
    i2c_ctrl_t *  p_lower_lvl_ctrl
    sf_i2c_dev_state_t  dev_state
    bool  locked
    bool  restarted
} sf_i2c_instance_ctrl_t
```

11.1.472.1 p_bus

sf_i2c_bus_t::p_bus

Brief description

Bus using this device. Copy from configuration structure.

11.1.472.2 p_lower_lvl_i2c

i2c_master_instance_t::p_lower_lvl_i2c

Brief description

I2C instance.

11.1.472.3 lower_lvl_cfg

i2c_cfg_t::lower_lvl_cfg

Brief description

Used to reconfigure I2C driver.

11.1.472.4 p_lower_lvl_ctrl

i2c_ctrl_t::p_lower_lvl_ctrl

Brief description

I2C peripheral control block.

11.1.472.5 dev_state

sf_i2c_dev_state_t::dev_state

Brief description

Device status.

11.1.472.6 locked

bool ::locked

Brief description

Lock and unlock bus for a device.

11.1.472.7 restarted

bool ::restarted

Brief description

Indicates whether device issued a restart.

11.1.473 sf_i2c_instance_t

```
typedef struct{
    sf_i2c_ctrl_t *  p_ctrl
    sf_i2c_cfg_t const  *  p_cfg
    sf_i2c_api_t const  *  p_api
} sf_i2c_instance_t
```

11.1.473.1 p_ctrl

sf_i2c_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.473.2 p_cfg

sf_i2c_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.473.3 p_api

sf_i2c_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.474 sf_jpeg_decode_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_jpeg_decode_ctrl_t *const p_ctrl, sf_jpeg_decode_cfg_t
const *const p_cfg)
    ssp_err_t(*  inputBufferSet)(sf_jpeg_decode_ctrl_t *const p_ctrl, void *const
p_buffer, uint32_t const buffer_size)
    ssp_err_t(*  outputBufferSet)(sf_jpeg_decode_ctrl_t *const p_ctrl, void
*p_buffer, uint32_t buffer_size)
    ssp_err_t(*  linesDecodedGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, uint32_t
*const p_lines)
    ssp_err_t(*  horizontalStrideSet)(sf_jpeg_decode_ctrl_t *const p_ctrl,
uint32_t horizontal_stride)
```

```
    ssp_err_t(*  imageSubsampleSet)(sf_jpeg_decode_ctrl_t *const p_ctrl,
jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t
vertical_subsample)
    ssp_err_t(*  wait)(sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t
*const p_status, uint32_t timeout)
    ssp_err_t(*  statusGet)(sf_jpeg_decode_ctrl_t *const p_ctrl,
jpeg_decode_status_t *const p_status)
    ssp_err_t(*  imageSizeGet)(sf_jpeg_decode_ctrl_t *const p_ctrl, uint16_t
*p_horizontal_size, uint16_t *p_vertical_size)
    ssp_err_t(*  pixelFormatGet)(sf_jpeg_decode_ctrl_t *const p_ctrl,
jpeg_decode_color_space_t *const p_color_space)
    ssp_err_t(*  close)(sf_jpeg_decode_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_jpeg_decode_api_t
```

11.1.475 sf_jpeg_decode_cfg_t

```
typedef struct{
    jpeg_decode_instance_t const  *  p_lower_lvl_jpeg_decode
} sf_jpeg_decode_cfg_t
```

11.1.475.1 p_lower_lvl_jpeg_decode

jpeg_decode_instance_t::p_lower_lvl_jpeg_decode

Detailed description

Pointer to a driver structure that implements this interface. Pre-configured driver structures are located in r_jpeg_decode.c and extern'ed in r_jpeg_decode.h.

11.1.476 sf_jpeg_decode_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    uint32_t  state
    TX_MUTEX  mutex
    TX_EVENT_FLAGS_GROUP  events
    jpeg_decode_instance_t const  *  p_lower_lvl_jpeg_decode
} sf_jpeg_decode_instance_ctrl_t
```

11.1.476.1 open

uint32_t ::open

Brief description

Indicate whether the driver is open.

11.1.476.2 state

```
uint32_t ::state
```

Brief description

Used by driver to check if pointer to control block is valid.

11.1.476.3 mutex

```
TX_MUTEX ::mutex
```

Brief description

Mutex used to protect access to lower level driver hardware.

11.1.476.4 events

```
TX_EVENT_FLAGS_GROUP ::events
```

Brief description

Event flags used by the HAL driver to notify the framework driver of.

11.1.476.5 p_lower_lvl_jpeg_decode

```
jpeg_decode_instance_t::p_lower_lvl_jpeg_decode
```

Brief description

Pointer to lower level instance.

11.1.477 sf_jpeg_decode_instance_t

```
typedef struct{
    sf_jpeg_decode_ctrl_t *  p_ctrl
    sf_jpeg_decode_cfg_t const  *  p_cfg
    sf_jpeg_decode_api_t const  *  p_api
} sf_jpeg_decode_instance_t
```

11.1.477.1 p_ctrl

```
sf_jpeg_decode_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.477.2 p_cfg

sf_jpeg_decode_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.477.3 p_api

sf_jpeg_decode_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.478 sf_message_acquire_cfg_t

```
typedef struct{
    bool  buffer_keep
} sf_message_acquire_cfg_t
```

11.1.478.1 buffer_keep

bool sf_message_acquire_cfg_t::buffer_keep

Brief description

Buffer keep option.

11.1.479 sf_message_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_message_ctrl_t *const p_ctrl, sf_message_cfg_t const
*const p_cfg)
    ssp_err_t(*  close)(sf_message_ctrl_t *const p_ctrl)
    ssp_err_t(*  bufferAcquire)(sf_message_ctrl_t const *const p_ctrl,
sf_message_header_t **pp_buffer, sf_message_acquire_cfg_t const *const
p_acquire_cfg, uint32_t const wait_option)
    ssp_err_t(*  bufferRelease)(sf_message_ctrl_t *const p_ctrl,
sf_message_header_t *const p_buffer, sf_message_release_option_t const option)
    ssp_err_t(*  post)(sf_message_ctrl_t *const p_ctrl, sf_message_header_t const
*const p_buffer, sf_message_post_cfg_t const *const p_post_cfg,
sf_message_post_err_t *const p_post_err, uint32_t const wait_option)
    ssp_err_t(*  pend)(sf_message_ctrl_t const *const p_ctrl, TX_QUEUE const
*const p_queue, sf_message_header_t **pp_buffer, uint32_t const wait_option)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_message_api_t
```

11.1.480 sf_message_buffer_ctrl_t

```
typedef struct{
    void(*  p_callback)(sf_message_callback_args_t *)
    void const  *  p_context
    struct sf_message_buffer_ctrl_t::st_buffer_ctrl_flag  flag_b
} sf_message_buffer_ctrl_t
```

11.1.480.1 p_callback

```
void(* sf_message_buffer_ctrl_t::p_callback) (sf_message_callback_args_t *)
```

Brief description

Optional user callback function.

11.1.480.2 p_context

```
void const* sf_message_buffer_ctrl_t::p_context
```

Brief description

Context provided to user during callback.

11.1.480.3 flag_b

```
sf_message_buffer_ctrl_t::st_buffer_ctrl_flag::flag_b
```

11.1.481 sf_message_buffer_ctrl_t::st_buffer_ctrl_flag

```
typedef struct{
    uint32_t  semaphore
    uint32_t  buffer_keep
    uint32_t  nak_response
    uint32_t  reserved
    uint32_t  in_use
} sf_message_buffer_ctrl_t::st_buffer_ctrl_flag
```

11.1.481.1 semaphore

```
uint32_t sf_message_buffer_ctrl_t::st_buffer_ctrl_flag::semaphore
```

Brief description

Counting semaphore to prevent a buffer from being released.

11.1.481.2 buffer_keep

```
uint32_t sf_message_buffer_ctrl_t::st_buffer_ctrl_flag::buffer_keep
```

Brief description

Buffer keep request.

11.1.481.3 nak_response

```
uint32_t sf_message_buffer_ctrl_t::st_buffer_ctrl_flag::nak_response
```

Brief description

NAK (ORed logic for multiple subscribers)

11.1.481.4 reserved

```
uint32_t sf_message_buffer_ctrl_t::st_buffer_ctrl_flag::reserved
```

Brief description

Reserved bits.

11.1.481.5 in_use

```
uint32_t sf_message_buffer_ctrl_t::st_buffer_ctrl_flag::in_use
```

Brief description

Buffer in-use.

11.1.482 sf_message_callback_args_t

```
typedef struct{
    sf_message_callback_event_t  event
    void const  *  p_context
} sf_message_callback_args_t
```

11.1.482.1 event

```
sf_message_callback_event_t::event
```

Brief description

Event code.

11.1.482.2 p_context

```
void const* sf_message_callback_args_t::p_context
```

Brief description

Context provided to user during callback.

11.1.483 sf_message_cfg_t

```
typedef struct{
    void *  p_work_memory_start
    uint32_t  work_memory_size_bytes
    uint32_t  buffer_size
    sf_message_subscriber_list_t **  pp_subscriber_lists
    uint8_t *  p_block_pool_name
} sf_message_cfg_t
```

11.1.483.1 p_work_memory_start

```
void* sf_message_cfg_t::p_work_memory_start
```

Brief description

Start address of the memory area.

11.1.483.2 work_memory_size_bytes

```
uint32_t sf_message_cfg_t::work_memory_size_bytes
```

Brief description

Size of working memory area in bytes.

11.1.483.3 buffer_size

```
uint32_t sf_message_cfg_t::buffer_size
```

Brief description

Bytes of the message block.

11.1.483.4 pp_subscriber_lists

```
sf_message_subscriber_list_t::pp_subscriber_lists
```

Brief description

Pointer array to the subscriber lists.

11.1.483.5 p_block_pool_name

```
uint8_t* sf_message_cfg_t::p_block_pool_name
```

Brief description

Pointer to the block pool name.

11.1.484 sf_message_header_t

```
typedef struct{
    uint32_t  event
    uint32_t  class_code
    uint32_t  class_instance
    uint32_t  code
    struct{}                event_b
    union{}                 union{}
} sf_message_header_t
```

11.1.484.1 event

```
uint32_t sf_message_header_t::event
```

11.1.484.2 class_code

```
uint32_t sf_message_header_t::class_code
```

Brief description

Event class code.

11.1.484.3 class_instance

```
uint32_t sf_message_header_t::class_instance
```

Brief description

Event class instance number.

11.1.484.4 code

```
uint32_t sf_message_header_t::code
```

Brief description

Event code.

11.1.484.5 event_b

```
See source code for the definition of this member.
```

11.1.484.6 union{}

See source code for the definition of this member.

11.1.485 sf_message_instance_ctrl_t

```
typedef struct{
    TX_BLOCK_POOL  block_pool
    sf_message_subscriber_list_t **  pp_subscriber_lists
    uint32_t  buffer_size
    uint32_t  number_of_buffers
    uint16_t  number_of_subscriber_groups
    sf_message_state_t  state
} sf_message_instance_ctrl_t
```

11.1.485.1 block_pool

TX_BLOCK_POOL ::block_pool

Brief description

Pointer to the memory block pool control.

11.1.485.2 pp_subscriber_lists

sf_message_subscriber_list_t::pp_subscriber_lists

Brief description

Pointer array to the subscriber lists.

11.1.485.3 buffer_size

uint32_t ::buffer_size

Brief description

Bytes of the message buffer.

11.1.485.4 number_of_buffers

uint32_t ::number_of_buffers

Brief description

The number of allocated buffers.

11.1.485.5 number_of_subscriber_groups

```
uint16_t ::number_of_subscriber_groups
```

Brief description

The number of subscriber groups.

11.1.485.6 state

```
sf_message_state_t::state
```

Brief description

Status of the message framework.

11.1.486 sf_message_instance_range_t

```
typedef struct{
    uint8_t  start
    uint8_t  end
} sf_message_instance_range_t
```

11.1.486.1 start

```
uint8_t sf_message_instance_range_t::start
```

Brief description

Start of the event class instance range.

11.1.486.2 end

```
uint8_t sf_message_instance_range_t::end
```

Brief description

End of the event class instance range.

11.1.487 sf_message_instance_t

```
typedef struct{
    sf_message_ctrl_t *  p_ctrl
    sf_message_cfg_t const  *  p_cfg
    sf_message_api_t const  *  p_api
} sf_message_instance_t
```

11.1.487.1 p_ctrl

sf_message_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.487.2 p_cfg

sf_message_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.487.3 p_api

sf_message_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.488 sf_message_post_cfg_t

```
typedef struct{
    sf_message_priority_t  priority
    void(*  p_callback)(sf_message_callback_args_t *)
    void const  *  p_context
} sf_message_post_cfg_t
```

11.1.488.1 priority

sf_message_priority_t::priority

Brief description

Message priority.

11.1.488.2 p_callback

void(* sf_message_post_cfg_t::p_callback) (sf_message_callback_args_t *)

Brief description

User callback function.

11.1.488.3 p_context

```
void const* sf_message_post_cfg_t::p_context
```

Brief description

Context provided to user during callback.

11.1.489 sf_message_post_err_t

```
typedef struct{
    TX_QUEUE *  p_queue
} sf_message_post_err_t
```

11.1.489.1 p_queue

```
TX_QUEUE* sf_message_post_err_t::p_queue
```

Brief description

Queue.

11.1.490 sf_message_subscriber_list_t

```
typedef struct{
    sf_message_event_class_t  event_class
    uint16_t  number_of_nodes
    sf_message_subscriber_t **  pp_subscriber_group
} sf_message_subscriber_list_t
```

11.1.490.1 event_class

```
sf_message_event_class_t sf_message_subscriber_list_t::event_class
```

Brief description

Event class code.

11.1.490.2 number_of_nodes

```
uint16_t sf_message_subscriber_list_t::number_of_nodes
```

Brief description

Number of nodes in the subscriber group.

11.1.490.3 pp_subscriber_group

sf_message_subscriber_t::pp_subscriber_group

Brief description

Subscriber group for the event class.

11.1.491 sf_message_subscriber_t

```
typedef struct{
    TX_QUEUE *  p_queue
    sf_message_instance_range_t  instance_range
} sf_message_subscriber_t
```

11.1.491.1 p_queue

TX_QUEUE* sf_message_subscriber_t::p_queue

Brief description

Pointer to the message queue for subscriber thread.

11.1.491.2 instance_range

sf_message_instance_range_t::instance_range

Brief description

Range of the event class instance to receive message.

11.1.492 sf_power_profiles_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_power_profiles_ctrl_t *const p_ctrl,
sf_power_profiles_cfg_t const *const p_cfg)
    ssp_err_t(*  sleep)(sf_power_profiles_ctrl_t *const p_ctrl)
    ssp_err_t(*  close)(sf_power_profiles_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_power_profiles_api_t
```

11.1.493 sf_power_profiles_callback_args_t

```
typedef struct{
    sf_power_profiles_event_t  event
    void *  p_context
} sf_power_profiles_callback_args_t
```

11.1.493.1 event

sf_power_profiles_event_t::event

Brief description

Power profiles callback event.

11.1.493.2 p_context

void* sf_power_profiles_callback_args_t::p_context

Brief description

Placeholder for user data.

11.1.494 sf_power_profiles_cfg_t

```
typedef struct{
    ioport_cfg_t const  *const  p_wake_ioport_pin_tbl
    ioport_cfg_t const  *const  p_sleep_ioport_pin_tbl
    sf_power_profiles_mode_t  operating_mode
    bool  retain_output_signals
    lpm_instance_t const  *const  p_lower_lvl_lpm
    rtc_instance_t const  *const  p_lower_lvl_rtc
    void(*  p_callback)(sf_power_profiles_callback_args_t *p_args)
    void *  p_context
} sf_power_profiles_cfg_t
```

11.1.494.1 p_wake_ioport_pin_tbl

ioport_cfg_t::p_wake_ioport_pin_tbl

Brief description

Pointer to ioport settings for wakeup.

11.1.494.2 p_sleep_ioport_pin_tbl

ioport_cfg_t::p_sleep_ioport_pin_tbl

Brief description

Pointer to ioport settings for sleep.

11.1.494.3 operating_mode

sf_power_profiles_mode_t::operating_mode

Brief description

Power profile mode to use.

11.1.494.4 retain_output_signals

```
bool sf_power_profiles_cfg_t::retain_output_signals
```

Detailed description

(Future implementation:) True (Default) ==> address bus and bus control signals retain the output state False ==> signals are set to high-impedance state

11.1.494.5 p_lower_lvl_lpm

```
lpm_instance_t::p_lower_lvl_lpm
```

Brief description

Pointer to the LPM instance.

11.1.494.6 p_lower_lvl_rtc

```
rtc_instance_t::p_lower_lvl_rtc
```

Brief description

Pointer to the RTC instance (if any)

11.1.494.7 p_callback

```
void(* sf_power_profiles_cfg_t::p_callback) (sf_power_profiles_callback_args_t
*p_args)
```

Brief description

Callback function.

11.1.494.8 p_context

```
void* sf_power_profiles_cfg_t::p_context
```

Brief description

Placeholder for user data.

11.1.495 sf_power_profiles_ctrl_t

```
typedef struct{
    uint32_t  open
```

```
    ioport_cfg_t const  *  p_wake_ioport_pin_tbl
    ioport_cfg_t const  *  p_sleep_ioport_pin_tbl
    sf_power_profiles_mode_t  operating_mode
    lpm_api_t const  *  p_api_lpm
    rtc_api_t const  *  p_api_rtc
    rtc_ctrl_t *  p_ctrl_rtc
    void(*  p_callback)(sf_power_profiles_callback_args_t *p_args)
    void *  p_context
} sf_power_profiles_ctrl_t
```

11.1.495.1 open

uint32_t sf_power_profiles_ctrl_t::open

Brief description

Used by driver to check if pointer to control block is valid.

11.1.495.2 p_wake_ioport_pin_tbl

ioport_cfg_t::p_wake_ioport_pin_tbl

Brief description

Pointer to ioport settings for wakeup.

11.1.495.3 p_sleep_ioport_pin_tbl

ioport_cfg_t::p_sleep_ioport_pin_tbl

Brief description

Pointer to ioport settings for sleep.

11.1.495.4 operating_mode

sf_power_profiles_mode_t::operating_mode

Brief description

Power profile mode to use.

11.1.495.5 p_api_lpm

lpm_api_t::p_api_lpm

Brief description

Pointer to lower level Low Power driver function pointers.

11.1.495.6 p_api_rtc

rtc_api_t::p_api_rtc

Brief description

Pointer to lower level RTC driver function pointers.

11.1.495.7 p_ctrl_rtc

rtc_ctrl_t::p_ctrl_rtc

Brief description

Pointer to lower level RTC driver control block.

11.1.495.8 p_callback

```
void(* sf_power_profiles_ctrl_t::p_callback) (sf_power_profiles_callback_args_t
*p_args)
```

Brief description

Callback function.

11.1.495.9 p_context

```
void* sf_power_profiles_ctrl_t::p_context
```

Brief description

Placeholder for user data.

11.1.496 sf_power_profiles_instance_t

```
typedef struct{
    sf_power_profiles_ctrl_t *  p_ctrl
    sf_power_profiles_cfg_t const  *  p_cfg
    sf_power_profiles_api_t const  *  p_api
} sf_power_profiles_instance_t
```

11.1.496.1 p_ctrl

sf_power_profiles_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.496.2 p_cfg

sf_power_profiles_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.496.3 p_api

sf_power_profiles_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.497 sf_power_profiles_v2_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_power_profiles_v2_ctrl_t *const p_ctrl,
sf_power_profiles_v2_cfg_t const *const p_cfg)
    ssp_err_t(*  runApply)(sf_power_profiles_v2_ctrl_t *const p_ctrl,
sf_power_profiles_v2_run_cfg_t const *const p_cfg)
    ssp_err_t(*  lowPowerApply)(sf_power_profiles_v2_ctrl_t *const p_ctrl,
sf_power_profiles_v2_low_power_cfg_t const *const p_cfg)
    ssp_err_t(*  close)(sf_power_profiles_v2_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_power_profiles_v2_api_t
```

11.1.498 sf_power_profiles_v2_callback_args_t

```
typedef struct{
    sf_power_profiles_v2_event_t  event
    void *  p_context
} sf_power_profiles_v2_callback_args_t
```

11.1.498.1 event

sf_power_profiles_v2_event_t::event

Brief description

Power profiles callback event.

11.1.498.2 p_context

void* sf_power_profiles_v2_callback_args_t::p_context

Brief description

Placeholder for user data.

11.1.499 sf_power_profiles_v2_cfg_t

```
typedef struct{
    void const  *  p_extend
} sf_power_profiles_v2_cfg_t
```

11.1.499.1 p_extend

```
void const* sf_power_profiles_v2_cfg_t::p_extend
```

Detailed description

Pointer to additional settings (not currently in use)

11.1.500 sf_power_profiles_v2_ctrl_t

```
typedef struct{
    uint32_t  open
} sf_power_profiles_v2_ctrl_t
```

11.1.500.1 open

```
uint32_t sf_power_profiles_v2_ctrl_t::open
```

Brief description

Used by driver to check if pointer to control block is valid.

11.1.501 sf_power_profiles_v2_instance_t

```
typedef struct{
    sf_power_profiles_v2_ctrl_t *  p_ctrl
    sf_power_profiles_v2_cfg_t const  *  p_cfg
    sf_power_profiles_v2_api_t const  *  p_api
} sf_power_profiles_v2_instance_t
```

11.1.501.1 p_ctrl

```
sf_power_profiles_v2_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.501.2 p_cfg

sf_power_profiles_v2_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.501.3 p_api

sf_power_profiles_v2_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.502 sf_power_profiles_v2_low_power_cfg_t

```
typedef struct{
    ioport_cfg_t const  *  p_ioport_pin_tbl_exit
    ioport_cfg_t const  *  p_ioport_pin_tbl_enter
    lpmv2_instance_t const  *  p_lower_lvl_lpm
    void(*  p_callback)(sf_power_profiles_v2_callback_args_t *p_args)
    void *  p_context
    void const  *  p_extend
} sf_power_profiles_v2_low_power_cfg_t
```

11.1.502.1 p_ioport_pin_tbl_exit

ioport_cfg_t::p_ioport_pin_tbl_exit

Detailed description

Pointer to IOPORT settings to apply after exiting the low power mode

11.1.502.2 p_ioport_pin_tbl_enter

ioport_cfg_t::p_ioport_pin_tbl_enter

Detailed description

Pointer to IOPORT settings to apply before entering low power mode

11.1.502.3 p_lower_lvl_lpm

lpmv2_instance_t::p_lower_lvl_lpm

Detailed description

Pointer to an LPMv2 instance

11.1.502.4 p_callback

```
void(* sf_power_profiles_v2_low_power_cfg_t::p_callback)
(sf_power_profiles_v2_callback_args_t *p_args)
```

Detailed description

Callback function

11.1.502.5 p_context

```
void* sf_power_profiles_v2_low_power_cfg_t::p_context
```

Detailed description

Placeholder for user data

11.1.502.6 p_extend

```
void const* sf_power_profiles_v2_low_power_cfg_t::p_extend
```

Detailed description

Pointer to additional settings

11.1.503 sf_power_profiles_v2_run_cfg_t

```
typedef struct{
    ioport_cfg_t const  *  p_ioport_pin_tbl
    cgc_clocks_cfg_t const  *  p_clock_cfg
    void const  *  p_extend
} sf_power_profiles_v2_run_cfg_t
```

11.1.503.1 p_ioport_pin_tbl

```
ioport_cfg_t::p_ioport_pin_tbl
```

Detailed description

Pointer to IOPORT settings

11.1.503.2 p_clock_cfg

```
cgc_clocks_cfg_t::p_clock_cfg
```

Detailed description

Pointer to a CGC configuration

11.1.503.3 p_extend

```
void const* sf_power_profiles_v2_run_cfg_t::p_extend
```

Detailed description

Pointer to additional settings

11.1.504 sf_slider_on_ctsu_cfg_t

```
typedef struct{
    const sf_slider_type_t  type
    uint32_t  num_slider_channels
    ctsu_channel_pair_t const  *  p_slider_channels
    int32_t const  *  p_normalization
    int32_t *  p_channel_average
    uint32_t *  p_offset
    uint16_t *const  p_slider_scount
    uint16_t *const  p_slider_baseline
    int32_t *const  p_slider_delta
    sf_touch_ctsu_slider_id_t  id
    int32_t  max_slider_value
    const uint16_t  slider_norm_max
    const int32_t  slider_threshold
    const int32_t  channel_average_weight
    const int32_t  prev_sum_weight
    const int32_t  cutoff
} sf_slider_on_ctsu_cfg_t
```

11.1.504.1 type

```
sf_slider_type_t::type
```

Detailed description

Linear or circular (wheel)

11.1.504.2 num_slider_channels

```
uint32_t ::num_slider_channels
```

11.1.504.3 p_slider_channels

```
ctsu_channel_pair_t::p_slider_channels
```

Detailed description

Define the channel/channel pairs which make up a slider.

11.1.504.4 p_normalization

```
int32_t const* ::p_normalization
```

11.1.504.5 p_channel_average

```
int32_t* ::p_channel_average
```

11.1.504.6 p_offset

```
uint32_t* ::p_offset
```

11.1.504.7 p_slider_scount

```
uint16_t* const ::p_slider_scount
```

11.1.504.8 p_slider_baseline

```
uint16_t* const ::p_slider_baseline
```

11.1.504.9 p_slider_delta

```
int32_t* const ::p_slider_delta
```

11.1.504.10 id

sf_touch_ctsu_slider_id_t::id

Detailed description

Unique identifier for slider

11.1.504.11 max_slider_value

```
int32_t ::max_slider_value
```

Detailed description

Maximum slider value, must be greater than 0; Minimum is always 0

11.1.504.12 slider_norm_max

```
const uint16_t ::slider_norm_max
```

Detailed description

Individual slider settings that can be modified by the user. Should be the same as in st_sf_touch_ctsu_slider_hdlTOUCH_SLIDER_CFG_NORM_MAX

11.1.504.13 slider_threshold

```
const int32_t ::slider_threshold
```

Detailed description

Touch threshold. Ensure that value is greater than 0

11.1.504.14 channel_average_weight

```
const int32_t ::channel_average_weight
```

Detailed description

Weight of running average of counts for each channel

11.1.504.15 prev_sum_weight

```
const int32_t ::prev_sum_weight
```

Detailed description

Weight of running average of previous sum in position calculation, must be greater than 0

11.1.504.16 cutoff

```
const int32_t ::cutoff
```

Detailed description

Defines how far below prev_sum running average to get before "SF_TOUCH_SLIDER_STATE_RELEASED" detected

11.1.505 sf_socket_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_socket_ctrl_t *p_ctrl, sf_socket_cfg_t const *const
p_cfg)
    ssp_err_t(*  close)(sf_socket_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_socket_api_t
```

11.1.506 sf_socket_cfg_t

```
typedef struct{
    sf_wifi_onchip_stack_instance_t *  p_lower_lvl_onchip_wifi
    void *  p_extend
} sf_socket_cfg_t
```

11.1.506.1 p_lower_lvl_onchip_wifi

sf_wifi_onchip_stack_instance_t::p_lower_lvl_onchip_wifi

Brief description

Pointer to SF on-chip stack instance.

11.1.506.2 p_extend

void* sf_socket_cfg_t::p_extend

Brief description

Extended configuration.

11.1.507 sf_socket_ctrl_t

```
typedef struct{
    sf_wifi_onchip_stack_instance_t *  p_lower_lvl_onchip_wifi
} sf_socket_ctrl_t
```

11.1.507.1 p_lower_lvl_onchip_wifi

sf_wifi_onchip_stack_instance_t::p_lower_lvl_onchip_wifi

Brief description

low level wifi interface

11.1.508 sf_socket_instance_t

```
typedef struct{
    sf_socket_ctrl_t *  p_ctrl
    sf_socket_cfg_t const  *  p_cfg
    sf_socket_api_t const  *  p_api
} sf_socket_instance_t
```

11.1.508.1 p_ctrl

sf_socket_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.508.2 p_cfg

sf_socket_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.508.3 p_api

sf_socket_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.509 sf_spi_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_spi_ctrl_t *p_ctrl, sf_spi_cfg_t const *const p_cfg)
    ssp_err_t(*  read)(sf_spi_ctrl_t *const p_ctrl, void *const p_dest, uint32_t
const length, spi_bit_width_t const bit_width, uint32_t const timeout)
    ssp_err_t(*  write)(sf_spi_ctrl_t *const p_ctrl, void *const p_src, uint32_t
const length, spi_bit_width_t const bit_width, uint32_t const timeout)
    ssp_err_t(*  writeRead)(sf_spi_ctrl_t *const p_ctrl, void *const p_src, void
*const p_dest, uint32_t const length, spi_bit_width_t const bit_width, uint32_t
const timeout)
    ssp_err_t(*  close)(sf_spi_ctrl_t *const p_ctrl)
    ssp_err_t(*  lock)(sf_spi_ctrl_t *const p_ctrl)
    ssp_err_t(*  unlock)(sf_spi_ctrl_t *const p_ctrl)
    ssp_err_t(*  version)(ssp_version_t *const p_version)
} sf_spi_api_t
```

11.1.510 sf_spi_bus_t

```
typedef struct{
    uint8_t  channel
    uint32_t  freq_hz_min
    TX_MUTEX *  p_lock_mutex
    TX_EVENT_FLAGS_GROUP *  p_sync_eventflag
    sf_spi_ctrl_t **  pp_curr_ctrl
    uint8_t *  p_bus_name
    spi_api_t const  *  p_lower_lvl_api
    uint8_t  device_count
} sf_spi_bus_t
```

11.1.510.1 channel

```
uint8_t sf_spi_bus_t::channel
```

Brief description

Channel.

11.1.510.2 freq_hz_min

```
uint32_t sf_spi_bus_t::freq_hz_min
```

Brief description

Bus min frequency supported.

11.1.510.3 p_lock_mutex

```
TX_MUTEX* sf_spi_bus_t::p_lock_mutex
```

Brief description

Lock mutex handle for this channel.

11.1.510.4 p_sync_eventflag

```
TX_EVENT_FLAGS_GROUP* sf_spi_bus_t::p_sync_eventflag
```

Brief description

Pointer to the event flag object for SPI data transfer.

11.1.510.5 pp_curr_ctrl

```
sf_spi_ctrl_t::pp_curr_ctrl
```

Brief description

Current device using the bus.

11.1.510.6 p_bus_name

```
uint8_t* sf_spi_bus_t::p_bus_name
```

Brief description

peripheral name SCI_SPI/RSPI

11.1.510.7 p_lower_lvl_api

```
spi_api_t::p_lower_lvl_api
```

Brief description

Pointer to SPI HAL interface to be used in the framework.

11.1.510.8 device_count

```
uint8_t sf_spi_bus_t::device_count
```

Brief description

Number of devices on the bus, initialize to 0.

11.1.511 sf_spi_cfg_t

```
typedef struct{
    sf_spi_bus_t *  p_bus
    ioport_port_pin_t  chip_select
    ioport_level_t  chip_select_level_active
    spi_cfg_t const  *  p_lower_lvl_cfg
} sf_spi_cfg_t
```

11.1.511.1 p_bus

```
sf_spi_bus_t::p_bus
```

Brief description

Bus used by the device.

11.1.511.2 chip_select

```
ioport_port_pin_t::chip_select
```

Brief description

Chip select for this device.

11.1.511.3 chip_select_level_active

```
ioport_level_t::chip_select_level_active
```

Brief description

Polarity of CS, active High or Low.

11.1.511.4 p_lower_lvl_cfg

```
spi_cfg_t::p_lower_lvl_cfg
```

Brief description

Pointer to SPI HAL configuration.

11.1.512 sf_spi_instance_ctrl_t

```
typedef struct{
    sf_spi_bus_t *  p_bus
    ioport_port_pin_t  chip_select
    ioport_level_t  chip_select_level_active
    spi_cfg_t  lower_lvl_cfg
    spi_ctrl_t *  p_lower_lvl_ctrl
    sf_spi_dev_state_t  dev_state
    bool  locked
} sf_spi_instance_ctrl_t
```

11.1.512.1 p_bus

sf_spi_bus_t::p_bus

Brief description

Bus using this device (copy from cfg)

11.1.512.2 chip_select

ioport_port_pin_t::chip_select

Brief description

Chip select for this device (copy from cfg)

11.1.512.3 chip_select_level_active

ioport_level_t::chip_select_level_active

Brief description

Polarity of CS, active High or Low (copy from cfg)

11.1.512.4 lower_lvl_cfg

spi_cfg_t::lower_lvl_cfg

Brief description

SPI peripheral configuration, use for bus reconfiguration.

11.1.512.5 p_lower_lvl_ctrl

spi_ctrl_t::p_lower_lvl_ctrl

Brief description

SPI peripheral control block.

11.1.512.6 dev_state

sf_spi_dev_state_t::dev_state

Brief description

Device status.

11.1.512.7 locked

bool ::locked

Brief description

Lock and unlock bus for a device.

11.1.513 sf_spi_instance_t

```
typedef struct{
    sf_spi_ctrl_t *  p_ctrl
    sf_spi_cfg_t const  *  p_cfg
    sf_spi_api_t const  *  p_api
} sf_spi_instance_t
```

11.1.513.1 p_ctrl

sf_spi_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.513.2 p_cfg

sf_spi_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.513.3 p_api

sf_spi_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.514 sf_thread_monitor_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_thread_monitor_ctrl_t *const p_ctrl,
sf_thread_monitor_cfg_t const *const p_cfg)
    ssp_err_t(*  close)(sf_thread_monitor_ctrl_t *const p_ctrl)
    ssp_err_t(*  threadRegister)(sf_thread_monitor_ctrl_t *const p_ctrl,
sf_thread_monitor_counter_min_max_t const *p_counter_min_max)
    ssp_err_t(*  threadUnregister)(sf_thread_monitor_ctrl_t *const p_ctrl)
    ssp_err_t(*  countIncrement)(sf_thread_monitor_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_thread_monitor_api_t
```

11.1.515 sf_thread_monitor_cfg_t

```
typedef struct{
    wdt_instance_t const  *  p_lower_lvl_wdt
    bool  profiling_mode_enabled
    UINT  priority
} sf_thread_monitor_cfg_t
```

11.1.515.1 p_lower_lvl_wdt

wdt_instance_t::p_lower_lvl_wdt

Brief description

Pointer to lower level watchdog instance.

11.1.515.2 profiling_mode_enabled

bool sf_thread_monitor_cfg_t::profiling_mode_enabled

Brief description

Enables or disables profiling mode.

11.1.515.3 priority

UINT sf_thread_monitor_cfg_t::priority

Brief description

Priority of thread monitor thread.

11.1.516 sf_thread_monitor_counter_min_max_t

```
typedef struct{
    uint32_t  minimum_count
    uint32_t  maximum_count
} sf_thread_monitor_counter_min_max_t
```

11.1.516.1 minimum_count

```
uint32_t sf_thread_monitor_counter_min_max_t::minimum_count
```

Detailed description

Minimum expected count value. If the current count is less than this value the watchdog will reset.

11.1.516.2 maximum_count

```
uint32_t sf_thread_monitor_counter_min_max_t::maximum_count
```

Detailed description

Maximum expected count value. If the current count is more than this value the watchdog will reset

11.1.517 sf_thread_monitor_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    wdt_instance_t const  *  p_lower_lvl_wdt
    uint32_t  timeout_period_msec
    uint32_t  timeout_period_watchdog_clocks
    bool  profiling_mode_enabled
    TX_MUTEX  mutex
    uint32_t  profiling_mode_check
    sf_thread_monitor_thread_counter_t  thread_counters[THREAD_MONITOR_CFG_MAX_N
UMBER_OF_THREADS]
    TX_THREAD  thread
    void const  *  p_extend
    uint8_t  stack[THREAD_MONITOR_THREAD_STACK_SIZE]
} sf_thread_monitor_instance_ctrl_t
```

11.1.517.1 open

```
uint32_t ::open
```

Detailed description

Used by driver to check if the control structure is valid

11.1.517.2 p_lower_lvl_wdt

`wdt_instance_t::p_lower_lvl_wdt`

Detailed description

Pointer to interface structure for the watchdog peripheral

11.1.517.3 timeout_period_msec

`uint32_t ::timeout_period_msec`

Detailed description

Time in milliseconds of the watchdog timeout period. Used to calculate the period of the monitoring thread.

11.1.517.4 timeout_period_watchdog_clocks

`uint32_t ::timeout_period_watchdog_clocks`

Detailed description

Maximum count value of the watchdog. Used to synchronise to the count.

11.1.517.5 profiling_mode_enabled

`bool ::profiling_mode_enabled`

Detailed description

Used by the driver to check if profiling mode is enabled.

11.1.517.6 mutex

`TX_MUTEX ::mutex`

Brief description

Mutex to protect access to the thread counters.

11.1.517.7 profiling_mode_check

`uint32_t ::profiling_mode_check`

Detailed description

Value used to verify profiling mode is enabled when prfiling_mode_enabled == true.

11.1.517.8 thread_counters

sf_thread_monitor_thread_counter_t::thread_counters

Detailed description

Data storage for the thread counter information.

11.1.517.9 thread

TX_THREAD ::thread

Brief description

Thread monitor thread.

11.1.517.10 p_extend

void const* ::p_extend

Brief description

Extended configuration data.

11.1.517.11 stack

uint8_t ::stack[THREAD_MONITOR_THREAD_STACK_SIZE]

Detailed description

Stack for thread monitor thread.

11.1.518 sf_thread_monitor_instance_t

```
typedef struct{
    sf_thread_monitor_ctrl_t *  p_ctrl
    sf_thread_monitor_cfg_t const  *  p_cfg
    sf_thread_monitor_api_t const  *  p_api
} sf_thread_monitor_instance_t
```

11.1.518.1 p_ctrl

sf_thread_monitor_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.518.2 p_cfg

sf_thread_monitor_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.518.3 p_api

sf_thread_monitor_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.519 sf_thread_monitor_thread_counter_t

```
typedef struct{
    uint32_t  current_count
    uint32_t  minimum_count
    uint32_t  maximum_count
    bool  active
    TX_THREAD *  p_thread
} sf_thread_monitor_thread_counter_t
```

11.1.519.1 current_count

uint32_t sf_thread_monitor_thread_counter_t::current_count

Brief description

Current count value for a thread.

11.1.519.2 minimum_count

uint32_t sf_thread_monitor_thread_counter_t::minimum_count

Detailed description

Minimum expected count value. If the current count is less than this value the watchdog will reset.

11.1.519.3 maximum_count

uint32_t sf_thread_monitor_thread_counter_t::maximum_count

Detailed description

Maximum expected count value. If the current count is more than this value the watchdog will reset

11.1.519.4 active

```
bool sf_thread_monitor_thread_counter_t::active
```

Detailed description

Indicates to the monitoring thread whether this count data is currently active. When a thread is registered this value will be set to false as the count is likely to be a partial count and so should not be monitored. This value will be set to true by the thread monitor thread when it clears all counts to zero.

11.1.519.5 p_thread

```
TX_THREAD* sf_thread_monitor_thread_counter_t::p_thread
```

Brief description

Pointer to thread for this counter data.

11.1.520 sf_touch_ctsu_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_touch_ctsu_ctrl_t *const p_ctrl, sf_touch_ctsu_cfg_t
const *const p_cfg)
    ssp_err_t(*  read)(sf_touch_ctsu_ctrl_t *const p_ctrl, void *p_dest,
ctsu_read_t opts, const ctsu_channel_pair_t *channels, const uint16_t count)
    ssp_err_t(*  close)(sf_touch_ctsu_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_touch_ctsu_api_t
```

11.1.521 sf_touch_ctsu_button_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_touch_ctsu_button_ctrl_t *const p_ctrl,
sf_touch_ctsu_button_cfg_t const *const p_cfg)
    ssp_err_t(*  enable)(sf_touch_ctsu_button_ctrl_t *const p_ctrl,
sf_touch_ctsu_button_id const button_id)
    ssp_err_t(*  disable)(sf_touch_ctsu_button_ctrl_t *const p_ctrl,
sf_touch_ctsu_button_id const button_id)
    ssp_err_t(*  close)(sf_touch_ctsu_button_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_touch_ctsu_button_api_t
```

11.1.522 sf_touch_ctsu_button_callback_args_t

```
typedef struct{
    sf_touch_ctsu_button_id  id
    sf_touch_button_state_t  event
```

```
    void const  *  p_context
} sf_touch_ctsu_button_callback_args_t
```

11.1.522.1 id

sf_touch_ctsu_button_id::id

Brief description

Unique id for button.

11.1.522.2 event

sf_touch_button_state_t::event

Brief description

Button callback event.

11.1.522.3 p_context

void const* sf_touch_ctsu_button_callback_args_t::p_context

Brief description

Placeholder for user data.

11.1.523 sf_touch_ctsu_button_cfg_t

```
typedef struct{
    sf_touch_ctsu_instance_t const  *const  p_lower_lvl_touch_framework
    uint32_t  button_count
    sf_touch_ctsu_button_individual_t **  pp_button_cfgs
    void(*  p_callback)(sf_touch_ctsu_button_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} sf_touch_ctsu_button_cfg_t
```

11.1.523.1 p_lower_lvl_touch_framework

sf_touch_ctsu_instance_t::p_lower_lvl_touch_framework

Brief description

Pointer to the Touch Framework instance.

11.1.523.2 button_count

uint32_t sf_touch_ctsu_button_cfg_t::button_count

Brief description

Number of buttons in configuration.

11.1.523.3 pp_button_cfgs

```
sf_touch_ctsu_button_individual_t::pp_button_cfgs
```

Brief description

Pointer to button configurations.

11.1.523.4 p_callback

```
void(* sf_touch_ctsu_button_cfg_t::p_callback)
(sf_touch_ctsu_button_callback_args_t *p_args)
```

Brief description

Callback function.

11.1.523.5 p_context

```
void const* sf_touch_ctsu_button_cfg_t::p_context
```

Brief description

Placeholder for user data.

11.1.523.6 p_extend

```
void const* sf_touch_ctsu_button_cfg_t::p_extend
```

Brief description

Extension parameter for instance specific settings.

11.1.524 sf_touch_ctsu_button_hdl

```
typedef struct{
    sf_touch_ctsu_button_individual_t  button_cfg
    sf_touch_button_state_t  state
    int16_t  offset
    sf_touch_button_state_t  prev_state
    uint32_t  debounce_counter
    uint32_t  active_event_counter
    uint32_t  press_down_counter
    uint32_t  open
} sf_touch_ctsu_button_hdl
```

11.1.524.1 button_cfg

sf_touch_ctsu_button_individual_t::button_cfg

Brief description

Individual button configuration.

11.1.524.2 state

sf_touch_button_state_t::state

Brief description

Represents the current state of the button.

11.1.524.3 offset

```
int16_t sf_touch_ctsu_button_hdl::offset
```

Brief description

Offset in the results array and bit offset in the binary.

11.1.524.4 prev_state

sf_touch_button_state_t::prev_state

Brief description

Holds previous state of the button.

11.1.524.5 debounce_counter

```
uint32_t sf_touch_ctsu_button_hdl::debounce_counter
```

Brief description

Debounce counter.

11.1.524.6 active_event_counter

```
uint32_t sf_touch_ctsu_button_hdl::active_event_counter
```

Brief description

Amount of time for which button is spending between states.

11.1.524.7 press_down_counter

```
uint32_t sf_touch_ctsu_button_hdl::press_down_counter
```

Brief description

Time for which button is held down.

11.1.524.8 open

```
uint32_t sf_touch_ctsu_button_hdl::open
```

Brief description

Indicate that button has been opened.

11.1.525 sf_touch_ctsu_button_individual_t

```
typedef struct{
    ctsu_channel_pair_t  button_channel
    uint8_t  release_enable
    uint8_t  press_enable
    uint8_t  repeat_enable
    uint8_t  shorthold_enable
    uint8_t  longhold_enable
    uint8_t  byte
    union{}                 event_enable
    uint16_t  debounce_threshold
    sf_touch_ctsu_button_id  id
} sf_touch_ctsu_button_individual_t
```

11.1.525.1 button_channel

```
ctsu_channel_pair_t::button_channel
```

Brief description

Define the channel/channel pairs which make up a button.

11.1.525.2 release_enable

```
uint8_t sf_touch_ctsu_button_individual_t::release_enable
```

Brief description

enable release events

11.1.525.3 press_enable

```
uint8_t sf_touch_ctsu_button_individual_t::press_enable
```

Brief description

enable press events

11.1.525.4 repeat_enable

`uint8_t sf_touch_ctsu_button_individual_t::repeat_enable`

Brief description

enable repeat events

11.1.525.5 shorthold_enable

`uint8_t sf_touch_ctsu_button_individual_t::shorthold_enable`

Brief description

enable short hold events

11.1.525.6 longhold_enable

`uint8_t sf_touch_ctsu_button_individual_t::longhold_enable`

Brief description

enable long hold events

11.1.525.7 byte

`uint8_t sf_touch_ctsu_button_individual_t::byte`

Brief description

Byte representation of events enabled.

11.1.525.8 event_enable

`See source code for the definition of this member.`

11.1.525.9 debounce_threshold

`uint16_t sf_touch_ctsu_button_individual_t::debounce_threshold`

Brief description

Number of consecutive times a button is determined as touched before state changes.

11.1.525.10 id

`sf_touch_ctsu_button_id::id`

Brief description

Unique id for button.

11.1.526 sf_touch_ctsu_button_instance_ctrl_t

```
typedef struct{
    uint32_t  opened
    sf_touch_ctsu_button_hdl *  p_button_hdl
    uint32_t  button_count
    sf_touch_ctsu_instance_t const  *  p_lower_lvl_ctsu
    void const  *  p_context
    void(*  p_callback)(sf_touch_ctsu_button_callback_args_t *p_args)
} sf_touch_ctsu_button_instance_ctrl_t
```

11.1.526.1 opened

```
uint32_t ::opened
```

Brief description

Save the initialization state.

11.1.526.2 p_button_hdl

```
sf_touch_ctsu_button_hdl::p_button_hdl
```

Brief description

Pointer to the button handle.

11.1.526.3 button_count

```
uint32_t ::button_count
```

Brief description

Button Count.

11.1.526.4 p_lower_lvl_ctsu

```
sf_touch_ctsu_instance_t::p_lower_lvl_ctsu
```

Brief description

Pointer to the lower level instance.

11.1.526.5 p_context

```
void const* ::p_context
```

Brief description

Placeholder for user data.

11.1.526.6 p_callback

```
void(* ::p_callback) ( *p_args)
```

Brief description

Callback function.

11.1.527 sf_touch_ctsu_button_instance_t

```
typedef struct{
    sf_touch_ctsu_button_ctrl_t *  p_ctrl
    sf_touch_ctsu_button_cfg_t const  *  p_cfg
    sf_touch_ctsu_button_api_t const  *  p_api
} sf_touch_ctsu_button_instance_t
```

11.1.527.1 p_ctrl

sf_touch_ctsu_button_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.527.2 p_cfg

sf_touch_ctsu_button_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.527.3 p_api

sf_touch_ctsu_button_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.528 sf_touch_ctsu_callback_args_t

```
typedef struct{
    void *  p_context
} sf_touch_ctsu_callback_args_t
```

11.1.528.1 p_context

```
void* sf_touch_ctsu_callback_args_t::p_context
```

11.1.529 sf_touch_ctsu_cfg_t

```
typedef struct{
    UINT  priority
    uint16_t  update_hz
    void(*  p_callback)(sf_touch_ctsu_callback_args_t *p_args)
    void *  p_context
    ctsu_instance_t *  p_ctsu_instance
} sf_touch_ctsu_cfg_t
```

11.1.529.1 priority

```
UINT sf_touch_ctsu_cfg_t::priority
```

Brief description

Priority of the touch panel thread.

11.1.529.2 update_hz

```
uint16_t sf_touch_ctsu_cfg_t::update_hz
```

Brief description

The frequency to run the scans. This is set by the latest open() call.

11.1.529.3 p_callback

```
void(* sf_touch_ctsu_cfg_t::p_callback) (sf_touch_ctsu_callback_args_t *p_args)
```

Brief description

Callback function;.

11.1.529.4 p_context

```
void* sf_touch_ctsu_cfg_t::p_context
```

Brief description

Arguments to the callback.

11.1.529.5 p_ctsu_instance

```
ctsu_instance_t::p_ctsu_instance
```

Brief description

CTSU instance.

11.1.530 sf_touch_ctsu_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    uint16_t  update_hz
    ctsu_instance_t *  p_lower_lvl_instance
    uint32_t  cb_index
} sf_touch_ctsu_instance_ctrl_t
```

11.1.530.1 open

```
uint32_t ::open
```

Brief description

Used by driver to check if control block is valid.

11.1.530.2 update_hz

```
uint16_t ::update_hz
```

Brief description

The frequency to run the scans at.

11.1.530.3 p_lower_lvl_instance

```
ctsu_instance_t::p_lower_lvl_instance
```

Brief description

Pointer to CTSU instance.

11.1.530.4 cb_index

```
uint32_t ::cb_index
```

Brief description

Indicates the index in callback registry table.

11.1.531 sf_touch_ctsu_instance_t

```
typedef struct{
    sf_touch_ctsu_ctrl_t *  p_ctrl
```

```
    sf_touch_ctsu_cfg_t const  *  p_cfg
    sf_touch_ctsu_api_t const  *  p_api
} sf_touch_ctsu_instance_t
```

11.1.531.1 p_ctrl

sf_touch_ctsu_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.531.2 p_cfg

sf_touch_ctsu_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.531.3 p_api

sf_touch_ctsu_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.532 sf_touch_ctsu_slider_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_touch_ctsu_slider_ctrl_t *const p_ctrl,
sf_touch_ctsu_slider_cfg_t const *const p_cfg)
    ssp_err_t(*  enable)(sf_touch_ctsu_slider_ctrl_t *const p_ctrl,
sf_touch_ctsu_slider_id_t const slider_id)
    ssp_err_t(*  disable)(sf_touch_ctsu_slider_ctrl_t *const p_ctrl,
sf_touch_ctsu_slider_id_t const slider_id)
    ssp_err_t(*  close)(sf_touch_ctsu_slider_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_touch_ctsu_slider_api_t
```

11.1.533 sf_touch_ctsu_slider_callback_args_t

```
typedef struct{
    sf_touch_ctsu_slider_id_t  id
    uint32_t  event
    uint32_t  current_position
```

```
    void const  *  p_context
} sf_touch_ctsu_slider_callback_args_t
```

11.1.533.1 id

sf_touch_ctsu_slider_id_t::id

Brief description

Unique slider identifier.

11.1.533.2 event

uint32_t sf_touch_ctsu_slider_callback_args_t::event

Brief description

Slider callback event.

11.1.533.3 current_position

uint32_t sf_touch_ctsu_slider_callback_args_t::current_position

Brief description

Current slider position.

11.1.533.4 p_context

void const* sf_touch_ctsu_slider_callback_args_t::p_context

Brief description

Placeholder for user data.

11.1.534 sf_touch_ctsu_slider_cfg_t

```
typedef struct{
    sf_touch_ctsu_instance_t *  p_lower_lvl_touch_framework
    uint32_t  slider_count
    void(*  p_callback)(sf_touch_ctsu_slider_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} sf_touch_ctsu_slider_cfg_t
```

11.1.534.1 p_lower_lvl_touch_framework

sf_touch_ctsu_instance_t::p_lower_lvl_touch_framework

Detailed description

Pointer to the Touch Framework instance

11.1.534.2 slider_count

```
uint32_t sf_touch_ctsu_slider_cfg_t::slider_count
```

Detailed description

Number of sliders in configuration

11.1.534.3 p_callback

```
void(* sf_touch_ctsu_slider_cfg_t::p_callback)
(sf_touch_ctsu_slider_callback_args_t *p_args)
```

Detailed description

Callback function

11.1.534.4 p_context

```
void const* sf_touch_ctsu_slider_cfg_t::p_context
```

Detailed description

Placeholder for user data

11.1.534.5 p_extend

```
void const* sf_touch_ctsu_slider_cfg_t::p_extend
```

Detailed description

Extension parameter for instance specific settings.

11.1.535 sf_touch_ctsu_slider_instance_ctrl_t

```
typedef struct{
    uint32_t  opened
    uint32_t  slider_count
    sf_touch_ctsu_instance_t const  *  p_lower_lvl_ctsu
    void const  *  p_context
    void(*  p_callback)(sf_touch_ctsu_slider_callback_args_t *p_args)
} sf_touch_ctsu_slider_instance_ctrl_t
```

11.1.535.1 opened

```
uint32_t ::opened
```

Brief description

Save the initialization state of the framework.

11.1.535.2 slider_count

```
uint32_t ::slider_count
```

Brief description

Slider Count.

11.1.535.3 p_lower_lvl_ctsu

```
sf_touch_ctsu_instance_t::p_lower_lvl_ctsu
```

Brief description

Pointer to the lower level instance.

11.1.535.4 p_context

```
void const* ::p_context
```

Brief description

Placeholder for user data.

11.1.535.5 p_callback

```
void(* ::p_callback) ( *p_args)
```

Detailed description

Function to call when an event occurs

11.1.536 sf_touch_ctsu_slider_instance_t

```
typedef struct{
    sf_touch_ctsu_slider_ctrl_t *  p_ctrl
    sf_touch_ctsu_slider_cfg_t const  *  p_cfg
    sf_touch_ctsu_slider_api_t const  *  p_api
} sf_touch_ctsu_slider_instance_t
```

11.1.536.1 p_ctrl

sf_touch_ctsu_slider_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.536.2 p_cfg

sf_touch_ctsu_slider_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.536.3 p_api

sf_touch_ctsu_slider_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.537 sf_touch_panel_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_touch_panel_ctrl_t *const p_ctrl, sf_touch_panel_cfg_t
const *const p_cfg)
    ssp_err_t(*  calibrate)(sf_touch_panel_ctrl_t *const p_ctrl,
sf_touch_panel_calibrate_t const *const p_expected, sf_touch_panel_payload_t
const *const p_actual, ULONG timeout)
    ssp_err_t(*  start)(sf_touch_panel_ctrl_t *const p_ctrl)
    ssp_err_t(*  stop)(sf_touch_panel_ctrl_t *const p_ctrl)
    ssp_err_t(*  reset)(sf_touch_panel_ctrl_t *const p_ctrl)
    ssp_err_t(*  close)(sf_touch_panel_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_touch_panel_api_t
```

11.1.538 sf_touch_panel_calibrate_t

```
typedef struct{
    uint16_t  x
    uint16_t  y
    uint16_t  tolerance_pixels
    void const  *  p_extend
} sf_touch_panel_calibrate_t
```

11.1.538.1 x

```
uint16_t sf_touch_panel_calibrate_t::x
```

Brief description

Expected x coordinate.

11.1.538.2 y

```
uint16_t sf_touch_panel_calibrate_t::y
```

Brief description

Expected y coordinate.

11.1.538.3 tolerance_pixels

```
uint16_t sf_touch_panel_calibrate_t::tolerance_pixels
```

Detailed description

Acceptable linear deviation from the expected coordinates in pixels.

11.1.538.4 p_extend

```
void const* sf_touch_panel_calibrate_t::p_extend
```

Detailed description

Pointer to hardware specific extension. See sf_touch_panel_<instance>_cfg_t in sf_touch_panel_<instance>.h.

11.1.539 sf_touch_panel_cfg_t

```
typedef struct{
    uint16_t  hsize_pixels
    uint16_t  vsize_pixels
    UINT  priority
    sf_message_instance_t const  *  p_message
    uint8_t  event_class_instance
    uint16_t  update_hz
    uint16_t  rotation_angle
    void const  *  p_extend
} sf_touch_panel_cfg_t
```

11.1.539.1 hsize_pixels

```
uint16_t sf_touch_panel_cfg_t::hsize_pixels
```

Brief description

Horizontal size of screen in pixels.

11.1.539.2 vsize_pixels

`uint16_t sf_touch_panel_cfg_t::vsize_pixels`

Brief description

Vertical size of screen in pixels.

11.1.539.3 priority

`UINT sf_touch_panel_cfg_t::priority`

Brief description

Priority of the touch panel thread.

11.1.539.4 p_message

`sf_message_instance_t::p_message`

Brief description

Pointer to messaging framework control block.

11.1.539.5 event_class_instance

`uint8_t sf_touch_panel_cfg_t::event_class_instance`

Brief description

Event class instance number for posting touch event class messages.

11.1.539.6 update_hz

`uint16_t sf_touch_panel_cfg_t::update_hz`

Detailed description

The frequency to report repeat (SF_TOUCH_PANEL_EVENT_DOWN or SF_TOUCH_PANEL_EVENT_HOLD) touch events in Hertz.
NOTE: This will be converted to RTOS ticks in the driver and rounded up to the nearest integer value of RTOS ticks.

11.1.539.7 rotation_angle

`uint16_t sf_touch_panel_cfg_t::rotation_angle`

Brief description

Touch coordinate rotation angle(0/90/180/270)

11.1.539.8 p_extend

```
void const* sf_touch_panel_cfg_t::p_extend
```

Detailed description

Pointer to hardware specific extension. See sf_touch_panel_<instance>.h.

11.1.540 sf_touch_panel_i2c_cfg_t

```
typedef struct{
    i2c_master_instance_t const  *  p_lower_lvl_i2c
    sf_external_irq_instance_t const  *  p_lower_lvl_irq
    ioport_port_pin_t  pin
    sf_touch_panel_i2c_chip_t const  *const  p_chip
} sf_touch_panel_i2c_cfg_t
```

11.1.540.1 p_lower_lvl_i2c

```
i2c_master_instance_t::p_lower_lvl_i2c
```

Detailed description

Pointer to lower level I2C.

11.1.540.2 p_lower_lvl_irq

```
sf_external_irq_instance_t::p_lower_lvl_irq
```

Detailed description

Pointer to lower level external IRQ.

11.1.540.3 pin

```
ioport_port_pin_t::pin
```

Detailed description

Port pin connected to reset line on touch controller chip. Set to SF_TOUCH_PANEL_I2C_RESET_PIN_UNUSED if unused.

11.1.540.4 p_chip

```
sf_touch_panel_i2c_chip_t::p_chip
```

Detailed description

Selected touch controller chip.

11.1.541 sf_touch_panel_i2c_chip_t

```
typedef struct{
    ssp_err_t(*  payloadGet)(sf_touch_panel_ctrl_t *const p_ctrl,
sf_touch_panel_payload_t *const p_payload)
    ssp_err_t(*  reset)(sf_touch_panel_ctrl_t *const p_ctrl)
} sf_touch_panel_i2c_chip_t
```

11.1.541.1 payloadGet

```
(* ::payloadGet) ( *const p_ctrl, *const p_payload)
```

Detailed description

Reads the touch chip and fills in the touch payload data.

Table 2390:

Name	Direction	Description
p_ctrl	inout	Pointer to a structure allocated by user. This control structure is initialized in this function.
p_payload	out	Pointer to the payload to data structure. Touch data provided should be processed to logical pixel values.

Parameter p_ctrl

Parameter p_payload

11.1.541.2 reset

```
(* ::reset) ( *const p_ctrl)
```

Detailed description

Resets the touch chip.

Table 2391:

Name	Direction	Description
p_ctrl	inout	Pointer to a structure allocated by user. This control structure is initialized in this function.

Parameter p_ctrl

11.1.542 sf_touch_panel_i2c_instance_ctrl_t

```
typedef struct{
    uint32_t  open
    uint16_t  hsize_pixels
    uint16_t  vsize_pixels
    sf_message_instance_t const  *  p_message
    uint8_t  event_class_instance
    sf_touch_panel_payload_t *  p_payload
    sf_touch_panel_payload_t  last_payload
    TX_MUTEX  mutex
    TX_EVENT_FLAGS_GROUP  flags
    TX_THREAD  thread
    ioport_port_pin_t  pin
    i2c_master_instance_t const  *  p_lower_lvl_i2c
    sf_external_irq_instance_t const  *  p_lower_lvl_irq
    uint8_t  stack[SF_TOUCH_PANEL_I2C_STACK_SIZE]
    sf_touch_panel_i2c_chip_t const  *  p_chip
    uint16_t  update_hz
    uint16_t  rotation_angle
} sf_touch_panel_i2c_instance_ctrl_t
```

11.1.542.1 open

```
uint32_t ::open
```

Brief description

Used by driver to check if control block is valid.

11.1.542.2 hsize_pixels

```
uint16_t ::hsize_pixels
```

Brief description

Horizontal size of screen in pixels.

11.1.542.3 vsize_pixels

```
uint16_t ::vsize_pixels
```

Brief description

Vertical size of screen in pixels.

11.1.542.4 p_message

```
sf_message_instance_t::p_message
```

Brief description

Pointer to messaging framework control block.

11.1.542.5 event_class_instance

```
uint8_t ::event_class_instance
```

Brief description

Event class instance number for posting touch event class messages.

11.1.542.6 p_payload

```
sf_touch_panel_payload_t::p_payload
```

Brief description

Pointer to buffer used to store payload.

11.1.542.7 last_payload

```
sf_touch_panel_payload_t::last_payload
```

Brief description

Stores most recent payload put on queue for comparison.

11.1.542.8 mutex

```
TX_MUTEX ::mutex
```

Brief description

Mutex used to protect access to shared resources.

11.1.542.9 flags

```
TX_EVENT_FLAGS_GROUP ::flags
```

Brief description

Event flags for internal communication.

11.1.542.10 thread

```
TX_THREAD ::thread
```

Brief description

Main touch panel thread.

11.1.542.11 pin

```
ioport_port_pin_t::pin
```

Brief description

Reset pin.

11.1.542.12 p_lower_lvl_i2c

```
i2c_master_instance_t::p_lower_lvl_i2c
```

Brief description

Lower level I2C.

11.1.542.13 p_lower_lvl_irq

```
sf_external_irq_instance_t::p_lower_lvl_irq
```

Brief description

Lower level external IRQ.

11.1.542.14 stack

```
uint8_t ::stack[SF_TOUCH_PANEL_I2C_STACK_SIZE]
```

Brief description

Stack for touch panel thread.

11.1.542.15 p_chip

```
sf_touch_panel_i2c_chip_t::p_chip
```

Brief description

Chip specific functions and definitions.

11.1.542.16 update_hz

```
uint16_t ::update_hz
```

Detailed description

The frequency to report repeat (SF_TOUCH_PANEL_EVENT_DOWN or SF_TOUCH_PANEL_EVENT_HOLD) touch events in Hertz.
NOTE: This will be converted to RTOS ticks in the driver and rounded up to the nearest integer value of RTOS ticks.

11.1.542.17 rotation_angle

```
uint16_t ::rotation_angle
```

Brief description

Touch coordinate rotation angle(0/90/180/270)

11.1.543 sf_touch_panel_instance_t

```
typedef struct{
    sf_touch_panel_ctrl_t *  p_ctrl
    sf_touch_panel_cfg_t const  *  p_cfg
    sf_touch_panel_api_t const  *  p_api
} sf_touch_panel_instance_t
```

11.1.543.1 p_ctrl

```
sf_touch_panel_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.543.2 p_cfg

```
sf_touch_panel_cfg_t::p_cfg
```

Brief description

Pointer to the configuration structure for this instance.

11.1.543.3 p_api

```
sf_touch_panel_api_t::p_api
```

Brief description

Pointer to the API structure for this instance.

11.1.544 sf_touch_panel_payload_t

```
typedef struct{
    sf_message_header_t  header
    int16_t  x
    int16_t  y
    sf_touch_panel_event_t  event_type
} sf_touch_panel_payload_t
```

11.1.544.1 header

sf_message_header_t::header

Brief description

Required header for messaging framework.

11.1.544.2 x

int16_t sf_touch_panel_payload_t::x

Brief description

X coordinate.

11.1.544.3 y

int16_t sf_touch_panel_payload_t::y

Brief description

Y coordinate.

11.1.544.4 event_type

sf_touch_panel_event_t::event_type

Brief description

Touch event type.

11.1.545 sf_uart_comms_cfg_t

```
typedef struct{
    uart_instance_t const  *  p_lower_lvl_uart
} sf_uart_comms_cfg_t
```

11.1.545.1 p_lower_lvl_uart

uart_instance_t::p_lower_lvl_uart

Detailed description

Pointer to UART Driver instance

11.1.546 sf_uart_comms_instance_ctrl_t

```
typedef struct{
    uint32_t  state
    uart_instance_t const  *  p_lower_lvl_uart
    TX_MUTEX  mutex[2]
    TX_EVENT_FLAGS_GROUP  eventflag[2]
    TX_QUEUE  queue
    uint32_t  queue_mem[SF_UART_COMMS_CFG_QUEUE_SIZE_WORDS]
} sf_uart_comms_instance_ctrl_t
```

11.1.546.1 state

uint32_t ::state

Brief description

UART status.

11.1.546.2 p_lower_lvl_uart

uart_instance_t::p_lower_lvl_uart

Brief description

Pointer to UART interface (copied from cfg)

11.1.546.3 mutex

TX_MUTEX ::mutex[2]

Brief description

Pointer to the mutex object for UART resource mutual exclusion.

11.1.546.4 eventflag

TX_EVENT_FLAGS_GROUP ::eventflag[2]

Brief description

Pointer to the event flag object for UART data transfer.

11.1.546.5 queue

```
TX_QUEUE ::queue
```

Brief description

Queue for reading.

11.1.546.6 queue_mem

```
uint32_t ::queue_mem[SF_UART_COMMS_CFG_QUEUE_SIZE_WORDS]
```

Brief description

Queue memory.

11.1.547 sf_wifi_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_wifi_ctrl_t *p_ctrl, sf_wifi_cfg_t const *const p_cfg)
    ssp_err_t(*  close)(sf_wifi_ctrl_t *const p_ctrl)
    ssp_err_t(*  multicastListAdd)(sf_wifi_ctrl_t *const p_ctrl, uint8_t const
*const p_mac_addr)
    ssp_err_t(*  multicastListDelete)(sf_wifi_ctrl_t *const p_ctrl, uint8_t const
*const p_mac_addr)
    ssp_err_t(*  statisticsGet)(sf_wifi_ctrl_t *const p_ctrl, sf_wifi_stats_t
*const p_wifi_device_stats)
    ssp_err_t(*  transmit)(sf_wifi_ctrl_t *const p_ctrl, uint8_t *const p_buf,
uint32_t length)
    ssp_err_t(*  provisioningSet)(sf_wifi_ctrl_t *const p_ctrl,
sf_wifi_provisioning_t const *const p_wifi_provisioning)
    ssp_err_t(*  provisioningGet)(sf_wifi_ctrl_t *const p_ctrl,
sf_wifi_provisioning_t *const p_wifi_provisioning)
    ssp_err_t(*  infoGet)(sf_wifi_ctrl_t *const p_ctrl, sf_wifi_info_t *const
p_wifi_info)
    ssp_err_t(*  scan)(sf_wifi_ctrl_t *const p_ctrl, sf_wifi_scan_t *const
p_scan, uint8_t *const p_cnt)
    ssp_err_t(*  ACLAdd)(sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const
p_mac)
    ssp_err_t(*  ACLDelete)(sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const
p_mac)
    ssp_err_t(*  macAddressGet)(sf_wifi_ctrl_t *const p_ctrl, uint8_t *const
p_mac)
    ssp_err_t(*  macAddressSet)(sf_wifi_ctrl_t *const p_ctrl, uint8_t const
*const p_mac)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_wifi_api_t
```

11.1.548 sf_wifi_callback_args_t

```
typedef struct{
    sf_wifi_event_t  event
    uint8_t *  p_data
    uint32_t  length
    uint8_t  mac_addr[SF_WIFI_MAC_ADDR_LENGTH]
    void const  *  p_context
} sf_wifi_callback_args_t
```

11.1.548.1 event

sf_wifi_event_t::event

Brief description

Event code.

11.1.548.2 p_data

uint8_t* sf_wifi_callback_args_t::p_data

Brief description

Packet data.

11.1.548.3 length

uint32_t sf_wifi_callback_args_t::length

Brief description

Packet Data length.

11.1.548.4 mac_addr

uint8_t sf_wifi_callback_args_t::mac_addr[SF_WIFI_MAC_ADDR_LENGTH]

Brief description

Client station MAC address.

11.1.548.5 p_context

void const* sf_wifi_callback_args_t::p_context

Brief description

Context provided to user during callback.

11.1.549 sf_wifi_cfg_t

```
typedef struct{
    uint8_t  mac_addr[6]
    sf_wifi_interface_hw_mode_t  hw_mode
    uint8_t  tx_power
    sf_wifi_rts_t  rts
    uint16_t  fragmentation
    uint8_t  dtim
    sf_wifi_high_throughput_t  high_throughput
    sf_wifi_preamble_t  preamble
    sf_wifi_wmm_t  wmm
    uint8_t  max_stations
    sf_wifi_ssid_broadcast_t  ssid_broadcast
    sf_wifi_access_control_t  access_control
    uint32_t  beacon
    uint32_t  station_inactivity_timeout
    sf_wifi_wds_t  wds
    void *  p_buffer_pool_rx
    sf_wifi_mandatory_high_throughput_t  req_high_throughput
    void(*  p_callback)(sf_wifi_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} sf_wifi_cfg_t
```

11.1.549.1 mac_addr

```
uint8_t sf_wifi_cfg_t::mac_addr[6]
```

Brief description

MAC address of WiFi Device.

11.1.549.2 hw_mode

```
sf_wifi_interface_hw_mode_t::hw_mode
```

Brief description

Modulation type: 11a/b/g/n.

11.1.549.3 tx_power

```
uint8_t sf_wifi_cfg_t::tx_power
```

Brief description

Sets transmit power in dBm.

11.1.549.4 rts

sf_wifi_rts_t::rts

Brief description

RTS/CTS handshake flag.

11.1.549.5 fragmentation

uint16_t sf_wifi_cfg_t::fragmentation

Brief description

Fragmentation threshold.

11.1.549.6 dtim

uint8_t sf_wifi_cfg_t::dtim

Brief description

Delivery traffic indication message interval.

11.1.549.7 high_throughput

sf_wifi_high_throughput_t::high_throughput

Brief description

High-throughput mode. Only valid for 802.11n.

11.1.549.8 preamble

sf_wifi_preamble_t::preamble

Brief description

Preamble type.

11.1.549.9 wmm

sf_wifi_wmm_t::wmm

Brief description

WiFi Multimedia Mode flag. If enabled, also requires.

11.1.549.10 max_stations

uint8_t sf_wifi_cfg_t::max_stations

Brief description

Maximum permitted stations. Valid in AP mode only.

11.1.549.11 ssid_broadcast

`sf_wifi_ssid_broadcast_t::ssid_broadcast`

Brief description

SSID broadcast flag. Valid in AP mode only.

11.1.549.12 access_control

`sf_wifi_access_control_t::access_control`

Brief description

Mode of access control MAC list.

11.1.549.13 beacon

`uint32_t sf_wifi_cfg_t::beacon`

Brief description

Beacon interval. Valid in AP mode only.

11.1.549.14 station_inactivity_timeout

`uint32_t sf_wifi_cfg_t::station_inactivity_timeout`

Brief description

Station inactivity timeout value. Valid in AP mode only.

11.1.549.15 wds

`sf_wifi_wds_t::wds`

Brief description

WDS flag. Valid in AP mode only.

11.1.549.16 p_buffer_pool_rx

`void* sf_wifi_cfg_t::p_buffer_pool_rx`

Brief description

Pointer to Network stack Rx buffer pool.

11.1.549.17 req_high_throughput

`sf_wifi_mandatory_high_throughput_t::req_high_throughput`

Brief description

Only allow HT mode. Valid in AP mode only.

11.1.549.18 p_callback

```
void(* sf_wifi_cfg_t::p_callback) (sf_wifi_callback_args_t *p_args)
```

Brief description

Pointer to callback function.

11.1.549.19 p_context

```
void const* sf_wifi_cfg_t::p_context
```

Brief description

User defined context passed into callback function.

11.1.549.20 p_extend

```
void const* sf_wifi_cfg_t::p_extend
```

Brief description

Instance specific configuration.

11.1.550 sf_wifi_ctrl_t

```
typedef struct{
    void *  p_driver_handle
} sf_wifi_ctrl_t
```

11.1.550.1 p_driver_handle

```
void* sf_wifi_ctrl_t::p_driver_handle
```

Detailed description

Storage for information needed for each WiFi device driver in the system.

11.1.551 sf_wifi_info_t

```
typedef struct{
    uint8_t *  p_chipset
    uint16_t  rssi
    uint16_t  noise_level
    uint16_t  link_quality
} sf_wifi_info_t
```

11.1.551.1 p_chipset

```
uint8_t* sf_wifi_info_t::p_chipset
```

Brief description

Pointer to sting showing WiFi chipset/driver information.

11.1.551.2 rssi

```
uint16_t sf_wifi_info_t::rssi
```

Brief description

Received signal strength indication.

11.1.551.3 noise_level

```
uint16_t sf_wifi_info_t::noise_level
```

Brief description

Signal to noise ratio.

11.1.551.4 link_quality

```
uint16_t sf_wifi_info_t::link_quality
```

Brief description

Signal strength / quality.

11.1.552 sf_wifi_instance_t

```
typedef struct{
    sf_wifi_ctrl_t *  p_ctrl
    sf_wifi_cfg_t const  *  p_cfg
    sf_wifi_api_t const  *  p_api
} sf_wifi_instance_t
```

11.1.552.1 p_ctrl

sf_wifi_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.552.2 p_cfg

sf_wifi_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.552.3 p_api

sf_wifi_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.553 sf_wifi_ip_addr_t

```
typedef struct{
    sf_wifi_ip_addr_version_t  version
    uint32_t  v4
    uint32_t  v6[4]
    union{}                addr
} sf_wifi_ip_addr_t
```

11.1.553.1 version

sf_wifi_ip_addr_version_t::version

Brief description

IP Address Version : v4 or v6.

11.1.553.2 v4

uint32_t sf_wifi_ip_addr_t::v4

11.1.553.3 v6

uint32_t sf_wifi_ip_addr_t::v6[4]

11.1.553.4 addr

See source code for the definition of this member.

Brief description

IP address.

11.1.554 sf_wifi_nsal_callback_args_t

```
typedef struct{
    NX_INTERFACE *  p_interface
    sf_wifi_nsal_cfg_t *  p_wifi_nsal_cfg
} sf_wifi_nsal_callback_args_t
```

11.1.554.1 p_interface

NX_INTERFACE* sf_wifi_nsal_callback_args_t::p_interface

Brief description

Pointer to NetX interface.

11.1.554.2 p_wifi_nsal_cfg

sf_wifi_nsal_cfg_t::p_wifi_nsal_cfg

Brief description

pointer to NSAL configuration

11.1.555 sf_wifi_nsal_cfg_t

```
typedef struct{
    sf_wifi_nsal_zero_copy_t  tx_zero_copy
    sf_wifi_nsal_zero_copy_t  rx_zero_copy
    uint8_t *  p_tx_packet_buffer
} sf_wifi_nsal_cfg_t
```

11.1.555.1 tx_zero_copy

sf_wifi_nsal_zero_copy_t::tx_zero_copy

Brief description

Transmit path zero copy support.

11.1.555.2 rx_zero_copy

sf_wifi_nsal_zero_copy_t::rx_zero_copy

Brief description

Receive path zero copy support.

11.1.555.3 p_tx_packet_buffer

uint8_t* sf_wifi_nsal_cfg_t::p_tx_packet_buffer

Brief description

Pointer to Tx buffer used to consolidate data from chained NetX packets.

11.1.556 sf_wifi_onchip_stack_api_t

```
typedef struct{
    ssp_err_t(*  open)(sf_wifi_onchip_stack_ctrl_t *p_ctrl,
sf_wifi_onchip_stack_cfg_t const *const p_cfg)
    ssp_err_t(*  close)(sf_wifi_onchip_stack_ctrl_t *const p_ctrl)
    ssp_err_t(*  ipAddressCfg)(sf_wifi_onchip_stack_ctrl_t *const p_ctrl,
sf_wifi_onchip_stack_ip_cfg_t *const p_cfg)
    ssp_err_t(*  dhcpServerStart)(sf_wifi_onchip_stack_ctrl_t *const p_ctrl,
sf_wifi_ip_addr_t const *const p_start_ip, sf_wifi_ip_addr_t const *const
p_end_ip)
    ssp_err_t(*  dhcpServerStop)(sf_wifi_onchip_stack_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} sf_wifi_onchip_stack_api_t
```

11.1.557 sf_wifi_onchip_stack_cfg_t

```
typedef struct{
    sf_wifi_instance_t const  *  p_lower_lvl_wifi
    void *  p_extend
} sf_wifi_onchip_stack_cfg_t
```

11.1.557.1 p_lower_lvl_wifi

sf_wifi_instance_t::p_lower_lvl_wifi

Brief description

Pointer to SF WiFi instance.

11.1.557.2 p_extend

```
void* sf_wifi_onchip_stack_cfg_t::p_extend
```

Brief description

Extended configuration.

11.1.558 sf_wifi_onchip_stack_ctrl_t

```
typedef struct{
    sf_wifi_instance_t *  p_lower_lvl_wifi
} sf_wifi_onchip_stack_ctrl_t
```

11.1.558.1 p_lower_lvl_wifi

```
sf_wifi_instance_t::p_lower_lvl_wifi
```

Brief description

Pointer to SF WiFi instance.

Detailed description

Storage for information needed for each WiFi device driver in the system.

11.1.559 sf_wifi_onchip_stack_instance_t

```
typedef struct{
    sf_wifi_onchip_stack_ctrl_t *  p_ctrl
    sf_wifi_onchip_stack_cfg_t const  *  p_cfg
    sf_wifi_onchip_stack_api_t const  *  p_api
} sf_wifi_onchip_stack_instance_t
```

11.1.559.1 p_ctrl

```
sf_wifi_onchip_stack_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.559.2 p_cfg

```
sf_wifi_onchip_stack_cfg_t::p_cfg
```

Brief description

Pointer to the configuration structure for this instance.

11.1.559.3 p_api

sf_wifi_onchip_stack_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.560 sf_wifi_onchip_stack_ip_cfg_t

```
typedef struct{
    sf_wifi_onchip_stack_ip_addr_mode_t  ip_addr_mode
    sf_wifi_ip_addr_t  ip_addr
    sf_wifi_ip_addr_t  netmask
    sf_wifi_ip_addr_t  gateway
} sf_wifi_onchip_stack_ip_cfg_t
```

11.1.560.1 ip_addr_mode

sf_wifi_onchip_stack_ip_addr_mode_t::ip_addr_mode

Brief description

Addressing mode.

11.1.560.2 ip_addr

sf_wifi_ip_addr_t::ip_addr

Brief description

Interface IP address.

11.1.560.3 netmask

sf_wifi_ip_addr_t::netmask

Brief description

Interface netmask.

11.1.560.4 gateway

sf_wifi_ip_addr_t::gateway

Brief description

Interface Gateway.

11.1.561 sf_wifi_provisioning_t

```
typedef struct{
    sf_wifi_interface_mode_t  mode
    uint8_t  channel
    uint8_t  ssid[SF_WIFI_SSID_LENGTH+1]
    sf_wifi_security_type_t  security
    sf_wifi_encryption_type_t  encryption
    uint8_t  key[SF_WIFI_SECURITY_KEY_LENGTH]
    void(*  p_callback)(sf_wifi_callback_args_t *p_args)
} sf_wifi_provisioning_t
```

11.1.561.1 mode

sf_wifi_interface_mode_t::mode

Brief description

Select AP or Client mode.

11.1.561.2 channel

uint8_t sf_wifi_provisioning_t::channel

Brief description

RF Channel number.

11.1.561.3 ssid

uint8_t sf_wifi_provisioning_t::ssid[SF_WIFI_SSID_LENGTH+1]

Brief description

SSID.

11.1.561.4 security

sf_wifi_security_type_t::security

Brief description

Security type.

11.1.561.5 encryption

sf_wifi_encryption_type_t::encryption

Brief description

Encryption type.

11.1.561.6 key

```
uint8_t sf_wifi_provisioning_t::key[SF_WIFI_SECURITY_KEY_LENGTH]
```

Brief description

Pre-shared key.

11.1.561.7 p_callback

```
void(* sf_wifi_provisioning_t::p_callback) (sf_wifi_callback_args_t *p_args)
```

Brief description

Pointer to Connection status notification callback function.

11.1.562 sf_wifi_scan_t

```
typedef struct{
    sf_wifi_interface_hw_mode_t  hw_mode
    uint8_t  rssi
    uint8_t  ssid[SF_WIFI_SSID_LENGTH+1]
    uint8_t  bssid[SF_WIFI_MAC_ADDR_LENGTH]
    uint8_t  channel
    sf_wifi_security_type_t  security
    sf_wifi_encryption_type_t  encryption
    sf_wifi_bss_type_t  bss_type
} sf_wifi_scan_t
```

11.1.562.1 hw_mode

```
sf_wifi_interface_hw_mode_t::hw_mode
```

Brief description

Hardware mode 802.11a/b/g/n.

11.1.562.2 rssi

```
uint8_t sf_wifi_scan_t::rssi
```

Brief description

Signal Strength.

11.1.562.3 ssid

```
uint8_t sf_wifi_scan_t::ssid[SF_WIFI_SSID_LENGTH+1]
```

Brief description

SSID name.

11.1.562.4 bssid

```
uint8_t sf_wifi_scan_t::bssid[SF_WIFI_MAC_ADDR_LENGTH]
```

Brief description

Basic Service Set Identification (i.e. MAC address of Access Point)

11.1.562.5 channel

```
uint8_t sf_wifi_scan_t::channel
```

Brief description

Radio channel that the AP beacon was received on.

11.1.562.6 security

```
sf_wifi_security_type_t::security
```

Brief description

Security type.

11.1.562.7 encryption

```
sf_wifi_encryption_type_t::encryption
```

Brief description

Encryption type.

11.1.562.8 bss_type

```
sf_wifi_bss_type_t::bss_type
```

Brief description

Network type.

11.1.563 sf_wifi_stats_t

```
typedef struct{
    uint32_t  rx_pkts
    uint32_t  tx_pkts
    uint32_t  tx_err
} sf_wifi_stats_t
```

11.1.563.1 rx_pkts

```
uint32_t sf_wifi_stats_t::rx_pkts
```

Brief description

Packets received successfully.

11.1.563.2 tx_pkts

```
uint32_t sf_wifi_stats_t::tx_pkts
```

Brief description

Packets transmitted successfully.

11.1.563.3 tx_err

```
uint32_t sf_wifi_stats_t::tx_err
```

Brief description

Transmit errors.

11.1.564 slcdc_api_t

```
typedef struct{
    ssp_err_t(*  open)(slcdc_ctrl_t *const p_ctrl, slcdc_cfg_t const *const
p_cfg)
    ssp_err_t(*  write)(slcdc_ctrl_t *const p_ctrl, slcdc_size_t const
start_segment, slcdc_size_t const *const p_data, slcdc_size_t const segment_count)
    ssp_err_t(*  modify)(slcdc_ctrl_t *const p_ctrl, slcdc_size_t const segment,
slcdc_size_t const data_mask, slcdc_size_t const data)
    ssp_err_t(*  start)(slcdc_ctrl_t *const p_ctrl)
    ssp_err_t(*  stop)(slcdc_ctrl_t *const p_ctrl)
    ssp_err_t(*  contrastIncrease)(slcdc_ctrl_t *const p_ctrl)
    ssp_err_t(*  contrastDecrease)(slcdc_ctrl_t *const p_ctrl)
    ssp_err_t(*  setdisplayArea)(slcdc_ctrl_t *const p_ctrl, slcdc_display_area_t
const display_area)
    ssp_err_t(*  close)(slcdc_ctrl_t *const p_ctrl)
```

```
    ssp_err_t(*  versionGet)(ssp_version_t *p_version)
} slcdc_api_t
```

11.1.565 slcdc_cfg_t

```
typedef struct{
    slcdc_display_clock_t  slcdc_clock
    slcdc_clk_div_t  slcdc_clock_setting
    slcdc_bias_method_t  bias_method
    slcdc_time_slice_t  time_slice
    slcdc_wave_form_t  wave_form
    slcdc_drive_volt_gen_t  drive_volt_gen
} slcdc_cfg_t
```

11.1.565.1 slcdc_clock

slcdc_display_clock_t::slcdc_clock

Brief description

LCD clock source (LCDSCKSEL)

11.1.565.2 slcdc_clock_setting

slcdc_clk_div_t::slcdc_clock_setting

Brief description

LCD clock setting (LCDC0)

11.1.565.3 bias_method

slcdc_bias_method_t::bias_method

Brief description

LCD display bias method select (LBAS bit).

11.1.565.4 time_slice

slcdc_time_slice_t::time_slice

Brief description

Time slice of LCD display select (LDTY bit)

11.1.565.5 wave_form

slcdc_wave_form_t::wave_form

Brief description

LCD display waveform select (LWAVE bit).

11.1.565.6 drive_volt_gen

`slcdc_drive_volt_gen_t::drive_volt_gen`

Brief description

LCD Drive Voltage Generator Select (MDSTET bit).

11.1.566 slcdc_instance_ctrl_t

```
typedef struct{
    slcdc_display_state_t  state
    slcdc_cfg_t  info
    void const  *  p_context
    R_LCD_Type *  p_reg
} slcdc_instance_ctrl_t
```

11.1.566.1 state

`slcdc_display_state_t::state`

Brief description

Status of SLCD module.

11.1.566.2 info

`slcdc_cfg_t::info`

Brief description

SLCDC config info.

11.1.566.3 p_context

`void const* ::p_context`

Brief description

Pointer to the higher level device context.

11.1.566.4 p_reg

`R_LCD_Type* ::p_reg`

Brief description

Pointer to register base address.

11.1.567 slcdc_instance_t

```
typedef struct{
    slcdc_ctrl_t *  p_ctrl
    slcdc_cfg_t const  *  p_cfg
    slcdc_api_t const  *  p_api
} slcdc_instance_t
```

11.1.567.1 p_ctrl

slcdc_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.567.2 p_cfg

slcdc_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.567.3 p_api

slcdc_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.568 sockaddr

```
typedef struct{
    short  sin_family
    unsigned short  sin_port
    struct in_addr  sin_addr
    char  sin_zero[8]
    USHORT  sa_family
    UCHAR  sa_data[14]
} sockaddr
```

11.1.568.1 sin_family

```
short sockaddr::sin_family
```

Brief description

Address family.

11.1.568.2 sin_port

```
unsigned short sockaddr::sin_port
```

Brief description

Port number.

11.1.568.3 sin_addr

```
in_addr::sin_addr
```

Brief description

IP Address.

11.1.568.4 sin_zero

```
char sockaddr::sin_zero
```

Brief description

zero this if you want to

Detailed description

Zero this if you want to.

11.1.568.5 sa_family

```
USHORT sockaddr::sa_family
```

11.1.568.6 sa_data

```
UCHAR sockaddr::sa_data
```

11.1.569 sockaddr_in

```
typedef struct{
    uint16_t  sin_family
    uint16_t  sin_port
    struct in_addr  sin_addr
```

```
    int8_t  sin_zero[8]
    USHORT  sin_family
    USHORT  sin_port
    CHAR  sin_zero[8]
} sockaddr_in
```

11.1.569.1 sin_family

```
USHORT sockaddr_in::sin_family
```

Brief description

Internet Protocol (AF_INET)

11.1.569.2 sin_port

```
USHORT sockaddr_in::sin_port
```

Brief description

Address port (16 bits)

11.1.569.3 sin_addr

```
in_addr::sin_addr
```

Brief description

Internet address (32 bits)

11.1.569.4 sin_zero

```
CHAR sockaddr_in::sin_zero
```

Brief description

Not used.

Detailed description

Not used structure member.

11.1.569.5 sin_family

```
USHORT sockaddr_in::sin_family
```

11.1.569.6 sin_port

```
USHORT sockaddr_in::sin_port
```

11.1.569.7 sin_zero

```
CHAR sockaddr_in::sin_zero[8]
```

11.1.570 spi_api_t

```
typedef struct{
    ssp_err_t(*  open)(spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg)
    ssp_err_t(*  read)(spi_ctrl_t *const p_ctrl, void const *p_dest, uint32_t
const length, spi_bit_width_t const bit_width)
    ssp_err_t(*  write)(spi_ctrl_t *const p_ctrl, void const *p_src, uint32_t
const length, spi_bit_width_t const bit_width)
    ssp_err_t(*  writeRead)(spi_ctrl_t *const p_ctrl, void const *p_src, void
const *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
    ssp_err_t(*  close)(spi_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *p_version)
} spi_api_t
```

11.1.571 spi_callback_args_t

```
typedef struct{
    uint32_t  channel
    spi_event_t  event
    void const  *  p_context
} spi_callback_args_t
```

11.1.571.1 channel

```
uint32_t spi_callback_args_t::channel
```

Brief description

Device channel number.

11.1.571.2 event

```
spi_event_t::event
```

Brief description

Event code.

11.1.571.3 p_context

```
void const* spi_callback_args_t::p_context
```

Brief description

Context provided to user during callback.

11.1.572 spi_cfg_t

```
typedef struct{
    uint8_t  channel
    uint8_t  rxi_ipl
    uint8_t  txi_ipl
    uint8_t  tei_ipl
    uint8_t  eri_ipl
    spi_mode_t  operating_mode
    spi_clk_phase_t  clk_phase
    spi_clk_polarity_t  clk_polarity
    spi_mode_fault_t  mode_fault
    spi_bit_order_t  bit_order
    uint32_t  bitrate
    transfer_instance_t const  *  p_transfer_tx
    transfer_instance_t const  *  p_transfer_rx
    void(*  p_callback)(spi_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} spi_cfg_t
```

11.1.572.1 channel

```
uint8_t spi_cfg_t::channel
```

Brief description

Channel number to be used.

11.1.572.2 rxi_ipl

```
uint8_t spi_cfg_t::rxi_ipl
```

Brief description

Receive interrupt priority.

11.1.572.3 txi_ipl

```
uint8_t spi_cfg_t::txi_ipl
```

Brief description

Transmit interrupt priority.

11.1.572.4 tei_ipl

`uint8_t spi_cfg_t::tei_ipl`

Brief description

Transmit end interrupt priority.

11.1.572.5 eri_ipl

`uint8_t spi_cfg_t::eri_ipl`

Brief description

Error interrupt priority.

11.1.572.6 operating_mode

`spi_mode_t::operating_mode`

Brief description

Select master or slave operating mode.

11.1.572.7 clk_phase

`spi_clk_phase_t::clk_phase`

Brief description

Data sampling on odd or even clock edge.

11.1.572.8 clk_polarity

`spi_clk_polarity_t::clk_polarity`

Brief description

Clock level when idle.

11.1.572.9 mode_fault

`spi_mode_fault_t::mode_fault`

Brief description

Mode fault error (master/slave conflict) flag.

11.1.572.10 bit_order

`spi_bit_order_t::bit_order`

Brief description

Select to transmit MSB/LSB first.

11.1.572.11 bitrate

```
uint32_t spi_cfg_t::bitrate
```

Brief description

Bits Per Second.

11.1.572.12 p_transfer_tx

```
transfer_instance_t::p_transfer_tx
```

Brief description

To use SPI DTC/DMA write transfer, link a DTC/DMA instance here. Set to NULL if unused.

11.1.572.13 p_transfer_rx

```
transfer_instance_t::p_transfer_rx
```

Brief description

To use SPI DTC/DMA read transfer, link a DTC/DMA instance here. Set to NULL if unused.

11.1.572.14 p_callback

```
void(* spi_cfg_t::p_callback) (spi_callback_args_t *p_args)
```

Brief description

Pointer to user callback function.

11.1.572.15 p_context

```
void const* spi_cfg_t::p_context
```

Brief description

User defined context passed to callback function.

11.1.572.16 p_extend

```
void const* spi_cfg_t::p_extend
```

Brief description

Extended SPI hardware dependent configuration.

11.1.573 spi_instance_t

```
typedef struct{
    spi_ctrl_t *  p_ctrl
    spi_cfg_t const  *  p_cfg
    spi_api_t const  *  p_api
} spi_instance_t
```

11.1.573.1 p_ctrl

spi_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.573.2 p_cfg

spi_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.573.3 p_api

spi_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.574 spi_on_rspi_cfg_t

```
typedef struct{
    rspi_operation_t  rspi_clksyn
    rspi_communication_t  rspi_comm
    rspi_ssl_polarity_t  ssl_polarity
    rspi_loopback_t  loopback
    rspi_mosi_idle_t  mosi_idle
    rspi_parity_t  parity
    rspi_ssl_select_t  ssl_select
    rspi_ssl_level_keep_t  ssl_level_keep
    rspi_clock_delay_t  clock_delay
    rspi_ssl_negation_delay_t  ssl_neg_delay
    rspi_access_delay_t  access_delay
} spi_on_rspi_cfg_t
```

11.1.574.1 rspi_clksyn

rspi_operation_t::rspi_clksyn

Detailed description

Select spi or clock syn mode operation

11.1.574.2 rspi_comm

rspi_communication_t::rspi_comm

Detailed description

Select full-duplex or transmit-only communication

11.1.574.3 ssl_polarity

rspi_ssl_polarity_t::ssl_polarity

Detailed description

Select SSLn signal polarity

11.1.574.4 loopback

rspi_loopback_t::loopback

Detailed description

Select loopback1 and loopback2

11.1.574.5 mosi_idle

rspi_mosi_idle_t::mosi_idle

Detailed description

Select mosi idle fixed value and selection

11.1.574.6 parity

rspi_parity_t::parity

Detailed description

Select parity and enable/disable parity

11.1.574.7 ssl_select

rspi_ssl_select_t::ssl_select

Detailed description

Select which slave to use;0-SSL0;1-SSL1;2-SSL2;3-SSL3

11.1.574.8 ssl_level_keep

rspi_ssl_level_keep_t::ssl_level_keep

Detailed description

Select SSL level after transfer completion;0-negate;1-keeps

11.1.574.9 clock_delay

rspi_clock_delay_t::clock_delay

Detailed description

Select clock delay from 0 to 7

11.1.574.10 ssl_neg_delay

rspi_ssl_negation_delay_t::ssl_neg_delay

Detailed description

Select Slave elect negation delay from 0 to 7

11.1.574.11 access_delay

rspi_access_delay_t::access_delay

Detailed description

Select next access delay from 0 to 7

11.1.575 ssi_instance_ctrl_t

```
typedef struct{
    void(*  p_callback)(i2s_callback_args_t *p_args)
    void const  *  p_context
    R_SSI0_Type *  p_reg
    timer_instance_t const  *  p_timer
    transfer_instance_t const  *  p_transfer_tx
    transfer_instance_t const  *  p_transfer_rx
    uint32_t const  *  p_tx_src
    uint32_t  tx_src_bytes
    uint32_t *  p_rx_dest
    uint32_t  rx_dest_bytes
    uint32_t  sampling_freq_hz
```

```
    uint8_t  channel
    uint8_t  fifo_access_bytes
    uint8_t  stop_requested_tx
    uint8_t  stop_requested_rx
    uint8_t  tx_in_progress
    uint8_t  tx_in_use
    uint8_t  rx_in_use
    uint8_t  zeros_written
    IRQn_Type  txi_irq
    IRQn_Type  rxi_irq
    IRQn_Type  int_irq
    uint32_t  open
} ssi_instance_ctrl_t
```

11.1.575.1 p_callback

```
void(* ::p_callback) ( *p_args)
```

Detailed description

Callback provided when an I2S ISR occurs. NULL indicates no CPU interrupt.

11.1.575.2 p_context

```
void const* ::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in i2s_callback_args_t.

11.1.575.3 p_reg

```
R_SSI0_Type* ::p_reg
```

Brief description

Pointer to SSI register base address.

11.1.575.4 p_timer

```
timer_instance_t::p_timer
```

Brief description

Timer used to generate audio clock.

11.1.575.5 p_transfer_tx

```
transfer_instance_t::p_transfer_tx
```

Brief description

Transfer used for hardware acceleration during write.

11.1.575.6 p_transfer_rx

`transfer_instance_t::p_transfer_rx`

Brief description

Transfer used for hardware acceleration during read.

11.1.575.7 p_tx_src

```
uint32_t const* ::p_tx_src
```

Detailed description

Source buffer pointer used to fill hardware FIFO from transmit ISR.

11.1.575.8 tx_src_bytes

```
uint32_t ::tx_src_bytes
```

Detailed description

Size of source buffer used to fill hardware FIFO from transmit ISR.

11.1.575.9 p_rx_dest

```
uint32_t* ::p_rx_dest
```

Detailed description

Destination buffer pointer used to fill from hardware FIFO in receive ISR.

11.1.575.10 rx_dest_bytes

```
uint32_t ::rx_dest_bytes
```

Detailed description

Size of destination buffer used to fill from hardware FIFO in receive ISR.

11.1.575.11 sampling_freq_hz

```
uint32_t ::sampling_freq_hz
```

Brief description

Sampling frequency in Hertz.

11.1.575.12 channel

```
uint8_t ::channel
```

Brief description

Channel number.

11.1.575.13 fifo_access_bytes

```
uint8_t ::fifo_access_bytes
```

Brief description

Byte access to FIFO.

11.1.575.14 stop_requested_tx

```
uint8_t ::stop_requested_tx
```

Brief description

Stops I2S after transmit is complete.

11.1.575.15 stop_requested_rx

```
uint8_t ::stop_requested_rx
```

Brief description

Stops I2S after receive is complete.

11.1.575.16 tx_in_progress

```
uint8_t ::tx_in_progress
```

Brief description

True if a transmit transfer is in progress.

11.1.575.17 tx_in_use

```
uint8_t ::tx_in_use
```

Brief description

True if a transmission is in progress.

11.1.575.18 rx_in_use

```
uint8_t ::rx_in_use
```

Brief description

True if a reception is in progress.

11.1.575.19 zeros_written

```
uint8_t ::zeros_written
```

Brief description

True if zeros have been transmitted.

11.1.575.20 txi_irq

```
IRQn_Type ::txi_irq
```

Brief description

Transmit IRQ number.

11.1.575.21 rxi_irq

```
IRQn_Type ::rxi_irq
```

Brief description

Receive IRQ number.

11.1.575.22 int_irq

```
IRQn_Type ::int_irq
```

Brief description

Idle/Error IRQ number.

11.1.575.23 open

```
uint32_t ::open
```

Brief description

Whether or not this control block is initialized.

11.1.576 st_sf_audio_playback_instance_ctrl

```
typedef struct{
    uint32_t  open
    TX_THREAD *  p_owner
    void(*  p_callback)(sf_message_callback_args_t *p_args)
```

```
    uint8_t  class_instance
    uint32_t  samples_remaining
    uint32_t  samples_total
    uint32_t  index
    uint32_t  end
    sf_audio_playback_data_t *  p_data[2]
    sf_audio_playback_status_t  status
    sf_audio_playback_common_instance_ctrl_t *  p_common_ctrl
} st_sf_audio_playback_instance_ctrl
```

11.1.576.1 open

```
uint32_t ::open
```

Brief description

Used to determine if driver is initialized.

11.1.576.2 p_owner

```
TX_THREAD* ::p_owner
```

Detailed description

Pointer to thread that began the stream at this index. Used to ensure multiple threads don't interleave data on the same stream.

11.1.576.3 p_callback

```
void(* ::p_callback) ( *p_args)
```

Detailed description

Callback called when playback of a buffer passed to start is complete.

11.1.576.4 class_instance

```
uint8_t ::class_instance
```

Brief description

Class instance used to identify the stream to the messaging framework.

11.1.576.5 samples_remaining

```
uint32_t ::samples_remaining
```

Brief description

Internal state of data samples remaining for this stream.

11.1.576.6 samples_total

`uint32_t ::samples_total`

Brief description

Total number of samples to play (independent of sample size like 8/12/16).

11.1.576.7 index

`uint32_t ::index`

Brief description

Internal state of current data index for this stream.

11.1.576.8 end

`uint32_t ::end`

Brief description

Used to track completion of looped playback.

11.1.576.9 p_data

`sf_audio_playback_data_t::p_data`

Brief description

Audio data read from queue.

11.1.576.10 status

`sf_audio_playback_status_t::status`

Brief description

Status of current stream.

11.1.576.11 p_common_ctrl

`sf_audio_playback_common_instance_ctrl_t::p_common_ctrl`

Detailed description

Pointer to the hardware control block used by this stream.

11.1.577 st_sf_ble_prf_htp_meas_intv_val_t

```
typedef struct{
    uint16_t  conhdl
    uint16_t  intv
} st_sf_ble_prf_htp_meas_intv_val_t
```

11.1.577.1 conhdl

```
uint16_t st_sf_ble_prf_htp_meas_intv_val_t::conhdl
```

Brief description

Connection handle.

11.1.577.2 intv

```
uint16_t st_sf_ble_prf_htp_meas_intv_val_t::intv
```

Brief description

Interval value.

11.1.578 st_sf_touch_ctsu_slider_hdl

```
typedef struct{
    uint32_t  open
    sf_touch_ctsu_slider_id_t  id
    sf_touch_ctsu_slider_state_t  state
    sf_slider_type_t  type
    uint32_t  num_slider_channels
    ctsu_channel_pair_t const  *  p_slider_channels
    int32_t const  *  p_normalization
    int32_t *  p_channel_average
    uint32_t *  p_offset
    uint16_t *  p_slider_scount
    uint16_t *  p_slider_baseline
    int32_t *  p_slider_delta
    uint32_t  position
    int32_t  prev_sum
    int32_t  max_slider_value
    ssp_err_t(*  p_update)(sf_touch_ctsu_slider_hdl_t *const hdl,
sf_touch_ctsu_instance_t const *const p_lower_lvl_touch_framework, uint32_t
*p_pos, sf_touch_ctsu_slider_state_t *const p_state)
    uint64_t  bit_mask[SF_TOUCH_CTSU_SLIDER_BIT_MASK_ARRAY_SIZE]
    uint16_t  slider_norm_max
    int32_t  slider_threshold
    int32_t  channel_average_weight
```

```
    int32_t  prev_sum_weight
    int32_t  cutoff
} st_sf_touch_ctsu_slider_hdl
```

11.1.578.1 open

uint32_t ::open

Detailed description

Indicate that slider has been opened

11.1.578.2 id

sf_touch_ctsu_slider_id_t::id

Detailed description

Unique identifier for slider.

11.1.578.3 state

sf_touch_ctsu_slider_state_t::state

Detailed description

Represents the current state of the slider.

11.1.578.4 type

sf_slider_type_t::type

Detailed description

Linear or circular (wheel)

11.1.578.5 num_slider_channels

uint32_t ::num_slider_channels

11.1.578.6 p_slider_channels

ctsu_channel_pair_t::p_slider_channels

Detailed description

Define the channel/channel pairs which make up a slider.

11.1.578.7 p_normalization

```
int32_t const* ::p_normalization
```

11.1.578.8 p_channel_average

```
int32_t* ::p_channel_average
```

11.1.578.9 p_offset

```
uint32_t* ::p_offset
```

11.1.578.10 p_slider_scount

```
uint16_t* ::p_slider_scount
```

11.1.578.11 p_slider_baseline

```
uint16_t* ::p_slider_baseline
```

11.1.578.12 p_slider_delta

```
int32_t* ::p_slider_delta
```

11.1.578.13 position

```
uint32_t ::position
```

Detailed description

Calculated position.

11.1.578.14 prev_sum

```
int32_t ::prev_sum
```

Detailed description

Used to store the running average of previous sum in position calculation

11.1.578.15 max_slider_value

```
int32_t ::max_slider_value
```

Detailed description

Maximum slider value, must be greater than 0; Minimum is always 0

11.1.578.16 p_update

```
(* ::p_update) ( *const hdl, const *const p_lower_lvl_touch_framework, uint32_t
*p_pos, *const p_state)
```

Detailed description

Function to calculate position of touch.

11.1.578.17 bit_mask

```
uint64_t ::bit_mask[SF_TOUCH_CTSU_SLIDER_BIT_MASK_ARRAY_SIZE]
```

Detailed description

Bit mask to be used in update function

11.1.578.18 slider_norm_max

```
uint16_t ::slider_norm_max
```

Detailed description

Individual slider settings that can be modified by the user. Should be the same as in sf_slider_on_ctsu_cfg_tTOUCH_SLIDER_CFG_NORM_MAX

11.1.578.19 slider_threshold

```
int32_t ::slider_threshold
```

Detailed description

Touch threshold. Ensure that value is greater than 0

11.1.578.20 channel_average_weight

```
int32_t ::channel_average_weight
```

Detailed description

Weight of running average of counts for each channel

11.1.578.21 prev_sum_weight

```
int32_t ::prev_sum_weight
```

Detailed description

Weight of running average of previous sum in position calculation, must be greater than 0

11.1.578.22 cutoff

```
int32_t ::cutoff
```

Detailed description

Defines how far below prev_sum running average to get before "SF_TOUCH_SLIDER_STATE_RELEASED" detected

11.1.579 tcon_func_t

```
typedef struct{
    void(*  tcon_select)(R_GLCDC_Type *p_glcd_reg, glcd_tcon_signal_select_t)
    void(*  invert)(R_GLCDC_Type *p_glcd_reg)
} tcon_func_t
```

11.1.579.1 tcon_select

```
void(* ::tcon_select) (R_GLCDC_Type *p_glcd_reg, )
```

11.1.579.2 invert

```
void(* ::invert) (R_GLCDC_Type *p_glcd_reg)
```

11.1.580 timer_api_t

```
typedef struct{
    ssp_err_t(*  open)(timer_ctrl_t *const p_ctrl, timer_cfg_t const *const
p_cfg)
    ssp_err_t(*  stop)(timer_ctrl_t *const p_ctrl)
    ssp_err_t(*  start)(timer_ctrl_t *const p_ctrl)
    ssp_err_t(*  reset)(timer_ctrl_t *const p_ctrl)
    ssp_err_t(*  counterGet)(timer_ctrl_t *const p_ctrl, timer_size_t *const
p_value)
    ssp_err_t(*  periodSet)(timer_ctrl_t *const p_ctrl, timer_size_t const
period, timer_unit_t const unit)
    ssp_err_t(*  dutyCycleSet)(timer_ctrl_t *const p_ctrl, timer_size_t const
duty_cycle, timer_pwm_unit_t const duty_cycle_unit, uint8_t const pin)
    ssp_err_t(*  infoGet)(timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)
    ssp_err_t(*  close)(timer_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
} timer_api_t
```

11.1.581 timer_callback_args_t

```
typedef struct{
    void const  *  p_context
```

```
    timer_event_t  event
} timer_callback_args_t
```

11.1.581.1 p_context

```
void const* timer_callback_args_t::p_context
```

Detailed description

Placeholder for user data. Set in open function in timer_cfg_t.

11.1.581.2 event

```
timer_event_t::event
```

Brief description

The event can be used to identify what caused the callback (overflow or error).

11.1.582 timer_cfg_t

```
typedef struct{
    timer_mode_t  mode
    uint32_t  period
    timer_unit_t  unit
    timer_size_t  duty_cycle
    timer_pwm_unit_t  duty_cycle_unit
    uint8_t  channel
    uint8_t  irq_ipl
    bool  autostart
    void(*  p_callback)(timer_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} timer_cfg_t
```

11.1.582.1 mode

```
timer_mode_t::mode
```

Brief description

Select enumerated value from timer_mode_t.

11.1.582.2 period

```
uint32_t timer_cfg_t::period
```

Detailed description

Defines when the timer should expire. For a free running counter, set to TIMER_MAX_CLOCK with unit TIMER_UNIT_PERIOD_RAW_COUNTS

11.1.582.3 unit

`timer_unit_t::unit`

Brief description

Units of period.

11.1.582.4 duty_cycle

`timer_size_t::duty_cycle`

Brief description

Duty cycle in units duty_cycle_unit.

11.1.582.5 duty_cycle_unit

`timer_pwm_unit_t::duty_cycle_unit`

Brief description

Units of duty_cycle.

11.1.582.6 channel

`uint8_t timer_cfg_t::channel`

Detailed description

Select a channel corresponding to the channel number of the hardware.

11.1.582.7 irq_ipl

`uint8_t timer_cfg_t::irq_ipl`

Brief description

Timer interrupt priority.

11.1.582.8 autostart

`bool timer_cfg_t::autostart`

Detailed description

Whether to start during Open call or not. True: Start during open. False: Don't start during open.

11.1.582.9 p_callback

```
void(* timer_cfg_t::p_callback) (timer_callback_args_t *p_args)
```

Detailed description

Callback provided when a timer ISR occurs. Set to NULL for no CPU interrupt.

11.1.582.10 p_context

```
void const* timer_cfg_t::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in timer_callback_args_t.

11.1.582.11 p_extend

```
void const* timer_cfg_t::p_extend
```

Brief description

Extension parameter for hardware specific settings.

11.1.583 timer_info_t

```
typedef struct{
    timer_direction_t  count_direction
    uint32_t  clock_frequency
    timer_size_t  period_counts
    timer_status_t  status
    elc_event_t  elc_event
} timer_info_t
```

11.1.583.1 count_direction

```
timer_direction_t::count_direction
```

Brief description

Clock counting direction of the timer resource.

11.1.583.2 clock_frequency

```
uint32_t timer_info_t::clock_frequency
```

Brief description

Clock frequency of the timer resource.

11.1.583.3 period_counts

timer_size_t::period_counts

Brief description

Time in clock counts until timer will expire.

11.1.583.4 status

timer_status_t::status

11.1.583.5 elc_event

elc_event_t::elc_event

Brief description

ELC event associated with the count operation of the timer resource.

11.1.584 timer_instance_t

```
typedef struct{
    timer_ctrl_t *  p_ctrl
    timer_cfg_t const  *  p_cfg
    timer_api_t const  *  p_api
} timer_instance_t
```

11.1.584.1 p_ctrl

timer_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.584.2 p_cfg

timer_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.584.3 p_api

timer_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.585 timer_on_agt_cfg_t

```
typedef struct{
    agt_count_source_t  count_source
    bool  agto_output_enabled
    bool  agtio_output_enabled
    bool  output_inverted
    bool  agtoa_output_enable
    bool  agtob_output_enable
} timer_on_agt_cfg_t
```

11.1.585.1 count_source

agt_count_source_t::count_source

Brief description

AGT channel clock source. Valid values are: AGT_CLOCK_PCLKB, AGT_CLOCK_LOCO, AGT_CLOCK_FSUB.

11.1.585.2 agto_output_enabled

bool ::agto_output_enabled

Brief description

AGTO pin is enabled for output compare (true, false)

11.1.585.3 agtio_output_enabled

bool ::agtio_output_enabled

Brief description

AGTIO pin is enabled for output compare (true, false)

11.1.585.4 output_inverted

bool ::output_inverted

Brief description

Output inverted (true, false)

11.1.585.5 agtoa_output_enable

```
bool ::agtoa_output_enable
```

Brief description

Enable comparator A output pin (true, false)

11.1.585.6 agtob_output_enable

```
bool ::agtob_output_enable
```

Brief description

Enable comparator B output pin (true, false)

11.1.586 timer_on_gpt_cfg_t

```
typedef struct{
    gpt_output_pin_t  gtioca
    gpt_output_pin_t  gtiocb
} timer_on_gpt_cfg_t
```

11.1.586.1 gtioca

```
gpt_output_pin_t::gtioca
```

Brief description

Configuration for GPT I/O pin A.

11.1.586.2 gtiocb

```
gpt_output_pin_t::gtiocb
```

Brief description

Configuration for GPT I/O pin B.

11.1.587 transfer_api_t

```
typedef struct{
    ssp_err_t(*  open)(transfer_ctrl_t *const p_ctrl, transfer_cfg_t const *const
p_cfg)
    ssp_err_t(*  reset)(transfer_ctrl_t *const p_ctrl, void const *p_src, void
*p_dest, uint16_t const num_transfers)
    ssp_err_t(*  enable)(transfer_ctrl_t *const p_ctrl)
    ssp_err_t(*  disable)(transfer_ctrl_t *const p_ctrl)
    ssp_err_t(*  start)(transfer_ctrl_t *const p_ctrl, transfer_start_mode_t
```

```
mode)
    ssp_err_t(*  stop)(transfer_ctrl_t *const p_ctrl)
    ssp_err_t(*  infoGet)(transfer_ctrl_t *const p_ctrl, transfer_properties_t
*const p_info)
    ssp_err_t(*  close)(transfer_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_version)
    ssp_err_t(*  blockReset)(transfer_ctrl_t *const p_ctrl, void const *p_src,
void *p_dest, uint16_t const length, transfer_size_t size, uint16_t const
num_transfers)
} transfer_api_t
```

11.1.588 transfer_callback_args_t

```
typedef struct{
    void const  *  p_context
} transfer_callback_args_t
```

11.1.588.1 p_context

```
void const* transfer_callback_args_t::p_context
```

Brief description

Placeholder for user data. Set in r_transfer_t::open function in transfer_cfg_t.

11.1.589 transfer_cfg_t

```
typedef struct{
    transfer_info_t *  p_info
    elc_event_t  activation_source
    bool  auto_enable
    uint8_t  irq_ipl
    void(*  p_callback)(transfer_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} transfer_cfg_t
```

11.1.589.1 p_info

```
transfer_info_t::p_info
```

Detailed description

Pointer to transfer configuration options. If using chain transfer (DTC only), this can be a pointer to an array of chained transfers that will be completed in order.

11.1.589.2 activation_source

```
elc_event_t::activation_source
```

Detailed description

Select which event will trigger the transfer.
NOTE: Select ELC_EVENT_ELC_SOFTWARE_EVENT_0 or ELC_EVENT_ELC_SOFTWARE_EVENT_0 for software activation. When using DTC, these events may only be used once each. DMAC uses internal software start when either of these events are selected.

11.1.589.3 auto_enable

```
bool transfer_cfg_t::auto_enable
```

Detailed description

Select whether the transfer should be enabled after open.

11.1.589.4 irq_ipl

```
uint8_t transfer_cfg_t::irq_ipl
```

Detailed description

Interrupt priority level.
ATTENTION: Unsupported for DTC except when ELC software events are used. DTC transfers trigger the interrupt associated with the activation source.

11.1.589.5 p_callback

```
void(* transfer_cfg_t::p_callback) (transfer_callback_args_t *p_args)
```

Detailed description

Callback for transfer end interrupt. Set to NULL for no CPU interrupt.
ATTENTION: Unsupported for DTC except when ELC software events are used. DTC transfers trigger the interrupt associated with the activation source.

11.1.589.6 p_context

```
void const* transfer_cfg_t::p_context
```

Detailed description

Placeholder for user data. Passed to the user p_callback in transfer_callback_args_t.

11.1.589.7 p_extend

```
void const* transfer_cfg_t::p_extend
```

Brief description

Extension parameter for hardware specific settings.

11.1.590 transfer_info_t

```
typedef struct{
    uint32_t  __pad0__
    uint32_t  __pad1__
    transfer_addr_mode_t  dest_addr_mode
    transfer_repeat_area_t  repeat_area
    transfer_irq_t  irq
    transfer_chain_mode_t  chain_mode
    uint32_t  __pad2__
    transfer_addr_mode_t  src_addr_mode
    transfer_size_t  size
    transfer_mode_t  mode
    void const  *volatile  p_src
    void *volatile  p_dest
    uint16_t  num_blocks
    uint16_t  length
} transfer_info_t
```

11.1.590.1 __pad0__

```
uint32_t transfer_info_t::__pad0__
```

11.1.590.2 __pad1__

```
uint32_t transfer_info_t::__pad1__
```

11.1.590.3 dest_addr_mode

```
transfer_addr_mode_t::dest_addr_mode
```

Detailed description

Select what happens to destination pointer after each transfer.

11.1.590.4 repeat_area

```
transfer_repeat_area_t::repeat_area
```

Detailed description

Select to repeat source or destination area, unused in TRANSFER_MODE_NORMAL.

11.1.590.5 irq

transfer_irq_t::irq

Detailed description

Select if interrupts should occur after each individual transfer or after the completion of all planned transfers.

11.1.590.6 chain_mode

transfer_chain_mode_t::chain_mode

Detailed description

Select when the chain transfer ends.

11.1.590.7 __pad2__

uint32_t transfer_info_t::__pad2__

11.1.590.8 src_addr_mode

transfer_addr_mode_t::src_addr_mode

Detailed description

Select what happens to source pointer after each transfer.

11.1.590.9 size

transfer_size_t::size

Detailed description

Select number of bytes to transfer at once. length.

11.1.590.10 mode

transfer_mode_t::mode

Detailed description

Select mode from transfer_mode_t.

11.1.590.11 p_src

```
void const* volatile transfer_info_t::p_src
```

Brief description

Source pointer.

11.1.590.12 p_dest

```
void* volatile transfer_info_t::p_dest
```

Brief description

Destination pointer.

11.1.590.13 num_blocks

```
volatile uint16_t transfer_info_t::num_blocks
```

Detailed description

Number of blocks to transfer when using TRANSFER_MODE_BLOCK (both DTC an DMAC) and TRANSFER_MODE_REPEAT (DMAC only), unused in other modes.

11.1.590.14 length

```
volatile uint16_t transfer_info_t::length
```

Detailed description

Length of each transfer. Range limited for TRANSFER_MODE_BLOCK and TRANSFER_MODE_REPEAT, see HAL driver for details.

11.1.591 transfer_instance_t

```
typedef struct{
    transfer_ctrl_t *  p_ctrl
    transfer_cfg_t const  *  p_cfg
    transfer_api_t const  *  p_api
} transfer_instance_t
```

11.1.591.1 p_ctrl

```
transfer_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.591.2 p_cfg

transfer_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.591.3 p_api

transfer_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.592 transfer_on_dmac_cfg_t

```
typedef struct{
    uint8_t  channel
} transfer_on_dmac_cfg_t
```

11.1.592.1 channel

uint8_t ::channel

Brief description

Channel number, does not apply to all HAL drivers.

11.1.593 transfer_properties_t

```
typedef struct{
    uint32_t  transfer_length_max
    uint16_t  transfer_length_remaining
    bool  in_progress
} transfer_properties_t
```

11.1.593.1 transfer_length_max

uint32_t transfer_properties_t::transfer_length_max

Brief description

Maximum number of transfers.

11.1.593.2 transfer_length_remaining

```
uint16_t transfer_properties_t::transfer_length_remaining
```

Brief description

Number of transfers remaining.

11.1.593.3 in_progress

```
bool transfer_properties_t::in_progress
```

Brief description

Whether or not this transfer is in progress.

11.1.594 trng_api_t

```
typedef struct{
    uint32_t(*  open)(trng_ctrl_t *const p_ctrl, trng_cfg_t const *const p_cfg)
    uint32_t(*  read)(trng_ctrl_t *const p_ctrl, uint32_t *const p_rngbuf,
uint32_t nwords)
    uint32_t(*  close)(trng_ctrl_t *const p_ctrl)
    uint32_t(*  versionGet)(ssp_version_t *const p_version)
} trng_api_t
```

11.1.595 trng_cfg_t

```
typedef struct{
    crypto_api_t const  *  p_crypto_api
    uint32_t  nattempts
} trng_cfg_t
```

11.1.595.1 p_crypto_api

```
const* trng_cfg_t::p_crypto_api
```

Brief description

pointer to crypto API

11.1.595.2 nattempts

```
uint32_t trng_cfg_t::nattempts
```

Brief description

number of retries when a continuous test failure occurs

11.1.596 trng_ctrl_t

```
typedef struct{
    uint32_t  nattempts
    crypto_ctrl_t *  p_crypto_ctrl
    crypto_api_t const  *  p_crypto_api
    uint32_t  prevbuf[TRNG_REGISTER_SIZE_WORDS]
    uint32_t  currbuf[TRNG_REGISTER_SIZE_WORDS]
} trng_ctrl_t
```

11.1.596.1 nattempts

```
uint32_t trng_ctrl_t::nattempts
```

Brief description

number of retries

11.1.596.2 p_crypto_ctrl

```
* trng_ctrl_t::p_crypto_ctrl
```

Brief description

pointer to crypto control structure

11.1.596.3 p_crypto_api

```
const* trng_ctrl_t::p_crypto_api
```

Brief description

pointer to crypto-engine API

11.1.596.4 prevbuf

```
uint32_t trng_ctrl_t::prevbuf[TRNG_REGISTER_SIZE_WORDS]
```

Brief description

previous random data

11.1.596.5 currbuf

```
uint32_t trng_ctrl_t::currbuf[TRNG_REGISTER_SIZE_WORDS]
```

Brief description

current random data

11.1.597 trng_instance_t

```
typedef struct{
    trng_ctrl_t *  p_ctrl
    trng_cfg_t const  *  p_cfg
    trng_api_t const  *  p_api
} trng_instance_t
```

11.1.597.1 p_ctrl

trng_ctrl_t::p_ctrl

Brief description

Pointer to the control structure for this instance.

11.1.597.2 p_cfg

trng_cfg_t::p_cfg

Brief description

Pointer to the configuration structure for this instance.

11.1.597.3 p_api

trng_api_t::p_api

Brief description

Pointer to the API structure for this instance.

11.1.598 uart_api_t

```
typedef struct{
    ssp_err_t(*  open)(uart_ctrl_t *const p_ctrl, uart_cfg_t const *const p_cfg)
    ssp_err_t(*  read)(uart_ctrl_t *const p_ctrl, uint8_t const *const p_dest,
uint32_t const bytes)
    ssp_err_t(*  write)(uart_ctrl_t *const p_ctrl, uint8_t const *const p_src,
uint32_t const bytes)
    ssp_err_t(*  baudSet)(uart_ctrl_t *const p_ctrl, uint32_t const baudrate)
    ssp_err_t(*  infoGet)(uart_ctrl_t *const p_ctrl, uart_info_t *const p_info)
    ssp_err_t(*  close)(uart_ctrl_t *const p_ctrl)
    ssp_err_t(*  versionGet)(ssp_version_t *p_version)
    ssp_err_t(*  communicationAbort)(uart_ctrl_t *const p_ctrl, uart_dir_t
communication_to_abort)
} uart_api_t
```

11.1.599 uart_callback_args_t

```
typedef struct{
    uint32_t  channel
    uart_event_t  event
    uint32_t  data
    void const  *  p_context
} uart_callback_args_t
```

11.1.599.1 channel

```
uint32_t uart_callback_args_t::channel
```

Brief description

Device channel number.

11.1.599.2 event

```
uart_event_t::event
```

Brief description

Event code.

11.1.599.3 data

```
uint32_t uart_callback_args_t::data
```

Detailed description

Contains the next character received for the events UART_EVENT_RX_CHAR, UART_EVENT_ERR_PARITY, UART_EVENT_ERR_FRAMING, or UART_EVENT_ERR_OVERFLOW. Otherwise unused.

11.1.599.4 p_context

```
void const* uart_callback_args_t::p_context
```

Brief description

Context provided to user during callback.

11.1.600 uart_cfg_t

```
typedef struct{
    uint8_t  channel
    uint32_t  baud_rate
    uart_data_bits_t  data_bits
    uart_parity_t  parity
```

```
    uart_stop_bits_t  stop_bits
    bool  ctsrts_en
    uint8_t  rxi_ipl
    uint8_t  txi_ipl
    uint8_t  tei_ipl
    uint8_t  eri_ipl
    transfer_instance_t const  *  p_transfer_rx
    transfer_instance_t const  *  p_transfer_tx
    void(*  p_callback)(uart_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} uart_cfg_t
```

11.1.600.1 channel

```
uint8_t uart_cfg_t::channel
```

Brief description

Select a channel corresponding to the channel number of the hardware.

11.1.600.2 baud_rate

```
uint32_t uart_cfg_t::baud_rate
```

Brief description

Baud rate, i.e. 9600, 19200, 115200.

11.1.600.3 data_bits

```
uart_data_bits_t::data_bits
```

Brief description

Data bit length (8 or 7 or 9)

11.1.600.4 parity

```
uart_parity_t::parity
```

Brief description

Parity type (none or odd or even)

11.1.600.5 stop_bits

```
uart_stop_bits_t::stop_bits
```

Brief description

Stop bit length (1 or 2)

11.1.600.6 ctsrts_en

```
bool uart_cfg_t::ctsrts_en
```

Brief description

CTS/RTS hardware flow control enable.

11.1.600.7 rxi_ipl

```
uint8_t uart_cfg_t::rxi_ipl
```

Brief description

Receive interrupt priority.

11.1.600.8 txi_ipl

```
uint8_t uart_cfg_t::txi_ipl
```

Brief description

Transmit interrupt priority.

11.1.600.9 tei_ipl

```
uint8_t uart_cfg_t::tei_ipl
```

Brief description

Transmit end interrupt priority.

11.1.600.10 eri_ipl

```
uint8_t uart_cfg_t::eri_ipl
```

Brief description

Error interrupt priority.

11.1.600.11 p_transfer_rx

```
transfer_instance_t::p_transfer_rx
```

Detailed description

Optional transfer instance used to receive multiple bytes without interrupts. Set to NULL if unused. If NULL, the number of bytes allowed in the read API is limited to one byte at a time.

11.1.600.12 p_transfer_tx

```
transfer_instance_t::p_transfer_tx
```

Detailed description

Optional transfer instance used to send multiple bytes without interrupts. Set to NULL if unused. If NULL, the number of bytes allowed in the write APIs is limited to one byte at a time.

11.1.600.13 p_callback

```
void(* uart_cfg_t::p_callback) (uart_callback_args_t *p_args)
```

Brief description

Pointer to callback function.

11.1.600.14 p_context

```
void const* uart_cfg_t::p_context
```

Brief description

User defined context passed into callback function.

11.1.600.15 p_extend

```
void const* uart_cfg_t::p_extend
```

Brief description

UART hardware dependent configuration.

11.1.601 uart_info_t

```
typedef struct{
    uint32_t  write_bytes_max
    uint32_t  read_bytes_max
} uart_info_t
```

11.1.601.1 write_bytes_max

```
uint32_t uart_info_t::write_bytes_max
```

Detailed description

Maximum bytes that can be written at this time. Only applies if p_transfer_tx is not NULL.

11.1.601.2 read_bytes_max

```
uint32_t uart_info_t::read_bytes_max
```

Detailed description

Maximum bytes that are available to read at one time. Only applies if p_transfer_rx is not NULL.

11.1.602 uart_instance_t

```
typedef struct{
    uart_ctrl_t *  p_ctrl
    uart_cfg_t const  *  p_cfg
    uart_api_t const  *  p_api
} uart_instance_t
```

11.1.602.1 p_ctrl

```
uart_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.602.2 p_cfg

```
uart_cfg_t::p_cfg
```

Brief description

Pointer to the configuration structure for this instance.

11.1.602.3 p_api

```
uart_api_t::p_api
```

Brief description

Pointer to the API structure for this instance.

11.1.603 uart_on_sci_cfg_t

```
typedef struct{
    sci_clk_src_t  clk_src
    bool  baudclk_out
```

```
    bool  rx_edge_start
    bool  noisecancel_en
    sci_uart_rx_fifo_trigger_t  rx_fifo_trigger
    void(*  p_extpin_ctrl)(uint32_t channel, uint32_t level)
    bool  bitrate_modulation
} uart_on_sci_cfg_t
```

11.1.603.1 clk_src

sci_clk_src_t::clk_src

Brief description

Use SCI_CLK_SRC_INT/EXT8X/EXT16X.

11.1.603.2 baudclk_out

bool ::baudclk_out

Brief description

Baud rate clock output enable.

11.1.603.3 rx_edge_start

bool ::rx_edge_start

Brief description

Start reception on falling edge.

11.1.603.4 noisecancel_en

bool ::noisecancel_en

Brief description

Noise cancel enable.

11.1.603.5 rx_fifo_trigger

sci_uart_rx_fifo_trigger_t::rx_fifo_trigger

Detailed description

Receive FIFO trigger level, unused if channel has no FIFO or if DTC is used.

11.1.603.6 p_extpin_ctrl

void(* ::p_extpin_ctrl) (uint32_t channel, uint32_t level)

Detailed description

Pointer to the user callback function to control external GPIO pin control used as RTS signal.

Table 2392:

Name	Direction	Description
channel	in	The UART channel used.
level	in	When level is 0, assert RTS. When level is 1, deassert RTS.

Parameter channel

`uint32_t`

Parameter level

`uint32_t`

11.1.603.7 bitrate_modulation

`bool ::bitrate_modulation`

Brief description

Bitrate Modulation Function enable or disable.

11.1.604 wdt_api_t

```
typedef struct{
    ssp_err_t(*  cfgGet)(wdt_ctrl_t *const p_ctrl, wdt_cfg_t *const p_cfg)
    ssp_err_t(*  open)(wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)
    ssp_err_t(*  refresh)(wdt_ctrl_t *const p_ctrl)
    ssp_err_t(*  statusGet)(wdt_ctrl_t *const p_ctrl, wdt_status_t *const
p_status)
    ssp_err_t(*  statusClear)(wdt_ctrl_t *const p_ctrl, const wdt_status_t
status)
    ssp_err_t(*  counterGet)(wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)
    ssp_err_t(*  timeoutGet)(wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t
*const p_timeout)
    ssp_err_t(*  versionGet)(ssp_version_t *const p_data)
} wdt_api_t
```

11.1.605 wdt_callback_args_t

```
typedef struct{
    void const  *  p_context
} wdt_callback_args_t
```

11.1.605.1 p_context

```
void const* wdt_callback_args_t::p_context
```

Brief description

Placeholder for user data. Set in open function in wdt_cfg_t.

11.1.606 wdt_cfg_t

```
typedef struct{
    wdt_start_mode_t  start_mode
    bool  autostart
    wdt_timeout_t  timeout
    wdt_clock_division_t  clock_division
    wdt_window_start_t  window_start
    wdt_window_end_t  window_end
    wdt_reset_control_t  reset_control
    wdt_stop_control_t  stop_control
    void(*  p_callback)(wdt_callback_args_t *p_args)
    void const  *  p_context
    void const  *  p_extend
} wdt_cfg_t
```

11.1.606.1 start_mode

```
wdt_start_mode_t::start_mode
```

Brief description

The expected start mode for the WDT.

11.1.606.2 autostart

```
bool wdt_cfg_t::autostart
```

Detailed description

When true the WDT is started as part of its configuration (register start mode). If false the WDT needs to be started manually by calling the refresh API.

11.1.606.3 timeout

wdt_timeout_t::timeout

Brief description

Timeout period.

11.1.606.4 clock_division

wdt_clock_division_t::clock_division

Brief description

Clock divider.

11.1.606.5 window_start

wdt_window_start_t::window_start

Brief description

Refresh permitted window start position.

11.1.606.6 window_end

wdt_window_end_t::window_end

Brief description

Refresh permitted window end position.

11.1.606.7 reset_control

wdt_reset_control_t::reset_control

Brief description

Select NMI or reset generated on underflow.

11.1.606.8 stop_control

wdt_stop_control_t::stop_control

Brief description

Select whether counter operates in sleep mode.

11.1.606.9 p_callback

```
void(* wdt_cfg_t::p_callback) (wdt_callback_args_t *p_args)
```

Brief description

Callback provided when a WDT NMI ISR occurs.

11.1.606.10 p_context

```
void const* wdt_cfg_t::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in wdt_callback_args_t.

11.1.606.11 p_extend

```
void const* wdt_cfg_t::p_extend
```

Brief description

Placeholder for user extension.

11.1.607 wdt_instance_ctrl_t

```
typedef struct{
    bool  wdt_open
    void const  *  p_context
    R_WDT_Type *  p_reg
    void(*  p_callback)(wdt_callback_args_t *p_args)
} wdt_instance_ctrl_t
```

11.1.607.1 wdt_open

```
bool ::wdt_open
```

Detailed description

Indicates whether the open() API has been successfully called.

11.1.607.2 p_context

```
void const* ::p_context
```

Detailed description

Placeholder for user data. Passed to the user callback in wdt_callback_args_t.

11.1.607.3 p_reg

```
R_WDT_Type* ::p_reg
```

Brief description

Pointer to register base address.

11.1.607.4 p_callback

```
void(* ::p_callback) ( *p_args)
```

Brief description

Callback provided when a WDT NMI ISR occurs.

11.1.608 wdt_instance_t

```
typedef struct{
    wdt_ctrl_t *  p_ctrl
    wdt_cfg_t const  *  p_cfg
    wdt_api_t const  *  p_api
} wdt_instance_t
```

11.1.608.1 p_ctrl

```
wdt_ctrl_t::p_ctrl
```

Brief description

Pointer to the control structure for this instance.

11.1.608.2 p_cfg

```
wdt_cfg_t::p_cfg
```

Brief description

Pointer to the configuration structure for this instance.

11.1.608.3 p_api

```
wdt_api_t::p_api
```

Brief description

Pointer to the API structure for this instance.

11.1.609 wdt_timeout_values_t

```
typedef struct{
    uint32_t  clock_frequency_hz
```

```
    uint32_t  timeout_clocks
} wdt_timeout_values_t
```

11.1.609.1 clock_frequency_hz

`uint32_t wdt_timeout_values_t::clock_frequency_hz`

Brief description

Frequency of watchdog clock after divider.

11.1.609.2 timeout_clocks

`uint32_t wdt_timeout_values_t::timeout_clocks`

Brief description

Timeout period in units of watchdog clock ticks.

11.2 Interface Functions

11.2.1 Interface: adc_api_t

Description: ADC Interface

Function name	Definition
.open	ssp_err_t(* adc_api_t::open) (adc_ctrl_t *const p_ctrl, adc_cfg_t const *const p_cfg)
.scanCfg	ssp_err_t(* adc_api_t::scanCfg) (adc_ctrl_t *const p_ctrl, adc_channel_cfg_t const *const p_channel_cfg)
.scanStart	ssp_err_t(* adc_api_t::scanStart) (adc_ctrl_t *const p_ctrl)
.scanStop	ssp_err_t(* adc_api_t::scanStop) (adc_ctrl_t *const p_ctrl)
.scanStatusGet	ssp_err_t(* adc_api_t::scanStatusGet) (adc_ctrl_t *const p_ctrl)
.read	ssp_err_t(* adc_api_t::read) (adc_ctrl_t *const p_ctrl, adc_register_t const reg_id, uint16_t *const p_data)
.read32	ssp_err_t(* adc_api_t::read32) (adc_ctrl_t *const p_ctrl, adc_register_t const reg_id, uint32_t *const p_data)

Function name	Definition
.sampleStateCountSet	ssp_err_t(* adc_api_t::sampleStateCountSet) (adc_ctrl_t *const p_ctrl, adc_sample_state_t *p_sample)
.calibrate	ssp_err_t(* adc_api_t::calibrate) (adc_ctrl_t *const p_ctrl, void *const p_extend)
.offsetSet	ssp_err_t(* adc_api_t::offsetSet) (adc_ctrl_t *const p_ctrl, adc_register_t const reg_id, int32_t const offset)
.close	ssp_err_t(* adc_api_t::close) (adc_ctrl_t *const p_ctrl)
.infoGet	ssp_err_t(* adc_api_t::infoGet) (adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info)
.versionGet	ssp_err_t(* adc_api_t::versionGet) (ssp_version_t *const p_version)

11.2.2 Interface: aes_api_t

Description: AES Interface

Function name	Definition
.open	uint32_t(* aes_api_t::open) (aes_ctrl_t *const p_ctrl, aes_cfg_t const *const p_cfg)
.createKey	uint32_t(* aes_api_t::createKey) (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_key)
.encrypt	uint32_t(* aes_api_t::encrypt) (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source, uint32_t *p_dest)
.addAdditionalAuthenticationData	uint32_t(* aes_api_t::addAdditionalAuthenticationData) (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_words, uint32_t *p_source)
.encryptFinal	uint32_t(* aes_api_t::encryptFinal) (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t input_num_words, uint32_t *p_source, uint32_t output_num_words, uint32_t *p_dest)
.decrypt	uint32_t(* aes_api_t::decrypt) (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t imaxcnt, uint32_t *p_source, uint32_t *p_dest)

Function name	Definition
.setGcmTag	uint32_t(* aes_api_t::setGcmTag) (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_source)
.getGcmTag	uint32_t(* aes_api_t::getGcmTag) (aes_ctrl_t *const p_ctrl, uint32_t num_words, uint32_t *p_dest)
.close	uint32_t(* aes_api_t::close) (aes_ctrl_t *const p_ctrl)
.zeroPaddingEncrypt	uint32_t(* aes_api_t::zeroPaddingEncrypt) (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest)
.zeroPaddingDecrypt	uint32_t(* aes_api_t::zeroPaddingDecrypt) (aes_ctrl_t *const p_ctrl, const uint32_t *p_key, uint32_t *p_iv, uint32_t num_bytes, uint32_t *p_source, uint32_t *p_dest)
.versionGet	uint32_t(* aes_api_t::versionGet) (ssp_version_t *const p_version)

11.2.3 Interface: arc4_api_t

Description: ARC4 Interface

Function name	Definition
.open	uint32_t(* arc4_api_t::open) (arc4_ctrl_t *const p_ctrl, arc4_cfg_t const *const p_cfg)
.keySet	uint32_t(* arc4_api_t::keySet) (arc4_ctrl_t *const p_ctrl, uint32_t length, uint8_t const *p_key)
.arc4Process	uint32_t(* arc4_api_t::arc4Process) (arc4_ctrl_t *const p_ctrl, uint32_t num_bytes, uint8_t *p_source, uint8_t *p_dest)
.close	uint32_t(* arc4_api_t::close) (arc4_ctrl_t *const p_ctrl)
.versionGet	uint32_t(* arc4_api_t::versionGet) (ssp_version_t *const p_version)

11.2.4 Interface: cac_api_t

Description: CAC Interface

Function name	Definition
.open	ssp_err_t(* cac_api_t::open) (cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)
.read	ssp_err_t(* cac_api_t::read) (cac_ctrl_t *const p_ctrl, uint8_t *const p_status, uint16_t *const p_counter)
.close	ssp_err_t(* cac_api_t::close) (cac_ctrl_t *const p_ctrl)
.stopMeasurement	ssp_err_t(* cac_api_t::stopMeasurement) (cac_ctrl_t *const p_ctrl)
.startMeasurement	ssp_err_t(* cac_api_t::startMeasurement) (cac_ctrl_t *const p_ctrl)
.reset	ssp_err_t(* cac_api_t::reset) (cac_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* cac_api_t::versionGet) (ssp_version_t *p_version)

11.2.5 Interface: can_api_t

Description: CAN Interface

Function name	Definition
.open	ssp_err_t(* can_api_t::open) (can_ctrl_t *const p_ctrl, can_cfg_t const *const p_cfg)
.read	ssp_err_t(* can_api_t::read) (can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t *const p_frame)
.write	ssp_err_t(* can_api_t::write) (can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t *const p_frame)
.close	ssp_err_t(* can_api_t::close) (can_ctrl_t *const p_ctrl)
.control	ssp_err_t(* can_api_t::control) (can_ctrl_t *const p_ctrl, can_command_t const command, void *p_data)

Function name	Definition
.infoGet	ssp_err_t(* can_api_t::infoGet) (can_ctrl_t *const p_ctrl, can_info_t *const p_info)
.versionGet	ssp_err_t(* can_api_t::versionGet) (ssp_version_t *const p_version)

11.2.6 Interface: cgc_api_t

Description: CGC Interface

Function name	Definition
.init	ssp_err_t(* cgc_api_t::init) (void)
.clocksCfg	ssp_err_t(* cgc_api_t::clocksCfg) (cgc_clocks_cfg_t const *const p_clock_cfg)
.clockStart	ssp_err_t(* cgc_api_t::clockStart) (cgc_clock_t clock_source, cgc_clock_cfg_t *p_clock_cfg)
.clockStop	ssp_err_t(* cgc_api_t::clockStop) (cgc_clock_t clock_source)
.systemClockSet	ssp_err_t(* cgc_api_t::systemClockSet) (cgc_clock_t clock_source, cgc_system_clock_cfg_t const *const p_clock_cfg)
.systemClockGet	ssp_err_t(* cgc_api_t::systemClockGet) (cgc_clock_t *p_clock_source, cgc_system_clock_cfg_t *p_set_clock_cfg)
.systemClockFreqGet	ssp_err_t(* cgc_api_t::systemClockFreqGet) (cgc_system_clocks_t clock, uint32_t *p_freq_hz)
.clockCheck	ssp_err_t(* cgc_api_t::clockCheck) (cgc_clock_t clock_source)
.oscStopDetect	ssp_err_t(* cgc_api_t::oscStopDetect) (void(*p_callback)(cgc_callback_args_t *p_args), bool enable)
.oscStopStatusClear	ssp_err_t(* cgc_api_t::oscStopStatusClear) (void)
.busClockOutCfg	ssp_err_t(* cgc_api_t::busClockOutCfg) (cgc_bclockout_dividers_t divider)

Function name	Definition
.busClockOutEnable	ssp_err_t(* cgc_api_t::busClockOutEnable) (void)
.busClockOutDisable	ssp_err_t(* cgc_api_t::busClockOutDisable) (void)
.clockOutCfg	ssp_err_t(* cgc_api_t::clockOutCfg) (cgc_clock_t clock, cgc_clockout_dividers_t divider)
.clockOutEnable	ssp_err_t(* cgc_api_t::clockOutEnable) (void)
.clockOutDisable	ssp_err_t(* cgc_api_t::clockOutDisable) (void)
.lcdClockCfg	ssp_err_t(* cgc_api_t::lcdClockCfg) (cgc_clock_t clock)
.lcdClockEnable	ssp_err_t(* cgc_api_t::lcdClockEnable) (void)
.lcdClockDisable	ssp_err_t(* cgc_api_t::lcdClockDisable) (void)
.sdadcClockCfg	ssp_err_t(* cgc_api_t::sdadcClockCfg) (cgc_clock_t clock)
.sdadcClockEnable	ssp_err_t(* cgc_api_t::sdadcClockEnable) (void)
.sdadcClockDisable	ssp_err_t(* cgc_api_t::sdadcClockDisable) (void)
.sdramClockOutEnable	ssp_err_t(* cgc_api_t::sdramClockOutEnable) (void)
.sdramClockOutDisable	ssp_err_t(* cgc_api_t::sdramClockOutDisable) (void)
.usbClockCfg	ssp_err_t(* cgc_api_t::usbClockCfg) (cgc_usb_clock_div_t divider)
.systickUpdate	ssp_err_t(* cgc_api_t::systickUpdate) (uint32_t period_count, cgc_systick_period_units_t units)
.versionGet	ssp_err_t(* cgc_api_t::versionGet) (ssp_version_t *p_version)

11.2.7 Interface: comparator_api_t

Description: COMPARATOR Interface

Function name	Definition
.open	ssp_err_t(* comparator_api_t::open) (comparator_ctrl_t *const p_ctrl, comparator_cfg_t const *const p_cfg)

Function name	Definition
.outputEnable	ssp_err_t(* comparator_api_t::outputEnable) (comparator_ctrl_t *const p_ctrl)
.infoGet	ssp_err_t(* comparator_api_t::infoGet) (comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info)
.statusGet	ssp_err_t(* comparator_api_t::statusGet) (comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status)
.close	ssp_err_t(* comparator_api_t::close) (comparator_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* comparator_api_t::versionGet) (ssp_version_t *const p_version)

11.2.8 Interface: crc_api_t

Description: CRC Interface

Function name	Definition
.open	ssp_err_t(* crc_api_t::open) (crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)
.close	ssp_err_t(* crc_api_t::close) (crc_ctrl_t *const p_ctrl)
.crcResultGet	ssp_err_t(* crc_api_t::crcResultGet) (crc_ctrl_t *const p_ctrl, uint32_t *crc_result)
.snoopEnable	ssp_err_t(* crc_api_t::snoopEnable) (crc_ctrl_t *const p_ctrl, uint32_t crc_seed)
.snoopDisable	ssp_err_t(* crc_api_t::snoopDisable) (crc_ctrl_t *const p_ctrl)
.snoopCfg	ssp_err_t(* crc_api_t::snoopCfg) (crc_ctrl_t *const p_ctrl, crc_snoop_cfg_t *const p_snoop_cfg)
.calculate	ssp_err_t(* crc_api_t::calculate) (crc_ctrl_t *const p_ctrl, void *p_input_buffer, uint32_t num_bytes, uint32_t crc_seed, uint32_t *p_crc_result)
.versionGet	ssp_err_t(* crc_api_t::versionGet) (ssp_version_t *version)

11.2.9 Interface: ctsu_api_t

Description: CTSU Interface

Function name	Definition
.open	ssp_err_t(* ctsu_api_t::open) (ctsu_ctrl_t *p_ctrl, ctsu_cfg_t *p_cfg)
.close	ssp_err_t(* ctsu_api_t::close) (ctsu_ctrl_t *p_ctrl, ctsu_close_option_t opts)
.scan	ssp_err_t(* ctsu_api_t::scan) (ctsu_ctrl_t *p_ctrl)
.update	ssp_err_t(* ctsu_api_t::update) (ctsu_ctrl_t *p_ctrl)
.read	ssp_err_t(* ctsu_api_t::read) (ctsu_ctrl_t *p_ctrl, void *p_dest, ctsu_read_t opts, const ctsu_channel_pair_t *channels, const uint16_t count)
.versionGet	ssp_err_t(* ctsu_api_t::versionGet) (ssp_version_t *const p_version)

11.2.10 Interface: dac_api_t

Description: DAC Interface

Function name	Definition
.open	ssp_err_t(* dac_api_t::open) (dac_ctrl_t *p_ctrl, dac_cfg_t const *const p_cfg)
.close	ssp_err_t(* dac_api_t::close) (dac_ctrl_t *p_ctrl)
.write	ssp_err_t(* dac_api_t::write) (dac_ctrl_t *p_ctrl, dac_size_t value)
.start	ssp_err_t(* dac_api_t::start) (dac_ctrl_t *p_ctrl)
.stop	ssp_err_t(* dac_api_t::stop) (dac_ctrl_t *p_ctrl)
.versionGet	ssp_err_t(* dac_api_t::versionGet) (ssp_version_t *p_version)
.infoGet	ssp_err_t(* dac_api_t::infoGet) (dac_info_t *const p_info)

11.2.11 Interface: display_api_t

Description: Display Interface

Function name	Definition
.open	ssp_err_t(* display_api_t::open) (display_ctrl_t *const p_ctrl, display_cfg_t const *const p_cfg)
.close	ssp_err_t(* display_api_t::close) (display_ctrl_t *const p_ctrl)
.start	ssp_err_t(* display_api_t::start) (display_ctrl_t *const p_ctrl)
.stop	ssp_err_t(* display_api_t::stop) (display_ctrl_t *const p_ctrl)
.layerChange	ssp_err_t(* display_api_t::layerChange) (display_ctrl_t const *const p_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t frame)
.correction	ssp_err_t(* display_api_t::correction) (display_ctrl_t const *const p_ctrl, display_correction_t const *const p_param)
.clut	ssp_err_t(* display_api_t::clut) (display_ctrl_t const *const p_ctrl, display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t frame)
.statusGet	ssp_err_t(* display_api_t::statusGet) (display_ctrl_t const *const p_ctrl, display_status_t *const p_status)
.versionGet	ssp_err_t(* display_api_t::versionGet) (ssp_version_t *p_version)

11.2.12 Interface: doc_api_t

Description: DOC Interface

Function name	Definition
.open	ssp_err_t(* doc_api_t::open) (doc_ctrl_t *const p_ctrl, doc_cfg_t const *const p_cfg)
.close	ssp_err_t(* doc_api_t::close) (doc_ctrl_t *const p_ctrl)

Function name	Definition
.statusGet	ssp_err_t(* doc_api_t::statusGet) (doc_ctrl_t *const p_ctrl, doc_status_t *p_status)
.statusClear	ssp_err_t(* doc_api_t::statusClear) (doc_ctrl_t *const p_ctrl)
.write	ssp_err_t(* doc_api_t::write) (doc_ctrl_t *const p_ctrl, doc_data_t *const p_data)
.inputRegisterWrite	ssp_err_t(* doc_api_t::inputRegisterWrite) (doc_ctrl_t *const p_ctrl, doc_size_t data)
.versionGet	ssp_err_t(* doc_api_t::versionGet) (ssp_version_t *const p_version)

11.2.13 Interface: elc_api_t

Description: ELC Interface

Function name	Definition
.init	ssp_err_t(* elc_api_t::init) (elc_cfg_t const *const p_cfg)
.softwareEventGenerate	ssp_err_t(* elc_api_t::softwareEventGenerate) (elc_software_event_t event_num)
.linkSet	ssp_err_t(* elc_api_t::linkSet) (elc_peripheral_t peripheral, elc_event_t signal)
.linkBreak	ssp_err_t(* elc_api_t::linkBreak) (elc_peripheral_t peripheral)
.enable	ssp_err_t(* elc_api_t::enable) (void)
.disable	ssp_err_t(* elc_api_t::disable) (void)
.versionGet	ssp_err_t(* elc_api_t::versionGet) (ssp_version_t *const p_version)

11.2.14 Interface: external_irq_api_t

Description: External IRQ Interface

Function name	Definition
.open	ssp_err_t(* external_irq_api_t::open) (external_irq_ctrl_t *const p_ctrl, external_irq_cfg_t const *const p_cfg)
.enable	ssp_err_t(* external_irq_api_t::enable) (external_irq_ctrl_t *const p_ctrl)
.disable	ssp_err_t(* external_irq_api_t::disable) (external_irq_ctrl_t *const p_ctrl)
.triggerSet	ssp_err_t(* external_irq_api_t::triggerSet) (external_irq_ctrl_t *const p_ctrl, external_irq_trigger_t const trigger)
.filterEnable	ssp_err_t(* external_irq_api_t::filterEnable) (external_irq_ctrl_t *const p_ctrl)
.filterDisable	ssp_err_t(* external_irq_api_t::filterDisable) (external_irq_ctrl_t *const p_ctrl)
.close	ssp_err_t(* external_irq_api_t::close) (external_irq_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* external_irq_api_t::versionGet) (ssp_version_t *const p_version)

11.2.15 Interface: flash_api_t

Description: Flash Interface

Function name	Definition
.open	ssp_err_t(* flash_api_t::open) (flash_ctrl_t *const p_ctrl, flash_cfg_t const *const p_cfg)
.write	ssp_err_t(* flash_api_t::write) (flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes)
.read	ssp_err_t(* flash_api_t::read) (flash_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const flash_address, uint32_t const num_bytes)
.erase	ssp_err_t(* flash_api_t::erase) (flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_blocks)

Function name	Definition
.blankCheck	ssp_err_t(* flash_api_t::blankCheck) (flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_bytes, flash_result_t *const p_blank_check_result)
.infoGet	ssp_err_t(* flash_api_t::infoGet) (flash_ctrl_t *const p_ctrl, flash_info_t *const p_info)
.close	ssp_err_t(* flash_api_t::close) (flash_ctrl_t *const p_ctrl)
.statusGet	ssp_err_t(* flash_api_t::statusGet) (flash_ctrl_t *const p_ctrl)
.accessWindowSet	ssp_err_t(* flash_api_t::accessWindowSet) (flash_ctrl_t *const p_ctrl, uint32_t const start_addr, uint32_t const end_addr)
.accessWindowClear	ssp_err_t(* flash_api_t::accessWindowClear) (flash_ctrl_t *const p_ctrl)
.reset	ssp_err_t(* flash_api_t::reset) (flash_ctrl_t *const p_ctrl)
.updateFlashClockFreq	ssp_err_t(* flash_api_t::updateFlashClockFreq) (flash_ctrl_t *const p_ctrl)
.startupAreaSelect	ssp_err_t(* flash_api_t::startupAreaSelect) (flash_ctrl_t *const p_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary)
.versionGet	ssp_err_t(* flash_api_t::versionGet) (ssp_version_t *p_version)

11.2.16 Interface: fmi_api_t

Description: FMI Interface

Function name	Definition
.init	ssp_err_t(* fmi_api_t::init) (void)
.productInfoGet	ssp_err_t(* fmi_api_t::productInfoGet) (fmi_product_info_t **pp_product_info)
.uniqueIdGet	ssp_err_t(* fmi_api_t::uniqueIdGet) (fmi_unique_id_t *p_unique_id)

Function name	Definition
.productFeatureGet	ssp_err_t(* fmi_api_t::productFeatureGet) (ssp_feature_t const *const p_feature, fmi_feature_info_t *const p_info)
.eventInfoGet	ssp_err_t(* fmi_api_t::eventInfoGet) (ssp_feature_t const *const p_feature, ssp_signal_t signal, fmi_event_info_t *const p_info)
.versionGet	ssp_err_t(* fmi_api_t::versionGet) (ssp_version_t *const p_version)

11.2.17 Interface: hash_api_t

Description: HASH Algorithm Interface

Function name	Definition
.open	uint32_t(* hash_api_t::open) (hash_ctrl_t *const p_ctrl, hash_cfg_t const *const p_cfg)
.updateHash	uint32_t(* hash_api_t::updateHash) (const uint32_t *p_source, uint32_t num_words, uint32_t *p_dest)
.hashUpdate	uint32_t(* hash_api_t::hashUpdate) (hash_ctrl_t *const p_ctrl, const uint32_t *p_source, uint32_t num_words, uint32_t *p_dest)
.close	uint32_t(* hash_api_t::close) (hash_ctrl_t *const p_ctrl)
.versionGet	uint32_t(* hash_api_t::versionGet) (ssp_version_t *const p_version)

11.2.18 Interface: i2s_api_t

Description: I2S Interface

Function name	Definition
.open	ssp_err_t(* i2s_api_t::open) (i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg)
.stop	ssp_err_t(* i2s_api_t::stop) (i2s_ctrl_t *const p_ctrl, i2s_dir_t const dir)

Function name	Definition
.mute	ssp_err_t(* i2s_api_t::mute) (i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)
.write	ssp_err_t(* i2s_api_t::write) (i2s_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint16_t const bytes)
.read	ssp_err_t(* i2s_api_t::read) (i2s_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint16_t const bytes)
.writeRead	ssp_err_t(* i2s_api_t::writeRead) (i2s_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint16_t const bytes)
.infoGet	ssp_err_t(* i2s_api_t::infoGet) (i2s_ctrl_t *const p_ctrl, i2s_info_t *const p_info)
.close	ssp_err_t(* i2s_api_t::close) (i2s_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* i2s_api_t::versionGet) (ssp_version_t *const p_version)

11.2.19 Interface: input_capture_api_t

Description: Input Capture Interface

Function name	Definition
.open	ssp_err_t(* input_capture_api_t::open) (input_capture_ctrl_t *const p_ctrl, input_capture_cfg_t const *const p_cfg)
.disable	ssp_err_t(* input_capture_api_t::disable) (input_capture_ctrl_t const *const p_ctrl)
.enable	ssp_err_t(* input_capture_api_t::enable) (input_capture_ctrl_t const *const p_ctrl)
.infoGet	ssp_err_t(* input_capture_api_t::infoGet) (input_capture_ctrl_t const *const p_ctrl, input_capture_info_t *const p_info)
.lastCaptureGet	ssp_err_t(* input_capture_api_t::lastCaptureGet) (input_capture_ctrl_t const *const p_ctrl, input_capture_capture_t *const p_counter)

Function name	Definition
.close	ssp_err_t(* input_capture_api_t::close) (input_capture_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* input_capture_api_t::versionGet) (ssp_version_t *const p_version)

11.2.20 Interface: ioport_api_t

Description: I/O Port Interface

Function name	Definition
.init	ssp_err_t(* ioport_api_t::init) (const ioport_cfg_t *p_cfg)
.pinsCfg	ssp_err_t(* ioport_api_t::pinsCfg) (const ioport_cfg_t *p_cfg)
.pinCfg	ssp_err_t(* ioport_api_t::pinCfg) (ioport_port_pin_t pin, uint32_t cfg)
.pinDirectionSet	ssp_err_t(* ioport_api_t::pinDirectionSet) (ioport_port_pin_t pin, ioport_direction_t direction)
.pinEventInputRead	ssp_err_t(* ioport_api_t::pinEventInputRead) (ioport_port_pin_t pin, ioport_level_t *p_pin_event)
.pinEventOutputWrite	ssp_err_t(* ioport_api_t::pinEventOutputWrite) (ioport_port_pin_t pin, ioport_level_t pin_value)
.pinEthernetModeCfg	ssp_err_t(* ioport_api_t::pinEthernetModeCfg) (ioport_ethernet_channel_t channel, ioport_ethernet_mode_t mode)
.pinRead	ssp_err_t(* ioport_api_t::pinRead) (ioport_port_pin_t pin, ioport_level_t *p_pin_value)
.pinWrite	ssp_err_t(* ioport_api_t::pinWrite) (ioport_port_pin_t pin, ioport_level_t level)
.portDirectionSet	ssp_err_t(* ioport_api_t::portDirectionSet) (ioport_port_t port, ioport_size_t direction_values, ioport_size_t mask)
.portEventInputRead	ssp_err_t(* ioport_api_t::portEventInputRead) (ioport_port_t port, ioport_size_t *p_event_data)

Function name	Definition
.portEventOutputWrite	ssp_err_t(* ioport_api_t::portEventOutputWrite) (ioport_port_t port, ioport_size_t event_data, ioport_size_t mask_value)
.portRead	ssp_err_t(* ioport_api_t::portRead) (ioport_port_t port, ioport_size_t *p_port_value)
.portWrite	ssp_err_t(* ioport_api_t::portWrite) (ioport_port_t port, ioport_size_t value, ioport_size_t mask)
.versionGet	ssp_err_t(* ioport_api_t::versionGet) (ssp_version_t *p_data)

11.2.21 Interface: jpeg_decode_api_t

Description: JPEG Decode Interface

Function name	Definition
.open	ssp_err_t(* jpeg_decode_api_t::open) (jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_cfg_t const *const p_cfg)
.outputBufferSet	ssp_err_t(* jpeg_decode_api_t::outputBufferSet) (jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)
.horizontalStrideSet	ssp_err_t(* jpeg_decode_api_t::horizontalStrideSet) (jpeg_decode_ctrl_t *const p_ctrl, uint32_t horizontal_stride)
.imageSubsampleSet	ssp_err_t(* jpeg_decode_api_t::imageSubsampleSet) (jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample)
.inputBufferSet	ssp_err_t(* jpeg_decode_api_t::inputBufferSet) (jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)
.linesDecodedGet	ssp_err_t(* jpeg_decode_api_t::linesDecodedGet) (jpeg_decode_ctrl_t *const p_ctrl, uint32_t *const p_lines)
.imageSizeGet	ssp_err_t(* jpeg_decode_api_t::imageSizeGet) (jpeg_decode_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)

Function name	Definition
.statusGet	ssp_err_t(* jpeg_decode_api_t::statusGet) (jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t *const p_status)
.close	ssp_err_t(* jpeg_decode_api_t::close) (jpeg_decode_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* jpeg_decode_api_t::versionGet) (ssp_version_t *p_version)
.pixelFormatGet	ssp_err_t(* jpeg_decode_api_t::pixelFormatGet) (jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_color_space_t *const p_color_space)

11.2.22 Interface: jpeg_encode_api_t

Description: JPEG Encode Interface

Function name	Definition
.open	ssp_err_t(* jpeg_encode_api_t::open) (jpeg_encode_ctrl_t *const p_ctrl, jpeg_encode_cfg_t const *const p_cfg)
.imageParameterSet	ssp_err_t(* jpeg_encode_api_t::imageParameterSet) (jpeg_encode_ctrl_t *const p_ctrl, jpeg_encode_raw_image_parameters *p_raw_image_parameters)
.outputBufferSet	ssp_err_t(* jpeg_encode_api_t::outputBufferSet) (jpeg_encode_ctrl_t *const p_ctrl, void *p_buffer)
.inputBufferSet	ssp_err_t(* jpeg_encode_api_t::inputBufferSet) (jpeg_encode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)
.statusGet	ssp_err_t(* jpeg_encode_api_t::statusGet) (jpeg_encode_ctrl_t *const p_ctrl, volatile jpeg_encode_status_t *p_status)
.close	ssp_err_t(* jpeg_encode_api_t::close) (jpeg_encode_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* jpeg_encode_api_t::versionGet) (ssp_version_t *p_version)

11.2.23 Interface: keymatrix_api_t

Description: Key Matrix Interface

Function name	Definition
.open	ssp_err_t(* keymatrix_api_t::open) (keymatrix_ctrl_t *const p_ctrl, keymatrix_cfg_t const *const p_cfg)
.enable	ssp_err_t(* keymatrix_api_t::enable) (keymatrix_ctrl_t *const p_ctrl)
.disable	ssp_err_t(* keymatrix_api_t::disable) (keymatrix_ctrl_t *const p_ctrl)
.triggerSet	ssp_err_t(* keymatrix_api_t::triggerSet) (keymatrix_ctrl_t *const p_ctrl, keymatrix_trigger_t const trigger)
.close	ssp_err_t(* keymatrix_api_t::close) (keymatrix_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* keymatrix_api_t::versionGet) (ssp_version_t *const p_version)

11.2.24 Interface: lpm_api_t

Description: Low Power Modes Interface

Function name	Definition
.init	ssp_err_t(* lpm_api_t::init) (lpm_cfg_t const *const p_cfg)
.mstpcrSet	ssp_err_t(* lpm_api_t::mstpcrSet) (uint32_t mstpcra_value, uint32_t mstpcrb_value, uint32_t mstpcrc_value, uint32_t mstpcrd_value)
.mstpcrGet	ssp_err_t(* lpm_api_t::mstpcrGet) (uint32_t *mstpcra_value, uint32_t *mstpcrb_value, uint32_t *mstpcrc_value, uint32_t *mstpcrd_value)
.moduleStop	ssp_err_t(* lpm_api_t::moduleStop) (lpm_mstp_t module)
.moduleStart	ssp_err_t(* lpm_api_t::moduleStart) (lpm_mstp_t module)

Function name	Definition
.operatingPowerModeSet	ssp_err_t(* lpm_api_t::operatingPowerModeSet) (lpm_operating_power_t power_mode, lpm_subosc_t subosc)
.snoozeEnable	ssp_err_t(* lpm_api_t::snoozeEnable) (lpm_snooze_rxd0_t rdx0_mode, lpm_snooze_dtc_t dtc_mode, lpm_snooze_request_t requests, lpm_snooze_end_t triggers)
.snoozeDisable	ssp_err_t(* lpm_api_t::snoozeDisable) (void)
.lowPowerCfg	ssp_err_t(* lpm_api_t::lowPowerCfg) (lpm_low_power_mode_t power_mode, lpm_output_port_enable_t output_port_enable, lpm_power_supply_t power_supply, lpm_io_port_t io_port_state)
.wupenSet	ssp_err_t(* lpm_api_t::wupenSet) (uint32_t wupen_value)
.wupenGet	ssp_err_t(* lpm_api_t::wupenGet) (uint32_t *wupen_value)
.deepStandbyCancelRequestEnable	ssp_err_t(* lpm_api_t::deepStandbyCancelRequestEnable) (lpm_deep_standby_t pin_signal, lpm_cancel_request_edge_t rising_falling)
.deepStandbyCancelRequestDisable	ssp_err_t(* lpm_api_t::deepStandbyCancelRequestDisable) (lpm_deep_standby_t pin_signal)
.lowPowerModeEnter	ssp_err_t(* lpm_api_t::lowPowerModeEnter) (void)
.versionGet	ssp_err_t(* lpm_api_t::versionGet) (ssp_version_t *const p_version)

11.2.25 Interface: lpmv2_api_t

Description: Low Power Modes V2 Interface

Function name	Definition
.init	ssp_err_t(* lpmv2_api_t::init) (void)
.lowPowerCfg	ssp_err_t(* lpmv2_api_t::lowPowerCfg) (lpmv2_cfg_t const *const p_cfg)

Function name	Definition
.lowPowerModeEnter	ssp_err_t(* lpmv2_api_t::lowPowerModeEnter) (void)
.versionGet	ssp_err_t(* lpmv2_api_t::versionGet) (ssp_version_t *const p_version)

11.2.26 Interface: lvd_api_t

Description: Low Voltage Detection Interface

Function name	Definition
.open	ssp_err_t(* lvd_api_t::open) (lvd_ctrl_t *const p_ctrl, lvd_cfg_t const *const p_cfg)
.statusGet	ssp_err_t(* lvd_api_t::statusGet) (lvd_ctrl_t *const p_ctrl, lvd_status_t *p_lvd_status)
.statusClear	ssp_err_t(* lvd_api_t::statusClear) (lvd_ctrl_t *const p_ctrl)
.close	ssp_err_t(* lvd_api_t::close) (lvd_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* lvd_api_t::versionGet) (ssp_version_t *const p_version)

11.2.27 Interface: opamp_api_t

Description: OPAMP Interface

Function name	Definition
.open	ssp_err_t(* opamp_api_t::open) (opamp_ctrl_t *const p_ctrl, opamp_cfg_t const *const p_cfg)
.start	ssp_err_t(* opamp_api_t::start) (opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask)
.stop	ssp_err_t(* opamp_api_t::stop) (opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask)

Function name	Definition
.trim	ssp_err_t(* opamp_api_t::trim) (opamp_ctrl_t *const p_ctrl, opamp_trim_cmd_t const cmd, opamp_trim_args_t const *const p_args)
.infoGet	ssp_err_t(* opamp_api_t::infoGet) (opamp_ctrl_t *const p_ctrl, opamp_info_t *const p_info)
.statusGet	ssp_err_t(* opamp_api_t::statusGet) (opamp_ctrl_t *const p_ctrl, opamp_status_t *const p_status)
.close	ssp_err_t(* opamp_api_t::close) (opamp_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* opamp_api_t::versionGet) (ssp_version_t *const p_version)

11.2.28 Interface: pdc_api_t

Description: PDC Interface

Function name	Definition
.open	ssp_err_t(* pdc_api_t::open) (pdc_ctrl_t *const p_ctrl, pdc_cfg_t const *const p_cfg)
.close	ssp_err_t(* pdc_api_t::close) (pdc_ctrl_t *const p_ctrl)
.captureStart	ssp_err_t(* pdc_api_t::captureStart) (pdc_ctrl_t *const p_ctrl, uint8_t *const p_buffer)
.stateGet	ssp_err_t(* pdc_api_t::stateGet) (pdc_ctrl_t *const p_ctrl, pdc_state_t *p_state)
.versionGet	ssp_err_t(* pdc_api_t::versionGet) (ssp_version_t *const p_data)

11.2.29 Interface: qspi_api_t

Description: Quad SPI Flash Interface

Function name	Definition
.open	ssp_err_t(* qspi_api_t::open) (qspi_ctrl_t *p_ctrl, qspi_cfg_t const *const p_cfg)
.close	ssp_err_t(* qspi_api_t::close) (qspi_ctrl_t *p_ctrl)
.read	ssp_err_t(* qspi_api_t::read) (qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint8_t *p_memory_address, uint32_t byte_count)
.pageProgram	ssp_err_t(* qspi_api_t::pageProgram) (qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint8_t *p_memory_address, uint32_t byte_count)
.erase	ssp_err_t(* qspi_api_t::erase) (qspi_ctrl_t *p_ctrl, uint8_t *p_device_address, uint32_t byte_count)
.infoGet	ssp_err_t(* qspi_api_t::infoGet) (qspi_ctrl_t *const p_ctrl, qspi_info_t *const p_info)
.sectorErase	ssp_err_t(* qspi_api_t::sectorErase) (qspi_ctrl_t *p_ctrl, uint8_t *p_device_address)
.statusGet	ssp_err_t(* qspi_api_t::statusGet) (qspi_ctrl_t *p_ctrl, bool *p_write_in_progress)
.bankSelect	ssp_err_t(* qspi_api_t::bankSelect) (uint32_t bank)
.versionGet	ssp_err_t(* qspi_api_t::versionGet) (ssp_version_t *const p_version)

11.2.30 Interface: rtc_api_t

Description: RTC Interface

Function name	Definition
.open	ssp_err_t(* rtc_api_t::open) (rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg)
.close	ssp_err_t(* rtc_api_t::close) (rtc_ctrl_t *const p_ctrl)
.configure	ssp_err_t(* rtc_api_t::configure) (rtc_ctrl_t *const p_ctrl, void *const p_extend)

Function name	Definition
.calendarTimeSet	ssp_err_t(* rtc_api_t::calendarTimeSet) (rtc_ctrl_t *const p_ctrl, rtc_time_t *p_time, bool clock_start)
.calendarTimeGet	ssp_err_t(* rtc_api_t::calendarTimeGet) (rtc_ctrl_t *const p_ctrl, rtc_time_t *p_time)
.calendarAlarmSet	ssp_err_t(* rtc_api_t::calendarAlarmSet) (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *p_alarm, bool irq_enable_flag)
.calendarAlarmGet	ssp_err_t(* rtc_api_t::calendarAlarmGet) (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *p_alarm)
.calendarCounterStart	ssp_err_t(* rtc_api_t::calendarCounterStart) (rtc_ctrl_t *const p_ctrl)
.calendarCounterStop	ssp_err_t(* rtc_api_t::calendarCounterStop) (rtc_ctrl_t *const p_ctrl)
.irqEnable	ssp_err_t(* rtc_api_t::irqEnable) (rtc_ctrl_t *const p_ctrl, rtc_event_t irq)
.irqDisable	ssp_err_t(* rtc_api_t::irqDisable) (rtc_ctrl_t *const p_ctrl, rtc_event_t irq)
.periodicIrqRateSet	ssp_err_t(* rtc_api_t::periodicIrqRateSet) (rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t rate)
.infoGet	ssp_err_t(* rtc_api_t::infoGet) (rtc_ctrl_t *p_ctrl, rtc_info_t *p_rtc_info)
.errorAdjustmentModeSet	ssp_err_t(* rtc_api_t::errorAdjustmentModeSet) (rtc_ctrl_t *p_ctrl, rtc_error_adjustment_mode_cfg_t *p_error_adjustment_mode)
.errorAdjustmentSet	ssp_err_t(* rtc_api_t::errorAdjustmentSet) (rtc_ctrl_t *p_ctrl, rtc_error_adjustment_cfg_t *p_error_adjustment_config)
.versionGet	ssp_err_t(* rtc_api_t::versionGet) (ssp_version_t *version)

11.2.31 Interface: sdmmc_api_t

Description: SD/MMC Interface

Function name	Definition
.open	ssp_err_t(* sdmmc_api_t::open) (sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const *const p_cfg)
.close	ssp_err_t(* sdmmc_api_t::close) (sdmmc_ctrl_t *const p_ctrl)
.read	ssp_err_t(* sdmmc_api_t::read) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)
.write	ssp_err_t(* sdmmc_api_t::write) (sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count)
.control	ssp_err_t(* sdmmc_api_t::control) (sdmmc_ctrl_t *const p_ctrl, ssp_command_t const command, void *p_data)
.readIo	ssp_err_t(* sdmmc_api_t::readIo) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address)
.writeIo	ssp_err_t(* sdmmc_api_t::writeIo) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write)
.readIoExt	ssp_err_t(* sdmmc_api_t::readIoExt) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
.writeIoExt	ssp_err_t(* sdmmc_api_t::writeIoExt) (sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
.ioIntEnable	ssp_err_t(* sdmmc_api_t::ioIntEnable) (sdmmc_ctrl_t *const p_ctrl, bool enable)
.versionGet	ssp_err_t(* sdmmc_api_t::versionGet) (ssp_version_t *const p_version)
.infoGet	ssp_err_t(* sdmmc_api_t::infoGet) (sdmmc_ctrl_t *const p_ctrl, sdmmc_info_t *const p_info)

Function name	Definition
.erase	ssp_err_t(* sdmmc_api_t::erase) (sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector, uint32_t const sector_count)

11.2.32 Interface: sf_adc_periodic_api_t

Description: ADC Periodic Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_adc_periodic_api_t::open) (sf_adc_periodic_ctrl_t *const p_ctrl, sf_adc_periodic_cfg_t const *const p_cfg)
.start	ssp_err_t(* sf_adc_periodic_api_t::start) (sf_adc_periodic_ctrl_t *const p_ctrl)
.stop	ssp_err_t(* sf_adc_periodic_api_t::stop) (sf_adc_periodic_ctrl_t *const p_ctrl)
.close	ssp_err_t(* sf_adc_periodic_api_t::close) (sf_adc_periodic_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_adc_periodic_api_t::versionGet) (ssp_version_t *const p_version)

11.2.33 Interface: sf_audio_playback_api_t

Description: Audio Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_audio_playback_api_t::open) (sf_audio_playback_ctrl_t *const p_ctrl, sf_audio_playback_cfg_t const *const p_cfg)
.close	ssp_err_t(* sf_audio_playback_api_t::close) (sf_audio_playback_ctrl_t *const p_ctrl)
.start	ssp_err_t(* sf_audio_playback_api_t::start) (sf_audio_playback_ctrl_t *const p_ctrl, sf_audio_playback_data_t *const p_data, UINT const timeout)

Function name	Definition
.pause	ssp_err_t(* sf_audio_playback_api_t::pause) (sf_audio_playback_ctrl_t *const p_ctrl)
.stop	ssp_err_t(* sf_audio_playback_api_t::stop) (sf_audio_playback_ctrl_t *const p_ctrl)
.resume	ssp_err_t(* sf_audio_playback_api_t::resume) (sf_audio_playback_ctrl_t *const p_ctrl)
.volumeSet	ssp_err_t(* sf_audio_playback_api_t::volumeSet) (sf_audio_playback_ctrl_t *const p_ctrl, uint8_t const volume)
.versionGet	ssp_err_t(* sf_audio_playback_api_t::versionGet) (ssp_version_t *const p_version)

11.2.34 Interface: sf_audio_playback_hw_api_t

Description: Audio Playback Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_audio_playback_hw_api_t::open) (sf_audio_playback_hw_ctrl_t *const p_ctrl, sf_audio_playback_hw_cfg_t const *const p_cfg)
.start	ssp_err_t(* sf_audio_playback_hw_api_t::start) (sf_audio_playback_hw_ctrl_t *const p_ctrl)
.stop	ssp_err_t(* sf_audio_playback_hw_api_t::stop) (sf_audio_playback_hw_ctrl_t *const p_ctrl)
.play	ssp_err_t(* sf_audio_playback_hw_api_t::play) (sf_audio_playback_hw_ctrl_t *const p_ctrl, int16_t const *const p_buffer, uint32_t length)
.dataTypeGet	ssp_err_t(* sf_audio_playback_hw_api_t::dataTypeGet) (sf_audio_playback_hw_ctrl_t *const p_ctrl, sf_audio_playback_data_type_t *const p_data_type)
.close	ssp_err_t(* sf_audio_playback_hw_api_t::close) (sf_audio_playback_hw_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_audio_playback_hw_api_t::versionGet) (ssp_version_t *const p_version)

11.2.35 Interface: sf_audio_record_api_t

Description: Audio Recording Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_audio_record_api_t::open) (sf_audio_record_ctrl_t *const p_ctrl, sf_audio_record_cfg_t const *const p_cfg)
.start	ssp_err_t(* sf_audio_record_api_t::start) (sf_audio_record_ctrl_t *const p_ctrl)
.stop	ssp_err_t(* sf_audio_record_api_t::stop) (sf_audio_record_ctrl_t *const p_ctrl)
.infoGet	ssp_err_t(* sf_audio_record_api_t::infoGet) (sf_audio_record_ctrl_t *const p_ctrl, sf_audio_record_info_t *p_info)
.close	ssp_err_t(* sf_audio_record_api_t::close) (sf_audio_record_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_audio_record_api_t::versionGet) (ssp_version_t *const p_version)

11.2.36 Interface: sf_ble_api_t

Description: SF BLE Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_ble_api_t::open) (sf_ble_ctrl_t *const p_ctrl, const sf_ble_cfg_t *p_cfg)
.close	ssp_err_t(* sf_ble_api_t::close) (sf_ble_ctrl_t *const p_ctrl)
.infoGet	ssp_err_t(* sf_ble_api_t::infoGet) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_info_t *p_ble_info)
.provisionGet	ssp_err_t(* sf_ble_api_t::provisionGet) (sf_ble_ctrl_t *const p_ctrl, sf_ble_provisioning_t *p_ble_provisioning)

Function name	Definition
.provisionSet	ssp_err_t(* sf_ble_api_t::provisionSet) (sf_ble_ctrl_t *const p_ctrl, const sf_ble_provisioning_t *p_ble_provisioning)
.scan	ssp_err_t(* sf_ble_api_t::scan) (sf_ble_ctrl_t *const p_ctrl, sf_ble_scan_t *p_scan, uint8_t *p_cnt, sf_ble_scan_info_t *p_scan_info)
.advertisementStart	ssp_err_t(* sf_ble_api_t::advertisementStart) (sf_ble_ctrl_t *const p_ctrl, sf_ble_adv_info_t *const p_advt_info)
.advertisementStop	ssp_err_t(* sf_ble_api_t::advertisementStop) (sf_ble_ctrl_t *const p_ctrl)
.whitelistAdd	ssp_err_t(* sf_ble_api_t::whitelistAdd) (sf_ble_ctrl_t *const p_ctrl, const uint8_t *p_bd_addr)
.whitelistDel	ssp_err_t(* sf_ble_api_t::whitelistDel) (sf_ble_ctrl_t *const p_ctrl, const uint8_t *p_bd_addr)
.bondingStart	ssp_err_t(* sf_ble_api_t::bondingStart) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, const uint8_t *p_bd_addr, sf_ble_bonding_start_t *p_bonding_start)
.bondingResponse	ssp_err_t(* sf_ble_api_t::bondingResponse) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, const uint8_t *p_bd_addr, sf_ble_bonding_response_t *p_bonding_resp)
.startEncryption	ssp_err_t(* sf_ble_api_t::startEncryption) (sf_ble_ctrl_t *const p_ctrl, sf_ble_sm_enc_info_t const *p_enc_info)
.connect	ssp_err_t(* sf_ble_api_t::connect) (sf_ble_ctrl_t *const p_ctrl, sf_ble_connection_t const *const p_conn, sf_ble_conn_handle_t *p_handle)
.listen	ssp_err_t(* sf_ble_api_t::listen) (sf_ble_ctrl_t *const p_ctrl)
.authorization	ssp_err_t(* sf_ble_api_t::authorization) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle)
.disconnect	ssp_err_t(* sf_ble_api_t::disconnect) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle)

Function name	Definition
.setTxPower	ssp_err_t(* sf_ble_api_t::setTxPower) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *const p_handle, sf_ble_set_tx_pwr_info_t *p_tx_power_info)
.gattAddCustomProfiles	ssp_err_t(* sf_ble_api_t::gattAddCustomProfiles) (sf_ble_ctrl_t *const p_ctrl, sf_ble_svc_attribute_t *p_svc_attr, uint32_t svc_attr_len, sf_ble_char_attribute_t *p_char_attr, uint32_t char_attr_len)
.gattServiceDiscovery	ssp_err_t(* sf_ble_api_t::gattServiceDiscovery) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_service_discovery_req_t const *const p_sf_ble_svc_dscv_req, sf_ble_service_discovery_rsp_t *const p_sf_ble_svc_dscv_rsp, uint32_t *const p_rsp_cnt)
.gattCharDiscovery	ssp_err_t(* sf_ble_api_t::gattCharDiscovery) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_char_discovery_req_t const *const p_sf_ble_char_dscv_req, sf_ble_char_discovery_rsp_t *const p_sf_ble_char_dscv_rsp, uint32_t *const p_rsp_cnt)
.gattCharDescDiscovery	ssp_err_t(* sf_ble_api_t::gattCharDescDiscovery) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t start_handle, uint16_t end_handle, sf_ble_char_desc_discovery_rsp_t *const p_sf_ble_chardesc_dscv_rsp, uint32_t *const p_rsp_cnt)
.gattCharRead	ssp_err_t(* sf_ble_api_t::gattCharRead) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_char_read_req_t const *const p_char_read_req, sf_ble_char_read_rsp_t *const p_char_read_rsp)
.gattCharWrite	ssp_err_t(* sf_ble_api_t::gattCharWrite) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_char_write_req_t const *const p_char_write_req)
.gattCharExecuteWrite	ssp_err_t(* sf_ble_api_t::gattCharExecuteWrite) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_ble_execute_write_t execute_flag)
.gattCharWriteLocal	ssp_err_t(* sf_ble_api_t::gattCharWriteLocal) (sf_ble_ctrl_t *const p_ctrl, uint16_t char_handle, uint16_t data_length, uint8_t *const p_data)
.gattSendNotify	ssp_err_t(* sf_ble_api_t::gattSendNotify) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t char_handle)

Function name	Definition
.gattSendIndicate	ssp_err_t(* sf_ble_api_t::gattSendIndicate) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t char_handle)
.gattWriteResponse	ssp_err_t(* sf_ble_api_t::gattWriteResponse) (sf_ble_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, uint16_t handle, sf_ble_attribute_error_code_t error_code)
.versionGet	ssp_err_t(* sf_ble_api_t::versionGet) (ssp_version_t *const p_version)

11.2.37 Interface: sf_ble_onboard_profile_api_t

Description: SF BLE On-Board Profile Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_ble_onboard_profile_api_t::open) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, const sf_ble_onboard_profile_cfg_t *p_cfg)
.close	ssp_err_t(* sf_ble_onboard_profile_api_t::close) (sf_ble_onboard_profile_ctrl_t *const p_ctrl)
.onbpEnable	ssp_err_t(* sf_ble_onboard_profile_api_t::onbpEnable) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_profile_callback_t p_prf_cb, sf_ble_prf_sec_t sec)
.onbpDisable	ssp_err_t(* sf_ble_onboard_profile_api_t::onbpDisable) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile)
.onbpServerWriteData	ssp_err_t(* sf_ble_onboard_profile_api_t::onbpServerWriteData) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t characteristics, const void *p_data)
.onbpServerSendNotification	ssp_err_t(* sf_ble_onboard_profile_api_t::onbpServerSendNotification) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t characteristics, const void *p_data)

Function name	Definition
.onbpServerSendIndication	ssp_err_t(* sf_ble_onboard_profile_api_t::onbpServerSendIndication) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t characteristics, const void *p_data)
.onbpClientWriteCCCD	ssp_err_t(* sf_ble_onboard_profile_api_t::onbpClientWriteCCCD) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t cccd_char, sf_ble_cccd_val_t cccd_val)
.onbpClientWriteChar	ssp_err_t(* sf_ble_onboard_profile_api_t::onbpClientWriteChar) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t characteristics, const void *p_data)
.onbpClientReadChar	ssp_err_t(* sf_ble_onboard_profile_api_t::onbpClientReadChar) (sf_ble_onboard_profile_ctrl_t *const p_ctrl, sf_ble_conn_handle_t *p_handle, sf_onbp_t profile, sf_ble_onbp_char_t characteristics)
.versionGet	ssp_err_t(* sf_ble_onboard_profile_api_t::versionGet) (ssp_version_t *const p_version)

11.2.38 Interface: sf_block_media_api_t

Description: Block Media Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_block_media_api_t::open) (sf_block_media_ctrl_t *p_ctrl, sf_block_media_cfg_t const *const p_cfg)
.read	ssp_err_t(* sf_block_media_api_t::read) (sf_block_media_ctrl_t *p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)
.write	ssp_err_t(* sf_block_media_api_t::write) (sf_block_media_ctrl_t *p_ctrl, uint8_t const *const p_src, uint32_t const start_sector, uint32_t const sector_count)

Function name	Definition
.ioctl	ssp_err_t(* sf_block_media_api_t::ioctl) (sf_block_media_ctrl_t *p_ctrl, ssp_command_t const command, void *p_data)
.close	ssp_err_t(* sf_block_media_api_t::close) (sf_block_media_ctrl_t *p_ctrl)
.versionGet	ssp_err_t(* sf_block_media_api_t::versionGet) (ssp_version_t *const p_version)

11.2.39 Interface: sf_cellular_api_t

Description: SF CELLULAR Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_cellular_api_t::open) (sf_cellular_ctrl_t *p_ctrl, sf_cellular_cfg_t const *const p_cfg)
.close	ssp_err_t(* sf_cellular_api_t::close) (sf_cellular_ctrl_t *const p_ctrl)
.provisioningGet	ssp_err_t(* sf_cellular_api_t::provisioningGet) (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_provisioning_t *const p_cellular_provisioning)
.provisioningSet	ssp_err_t(* sf_cellular_api_t::provisioningSet) (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_provisioning_t const *const p_cellular_provisioning)
.infoGet	ssp_err_t(* sf_cellular_api_t::infoGet) (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_info_t *const p_cellular_info)
.statisticsGet	ssp_err_t(* sf_cellular_api_t::statisticsGet) (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_stats_t *const p_cellular_device_stats)
.transmit	ssp_err_t(* sf_cellular_api_t::transmit) (sf_cellular_ctrl_t *const p_ctrl, uint8_t *const p_buf, uint32_t length)
.versionGet	ssp_err_t(* sf_cellular_api_t::versionGet) (ssp_version_t *const p_version)

Function name	Definition
.networkConnect	ssp_err_t(* sf_cellular_api_t::networkConnect) (sf_cellular_ctrl_t *const p_ctrl)
.networkDisconnect	ssp_err_t(* sf_cellular_api_t::networkDisconnect) (sf_cellular_ctrl_t *const p_ctrl)
.networkStatusGet	ssp_err_t(* sf_cellular_api_t::networkStatusGet) (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_network_status_t *p_network_status)
.simPinSet	ssp_err_t(* sf_cellular_api_t::simPinSet) (sf_cellular_ctrl_t *const p_ctrl, uint8_t *const p_old_pin, uint8_t *const p_new_pin)
.simLock	ssp_err_t(* sf_cellular_api_t::simLock) (sf_cellular_ctrl_t *const p_ctrl, uint8_t *const p_pin)
.simUnlock	ssp_err_t(* sf_cellular_api_t::simUnlock) (sf_cellular_ctrl_t *const p_ctrl, uint8_t *const p_pin)
.simIDGet	ssp_err_t(* sf_cellular_api_t::simIDGet) (sf_cellular_ctrl_t *const p_ctrl, uint8_t *p_sim_id)
.fotaCheck	ssp_err_t(* sf_cellular_api_t::fotaCheck) (sf_cellular_ctrl_t *const p_ctrl, void *p_fotacheck)
.fotaStart	ssp_err_t(* sf_cellular_api_t::fotaStart) (sf_cellular_ctrl_t *const p_ctrl, void *p_fotastart)
.fotaStop	ssp_err_t(* sf_cellular_api_t::fotaStop) (sf_cellular_ctrl_t *const p_ctrl, void *p_fotastop)
.reset	ssp_err_t(* sf_cellular_api_t::reset) (sf_cellular_ctrl_t *const p_ctrl, sf_cellular_reset_type_t reset_type)

11.2.40 Interface: sf_cellular_socket_api_t

Description: SF Socket CELLULAR Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_cellular_socket_api_t::open) (sf_cellular_socket_ctrl_t *p_ctrl, sf_cellular_socket_cfg_t const *const p_cfg)

Function name	Definition
.close	ssp_err_t(* sf_cellular_socket_api_t::close) (sf_cellular_socket_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_cellular_socket_api_t::versionGet) (ssp_version_t *const p_version)

11.2.41 Interface: sf_comms_api_t

Description: Communications Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_comms_api_t::open) (sf_comms_ctrl_t *const p_ctrl, sf_comms_cfg_t const *const p_cfg)
.close	ssp_err_t(* sf_comms_api_t::close) (sf_comms_ctrl_t *const p_ctrl)
.read	ssp_err_t(* sf_comms_api_t::read) (sf_comms_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, UINT const timeout)
.write	ssp_err_t(* sf_comms_api_t::write) (sf_comms_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes, UINT const timeout)
.lock	ssp_err_t(* sf_comms_api_t::lock) (sf_comms_ctrl_t *const p_ctrl, sf_comms_lock_t lock_type, UINT timeout)
.unlock	ssp_err_t(* sf_comms_api_t::unlock) (sf_comms_ctrl_t *const p_ctrl, sf_comms_lock_t lock_type)
.versionGet	ssp_err_t(* sf_comms_api_t::versionGet) (ssp_version_t *const p_version)

11.2.42 Interface: sf_console_api_t

Description: Console Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_console_api_t::open) (sf_console_ctrl_t *const p_ctrl, sf_console_cfg_t const *const p_cfg)
.close	ssp_err_t(* sf_console_api_t::close) (sf_console_ctrl_t *const p_ctrl)
.prompt	ssp_err_t(* sf_console_api_t::prompt) (sf_console_ctrl_t *const p_ctrl, sf_console_menu_t const *const p_menu, UINT const timeout)
.parse	ssp_err_t(* sf_console_api_t::parse) (sf_console_ctrl_t *const p_ctrl, sf_console_menu_t const *const p_cmd_list, uint8_t const *const p_input, uint32_t const bytes)
.read	ssp_err_t(* sf_console_api_t::read) (sf_console_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, uint32_t const timeout)
.write	ssp_err_t(* sf_console_api_t::write) (sf_console_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const timeout)
.argumentFind	ssp_err_t(* sf_console_api_t::argumentFind) (uint8_t const *const p_arg, uint8_t const *const p_str, int32_t *const p_index, int32_t *const p_data)
.versionGet	ssp_err_t(* sf_console_api_t::versionGet) (ssp_version_t *const p_version)

11.2.43 Interface: sf_el_gx_api_t

Description: GUIX Interface

Function name	Definition
.open	ssp_err_t(* sf_el_gx_api_t::open) (sf_el_gx_ctrl_t *const p_ctrl, sf_el_gx_cfg_t const *const p_cfg)
.close	ssp_err_t(* sf_el_gx_api_t::close) (sf_el_gx_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_el_gx_api_t::versionGet) (ssp_version_t *p_version)

Function name	Definition
.setup	UINT(* sf_el_gx_api_t::setup) (GX_DISPLAY *p_display)
.canvasInit	ssp_err_t(* sf_el_gx_api_t::canvasInit) (sf_el_gx_ctrl_t *const p_ctrl, GX_WINDOW_ROOT *p_window_root)

11.2.44 Interface: sf_external_irq_api_t

Description: External IRQ Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_external_irq_api_t::open) (sf_external_irq_ctrl_t *const p_ctrl, sf_external_irq_cfg_t const *const p_cfg)
.wait	ssp_err_t(* sf_external_irq_api_t::wait) (sf_external_irq_ctrl_t *const p_ctrl, ULONG const timeout)
.versionGet	ssp_err_t(* sf_external_irq_api_t::versionGet) (ssp_version_t *const p_version)
.close	ssp_err_t(* sf_external_irq_api_t::close) (sf_external_irq_ctrl_t *const p_ctrl)

11.2.45 Interface: sf_i2c_api_t

Description: I2C Framework

Function name	Definition
.open	ssp_err_t(* sf_i2c_api_t::open) (sf_i2c_ctrl_t *p_ctrl, sf_i2c_cfg_t const *const p_cfg)
.read	ssp_err_t(* sf_i2c_api_t::read) (sf_i2c_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart, uint32_t const timeout)
.write	ssp_err_t(* sf_i2c_api_t::write) (sf_i2c_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart, uint32_t const timeout)

Function name	Definition
.reset	ssp_err_t(* sf_i2c_api_t::reset) (sf_i2c_ctrl_t *const p_ctrl, uint32_t const timeout)
.close	ssp_err_t(* sf_i2c_api_t::close) (sf_i2c_ctrl_t *const p_ctrl)
.lock	ssp_err_t(* sf_i2c_api_t::lock) (sf_i2c_ctrl_t *const p_ctrl)
.unlock	ssp_err_t(* sf_i2c_api_t::unlock) (sf_i2c_ctrl_t *const p_ctrl)
.version	ssp_err_t(* sf_i2c_api_t::version) (ssp_version_t *const p_version)

11.2.46 Interface: sf_jpeg_decode_api_t

Description: JPEG Decode Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_jpeg_decode_api_t::open) (sf_jpeg_decode_ctrl_t *const p_ctrl, sf_jpeg_decode_cfg_t const *const p_cfg)
.inputBufferSet	ssp_err_t(* sf_jpeg_decode_api_t::inputBufferSet) (sf_jpeg_decode_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t const buffer_size)
.outputBufferSet	ssp_err_t(* sf_jpeg_decode_api_t::outputBufferSet) (sf_jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)
.linesDecodedGet	ssp_err_t(* sf_jpeg_decode_api_t::linesDecodedGet) (sf_jpeg_decode_ctrl_t *const p_ctrl, uint32_t *const p_lines)
.horizontalStrideSet	ssp_err_t(* sf_jpeg_decode_api_t::horizontalStrideSet) (sf_jpeg_decode_ctrl_t *const p_ctrl, uint32_t horizontal_stride)
.imageSubsampleSet	ssp_err_t(* sf_jpeg_decode_api_t::imageSubsampleSet) (sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample)

Function name	Definition
.wait	ssp_err_t(* sf_jpeg_decode_api_t::wait) (sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t *const p_status, uint32_t timeout)
.statusGet	ssp_err_t(* sf_jpeg_decode_api_t::statusGet) (sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t *const p_status)
.imageSizeGet	ssp_err_t(* sf_jpeg_decode_api_t::imageSizeGet) (sf_jpeg_decode_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)
.pixelFormatGet	ssp_err_t(* sf_jpeg_decode_api_t::pixelFormatGet) (sf_jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_color_space_t *const p_color_space)
.close	ssp_err_t(* sf_jpeg_decode_api_t::close) (sf_jpeg_decode_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_jpeg_decode_api_t::versionGet) (ssp_version_t *const p_version)

11.2.47 Interface: sf_message_api_t

Description: Messaging Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_message_api_t::open) (sf_message_ctrl_t *const p_ctrl, sf_message_cfg_t const *const p_cfg)
.close	ssp_err_t(* sf_message_api_t::close) (sf_message_ctrl_t *const p_ctrl)
.bufferAcquire	ssp_err_t(* sf_message_api_t::bufferAcquire) (sf_message_ctrl_t const *const p_ctrl, sf_message_header_t **pp_buffer, sf_message_acquire_cfg_t const *const p_acquire_cfg, uint32_t const wait_option)
.bufferRelease	ssp_err_t(* sf_message_api_t::bufferRelease) (sf_message_ctrl_t *const p_ctrl, sf_message_header_t *const p_buffer, sf_message_release_option_t const option)

Function name	Definition
.post	ssp_err_t(* sf_message_api_t::post) (sf_message_ctrl_t *const p_ctrl, sf_message_header_t const *const p_buffer, sf_message_post_cfg_t const *const p_post_cfg, sf_message_post_err_t *const p_post_err, uint32_t const wait_option)
.pend	ssp_err_t(* sf_message_api_t::pend) (sf_message_ctrl_t const *const p_ctrl, TX_QUEUE const *const p_queue, sf_message_header_t **pp_buffer, uint32_t const wait_option)
.versionGet	ssp_err_t(* sf_message_api_t::versionGet) (ssp_version_t *const p_version)

11.2.48 Interface: sf_power_profiles_api_t

Description: Power Profiles Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_power_profiles_api_t::open) (sf_power_profiles_ctrl_t *const p_ctrl, sf_power_profiles_cfg_t const *const p_cfg)
.sleep	ssp_err_t(* sf_power_profiles_api_t::sleep) (sf_power_profiles_ctrl_t *const p_ctrl)
.close	ssp_err_t(* sf_power_profiles_api_t::close) (sf_power_profiles_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_power_profiles_api_t::versionGet) (ssp_version_t *const p_version)

11.2.49 Interface: sf_power_profiles_v2_api_t

Description: Power Profiles Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_power_profiles_v2_api_t::open) (sf_power_profiles_v2_ctrl_t *const p_ctrl, sf_power_profiles_v2_cfg_t const *const p_cfg)

Function name	Definition
.runApply	ssp_err_t(* sf_power_profiles_v2_api_t::runApply) (sf_power_profiles_v2_ctrl_t *const p_ctrl, sf_power_profiles_v2_run_cfg_t const *const p_cfg)
.lowPowerApply	ssp_err_t(* sf_power_profiles_v2_api_t::lowPowerApply) (sf_power_profiles_v2_ctrl_t *const p_ctrl, sf_power_profiles_v2_low_power_cfg_t const *const p_cfg)
.close	ssp_err_t(* sf_power_profiles_v2_api_t::close) (sf_power_profiles_v2_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_power_profiles_v2_api_t::versionGet) (ssp_version_t *const p_version)

11.2.50 Interface: sf_socket_api_t

Description: SF Socket WIFI Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_socket_api_t::open) (sf_socket_ctrl_t *p_ctrl, sf_socket_cfg_t const *const p_cfg)
.close	ssp_err_t(* sf_socket_api_t::close) (sf_socket_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_socket_api_t::versionGet) (ssp_version_t *const p_version)

11.2.51 Interface: sf_spi_api_t

Description: SPI Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_spi_api_t::open) (sf_spi_ctrl_t *p_ctrl, sf_spi_cfg_t const *const p_cfg)
.read	ssp_err_t(* sf_spi_api_t::read) (sf_spi_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout)

Function name	Definition
.write	ssp_err_t(* sf_spi_api_t::write) (sf_spi_ctrl_t *const p_ctrl, void *const p_src, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout)
.writeRead	ssp_err_t(* sf_spi_api_t::writeRead) (sf_spi_ctrl_t *const p_ctrl, void *const p_src, void *const p_dest, uint32_t const length, spi_bit_width_t const bit_width, uint32_t const timeout)
.close	ssp_err_t(* sf_spi_api_t::close) (sf_spi_ctrl_t *const p_ctrl)
.lock	ssp_err_t(* sf_spi_api_t::lock) (sf_spi_ctrl_t *const p_ctrl)
.unlock	ssp_err_t(* sf_spi_api_t::unlock) (sf_spi_ctrl_t *const p_ctrl)
.version	ssp_err_t(* sf_spi_api_t::version) (ssp_version_t *const p_version)

11.2.52 Interface: sf_thread_monitor_api_t

Description: Thread Monitor Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_thread_monitor_api_t::open) (sf_thread_monitor_ctrl_t *const p_ctrl, sf_thread_monitor_cfg_t const *const p_cfg)
.close	ssp_err_t(* sf_thread_monitor_api_t::close) (sf_thread_monitor_ctrl_t *const p_ctrl)
.threadRegister	ssp_err_t(* sf_thread_monitor_api_t::threadRegister) (sf_thread_monitor_ctrl_t *const p_ctrl, sf_thread_monitor_counter_min_max_t const *p_counter_min_max)
.threadUnregister	ssp_err_t(* sf_thread_monitor_api_t::threadUnregister) (sf_thread_monitor_ctrl_t *const p_ctrl)
.countIncrement	ssp_err_t(* sf_thread_monitor_api_t::countIncrement) (sf_thread_monitor_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_thread_monitor_api_t::versionGet) (ssp_version_t *const p_version)

11.2.53 Interface: sf_touch_ctsu_api_t

Description: CTSU Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_touch_ctsu_api_t::open) (sf_touch_ctsu_ctrl_t *const p_ctrl, sf_touch_ctsu_cfg_t const *const p_cfg)
.read	ssp_err_t(* sf_touch_ctsu_api_t::read) (sf_touch_ctsu_ctrl_t *const p_ctrl, void *p_dest, ctsu_read_t opts, const ctsu_channel_pair_t *channels, const uint16_t count)
.close	ssp_err_t(* sf_touch_ctsu_api_t::close) (sf_touch_ctsu_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_touch_ctsu_api_t::versionGet) (ssp_version_t *const p_version)

11.2.54 Interface: sf_touch_ctsu_button_api_t

Description: CTSU Button Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_touch_ctsu_button_api_t::open) (sf_touch_ctsu_button_ctrl_t *const p_ctrl, sf_touch_ctsu_button_cfg_t const *const p_cfg)
.enable	ssp_err_t(* sf_touch_ctsu_button_api_t::enable) (sf_touch_ctsu_button_ctrl_t *const p_ctrl, sf_touch_ctsu_button_id const button_id)
.disable	ssp_err_t(* sf_touch_ctsu_button_api_t::disable) (sf_touch_ctsu_button_ctrl_t *const p_ctrl, sf_touch_ctsu_button_id const button_id)
.close	ssp_err_t(* sf_touch_ctsu_button_api_t::close) (sf_touch_ctsu_button_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_touch_ctsu_button_api_t::versionGet) (ssp_version_t *const p_version)

11.2.55 Interface: sf_touch_ctsu_slider_api_t

Description: CTSU Slider Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_touch_ctsu_slider_api_t::open) (sf_touch_ctsu_slider_ctrl_t *const p_ctrl, sf_touch_ctsu_slider_cfg_t const *const p_cfg)
.enable	ssp_err_t(* sf_touch_ctsu_slider_api_t::enable) (sf_touch_ctsu_slider_ctrl_t *const p_ctrl, sf_touch_ctsu_slider_id_t const slider_id)
.disable	ssp_err_t(* sf_touch_ctsu_slider_api_t::disable) (sf_touch_ctsu_slider_ctrl_t *const p_ctrl, sf_touch_ctsu_slider_id_t const slider_id)
.close	ssp_err_t(* sf_touch_ctsu_slider_api_t::close) (sf_touch_ctsu_slider_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_touch_ctsu_slider_api_t::versionGet) (ssp_version_t *const p_version)

11.2.56 Interface: sf_touch_panel_api_t

Description: Touch Panel Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_touch_panel_api_t::open) (sf_touch_panel_ctrl_t *const p_ctrl, sf_touch_panel_cfg_t const *const p_cfg)
.calibrate	ssp_err_t(* sf_touch_panel_api_t::calibrate) (sf_touch_panel_ctrl_t *const p_ctrl, sf_touch_panel_calibrate_t const *const p_expected, sf_touch_panel_payload_t const *const p_actual, ULONG timeout)
.start	ssp_err_t(* sf_touch_panel_api_t::start) (sf_touch_panel_ctrl_t *const p_ctrl)
.stop	ssp_err_t(* sf_touch_panel_api_t::stop) (sf_touch_panel_ctrl_t *const p_ctrl)

Function name	Definition
.reset	ssp_err_t(* sf_touch_panel_api_t::reset) (sf_touch_panel_ctrl_t *const p_ctrl)
.close	ssp_err_t(* sf_touch_panel_api_t::close) (sf_touch_panel_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_touch_panel_api_t::versionGet) (ssp_version_t *const p_version)

11.2.57 Interface: sf_wifi_api_t

Description: SF WIFI Framework Interface

Function name	Definition
.open	ssp_err_t(* sf_wifi_api_t::open) (sf_wifi_ctrl_t *p_ctrl, sf_wifi_cfg_t const *const p_cfg)
.close	ssp_err_t(* sf_wifi_api_t::close) (sf_wifi_ctrl_t *const p_ctrl)
.multicastListAdd	ssp_err_t(* sf_wifi_api_t::multicastListAdd) (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac_addr)
.multicastListDelete	ssp_err_t(* sf_wifi_api_t::multicastListDelete) (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac_addr)
.statisticsGet	ssp_err_t(* sf_wifi_api_t::statisticsGet) (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_stats_t *const p_wifi_device_stats)
.transmit	ssp_err_t(* sf_wifi_api_t::transmit) (sf_wifi_ctrl_t *const p_ctrl, uint8_t *const p_buf, uint32_t length)
.provisioningSet	ssp_err_t(* sf_wifi_api_t::provisioningSet) (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_provisioning_t const *const p_wifi_provisioning)
.provisioningGet	ssp_err_t(* sf_wifi_api_t::provisioningGet) (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_provisioning_t *const p_wifi_provisioning)
.infoGet	ssp_err_t(* sf_wifi_api_t::infoGet) (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_info_t *const p_wifi_info)

Function name	Definition
.scan	ssp_err_t(* sf_wifi_api_t::scan) (sf_wifi_ctrl_t *const p_ctrl, sf_wifi_scan_t *const p_scan, uint8_t *const p_cnt)
.ACLAdd	ssp_err_t(* sf_wifi_api_t::ACLAdd) (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac)
.ACLDelete	ssp_err_t(* sf_wifi_api_t::ACLDelete) (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac)
.macAddressGet	ssp_err_t(* sf_wifi_api_t::macAddressGet) (sf_wifi_ctrl_t *const p_ctrl, uint8_t *const p_mac)
.macAddressSet	ssp_err_t(* sf_wifi_api_t::macAddressSet) (sf_wifi_ctrl_t *const p_ctrl, uint8_t const *const p_mac)
.versionGet	ssp_err_t(* sf_wifi_api_t::versionGet) (ssp_version_t *const p_version)

11.2.58 Interface: sf_wifi_onchip_stack_api_t

Description: SF WIFI On-Chip Stack Interface

Function name	Definition
.open	ssp_err_t(* sf_wifi_onchip_stack_api_t::open) (sf_wifi_onchip_stack_ctrl_t *p_ctrl, sf_wifi_onchip_stack_cfg_t const *const p_cfg)
.close	ssp_err_t(* sf_wifi_onchip_stack_api_t::close) (sf_wifi_onchip_stack_ctrl_t *const p_ctrl)
.ipAddressCfg	ssp_err_t(* sf_wifi_onchip_stack_api_t::ipAddressCfg) (sf_wifi_onchip_stack_ctrl_t *const p_ctrl, sf_wifi_onchip_stack_ip_cfg_t *const p_cfg)
.dhcpServerStart	ssp_err_t(* sf_wifi_onchip_stack_api_t::dhcpServerStart) (sf_wifi_onchip_stack_ctrl_t *const p_ctrl, sf_wifi_ip_addr_t const *const p_start_ip, sf_wifi_ip_addr_t const *const p_end_ip)
.dhcpServerStop	ssp_err_t(* sf_wifi_onchip_stack_api_t::dhcpServerStop) (sf_wifi_onchip_stack_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* sf_wifi_onchip_stack_api_t::versionGet) (ssp_version_t *const p_version)

11.2.59 Interface: slcdc_api_t

Description: SLCDC Interface

Function name	Definition
.open	ssp_err_t(* slcdc_api_t::open) (slcdc_ctrl_t *const p_ctrl, slcdc_cfg_t const *const p_cfg)
.write	ssp_err_t(* slcdc_api_t::write) (slcdc_ctrl_t *const p_ctrl, slcdc_size_t const start_segment, slcdc_size_t const *const p_data, slcdc_size_t const segment_count)
.modify	ssp_err_t(* slcdc_api_t::modify) (slcdc_ctrl_t *const p_ctrl, slcdc_size_t const segment, slcdc_size_t const data_mask, slcdc_size_t const data)
.start	ssp_err_t(* slcdc_api_t::start) (slcdc_ctrl_t *const p_ctrl)
.stop	ssp_err_t(* slcdc_api_t::stop) (slcdc_ctrl_t *const p_ctrl)
.contrastIncrease	ssp_err_t(* slcdc_api_t::contrastIncrease) (slcdc_ctrl_t *const p_ctrl)
.contrastDecrease	ssp_err_t(* slcdc_api_t::contrastDecrease) (slcdc_ctrl_t *const p_ctrl)
.setdisplayArea	ssp_err_t(* slcdc_api_t::setdisplayArea) (slcdc_ctrl_t *const p_ctrl, slcdc_display_area_t const display_area)
.close	ssp_err_t(* slcdc_api_t::close) (slcdc_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* slcdc_api_t::versionGet) (ssp_version_t *p_version)

11.2.60 Interface: spi_api_t

Description: SPI Interface

Function name	Definition
.open	ssp_err_t(* spi_api_t::open) (spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg)

Function name	Definition
.read	ssp_err_t(* spi_api_t::read) (spi_ctrl_t *const p_ctrl, void const *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
.write	ssp_err_t(* spi_api_t::write) (spi_ctrl_t *const p_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)
.writeRead	ssp_err_t(* spi_api_t::writeRead) (spi_ctrl_t *const p_ctrl, void const *p_src, void const *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
.close	ssp_err_t(* spi_api_t::close) (spi_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* spi_api_t::versionGet) (ssp_version_t *p_version)

11.2.61 Interface: timer_api_t

Description: Timer Interface

Function name	Definition
.open	ssp_err_t(* timer_api_t::open) (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)
.stop	ssp_err_t(* timer_api_t::stop) (timer_ctrl_t *const p_ctrl)
.start	ssp_err_t(* timer_api_t::start) (timer_ctrl_t *const p_ctrl)
.reset	ssp_err_t(* timer_api_t::reset) (timer_ctrl_t *const p_ctrl)
.counterGet	ssp_err_t(* timer_api_t::counterGet) (timer_ctrl_t *const p_ctrl, timer_size_t *const p_value)
.periodSet	ssp_err_t(* timer_api_t::periodSet) (timer_ctrl_t *const p_ctrl, timer_size_t const period, timer_unit_t const unit)
.dutyCycleSet	ssp_err_t(* timer_api_t::dutyCycleSet) (timer_ctrl_t *const p_ctrl, timer_size_t const duty_cycle, timer_pwm_unit_t const duty_cycle_unit, uint8_t const pin)
.infoGet	ssp_err_t(* timer_api_t::infoGet) (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)

Function name	Definition
.close	ssp_err_t(* timer_api_t::close) (timer_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* timer_api_t::versionGet) (ssp_version_t *const p_version)

11.2.62 Interface: transfer_api_t

Description: Transfer Interface

Function name	Definition
.open	ssp_err_t(* transfer_api_t::open) (transfer_ctrl_t *const p_ctrl, transfer_cfg_t const *const p_cfg)
.reset	ssp_err_t(* transfer_api_t::reset) (transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const num_transfers)
.enable	ssp_err_t(* transfer_api_t::enable) (transfer_ctrl_t *const p_ctrl)
.disable	ssp_err_t(* transfer_api_t::disable) (transfer_ctrl_t *const p_ctrl)
.start	ssp_err_t(* transfer_api_t::start) (transfer_ctrl_t *const p_ctrl, transfer_start_mode_t mode)
.stop	ssp_err_t(* transfer_api_t::stop) (transfer_ctrl_t *const p_ctrl)
.infoGet	ssp_err_t(* transfer_api_t::infoGet) (transfer_ctrl_t *const p_ctrl, transfer_properties_t *const p_info)
.close	ssp_err_t(* transfer_api_t::close) (transfer_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* transfer_api_t::versionGet) (ssp_version_t *const p_version)
.blockReset	ssp_err_t(* transfer_api_t::blockReset) (transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const length, transfer_size_t size, uint16_t const num_transfers)

11.2.63 Interface: trng_api_t

Description: Random number generation

Function name	Definition
.open	uint32_t(* trng_api_t::open) (trng_ctrl_t *const p_ctrl, trng_cfg_t const *const p_cfg)
.read	uint32_t(* trng_api_t::read) (trng_ctrl_t *const p_ctrl, uint32_t *const p_rngbuf, uint32_t nwords)
.close	uint32_t(* trng_api_t::close) (trng_ctrl_t *const p_ctrl)
.versionGet	uint32_t(* trng_api_t::versionGet) (ssp_version_t *const p_version)

11.2.64 Interface: uart_api_t

Description: UART Interface

Function name	Definition
.open	ssp_err_t(* uart_api_t::open) (uart_ctrl_t *const p_ctrl, uart_cfg_t const *const p_cfg)
.read	ssp_err_t(* uart_api_t::read) (uart_ctrl_t *const p_ctrl, uint8_t const *const p_dest, uint32_t const bytes)
.write	ssp_err_t(* uart_api_t::write) (uart_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes)
.baudSet	ssp_err_t(* uart_api_t::baudSet) (uart_ctrl_t *const p_ctrl, uint32_t const baudrate)
.infoGet	ssp_err_t(* uart_api_t::infoGet) (uart_ctrl_t *const p_ctrl, uart_info_t *const p_info)
.close	ssp_err_t(* uart_api_t::close) (uart_ctrl_t *const p_ctrl)
.versionGet	ssp_err_t(* uart_api_t::versionGet) (ssp_version_t *p_version)
.communicationAbort	ssp_err_t(* uart_api_t::communicationAbort) (uart_ctrl_t *const p_ctrl, uart_dir_t communication_to_abort)

11.2.65 Interface: wdt_api_t

Description: WDT Interface

Function name	Definition
.cfgGet	ssp_err_t(* wdt_api_t::cfgGet) (wdt_ctrl_t *const p_ctrl, wdt_cfg_t *const p_cfg)
.open	ssp_err_t(* wdt_api_t::open) (wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)
.refresh	ssp_err_t(* wdt_api_t::refresh) (wdt_ctrl_t *const p_ctrl)
.statusGet	ssp_err_t(* wdt_api_t::statusGet) (wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)
.statusClear	ssp_err_t(* wdt_api_t::statusClear) (wdt_ctrl_t *const p_ctrl, const wdt_status_t status)
.counterGet	ssp_err_t(* wdt_api_t::counterGet) (wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)
.timeoutGet	ssp_err_t(* wdt_api_t::timeoutGet) (wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout)
.versionGet	ssp_err_t(* wdt_api_t::versionGet) (ssp_version_t *const p_data)

Legal Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.

3. This document contains information that is the exclusive property of Renesas Electronics. Your use of this document and the material in the document is subject to the terms and conditions of your Renesas SSP license agreement. Unless otherwise agreed in an SSP license agreement with Renesas: 1) you may not use, copy, modify, distribute, display, or perform the document or any of its contents; and 2) you may not use any name or mark of Renesas Electronics for advertising or publicity purposes or in connection with your use of the contents.

4. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

5. You may create a printed copy of this document solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Renesas Electronics. Renesas Electronics reserves all rights to this document not expressly granted above.

6. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.

7. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

“Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

“High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anticrime systems; and safety equipment etc.

Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.

8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.

9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.

10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

11. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.

12. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.

13. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.

14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

Renesas Synergy™ SSP

User's Manual

Publication Date: Rev.01.03 Apr 2018

Renesas Synergy™ SSP v1.4.0

User's Manual

Renesas Electronics Corporation