User's Manual

# Renesas Flexible Software Package (FSP) v1.1.0

User's Manual

Renesas RA Family

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Revision 1.00 Apr.5.20

# Table of Contents


**Chapter 1 Introduction** . . . 7
1.1 Overview . . . 7
1.2 How to Read this Manual . . . 7
1.3 Documentation Standard . . . 7
**Chapter 2 Starting Development** . . . 9
2.1 Starting Development Introduction . . . 9
2.1.1 Getting Started with e2 studio and FSP . . . 9
2.2 e2 studio User Guide . . . 10
2.2.1 What is e2 studio? . . . 10
2.2.2 e2 studio Prerequisites . . . 12
2.2.2.1 Obtaining an RA MCU Kit . . . 12
2.2.2.2 PC Requirements . . . 12
2.2.2.3 Installing e2 studio, platform installer and the FSP package . . . 12
2.2.2.4 Choosing a Toolchain . . . 12
2.2.2.5 Licensing . . . 13
2.2.3 What is a Project? . . . 13
2.2.4 Creating a Project . . . 14
2.2.4.1 Creating a New Project . . . 15
2.2.4.2 Selecting a Board and Toolchain . . . 16
2.2.4.3 Selecting a Project Template . . . 17
2.2.5 Configuring a Project . . . 18
2.2.5.1 Configuring the BSP with e2 studio . . . 19
2.2.5.2 Configuring Clocks . . . 20
2.2.5.3 Configuring Pins . . . 20
2.2.5.4 Configuring Interrupts . . . 23
2.2.5.5 Viewing Event Links . . . 24
2.2.6 Adding Threads and Drivers . . . 25
2.2.6.1 Adding and Configuring HAL Drivers . . . 26
2.2.6.2 Adding Drivers to a Thread and Configuring the Drivers . . . 27
2.2.6.3 Configuring Threads . . . 30
2.2.7 Reviewing and Adding Components . . . 31
2.2.8 Writing the Application . . . 31
2.2.8.1 Coding Features . . . 31
2.2.8.2 RTOS-independent Applications . . . 37
2.2.8.3 RTOS Applications . . . 38
2.2.9 Debugging the Project . . . 39
2.2.10 Modifying Toolchain Settings . . . 40
2.2.11 Importing an Existing Project into e2 studio . . . 41
2.3 Tutorial: Your First RA MCU Project - Blinky . . . 45
2.3.1 Tutorial Blinky . . . 45
2.3.2 What Does Blinky Do? . . . 45
2.3.3 Prerequisites . . . 45
2.3.4 Create a New Project for Blinky . . . 45
2.3.4.1 Details about the Blinky Configuration . . . 48
2.3.4.2 Configuring the Blinky Clocks . . . 48
2.3.4.3 Configuring the Blinky Pins . . . 48
2.3.4.4 Configuring the Parameters for Blinky Components . . . 48
2.3.4.5 Where is main()? . . . 48
2.3.4.6 Blinky Example Code . . . 48
2.3.5 Build the Blinky Project . . . 49
2.3.6 Debug the Blinky Project . . . 49

2.3.6.1 Debug prerequisites . . . 49
2.3.6.2 Debug steps . . . 50
2.3.6.3 Details about the Debug Process . . . 51
2.3.7 Run the Blinky Project . . . 52
2.4 Tutorial: Using HAL Drivers - Programming the WDT . . . 52
2.4.1 Application WDT . . . 52
2.4.2 Creating a WDT Application Using the RA MCU FSP and e2 studio . . . 52
2.4.2.1 Using the FSP and e2 studio . . . 52
2.4.2.2 The WDT Application . . . 52
2.4.2.3 WDT Application flow . . . 53
2.4.3 Creating the Project with e2 studio . . . 53
2.4.4 Configuring the Project with e2 studio . . . 56
2.4.4.1 BSP Tab . . . 57
2.4.4.2 Clocks Tab . . . 57
2.4.4.3 Pins Tab . . . 58
2.4.4.4 Stacks Tab . . . 58
2.4.4.5 Components Tab . . . 60
2.4.5 WDT Generated Project Files . . . 61
2.4.5.1 WDT hal_data.h . . . 62
2.4.5.2 WDT hal_data.c . . . 63
2.4.5.3 WDT main.c . . . 64
2.4.5.4 WDT hal_entry.c . . . 65
2.4.6 Building and Testing the Project . . . 68
2.5 RA SC User Guide for MDK and IAR . . . 69
2.5.1 What is RA SC? . . . 69
2.5.2 Using RA Smart Configurator with Keil MDK . . . 69
2.5.2.1 Prerequisites . . . 69
2.5.2.2 Create new RA project . . . 70
2.5.2.3 Modify existing RA project . . . 73
2.5.2.4 Build and Debug RA project . . . 73
2.5.2.5 Notes and Restrictions . . . 74
2.5.3 Using RA Smart Configurator with IAR EWARM . . . 74
2.5.3.1 Prerequisites . . . 75
2.5.3.2 Create new RA project . . . 75
**Chapter 3 FSP Architecture** . . . 77
3.1 FSP Architecture Overview . . . 77
3.1.1 C99 Use . . . 77
3.1.2 Doxygen . . . 77
3.1.3 Weak Symbols . . . 77
3.1.4 Memory Allocation . . . 77
3.1.5 FSP Terms . . . 77
3.2 FSP Modules . . . 79
3.3 FSP Stacks . . . 80
3.4 FSP Interfaces . . . 81
3.4.1 FSP Interface Enumerations . . . 81
3.4.2 FSP Interface Callback Functions . . . 81
3.4.3 FSP Interface Data Structures . . . 84
3.4.3.1 FSP Interface Configuration Structure . . . 84
3.4.3.2 FSP Interface API Structure . . . 84
3.4.3.3 FSP Interface Instance Structure . . . 87
3.5 FSP Instances . . . 88
3.5.1 FSP Instance Control Structure . . . 88
3.5.2 FSP Interface Extensions . . . 89
3.5.2.1 FSP Extended Configuration Structure . . . 89

3.5.3 FSP Instance API . . . 89
3.6 FSP API Standards . . . 89
3.6.1 FSP Function Names . . . 89
3.6.2 Use of const in API parameters . . . 90
3.6.3 FSP Version Information . . . 90
3.7 FSP Build Time Configurations . . . 91
3.8 FSP File Structure . . . 91
3.9 FSP Architecture in Practice . . . 92
3.9.1 FSP Connecting Layers . . . 92
3.9.2 Using FSP Modules in an Application . . . 92
3.9.2.1 Create a Module Instance in the RA Configuration Editor . . . 92
3.9.2.2 Use the Instance API in the Application . . . 93
**Chapter 4 API Reference** . . . 94
4.1 BSP . . . 94
4.1.1 Common Error Codes . . . 96
4.1.2 MCU Board Support Package . . . 106
4.1.2.1 RA2A1 . . . 134
4.1.2.2 RA4M1 . . . 139
4.1.2.3 RA6M1 . . . 143
4.1.2.4 RA6M2 . . . 147
4.1.2.5 RA6M3 . . . 151
4.1.3 BSP I/O access . . . 155
4.2 Modules . . . 166
4.2.1 High-Speed Analog Comparator (r_acmphs) . . . 174
4.2.2 Low-Power Analog Comparator (r_acmplp) . . . 181
4.2.3 Analog to Digital Converter (r_adc) . . . 189
4.2.4 Asynchronous General Purpose Timer (r_agt) . . . 214
4.2.5 Bluetooth Low Energy Library (r_ble) . . . 239
4.2.5.2 GATT_COMMON . . . 245
4.2.6 Clock Frequency Accuracy Measurement Circuit (r_cac) . . . 246
4.2.7 Controller Area Network (r_can) . . . 252
4.2.8 Clock Generation Circuit (r_cgc) . . . 274
4.2.9 Cyclic Redundancy Check (CRC) Calculator (r_crc) . . . 294
4.2.10 Capacitive Touch Sensing Unit (r_ctsu) . . . 301
4.2.11 Digital to Analog Converter (r_dac) . . . 316
4.2.12 Digital to Analog Converter (r_dac8) . . . 322
4.2.13 Direct Memory Access Controller (r_dmac) . . . 328
4.2.14 Data Operation Circuit (r_doc) . . . 341
4.2.15 D/AVE 2D Port Interface (r_drw) . . . 347
4.2.16 Data Transfer Controller (r_dtc) . . . 349
4.2.17 Event Link Controller (r_elc) . . . 361
4.2.18 Ethernet (r_ether) . . . 369
4.2.19 Ethernet PHY (r_ether_phy) . . . 385
4.2.20 High-Performance Flash Driver (r_flash_hp) . . . 391
4.2.21 Low-Power Flash Driver (r_flash_lp) . . . 410
4.2.22 Graphics LCD Controller (r_glcdc) . . . 427
4.2.23 General PWM Timer (r_gpt) . . . 462
4.2.24 General PWM Timer Three-Phase Motor Control Driver (r_gpt_three_phase) . . . 500
4.2.25 Interrupt Controller Unit (r_icu) . . . 508
4.2.26 I2C Master on IIC (r_iic_master) . . . 514
4.2.27 I2C Slave on IIC (r_iic_slave) . . . 526
4.2.28 I/O Ports (r_ioport) . . . 537
4.2.29 Independent Watchdog Timer (r_iwdt) . . . 558
4.2.30 JPEG Codec (r_jpeg) . . . 567

4.2.31 Key Interrupt (r_kint) . . . . . 594
4.2.32 Low Power Modes (r_lpm) . . . . . 599
4.2.33 Low Voltage Detection (r_lvd) . . . . . 609
4.2.34 Operational Amplifier (r_opamp) . . . . . 616
4.2.35 Port Output Enable for GPT (r_poeg) . . . . . 635
4.2.36 Quad Serial Peripheral Interface Flash (r_qspi) . . . . . 641
4.2.37 Realtime Clock (r_rtc) . . . . . 659
4.2.38 Serial Communications Interface (SCI) I2C (r_sci_i2c) . . . . . 670
4.2.39 Serial Communications Interface (SCI) SPI (r_sci_spi) . . . . . 681
4.2.40 Serial Communications Interface (SCI) UART (r_sci_uart) . . . . . 692
4.2.41 Sigma Delta Analog to Digital Converter (r_sdadc) . . . . . 708
4.2.42 SD/MMC Host Interface (r_sdhi) . . . . . 731
4.2.43 Segment LCD Controller (r_slcdc) . . . . . 746
4.2.44 Serial Peripheral Interface (r_spi) . . . . . 754
4.2.45 Serial Sound Interface (r_ssi) . . . . . 772
4.2.46 USB (r_usb_basic) . . . . . 787
4.2.47 USB Host Communications Device Class Driver (r_usb_hcdc) . . . . . 814
4.2.48 USB Host Human Interface Device Class Driver (r_usb_hhid) . . . . . 831
4.2.49 USB Host Mass Storage Class Driver (r_usb_hmsc) . . . . . 841
4.2.50 USB Peripheral Communication Device Class (r_usb_pcdc) . . . . . 848
4.2.51 USB Peripheral Human Interface Device Class (r_usb_phid) . . . . . 858
4.2.52 USB Peripheral Mass Storage Class (r_usb_pmsc) . . . . . 873
4.2.53 Watchdog Timer (r_wdt) . . . . . 879
4.2.54 AWS PKCS11 PAL (rm_aws_pkcs11_pal) . . . . . 890
4.2.55 AWS PKCS11 PAL LITTLEFS (rm_aws_pkcs11_pal_littlefs) . . . . . 891
4.2.56 Bluetooth Low Energy Abstraction (rm_ble_abs) . . . . . 891
4.2.57 SD/MMC Block Media Implementation (rm_block_media_sdmmc) . . . . . 918
4.2.58 USB HMSC Block Media Implementation (rm_block_media_usb) . . . . . 925
4.2.59 SEGGER emWin Port (rm_emwin_port) . . . . . 932
4.2.60 FreeRTOS+FAT Port (rm_freertos_plus_fat) . . . . . 939
4.2.61 FreeRTOS Plus TCP (rm_freertos_plus_tcp) . . . . . 951
4.2.62 FreeRTOS Port (rm_freertos_port) . . . . . 957
4.2.63 LittleFS Flash Port (rm_littlefs_flash) . . . . . 984
4.2.64 Crypto Middleware (rm_psa_crypto) . . . . . 992
4.2.65 Capacitive Touch Middleware (rm_touch) . . . . . 1029
4.2.66 AWS Device Provisioning . . . . . 1038
4.2.67 AWS MQTT . . . . . 1042
4.2.68 Wifi Middleware (rm_wifi_onchip_silex) . . . . . 1047
4.2.69 AWS Secure Sockets . . . . . 1077
4.3 Interfaces . . . . . 1083
4.3.1 ADC Interface . . . . . 1088
4.3.2 BLE Interface . . . . . 1101
4.3.3 CAC Interface . . . . . 1103
4.3.4 CAN Interface . . . . . 1112
4.3.5 CGC Interface . . . . . 1127
4.3.6 Comparator Interface . . . . . 1140
4.3.7 CRC Interface . . . . . 1149
4.3.8 CTSU Interface . . . . . 1156
4.3.9 DAC Interface . . . . . 1168
4.3.10 Display Interface . . . . . 1173
4.3.11 DOC Interface . . . . . 1191
4.3.12 ELC Interface . . . . . 1196
4.3.13 Ethernet Interface . . . . . 1201
4.3.14 Ethernet PHY Interface . . . . . 1210

4.3.15 External IRQ Interface . . . . . 1216
4.3.16 Flash Interface . . . . . 1222
4.3.17 I2C Master Interface . . . . . 1238
4.3.18 I2C Slave Interface . . . . . 1246
4.3.19 I2S Interface . . . . . 1253
4.3.20 I/O Port Interface . . . . . 1265
4.3.21 JPEG Codec Interface . . . . . 1279
4.3.22 Key Matrix Interface . . . . . 1294
4.3.23 Low Power Modes Interface . . . . . 1299
4.3.24 Low Voltage Detection Interface . . . . . 1313
4.3.25 OPAMP Interface . . . . . 1323
4.3.26 POEG Interface . . . . . 1329
4.3.27 RTC Interface . . . . . 1337
4.3.28 SD/MMC Interface . . . . . 1348
4.3.29 SLCDC Interface . . . . . 1364
4.3.30 SPI Interface . . . . . 1375
4.3.31 SPI Flash Interface . . . . . 1386
4.3.32 Three-Phase Interface . . . . . 1397
4.3.33 Timer Interface . . . . . 1403
4.3.34 Transfer Interface . . . . . 1415
4.3.35 UART Interface . . . . . 1427
4.3.36 USB Interface . . . . . 1437
4.3.37 USB HCDC Interface . . . . . 1465
4.3.38 USB HHID Interface . . . . . 1465
4.3.39 USB HMSC Interface . . . . . 1467
4.3.40 USB PCDC Interface . . . . . 1473
4.3.41 USB PHID Interface . . . . . 1474
4.3.42 USB PMSC Interface . . . . . 1474
4.3.43 WDT Interface . . . . . 1474
4.3.44 BLE ABS Interface . . . . . 1484
4.3.45 Block Media Interface . . . . . 1500
4.3.46 FreeRTOS+FAT Port Interface . . . . . 1508
4.3.47 LittleFS Interface . . . . . 1514
4.3.48 Touch Middleware Interface . . . . . 1517
**Chapter 5 Copyright** . . . . . 1523

# Chapter 1 Introduction

## 1.1 Overview

This manual describes how to use the Renesas Flexible Software Package (FSP) for writing applications for the RA microcontroller series.

## 1.2 How to Read this Manual

For help getting started with the FSP, see:

- Starting Development

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

- FSP Architecture
- MCU Board Support Package

For user guides describing the FSP modules, see:

- Modules

For shared interface API documentation, see:

- Interfaces

## 1.3 Documentation Standard

Each module user guide outlines the following:

- Features: A bullet list of high level features provided by the module.
- Configuration: A description of module specific configurations available in the RA Configuration editor.
- Usage Notes: Module specific documentation and limitations.
- Examples: Example code provided to help the user get started.
- API Reference: Usage notes for each API in the module, including the function prototype and hyperlinks to the interface documentation for parameter definitions.

Interface documentation includes typed enumerations and structures–including a structure of function pointers that defines the API–that are shared by all modules that implement the interface.

## Introduction to FSP

### Purpose

The Renesas Flexible Software Package (FSP) is an optimized software package designed to provide easy to use, scalable, high quality software for embedded system design. The primary goal is to

provide lightweight, efficient drivers that meet common use cases in embedded systems.

### Quality

FSP code quality is enforced by peer reviews, automated requirements-based testing, and automated static analysis.

### Ease of Use

The FSP provides uniform and intuitive APIs that are well documented. Each module is supported with detailed user documentation including example code.

### Scalability

FSP modules can be used on any MCU in the RA family, provided the MCU has any peripherals required by the module.

FSP modules also have build time configurations that can be used to optimize the size of the module for the feature set required by the application.

# Chapter 2 Starting Development

## 2.1 Starting Development Introduction

The Renesas Flexible Software Package (FSP) provides a host of efficiency enhancing tools for developing projects targeting the Renesas RA series of MCU devices. e2 studio provides a familiar development cockpit from which the key steps of project creation, module selection and configuration, code development, code generation, and debugging are all managed. FSP runs within e2 studio and enables the module selection, configuration, and code generation steps. FSP uses a Graphical User Interface (GUI) to simplify the selection, configuration, code generation and code development of high level modules and their associated Application Program Interfaces (APIs) to dramatically accelerate the development process.

The wealth of resources available to learn about and use e2 studio and FSP can be overwhelming on first inspection, so the following section provides a Getting Started Guide with a list of the most important first steps. Following these highly recommended first 10 steps will bring you up to speed on the development environment in record time. Even experienced developers can benefit from the use of this guide, to learn the terminology that might be unfamiliar or different from previous environments.

### 2.1.1 Getting Started with e2 studio and FSP

This section describes how to use Renesas e2 studio to develop applications with the Renesas Flexible Software Package (FSP). Here is the recommended sequence for quickly Getting Started with using e2 when developing with the RA MCU Family:

1. Read over the section What is e2 studio?, up to but not including e2 studio Prerequisites. This will provide a description of the various windows and views to use e2 to create a project, add modules and threads, configure module properties, add code, and debug a project. It also describes how to use key coding 'accelerators' like Developer Assist (to drag and drop parameter populated API function calls right into your code), a context aware Autocomplete (to easily find and select from suggested enumerations, functions, types, and many other coding elements), and many other similar productivity enhancers.
2. Read over the FSP Architecture sections FSP Architecture, FSP Modules and FSP Stacks. These provide the basic background on how FSP modules and stacks are used to construct your application. Understanding their definitions and the theory behind how they combine will make it easier to develop with FSP.
3. Read over a few "API Reference" sections to see how to use API function calls, structures, enumerations, types and callbacks. These user guides provide the information you will use to implement your project code. (Much of the details are provided with Developer Assistance, covered in step 5, below.
4. If you don't have a kit. you can order one using the link included in the e2 studio Prerequisites section. Then, if you haven't yet downloaded and installed e2 studio and FSP, use the link included in the e2 studio Prerequisites section to download the tools. Then you can build and debug a simple project to prove out you installation, tool flow, and the kit. The simple "Blinky" project, that blinks an LED on and off, is located in the Tutorial: Your First RA MCU Project - Blinky section. Follow the instructions for importing and running this project. It will use some of the key steps for managing projects within e2 and is a good way to learn the basics.

5. Once you have successfully run Blinky you have a good starting point for using FSP for more complex projects. The Watchdog Timer hands-on lab, available in the Tutorial: Using HAL Drivers - Programming the WDT section, shows how to create a project from scratch and use FSP API functions, and demonstrates the use of some of the coding efficiency tools like Developer Assistance and Autocomplete. Run through this lab to establish a good starting point for developing custom projects.
6. The balance of the FSP Architecture sections, those not called out in step 2 above, contain additional reference material that may be helpful in the future. Scan them over so you know what they contain, in case you need them.
7. The balance of the e2 studio User Guide, starting with the What is a Project? section up to Writing the Application section, provides a detailed description of each of the key steps, windows, and entries used to create, manage, configure, build and debug a project. Most of this will be familiar after doing the Blinky and WDT exercises from steps 4 and 5 above. Skim over these references so you know to come back to them when questions come up. Make sure you have a good grasp of what each of the configuration tabs are used for since that is where the bulk of the project preparation work takes place prior to writing code.
8. Read over the Writing the Application section to get a short introduction to the steps used when creating application code with FSP. It covers both RTOS-independent and RTOS-dependent applications. The Tutorial: Using HAL Drivers - Programming the WDT section is a good introduction to the key steps for an RTOS-independent application. Make sure you have run through it at least once before doing a custom project.
9. Scan the Debugging the Project section to see the steps required to download and start a debug session.
10. Explore the additional material available on the following web pages and bookmark the resources that look most valuable to you:
    a. RA Landing Page: https://www.renesas.com/ra
    b. FSP Landing Page: https://www.renesas.com/fsp

# 2.2 e2 studio User Guide

## 2.2.1 What is e2 studio?

Renesas e2 studio is a development tool encompassing code development, build, and debug. e2 studio is based on the open-source Eclipse IDE and the associated C/C++ Development Tooling (CDT).

When developing for RA MCUs, e2 studio hosts the Renesas Flexible Software Package (FSP). FSP provides a wide range of time saving tools to simplify the selection, configuration, and management of modules and threads, to easily implement complex applications. The time saving tools available in e2 studio and FSP include the following:

- A Graphical User Interface (GUI) (see Adding Threads and Drivers) with numerous wizards for configuring and auto-generating code
- A context sensitive Autocomplete (see Tutorial: Using HAL Drivers - Programming the WDT) feature that provides intelligent options for completing a programming element
- A Developer Assistance tool for selection of and drag and drop placement of API functions directly in application code
- A Smart Manual provides driver and device documentation in the form of tooltips right in the code
- An Edit Hover feature to show detailed descriptions of code elements while editing
- A Welcome Window with links to example projects, application notes and a variety of other self-help support resources
- An Information Icon from each module is provided in the graphic configuration viewer that links to specific design resources, including code 'cheat sheets' that provide useful starting

points for common application implementations.

Figure 1: e2 studio Splash Screen

e2 studio organizes project work based on Perspectives, Views, Windows, Panes, and Pages (sometimes called Tabs). A window is a section of the e2 studio GUI that presents information on a key topic. Windows often use tabs to select sub-topics. For example, an editor window might have a tab available for each open file, so it is easy to switch back and forth between them. A window Pane is a section of a window. Within a window, multiple Panes can be opened and viewed simultaneously, as opposed to a tabbed window, where only individual content is displayed. A memory-display Window, for example, might have multiple Panes that allow the data to be displayed in different formats, simultaneously. A Perspective is a collection of Views and Windows typical for a specific stage of development. The default perspectives are a C/C++ Perspective, an FSP Configuration Perspective and a Debug Perspective. These provide specific Views, Windows, Tabs, and Panes tailored for the common tasks needed during the specific development stage. These three default perspectives are each illustrated in the below screen shots, along with graphic indicators helpful in identifying example Views, Windows, Tabs and Panes.

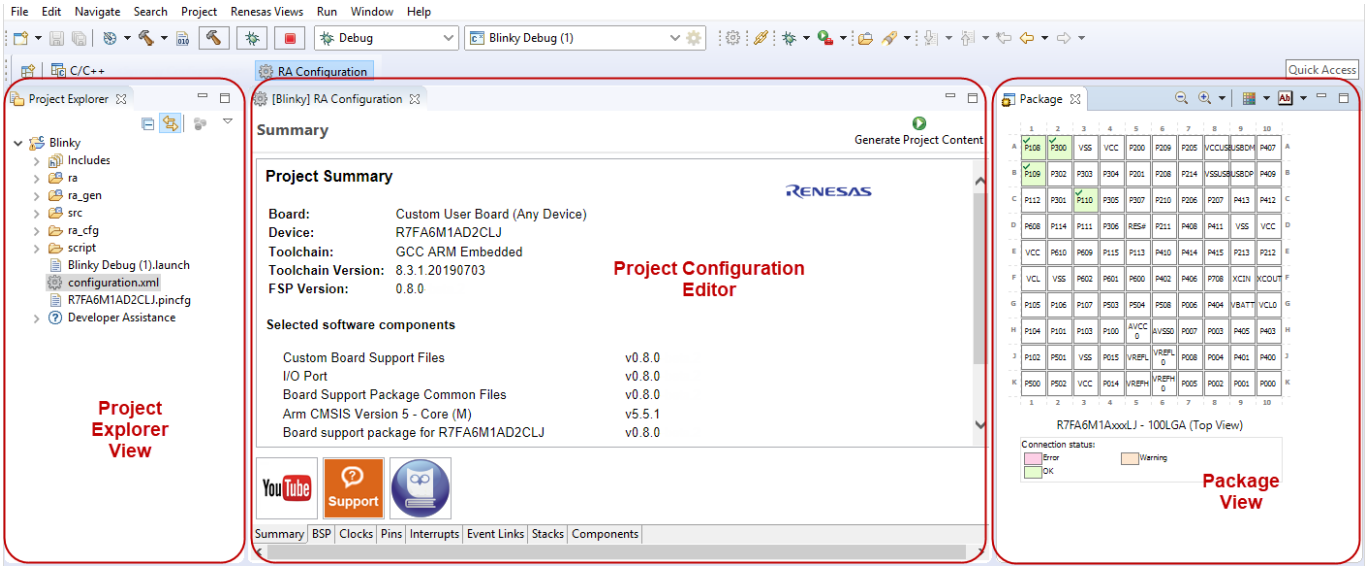



Figure 2: Default Perspective

In addition to managing project development, selecting modules, configuring them and simplifying

code development, e2 studio also hosts the engine for automatically generating code based on module selections and configurations. The engine continually checks for dependencies and automatically adds any needed lower level modules to the module stack. It also identifies any lower level modules that require configuration (for example, an interrupt that needs to have a priority assigned). It also provides a guide for selecting between multiple choices or options to make it easy to complete a fully functional module stack.

The Generate Project Content function takes the selected and configured modules and automatically generates the complete and correct configuration code. The code is added to the folders visible in the **Project Explorer** window in e2 studio. The configuration.xml file in the project folder holds all the generated configuration settings. This file can be opened in the GUI-based RA Configuration editor to make further edits and changes. Once a project has been generated, you can go back and reconfigure any of the modules and settings if required using this editor.

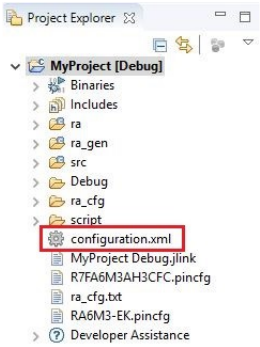



Figure 3: Project Explorer Window showing generated folders and configuration.xml file

## 2.2.2 e2 studio Prerequisites

### 2.2.2.1 Obtaining an RA MCU Kit

To develop applications with FSP, start with one of the Renesas RA MCU Evaluation Kits. The Renesas RA MCU Evaluation Kits are designed to seamlessly integrate with the e2 studio.

Ordering information, Quick Start Guides, User Manuals, and other related documents for all RA MCU Evaluation Kits are available at https://www.renesas.com/ra.

### 2.2.2.2 PC Requirements

The following are the minimum PC requirements to use e2 studio:

- Windows 10 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

### 2.2.2.3 Installing e2 studio, platform installer and the FSP package

Detailed installation instructions for the e2 studio and the FSP are available on the Renesas website https://www.renesas.com/fsp. Review the release notes for e2 studio to ensure that the e2 studio version supports the selected FSP version. The starting version of the installer includes all features of the RA MCUs.

### 2.2.2.4 Choosing a Toolchain

e2 studio can work with several toolchains and toolchain versions such as the GNU ARM compiler, AC6. A version of the GNU ARM compiler is included in the e2 studio installer and has been verified to run with the FSP version.

#### 2.2.2.5 Licensing

FSP licensing includes full source code, limited to Renesas hardware only.

### 2.2.3 What is a Project?

In e2 studio, all FSP applications are organized in RA MCU projects. Setting up an RA MCU project involves:

1. Creating a Project
2. Configuring a Project

These steps are described in detail in the next two sections. When you have existing projects already, after you launch e2 studio and select a workspace, all projects previously saved in the selected workspace are loaded and displayed in the **Project Explorer** window. Each project has an associated configuration file named configuration.xml, which is located in the project's root directory.

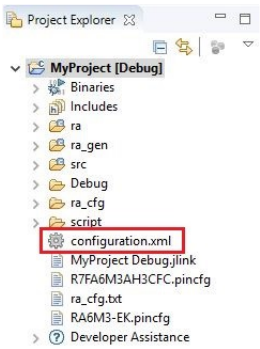



Figure 4: e2 studio Project Configuration file

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

Figure 5: e2 studio RA Configuration Perspective

*Note*

*Whenever the RA project configuration (that is, the configuration.xml file) is saved, a verbose RA Project Report file (ra_cfg.txt) with all the project settings is generated. The format allows differences to be easily viewed using a text comparison tool. The generated file is located in the project root directory.*

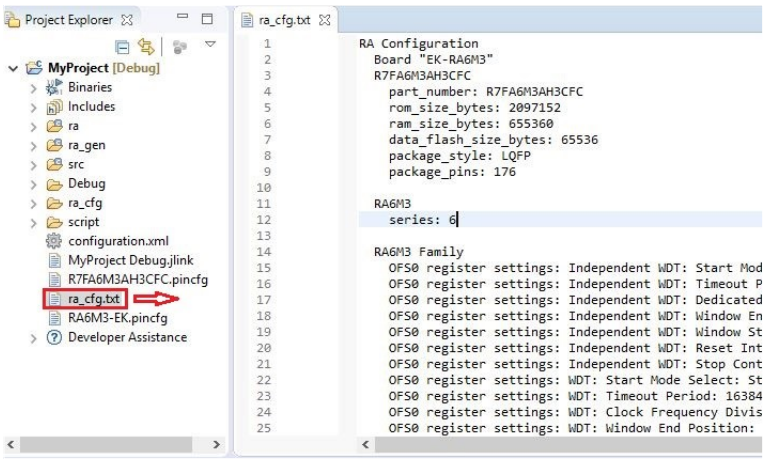



Figure 6: RA Project Report

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

*Note*

*The tabs available in the RA Project Editor depend on the e2 studio version.*

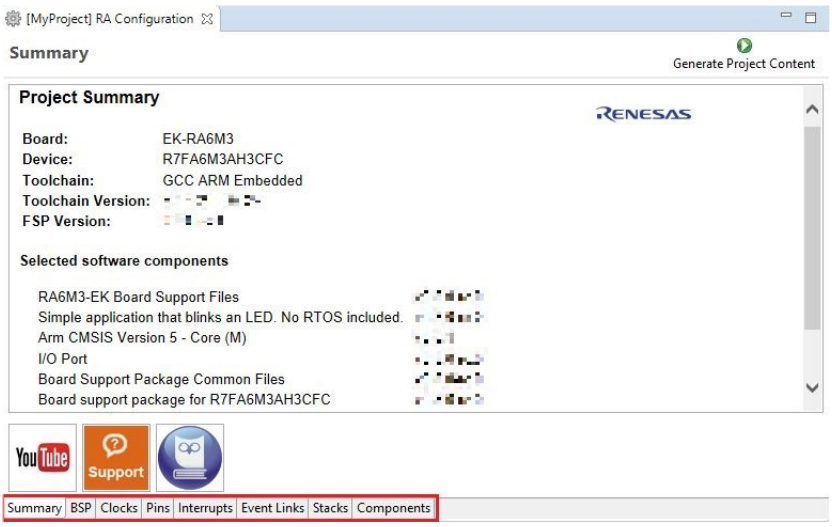



Figure 7: RA Project Summary tabs

- Click on the YouTube icon to visit the Renesas FSP playlist on YouTube
- Click on the Support icon to visit RA support pages at Renesas.com
- Click on the user manual (owl) icon to open the RA software package User's Manual

## 2.2.4 Creating a Project

During project creation, you specify the type of project, give it a project name and location, and configure the project settings for version, target board, whether an RTOS is included, the toolchain version, and the beginning template. This section includes easy-to-follow step-by-step instructions for all of the project creation tasks. Once you have created the project, you can move to configuring the project hardware (clocks, pins, interrupts) and the parameters of all the modules that are part of

your application.

### 2.2.4.1 Creating a New Project

For RA MCU applications, generate a new project using the following steps:

1. Click on **File > New > RA C/C++ Project**.

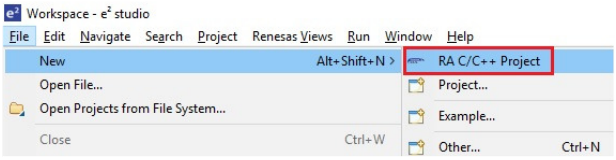



Figure 8: New RA MCU Project

Then click on the type of template for the type of project you are creating.

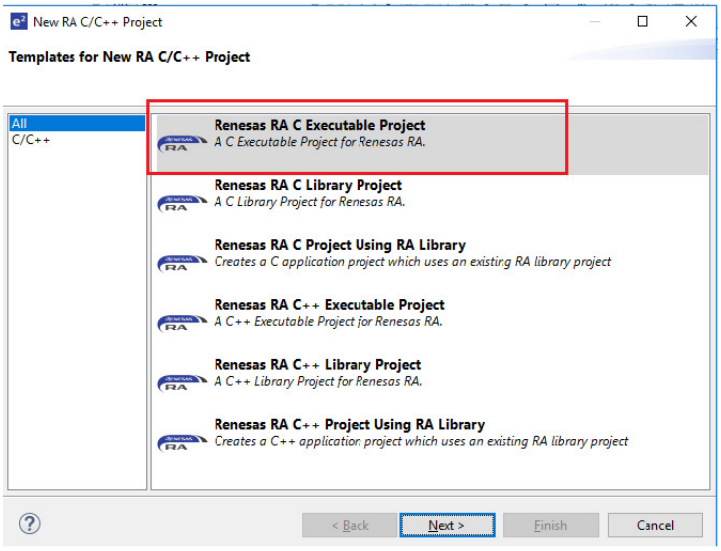



Figure 9: New Project Templates

2. Select a project name and location.

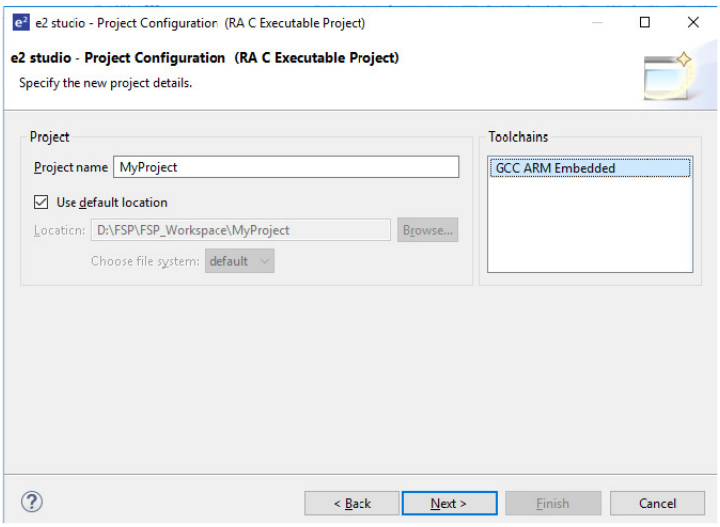



Figure 10: RA MCU Project Generator (Screen 1)

3. Click **Next**.

### 2.2.4.2 Selecting a Board and Toolchain

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

1. Select the **FSP version**.
2. Select the **Board** for your application. You can select an existing RA MCU Evaluation Kit or select **Custom User Board** for any of the RA MCU devices with your own BSP definition.
3. Select the **Device**. The **Device** is automatically populated based on the **Board** selection. Only change the **Device** when using the **Custom User Board (Any Device)** board selection.
4. To add threads, select **RTOS**, or **No RTOS** if an RTOS is not being used.
5. The **Toolchain** selection defaults to **GCC ARM Embedded**.
6. Select the **Toolchain version**. This should default to the installed toolchain version.
7. Select the **Debugger**. The J-Link ARM Debugger is preselected.

8. Click **Next**.

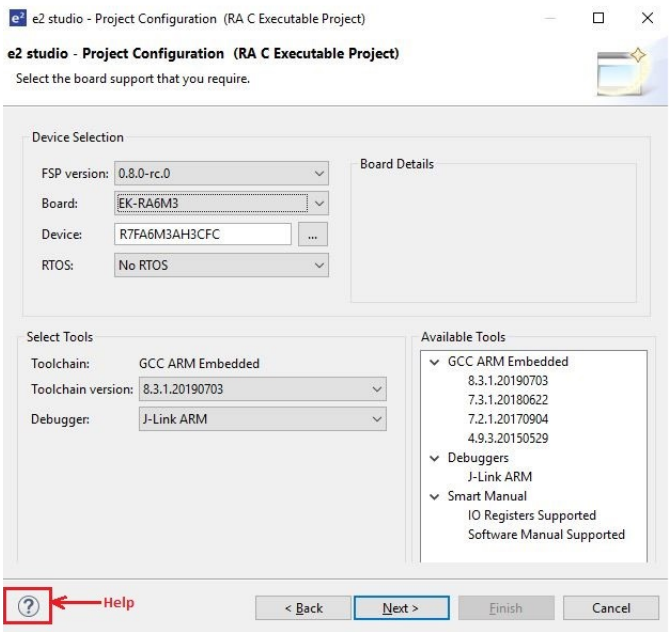



Figure 11: RA MCU Project Generator (Screen 2)

Click on the **Help** icon (?) for user guides, RA contents, and other documents.

### 2.2.4.3 Selecting a Project Template

In the next window, select a project template from the list of available templates. By default, this screen shows the templates that are included in your current RA MCU pack. Once you have selected the appropriate template, click **Finish**.

*Note*

*If you want to develop your own application, select the basic template for your board,* ***Bare Metal - Minimal****.*

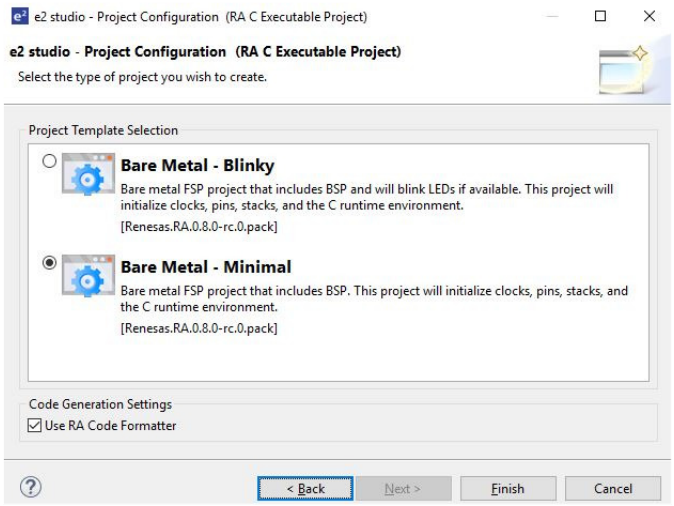



Figure 12: RA MCU Project Generator (Screen 3)

When the project is created, e2 studio displays a summary of the current project configuration in the

RA MCU Project Editor.

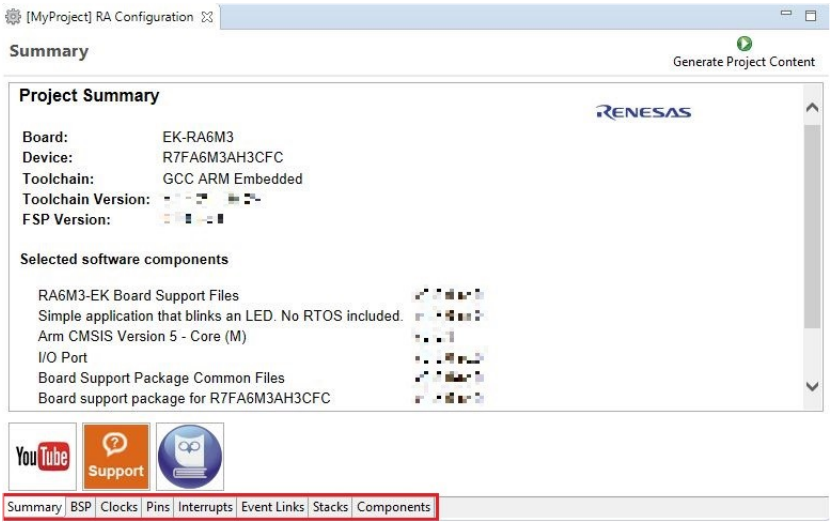



Figure 13: RA MCU Project Editor and available editor tabs

On the bottom of the RA MCU 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 **Stacks** tab, you can add FSP modules for non-RTOS applications and configure the modules. For each module selected in this tab, the **Properties** window provides access to the configuration parameters, interrupt priorities, and pin selections.
- With the **Interrupt** tab, you can add new user events/interrupts.
- With the **Event Links** tab, you can configure events used by the Event Link Controller.
- The **Components** tab provides an overview of the selected modules. You can also add drivers for specific FSP releases and application sample code here.

The functions and use of each of these tabs is explained in detail in the next section.

## 2.2.5 Configuring a Project

Each of the configurable elements in an FSP project can be edited using the appropriate tab in the RA Configuration editor window. Importantly, the initial configuration of the MCU after reset and before any user code is executed is set by the configuration settings in the **BSP**, **Clocks** and **Pins** tabs. When you select a project template during project creation, e2 studio configures default values that are appropriate for the associated board. You can change those default values as needed. The following sections detail the process of configuring each of the project elements for each of the associated tabs.

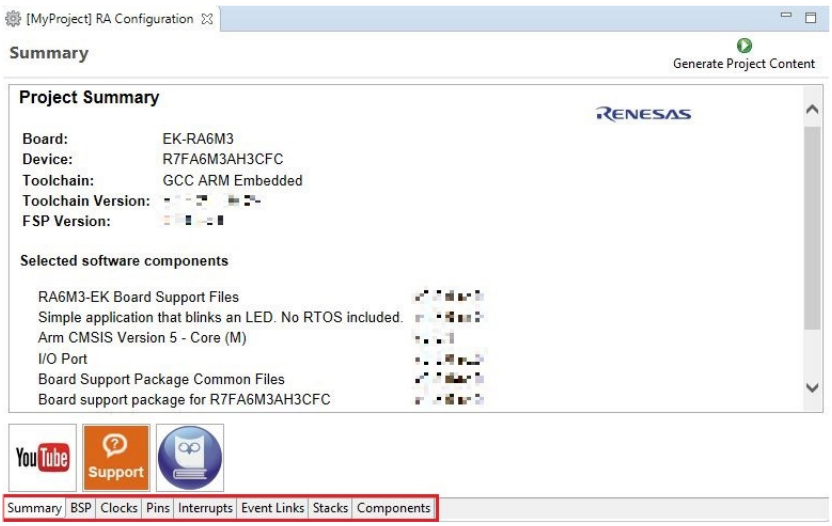



Figure 14: RA MCU Project Editor and available editor tabs

### 2.2.5.1 Configuring the BSP with e2 studio

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.*

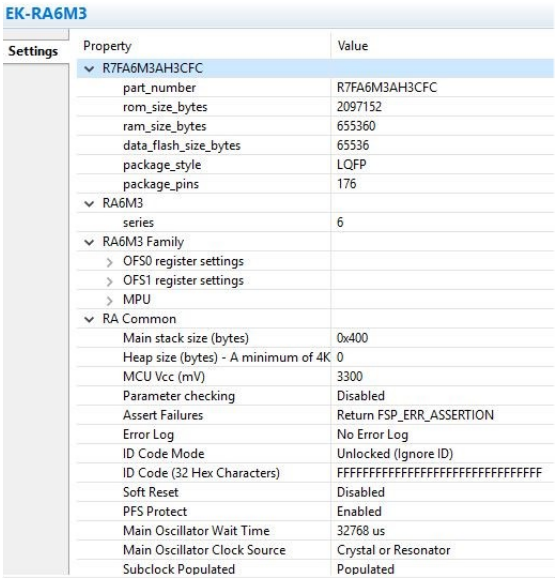



Figure 15: Configuration BSP tab

The **Properties** view shows the configurable options available for the BSP. These can be changed as required. The BSP is the FSP layer above the MCU hardware. e2 studio checks the entry fields to flag invalid entries. For example, only valid numeric values can be entered for the stack size.

When you click the **Generate Project Content** button, the BSP configuration contents are written to ra_cfg/fsp_cfg/bsp/bsp_cfg.h

This file is created if it does not already exist.

Warning
Do not edit this file as it is overwritten whenever the **Generate Project Content** button is clicked.

### 2.2.5.2 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 HOCO has been changed so the resulting clock frequency is 24 MHz instead of the required 48 MHz. This parameter is colored red.

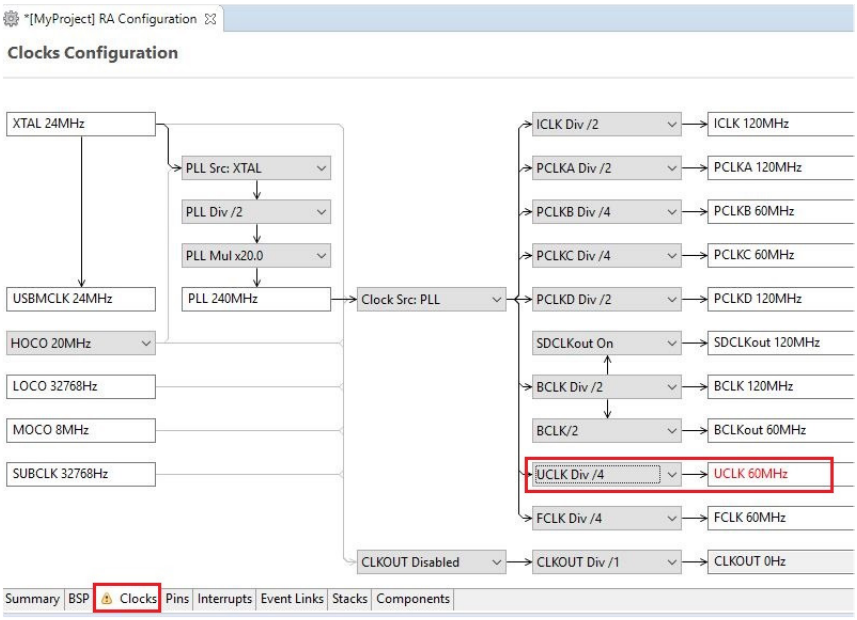



Figure 16: Configuration Clocks tab

When you click the **Generate Project Content** button, the clock configuration contents are written to: ra_gen/bsp_clock_cfg.h

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

Warning
Do not edit this file as it is overwritten whenever the **Generate Project Content** button is clicked.

### 2.2.5.3 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 e2 studio, 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 RA6M3, some peripherals connected on the board are preselected.

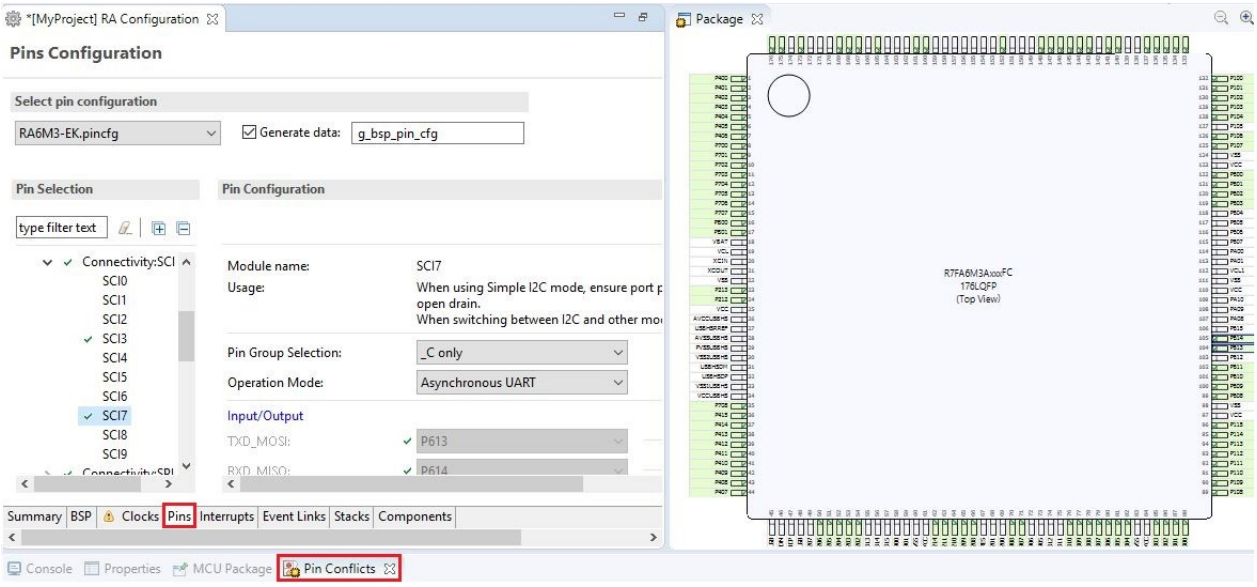



Figure 17: Pins Configuration

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 P611 is already used by the CAC, 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 CAC in the **Pin Selection** pane on the left side of the tab.

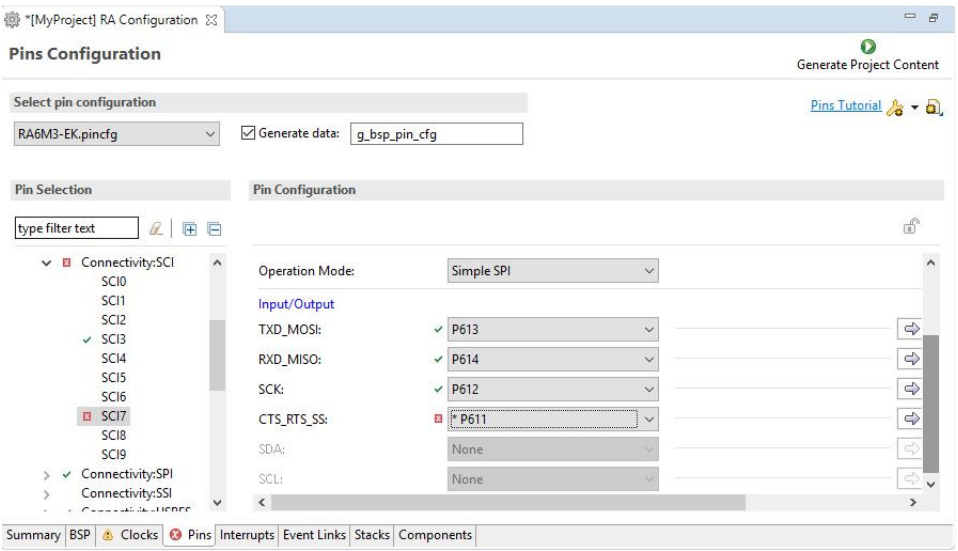



Figure 18: e2 studio Pin configurator

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

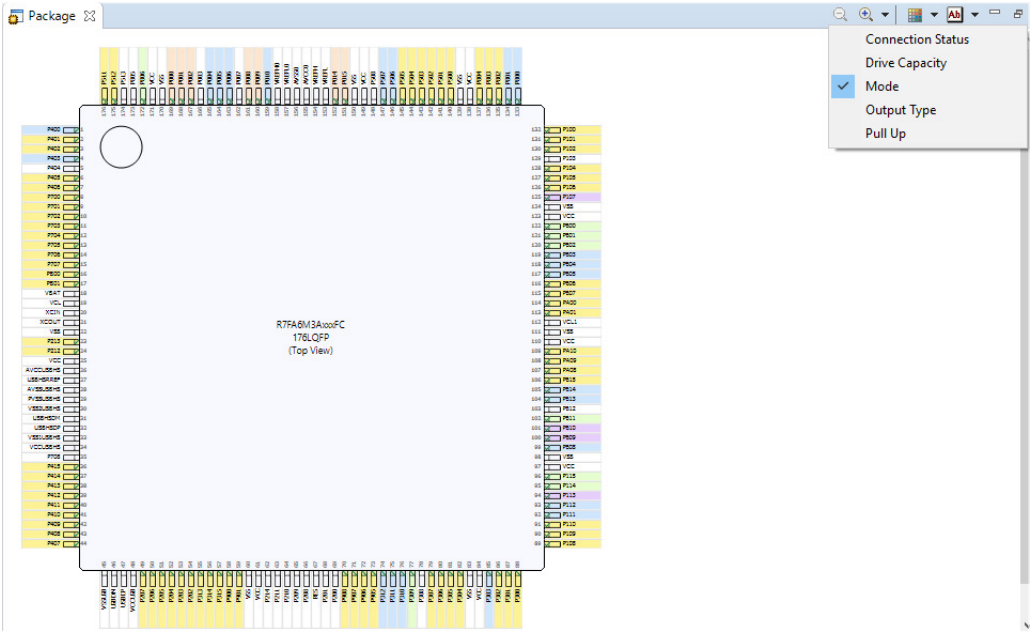



Figure 19: e2 studio Pin configurator package view

When you click the **Generate Project Content** button, the pin configuration contents are written to: ra_gen\bsp_pin_cfg.h

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

Warning

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

To make it easy to share pinning information for your project, e2 studio exports your pin configuration settings to a csv format and copies the csv file to ra_gen/<MCU package>.csv.

### 2.2.5.4 Configuring Interrupts

You can use the **Properties** view in the **Stacks** tab to enable interrupts by setting the interrupt priority. Select the driver in the **Stacks** pane to view and edit its properties.

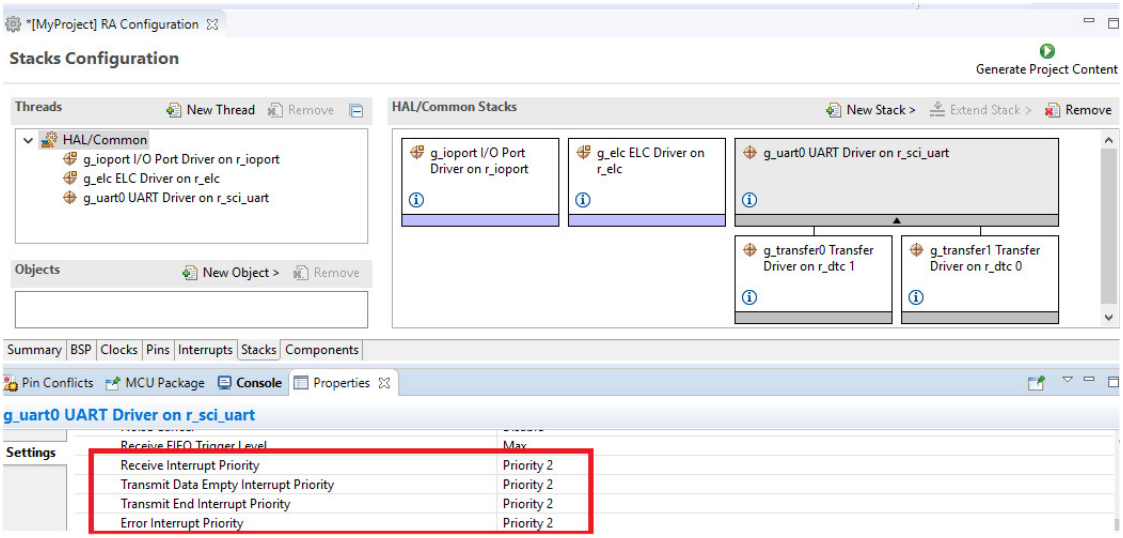



Figure 20: Configuring Interrupt on the Stacks tab

### Interrupts

In the **Interrupt** tab, the user can bypass a peripheral interrupt and have user-defined ISRs for the peripheral interrupt. This can be done by adding a new event with the user define tab (**New User Event**).

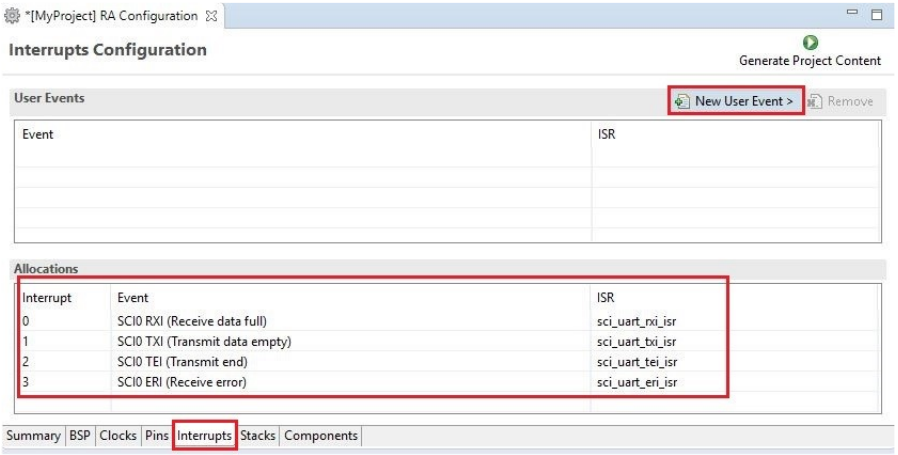



Figure 21: Configuring interrupt in Interrupt Tab

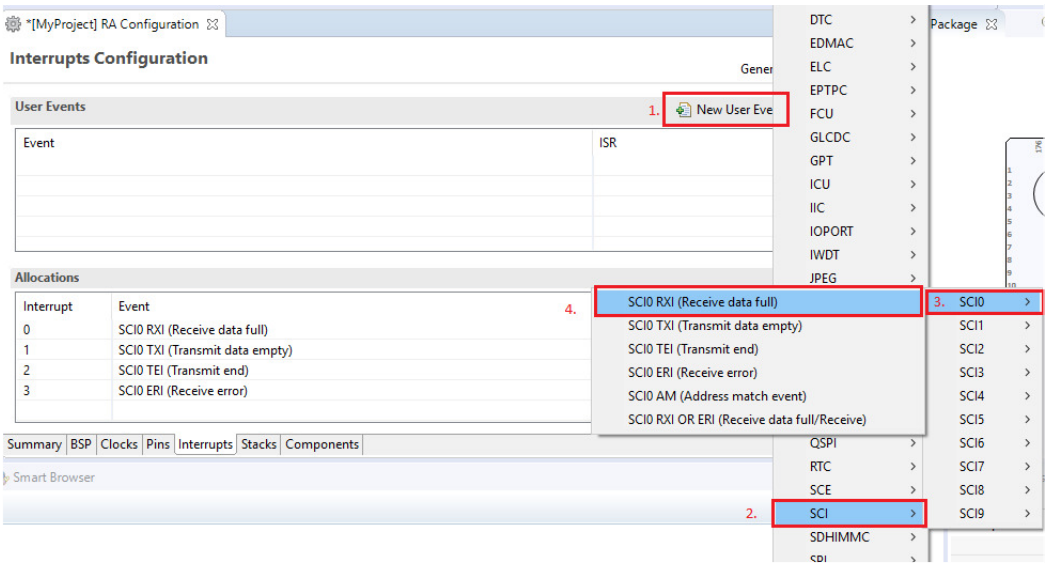



Figure 22: Adding user-defined event

Enter the name of ISR for the new user event.

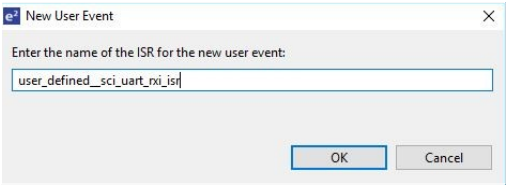



Figure 23: User-defined event ISR

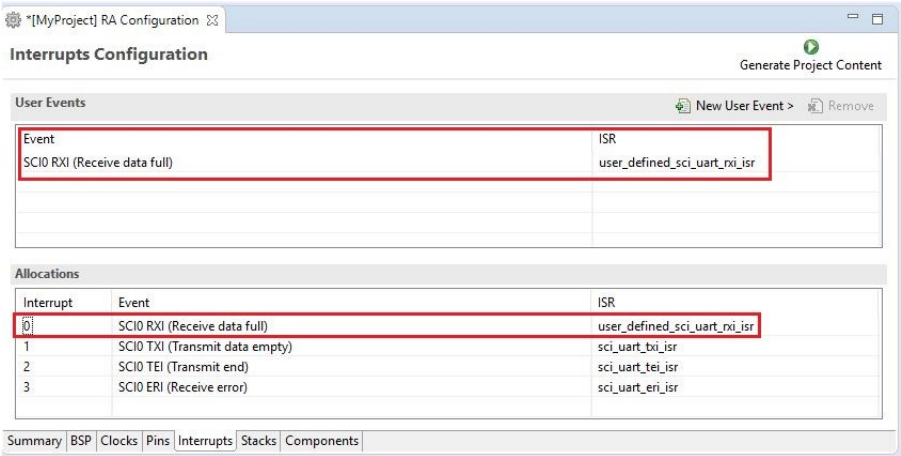



Figure 24: Using a user-defined event

### 2.2.5.5 Viewing Event Links

The Event Links tab can be used to view the Event Link Controller events. The events are sorted by peripheral to make it easy to find and verify them.

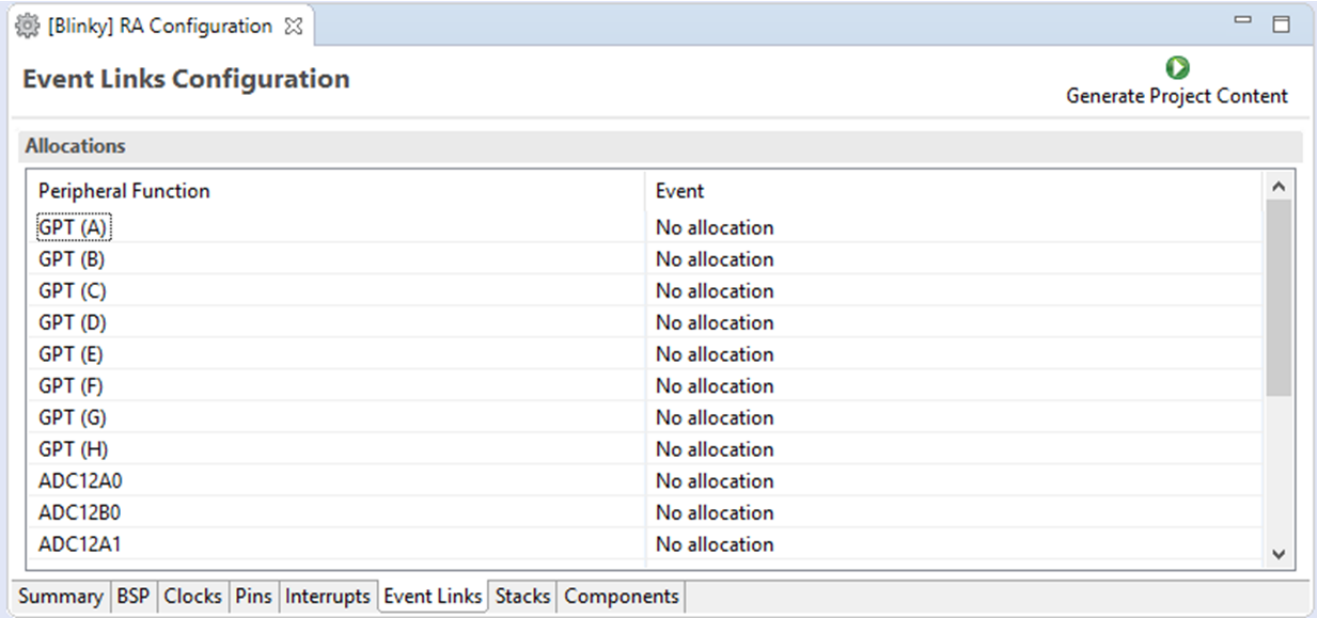



Figure 25: Viewing Event Links

Like the Interrupts tab, user-defined event sources and destinations (producers and consumers) can be defined by clicking the relevant **New User Event** button.

*Note*

*When selecting an ELC event to receive for a module (or when manually defining an event link), only the events that are made available by the modules configured in the project will be shown.*

## 2.2.6 Adding Threads and Drivers

Every FreeRTOS-based RA Project includes at least one RTOS Thread and a stack of FSP modules running in that thread. The **Stacks** 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, e2 studio automatically generates the code reflecting your configuration choices.

For any driver, or, more generally, any module that you add to a thread, e2 studio automatically resolves all dependencies with other modules and creates the appropriate stack. This stack is displayed in the Stacks pane, which e2 studio populates with the selected modules and module options for the selected thread.

The default view of the **Stacks** tab includes a Common Thread called **HAL/Common**. This thread includes the driver for I/O control (IOPORT). The default stack is shown in the **HAL/Common Stacks** pane. The default modules added to the HAL/Common driver are special in that the FSP only requires a single instance of each, which e2 studio 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 stacks, 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 FSP pack and can therefore change when the FSP version changes.*

### 2.2.6.1 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 **Stacks** pane. The Modules pane changes to **HAL/Common Stacks**.

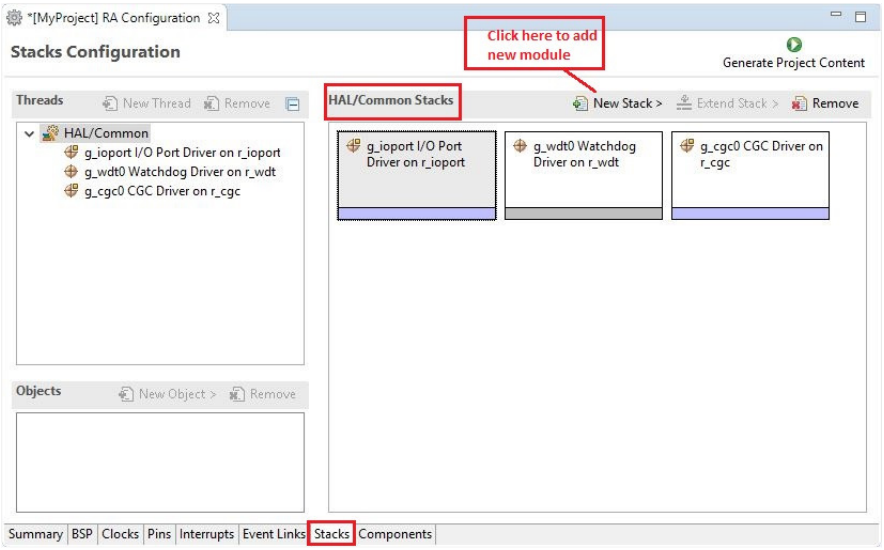



Figure 26: e2 studio Project configurator - Adding drivers

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

3. Select a driver from the menu **New Stack > Driver**.

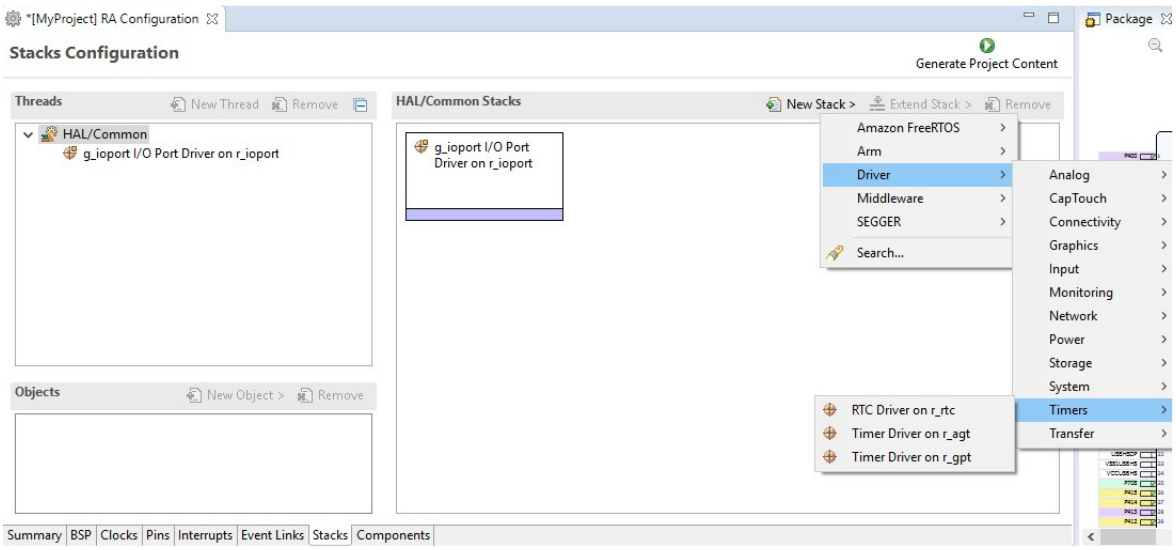



Figure 27: Select a driver

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

e2 studio adds the following files when you click the **Generate Project Content** button:

- The selected driver module and its files to the ra/fsp 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? |
|---|---|---|
| ra_gen/main.c | Contains main() calling generated and user code. When called, the BSP already has Initialized the MCU. | Yes |
| ra_gen/hal_data.c | Configuration structures for HAL Driver only modules. | Yes |
| ra_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: ra_cfg/fsp_cfg

### 2.2.6.2 Adding Drivers to a Thread and Configuring the Drivers

For an application that uses the 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 the Driver dropdown menu. To add modules to a thread, follow these steps:

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

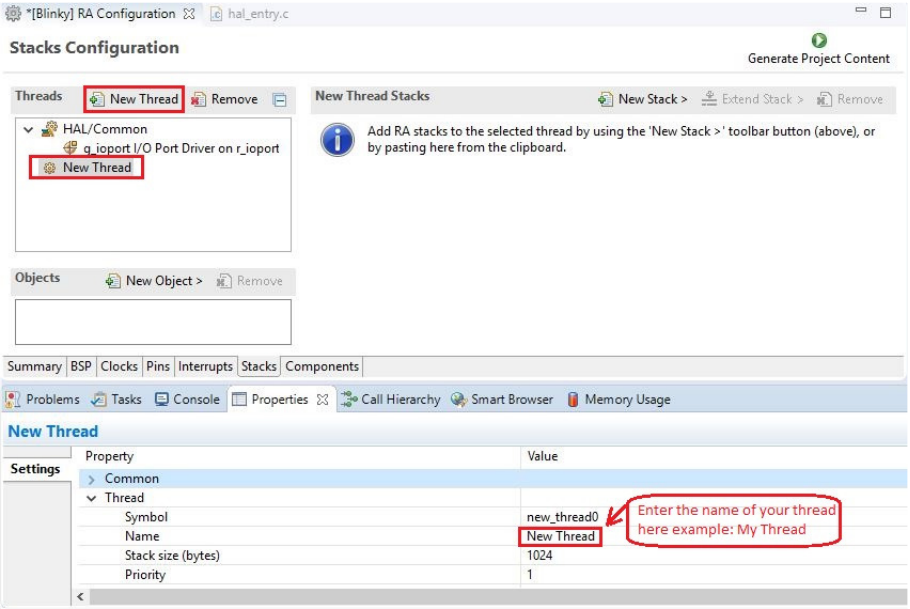



Figure 28: Adding a new RTOS Thread on the Stacks tab

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

   *Note*

   *e2 studio updates the name of the thread stacks pane to **My Thread Stacks**.*

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

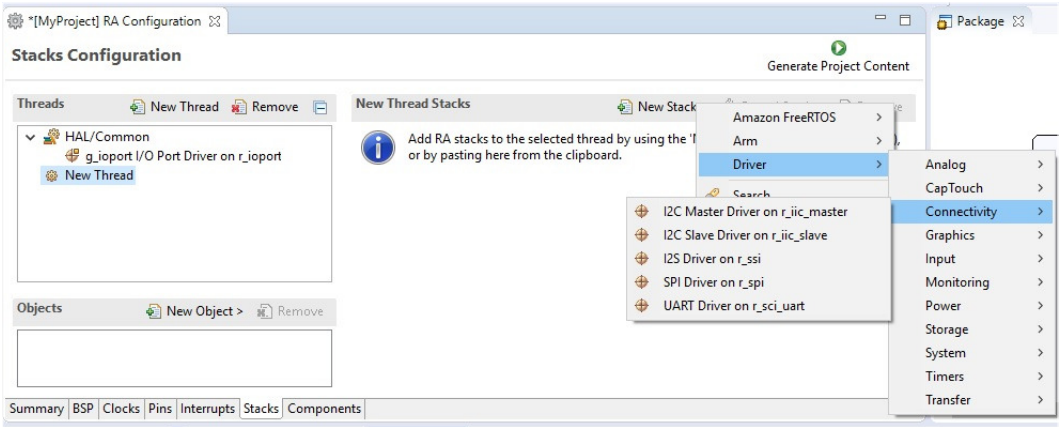



Figure 29: Adding Modules and Drivers to a thread

4. Select a module or driver from the list.

5. 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.

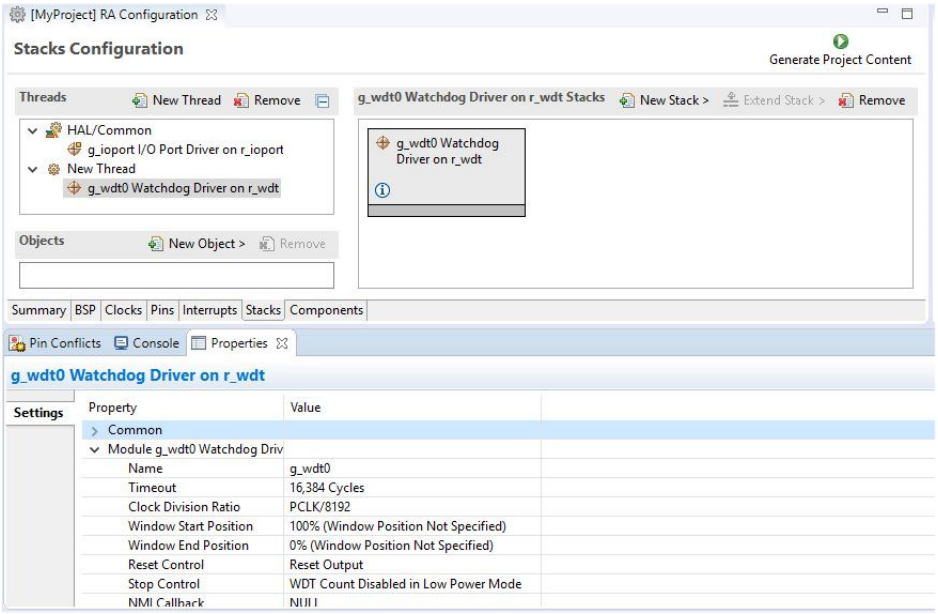



Figure 30: Configuring Module or Driver properties

6. If needed, add another thread by clicking **New Thread** in the **Threads** pane.

When you press the **Generate Project Content** button for the example above, e2 studio creates the files as shown in the following table:

| File | Contents | Overwritten by Generate Project Content? |
|---|---|---|
| ra_gen/main.c | Contains main() calling generated and user code. When called the BSP will have initialized the MCU. | Yes |
| ra_gen/my_thread.c | Generated thread "my_thread" and configuration structures for modules added to this thread. | Yes |
| ra_gen/my_thread.h | Header file for thread "my_thread" | Yes |
| ra_gen/hal_data.c | Configuration structures for HAL Driver only modules. | Yes |
| ra_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: ra_cfg/fsp_cfg/<header files>

### 2.2.6.3 Configuring Threads

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

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

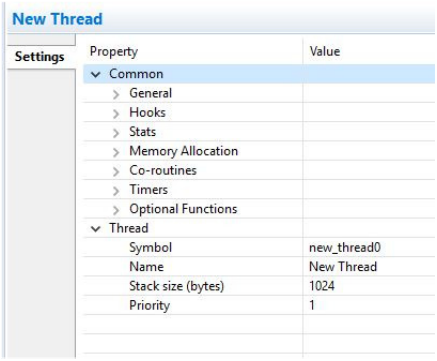



Figure 31: New Thread Properties

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. e2 studio checks that the entries in the property field are valid. For example, it will verify that the field **Priority**, which requires an integer value, only contains numeric values between 0 and 9.

To add FreeRTOS 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**.

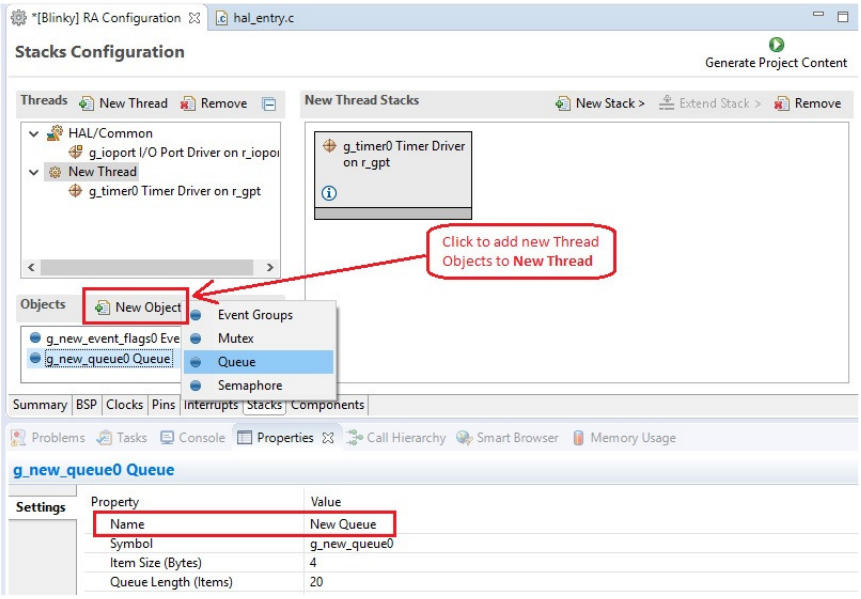
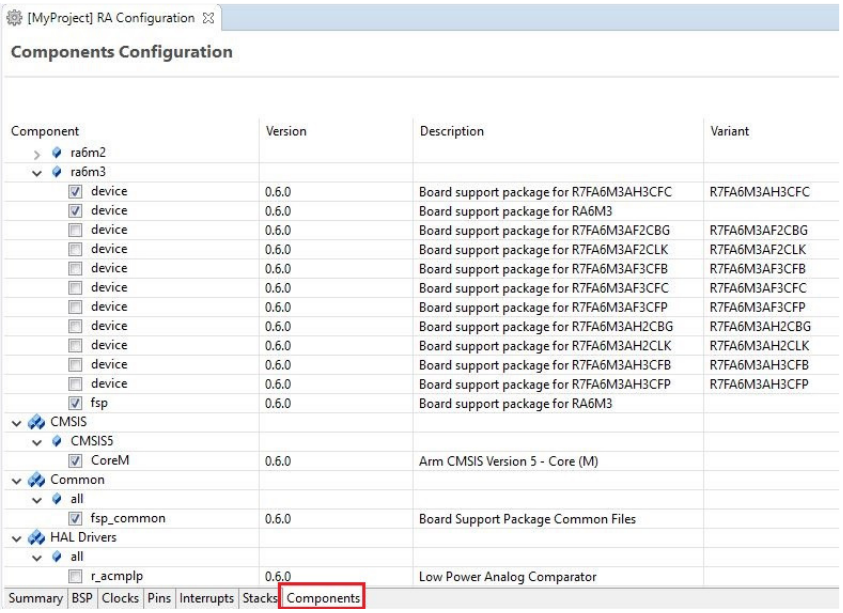



Figure 32: Configuring Thread Object Properties

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

## 2.2.7 Reviewing and Adding Components

The **Components** tab enables the individual modules required by the application to be included or excluded. Modules common to all RA MCU 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 **Stacks** tab are included automatically. You can include or exclude additional modules by ticking the box next to the required component.




| Component | Version | Description | Variant |
|---|---|---|---|
| ra6m2 | | | |
| ra6m3 | | | |
| device | 0.6.0 | Board support package for R7FA6M3AH3CFC | R7FA6M3AH3CFC |
| device | 0.6.0 | Board support package for RA6M3 | |
| device | 0.6.0 | Board support package for R7FA6M3AF2CBG | R7FA6M3AF2CBG |
| device | 0.6.0 | Board support package for R7FA6M3AF2CLK | R7FA6M3AF2CLK |
| device | 0.6.0 | Board support package for R7FA6M3AF3CFB | R7FA6M3AF3CFB |
| device | 0.6.0 | Board support package for R7FA6M3AF3CFC | R7FA6M3AF3CFC |
| device | 0.6.0 | Board support package for R7FA6M3AF3CFP | R7FA6M3AF3CFP |
| device | 0.6.0 | Board support package for R7FA6M3AH2CBG | R7FA6M3AH2CBG |
| device | 0.6.0 | Board support package for R7FA6M3AH2CLK | R7FA6M3AH2CLK |
| device | 0.6.0 | Board support package for R7FA6M3AH3CFB | R7FA6M3AH3CFB |
| device | 0.6.0 | Board support package for R7FA6M3AH3CFP | R7FA6M3AH3CFP |
| fsp | 0.6.0 | Board support package for RA6M3 | |
| CMSIS | | | |
| CMSIS5 | | | |
| CoreM | 0.6.0 | Arm CMSIS Version 5 - Core (M) | |
| Common | | | |
| all | | | |
| fsp_common | 0.6.0 | Board Support Package Common Files | |
| HAL Drivers | | | |
| all | | | |
| r_acmplp | 0.6.0 | Low Power Analog Comparator | |

Figure 33: Components Tab

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

- ra/fsp/inc/api
- ra/fsp/inc/instances
- ra/fsp/src/bsp
- ra/fsp/src/<Driver_Name>

e2 studio also creates configuration files in the ra_cfg/fsp_cfg folder with configuration options included from the remaining **Stacks** tabs.

## 2.2.8 Writing the Application

Once you have added Modules and drivers and set their configuration parameters in the **Stacks** 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.*

### 2.2.8.1 Coding Features

e2 studio provides several efficiency improving features that help write code. Review these features prior to digging into the code development step-by-step sections that follow.

#### Edit Hover

e2 studio supports hovers in the textual editor. This function can be enabled or disabled via **Window > Preferences > C/C++ > Editor > Hovers**.

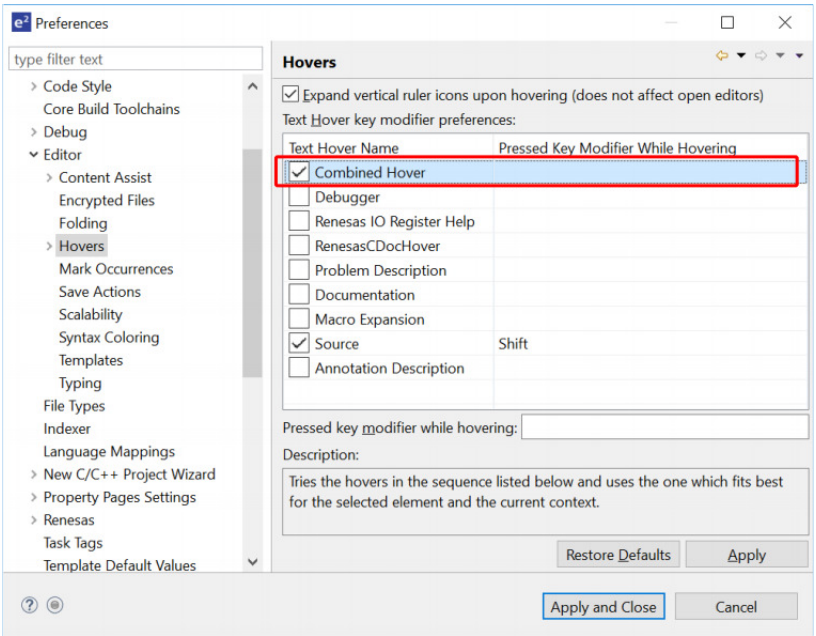



Figure 34: Hover preference

To enable hover, check **Combined Hover** box. To disable it, uncheck this box. By default, it is enabled. The Hover function allows a user to view detailed information about any identifiers in the source code by hovering the mouse over an identifier and checking the pop-up.

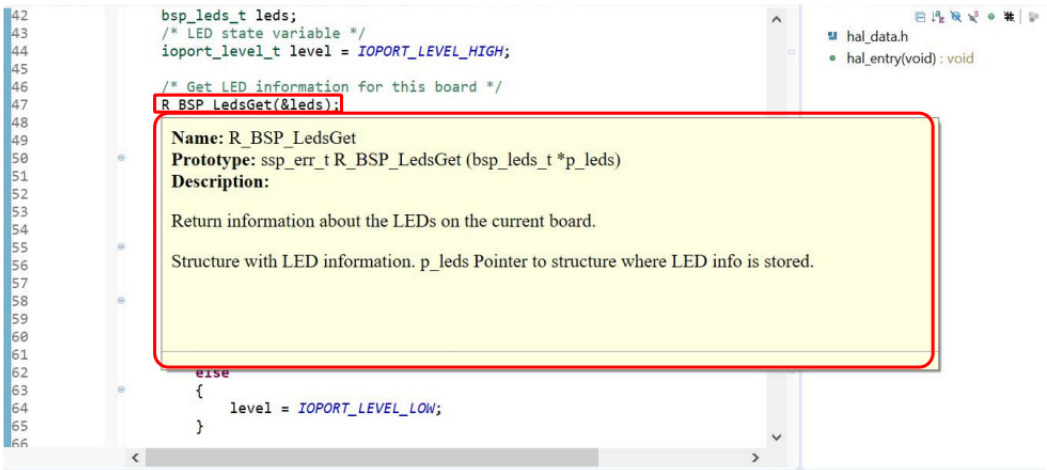



Figure 35: Hover Example

## Welcome Window

The e2 studio Welcome window displays useful information and common links to assist in development. Check out these resources to see what is available. They are updated with each release, so check back to see what has been added after a new release.

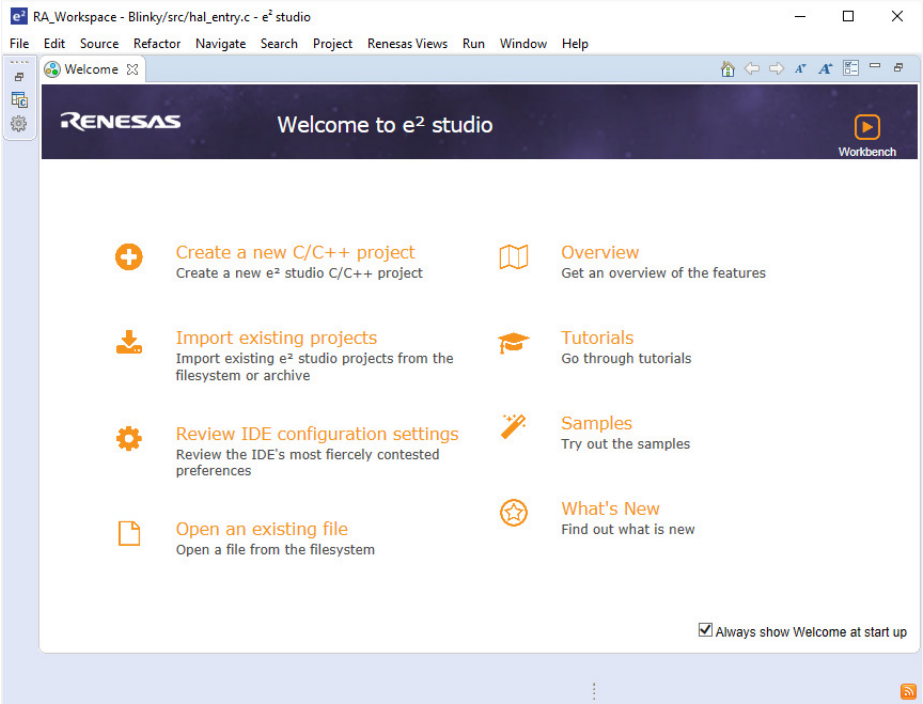



Figure 36: Welcome window

## Cheat Sheets

Cheat sheets are macro driven illustrations of some common tasks. They show, step-by-step, what commands and menus are used. These will be populated with more examples on each release. Cheat Sheets are available from the **Help** menu.

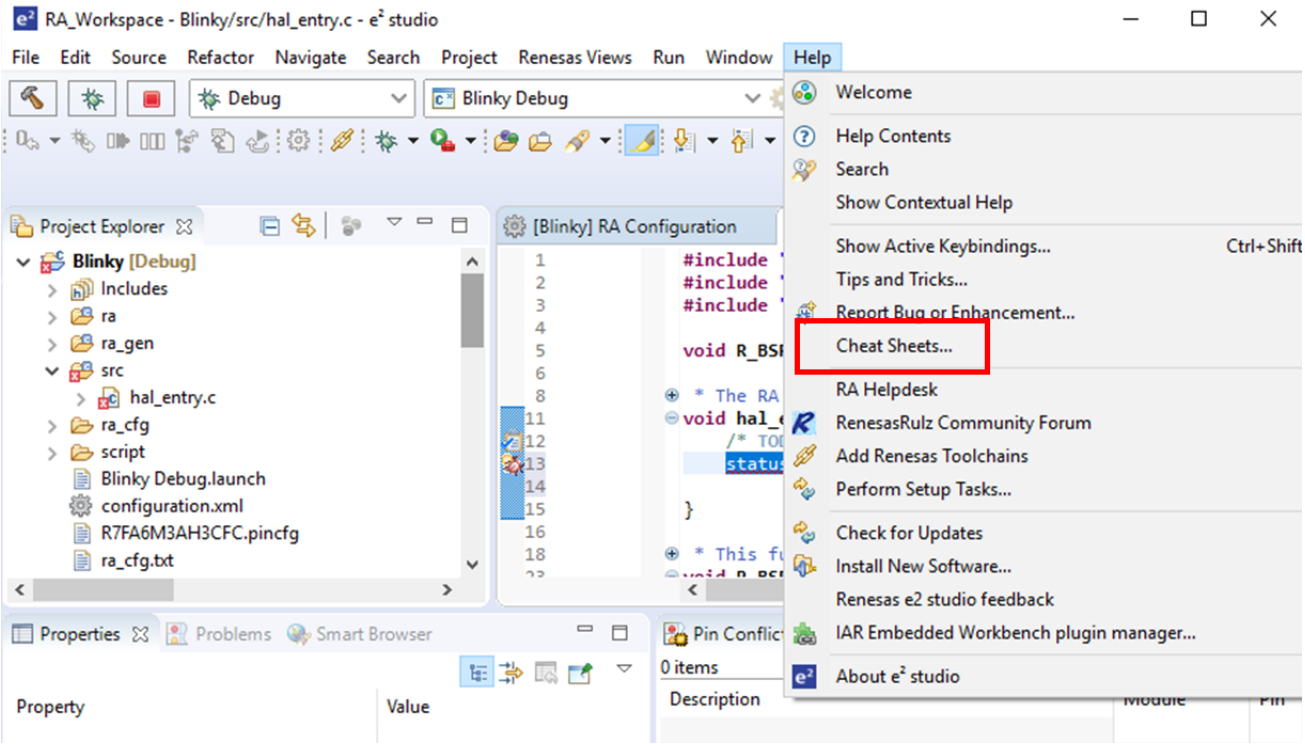



Figure 37: Cheat Sheets

## Developer Assistance

FSP Developer Assistance provides developers with module and Application Programming Interface (API) reference documentation in e2 studio. After configuring the threads and software stacks for an FSP project with the RA Configuration editor, Developer Assistance quickly helps you get started writing C/C++ application code for the project using the configured stack modules.

1. Expand the project explorer to view Developer Assistance

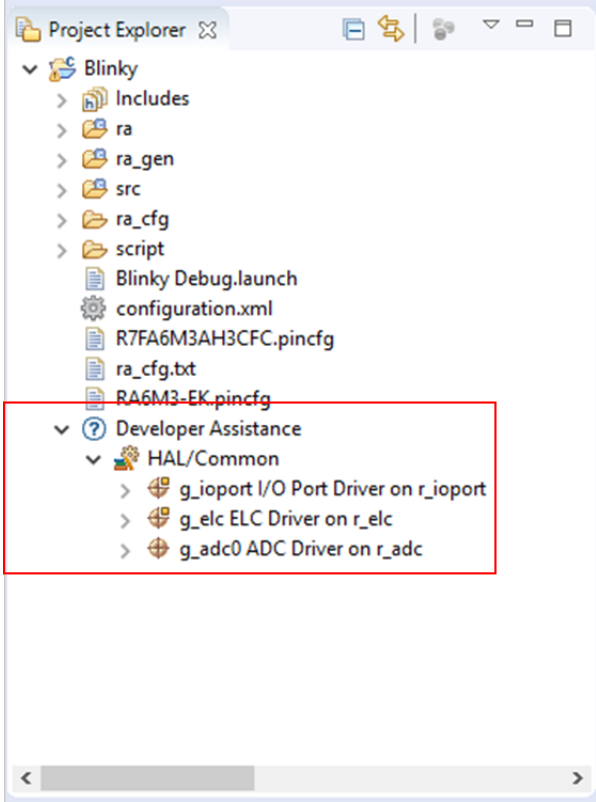



Figure 38: Developer Assistance

2. Expand a stack module to show its APIs

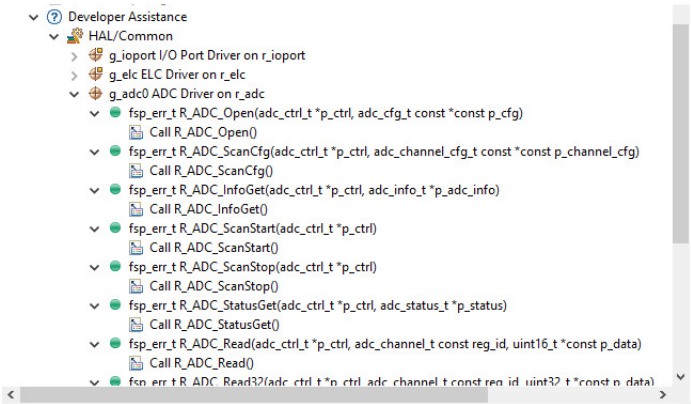



Figure 39: Developer Assistance APIs

3. Dragging and dropping an API from Develop Assistance to a source file helps to write source code quickly.

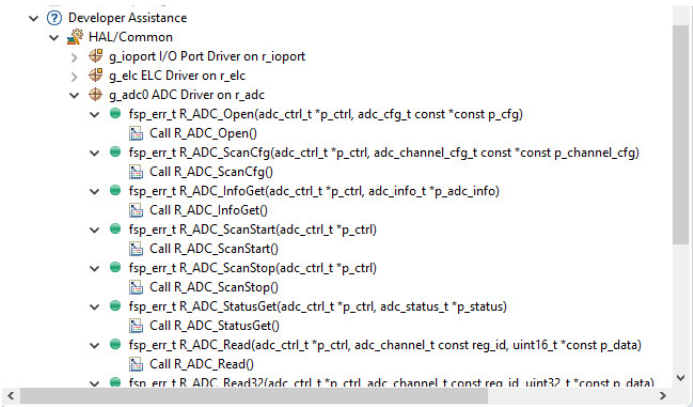



Figure 40: Dragging and Dropping an API in Developer Assistance

### Information Icon

Information icons are available on each module in the thread stack. Clicking on these icons opens a module folder on GitHub that contains additional information on the module. An example information Icon is shown below:

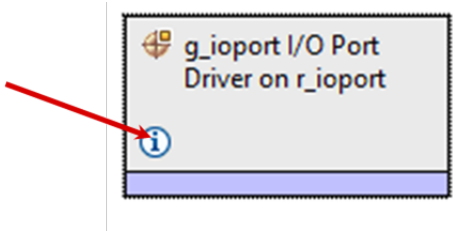



Figure 41: Information icon

### Smart Manual

Smart Manual is the view that displays information (register information/search results by keyword) extracted from the hardware user's manual. Smart Manual provides search capability of hardware manual information (register information search and keyword search result) and provides a view displaying result.

You can open Smart Manual view by selecting the menu: **Renesas Views > Solution Toolkit > Smart Manual**. Register search and Keyword search are both available by selecting the appropriate tab.

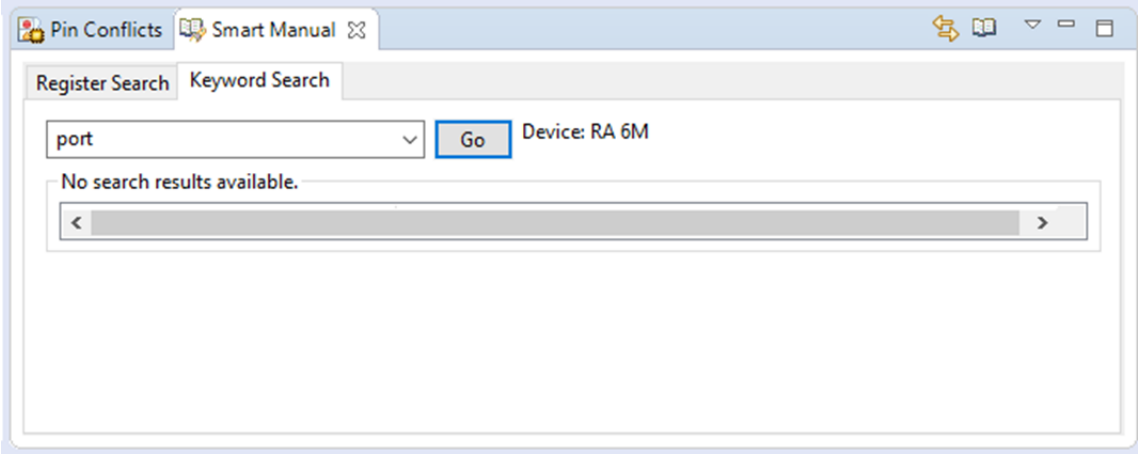


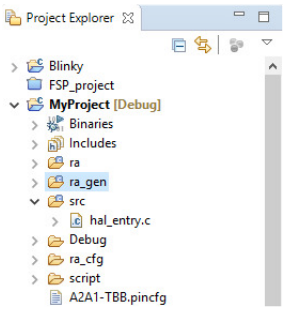

Figure 42: Smart Manual

### 2.2.8.2 RTOS-independent Applications

To write application code:

1. Add all drivers and modules in the **Stacks** tab and resolve all dependencies flagged by e2 studio such as missing interrupts or drivers.
2. Configure the drivers in the **Properties** view.
3. In the Project Configuration view, click 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.




*Note*

*All configuration structures necessary for the driver to be called in the application are initialized in ra_gen/hal_data.c.*

Warning

Do not modify the files in the directory ra_gen. These files are overwritten every time you push the **Generate Project Content** button.

5. Add your application code here:

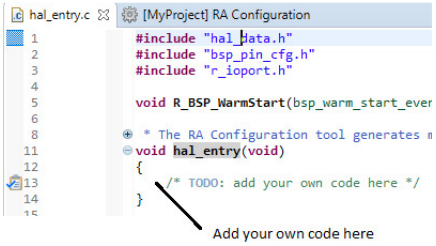



Figure 43: Adding user code to hal_entry.c

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.

### 2.2.8.3 RTOS Applications

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

1. Add a thread using the **Stacks** 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 e2 studio 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 **RA Project Editor**, click 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.

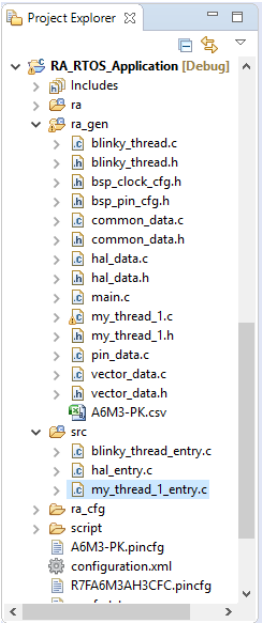



Figure 44: Generated files for an RTOS application

*Note*

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

Warning

Do not modify the files in the directory ra_gen. These files are overwritten every time you push the **Generate Project Content** button.

9. Add your application code here:

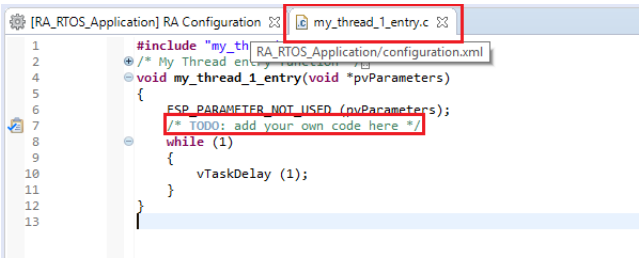



Figure 45: Adding user code to my_thread_1.entry

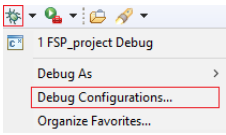
10. Repeat steps 1 to 9 for the next thread.
11. Build your project without errors by clicking on **Project > Build Project**.

## 2.2.9 Debugging the Project

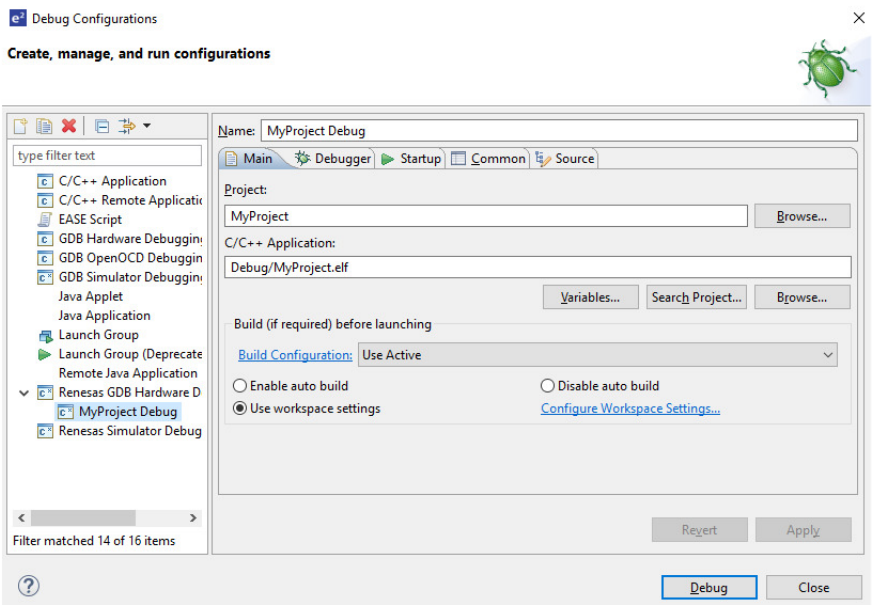
Once your project builds without errors, you can use the Debugger to download your application to the board and execute it.

To debug an application follow these steps:

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




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




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 RA EKs) 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 RA MCU Kit.*

## 2.2.10 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 e2 studio through the menu **Project > Properties > 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.

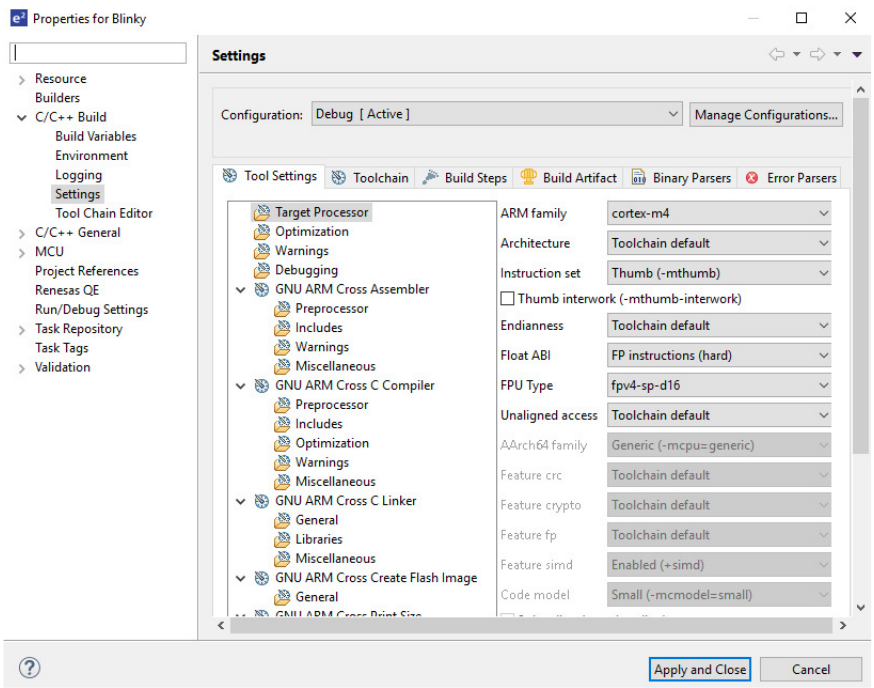



Figure 46: e2 studio Project toolchain settings

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/a6m3.ld).

## 2.2.11 Importing an Existing Project into e2 studio

1. Start by opening e2 studio.
2. Open an existing Workspace to import the project and skip to step d. If the workspace doesn't exist, proceed with the following steps:

    a. At the end of e2 studio startup, you will see the Workspace Launcher Dialog box as shown in the following figure.

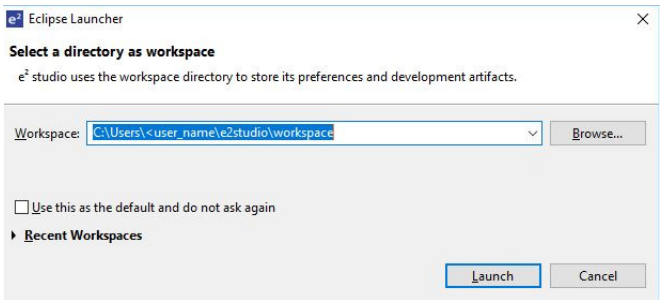



Figure 47: Workspace Launcher dialog

b. Enter a new workspace name in the Workspace Launcher Dialog as shown in the following figure. e2 studio creates a new workspace with this name.

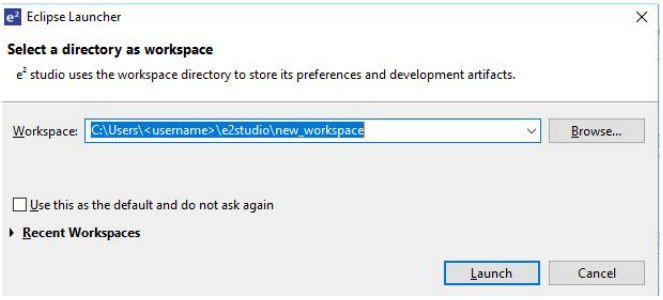



Figure 48: Workspace Launcher dialog - Select Workspace

c. Click **Launch**.

d. When the workspace is opened, you may see the Welcome Window. Click on the **Workbench** arrow button to proceed past the Welcome Screen as seen in the following figure.

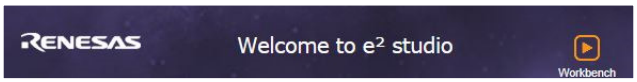



Figure 49: Workbench arrow button

3. You are now in the workspace that you want to import the project into. Click the **File** menu in the menu bar, as shown in the following figure.

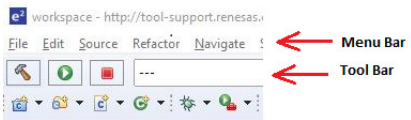



Figure 50: Menu and tool bar

4. Click **Import** on the **File** menu or in the menu bar, as shown in the following figure.

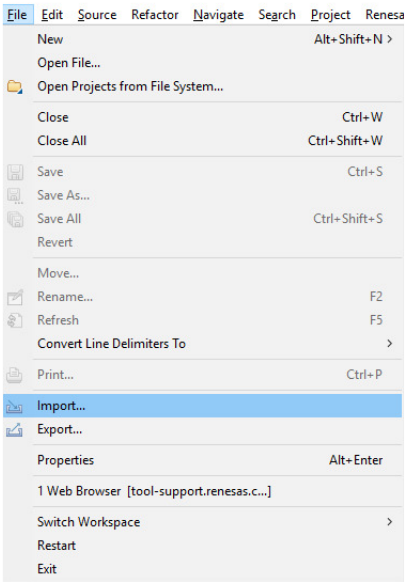



Figure 51: File drop-down menu

5. In the **Import** dialog box, as shown in the following figure, choose the **General** option, then **Existing Projects into Workspace**, to import the project into the current workspace.

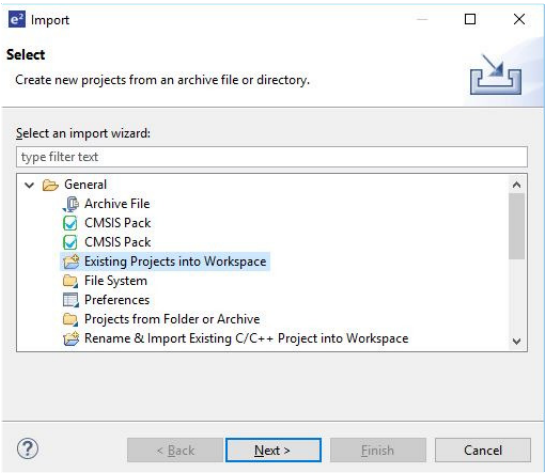



Figure 52: Project Import dialog with "Existing Projects into Workspace" option selected

6. Click **Next**.
7. To import the project, use either **Select archive file** or **Select root directory**.

   a. Click **Select archive file** as shown in the following figure.

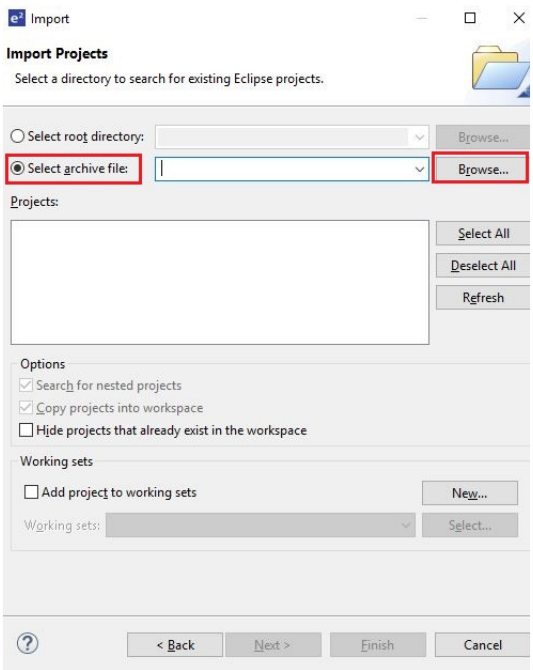



Figure 53: Import Existing Project dialog 1 - Select archive file

b. Click **Select root directory** as shown in the following figure.

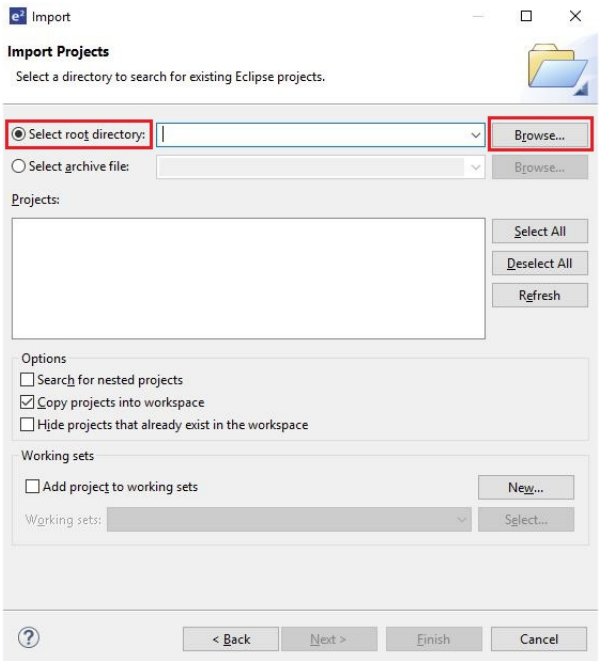



Figure 54: Import Existing Project dialog 1 - Select root directory

8. Click **Browse**.
9. For **Select archive file**, browse to the folder where the zip file for the project you want to import is located. For **Select root directory**, browse to the project folder that you want to import.
10. Select the file for import. In our example, it is CAN_HAL_MG_AP.zip or CAN_HAL_MG_AP.

11. Click **Open**.

12. Select the project to import from the list of **Projects**, as shown in the following figure.

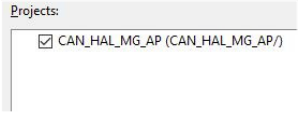



Figure 55: Import Existing Project dialog 2

13. Click **Finish** to import the project.

# 2.3 Tutorial: Your First RA MCU Project - Blinky

## 2.3.1 Tutorial Blinky

The goal of this tutorial is to quickly get acquainted with the Flexible Platform by moving through the steps of creating a simple application using e2 studio and running that application on an RA MCU board.

## 2.3.2 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 RA 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.

## 2.3.3 Prerequisites

To follow this tutorial, you need:

- Windows based PC
- e2 studio
- Flexible Software Package
- An RA MCU board kit

## 2.3.4 Create a New Project for Blinky

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

Follow these steps to create an RA MCU project:

1. In e2 studio, click **File > New > RA Project** and select **Renesas RA C Executable Project**.
2. Assign a name to this new project. Blinky is a good name to use for this tutorial.

3. Click **Next**. The **Project Configuration** window shows your selection.

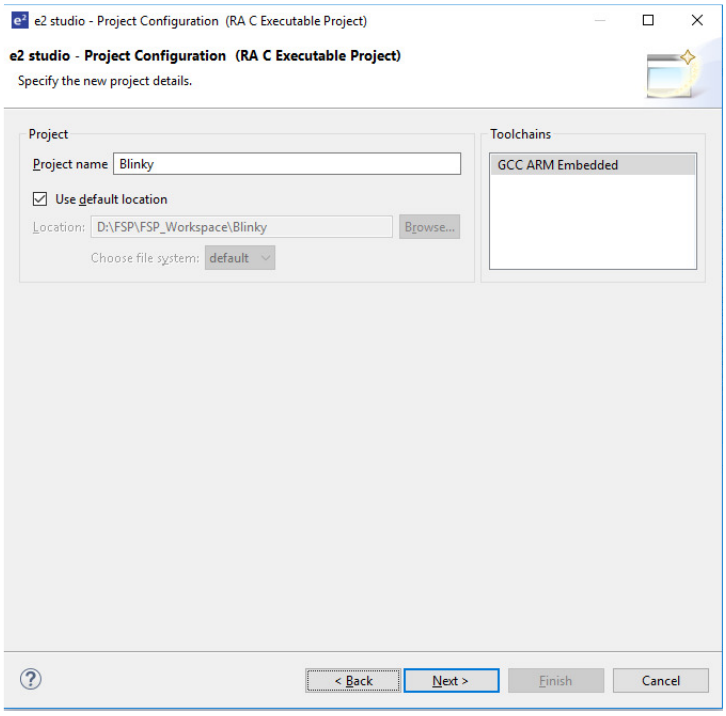



Figure 56: e2 studio Project Configuration window (part 1)

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

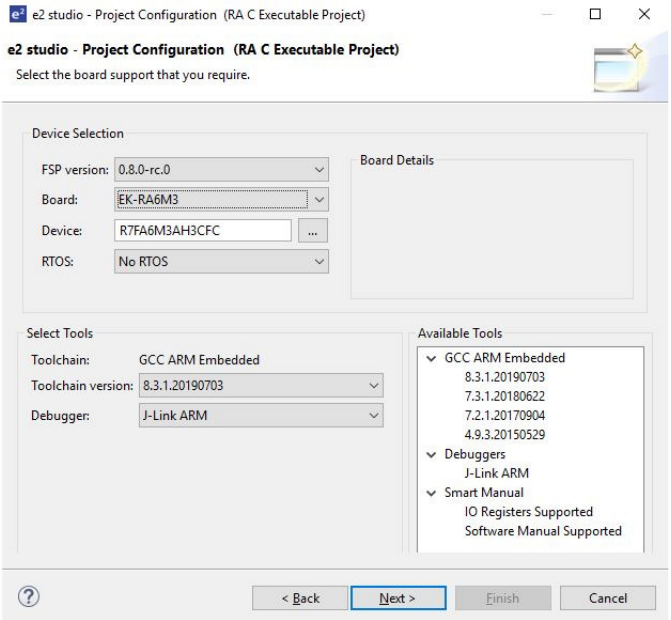



Figure 57: e2 studio Project Configuration window (part 2)

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

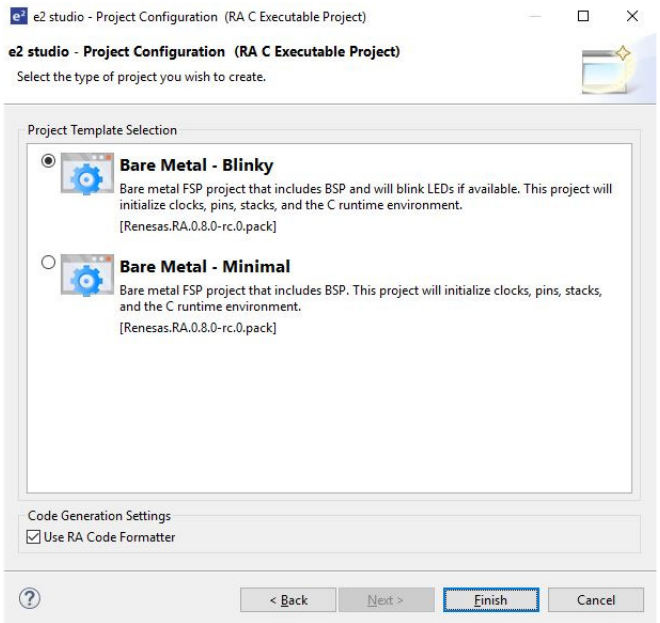



Figure 58: e2 studio Project Configuration window (part 3)

Once the project has been created, the name of the project will show up in the **Project Explorer** window of e2 studio. Now click the **Generate Project Content** button in the top right corner of the **Project Configuration** window to generate your board specific files.

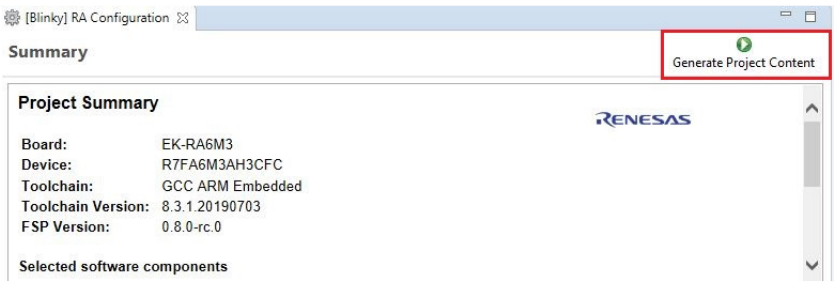



Figure 59: e2 studio Project Configuration tab

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

#### 2.3.4.1 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 click 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.

#### 2.3.4.2 Configuring the Blinky Clocks

By selecting the Blinky template, the clocks are configured by e2 studio for the Blinky application. The 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).

#### 2.3.4.3 Configuring the Blinky Pins

By selecting the Blinky template, the GPIO pins used to toggle the LED1 are configured by e2 studio for the Blinky application. The 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).

#### 2.3.4.4 Configuring the Parameters for Blinky Components

The Blinky project automatically selects the following HAL components in the Components tab:

- 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).

#### 2.3.4.5 Where is main()?

The main function is located in < project >/ra_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, see Adding and Configuring HAL Drivers.

#### 2.3.4.6 Blinky Example Code

The blinky application is stored in the hal_entry.c file. This file is generated by e2 studio 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 bsp_leds_t structure.
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 R_BSP_PinWrite((bsp_io_port_pin_t) pin, pin_level);

## 2.3.5 Build the Blinky Project

Highlight the new project in the **Project Explorer** window by clicking on it and build it.

There are three ways to build a project:

1. Click on **Project** in the menu bar and select **Build Project**.
2. Click on the hammer icon.
3. Right-click on the project and select **Build Project**.

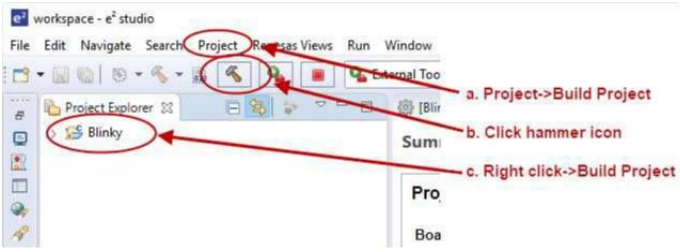



Figure 60: e2 studio Project Explorer window

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.

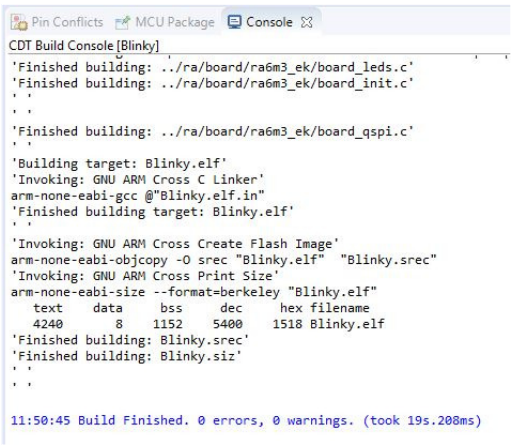



Figure 61: e2 studio Project Build console

## 2.3.6 Debug the Blinky Project

### 2.3.6.1 Debug prerequisites

To debug the project on a board, you need

- The board to be connected to e2 studio
- 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 e2 studio.

### 2.3.6.2 Debug steps

To debug the Blinky application, follow these steps:

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

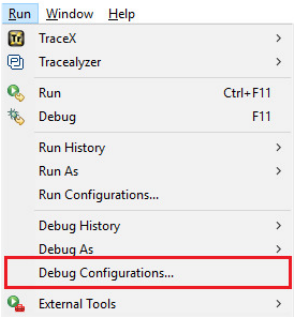



Figure 62: e2 studio Debug icon

or by selecting the drop-down menu next to the bug icon and selecting **Debugger Configurations ...**

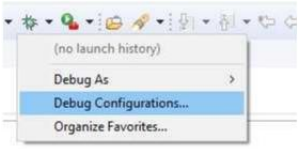



Figure 63: e2 studio Debugger Configurations selection option

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.

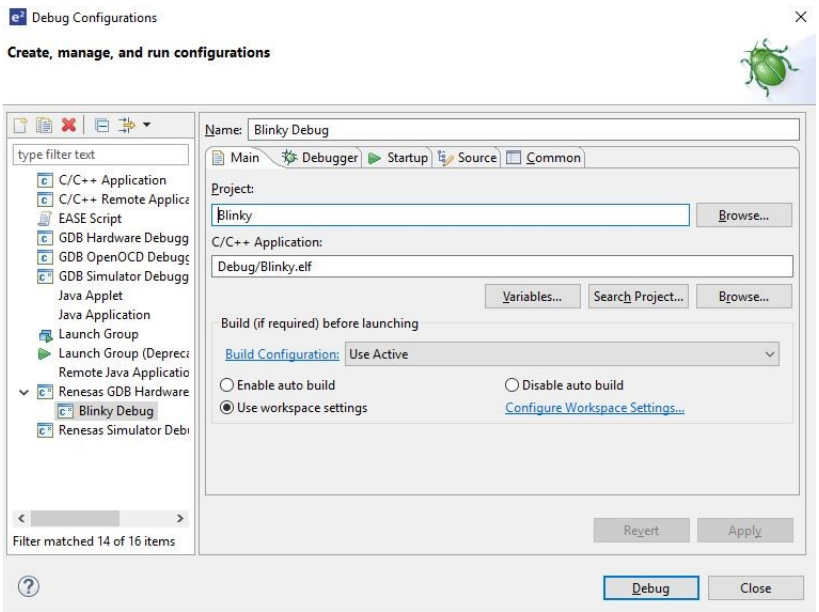
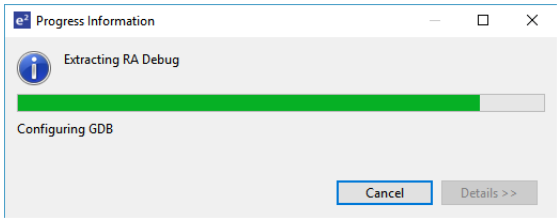



Figure 64: e2 studio Debugger Configurations window with Blinky project

3. Click **Debug** to begin debugging the application.

4. Extracting RA Debug.




### 2.3.6.3 Details about the Debug Process

In debug mode, e2 studio 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.

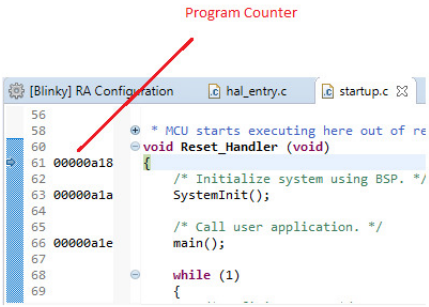



Figure 65: e2 studio Debugger memory window

### 2.3.7 Run the Blinky Project

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

Figure 66: e2 studio Debugger Play icon

The LEDs on the board marked LED1, LED2, and LED3 should now be blinking.

## 2.4 Tutorial: Using HAL Drivers - Programming the WDT

### 2.4.1 Application WDT

This application uses the WDT Interface implemented by the WDT HAL Driver WDT. This document describes how to use e2 studio and FSP to create an application for the RA MCU Watchdog Timer (WDT) peripheral. This application makes use of the following FSP modules:

- MCU Board Support Package
- Watchdog Timer (r_wdt)
- I/O Ports (r_ioport)

### 2.4.2 Creating a WDT Application Using the RA MCU FSP and e2 studio

#### 2.4.2.1 Using the FSP and e2 studio

The Flexible Software Package (FSP) from Renesas provides a complete driver library for developing RA MCU applications. The FSP provides Hardware Abstraction Layer (HAL) drivers, Board Support Package (BSP) drivers for the developer to use to create applications. The FSP is integrated into Renesas e2 studio based on eclipse providing build (editor, compiler and linker) and debug phases with an extended GNU Debug (GDB) interface.

#### 2.4.2.2 The WDT Application

The flowchart for the WDT application is shown below.

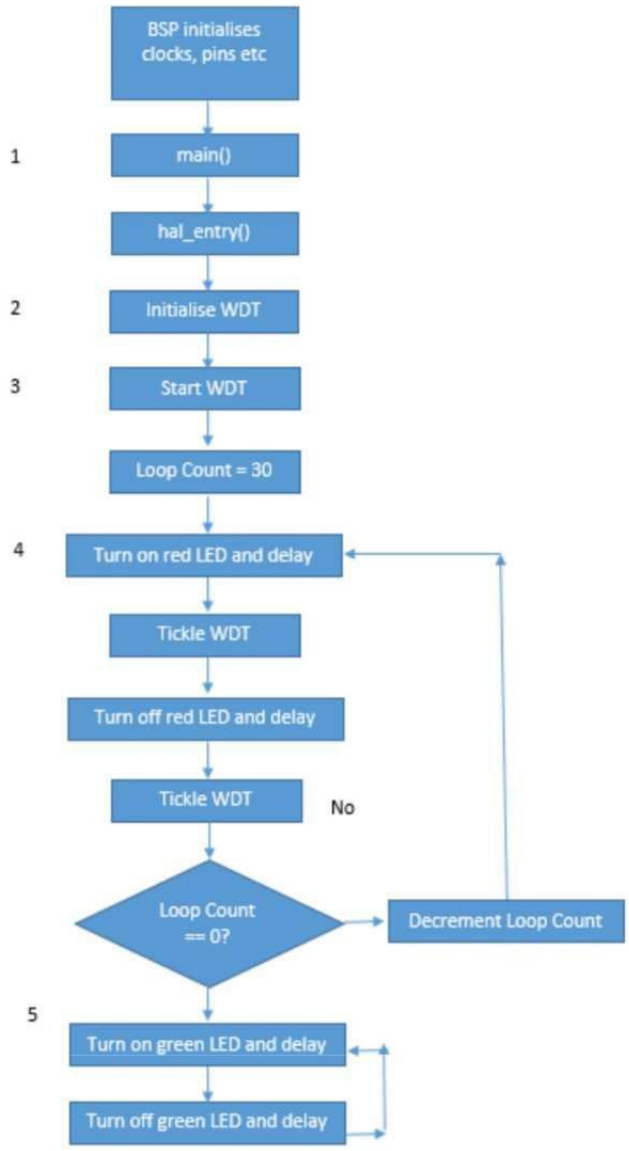



Figure 67: WDT Application flow diagram

#### 2.4.2.3 WDT Application flow

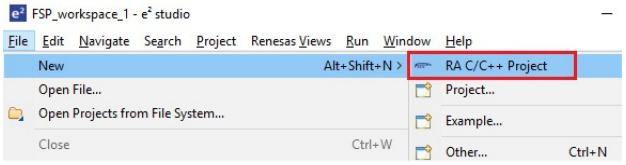
These are the main parts of the WDT application:

1. main() calls hal_entry(). The function hal_entry() is created by the FSP 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.

### 2.4.3 Creating the Project with e2 studio

Start e2 studio and choose a workspace folder in the Workspace Launcher. Configure a new RA MCU project as follows.

1. Select **File > New > RA C/C++ Project**. Then select the template for the project.



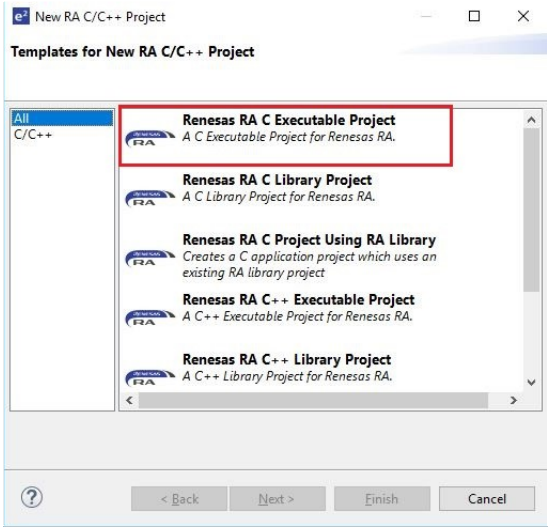




Figure 68: Creating a new project

2. In e2 studio Project **Configuration (RA Project)** window enter a project name, for example, WDT_Application. In addition select the toolchain. If you want to choose new locations for the project unselect **Use default location**. Click **Next**.

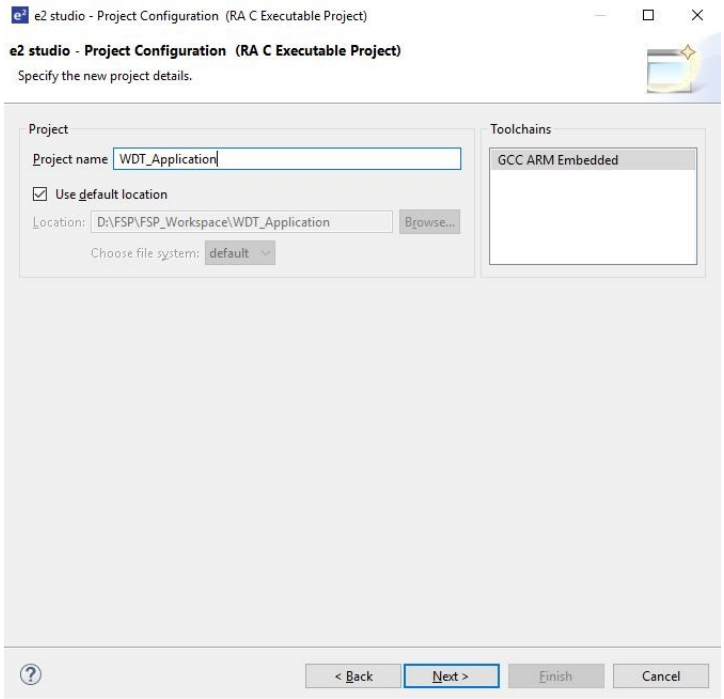



Figure 69: Project configuration (part 1)

3. This application runs on the RA6M3 board. So, for the **Board** select **EK-RA6M3**.

   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**. Click **Next** to configure the project.

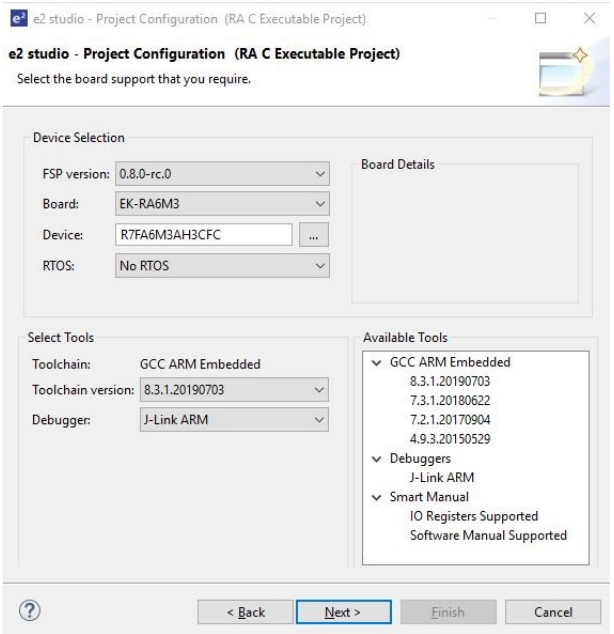



Figure 70: Project configuration (part 2)

The project template is now selected. As no RTOS is required select **Bare Metal - Blinky**.

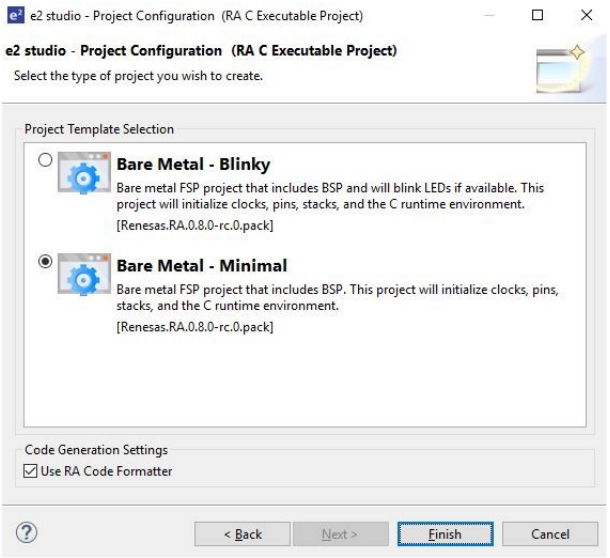



Figure 71: Project configuration (part 3)

4. Click **Finish**.

   e2 studio creates the project and opens the **Project Explorer** and **Project Configuration Settings** views with the **Summary** page showing a summary of the project configuration.

## 2.4.4 Configuring the Project with e2 studio

e2 studio simplifies and accelerates the project configuration process by providing a GUI interface for selecting the options to configure the project.

e2 studio offers a selection of perspectives presenting different windows to the user depending on the operation in progress. The default perspectives are **C/C++**, **RA Configuration** and **Debug**. The perspective can be changed by selecting a new one from the buttons at the top right.

Figure 72: Selecting a perspective

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

1. In order to configure the project settings ensure the **RA 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 **RA Configuration** perspective is selected and double-click on the configuration.xml file in the Project Explorer pane on the right side of e2 studio.

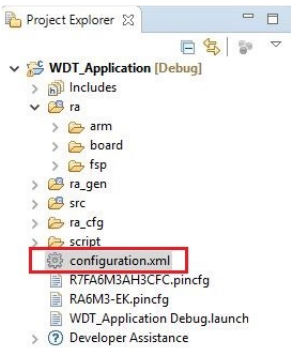



Figure 73: RA MCU Project Configuration Settings

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.

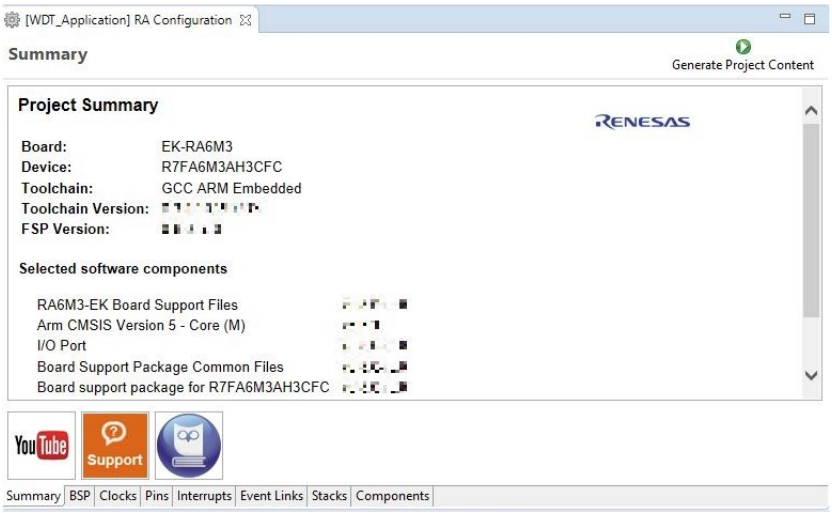



Figure 74: Project Configuration Tabs

### 2.4.4.1 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 RA Hardware User's Manual for details on the WDT autostart mode.

### 2.4.4.2 Clocks Tab

The **Clocks** tab presents a graphical view of the clock tree of the device. 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.

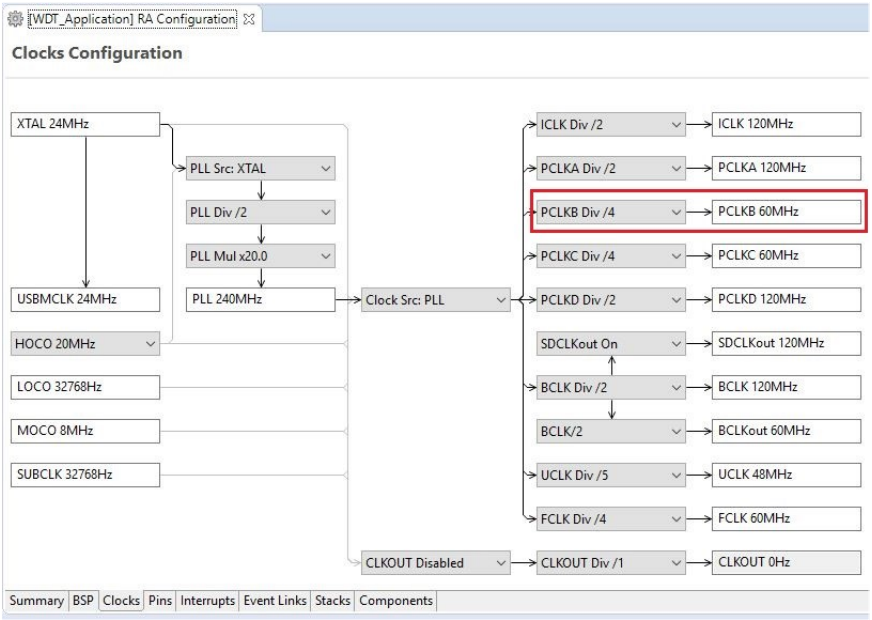



Figure 75: Clock configuration

### 2.4.4.3 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.

### 2.4.4.4 Stacks Tab

You can add any driver to the project using the **Stacks** tab. The HAL driver IO port pins are added automatically by e2 studio when the project is configured. The WDT application uses no RTOS Resources, so you only need to add the HAL WDT driver.

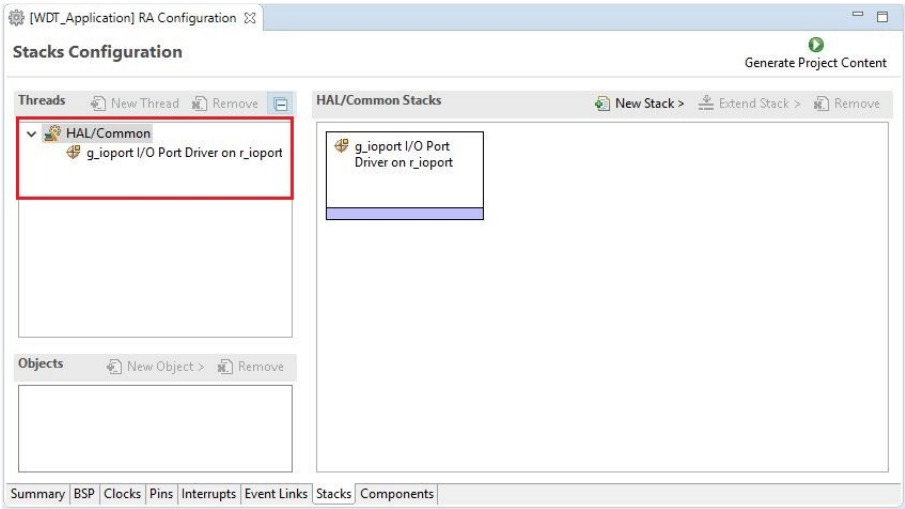



Figure 76: Stacks tab

1. Click on the **HAL/Common Panel** in the Threads Window as indicated in the figure above.

The Stacks Panel becomes a **HAL/Common Stacks** panel and is populated with the modules preselected by e2 studio.

2. Click on **New Stack** to find a pop-up window with the available HAL level drivers.
3. Select **WATCHDOG Driver on r_wdt**.

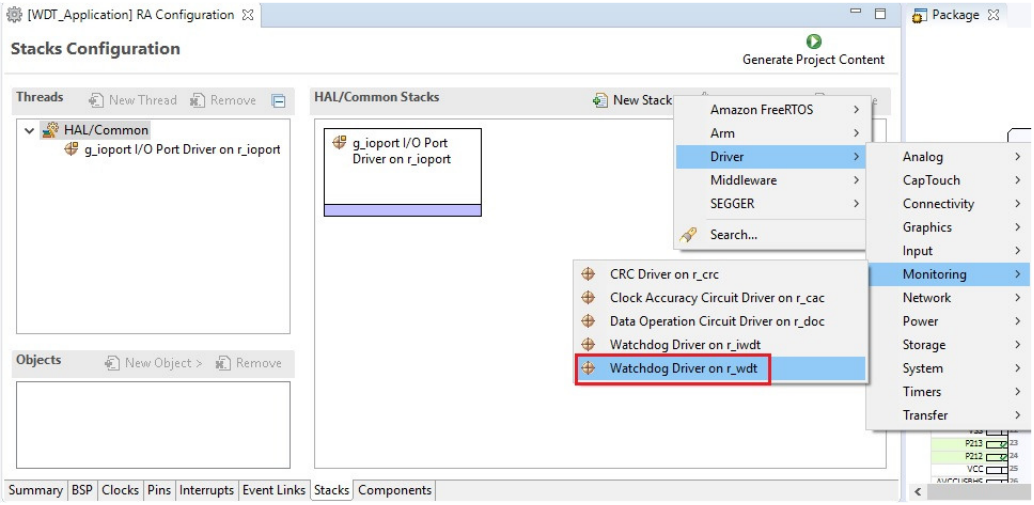



Figure 77: Module Selection

The selected HAL WDT driver is added to the **HAL/Common Stacks** 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 is not visible, check that the **RA Configuration** perspective is selected.

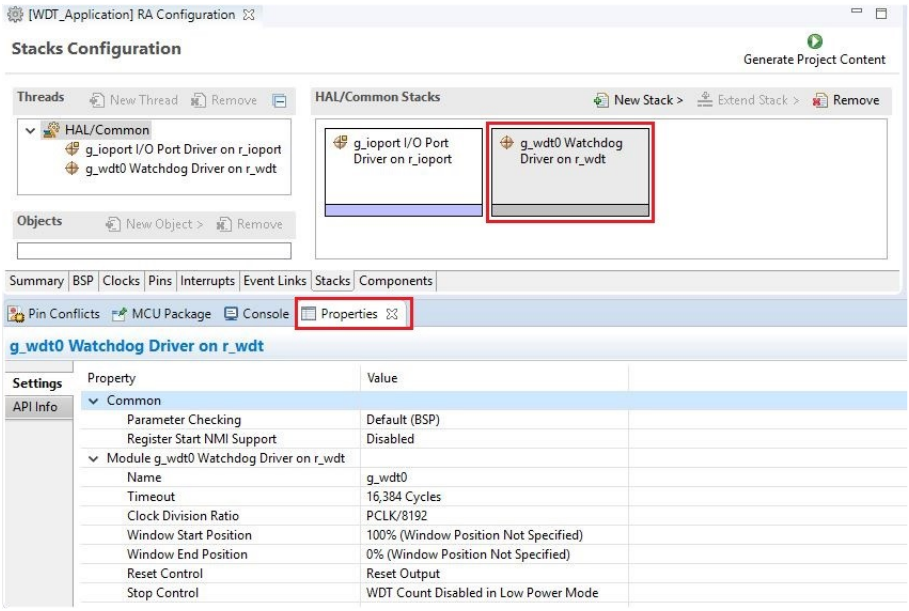



Figure 78: Module Properties

All parameters can be left with their default values.

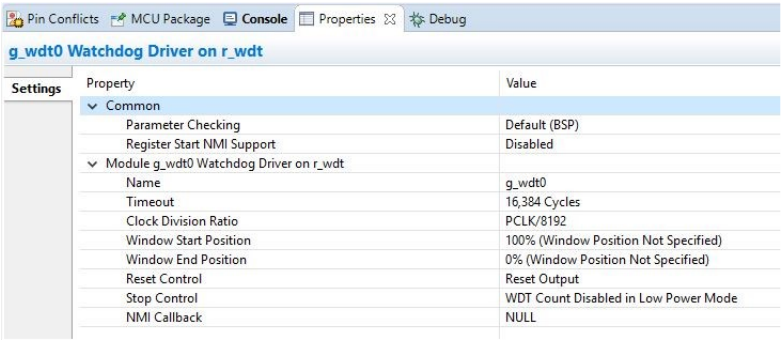



Figure 79: g_wdt WATCHDOG Driver on WDT properties

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.

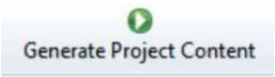



Figure 80: Generate Project Content button

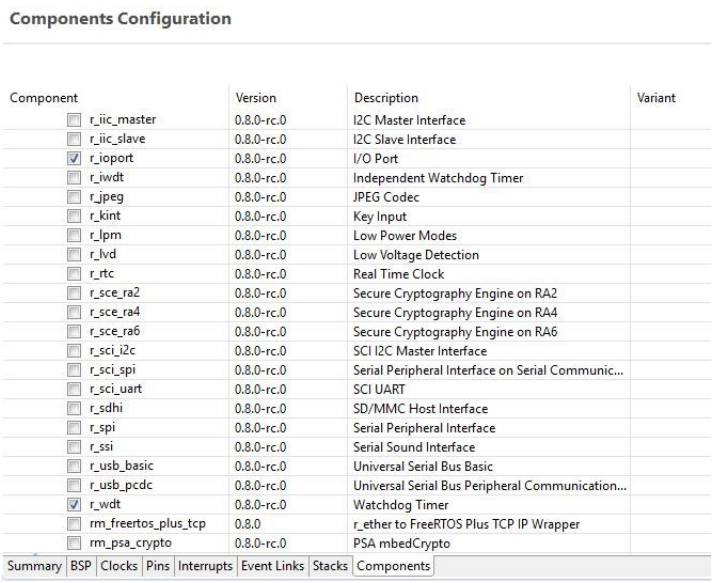
e2 studio generates the project files.

#### 2.4.4.5 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 after they are added in the Stacks Tab.

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

1. HAL_Drivers -> r_ioport
2. HAL_Drivers -> r_wdt



Components Configuration

| Component | Version | Description | Variant |
|---|---|---|---|
| r_iic_master | 0.8.0-rc.0 | I2C Master Interface | |
| r_iic_slave | 0.8.0-rc.0 | I2C Slave Interface | |
| r_ioport | 0.8.0-rc.0 | I/O Port | |
| r_iwdt | 0.8.0-rc.0 | Independent Watchdog Timer | |
| r_jpeg | 0.8.0-rc.0 | JPEG Codec | |
| r_kint | 0.8.0-rc.0 | Key Input | |
| r_lpm | 0.8.0-rc.0 | Low Power Modes | |
| r_lvd | 0.8.0-rc.0 | Low Voltage Detection | |
| r_rtc | 0.8.0-rc.0 | Real Time Clock | |
| r_sce_ra2 | 0.8.0-rc.0 | Secure Cryptography Engine on RA2 | |
| r_sce_ra4 | 0.8.0-rc.0 | Secure Cryptography Engine on RA4 | |
| r_sce_ra6 | 0.8.0-rc.0 | Secure Cryptography Engine on RA6 | |
| r_sci_i2c | 0.8.0-rc.0 | SCI I2C Master Interface | |
| r_sci_spi | 0.8.0-rc.0 | Serial Peripheral Interface on Serial Communic... | |
| r_sci_uart | 0.8.0-rc.0 | SCI UART | |
| r_sdhi | 0.8.0-rc.0 | SD/MMC Host Interface | |
| r_spi | 0.8.0-rc.0 | Serial Peripheral Interface | |
| r_ssi | 0.8.0-rc.0 | Serial Sound Interface | |
| r_usb_basic | 0.8.0-rc.0 | Universal Serial Bus Basic | |
| r_usb_pcdc | 0.8.0-rc.0 | Universal Serial Bus Peripheral Communication... | |
| r_wdt | 0.8.0-rc.0 | Watchdog Timer | |
| rm_freertos_plus_tcp | 0.8.0 | r_ether to FreeRTOS Plus TCP IP Wrapper | |
| rm_psa_crypto | 0.8.0-rc.0 | PSA mbedCrypto | |

Summary | BSP | Clocks | Pins | Interrupts | Event Links | Stacks | Components

Figure 81: Component Selection

*Note*

*The list of modules displayed in the Components tab depends on the installed FSP version.*

## 2.4.5 WDT Generated Project Files

Clicking the Generate Project Content button performs the following tasks.

- r_wdt folder and WDT driver contents created at:

  ra/fsp/src

- r_wdt_api.h created in:

  ra/fsp/inc/api

- r_wdt.h created in:

  ra/fsp/inc/instance

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:

ra_cfg/fsp_cfg/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.

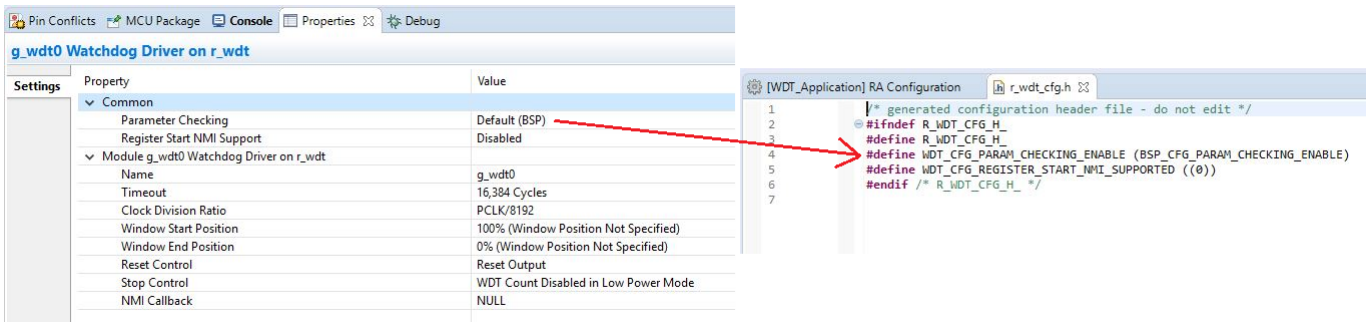



Figure 82: r_wdt_cfg.h contents

Warning

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

The r_ioport folder is not created at ra/fsp/src 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 ra_gen/hal_data.c–see later in this document for further details. For the same reason the other IOPORT header files– ra/fsp/inc/api/r_ioport_api.handra/fsp/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 e2 studio also generates files containing configuration data for the WDT and a file where user code can safely be added. These files are shown below.

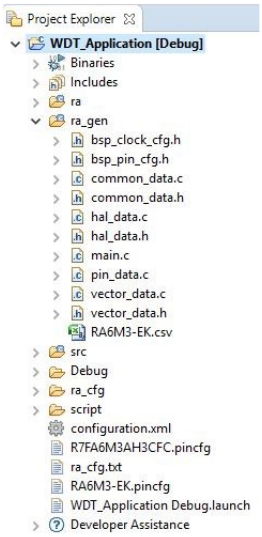



Figure 83: WDT project files

### 2.4.5.1 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 "common_data.h"
 #include "r_wdt.h"
 #include "r_wdt_api.h"
 #ifdef __cplusplus
extern "C"
{
 #endif
extern const wdt_instance_t g_wdt0;
 #ifndef NULL
void NULL(wdt_callback_args_t * p_args);
 #endif
extern wdt_instance_ctrl_t g_wdt0_ctrl;
extern const wdt_cfg_t g_wdt0_cfg;
void hal_entry(void);
void g_hal_init(void);
 #ifdef __cplusplus
} /* extern "C" */
 #endif
#endif  /* HAL_DATA_H_ */
```

hal_data.h contains the header files required by the 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.

Warning

This file is regenerated each time Generate Project Content is clicked and must not be edited.

### 2.4.5.2 WDT hal_data.c

The contents of hal_data.c are shown below.

```
/* generated HAL source file - do not edit */
#include "hal_data.h"
wdt_instance_ctrl_t g_wdt0_ctrl;
```

```
const wdt_cfg_t g_wdt0_cfg =
{
    .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_ENABLE,
    .p_callback    = NULL,
};
/* Instance structure to use this module. */
const wdt_instance_t g_wdt0 =
{.p_ctrl = &g_wdt0_ctrl, .p_cfg = &g_wdt0_cfg, .p_api = &g_wdt_on_wdt};
void g_hal_init (void)
{
    g_common_init();
}
```

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 initialized 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 **e2 studio Project Configuration HAL** tab. If the contents of this structure do not reflect the settings made in e2 studio, ensure the **Project Configuration** settings are saved before clicking the **Generate Project Content** button.

Warning

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

### 2.4.5.3 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*/
#include "hal_data.h"
int main (void)
{
    hal_entry();
```

```
 return 0;
}
```

Warning

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

#### 2.4.5.4 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.

```
#include "hal_data.h"
#include "bsp_pin_cfg.h"
#include "r_ioport.h"
#define RED_LED_NO_OF_FLASHES 30
#define RED_LED_PIN BSP_IO_PORT_01_PIN_00
#define GREEN_LED_PIN BSP_IO_PORT_04_PIN_00
#define RED_LED_DELAY_COUNT 1500000
#define GRN_LED_DELAY_COUNT 1200000
volatile uint32_t delay_counter;
volatile uint16_t loop_counter;
void R_BSP_WarmStart(bsp_warm_start_event_t event);
/* global variable to access board LEDs */
extern bsp_leds_t g_bsp_leds;
/*******************************************************************************
******************************/
void hal_entry (void) {
 /* Open the WDT */
 R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
 /* Start the WDT by refreshing it */
 R_WDT_Refresh(&g_wdt0_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 */
R_IOPORT_PinWrite(&g_ioport_ctrl, RED_LED_PIN, BSP_IO_LEVEL_LOW);
/* Delay */
for (delay_counter = 0; delay_counter < RED_LED_DELAY_COUNT; delay_counter++)
     {
/* Do nothing. */
      }
/* Refresh WDT */
R_WDT_Refresh(&g_wdt0_ctrl);
R_IOPORT_PinWrite(&g_ioport_ctrl, RED_LED_PIN, BSP_IO_LEVEL_HIGH);
/* Delay */
for (delay_counter = 0; delay_counter < RED_LED_DELAY_COUNT; delay_counter++)
     {
/* Do nothing. */
      }
/* Refresh WDT */
R_WDT_Refresh(&g_wdt0_ctrl);
   }
/* Flash green LED but STOP tickling the WDT. WDT should reset the
 * device */
while (1)
   {
/* Turn green LED on */
R_IOPORT_PinWrite(&g_ioport_ctrl, GREEN_LED_PIN, BSP_IO_LEVEL_LOW);
/* Delay */
for (delay_counter = 0; delay_counter < GRN_LED_DELAY_COUNT; delay_counter++)
     {
/* Do nothing. */
      }
/* Turn green off */
R_IOPORT_PinWrite(&g_ioport_ctrl, GREEN_LED_PIN, BSP_IO_LEVEL_HIGH);
/* Delay */
for (delay_counter = 0; delay_counter < GRN_LED_DELAY_COUNT; delay_counter++)
     {
```

```
 /* Do nothing. */
       }
    }
}
/*******************************************************************************
******************************/
void R_BSP_WarmStart (bsp_warm_start_event_t event)
{
 if (BSP_WARM_START_RESET == event)
    {
#if BSP_FEATURE_FLASH_LP_VERSION != 0
 /* Enable reading from data flash. */
       R_FACI_LP->DFLCTL = 1U;
 /* Would normally have to wait for tDSTOP(6us) for data flash recovery. Placing the
enable here, before clock and
  * C runtime initialization, should negate the need for a delay since the
initialization will typically take more than 6us. */
#endif
    }
 if (BSP_WARM_START_POST_C == event)
    {
 /* C runtime environment and system clocks are setup. */
 /* Configure pins. */
 R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
    }
}
```

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:

```
 /* Open the WDT */
 R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
```

The first passed parameter is the pointer to the control structure g_wdt_ctrl instantiated

inhal_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:

```
/* Start the WDT by refreshing it */

R_WDT_Refresh(&g_wdt0_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.

### 2.4.6 Building and Testing the Project

Build the project by clicking **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 e2 studio 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.

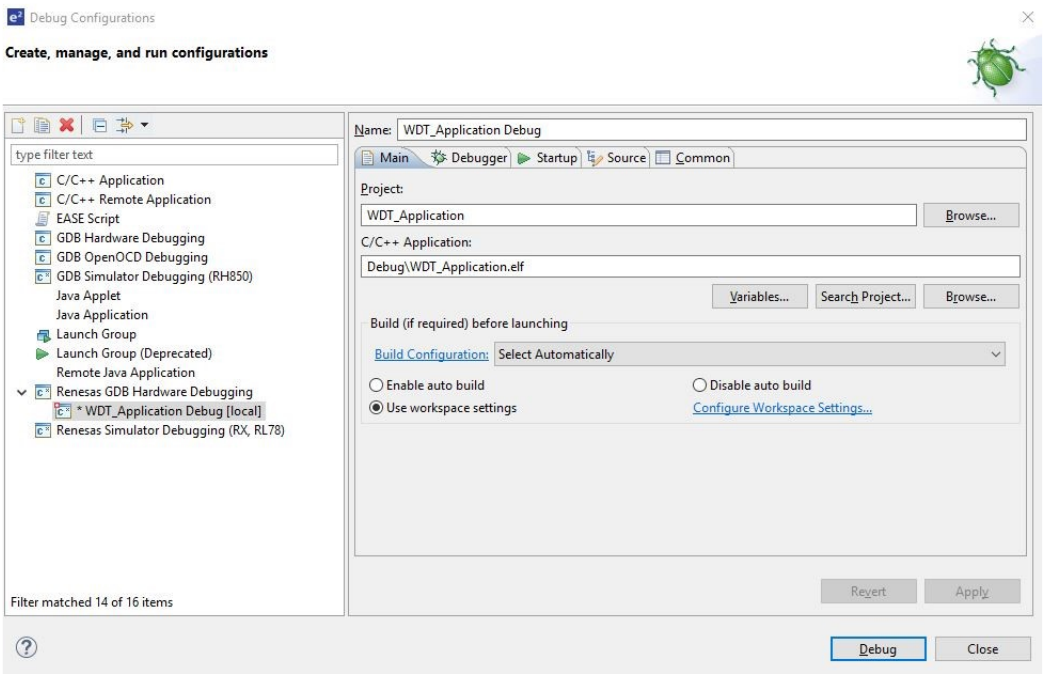
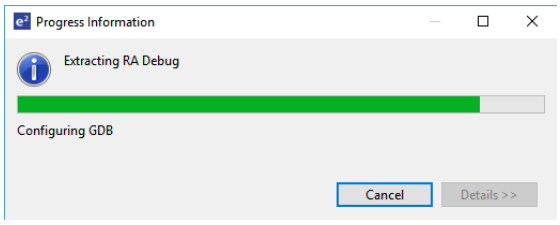



Figure 84: Debug configuration

4. Click the **Debug** button. Click Yes to the debug perspective if asked.




5. The code should run 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 e2 studio via **Run > Terminate**.
2. Click the reset button on the target board. The LEDs begin flashing.

# 2.5 RA SC User Guide for MDK and IAR

## 2.5.1 What is RA SC?

The Renesas RA Smart Configurator (RA SC) is a desktop application designed to configure device hardware such as clock set up and pin assignment as well as initialization of FSP software components for a Renesas RA microcontroller project when using a 3rd-party IDE and toolchain.

The RA Smart Configurator can currently be used with

1. Keil MDK and the ARM compiler toolchain.
2. IAR EWARM with IAR toolchain for ARM

Projects can be configured and the project content generated in the same way as in e2 studio. Please refer to Configuring a Project section for more details.

## 2.5.2 Using RA Smart Configurator with Keil MDK

### 2.5.2.1 Prerequisites

- Keil MDK and ARM compiler are installed and licensed. Please refer to the Release notes for the version to be installed.
- Import the RA device pack. Download the RA device pack archive file (ex: MDK_Device_Packs_x.x.x.zip) from the FSP GitHub release page. Extract the archive file to locate the RA device pack. To import the RA device pack, launch the PackInstaller.exe from <keil_mdk_install_dir>\UV4. Select the menu item **File > Import...** and browse to the extracted .pack file.
- Verify that the latest updates for RA devices are included in Keil MDK. To verify, select the

menu "Packs" in Pack Installer and verify that the menu item **Check for Updates on Launch** is selected. If not, select **Check for Updates on Launch** and relaunch Pack Installer.

- For flashing and debugging, the latest Segger J-Link DLL is installed into Keil MDK.
- Install RA SC and FSP using the Platform Installer from the GitHub release page.

### 2.5.2.2 Create new RA project

The following steps are required to create an RA project using Keil MDK, RA SC and FSP:

1. To create an RA project in Keil MDK, an example template needs to be copied from the Pack Installer. The Pack Installer can be launched by running PackInstaller.exe from <keil_mdk_install_dir>\UV4.

2. Select the device family or a device in the left pane of pack installer to filter the example templates in Examples tab in the right pane. The search bar in left pane helps to easily find a device. It is important to select the correct device and package type as this will be used by RA SC to configure pins.

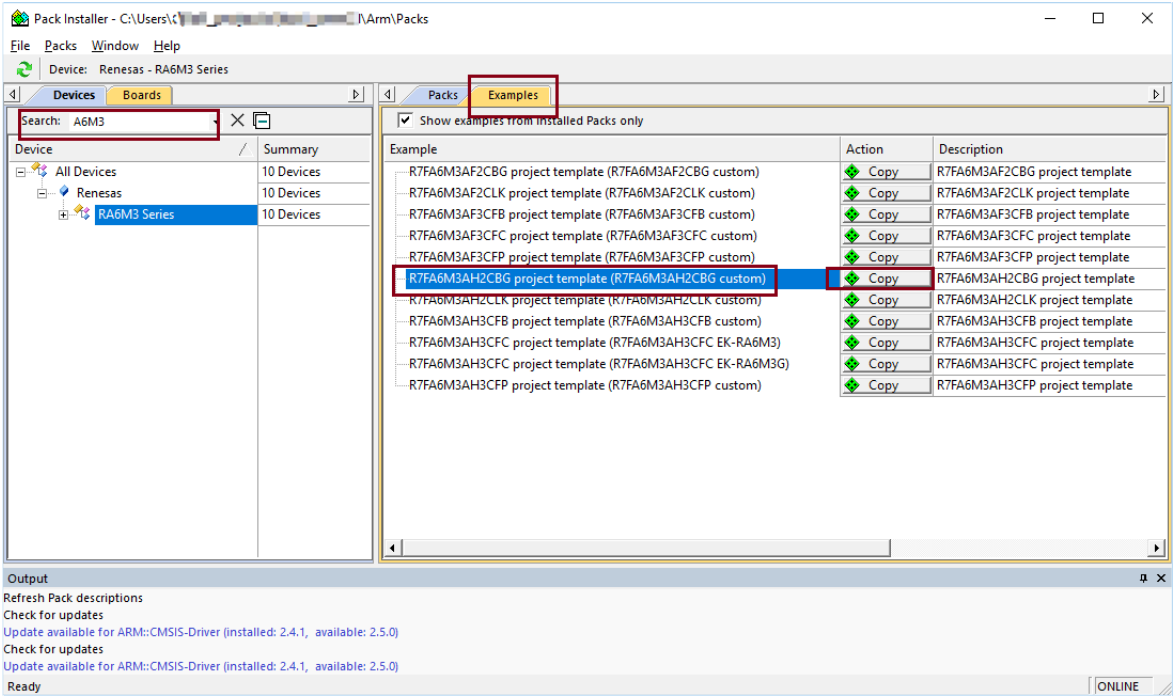



Figure 85: PackInstaller device example template

3. Click the **Copy** button for the example template to launch a dialog box and select where to copy the example project. The default project name will be the target device name.

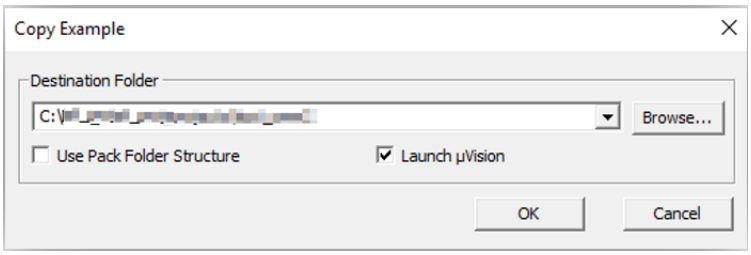



Figure 86: Copy Example dialog

Click **OK** to launch Keil uVision with the new project.

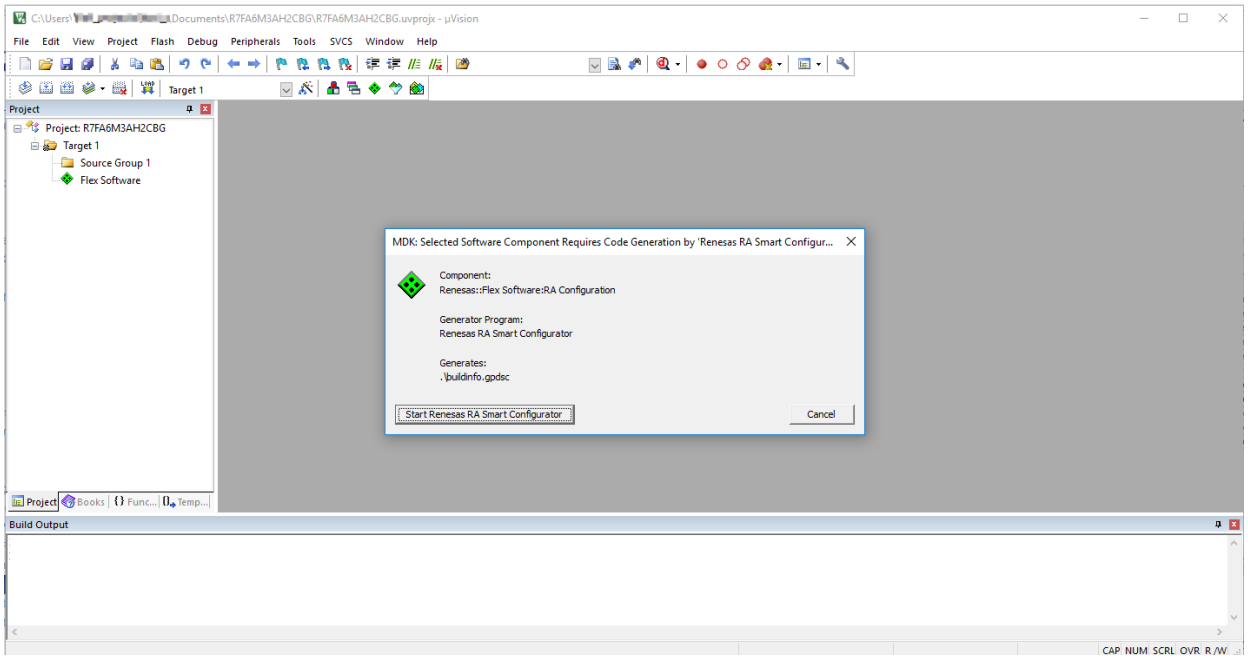



Figure 87: uVision

If the project name needs to be changed then deselect **Launch uVision_ in the Copy Example dialog and click __OK**. Follow project rename instructions here: http://www.keil.com/support/docs/3579.htm Once renamed, open the project using menu item **Project > Open Project...** in uVision and continue with steps in Modify existing RA project.

4. Click **Start Renesas RA Smart Configurator** to launch the RA Smart Configurator.

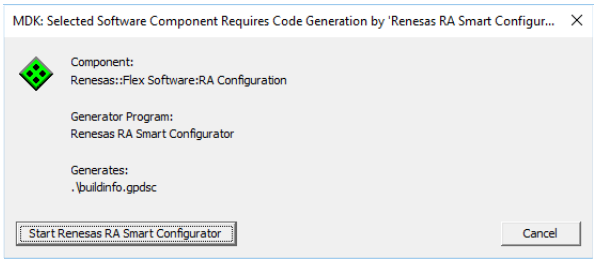



Figure 88: Launch RA SC confirmation dialog

5. If multiple versions of RA SC are installed, select the appropriate version of RA SC to run.

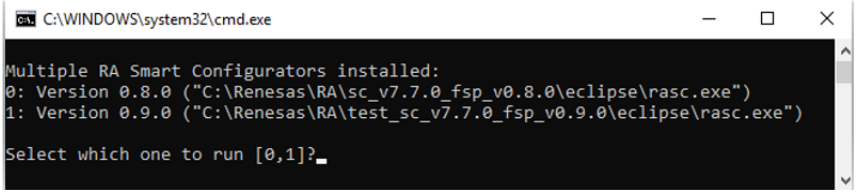



Figure 89: RA SC version selection

6. RA SC will be launched with project generator wizard.
7. The configuration window opens once the project wizard is closed. Refer to Configuring a Project for more details on how to configure the project.

8. After clicking **Generate Project Content** in the RA Smart Configurator, return to uVision. uVision offers a dialog to import the changes and updates to the project made in RA SC. Select **Yes** to import the updated project and the project is ready to build.

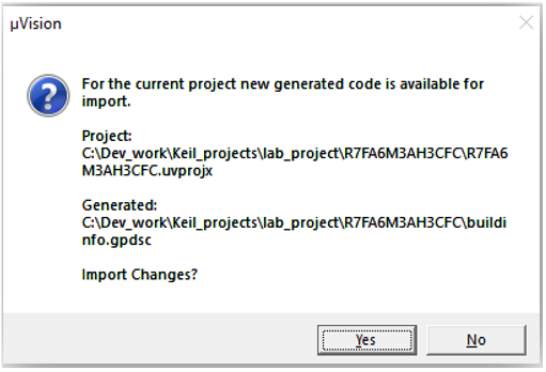



Figure 90: Import project data

RA SC will place the necessary FSP source code and header files into the project workspace. The folder structure is defined as below.

- Source Group 1 User source code should be added to the project in this folder
- Renesas RA Smart Configurator: Common Sources These source files are generated by RA Smart Configurator and can be edited as necessary

- Flex Software These are the source files from FSP and can be modified if needed. However, it is recommended NOT to edit these files as this may impact dependencies or functionality.

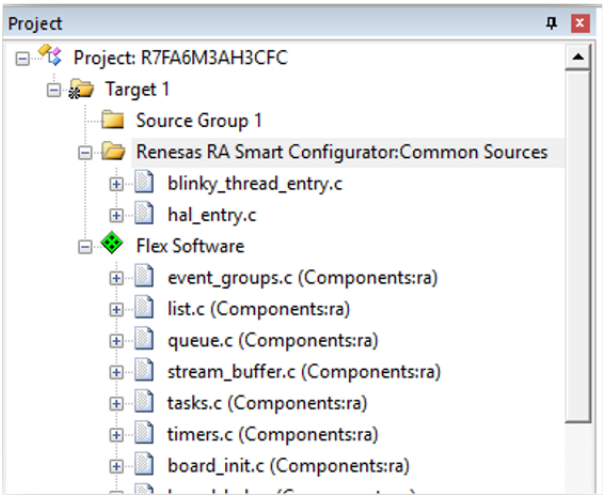



Figure 91: uVision project workspace with imported project data

### 2.5.2.3 Modify existing RA project

Once an initial project has been generated and configured, it is also possible to make changes using RA SC as follows:

1. If the desired project is not already open in uVision, the project can be opened using menu item **Project > Open project...** or selecting from the list of previous projects.
2. Select menu item **Project > Manage > Run-time Environment...** or tool bar button **Manage Run-Time Environment**.

3. Expand the **Flex Software** tree item in the dialog shown and click the green run button next to **RA Configuration**. This launches RA SC and the FSP project configuration can be modified and updated.

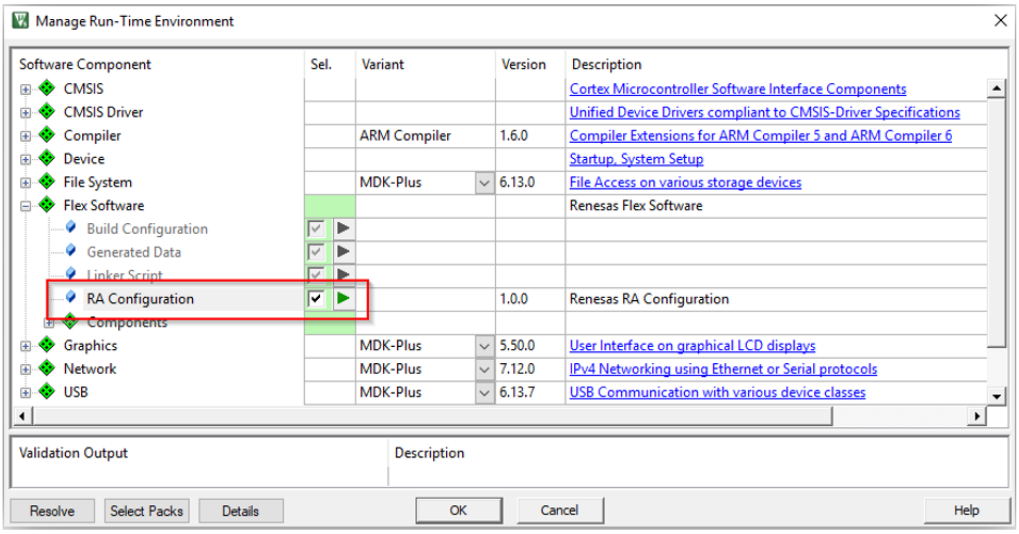



Figure 92: Manage run-time environment

### 2.5.2.4 Build and Debug RA project

The project can be built by selecting the menu item **Project > Build Target** or tool bar item **Rebuild** or the keyboard shortcut F7.

Assembler, Compiler, Linker and Debugger settings can be changed in **Options for Target** dialog, which can be launched using the menu item **Project > Options for Target**, the tool bar item **Options for Target** or the keyboard shortcut Alt+F7.

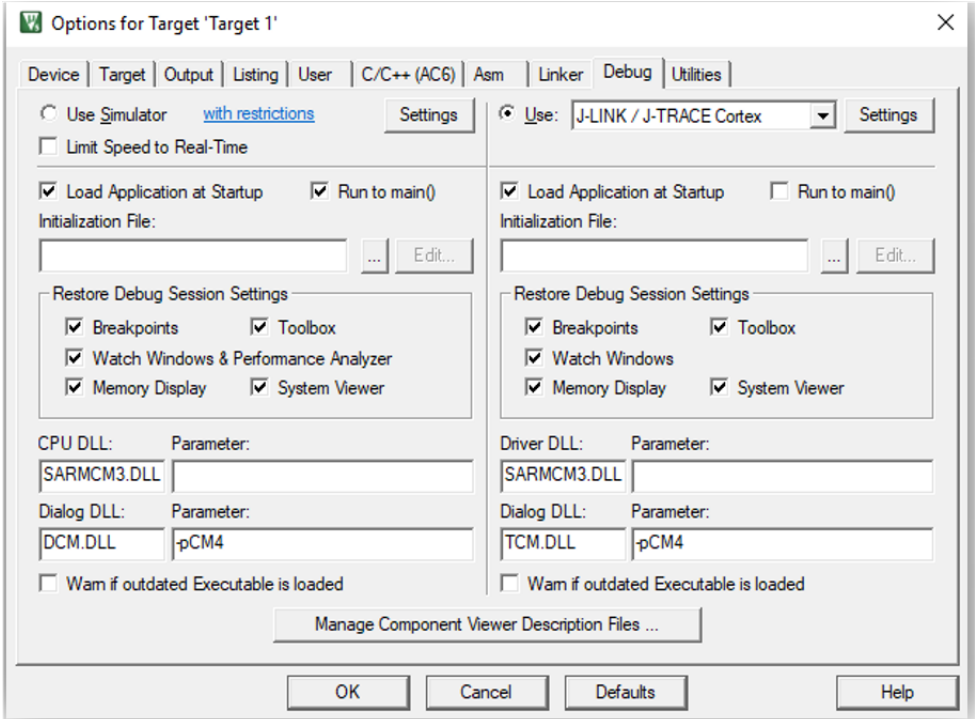



Figure 93: Options for Target

RA SC will set up the uVision project to debug the selected device using J-Link or J-Link OB debugger by default.

A Debug session can be started or stopped by selecting the menu item **Debug > Start/Stop Debug Session** or keyboard shortcut CTRL+F5. When debugging for the first time, J-Link firmware update may be needed if requested by the tool.

Refer to the documentation from Keil to get more information on the debug features in uVision. Note that not all features supported by uVision debugger are implemented in the J-Link interface. Consult SEGGER J-Link documentation for more information.

#### 2.5.2.5 Notes and Restrictions

1. When creating a new RA project, do not create a new project directly inside uVision. Follow the steps as mentioned in Create new RA project
2. RA FSP contains a full set of drivers and middleware and may not be compatible with other CMSIS packs from Keil, Arm or third parties.
3. Flash programming is currently only supported through the debugger connection.

### 2.5.3 Using RA Smart Configurator with IAR EWARM

IAR Systems Embedded Workbench for Arm (EWARM) includes support for Renesas RA devices.

These can be set up as bare metal designs within EWARM. However, most RA developers will want to integrate RA FSP drivers and middleware into their designs. RA SC will facilitate this.

RA SC generates a "Project Connection" file that can be loaded directly into EWARM to update project files.

#### 2.5.3.1 Prerequisites

- IAR EWARM installed and licensed. Pleae refer to the Release notes for the version to be installed.
- RA SC and FSP Installed

#### 2.5.3.2 Create new RA project

The following steps are required to create an RA project using IAR EWARM, RA SC and FSP:

1. To Use RA SC with EWARM, RA SC needs to configured as a tool in EWARM by selecting the menu item **Tools > Configure Tools...**. Select **New** to create a new tool in the dialog shown and add the following information:

    - Menu Text: RA Smart Configurator
    - Command: Select Browse... and navigate to rasc.exe in the installed RA SC
    - Argument: –compiler IAR configuration.xml
    - Initial Directory: $PROJ_DIR$
    - Tool Available: Always

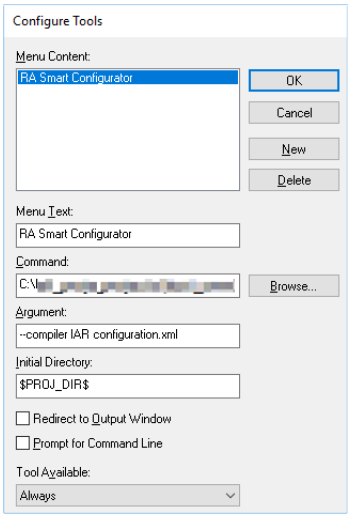



Figure 94: Tool_setup

2. A new EWARM project can be created using the menu item **Project > Create New Project...** and selecting the **Empty Project** and toolchain as ARM. Save the project to an empty folder.

3. RA SC can now be launched from EWARM using the menu item **Tools > RA Smart Configurator**.

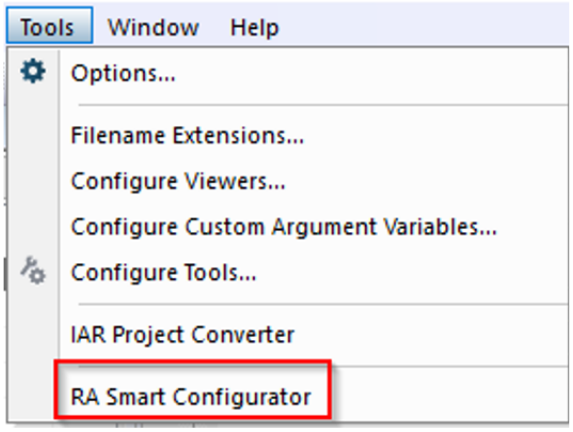



Figure 95: RA SC Menu Item

RA SC will be launched with project generator wizard. The configuration window opens once the project wizard is closed. Refer to Configuring a Project for more details on how to configure the project. After configuring the project, click **Generate Project Content**. Changes to the RA configuration will be reflected in the EWARM project.

4. A Project connection needs to be set up in EWARM to build the project. Select **Project > Add Project Connection** in EWARM and select **IAR Project Connection**. Navigate to the project folder and select buildinfo.ipcf and click open. The project can now build in EWARM.

# Chapter 3 FSP Architecture

## 3.1 FSP Architecture Overview

This guide describes the Renesas Flexible Software Package (FSP) architecture and how to use the FSP Application Programming Interface (API).

### 3.1.1 C99 Use

The FSP 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.

### 3.1.2 Doxygen

Doxygen is the default documentation tool used by FSP. You can find Doxygen comments throughout the FSP source.

### 3.1.3 Weak Symbols

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

### 3.1.4 Memory Allocation

Dynamic memory allocation through use of the malloc() and free() functions are not used in FSP modules; all memory required by FSP modules is allocated in the application and passed to the module in a pointer. Exceptions are considered only for ports of 3rd party code that require dynamic memory.

### 3.1.5 FSP Terms

| Term | Description | Reference |
|---|---|---|
| BSP | Short for Board Support Package. In the FSP the BSP provides just enough foundation to allow other FSP modules to work together without issue. | MCU Board Support Package |

| | | |
|---|---|---|
| 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. Applications can be built by combining multiple modules to provide the user with the features they need. | FSP Modules |
| Driver | A driver is a specific kind of module that directly modifies registers on the MCU. | - |
| Interface | An interface contains API definitions that can be shared by modules with similar features. Interfaces are definitions only and do not add to code size. | FSP Interfaces |
| Stacks | The FSP architecture is designed such that modules work together to form a stack. A stack consists of a top level module and all its dependencies. | FSP Stacks |
| Module Instance | Single and independent instantiation of a module. An application may require two GPT timers. Each of these timers is a module instance of the r_gpt module. | - |
| Application | Code that is owned and maintained by the user. Application code may be based on sample application code provided by Renesas, but it is the responsibility of the user to maintain as necessary. | - |

| | | |
|---|---|---|
| Callback Function | This term refers to a function that is called when an event occurs. As an example, suppose the user would like to be notified every second based on the RTC. As part of the RTC configuration, a callback function can be supplied that will be jumped to during each RTC interrupt. When a single callback services multiple events, the arguments contain the triggering event. Callback functions for interrupts should be kept short and handled carefully because when they are called the MCU is still inside of an interrupt, delaying any pending interrupts. | - |

# 3.2 FSP Modules

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

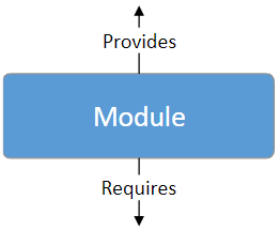



Figure 96: Modules

The amount of functionality provided by a module is determined based on functional use cases. Common functionality required by multiple modules is often placed into a self-contained submodule so it can be reused. Code size, speed and complexity are also considered when defining a module.

The simplest FSP application consists of one module with the Board Support Package (BSP) and the user application on top.

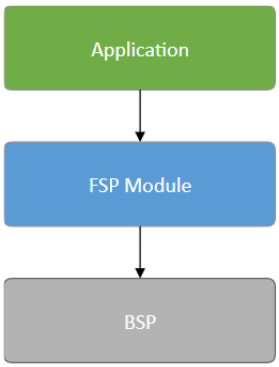



Figure 97: Module with application

The Board Support Package (BSP) is the foundation for FSP modules, providing functionality to determine the MCU used as well as configuring clocks, interrupts and pins. For the sake of clarity, the BSP will be omitted from further diagrams.

## 3.3 FSP Stacks

When modules are layered atop one another, an FSP stack is formed. The stacking process is performed by matching what one module provides with what another module requires. For example, the SPI module (Serial Peripheral Interface (r_spi)) requires a module that provides the transfer interface (Transfer Interface) to send or receive data without a CPU interrupt. The transfer interface requirement can be fulfilled by the DTC driver module (Data Transfer Controller (r_dtc)).

Through this methodology the same code can be shared by several modules simultaneously. The example below illustrates how the same DTC module can be used with SPI (Serial Peripheral Interface (r_spi)), UART (Serial Communications Interface (SCI) UART (r_sci_uart)) and SDHI (SD/MMC Host Interface (r_sdhi)).

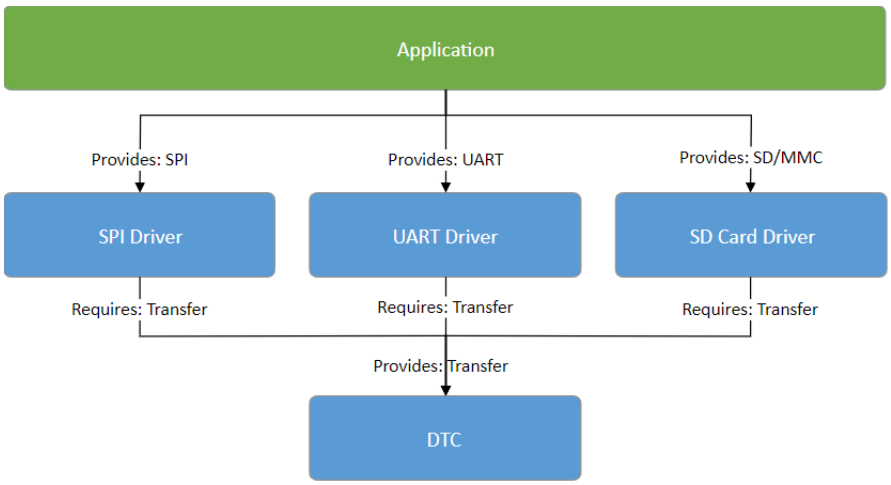



Figure 98: Stacks -- Shared DTC Module

The ability to stack modules ensures the flexibility of the architecture as a whole. If multiple modules include the same functionality issues arise when application features must work across different user designs. To ensure that modules are reusable, any dependent modules must be capable of being swapped out for other modules that provide the same features. The FSP

architecture provides this flexibility to swap modules in and out through the use of FSP interfaces.

# 3.4 FSP 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 established in the contract.

On RA 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. However, 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 capabilities of the fully-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 IIC) might not be available in the interface. In most cases these features are still available through interface extensions.

In FSP design, interfaces are defined in header files. All interface header files are located in the folder ra/fsp/inc/api and end with *_api.h. Interface extensions are defined in header files in the folder ra/fsp/inc/instances. The following sections detail what makes up an interface.

## 3.4.1 FSP Interface Enumerations

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

```
typedef enum e_i2c_master_addr_mode
{
    I2C_MASTER_ADDR_MODE_7BIT = 1,    ///< Use 7-bit addressing mode
    I2C_MASTER_ADDR_MODE_10BIT = 2,    ///< Use 10-bit addressing mode
} i2c_master_addr_mode_t;
```

Enumerations remove uncertainty when deciding what values are available for a parameter. FSP 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 e2 studio (Ctrl + Space) provides the benefits of rapid coding while maintaining high readability.

## 3.4.2 FSP Interface Callback Functions

Callback functions allow modules to asynchronously alert the user application when an event has occurred, such as when a byte has been received over a UART channel or an IRQ pin is toggled. FSP driver modules define and handle the interrupt service routines for RA MCU peripherals to ensure any required hardware procedures are implemented. The interrupt service routines in FSP modules then call the user-defined callbacks to allow the application to respond.

Callback functions must be defined in the user application. They always return void and take a structure for their one parameter. The structure is defined 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 an enumeration defined in the interface used by the application to determine why the callback was called. Using the UART example, the callback could be triggered for many different reasons, including when a byte is received, all bytes have been transmitted, or a framing error has occurred. The event member allows the application to determine which of these three events has occurred and handle it appropriately.

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 determine which module instance the callback is for. For example, if the callback wanted to make an FSP API call in the callback, then at a minimum the callback will need a reference to the relevant 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 passed in the p_context element of 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.

```
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;
      }
case FLASH_EVENT_ERR_FAILURE:
     {
/* Handle error. */
break;
      }
case FLASH_EVENT_ERR_ONE_BIT:
     {
/* Handle error. */
break;
      }
   }
}
```

When a module is not directly used in the user application (that is, it is not the top layer of the stack), its callback function will be handled by the module above. For example, if a module requires

a UART interface module the upper layer module will control and use the UART's callback function. In this case the user would not need to create a callback function for the UART module in their application code.

## 3.4.3 FSP Interface Data Structures

At a minimum, all FSP interfaces include three data structures: a configuration structure, an API structure, and an instance structure.

### 3.4.3.1 FSP Interface Configuration Structure

The configuration structure is used for the initial configuration of a module during the <MODULE>_Open() call. The structure consists of members such as channel number, bitrate, and operating mode.

The configuration structure is used purely as an input into the module. It may be stored and referenced by the module, so the configuration structure and anything it references must persist as long as the module is open.

The configuration structure is allocated for each module instance in files generated by the RA Configuration editor.

When FSP stacks are used, it is also important to understand 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 defined in the UART module instance. Any modules that use the UART interface rely on the baud rate being provided in the UART module instance and do not offer it in their own configuration structures.

### 3.4.3.2 FSP Interface API Structure

All interfaces include an API structure which contains function pointers for all the supported interface functions. An example structure for the Digital to Analog Converter (r_dac) is shown below.

```
typedef struct st_dac_api
{
    /** Initial configuration.
    * @par Implemented as
    * - @ref R_DAC_Open()
    * - @ref R_DAC8_Open()
    *
    * @param[in] p_ctrl Pointer to control block. Must be declared by user. Elements
set here.
    * @param[in] p_cfg Pointer to configuration structure. All elements of this
structure must be set by user.
    */
    fsp_err_t (* open)(dac_ctrl_t * const p_ctrl, dac_cfg_t const * const p_cfg);
```

```
    /** Close the D/A Converter.
    * @par Implemented as
    * - @ref R_DAC_Close()
    * - @ref R_DAC8_Close()
    *
    * @param[in] p_ctrl Control block set in @ref dac_api_t::open call for this
timer.
    */
    fsp_err_t (* close)(dac_ctrl_t * const p_ctrl);
    /** Write sample value to the D/A Converter.
    * @par Implemented as
    * - @ref R_DAC_Write()
    * - @ref R_DAC8_Write()
    *
    * @param[in] p_ctrl Control block set in @ref dac_api_t::open call for this
timer.
    * @param[in] value Sample value to be written to the D/A Converter.
    */
    fsp_err_t (* write)(dac_ctrl_t * const p_ctrl, uint16_t value);
    /** Start the D/A Converter if it has not been started yet.
    * @par Implemented as
    * - @ref R_DAC_Start()
    * - @ref R_DAC8_Start()
    *
    * @param[in] p_ctrl Control block set in @ref dac_api_t::open call for this
timer.
    */
    fsp_err_t (* start)(dac_ctrl_t * const p_ctrl);
    /** Stop the D/A Converter if the converter is running.
    * @par Implemented as
    * - @ref R_DAC_Stop()
    * - @ref R_DAC8_Stop()
    *
    * @param[in] p_ctrl Control block set in @ref dac_api_t::open call for this
```

```
timer.
    */
    fsp_err_t (* stop)(dac_ctrl_t * const p_ctrl);
    /** Get version and store it in provided pointer p_version.
    * @par Implemented as
    * - @ref R_DAC_VersionGet()
    * - @ref R_DAC8_VersionGet()
    *
    * @param[out] p_version Code and API version used.
    */
    fsp_err_t (* versionGet)(fsp_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.

RA MCUs have an internal DAC peripheral. If the DAC API structure in the DAC interface is not used the application can 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.

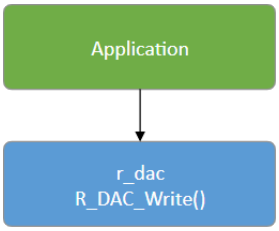



Figure 99: DAC Write example

Now let's assume that the user needs more DAC channels than are available on the MCU and decides to add an external DAC module named dac_external using I2C for communications. The application must now distinguish between the two modules, adding complexity and further dependencies to the application.

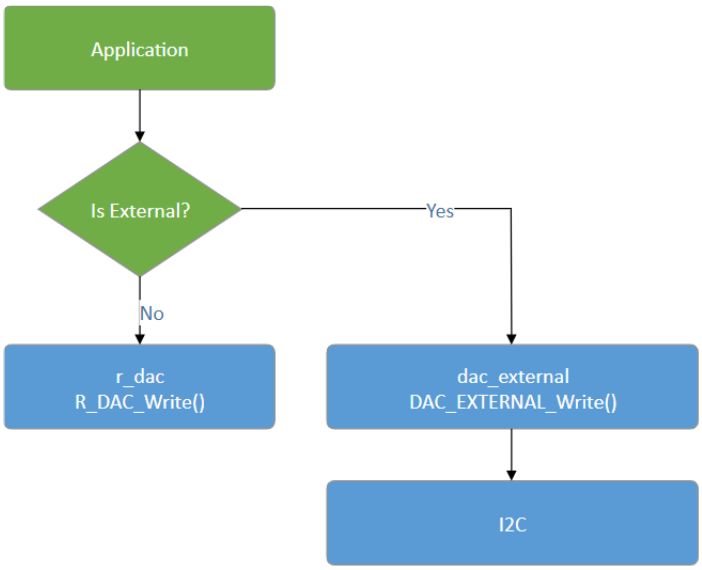



Figure 100: DAC Write with two write modules

The use of interfaces and the API structure allows for the use of an abstracted DAC. This means that no extra logic is needed if the user's dac_external module implements the FSP DAC interface, so the application no longer depends upon hard-coded module function names. Instead the application now depends on the DAC interface API which can be implemented by any number of modules.

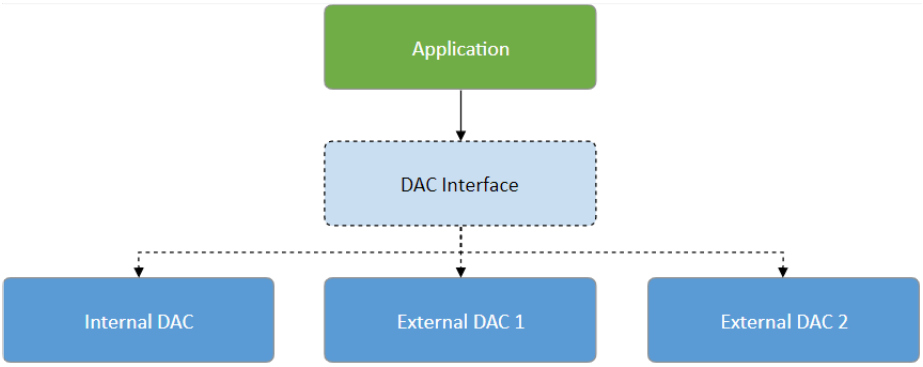



Figure 101: DAC Interface

### 3.4.3.3 FSP Interface Instance Structure

Every FSP interface also has an instance structure. The instance structure encapsulates everything required to use the module:

- A pointer to the instance API structure (FSP Instance API)
- A pointer to the configuration structure
- A pointer to the control structure

The instance structure is not required at the application layer. It is used to connect modules to their dependencies (other than the BSP).

Instance structures have a standardized name of <interface>_instance_t. An example from the Transfer Interface is shown below.

```
typedef struct st_transfer_instance
{
  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
} transfer_instance_t;
```

Note that when an instance structure variable is declared, the API is the only thing that is instance specific, not *module instance* specific. This is because all module instances of the same module share the same underlying module source code. If SPI is being used on SCI channels 0 and 2 then both module instances use the same API while the configuration and control structures are typically different.

# 3.5 FSP 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 an application that uses an interface to swap out the instance when needed.

On RA MCUs some peripherals are used to implement multiple interfaces. In the example below the IIC and SPI peripherals map to only one interface each while the SCI peripheral implements three interfaces.

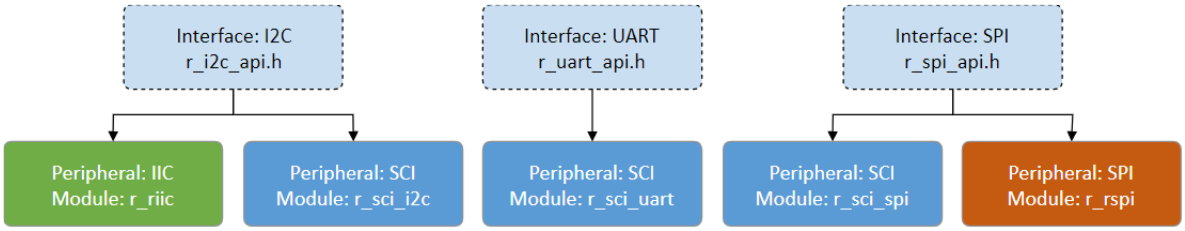



Figure 102: Instances

In FSP design, instances consist of the interface extension and API defined in the instance header file located in the folder ra/fsp/inc/instances and the module source ra/fsp/src/<module>.

## 3.5.1 FSP Instance Control Structure

The control structure is used as a unique identifier for the module instance and contains memory required by the module. Elements in the control structure are owned by the module and *must not be modified* by the application. The user allocates storage for a control structure, often as a global variable, then sends a pointer to it into the <MODULE>_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.

### 3.5.2 FSP Interface Extensions

In some cases, instances require more information than is provided in the interface. This situation can occur in the following cases:

- An instance offers extra features that are not common to most instances of the interface. An example of this is the start source selection of the GPT (General PWM Timer (r_gpt)). The GPT can be configured to start based on hardware events such as a falling edge on a trigger pin. This feature is not common to all timers, so it is included in the GPT instance.
- An interface must be very generic out of necessity. As an interface becomes more generic, the number of possible instances increases. An example of an interface that must be generic is a block media interface that abstracts functions required by a file system. Possible instances include SD card, SPI Flash, SDRAM, USB, and many more.

The p_extend member provides this extension function.

Use of interface extensions is not always necessary. Some instances do not offer an extension since all functionality is provided in the interface. In these cases the p_extend member can be set to NULL. The documentation for each instance indicates whether an interface extension is available and whether it is mandatory or optional.

#### 3.5.2.1 FSP Extended Configuration Structure

When extended configuration is required it can be supplied through the p_extend parameter of the interface configuration structure.

The extended configuration structure is part of the instance, but it is also still considered to be part of the configuration structure. All usage notes about the configuration structure described in FSP Interface Configuration Structure apply to the extended configuration structure as well.

The extended configuration structure and all typed structures and enumerations required to define it make up the interface extension.

### 3.5.3 FSP Instance API

Each instance includes a constant global variable tying the interface API functions to the functions provided by the module. 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 instance header file (r_spi.h, r_dtc.h, and r_adc.h respectively).

## 3.6 FSP API Standards

### 3.6.1 FSP Function Names

FSP functions start with the uppercase module name (<MODULE>). All modules have <MODULE>_Open() and <MODULE>_Close() functions. The <MODULE>_Open() function must be called before any of the other functions. The only exception is the <MODULE>_VersionGet() function which is not dependent upon any user provided information.

Other functions that will commonly be found are <MODULE>_Read(), <MODULE>_Write(),

<MODULE>_InfoGet(), and <MODULE>_StatusGet(). The <MODULE>_StatusGet() function provides a status that could change asynchronously, while <MODULE>_InfoGet() provides information that cannot change after open or can only be updated by API calls. Example function names include:

- R_SPI_Read(), R_SPI_Write(), R_SPI_WriteRead()
- R_SDHI_StatusGet()
- R_RTC_CalendarAlarmSet(), R_RTC_CalendarAlarmGet()
- R_FLASH_HP_AccessWindowSet(), R_FLASH_HP_AccessWindowClear()

### 3.6.2 Use of const in API parameters

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

```
fsp_err_t R_FLASH_HP_Open(flash_ctrl_t * const p_api_ctrl, flash_cfg_t const * const
p_cfg);
```

In this example, flash_cfg_t is a structure of configuration parameters for the r_flash_hp module. The parameter p_cfg is a pointer to this structure. The first const qualifier on p_cfg ensures the flash_cfg_t structure cannot be modified by R_FLASH_HP_Open(). This allows the structure to be allocated as a const variable and stored in ROM instead of RAM.

The const qualifier after the pointer star for both p_ctrl and p_cfg ensures the FSP function does not modify the input pointer addresses. While not fool-proof by any means this does provide some extra checking inside the FSP code to ensure that arguments that should not be altered are treated as such.

### 3.6.3 FSP Version Information

All instances supply a <MODULE>_VersionGet() function which fills in a structure of type fsp_version_t. This structure is made up of two version numbers: one for the interface (the API) and one for the underlying instance that is currently being used.

```
typedef union st_fsp_version
{
    /** Version id */
    uint32_t version_id;
    /** Code version parameters */
    struct
    {
       uint8_t code_version_minor;    ///< Code minor version
       uint8_t code_version_major;    ///< Code major version
       uint8_t api_version_minor;    ///< API minor version
       uint8_t api_version_major;    ///< API major version
    };
```

```
} fsp_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 due to bug fixes, enhancements, and additional features. Changes to the code version typically do not require changes to user code.

# 3.7 FSP Build Time Configurations

All modules have a build-time configuration header file. Most configuration options are supplied at run time, though 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 a build time option to enable or disable parameter checking for the module. FSP modules check function arguments for validity when possible, though this feature is disabled by default to reduce code size. Enabling it can help catch parameter errors during development and debugging. By default, each module's parameter checking configuration inherits the BSP parameter checking setting (set on the BSP tab of the RA Configuration editor). Leaving each module's parameter checking configuration set to Default (BSP) allows parameter checking to be enabled or disabled globally in all FSP code through the parameter checking setting on the BSP tab.

If an error condition can reasonably be avoided it is only checked in a section of code that can be disabled by disabling parameter checking. Most FSP APIs can only return FSP_SUCCESS if parameter checking is disabled. An example of an error that cannot be reasonably avoided is the "bus busy" error that occurs when another master is using an I2C bus. This type of error can be returned even if parameter checking is disabled.

# 3.8 FSP File Structure

The high-level file structure of an FSP project is shown below.

```
ra_gen
ra
+---fsp
    +---inc
    |   +---api
    |   \---instances
    \---src
       +---bsp
       \---r_module
ra_cfg
+---fsp_cfg
    +---bsp
```

```
    +---driver
```

Directly underneath the base ra 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 ra_gen folder contains code generated by the RA Configuration editor. This includes global variables for the control structure and configuration structure for each module.

The ra_cfg folder is where configuration header files are stored for each module. See FSP Build Time Configurations for information on what is provided in these header files.

# 3.9 FSP Architecture in Practice

## 3.9.1 FSP Connecting Layers

FSP 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.

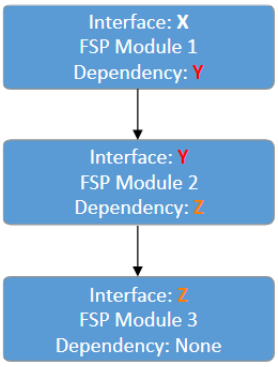



Figure 103: Connecting layers

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.

## 3.9.2 Using FSP Modules in an Application

The typical use of an FSP module involves generating required module data then using the API in the application.

### 3.9.2.1 Create a Module Instance in the RA Configuration Editor

The RA Configuration editor (available both in the Renesas e2 studio IDE as well as through the standalone RA Smart Configurator) provides a graphical user interface for setting the parameters of the interface and instance configuration structures. It also automatically includes those structures (once they are configured in the GUI) in application-specific header files that can be included in application code.

The RA Configuration editor allocates storage for the control structures, all required configuration structures, and the instance structure in generated files in the ra_gen folder. Use the **Properties** window to set the values for the members of the configuration structures as needed. Refer to the Configuration section of the module usage notes for documentation about the configuration options.

If the interface has a callback function option then the application must declare and define the function. 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. Callback function names can be assigned through the **Properties** window for the selected module.

#### 3.9.2.2 Use the Instance API in the Application

Call the module's <MODULE>_Open() function. Pass pointers to the generated control structure and configuration structure. The names of these structures are based on the 'Name' field provided in the configuration editor. The control structure is <Name>_ctrl and the configuration structure is <Name>_cfg. An example <MODULE>_Open() call for an r_rtc module instance named g_clock is:

```
R_RTC_Open(&g_clock_ctrl, &g_clock_cfg);
```

*Note*

*Each layer in the FSP Stack is responsible for calling the API functions of its dependencies. This means that users are only responsible for calling the API functions at the layer at which they are interfacing. Using the example above of a SPI module with a DTC dependency, the application uses only SPI APIs. The application starts by calling R_SPI_Open(). Internally, the SPI module opens the DTC. It locates R_DTC_Open() by accessing the dependent transfer interface function pointers from the pointers DTC instances (spi_cfg_t::p_transfer_tx and spi_cfg_t::p_transfer_rx) to open the DTC.*

Refer to the module usage notes for example code to help get started with any particular module.

# Chapter 4 API Reference

This section includes the FSP API Reference for the Module and Interface level functions.

| | |
|---|---|
| ►BSP | Common code shared by FSP drivers |
| ►Modules | Modules are the smallest unit of software available in the FSP. Each module implements one interface |
| ►Interfaces | The FSP interfaces provide APIs for common functionality. They can be implemented by one or more modules. Modules can use other modules as dependencies using this interface layer |

## 4.1 BSP

### Detailed Description

Common code shared by FSP drivers.

### Modules

| | |
|---|---|
| | Common Error Codes |
| | MCU Board Support Package<br>The BSP is responsible for getting the MCU from reset to the user's application. Before reaching the user's application, the BSP sets up the stacks, heap, clocks, interrupts, C runtime environment, and stack monitor. |
| | BSP I/O access<br>This module provides basic read/write access to port pins. |

### Data Structures

| | |
|---|---|
| union | fsp_pack_version_t |
| struct | fsp_pack_version_t.__unnamed__ |

### Macros

| | |
|---|---|
| #define | FSP_VERSION_MAJOR |
| #define | FSP_VERSION_MINOR |
| #define | FSP_VERSION_PATCH |
| #define | FSP_VERSION_BUILD |
| #define | FSP_VERSION_STRING |
| #define | FSP_VERSION_BUILD_STRING |

## Data Structure Documentation

### ◆ fsp_pack_version_t

| union fsp_pack_version_t | | |
|---|---|---|
| FSP Pack version structure | | |
| Data Fields | | |
| uint32_t | version_id | Version id |
| struct fsp_pack_version_t | __unnamed__ | Code version parameters, little endian order. |

### ◆ fsp_pack_version_t.__unnamed__

| struct fsp_pack_version_t.__unnamed__ | | |
|---|---|---|
| Code version parameters, little endian order. | | |
| Data Fields | | |
| uint8_t | build | Build version of FSP Pack. |
| uint8_t | patch | Patch version of FSP Pack. |
| uint8_t | minor | Minor version of FSP Pack. |
| uint8_t | major | Major version of FSP Pack. |

## Macro Definition Documentation

### ◆ FSP_VERSION_MAJOR

| #define FSP_VERSION_MAJOR |
|---|
| FSP pack major version. |

### ◆ FSP_VERSION_MINOR

| #define FSP_VERSION_MINOR |
|---|
| FSP pack minor version. |

### ◆ FSP_VERSION_PATCH

| #define FSP_VERSION_PATCH |
|---|
| FSP pack patch version. |

### ◆ FSP_VERSION_BUILD

| #define FSP_VERSION_BUILD |
|---|
| FSP pack version build number (currently unused). |

### ◆ FSP_VERSION_STRING

| #define FSP_VERSION_STRING |
|---|
| Public FSP version name. |

### ◆ FSP_VERSION_BUILD_STRING

| #define FSP_VERSION_BUILD_STRING |
|---|
| Unique FSP version ID. |

## 4.1.1 Common Error Codes

BSP

#### Detailed Description

All FSP modules share these common error codes.

#### Data Structures

| | |
|---|---|
| union | fsp_version_t |
| struct | fsp_version_t.__unnamed__ |

### Macros

| | |
|---|---|
| #define | FSP_PARAMETER_NOT_USED(p) |
| #define | FSP_CPP_HEADER |
| #define | FSP_HEADER |

### Enumerations

| | |
|---|---|
| enum | fsp_err_t |

## Data Structure Documentation

### ◆ fsp_version_t

| union fsp_version_t | | |
|---|---|---|
| Common version structure | | |
| Data Fields | | |
| uint32_t | version_id | Version id |
| struct fsp_version_t | __unnamed__ | Code version parameters |

### ◆ fsp_version_t.__unnamed__

| struct fsp_version_t.__unnamed__ | | |
|---|---|---|
| Code version parameters | | |
| Data Fields | | |
| uint8_t | code_version_minor | Code minor version. |
| uint8_t | code_version_major | Code major version. |
| uint8_t | api_version_minor | API minor version. |
| uint8_t | api_version_major | API major version. |

## Macro Definition Documentation

### ◆ FSP_PARAMETER_NOT_USED

| #define FSP_PARAMETER_NOT_USED ( 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. |

### ◆ FSP_CPP_HEADER

| #define FSP_CPP_HEADER |
|---|
| Determine if a C++ compiler is being used. If so, ensure that standard C is used to process the API information. |

### ◆ FSP_HEADER

| #define FSP_HEADER |
|---|
| FSP Header and Footer definitions |

## Enumeration Type Documentation

### ◆ fsp_err_t

| enum fsp_err_t | |
|---|---|
| Common error codes | |
| Enumerator | |
| FSP_ERR_ASSERTION | A critical assertion has failed. |
| FSP_ERR_INVALID_POINTER | Pointer points to invalid memory location. |
| FSP_ERR_INVALID_ARGUMENT | Invalid input parameter. |
| FSP_ERR_INVALID_CHANNEL | Selected channel does not exist. |
| FSP_ERR_INVALID_MODE | Unsupported or incorrect mode. |
| FSP_ERR_UNSUPPORTED | Selected mode not supported by this API. |
| FSP_ERR_NOT_OPEN | Requested channel is not configured or API not open. |
| FSP_ERR_IN_USE | Channel/peripheral is running/busy. |
| FSP_ERR_OUT_OF_MEMORY | Allocate more memory in the driver's cfg.h. |
| FSP_ERR_HW_LOCKED | Hardware is locked. |
| FSP_ERR_IRQ_BSP_DISABLED | IRQ not enabled in BSP. |
| FSP_ERR_OVERFLOW | Hardware overflow. |
| FSP_ERR_UNDERFLOW | |

| | |
|---|---|
| | Hardware underflow. |
| FSP_ERR_ALREADY_OPEN | Requested channel is already open in a different configuration. |
| FSP_ERR_APPROXIMATION | Could not set value to exact result. |
| FSP_ERR_CLAMPED | Value had to be limited for some reason. |
| FSP_ERR_INVALID_RATE | Selected rate could not be met. |
| FSP_ERR_ABORTED | An operation was aborted. |
| FSP_ERR_NOT_ENABLED | Requested operation is not enabled. |
| FSP_ERR_TIMEOUT | Timeout error. |
| FSP_ERR_INVALID_BLOCKS | Invalid number of blocks supplied. |
| FSP_ERR_INVALID_ADDRESS | Invalid address supplied. |
| FSP_ERR_INVALID_SIZE | Invalid size/length supplied for operation. |
| FSP_ERR_WRITE_FAILED | Write operation failed. |
| FSP_ERR_ERASE_FAILED | Erase operation failed. |
| FSP_ERR_INVALID_CALL | Invalid function call is made. |
| FSP_ERR_INVALID_HW_CONDITION | Detected hardware is in invalid condition. |
| FSP_ERR_INVALID_FACTORY_FLASH | Factory flash is not available on this MCU. |
| FSP_ERR_INVALID_STATE | API or command not valid in the current state. |
| FSP_ERR_NOT_ERASED | Erase verification failed. |
| FSP_ERR_SECTOR_RELEASE_FAILED | Sector release failed. |
| FSP_ERR_NOT_INITIALIZED | Required initialization not complete. |
| FSP_ERR_INTERNAL | Internal error. |
| FSP_ERR_WAIT_ABORTED | Wait aborted. |
| FSP_ERR_FRAMING | Framing error occurs. |
| FSP_ERR_BREAK_DETECT | Break signal detects. |

| | |
|---|---|
| FSP_ERR_PARITY | Parity error occurs. |
| FSP_ERR_RXBUF_OVERFLOW | Receive queue overflow. |
| FSP_ERR_QUEUE_UNAVAILABLE | Can't open s/w queue. |
| FSP_ERR_INSUFFICIENT_SPACE | Not enough space in transmission circular buffer. |
| FSP_ERR_INSUFFICIENT_DATA | Not enough data in receive circular buffer. |
| FSP_ERR_TRANSFER_ABORTED | The data transfer was aborted. |
| FSP_ERR_MODE_FAULT | Mode fault error. |
| FSP_ERR_READ_OVERFLOW | Read overflow. |
| FSP_ERR_SPI_PARITY | Parity error. |
| FSP_ERR_OVERRUN | Overrun error. |
| FSP_ERR_CLOCK_INACTIVE | Inactive clock specified as system clock. |
| FSP_ERR_CLOCK_ACTIVE | Active clock source cannot be modified without stopping first. |
| FSP_ERR_NOT_STABILIZED | Clock has not stabilized after its been turned on/off. |
| FSP_ERR_PLL_SRC_INACTIVE | PLL initialization attempted when PLL source is turned off. |
| FSP_ERR_OSC_STOP_DET_ENABLED | Illegal attempt to stop LOCO when Oscillation stop is enabled. |
| FSP_ERR_OSC_STOP_DETECTED | The Oscillation stop detection status flag is set. |
| FSP_ERR_OSC_STOP_CLOCK_ACTIVE | Attempt to clear Oscillation Stop Detect Status with PLL/MAIN_OSC active. |
| FSP_ERR_CLKOUT_EXCEEDED | Output on target output clock pin exceeds maximum supported limit. |
| FSP_ERR_USB_MODULE_ENABLED | USB clock configure request with USB Module enabled. |
| FSP_ERR_HARDWARE_TIMEOUT | A register read or write timed out. |

| | |
|---|---|
| FSP_ERR_LOW_VOLTAGE_MODE | Invalid clock setting attempted in low voltage mode. |
| FSP_ERR_PE_FAILURE | Unable to enter Programming mode. |
| FSP_ERR_CMD_LOCKED | Peripheral in command locked state. |
| FSP_ERR_FCLK | FCLK must be >= 4 MHz. |
| FSP_ERR_INVALID_LINKED_ADDRESS | Function or data are linked at an invalid region of memory. |
| FSP_ERR_BLANK_CHECK_FAILED | Blank check operation failed. |
| FSP_ERR_INVALID_CAC_REF_CLOCK | Measured clock rate < reference clock rate. |
| FSP_ERR_CLOCK_GENERATION | Clock cannot be specified as system clock. |
| FSP_ERR_INVALID_TIMING_SETTING | Invalid timing parameter. |
| FSP_ERR_INVALID_LAYER_SETTING | Invalid layer parameter. |
| FSP_ERR_INVALID_ALIGNMENT | Invalid memory alignment found. |
| FSP_ERR_INVALID_GAMMA_SETTING | Invalid gamma correction parameter. |
| FSP_ERR_INVALID_LAYER_FORMAT | Invalid color format in layer. |
| FSP_ERR_INVALID_UPDATE_TIMING | Invalid timing for register update. |
| FSP_ERR_INVALID_CLUT_ACCESS | Invalid access to CLUT entry. |
| FSP_ERR_INVALID_FADE_SETTING | Invalid fade-in/fade-out setting. |
| FSP_ERR_INVALID_BRIGHTNESS_SETTING | Invalid gamma correction parameter. |
| FSP_ERR_JPEG_ERR | JPEG error. |
| FSP_ERR_JPEG_SOI_NOT_DETECTED | SOI not detected until EOI detected. |
| FSP_ERR_JPEG_SOF1_TO_SOFF_DETECTED | SOF1 to SOFF detected. |
| FSP_ERR_JPEG_UNSUPPORTED_PIXEL_FORMAT | Unprovided pixel format detected. |
| FSP_ERR_JPEG_SOF_ACCURACY_ERROR | SOF accuracy error: other than 8 detected. |
| FSP_ERR_JPEG_DQT_ACCURACY_ERROR | DQT accuracy error: other than 0 detected. |
| | |

| | |
|---|---|
| FSP_ERR_JPEG_COMPONENT_ERROR1 | Component error 1: the number of SOF0 header components detected is other than 1, 3, or 4. |
| FSP_ERR_JPEG_COMPONENT_ERROR2 | Component error 2: the number of components differs between SOF0 header and SOS. |
| FSP_ERR_JPEG_SOF0_DQT_DHT_NOT_DETECTED | SOF0, DQT, and DHT not detected when SOS detected. |
| FSP_ERR_JPEG_SOS_NOT_DETECTED | SOS not detected: SOS not detected until EOI detected. |
| FSP_ERR_JPEG_EOI_NOT_DETECTED | EOI not detected (default) |
| FSP_ERR_JPEG_RESTART_INTERVAL_DATA_NUMBER_ERROR | Restart interval data number error detected. |
| FSP_ERR_JPEG_IMAGE_SIZE_ERROR | Image size error detected. |
| FSP_ERR_JPEG_LAST_MCU_DATA_NUMBER_ERROR | Last MCU data number error detected. |
| FSP_ERR_JPEG_BLOCK_DATA_NUMBER_ERROR | Block data number error detected. |
| FSP_ERR_JPEG_BUFFERSIZE_NOT_ENOUGH | User provided buffer size not enough. |
| FSP_ERR_JPEG_UNSUPPORTED_IMAGE_SIZE | JPEG Image size is not aligned with MCU. |
| FSP_ERR_CALIBRATE_FAILED | Calibration failed. |
| FSP_ERR_IP_HARDWARE_NOT_PRESENT | Requested IP does not exist on this device. |
| FSP_ERR_IP_UNIT_NOT_PRESENT | Requested unit does not exist on this device. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Requested channel does not exist on this device. |
| FSP_ERR_NO_MORE_BUFFER | No more buffer found in the memory block pool. |
| FSP_ERR_ILLEGAL_BUFFER_ADDRESS | Buffer address is out of block memory pool. |
| FSP_ERR_INVALID_WORKBUFFER_SIZE | Work buffer size is invalid. |
| FSP_ERR_INVALID_MSG_BUFFER_SIZE | Message buffer size is invalid. |
| FSP_ERR_TOO_MANY_BUFFERS | Number of buffer is too many. |
| FSP_ERR_NO_SUBSCRIBER_FOUND | No message subscriber found. |

| | |
|---|---|
| FSP_ERR_MESSAGE_QUEUE_EMPTY | No message found in the message queue. |
| FSP_ERR_MESSAGE_QUEUE_FULL | No room for new message in the message queue. |
| FSP_ERR_ILLEGAL_SUBSCRIBER_LISTS | Message subscriber lists is illegal. |
| FSP_ERR_BUFFER_RELEASED | Buffer has been released. |
| FSP_ERR_D2D_ERROR_INIT | D/AVE 2D has an error in the initialization. |
| FSP_ERR_D2D_ERROR_DEINIT | D/AVE 2D has an error in the initialization. |
| FSP_ERR_D2D_ERROR_RENDERING | D/AVE 2D has an error in the rendering. |
| FSP_ERR_D2D_ERROR_SIZE | D/AVE 2D has an error in the rendering. |
| FSP_ERR_ETHER_ERROR_NO_DATA | No Data in Receive buffer. |
| FSP_ERR_ETHER_ERROR_LINK | ETHERC/EDMAC has an error in the Auto-negotiation. |
| FSP_ERR_ETHER_ERROR_MAGIC_PACKET_MODE | As a Magic Packet is being detected, and transmission/reception is not enabled. |
| FSP_ERR_ETHER_ERROR_TRANSMIT_BUFFER_FULL | Transmit buffer is not empty. |
| FSP_ERR_ETHER_ERROR_FILTERING | Detect multicast frame when multicast frame filtering enable. |
| FSP_ERR_ETHER_ERROR_PHY_COMMUNICATION | ETHERC/EDMAC has an error in the phy communication. |
| FSP_ERR_ETHER_PHY_ERROR_LINK | PHY is not link up. |
| FSP_ERR_ETHER_PHY_NOT_READY | PHY has an error in the Auto-negotiation. |
| FSP_ERR_QUEUE_FULL | Queue is full, cannot queue another data. |
| FSP_ERR_QUEUE_EMPTY | Queue is empty, no data to dequeue. |
| FSP_ERR_CTSU_SCANNING | Scanning. |
| FSP_ERR_CTSU_NOT_GET_DATA | Not processed previous scan data. |
| FSP_ERR_CTSU_INCOMPLETE_TUNING | Incomplete initial offset tuning. |
| FSP_ERR_CARD_INIT_FAILED | SD card or eMMC device failed to initialize. |

| | |
|---|---|
| FSP_ERR_CARD_NOT_INSERTED | SD card not installed. |
| FSP_ERR_DEVICE_BUSY | Device is holding DAT0 low or another operation is ongoing. |
| FSP_ERR_CARD_NOT_INITIALIZED | SD card was removed. |
| FSP_ERR_CARD_WRITE_PROTECTED | Media is write protected. |
| FSP_ERR_TRANSFER_BUSY | Transfer in progress. |
| FSP_ERR_RESPONSE | Card did not respond or responded with an error. |
| FSP_ERR_MEDIA_FORMAT_FAILED | Media format failed. |
| FSP_ERR_MEDIA_OPEN_FAILED | Media open failed. |
| FSP_ERR_CAN_DATA_UNAVAILABLE | No data available. |
| FSP_ERR_CAN_MODE_SWITCH_FAILED | Switching operation modes failed. |
| FSP_ERR_CAN_INIT_FAILED | Hardware initialization failed. |
| FSP_ERR_CAN_TRANSMIT_NOT_READY | Transmit in progress. |
| FSP_ERR_CAN_RECEIVE_MAILBOX | Mailbox is setup as a receive mailbox. |
| FSP_ERR_CAN_TRANSMIT_MAILBOX | Mailbox is setup as a transmit mailbox. |
| FSP_ERR_CAN_MESSAGE_LOST | Receive message has been overwritten or overrun. |
| FSP_ERR_WIFI_CONFIG_FAILED | WiFi module Configuration failed. |
| FSP_ERR_WIFI_INIT_FAILED | WiFi module initialization failed. |
| FSP_ERR_WIFI_TRANSMIT_FAILED | Transmission failed. |
| FSP_ERR_WIFI_INVALID_MODE | API called when provisioned in client mode. |
| FSP_ERR_WIFI_FAILED | WiFi Failed. |
| FSP_ERR_CELLULAR_CONFIG_FAILED | Cellular module Configuration failed. |
| FSP_ERR_CELLULAR_INIT_FAILED | Cellular module initialization failed. |
| FSP_ERR_CELLULAR_TRANSMIT_FAILED | Transmission failed. |

| | |
|---|---|
| FSP_ERR_CELLULAR_FW_UPTODATE | Firmware is uptodate. |
| FSP_ERR_CELLULAR_FW_UPGRADE_FAILED | Firmware upgrade failed. |
| FSP_ERR_CELLULAR_FAILED | Cellular Failed. |
| FSP_ERR_CELLULAR_INVALID_STATE | API Called in invalid state. |
| FSP_ERR_CELLULAR_REGISTRATION_FAILED | Cellular Network registration failed. |
| FSP_ERR_BLE_FAILED | BLE operation failed. |
| FSP_ERR_BLE_INIT_FAILED | BLE device initialization failed. |
| FSP_ERR_BLE_CONFIG_FAILED | BLE device configuration failed. |
| FSP_ERR_BLE_PRF_ALREADY_ENABLED | BLE device Profile already enabled. |
| FSP_ERR_BLE_PRF_NOT_ENABLED | BLE device not enabled. |
| FSP_ERR_BLE_ABS_INVALID_OPERATION | Invalid operation is executed. |
| FSP_ERR_BLE_ABS_NOT_FOUND | Valid data or free space is not found. |
| FSP_ERR_CRYPTO_CONTINUE | Continue executing function. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | Hardware resource busy. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal I/O buffer is not empty. |
| FSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX | Invalid index. |
| FSP_ERR_CRYPTO_SCE_RETRY | Retry. |
| FSP_ERR_CRYPTO_SCE_VERIFY_FAIL | Verify is failed. |
| FSP_ERR_CRYPTO_SCE_ALREADY_OPEN | HW SCE module is already opened. |
| FSP_ERR_CRYPTO_NOT_OPEN | Hardware module is not initialized. |
| FSP_ERR_CRYPTO_UNKNOWN | Some unknown error occurred. |
| FSP_ERR_CRYPTO_NULL_POINTER | Null pointer input as a parameter. |
| FSP_ERR_CRYPTO_NOT_IMPLEMENTED | Algorithm/size not implemented. |
| FSP_ERR_CRYPTO_RNG_INVALID_PARAM | An invalid parameter is specified. |

| | |
|---|---|
| FSP_ERR_CRYPTO_RNG_FATAL_ERROR | A fatal error occurred. |
| FSP_ERR_CRYPTO_INVALID_SIZE | Size specified is invalid. |
| FSP_ERR_CRYPTO_INVALID_STATE | Function used in an valid state. |
| FSP_ERR_CRYPTO_ALREADY_OPEN | control block is already opened |
| FSP_ERR_CRYPTO_INSTALL_KEY_FAILED | Specified input key is invalid. |
| FSP_ERR_CRYPTO_AUTHENTICATION_FAILED | Authentication failed. |
| FSP_ERR_CRYPTO_COMMON_NOT_OPENED | Crypto Framework Common is not opened. |
| FSP_ERR_CRYPTO_HAL_ERROR | Cryoto HAL module returned an error. |
| FSP_ERR_CRYPTO_KEY_BUF_NOT_ENOUGH | Key buffer size is not enough to generate a key. |
| FSP_ERR_CRYPTO_BUF_OVERFLOW | Attempt to write data larger than what the buffer can hold. |
| FSP_ERR_CRYPTO_INVALID_OPERATION_MODE | Invalid operation mode. |
| FSP_ERR_MESSAGE_TOO_LONG | Message for RSA encryption is too long. |
| FSP_ERR_RSA_DECRYPTION_ERROR | RSA Decryption error. |

## 4.1.2 MCU Board Support Package

BSP

### Functions

| | |
|---|---|
| fsp_err_t | R_FSP_VersionGet (fsp_pack_version_t *const p_version) |
| void | Reset_Handler (void) |
| void | Default_Handler (void) |
| void | SystemInit (void) |
| void | R_BSP_WarmStart (bsp_warm_start_event_t event) |
| fsp_err_t | R_BSP_VersionGet (fsp_version_t *p_version) |

| | |
|---|---|
| void | R_BSP_SoftwareDelay (uint32_t delay, bsp_delay_units_t units) |
| fsp_err_t | R_BSP_GroupIrqWrite (bsp_grp_irq_t irq, void(*p_callback)(bsp_grp_irq_t irq)) |
| void | NMI_Handler (void) |
| void | R_BSP_RegisterProtectEnable (bsp_reg_protect_t regs_to_protect) |
| void | R_BSP_RegisterProtectDisable (bsp_reg_protect_t regs_to_unprotect) |

### Detailed Description

The BSP is responsible for getting the MCU from reset to the user's application. Before reaching the user's application, the BSP sets up the stacks, heap, clocks, interrupts, C runtime environment, and stack monitor.

- BSP Features
- BSP Clock Configuration
- System Interrupts
- Group Interrupts
- External and Peripheral Interrupts
- Error Logging
- BSP Weak Symbols
- Warm Start Callbacks
- Register Protection
- ID Codes
- Software Delay
- Board Specific Features
- Configuration

# Overview

### BSP Features

### 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 RA Configuration editor **Clocks** tab.

- 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 (that is, 32 kHz) clock.
- The BSP implements the required delays to allow the selected clock to stabilize.
- The BSP will configure the CMSIS SystemCoreClock variable after clock initialization with the current system clock frequency.

### System Interrupts

As RA 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.

**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 RA 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.

**External and Peripheral Interrupts**

User configurable interrupts begin with slot 16. These may be external, or peripheral generated

interrupts.

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 RA6M3 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.

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 RA implementation there is a vector entry for each of the SCI0 events that we are interested in.

**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. 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.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 three points where the user can request a callback. These are defined as the 'Pre Clock Init', 'Post Clock Init' 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.c into the application code to get a callback. R_BSP_WarmStart() takes an event parameter of type bsp_warm_start_event_t which describes the type of warm start callback being made.

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.c. To use this function just copy this function into your own code and modify it to meet your needs.

**Heap Allocation**

The relatively low amount of on-chip SRAM available and lack of memory protection in an MCU means that heap use must be very carefully controlled to avoid memory leaks, overruns and attempted overallocation. Further, many RTOSes provide their own dynamic memory allocation system. For these reasons the default heap size is set at 0 bytes, effectively disabling dynamic memory. If it is required for an application setting a positive value to the "Heap size (bytes)" option in the RA Common configurations on the **BSP** tab will allocate a heap.

*Note*

*When using printf/sprintf (and other variants) to output floating point numbers a heap is required. A minimum size of 0x1000 (4096) bytes is recommended when starting development in this case.*

### Error Logging

When error logging is enabled, the error logging function can be redefined on the command line by defining FSP_ERROR_LOG(err) to the desired function call. The default function implementation is FSP_ERROR_LOG(err)=fsp_error_log(err, **FILE**, **LINE**). This implementation uses the predefined macros **FILE** and **LINE** to help identify the location where the error occurred. Removing the line from the function call can reduce code size when error logging is enabled. Some compilers may support other predefined macros like **FUNCTION**, which could be helpful for customizing the error logger.

### Register Protection

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 R_BSP_RegisterProtectDisable() is called, the respective reference counter is incremented.

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

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

```
/* Enable writing to protected CGC registers */
R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_CGC);
/* Insert code to modify protected CGC registers. */
/* Disable writing to protected CGC registers */
R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_CGC);
```

### ID Codes

The ID code is a 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.

### Software Delay

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.

```
 /* Delay at least 1 second. Depending on the number of wait states required for the
region of memory
  * that the software_delay_loop has been linked in this could take longer. The
default is 4 cycles per loop.
  * This can be modified by redefining DELAY_LOOP_CYCLES. BSP_DELAY_UNITS_SECONDS,
BSP_DELAY_UNITS_MILLISECONDS,
  * and BSP_DELAY_UNITS_MICROSECONDS can all be used with R_BSP_SoftwareDelay. */
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
```

### Critical Section Macors

Implements a critical section. Some MCUs (MCUs with the BASEPRI register) support allowing high priority interrupts to execute during critical sections. On these MCUs, interrupts with priority less than or equal to BSP_CFG_IRQ_MASK_LEVEL_FOR_CRITICAL_SECTION are not serviced in critical sections. Interrupts with higher priority than BSP_CFG_IRQ_MASK_LEVEL_FOR_CRITICAL_SECTION still execute in critical sections.

```
    FSP_CRITICAL_SECTION_DEFINE;
 /* Store the current interrupt posture. */
 FSP_CRITICAL_SECTION_ENTER;
 /* Interrupts cannot run in this section unless their priority is less than
BSP_CFG_IRQ_MASK_LEVEL_FOR_CRITICAL_SECTION. */
 /* Restore saved interrupt posture. */
 FSP_CRITICAL_SECTION_EXIT;
```

# Board Specific Features

The BSP will call the board's initialization function (bsp_init) which can initialize board specific features. Possible board features are listed below.

| Board Feature | Description |
|---|---|
| SDRAM Support | The BSP will initialize SDRAM if the board supports it |
| QSPI Support | The BSP will initialize QSPI if the board supports it and put it into ROM mode. Use the R_QSPI module to write and erase the QSPI chip. |

# Configuration

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 when the project is built and/or when the Generate Project Content button is clicked in the RA Configuration editor.

### Build Time Configurations for fsp_common

The following build time configurations are defined in fsp_cfg/bsp/bsp_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Main stack size (bytes) | Value must be an integer multiple of 8 and between 8 and 0xFFFFFFFF | 0x400 | Set the size of the main program stack.<br><br>NOTE: This entry is for the main stack. When using an RTOS, thread stacks can be configured in the properties for each thread. |
| Heap size (bytes) | Value must be 0 or an integer multiple of 8 between 8 and 0xFFFFFFFF. | 0 | The main heap is disabled by default. Set the heap size to a positive integer divisible by 8 to enable it.<br><br>A minimum of 4K (0x1000) is recommended if standard library functions are to be used. |
| MCU Vcc (mV) | Value must between 0 and 5500 (5.5V) | 3300 | Some peripherals require different settings based on the supplied voltage. Entering Vcc here (in mV) allows the relevant driver modules to configure the associated peripherals accordingly. |
| Parameter checking | • Enabled<br>• Disabled | Disabled | When enabled, parameter checking for the BSP is turned on. In addition, any modules whose parameter checking configuration is set to 'Default (BSP)' will perform parameter |

| | | | |
|---|---|---|---|
| | | | checking as well. |
| Assert Failures | • Return FSP_ERR_ASSERTION<br>• Call fsp_error_log then Return FSP_ERR_ASSERTION<br>• Use assert() to Halt Execution<br>• Disable checks that would return FSP_ERR_ASSERTION | Return FSP_ERR_ASSERTION | Define the behavior of the FSP_ASSERT() macro. |
| Error Log | • No Error Log<br>• Errors Logged via fsp_error_log | No Error Log | Specify error logging behavior. |
| ID Code Mode | • Unlocked (Ignore ID)<br>• Locked with All Erase support<br>• Locked | Unlocked (Ignore ID) | When set to 'Locked with All Erase support', the ID Code must be set in the debugger to read or write data to the MCU, but the All Erase command is still accepted regardless. When set to 'Locked', all erase/download/debug access is disabled unless the ID Code is provided. |
| ID Code (32 Hex Characters) | Value must be a 32 character long hex string | FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF | Set the ID Code for locking debug access. This setting is only used when the ID Code Mode is not set to Unlocked. |
| Soft Reset | • Disabled<br>• Enabled | Disabled | Support for soft reset. If disabled, registers are assumed to be set to their default value during startup. |
| Main Oscillator Populated | • Populated<br>• Not Populated | Populated | Select whether or not there is a main oscillator (XTAL) on the board. This setting can be overridden in board_cfg.h. |
| PFS Protect | • Disabled<br>• Enabled | Enabled | Keep the PFS registers locked when they are |

| | | | |
|---|---|---|---|
| | | | not being modified. If disabled they will be unlocked during startup. |
| Main Oscillator Wait Time | • 0.25 us<br>• 128 us<br>• 256 us<br>• 512 us<br>• 1024 us<br>• 2048 us<br>• 4096 us<br>• 8192 us<br>• 16384 us<br>• 32768 us | 32768 us | Number of cycles to wait for the main oscillator clock to stabilize. This setting can be overridden in board_cfg.h |
| Main Oscillator Clock Source | • External Oscillator<br>• Crystal or Resonator | Crystal or Resonator | Select the main oscillator clock source. This setting can be overridden in board_cfg.h |
| Subclock Populated | • Populated<br>• Not Populated | Populated | Select whether or not there is a subclock crystal on the board. This setting can be overridden in board_cfg.h. |
| Subclock Drive | • Low<br>• Standard | Standard | Select the subclock oscillator drive capacitance. This setting can be overridden in board_cfg.h |
| Subclock Stabilization Time (ms) | Value must between 0 and 10000 | 1000 | Select the subclock oscillator stabilization time. This is only used in the startup code if the subclock is selected as the system clock on the Clocks tab. This setting can be overridden in board_cfg.h |

**Modules**

| |
|---|
| RA2A1 |
| RA4M1 |
| RA6M1 |
| RA6M2 |

| | |
|---|---|
| | RA6M3 |

## Macros

| | |
|---|---|
| #define | BSP_IRQ_DISABLED |
| #define | FSP_RETURN(err) |
| #define | FSP_ERROR_LOG(err) |
| #define | FSP_ASSERT(a) |
| #define | FSP_ERROR_RETURN(a, err) |
| #define | FSP_CRITICAL_SECTION_ENTER |
| #define | FSP_CRITICAL_SECTION_EXIT |
| #define | FSP_INVALID_VECTOR |
| #define | BSP_STACK_ALIGNMENT |
| #define | R_BSP_MODULE_START(ip, channel) |
| #define | R_BSP_MODULE_STOP(ip, channel) |

## Enumerations

| | |
|---|---|
| enum | fsp_ip_t |
| enum | fsp_signal_t |
| enum | bsp_warm_start_event_t |
| enum | bsp_delay_units_t |
| enum | bsp_grp_irq_t |
| enum | bsp_reg_protect_t |

## Variables

| | |
|---|---|
| uint32_t | SystemCoreClock |
| const fsp_version_t | g_bsp_version<br>Default initialization function. More... |

## Macro Definition Documentation

### ◆ BSP_IRQ_DISABLED

| #define BSP_IRQ_DISABLED |
|---|
| Used to signify that an ELC event is not able to be used as an interrupt. |

### ◆ FSP_RETURN

| #define FSP_RETURN ( err) |
|---|
| Macro to log and return error without an assertion. |

### ◆ FSP_ERROR_LOG

| #define FSP_ERROR_LOG ( err) |
|---|
| This function is called before returning an error code. To stop on a runtime error, define fsp_error_log in user code and do required debugging (breakpoints, stack dump, etc) in this function. |

### ◆ FSP_ASSERT

| #define FSP_ASSERT ( a) |
|---|
| Default assertion calls FSP_ERROR_RETURN if condition "a" is false. Used to identify incorrect use of API's in FSP functions. |

### ◆ FSP_ERROR_RETURN

| #define FSP_ERROR_RETURN ( a, err ) |
|---|
| All FSP error codes are returned using this macro. Calls FSP_ERROR_LOG function if condition "a" is false. Used to identify runtime errors in FSP functions. |

### ◆ FSP_CRITICAL_SECTION_ENTER

| #define FSP_CRITICAL_SECTION_ENTER |
|---|
| This macro temporarily saves the current interrupt state and disables interrupts. |

### ◆ FSP_CRITICAL_SECTION_EXIT

| #define FSP_CRITICAL_SECTION_EXIT |
|---|
| This macro restores the previously saved interrupt state, reenabling interrupts. |

### ◆ FSP_INVALID_VECTOR

| #define FSP_INVALID_VECTOR |
|---|
| Used to signify that the requested IRQ vector is not defined in this system. |

### ◆ BSP_STACK_ALIGNMENT

| #define BSP_STACK_ALIGNMENT |
|---|
| Stacks (and heap) must be sized and aligned to an integer multiple of this number. |

### ◆ R_BSP_MODULE_START

#define R_BSP_MODULE_START ( ip, channel )

Cancels the module stop state.

**Parameters**

| | |
|---|---|
| ip | fsp_ip_t enum value for the module to be stopped |
| channel | The channel. Use channel 0 for modules without channels. |

### ◆ R_BSP_MODULE_STOP

#define R_BSP_MODULE_STOP ( ip, channel )

Enables the module stop state.

**Parameters**

| | |
|---|---|
| ip | fsp_ip_t enum value for the module to be stopped |
| channel | The channel. Use channel 0 for modules without channels. |

## Enumeration Type Documentation

### ◆ fsp_ip_t

| enum fsp_ip_t | |
|---|---|
| Available modules. | |
| Enumerator | |
| FSP_IP_CFLASH | Code Flash. |
| FSP_IP_DFLASH | Data Flash. |
| FSP_IP_RAM | RAM. |
| FSP_IP_LVD | Low Voltage Detection. |
| FSP_IP_CGC | Clock Generation Circuit. |
| FSP_IP_LPM | Low Power Modes. |
| FSP_IP_FCU | Flash Control Unit. |
| FSP_IP_ICU | Interrupt Control Unit. |
| FSP_IP_DMAC | DMA Controller. |
| FSP_IP_DTC | Data Transfer Controller. |
| FSP_IP_IOPORT | I/O Ports. |
| FSP_IP_PFS | Pin Function Select. |
| FSP_IP_ELC | Event Link Controller. |
| FSP_IP_MPU | Memory Protection Unit. |
| FSP_IP_MSTP | Module Stop. |
| FSP_IP_MMF | Memory Mirror Function. |
| FSP_IP_KEY | Key Interrupt Function. |
| FSP_IP_CAC | Clock Frequency Accuracy Measurement Circuit. |
| FSP_IP_DOC | Data Operation Circuit. |
| FSP_IP_CRC | Cyclic Redundancy Check Calculator. |
| FSP_IP_SCI | Serial Communications Interface. |

| | |
|---|---|
| FSP_IP_IIC | I2C Bus Interface. |
| FSP_IP_SPI | Serial Peripheral Interface. |
| FSP_IP_CTSU | Capacitive Touch Sensing Unit. |
| FSP_IP_SCE | Secure Cryptographic Engine. |
| FSP_IP_SLCDC | Segment LCD Controller. |
| FSP_IP_AES | Advanced Encryption Standard. |
| FSP_IP_TRNG | True Random Number Generator. |
| FSP_IP_FCACHE | Flash Cache. |
| FSP_IP_SRAM | SRAM. |
| FSP_IP_ADC | A/D Converter. |
| FSP_IP_DAC | 12-Bit D/A Converter |
| FSP_IP_TSN | Temperature Sensor. |
| FSP_IP_DAAD | D/A A/D Synchronous Unit. |
| FSP_IP_ACMPHS | High Speed Analog Comparator. |
| FSP_IP_ACMPLP | Low Power Analog Comparator. |
| FSP_IP_OPAMP | Operational Amplifier. |
| FSP_IP_SDADC | Sigma Delta A/D Converter. |
| FSP_IP_RTC | Real Time Clock. |
| FSP_IP_WDT | Watch Dog Timer. |
| FSP_IP_IWDT | Independent Watch Dog Timer. |
| FSP_IP_GPT | General PWM Timer. |
| FSP_IP_POEG | Port Output Enable for GPT. |
| FSP_IP_OPS | Output Phase Switch. |
| FSP_IP_AGT | Asynchronous General-Purpose Timer. |

| | |
|---|---|
| FSP_IP_CAN | Controller Area Network. |
| FSP_IP_IRDA | Infrared Data Association. |
| FSP_IP_QSPI | Quad Serial Peripheral Interface. |
| FSP_IP_USBFS | USB Full Speed. |
| FSP_IP_SDHI | SD/MMC Host Interface. |
| FSP_IP_SRC | Sampling Rate Converter. |
| FSP_IP_SSI | Serial Sound Interface. |
| FSP_IP_DALI | Digital Addressable Lighting Interface. |
| FSP_IP_ETHER | Ethernet MAC Controller. |
| FSP_IP_EDMAC | Ethernet DMA Controller. |
| FSP_IP_EPTPC | Ethernet PTP Controller. |
| FSP_IP_PDC | Parallel Data Capture Unit. |
| FSP_IP_GLCDC | Graphics LCD Controller. |
| FSP_IP_DRW | 2D Drawing Engine |
| FSP_IP_JPEG | JPEG. |
| FSP_IP_DAC8 | 8-Bit D/A Converter |
| FSP_IP_USBHS | USB High Speed. |

### ◆ fsp_signal_t

| enum fsp_signal_t | |
|---|---|
| Signals that can be mapped to an interrupt. | |
| Enumerator | |
| FSP_SIGNAL_ADC_COMPARE_MATCH | ADC COMPARE MATCH. |
| FSP_SIGNAL_ADC_COMPARE_MISMATCH | ADC COMPARE MISMATCH. |
| FSP_SIGNAL_ADC_SCAN_END | ADC SCAN END. |
| FSP_SIGNAL_ADC_SCAN_END_B | ADC SCAN END B. |
| FSP_SIGNAL_ADC_WINDOW_A | ADC WINDOW A. |
| FSP_SIGNAL_ADC_WINDOW_B | ADC WINDOW B. |
| FSP_SIGNAL_AES_RDREQ | AES RDREQ. |
| FSP_SIGNAL_AES_WRREQ | AES WRREQ. |
| FSP_SIGNAL_AGT_COMPARE_A | AGT COMPARE A. |
| FSP_SIGNAL_AGT_COMPARE_B | AGT COMPARE B. |
| FSP_SIGNAL_AGT_INT | AGT INT. |
| FSP_SIGNAL_CAC_FREQUENCY_ERROR | CAC FREQUENCY ERROR. |
| FSP_SIGNAL_CAC_MEASUREMENT_END | CAC MEASUREMENT END. |
| FSP_SIGNAL_CAC_OVERFLOW | CAC OVERFLOW. |
| FSP_SIGNAL_CAN_ERROR | CAN ERROR. |
| FSP_SIGNAL_CAN_FIFO_RX | CAN FIFO RX. |
| FSP_SIGNAL_CAN_FIFO_TX | CAN FIFO TX. |
| FSP_SIGNAL_CAN_MAILBOX_RX | CAN MAILBOX RX. |
| FSP_SIGNAL_CAN_MAILBOX_TX | CAN MAILBOX TX. |
| FSP_SIGNAL_CGC_MOSC_STOP | CGC MOSC STOP. |
| FSP_SIGNAL_LPM_SNOOZE_REQUEST | LPM SNOOZE REQUEST. |

| | |
|---|---|
| FSP_SIGNAL_LVD_LVD1 | LVD LVD1. |
| FSP_SIGNAL_LVD_LVD2 | LVD LVD2. |
| FSP_SIGNAL_VBATT_LVD | VBATT LVD. |
| FSP_SIGNAL_LVD_VBATT | LVD VBATT. |
| FSP_SIGNAL_ACMPHS_INT | ACMPHS INT. |
| FSP_SIGNAL_ACMPLP_INT | ACMPLP INT. |
| FSP_SIGNAL_CTSU_END | CTSU END. |
| FSP_SIGNAL_CTSU_READ | CTSU READ. |
| FSP_SIGNAL_CTSU_WRITE | CTSU WRITE. |
| FSP_SIGNAL_DALI_DEI | DALI DEI. |
| FSP_SIGNAL_DALI_CLI | DALI CLI. |
| FSP_SIGNAL_DALI_SDI | DALI SDI. |
| FSP_SIGNAL_DALI_BPI | DALI BPI. |
| FSP_SIGNAL_DALI_FEI | DALI FEI. |
| FSP_SIGNAL_DALI_SDI_OR_BPI | DALI SDI OR BPI. |
| FSP_SIGNAL_DMAC_INT | DMAC INT. |
| FSP_SIGNAL_DOC_INT | DOC INT. |
| FSP_SIGNAL_DRW_INT | DRW INT. |
| FSP_SIGNAL_DTC_COMPLETE | DTC COMPLETE. |
| FSP_SIGNAL_DTC_END | DTC END. |
| FSP_SIGNAL_EDMAC_EINT | EDMAC EINT. |
| FSP_SIGNAL_ELC_SOFTWARE_EVENT_0 | ELC SOFTWARE EVENT 0. |
| FSP_SIGNAL_ELC_SOFTWARE_EVENT_1 | ELC SOFTWARE EVENT 1. |
| FSP_SIGNAL_EPTPC_IPLS | EPTPC IPLS. |

| | |
|---|---|
| FSP_SIGNAL_EPTPC_MINT | EPTPC MINT. |
| FSP_SIGNAL_EPTPC_PINT | EPTPC PINT. |
| FSP_SIGNAL_EPTPC_TIMER0_FALL | EPTPC TIMER0 FALL. |
| FSP_SIGNAL_EPTPC_TIMER0_RISE | EPTPC TIMER0 RISE. |
| FSP_SIGNAL_EPTPC_TIMER1_FALL | EPTPC TIMER1 FALL. |
| FSP_SIGNAL_EPTPC_TIMER1_RISE | EPTPC TIMER1 RISE. |
| FSP_SIGNAL_EPTPC_TIMER2_FALL | EPTPC TIMER2 FALL. |
| FSP_SIGNAL_EPTPC_TIMER2_RISE | EPTPC TIMER2 RISE. |
| FSP_SIGNAL_EPTPC_TIMER3_FALL | EPTPC TIMER3 FALL. |
| FSP_SIGNAL_EPTPC_TIMER3_RISE | EPTPC TIMER3 RISE. |
| FSP_SIGNAL_EPTPC_TIMER4_FALL | EPTPC TIMER4 FALL. |
| FSP_SIGNAL_EPTPC_TIMER4_RISE | EPTPC TIMER4 RISE. |
| FSP_SIGNAL_EPTPC_TIMER5_FALL | EPTPC TIMER5 FALL. |
| FSP_SIGNAL_EPTPC_TIMER5_RISE | EPTPC TIMER5 RISE. |
| FSP_SIGNAL_FCU_FIFERR | FCU FIFERR. |
| FSP_SIGNAL_FCU_FRDYI | FCU FRDYI. |
| FSP_SIGNAL_GLCDC_LINE_DETECT | GLCDC LINE DETECT. |
| FSP_SIGNAL_GLCDC_UNDERFLOW_1 | GLCDC UNDERFLOW 1. |
| FSP_SIGNAL_GLCDC_UNDERFLOW_2 | GLCDC UNDERFLOW 2. |
| FSP_SIGNAL_GPT_CAPTURE_COMPARE_A | GPT CAPTURE COMPARE A. |
| FSP_SIGNAL_GPT_CAPTURE_COMPARE_B | GPT CAPTURE COMPARE B. |
| FSP_SIGNAL_GPT_COMPARE_C | GPT COMPARE C. |
| FSP_SIGNAL_GPT_COMPARE_D | GPT COMPARE D. |
| FSP_SIGNAL_GPT_COMPARE_E | GPT COMPARE E. |

| | |
|---|---|
| FSP_SIGNAL_GPT_COMPARE_F | GPT COMPARE F. |
| FSP_SIGNAL_GPT_COUNTER_OVERFLOW | GPT COUNTER OVERFLOW. |
| FSP_SIGNAL_GPT_COUNTER_UNDERFLOW | GPT COUNTER UNDERFLOW. |
| FSP_SIGNAL_GPT_AD_TRIG_A | GPT AD TRIG A. |
| FSP_SIGNAL_GPT_AD_TRIG_B | GPT AD TRIG B. |
| FSP_SIGNAL_OPS_UVW_EDGE | OPS UVW EDGE. |
| FSP_SIGNAL_ICU_IRQ0 | ICU IRQ0. |
| FSP_SIGNAL_ICU_IRQ1 | ICU IRQ1. |
| FSP_SIGNAL_ICU_IRQ2 | ICU IRQ2. |
| FSP_SIGNAL_ICU_IRQ3 | ICU IRQ3. |
| FSP_SIGNAL_ICU_IRQ4 | ICU IRQ4. |
| FSP_SIGNAL_ICU_IRQ5 | ICU IRQ5. |
| FSP_SIGNAL_ICU_IRQ6 | ICU IRQ6. |
| FSP_SIGNAL_ICU_IRQ7 | ICU IRQ7. |
| FSP_SIGNAL_ICU_IRQ8 | ICU IRQ8. |
| FSP_SIGNAL_ICU_IRQ9 | ICU IRQ9. |
| FSP_SIGNAL_ICU_IRQ10 | ICU IRQ10. |
| FSP_SIGNAL_ICU_IRQ11 | ICU IRQ11. |
| FSP_SIGNAL_ICU_IRQ12 | ICU IRQ12. |
| FSP_SIGNAL_ICU_IRQ13 | ICU IRQ13. |
| FSP_SIGNAL_ICU_IRQ14 | ICU IRQ14. |
| FSP_SIGNAL_ICU_IRQ15 | ICU IRQ15. |
| FSP_SIGNAL_ICU_SNOOZE_CANCEL | ICU SNOOZE CANCEL. |
| FSP_SIGNAL_IIC_ERI | IIC ERI. |

| | |
|---|---|
| FSP_SIGNAL_IIC_RXI | IIC RXI. |
| FSP_SIGNAL_IIC_TEI | IIC TEI. |
| FSP_SIGNAL_IIC_TXI | IIC TXI. |
| FSP_SIGNAL_IIC_WUI | IIC WUI. |
| FSP_SIGNAL_IOPORT_EVENT_1 | IOPORT EVENT 1. |
| FSP_SIGNAL_IOPORT_EVENT_2 | IOPORT EVENT 2. |
| FSP_SIGNAL_IOPORT_EVENT_3 | IOPORT EVENT 3. |
| FSP_SIGNAL_IOPORT_EVENT_4 | IOPORT EVENT 4. |
| FSP_SIGNAL_IWDT_UNDERFLOW | IWDT UNDERFLOW. |
| FSP_SIGNAL_JPEG_JDTI | JPEG JDTI. |
| FSP_SIGNAL_JPEG_JEDI | JPEG JEDI. |
| FSP_SIGNAL_KEY_INT | KEY INT. |
| FSP_SIGNAL_PDC_FRAME_END | PDC FRAME END. |
| FSP_SIGNAL_PDC_INT | PDC INT. |
| FSP_SIGNAL_PDC_RECEIVE_DATA_READY | PDC RECEIVE DATA READY. |
| FSP_SIGNAL_POEG_EVENT | POEG EVENT. |
| FSP_SIGNAL_QSPI_INT | QSPI INT. |
| FSP_SIGNAL_RTC_ALARM | RTC ALARM. |
| FSP_SIGNAL_RTC_PERIOD | RTC PERIOD. |
| FSP_SIGNAL_RTC_CARRY | RTC CARRY. |
| FSP_SIGNAL_SCE_INTEGRATE_RDRDY | SCE INTEGRATE RDRDY. |
| FSP_SIGNAL_SCE_INTEGRATE_WRRDY | SCE INTEGRATE WRRDY. |
| FSP_SIGNAL_SCE_LONG_PLG | SCE LONG PLG. |
| FSP_SIGNAL_SCE_PROC_BUSY | SCE PROC BUSY. |

| | |
|---|---|
| FSP_SIGNAL_SCE_RDRDY_0 | SCE RDRDY 0. |
| FSP_SIGNAL_SCE_RDRDY_1 | SCE RDRDY 1. |
| FSP_SIGNAL_SCE_ROMOK | SCE ROMOK. |
| FSP_SIGNAL_SCE_TEST_BUSY | SCE TEST BUSY. |
| FSP_SIGNAL_SCE_WRRDY_0 | SCE WRRDY 0. |
| FSP_SIGNAL_SCE_WRRDY_1 | SCE WRRDY 1. |
| FSP_SIGNAL_SCE_WRRDY_4 | SCE WRRDY 4. |
| FSP_SIGNAL_SCI_AM | SCI AM. |
| FSP_SIGNAL_SCI_ERI | SCI ERI. |
| FSP_SIGNAL_SCI_RXI | SCI RXI. |
| FSP_SIGNAL_SCI_RXI_OR_ERI | SCI RXI OR ERI. |
| FSP_SIGNAL_SCI_TEI | SCI TEI. |
| FSP_SIGNAL_SCI_TXI | SCI TXI. |
| FSP_SIGNAL_SDADC_ADI | SDADC ADI. |
| FSP_SIGNAL_SDADC_SCANEND | SDADC SCANEND. |
| FSP_SIGNAL_SDADC_CALIEND | SDADC CALIEND. |
| FSP_SIGNAL_SDHIMMC_ACCS | SDHIMMC ACCS. |
| FSP_SIGNAL_SDHIMMC_CARD | SDHIMMC CARD. |
| FSP_SIGNAL_SDHIMMC_DMA_REQ | SDHIMMC DMA REQ. |
| FSP_SIGNAL_SDHIMMC_SDIO | SDHIMMC SDIO. |
| FSP_SIGNAL_SPI_ERI | SPI ERI. |
| FSP_SIGNAL_SPI_IDLE | SPI IDLE. |
| FSP_SIGNAL_SPI_RXI | SPI RXI. |
| FSP_SIGNAL_SPI_TEI | SPI TEI. |

| | |
|---|---|
| FSP_SIGNAL_SPI_TXI | SPI TXI. |
| FSP_SIGNAL_SRC_CONVERSION_END | SRC CONVERSION END. |
| FSP_SIGNAL_SRC_INPUT_FIFO_EMPTY | SRC INPUT FIFO EMPTY. |
| FSP_SIGNAL_SRC_OUTPUT_FIFO_FULL | SRC OUTPUT FIFO FULL. |
| FSP_SIGNAL_SRC_OUTPUT_FIFO_OVERFLOW | SRC OUTPUT FIFO OVERFLOW. |
| FSP_SIGNAL_SRC_OUTPUT_FIFO_UNDERFLOW | SRC OUTPUT FIFO UNDERFLOW. |
| FSP_SIGNAL_SSI_INT | SSI INT. |
| FSP_SIGNAL_SSI_RXI | SSI RXI. |
| FSP_SIGNAL_SSI_TXI | SSI TXI. |
| FSP_SIGNAL_SSI_TXI_RXI | SSI TXI RXI. |
| FSP_SIGNAL_TRNG_RDREQ | TRNG RDREQ. |
| FSP_SIGNAL_USB_FIFO_0 | USB FIFO 0. |
| FSP_SIGNAL_USB_FIFO_1 | USB FIFO 1. |
| FSP_SIGNAL_USB_INT | USB INT. |
| FSP_SIGNAL_USB_RESUME | USB RESUME. |
| FSP_SIGNAL_USB_USB_INT_RESUME | USB USB INT RESUME. |
| FSP_SIGNAL_WDT_UNDERFLOW | WDT UNDERFLOW. |

### ◆ bsp_warm_start_event_t

| enum bsp_warm_start_event_t | |
|---|---|
| Different warm start entry locations in the BSP. | |
| Enumerator | |
| BSP_WARM_START_RESET | Called almost immediately after reset. No C runtime environment, clocks, or IRQs. |
| BSP_WARM_START_POST_CLOCK | Called after clock initialization. No C runtime environment or IRQs. |
| BSP_WARM_START_POST_C | Called after clocks and C runtime environment have been set up. |

### ◆ bsp_delay_units_t

| enum bsp_delay_units_t | |
|---|---|
| Available delay units for R_BSP_SoftwareDelay(). These are ultimately used to calculate a total # of microseconds | |
| Enumerator | |
| BSP_DELAY_UNITS_SECONDS | Requested delay amount is in seconds. |
| BSP_DELAY_UNITS_MILLISECONDS | Requested delay amount is in milliseconds. |
| BSP_DELAY_UNITS_MICROSECONDS | Requested delay amount is in microseconds. |

### ◆ bsp_grp_irq_t

| enum bsp_grp_irq_t | |
|---|---|
| Which interrupts can have callbacks registered. | |
| Enumerator | |
| BSP_GRP_IRQ_IWDT_ERROR | IWDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_WDT_ERROR | WDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_LVD1 | Voltage monitoring 1 interrupt. |
| BSP_GRP_IRQ_LVD2 | Voltage monitoring 2 interrupt. |
| BSP_GRP_IRQ_VBATT | VBATT monitor interrupt. |
| BSP_GRP_IRQ_OSC_STOP_DETECT | Oscillation stop is detected. |
| BSP_GRP_IRQ_NMI_PIN | NMI Pin interrupt. |
| BSP_GRP_IRQ_RAM_PARITY | RAM Parity Error. |
| BSP_GRP_IRQ_RAM_ECC | RAM ECC Error. |
| BSP_GRP_IRQ_MPU_BUS_SLAVE | MPU Bus Slave Error. |
| BSP_GRP_IRQ_MPU_BUS_MASTER | MPU Bus Master Error. |
| BSP_GRP_IRQ_MPU_STACK | MPU Stack Error. |

### ◆ bsp_reg_protect_t

| enum bsp_reg_protect_t | |
|---|---|
| The different types of registers that can be protected. | |
| Enumerator | |
| BSP_REG_PROTECT_CGC | Enables writing to the registers related to the clock generation circuit. |
| BSP_REG_PROTECT_OM_LPC_BATT | Enables writing to the registers related to operating modes, low power consumption, and battery backup function. |
| BSP_REG_PROTECT_LVD | Enables writing to the registers related to the LVD: LVCMPCR, LVDLVLR, LVD1CR0, LVD1CR1, LVD1SR, LVD2CR0, LVD2CR1, LVD2SR. |

## Function Documentation

### ◆ R_FSP_VersionGet()

fsp_err_t R_FSP_VersionGet ( fsp_pack_version_t *const *p_version*)

Get the FSP version based on compile time macros.

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Memory address to return version information to. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version information stored. |
| FSP_ERR_ASSERTION | The parameter p_version is NULL. |

### ◆ Reset_Handler()

void Reset_Handler ( void )

MCU starts executing here out of reset. Main stack pointer is set up already.

### ◆ Default_Handler()

void Default_Handler ( void )

Default exception handler.

### ◆ SystemInit()

| void SystemInit ( void ) |
|---|
| Initialize the MCU and the runtime environment. |

### ◆ R_BSP_WarmStart()

| void R_BSP_WarmStart ( bsp_warm_start_event_t *event*) |
|---|
| This function is called at various points during the startup process. This function is declared as a weak symbol higher up in this file because it is meant to be overridden by a user implemented version. One of the main uses for this function is to call functional safety code during the startup process. To use this function just copy this function into your own code and modify it to meet your needs. |

**Parameters**

| | | |
|---|---|---|
| [in] | event | Where the code currently is in the start up process |

### ◆ R_BSP_VersionGet()

| fsp_err_t R_BSP_VersionGet ( fsp_version_t * *p_version*) |
|---|
| Get the BSP version based on compile time macros. |

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Memory address to return version information to. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version information stored. |
| FSP_ERR_ASSERTION | The parameter p_version is NULL. |

### ◆ R_BSP_SoftwareDelay()

void R_BSP_SoftwareDelay ( uint32_t *delay*, bsp_delay_units_t *units* )

Delay for at least the specified duration in units and return.

**Parameters**

| | | |
|---|---|---|
| [in] | delay | The number of 'units' to delay. |
| [in] | units | 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.<br>For example:<br>At 1 MHz one cycle takes 1 microsecond (.000001 seconds).<br>At 12 MHz one cycle takes 1/12 microsecond or 83 nanoseconds.<br>Therefore one run through bsp_prv_software_delay_loop() takes: ~ (83 * BSP_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. @120MHz: ((0xFFFFFFFF loops * 4 cycles /loop) / 120000000) = 143 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 longer. @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 bsp_prv_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.*

### ◆ R_BSP_GroupIrqWrite()

fsp_err_t R_BSP_GroupIrqWrite ( bsp_grp_irq_t *irq*, void(*)(bsp_grp_irq_t irq) *p_callback* )

Register a callback function for supported interrupts. If NULL is passed for the callback argument then any previously registered callbacks are unregistered.

**Parameters**

| | | |
|---|---|---|
| [in] | irq | Interrupt for which to register a callback. |
| [in] | p_callback | Pointer to function to call when interrupt occurs. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback registered |
| FSP_ERR_ASSERTION | Callback pointer is NULL |

### ◆ NMI_Handler()

void NMI_Handler ( void )

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.

### ◆ R_BSP_RegisterProtectEnable()

void R_BSP_RegisterProtectEnable ( bsp_reg_protect_t *regs_to_protect*)

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).

**Parameters**

| | | |
|---|---|---|
| [in] | regs_to_protect | Registers which have write protection enabled. |

### ◆ R_BSP_RegisterProtectDisable()

| void R_BSP_RegisterProtectDisable ( bsp_reg_protect_t *regs_to_unprotect*) |
|---|
| 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). |

**Parameters**

| | | |
|---|---|---|
| [in] | regs_to_unprotect | Registers which have write protection disabled. |

## Variable Documentation

### ◆ SystemCoreClock

| uint32_t SystemCoreClock |
|---|
| System Clock Frequency (Core Clock) |

### ◆ g_bsp_version

| const fsp_version_t g_bsp_version |
|---|
| Default initialization function.<br>Version data structure used by error logger macro. |

### 4.1.2.1 RA2A1

BSP » MCU Board Support Package

## Detailed Description

### Build Time Configurations for ra2a1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| OFS0 register settings > Independent WDT > Start Mode | • IWDT is Disabled<br>• IWDT is automatically | IWDT is Disabled | |

| | | |
|---|---|---|
| | activated after a reset (Autostart mode) | |
| OFS0 register settings > Independent WDT > Timeout Period | • 128 cycles<br>• 512 cycles<br>• 1024 cycles<br>• 2048 cycles | 2048 cycles |
| OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor | • 1<br>• 16<br>• 32<br>• 64<br>• 128<br>• 256 | 128 |
| OFS0 register settings > Independent WDT > Window End Position | • 75%<br>• 50%<br>• 25%<br>• 0% (no window end position) | 0% (no window end position) |
| OFS0 register settings > Independent WDT > Window Start Position | • 25%<br>• 50%<br>• 75%<br>• 100% (no window start position) | 100% (no window start position) |
| OFS0 register settings > Independent WDT > Reset Interrupt Request Select | • NMI request or interrupt request is enabled<br>• Reset is enabled | Reset is enabled |
| OFS0 register settings > Independent WDT > Stop Control | • Counting continues<br>• Stop counting when in Sleep, Snooze mode, or Software Standby | Stop counting when in Sleep, Snooze mode, or Software Standby |
| OFS0 register settings > WDT > Start Mode Select | • Automatically activate WDT after a reset (auto-start mode)<br>• Stop WDT after a reset (register-start mode) | Stop WDT after a reset (register-start mode) |
| OFS0 register settings > WDT > Timeout Period | • 1024 cycles<br>• 4096 cycles<br>• 8192 cycles<br>• 16384 cycles | 16384 cycles |

| | | | |
|---|---|---|---|
| OFS0 register settings > WDT > Clock Frequency Division Ratio | • 4<br>• 64<br>• 128<br>• 512<br>• 2048<br>• 8192 | 128 | |
| OFS0 register settings > WDT > Window End Position | • 75%<br>• 50%<br>• 25%<br>• 0% (no window end position) | 0% (no window end position) | |
| OFS0 register settings > WDT > Window Start Position | • 25%<br>• 50%<br>• 75%<br>• 100% (no window start position) | 100% (no window start position) | |
| OFS0 register settings > WDT > Reset Interrupt Request | • NMI<br>• Reset | Reset | |
| OFS0 register settings > WDT > Stop Control | • Counting continues<br>• Stop counting when entering Sleep mode | Stop counting when entering Sleep mode | |
| OFS1 register settings > Voltage Detection 0 Circuit Start | • Voltage monitor 0 reset is enabled after reset<br>• Voltage monitor 0 reset is disabled after reset | Voltage monitor 0 reset is disabled after reset | |
| OFS1 register settings > Voltage Detection 0 Level | • 3.84 V<br>• 2.82 V<br>• 2.51 V<br>• 1.90 V<br>• 1.70 V | 1.90 V | |
| OFS1 register settings > HOCO Oscillation Enable | HOCO oscillation is enabled after reset | HOCO oscillation is enabled after reset | HOCO must be enabled out of reset because the MCU starts up in low voltage mode and the HOCO must be operating in low voltage mode. |
| MPU > Enable or disable PC Region 0 | • Enabled<br>• Disabled | Disabled | |
| MPU > PC0 Start | Value must be an integer between 0 and | 0x000FFFFC | |

| | | |
|---|---|---|
| | 0x000FFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM) | |
| MPU > PC0 End | Value must be an integer between 0x00000003 and 0x000FFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM) | 0x000FFFFF |
| MPU > Enable or disable PC Region 1 | • Enabled<br>• Disabled | Disabled |
| MPU > PC1 Start | Value must be an integer between 0 and 0x000FFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM) | 0x000FFFFC |
| MPU > PC1 End | Value must be an integer between 0x00000003 and 0x000FFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM) | 0x000FFFFF |
| MPU > Enable or disable Memory Region 0 | • Enabled<br>• Disabled | Disabled |
| MPU > Memory Region 0 Start | Value must be an integer between 0 and 0x000FFFFC | 0x000FFFFC |
| MPU > Memory Region 0 End | Value must be an integer between 0x00000003 and 0x000FFFFF | 0x000FFFFF |
| MPU > Enable or disable Memory Region 1 | • Enabled<br>• Disabled | Disabled |
| MPU > Memory Region 1 Start | Value must be an integer between 0x1FF00000 and 0x200FFFFC | 0x200FFFFC |
| MPU > Memory Region 1 End | Value must be an integer between 0x1FF00003 and 0x200FFFFF | 0x200FFFFF |
| MPU > Enable or disable Memory Region 2 | • Enabled<br>• Disabled | Disabled |
| MPU > Memory Region | Value must be an | 0x407FFFFC |

| | | | |
|---|---|---|---|
| 2 Start | integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC | | |
| MPU > Memory Region 2 End | Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF | 0x407FFFFF | |
| MPU > Enable or disable Memory Region 3 | • Enabled<br>• Disabled | Disabled | |
| MPU > Memory Region 3 Start | Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC | 0x400DFFFC | |
| MPU > Memory Region 3 End | Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF | 0x400DFFFF | |
| Use Low Voltage Mode | • Enable<br>• Disable | Disable | Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4 when oscillation stop detection is used. |

## Enumerations

| | |
|---|---|
| enum | elc_event_t |

## Enumeration Type Documentation

### ◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

***Note***

*This list may change based on based on the device.*

### 4.1.2.2 RA4M1

BSP » MCU Board Support Package

#### Detailed Description

#### Build Time Configurations for ra4m1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| OFS0 register settings > Independent WDT > Start Mode | • IWDT is Disabled<br>• IWDT is automatically activated after a reset (Autostart mode) | IWDT is Disabled | |
| OFS0 register settings > Independent WDT > Timeout Period | • 128 cycles<br>• 512 cycles<br>• 1024 cycles<br>• 2048 cycles | 2048 cycles | |
| OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor | • 1<br>• 16<br>• 32<br>• 64<br>• 128<br>• 256 | 128 | |
| OFS0 register settings > Independent WDT > Window End Position | • 75%<br>• 50%<br>• 25%<br>• 0% (no window end position) | 0% (no window end position) | |
| OFS0 register settings > Independent WDT > Window Start Position | • 25%<br>• 50%<br>• 75%<br>• 100% (no window start position) | 100% (no window start position) | |
| OFS0 register settings > Independent WDT > Reset Interrupt Request Select | • NMI request or interrupt request is enabled<br>• Reset is enabled | Reset is enabled | |

| | | |
|---|---|---|
| OFS0 register settings > Independent WDT > Stop Control | • Counting continues<br>• Stop counting when in Sleep, Snooze mode, or Software Standby | Stop counting when in Sleep, Snooze mode, or Software Standby |
| OFS0 register settings > WDT > Start Mode Select | • Automatically activate WDT after a reset (auto-start mode)<br>• Stop WDT after a reset (register-start mode) | Stop WDT after a reset (register-start mode) |
| OFS0 register settings > WDT > Timeout Period | • 1024 cycles<br>• 4096 cycles<br>• 8192 cycles<br>• 16384 cycles | 16384 cycles |
| OFS0 register settings > WDT > Clock Frequency Division Ratio | • 4<br>• 64<br>• 128<br>• 512<br>• 2048<br>• 8192 | 128 |
| OFS0 register settings > WDT > Window End Position | • 75%<br>• 50%<br>• 25%<br>• 0% (no window end position) | 0% (no window end position) |
| OFS0 register settings > WDT > Window Start Position | • 25%<br>• 50%<br>• 75%<br>• 100% (no window start position) | 100% (no window start position) |
| OFS0 register settings > WDT > Reset Interrupt Request | • NMI<br>• Reset | Reset |
| OFS0 register settings > WDT > Stop Control | • Counting continues<br>• Stop counting when entering Sleep mode | Stop counting when entering Sleep mode |
| OFS1 register settings > Voltage Detection 0 Circuit Start | • Voltage monitor 0 reset is enabled after reset<br>• Voltage monitor 0 reset is | Voltage monitor 0 reset is disabled after reset |

| | | | |
|---|---|---|---|
| | disabled after reset | | |
| OFS1 register settings > Voltage Detection 0 Level | • 3.84 V<br>• 2.82 V<br>• 2.51 V<br>• 1.90 V<br>• 1.70 V | 1.90 V | |
| OFS1 register settings > HOCO Oscillation Enable | HOCO oscillation is enabled after reset | HOCO oscillation is enabled after reset | HOCO must be enabled out of reset because the MCU starts up in low voltage mode and the HOCO must be operating in low voltage mode. |
| MPU > Enable or disable PC Region 0 | • Enabled<br>• Disabled | Disabled | |
| MPU > PC0 Start | Value must be an integer between 0 and 0x00FFFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM) | 0x00FFFFFC | |
| MPU > PC0 End | Value must be an integer between 0x00000003 and 0x00FFFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM) | 0x00FFFFFF | |
| MPU > Enable or disable PC Region 1 | • Enabled<br>• Disabled | Disabled | |
| MPU > PC1 Start | Value must be an integer between 0 and 0x00FFFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM) | 0x00FFFFFC | |
| MPU > PC1 End | Value must be an integer between 0x00000003 and 0x00FFFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM) | 0x00FFFFFF | |
| MPU > Enable or disable Memory Region 0 | • Enabled<br>• Disabled | Disabled | |
| MPU > Memory Region 0 Start | Value must be an integer between 0 and 0x00FFFFFC | 0x00FFFFFC | |
| MPU > Memory Region 0 End | Value must be an integer between | 0x00FFFFFF | |

| | | | |
|---|---|---|---|
| | 0x00000003 and 0x00FFFFFF | | |
| MPU > Enable or disable Memory Region 1 | • Enabled<br>• Disabled | Disabled | |
| MPU > Memory Region 1 Start | Value must be an integer between 0x1FF00000 and 0x200FFFFC | 0x200FFFFC | |
| MPU > Memory Region 1 End | Value must be an integer between 0x1FF00003 and 0x200FFFFF | 0x200FFFFF | |
| MPU > Enable or disable Memory Region 2 | • Enabled<br>• Disabled | Disabled | |
| MPU > Memory Region 2 Start | Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC | 0x407FFFFC | |
| MPU > Memory Region 2 End | Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF | 0x407FFFFF | |
| MPU > Enable or disable Memory Region 3 | • Enabled<br>• Disabled | Disabled | |
| MPU > Memory Region 3 Start | Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC | 0x400DFFFC | |
| MPU > Memory Region 3 End | Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF | 0x400DFFFF | |
| Use Low Voltage Mode | • Enable<br>• Disable | Disable | Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least |

4.

### Enumerations

| | |
|---|---|
| enum | elc_event_t |

### Enumeration Type Documentation

#### ◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

***Note***
*This list may change based on based on the device.*

### 4.1.2.3 RA6M1

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra6m1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| OFS0 register settings > Independent WDT > Start Mode | • IWDT is Disabled<br>• IWDT is automatically activated after a reset (Autostart mode) | IWDT is Disabled | |
| OFS0 register settings > Independent WDT > Timeout Period | • 128 cycles<br>• 512 cycles<br>• 1024 cycles<br>• 2048 cycles | 2048 cycles | |
| OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor | • 1<br>• 16<br>• 32<br>• 64<br>• 128<br>• 256 | 128 | |

| | | |
|---|---|---|
| OFS0 register settings > Independent WDT > Window End Position | • 75%<br>• 50%<br>• 25%<br>• 0% (no window end position) | 0% (no window end position) |
| OFS0 register settings > Independent WDT > Window Start Position | • 25%<br>• 50%<br>• 75%<br>• 100% (no window start position) | 100% (no window start position) |
| OFS0 register settings > Independent WDT > Reset Interrupt Request Select | • NMI request or interrupt request is enabled<br>• Reset is enabled | Reset is enabled |
| OFS0 register settings > Independent WDT > Stop Control | • Counting continues (Note: Device will not enter Deep Standby Mode when selected. Device will enter Software Standby Mode)<br>• Stop counting when in Sleep, Snooze mode, or Software Standby | Stop counting when in Sleep, Snooze mode, or Software Standby |
| OFS0 register settings > WDT > Start Mode Select | • Automatically activate WDT after a reset (auto-start mode)<br>• Stop WDT after a reset (register-start mode) | Stop WDT after a reset (register-start mode) |
| OFS0 register settings > WDT > Timeout Period | • 1024 cycles<br>• 4096 cycles<br>• 8192 cycles<br>• 16384 cycles | 16384 cycles |
| OFS0 register settings > WDT > Clock Frequency Division Ratio | • 4<br>• 64<br>• 128<br>• 512<br>• 2048<br>• 8192 | 128 |
| OFS0 register settings | • 75% | 0% (no window end |

| | | |
|---|---|---|
| > WDT > Window End Position | • 50%<br>• 25%<br>• 0% (no window end position) | position) |
| OFS0 register settings > WDT > Window Start Position | • 25%<br>• 50%<br>• 75%<br>• 100% (no window start position) | 100% (no window start position) |
| OFS0 register settings > WDT > Reset Interrupt Request | • NMI<br>• Reset | Reset |
| OFS0 register settings > WDT > Stop Control | • Counting continues<br>• Stop counting when entering Sleep mode | Stop counting when entering Sleep mode |
| OFS1 register settings > Voltage Detection 0 Circuit Start | • Voltage monitor 0 reset is enabled after reset<br>• Voltage monitor 0 reset is disabled after reset | Voltage monitor 0 reset is disabled after reset |
| OFS1 register settings > Voltage Detection 0 Level | • 2.94 V<br>• 2.87 V<br>• 2.80 V | 2.80 V |
| OFS1 register settings > HOCO Oscillation Enable | • HOCO oscillation is enabled after reset<br>• HOCO oscillation is disabled after reset | HOCO oscillation is disabled after reset |
| MPU > Enable or disable PC Region 0 | • Enabled<br>• Disabled | Disabled |
| MPU > PC0 Start | Value must be an integer between 0 and 0xFFFFFFFC | 0xFFFFFFFC |
| MPU > PC0 End | Value must be an integer between 0x00000003 and 0xFFFFFFFF | 0xFFFFFFFF |
| MPU > Enable or disable PC Region 1 | • Enabled<br>• Disabled | Disabled |

| | | |
|---|---|---|
| MPU > PC1 Start | Value must be an integer between 0 and 0xFFFFFFFC | 0xFFFFFFFC |
| MPU > PC1 End | Value must be an integer between 0x00000003 and 0xFFFFFFFF | 0xFFFFFFFF |
| MPU > Enable or disable Memory Region 0 | • Enabled<br>• Disabled | Disabled |
| MPU > Memory Region 0 Start | Value must be an integer between 0 and 0x00FFFFFC | 0x00FFFFFC |
| MPU > Memory Region 0 End | Value must be an integer between 0x00000003 and 0x00FFFFFF | 0x00FFFFFF |
| MPU > Enable or disable Memory Region 1 | • Enabled<br>• Disabled | Disabled |
| MPU > Memory Region 1 Start | Value must be an integer between 0x1FF00000 and 0x200FFFFC | 0x200FFFFC |
| MPU > Memory Region 1 End | Value must be an integer between 0x1FF00003 and 0x200FFFFF | 0x200FFFFF |
| MPU > Enable or disable Memory Region 2 | • Enabled<br>• Disabled | Disabled |
| MPU > Memory Region 2 Start | Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC | 0x407FFFFC |
| MPU > Memory Region 2 End | Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF | 0x407FFFFF |
| MPU > Enable or disable Memory Region 3 | • Enabled<br>• Disabled | Disabled |
| MPU > Memory Region | Value must be an | 0x400DFFFC |

| | | |
|---|---|---|
| 3 Start | integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC | |
| MPU > Memory Region 3 End | Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF | 0x400DFFFF |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |

### Enumeration Type Documentation

#### ◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

***Note***
*This list may change based on based on the device.*

### 4.1.2.4 RA6M2

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra6m2_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| OFS0 register settings > Independent WDT > Start Mode | • IWDT is Disabled<br>• IWDT is automatically activated after a reset (Autostart mode) | IWDT is Disabled | |

| | | |
|---|---|---|
| OFS0 register settings > Independent WDT > Timeout Period | • 128 cycles<br>• 512 cycles<br>• 1024 cycles<br>• 2048 cycles | 2048 cycles |
| OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor | • 1<br>• 16<br>• 32<br>• 64<br>• 128<br>• 256 | 128 |
| OFS0 register settings > Independent WDT > Window End Position | • 75%<br>• 50%<br>• 25%<br>• 0% (no window end position) | 0% (no window end position) |
| OFS0 register settings > Independent WDT > Window Start Position | • 25%<br>• 50%<br>• 75%<br>• 100% (no window start position) | 100% (no window start position) |
| OFS0 register settings > Independent WDT > Reset Interrupt Request Select | • NMI request or interrupt request is enabled<br>• Reset is enabled | Reset is enabled |
| OFS0 register settings > Independent WDT > Stop Control | • Counting continues (Note: Device will not enter Deep Standby Mode when selected. Device will enter Software Standby Mode)<br>• Stop counting when in Sleep, Snooze mode, or Software Standby | Stop counting when in Sleep, Snooze mode, or Software Standby |
| OFS0 register settings > WDT > Start Mode Select | • Automatically activate WDT after a reset (auto-start mode)<br>• Stop WDT after a reset (register-start mode) | Stop WDT after a reset (register-start mode) |
| OFS0 register settings | • 1024 cycles | 16384 cycles |

| | | |
|---|---|---|
| > WDT > Timeout Period | • 4096 cycles<br>• 8192 cycles<br>• 16384 cycles | |
| OFS0 register settings > WDT > Clock Frequency Division Ratio | • 4<br>• 64<br>• 128<br>• 512<br>• 2048<br>• 8192 | 128 |
| OFS0 register settings > WDT > Window End Position | • 75%<br>• 50%<br>• 25%<br>• 0% (no window end position) | 0% (no window end position) |
| OFS0 register settings > WDT > Window Start Position | • 25%<br>• 50%<br>• 75%<br>• 100% (no window start position) | 100% (no window start position) |
| OFS0 register settings > WDT > Reset Interrupt Request | • NMI<br>• Reset | Reset |
| OFS0 register settings > WDT > Stop Control | • Counting continues<br>• Stop counting when entering Sleep mode | Stop counting when entering Sleep mode |
| OFS1 register settings > Voltage Detection 0 Circuit Start | • Voltage monitor 0 reset is enabled after reset<br>• Voltage monitor 0 reset is disabled after reset | Voltage monitor 0 reset is disabled after reset |
| OFS1 register settings > Voltage Detection 0 Level | • 2.94 V<br>• 2.87 V<br>• 2.80 V | 2.80 V |
| OFS1 register settings > HOCO Oscillation Enable | • HOCO oscillation is enabled after reset<br>• HOCO oscillation is disabled after reset | HOCO oscillation is disabled after reset |
| MPU > Enable or disable PC Region 0 | • Enabled<br>• Disabled | Disabled |

| | | |
|---|---|---|
| MPU > PC0 Start | Value must be an integer between 0 and 0xFFFFFFFC | 0xFFFFFFFC |
| MPU > PC0 End | Value must be an integer between 0x00000003 and 0xFFFFFFFF | 0xFFFFFFFF |
| MPU > Enable or disable PC Region 1 | • Enabled<br>• Disabled | Disabled |
| MPU > PC1 Start | Value must be an integer between 0 and 0xFFFFFFFC | 0xFFFFFFFC |
| MPU > PC1 End | Value must be an integer between 0x00000003 and 0xFFFFFFFF | 0xFFFFFFFF |
| MPU > Enable or disable Memory Region 0 | • Enabled<br>• Disabled | Disabled |
| MPU > Memory Region 0 Start | Value must be an integer between 0 and 0x00FFFFFC | 0x00FFFFFC |
| MPU > Memory Region 0 End | Value must be an integer between 0x00000003 and 0x00FFFFFF | 0x00FFFFFF |
| MPU > Enable or disable Memory Region 1 | • Enabled<br>• Disabled | Disabled |
| MPU > Memory Region 1 Start | Value must be an integer between 0x1FF00000 and 0x200FFFFC | 0x200FFFFC |
| MPU > Memory Region 1 End | Value must be an integer between 0x1FF00003 and 0x200FFFFF | 0x200FFFFF |
| MPU > Enable or disable Memory Region 2 | • Enabled<br>• Disabled | Disabled |
| MPU > Memory Region 2 Start | Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC | 0x407FFFFC |
| MPU > Memory Region | Value must be an | 0x407FFFFF |

| | | |
|---|---|---|
| 2 End | integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF | |
| MPU > Enable or disable Memory Region 3 | • Enabled<br>• Disabled | Disabled |
| MPU > Memory Region 3 Start | Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC | 0x400DFFFC |
| MPU > Memory Region 3 End | Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF | 0x400DFFFF |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |

### Enumeration Type Documentation

#### ◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

***Note***

*This list may change based on based on the device.*

### 4.1.2.5 RA6M3

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra6m3_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| | | | |
|---|---|---|---|

| Configuration | Options | Default | Description |
|---|---|---|---|
| OFS0 register settings > Independent WDT > Start Mode | • IWDT is Disabled<br>• IWDT is automatically activated after a reset (Autostart mode) | IWDT is Disabled | |
| OFS0 register settings > Independent WDT > Timeout Period | • 128 cycles<br>• 512 cycles<br>• 1024 cycles<br>• 2048 cycles | 2048 cycles | |
| OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor | • 1<br>• 16<br>• 32<br>• 64<br>• 128<br>• 256 | 128 | |
| OFS0 register settings > Independent WDT > Window End Position | • 75%<br>• 50%<br>• 25%<br>• 0% (no window end position) | 0% (no window end position) | |
| OFS0 register settings > Independent WDT > Window Start Position | • 25%<br>• 50%<br>• 75%<br>• 100% (no window start position) | 100% (no window start position) | |
| OFS0 register settings > Independent WDT > Reset Interrupt Request Select | • NMI request or interrupt request is enabled<br>• Reset is enabled | Reset is enabled | |
| OFS0 register settings > Independent WDT > Stop Control | • Counting continues (Note: Device will not enter Deep Standby Mode when selected. Device will enter Software Standby Mode)<br>• Stop counting when in Sleep, Snooze mode, or Software Standby | Stop counting when in Sleep, Snooze mode, or Software Standby | |

| | | |
|---|---|---|
| OFS0 register settings > WDT > Start Mode Select | • Automatically activate WDT after a reset (auto-start mode)<br>• Stop WDT after a reset (register-start mode) | Stop WDT after a reset (register-start mode) |
| OFS0 register settings > WDT > Timeout Period | • 1024 cycles<br>• 4096 cycles<br>• 8192 cycles<br>• 16384 cycles | 16384 cycles |
| OFS0 register settings > WDT > Clock Frequency Division Ratio | • 4<br>• 64<br>• 128<br>• 512<br>• 2048<br>• 8192 | 128 |
| OFS0 register settings > WDT > Window End Position | • 75%<br>• 50%<br>• 25%<br>• 0% (no window end position) | 0% (no window end position) |
| OFS0 register settings > WDT > Window Start Position | • 25%<br>• 50%<br>• 75%<br>• 100% (no window start position) | 100% (no window start position) |
| OFS0 register settings > WDT > Reset Interrupt Request | • NMI<br>• Reset | Reset |
| OFS0 register settings > WDT > Stop Control | • Counting continues<br>• Stop counting when entering Sleep mode | Stop counting when entering Sleep mode |
| OFS1 register settings > Voltage Detection 0 Circuit Start | • Voltage monitor 0 reset is enabled after reset<br>• Voltage monitor 0 reset is disabled after reset | Voltage monitor 0 reset is disabled after reset |
| OFS1 register settings > Voltage Detection 0 Level | • 2.94 V<br>• 2.87 V<br>• 2.80 V | 2.80 V |
| OFS1 register settings | • HOCO | HOCO oscillation is |

| | | |
|---|---|---|
| > HOCO Oscillation Enable | oscillation is enabled after reset<br>• HOCO oscillation is disabled after reset | disabled after reset |
| MPU > Enable or disable PC Region 0 | • Enabled<br>• Disabled | Disabled |
| MPU > PC0 Start | Value must be an integer between 0 and 0xFFFFFFFC | 0xFFFFFFFC |
| MPU > PC0 End | Value must be an integer between 0x00000003 and 0xFFFFFFFF | 0xFFFFFFFF |
| MPU > Enable or disable PC Region 1 | • Enabled<br>• Disabled | Disabled |
| MPU > PC1 Start | Value must be an integer between 0 and 0xFFFFFFFC | 0xFFFFFFFC |
| MPU > PC1 End | Value must be an integer between 0x00000003 and 0xFFFFFFFF | 0xFFFFFFFF |
| MPU > Enable or disable Memory Region 0 | • Enabled<br>• Disabled | Disabled |
| MPU > Memory Region 0 Start | Value must be an integer between 0 and 0x00FFFFFC | 0x00FFFFFC |
| MPU > Memory Region 0 End | Value must be an integer between 0x00000003 and 0x00FFFFFF | 0x00FFFFFF |
| MPU > Enable or disable Memory Region 1 | • Enabled<br>• Disabled | Disabled |
| MPU > Memory Region 1 Start | Value must be an integer between 0x1FF00000 and 0x200FFFFC | 0x200FFFFC |
| MPU > Memory Region 1 End | Value must be an integer between 0x1FF00003 and 0x200FFFFF | 0x200FFFFF |
| MPU > Enable or | • Enabled | Disabled |

| | | |
|---|---|---|
| disable Memory Region 2 | • Disabled | |
| MPU > Memory Region 2 Start | Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC | 0x407FFFFC |
| MPU > Memory Region 2 End | Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF | 0x407FFFFF |
| MPU > Enable or disable Memory Region 3 | • Enabled<br>• Disabled | Disabled |
| MPU > Memory Region 3 Start | Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC | 0x400DFFFC |
| MPU > Memory Region 3 End | Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF | 0x400DFFFF |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |

### Enumeration Type Documentation

#### ◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

***Note***
*This list may change based on based on the device.*

## 4.1.3 BSP I/O access

BSP

### Functions

| | |
|---|---|
| __STATIC_INLINE uint32_t | R_BSP_PinRead (bsp_io_port_pin_t pin) |
| __STATIC_INLINE void | R_BSP_PinWrite (bsp_io_port_pin_t pin, bsp_io_level_t level) |
| __STATIC_INLINE void | R_BSP_PinAccessEnable (void) |
| __STATIC_INLINE void | R_BSP_PinAccessDisable (void) |

### Detailed Description

This module provides basic read/write access to port pins.

### Enumerations

| | |
|---|---|
| enum | bsp_io_level_t |
| enum | bsp_io_direction_t |
| enum | bsp_io_port_t |
| enum | bsp_io_port_pin_t |

### Enumeration Type Documentation

#### ◆ bsp_io_level_t

| enum bsp_io_level_t | |
|---|---|
| Levels that can be set and read for individual pins | |
| Enumerator | |
| BSP_IO_LEVEL_LOW | Low. |
| BSP_IO_LEVEL_HIGH | High. |

#### ◆ bsp_io_direction_t

| enum bsp_io_direction_t | |
|---|---|
| Direction of individual pins | |
| Enumerator | |
| BSP_IO_DIRECTION_INPUT | Input. |
| BSP_IO_DIRECTION_OUTPUT | Output. |

### ◆ bsp_io_port_t

| enum bsp_io_port_t | |
|---|---|
| Superset list of all possible IO ports. | |
| Enumerator | |
| BSP_IO_PORT_00 | IO port 0. |
| BSP_IO_PORT_01 | IO port 1. |
| BSP_IO_PORT_02 | IO port 2. |
| BSP_IO_PORT_03 | IO port 3. |
| BSP_IO_PORT_04 | IO port 4. |
| BSP_IO_PORT_05 | IO port 5. |
| BSP_IO_PORT_06 | IO port 6. |
| BSP_IO_PORT_07 | IO port 7. |
| BSP_IO_PORT_08 | IO port 8. |
| BSP_IO_PORT_09 | IO port 9. |
| BSP_IO_PORT_10 | IO port 10. |
| BSP_IO_PORT_11 | IO port 11. |

### ◆ bsp_io_port_pin_t

| enum bsp_io_port_pin_t | |
|---|---|
| Superset list of all possible IO port pins. | |
| Enumerator | |
| BSP_IO_PORT_00_PIN_00 | IO port 0 pin 0. |
| BSP_IO_PORT_00_PIN_01 | IO port 0 pin 1. |
| BSP_IO_PORT_00_PIN_02 | IO port 0 pin 2. |
| BSP_IO_PORT_00_PIN_03 | IO port 0 pin 3. |
| BSP_IO_PORT_00_PIN_04 | IO port 0 pin 4. |

| | |
|---|---|
| BSP_IO_PORT_00_PIN_05 | IO port 0 pin 5. |
| BSP_IO_PORT_00_PIN_06 | IO port 0 pin 6. |
| BSP_IO_PORT_00_PIN_07 | IO port 0 pin 7. |
| BSP_IO_PORT_00_PIN_08 | IO port 0 pin 8. |
| BSP_IO_PORT_00_PIN_09 | IO port 0 pin 9. |
| BSP_IO_PORT_00_PIN_10 | IO port 0 pin 10. |
| BSP_IO_PORT_00_PIN_11 | IO port 0 pin 11. |
| BSP_IO_PORT_00_PIN_12 | IO port 0 pin 12. |
| BSP_IO_PORT_00_PIN_13 | IO port 0 pin 13. |
| BSP_IO_PORT_00_PIN_14 | IO port 0 pin 14. |
| BSP_IO_PORT_00_PIN_15 | IO port 0 pin 15. |
| BSP_IO_PORT_01_PIN_00 | IO port 1 pin 0. |
| BSP_IO_PORT_01_PIN_01 | IO port 1 pin 1. |
| BSP_IO_PORT_01_PIN_02 | IO port 1 pin 2. |
| BSP_IO_PORT_01_PIN_03 | IO port 1 pin 3. |
| BSP_IO_PORT_01_PIN_04 | IO port 1 pin 4. |
| BSP_IO_PORT_01_PIN_05 | IO port 1 pin 5. |
| BSP_IO_PORT_01_PIN_06 | IO port 1 pin 6. |
| BSP_IO_PORT_01_PIN_07 | IO port 1 pin 7. |
| BSP_IO_PORT_01_PIN_08 | IO port 1 pin 8. |
| BSP_IO_PORT_01_PIN_09 | IO port 1 pin 9. |
| BSP_IO_PORT_01_PIN_10 | IO port 1 pin 10. |
| BSP_IO_PORT_01_PIN_11 | IO port 1 pin 11. |
| BSP_IO_PORT_01_PIN_12 | IO port 1 pin 12. |

| | |
|---|---|
| BSP_IO_PORT_01_PIN_13 | IO port 1 pin 13. |
| BSP_IO_PORT_01_PIN_14 | IO port 1 pin 14. |
| BSP_IO_PORT_01_PIN_15 | IO port 1 pin 15. |
| BSP_IO_PORT_02_PIN_00 | IO port 2 pin 0. |
| BSP_IO_PORT_02_PIN_01 | IO port 2 pin 1. |
| BSP_IO_PORT_02_PIN_02 | IO port 2 pin 2. |
| BSP_IO_PORT_02_PIN_03 | IO port 2 pin 3. |
| BSP_IO_PORT_02_PIN_04 | IO port 2 pin 4. |
| BSP_IO_PORT_02_PIN_05 | IO port 2 pin 5. |
| BSP_IO_PORT_02_PIN_06 | IO port 2 pin 6. |
| BSP_IO_PORT_02_PIN_07 | IO port 2 pin 7. |
| BSP_IO_PORT_02_PIN_08 | IO port 2 pin 8. |
| BSP_IO_PORT_02_PIN_09 | IO port 2 pin 9. |
| BSP_IO_PORT_02_PIN_10 | IO port 2 pin 10. |
| BSP_IO_PORT_02_PIN_11 | IO port 2 pin 11. |
| BSP_IO_PORT_02_PIN_12 | IO port 2 pin 12. |
| BSP_IO_PORT_02_PIN_13 | IO port 2 pin 13. |
| BSP_IO_PORT_02_PIN_14 | IO port 2 pin 14. |
| BSP_IO_PORT_02_PIN_15 | IO port 2 pin 15. |
| BSP_IO_PORT_03_PIN_00 | IO port 3 pin 0. |
| BSP_IO_PORT_03_PIN_01 | IO port 3 pin 1. |
| BSP_IO_PORT_03_PIN_02 | IO port 3 pin 2. |
| BSP_IO_PORT_03_PIN_03 | IO port 3 pin 3. |
| BSP_IO_PORT_03_PIN_04 | IO port 3 pin 4. |

| | |
|---|---|
| BSP_IO_PORT_03_PIN_05 | IO port 3 pin 5. |
| BSP_IO_PORT_03_PIN_06 | IO port 3 pin 6. |
| BSP_IO_PORT_03_PIN_07 | IO port 3 pin 7. |
| BSP_IO_PORT_03_PIN_08 | IO port 3 pin 8. |
| BSP_IO_PORT_03_PIN_09 | IO port 3 pin 9. |
| BSP_IO_PORT_03_PIN_10 | IO port 3 pin 10. |
| BSP_IO_PORT_03_PIN_11 | IO port 3 pin 11. |
| BSP_IO_PORT_03_PIN_12 | IO port 3 pin 12. |
| BSP_IO_PORT_03_PIN_13 | IO port 3 pin 13. |
| BSP_IO_PORT_03_PIN_14 | IO port 3 pin 14. |
| BSP_IO_PORT_03_PIN_15 | IO port 3 pin 15. |
| BSP_IO_PORT_04_PIN_00 | IO port 4 pin 0. |
| BSP_IO_PORT_04_PIN_01 | IO port 4 pin 1. |
| BSP_IO_PORT_04_PIN_02 | IO port 4 pin 2. |
| BSP_IO_PORT_04_PIN_03 | IO port 4 pin 3. |
| BSP_IO_PORT_04_PIN_04 | IO port 4 pin 4. |
| BSP_IO_PORT_04_PIN_05 | IO port 4 pin 5. |
| BSP_IO_PORT_04_PIN_06 | IO port 4 pin 6. |
| BSP_IO_PORT_04_PIN_07 | IO port 4 pin 7. |
| BSP_IO_PORT_04_PIN_08 | IO port 4 pin 8. |
| BSP_IO_PORT_04_PIN_09 | IO port 4 pin 9. |
| BSP_IO_PORT_04_PIN_10 | IO port 4 pin 10. |
| BSP_IO_PORT_04_PIN_11 | IO port 4 pin 11. |
| BSP_IO_PORT_04_PIN_12 | IO port 4 pin 12. |

| | |
|---|---|
| BSP_IO_PORT_04_PIN_13 | IO port 4 pin 13. |
| BSP_IO_PORT_04_PIN_14 | IO port 4 pin 14. |
| BSP_IO_PORT_04_PIN_15 | IO port 4 pin 15. |
| BSP_IO_PORT_05_PIN_00 | IO port 5 pin 0. |
| BSP_IO_PORT_05_PIN_01 | IO port 5 pin 1. |
| BSP_IO_PORT_05_PIN_02 | IO port 5 pin 2. |
| BSP_IO_PORT_05_PIN_03 | IO port 5 pin 3. |
| BSP_IO_PORT_05_PIN_04 | IO port 5 pin 4. |
| BSP_IO_PORT_05_PIN_05 | IO port 5 pin 5. |
| BSP_IO_PORT_05_PIN_06 | IO port 5 pin 6. |
| BSP_IO_PORT_05_PIN_07 | IO port 5 pin 7. |
| BSP_IO_PORT_05_PIN_08 | IO port 5 pin 8. |
| BSP_IO_PORT_05_PIN_09 | IO port 5 pin 9. |
| BSP_IO_PORT_05_PIN_10 | IO port 5 pin 10. |
| BSP_IO_PORT_05_PIN_11 | IO port 5 pin 11. |
| BSP_IO_PORT_05_PIN_12 | IO port 5 pin 12. |
| BSP_IO_PORT_05_PIN_13 | IO port 5 pin 13. |
| BSP_IO_PORT_05_PIN_14 | IO port 5 pin 14. |
| BSP_IO_PORT_05_PIN_15 | IO port 5 pin 15. |
| BSP_IO_PORT_06_PIN_00 | IO port 6 pin 0. |
| BSP_IO_PORT_06_PIN_01 | IO port 6 pin 1. |
| BSP_IO_PORT_06_PIN_02 | IO port 6 pin 2. |
| BSP_IO_PORT_06_PIN_03 | IO port 6 pin 3. |
| BSP_IO_PORT_06_PIN_04 | IO port 6 pin 4. |

| | |
|---|---|
| BSP_IO_PORT_06_PIN_05 | IO port 6 pin 5. |
| BSP_IO_PORT_06_PIN_06 | IO port 6 pin 6. |
| BSP_IO_PORT_06_PIN_07 | IO port 6 pin 7. |
| BSP_IO_PORT_06_PIN_08 | IO port 6 pin 8. |
| BSP_IO_PORT_06_PIN_09 | IO port 6 pin 9. |
| BSP_IO_PORT_06_PIN_10 | IO port 6 pin 10. |
| BSP_IO_PORT_06_PIN_11 | IO port 6 pin 11. |
| BSP_IO_PORT_06_PIN_12 | IO port 6 pin 12. |
| BSP_IO_PORT_06_PIN_13 | IO port 6 pin 13. |
| BSP_IO_PORT_06_PIN_14 | IO port 6 pin 14. |
| BSP_IO_PORT_06_PIN_15 | IO port 6 pin 15. |
| BSP_IO_PORT_07_PIN_00 | IO port 7 pin 0. |
| BSP_IO_PORT_07_PIN_01 | IO port 7 pin 1. |
| BSP_IO_PORT_07_PIN_02 | IO port 7 pin 2. |
| BSP_IO_PORT_07_PIN_03 | IO port 7 pin 3. |
| BSP_IO_PORT_07_PIN_04 | IO port 7 pin 4. |
| BSP_IO_PORT_07_PIN_05 | IO port 7 pin 5. |
| BSP_IO_PORT_07_PIN_06 | IO port 7 pin 6. |
| BSP_IO_PORT_07_PIN_07 | IO port 7 pin 7. |
| BSP_IO_PORT_07_PIN_08 | IO port 7 pin 8. |
| BSP_IO_PORT_07_PIN_09 | IO port 7 pin 9. |
| BSP_IO_PORT_07_PIN_10 | IO port 7 pin 10. |
| BSP_IO_PORT_07_PIN_11 | IO port 7 pin 11. |
| BSP_IO_PORT_07_PIN_12 | IO port 7 pin 12. |

| | |
|---|---|
| BSP_IO_PORT_07_PIN_13 | IO port 7 pin 13. |
| BSP_IO_PORT_07_PIN_14 | IO port 7 pin 14. |
| BSP_IO_PORT_07_PIN_15 | IO port 7 pin 15. |
| BSP_IO_PORT_08_PIN_00 | IO port 8 pin 0. |
| BSP_IO_PORT_08_PIN_01 | IO port 8 pin 1. |
| BSP_IO_PORT_08_PIN_02 | IO port 8 pin 2. |
| BSP_IO_PORT_08_PIN_03 | IO port 8 pin 3. |
| BSP_IO_PORT_08_PIN_04 | IO port 8 pin 4. |
| BSP_IO_PORT_08_PIN_05 | IO port 8 pin 5. |
| BSP_IO_PORT_08_PIN_06 | IO port 8 pin 6. |
| BSP_IO_PORT_08_PIN_07 | IO port 8 pin 7. |
| BSP_IO_PORT_08_PIN_08 | IO port 8 pin 8. |
| BSP_IO_PORT_08_PIN_09 | IO port 8 pin 9. |
| BSP_IO_PORT_08_PIN_10 | IO port 8 pin 10. |
| BSP_IO_PORT_08_PIN_11 | IO port 8 pin 11. |
| BSP_IO_PORT_08_PIN_12 | IO port 8 pin 12. |
| BSP_IO_PORT_08_PIN_13 | IO port 8 pin 13. |
| BSP_IO_PORT_08_PIN_14 | IO port 8 pin 14. |
| BSP_IO_PORT_08_PIN_15 | IO port 8 pin 15. |
| BSP_IO_PORT_09_PIN_00 | IO port 9 pin 0. |
| BSP_IO_PORT_09_PIN_01 | IO port 9 pin 1. |
| BSP_IO_PORT_09_PIN_02 | IO port 9 pin 2. |
| BSP_IO_PORT_09_PIN_03 | IO port 9 pin 3. |
| BSP_IO_PORT_09_PIN_04 | IO port 9 pin 4. |

| | |
|---|---|
| BSP_IO_PORT_09_PIN_05 | IO port 9 pin 5. |
| BSP_IO_PORT_09_PIN_06 | IO port 9 pin 6. |
| BSP_IO_PORT_09_PIN_07 | IO port 9 pin 7. |
| BSP_IO_PORT_09_PIN_08 | IO port 9 pin 8. |
| BSP_IO_PORT_09_PIN_09 | IO port 9 pin 9. |
| BSP_IO_PORT_09_PIN_10 | IO port 9 pin 10. |
| BSP_IO_PORT_09_PIN_11 | IO port 9 pin 11. |
| BSP_IO_PORT_09_PIN_12 | IO port 9 pin 12. |
| BSP_IO_PORT_09_PIN_13 | IO port 9 pin 13. |
| BSP_IO_PORT_09_PIN_14 | IO port 9 pin 14. |
| BSP_IO_PORT_09_PIN_15 | IO port 9 pin 15. |
| BSP_IO_PORT_10_PIN_00 | IO port 10 pin 0. |
| BSP_IO_PORT_10_PIN_01 | IO port 10 pin 1. |
| BSP_IO_PORT_10_PIN_02 | IO port 10 pin 2. |
| BSP_IO_PORT_10_PIN_03 | IO port 10 pin 3. |
| BSP_IO_PORT_10_PIN_04 | IO port 10 pin 4. |
| BSP_IO_PORT_10_PIN_05 | IO port 10 pin 5. |
| BSP_IO_PORT_10_PIN_06 | IO port 10 pin 6. |
| BSP_IO_PORT_10_PIN_07 | IO port 10 pin 7. |
| BSP_IO_PORT_10_PIN_08 | IO port 10 pin 8. |
| BSP_IO_PORT_10_PIN_09 | IO port 10 pin 9. |
| BSP_IO_PORT_10_PIN_10 | IO port 10 pin 10. |
| BSP_IO_PORT_10_PIN_11 | IO port 10 pin 11. |
| BSP_IO_PORT_10_PIN_12 | IO port 10 pin 12. |

| | |
|---|---|
| BSP_IO_PORT_10_PIN_13 | IO port 10 pin 13. |
| BSP_IO_PORT_10_PIN_14 | IO port 10 pin 14. |
| BSP_IO_PORT_10_PIN_15 | IO port 10 pin 15. |
| BSP_IO_PORT_11_PIN_00 | IO port 11 pin 0. |
| BSP_IO_PORT_11_PIN_01 | IO port 11 pin 1. |
| BSP_IO_PORT_11_PIN_02 | IO port 11 pin 2. |
| BSP_IO_PORT_11_PIN_03 | IO port 11 pin 3. |
| BSP_IO_PORT_11_PIN_04 | IO port 11 pin 4. |
| BSP_IO_PORT_11_PIN_05 | IO port 11 pin 5. |
| BSP_IO_PORT_11_PIN_06 | IO port 11 pin 6. |
| BSP_IO_PORT_11_PIN_07 | IO port 11 pin 7. |
| BSP_IO_PORT_11_PIN_08 | IO port 11 pin 8. |
| BSP_IO_PORT_11_PIN_09 | IO port 11 pin 9. |
| BSP_IO_PORT_11_PIN_10 | IO port 11 pin 10. |
| BSP_IO_PORT_11_PIN_11 | IO port 11 pin 11. |
| BSP_IO_PORT_11_PIN_12 | IO port 11 pin 12. |
| BSP_IO_PORT_11_PIN_13 | IO port 11 pin 13. |
| BSP_IO_PORT_11_PIN_14 | IO port 11 pin 14. |
| BSP_IO_PORT_11_PIN_15 | IO port 11 pin 15. |

## Function Documentation

### ◆ R_BSP_PinRead()

__STATIC_INLINE uint32_t R_BSP_PinRead ( bsp_io_port_pin_t *pin*)

Read the current input level of the pin.

**Parameters**

| | | |
|---|---|---|
| [in] | pin | The pin |

**Return values**

| | |
|---|---|
| Current | input level |

### ◆ R_BSP_PinWrite()

__STATIC_INLINE void R_BSP_PinWrite ( bsp_io_port_pin_t *pin*, bsp_io_level_t *level* )

Set a pin to output and set the output level to the level provided

**Parameters**

| | | |
|---|---|---|
| [in] | pin | The pin |
| [in] | level | The level |

### ◆ R_BSP_PinAccessEnable()

__STATIC_INLINE void R_BSP_PinAccessEnable ( void )

Enable access to the PFS registers. Uses a reference counter to protect against interrupts that could occur via multiple threads or an ISR re-entering this code.

### ◆ R_BSP_PinAccessDisable()

__STATIC_INLINE void R_BSP_PinAccessDisable ( void )

Disable access to the PFS registers. Uses a reference counter to protect against interrupts that could occur via multiple threads or an ISR re-entering this code.

## 4.2 Modules

### Detailed Description

Modules are the smallest unit of software available in the FSP. Each module implements one interface.

## Modules

| | |
|---|---|
| | High-Speed Analog Comparator (r_acmphs)<br><br>Driver for the ACMPHS peripheral on RA MCUs. This module implements the Comparator Interface. |
| | Low-Power Analog Comparator (r_acmplp)<br><br>Driver for the ACMPLP peripheral on RA MCUs. This module implements the Comparator Interface. |
| | Analog to Digital Converter (r_adc)<br><br>Driver for the ADC12, ADC14, and ADC16 peripherals on RA MCUs. This module implements the ADC Interface. |
| | Asynchronous General Purpose Timer (r_agt)<br><br>Driver for the AGT peripheral on RA MCUs. This module implements the Timer Interface. |
| | Bluetooth Low Energy Library (r_ble)<br><br>Driver for the Radio peripheral on RA MCUs. This module implements the BLE Interface. |
| | Clock Frequency Accuracy Measurement Circuit (r_cac)<br><br>Driver for the CAC peripheral on RA MCUs. This module implements the CAC Interface. |
| | Controller Area Network (r_can)<br><br>Driver for the CAN peripheral on RA MCUs. This module implements the CAN Interface. |
| | Clock Generation Circuit (r_cgc)<br><br>Driver for the CGC peripheral on RA MCUs. This module implements the CGC Interface. |
| | Cyclic Redundancy Check (CRC) Calculator (r_crc)<br><br>Driver for the CRC peripheral on RA MCUs. This module implements the CRC Interface. |
| | |

| | |
|---|---|
| | Capacitive Touch Sensing Unit (r_ctsu)<br><br>This HAL driver supports the Capacitive Touch Sensing Unit (CTSU). It implements the CTSU Interface. |
| | Digital to Analog Converter (r_dac)<br><br>Driver for the DAC12 peripheral on RA MCUs. This module implements the DAC Interface. |
| | Digital to Analog Converter (r_dac8)<br><br>Driver for the DAC8 peripheral on RA MCUs. This module implements the DAC Interface. |
| | Direct Memory Access Controller (r_dmac)<br><br>Driver for the DMAC peripheral on RA MCUs. This module implements the Transfer Interface. |
| | Data Operation Circuit (r_doc)<br><br>Driver for the DOC peripheral on RA MCUs. This module implements the DOC Interface. |
| | D/AVE 2D Port Interface (r_drw)<br><br>Driver for the DRW peripheral on RA MCUs. This module is a port of D/AVE 2D. |
| | Data Transfer Controller (r_dtc)<br><br>Driver for the DTC peripheral on RA MCUs. This module implements the Transfer Interface. |
| | Event Link Controller (r_elc)<br><br>Driver for the ELC peripheral on RA MCUs. This module implements the ELC Interface. |
| | Ethernet (r_ether)<br><br>Driver for the Ethernet peripheral on RA MCUs. This module implements the Ethernet Interface. |
| | Ethernet PHY (r_ether_phy) |

| | |
|---|---|
| | The Ethernet PHY module (r_ether_phy) provides an API for standard Ethernet PHY communications applications that use the ETHERC peripheral. It implements the Ethernet PHY Interface. |
| | High-Performance Flash Driver (r_flash_hp)<br><br>Driver for the flash memory on RA high-performance MCUs. This module implements the Flash Interface. |
| | Low-Power Flash Driver (r_flash_lp)<br><br>Driver for the flash memory on RA low-power MCUs. This module implements the Flash Interface. |
| | Graphics LCD Controller (r_glcdc)<br><br>Driver for the GLCDC peripheral on RA MCUs. This module implements the Display Interface. |
| | General PWM Timer (r_gpt)<br><br>Driver for the GPT32 and GPT16 peripherals on RA MCUs. This module implements the Timer Interface. |
| | General PWM Timer Three-Phase Motor Control Driver (r_gpt_three_phase)<br><br>Driver for 3-phase motor control using the GPT peripheral on RA MCUs. This module implements the Three-Phase Interface. |
| | Interrupt Controller Unit (r_icu)<br><br>Driver for the ICU peripheral on RA MCUs. This module implements the External IRQ Interface. |
| | I2C Master on IIC (r_iic_master)<br><br>Driver for the IIC peripheral on RA MCUs. This module implements the I2C Master Interface. |
| | I2C Slave on IIC (r_iic_slave)<br><br>Driver for the IIC peripheral on RA MCUs. This module implements the I2C Slave Interface. |
| | I/O Ports (r_ioport) |

| | |
|---|---|
| | Driver for the I/O Ports peripheral on RA MCUs. This module implements the I/O Port Interface. |
| | Independent Watchdog Timer (r_iwdt)<br><br>Driver for the IWDT peripheral on RA MCUs. This module implements the WDT Interface. |
| | JPEG Codec (r_jpeg)<br><br>Driver for the JPEG peripheral on RA MCUs. This module implements the JPEG Codec Interface. |
| | Key Interrupt (r_kint)<br><br>Driver for the KINT peripheral on RA MCUs. This module implements the Key Matrix Interface. |
| | Low Power Modes (r_lpm)<br><br>Driver for the LPM peripheral on RA MCUs. This module implements the Low Power Modes Interface. |
| | Low Voltage Detection (r_lvd)<br><br>Driver for the LVD peripheral on RA MCUs. This module implements the Low Voltage Detection Interface. |
| | Operational Amplifier (r_opamp)<br><br>Driver for the OPAMP peripheral on RA MCUs. This module implements the OPAMP Interface. |
| | Port Output Enable for GPT (r_poeg)<br><br>Driver for the POEG peripheral on RA MCUs. This module implements the POEG Interface. |
| | Quad Serial Peripheral Interface Flash (r_qspi)<br><br>Driver for the QSPI peripheral on RA MCUs. This module implements the SPI Flash Interface. |
| | Realtime Clock (r_rtc)<br><br>Driver for the RTC peripheral on RA MCUs. This module implements |

| | |
|---|---|
| | the RTC Interface. |
| | Serial Communications Interface (SCI) I2C (r_sci_i2c)<br><br>Driver for the SCI peripheral on RA MCUs. This module implements the I2C Master Interface. |
| | Serial Communications Interface (SCI) SPI (r_sci_spi)<br><br>Driver for the SCI peripheral on RA MCUs. This module implements the SPI Interface. |
| | Serial Communications Interface (SCI) UART (r_sci_uart)<br><br>Driver for the SCI peripheral on RA MCUs. This module implements the UART Interface. |
| | Sigma Delta Analog to Digital Converter (r_sdadc)<br><br>Driver for the SDADC24 peripheral on RA MCUs. This module implements the ADC Interface. |
| | SD/MMC Host Interface (r_sdhi)<br><br>Driver for the SD/MMC Host Interface (SDHI) peripheral on RA MCUs. This module implements the SD/MMC Interface. |
| | Segment LCD Controller (r_slcdc)<br><br>Driver for the SLCDC peripheral on RA MCUs. This module implements the SLCDC Interface. |
| | Serial Peripheral Interface (r_spi)<br><br>Driver for the SPI peripheral on RA MCUs. This module implements the SPI Interface. |
| | Serial Sound Interface (r_ssi)<br><br>Driver for the SSIE peripheral on RA MCUs. This module implements the I2S Interface. |
| | USB (r_usb_basic)<br><br>The USB module (r_usb_basic) provides an API to perform hardware control of USB communication. It implements the USB Interface. |

| | |
|---|---|
| | USB Host Communications Device Class Driver (r_usb_hcdc)<br><br>This module is USB Host Communication Device Class Driver (HCDC). It implements the USB HCDC Interface. This module works in combination with (r_usb_basic module). |
| | USB Host Human Interface Device Class Driver (r_usb_hhid)<br><br>The USB module (r_usb_hhid) provides an API to perform hardware control of USB communications. It implements the USB HHID Interface. |
| | USB Host Mass Storage Class Driver (r_usb_hmsc)<br><br>The USB module (r_usb_hmsc) provides an API to perform hardware control of USB communications. It implements the USB HMSC Interface. |
| | USB Peripheral Communication Device Class (r_usb_pcdc)<br><br>This module is USB Peripheral Communication Device Class Driver (PCDC). It implements the USB PCDC Interface. This module works in combination with (r_usb_basic module). |
| | USB Peripheral Human Interface Device Class (r_usb_phid)<br><br>This module is USB Peripheral Human Interface Device Class Driver (PHID). It implements the USB PHID Interface.<br>This module works in combination with (r_usb_basic module). |
| | USB Peripheral Mass Storage Class (r_usb_pmsc)<br><br>This module is USB Peripheral Mass Storage Class Driver (PMSC). It implements the USB PMSC Interface. This module works in combination with (r_usb_basic module). |
| | Watchdog Timer (r_wdt)<br><br>Driver for the WDT peripheral on RA MCUs. This module implements the WDT Interface. |
| | AWS PKCS11 PAL (rm_aws_pkcs11_pal)<br><br>PKCS#11 PAL layer implementation for use by FreeRTOS TLS. |
| | AWS PKCS11 PAL LITTLEFS (rm_aws_pkcs11_pal_littlefs) |

| | |
|---|---|
| | PKCS#11 PAL LittleFS layer implementation for use by FreeRTOS TLS. |
| | Bluetooth Low Energy Abstraction (rm_ble_abs)<br><br>Middleware for the Bluetooth peripheral on RA MCUs. This module implements the BLE ABS Interface. |
| | SD/MMC Block Media Implementation (rm_block_media_sdmmc)<br><br>Middleware to implement the block media interface on SD cards. This module implements the Block Media Interface. |
| | USB HMSC Block Media Implementation (rm_block_media_usb)<br><br>Middleware to implement the block media interface on USB mass storage devices. This module implements the Block Media Interface. |
| | SEGGER emWin Port (rm_emwin_port)<br><br>SEGGER emWin port for RA MCUs. |
| | FreeRTOS+FAT Port (rm_freertos_plus_fat)<br><br>Middleware for the FAT File System control on RA MCUs. |
| | FreeRTOS Plus TCP (rm_freertos_plus_tcp)<br><br>Middleware for using TCP on RA MCUs. |
| | FreeRTOS Port (rm_freertos_port)<br><br>FreeRTOS port for RA MCUs. |
| | LittleFS Flash Port (rm_littlefs_flash)<br><br>Middleware for the LittleFS File System control on RA MCUs. |
| | Crypto Middleware (rm_psa_crypto)<br><br>Hardware acceleration for the mbedCrypto implementation of the ARM PSA Crypto API. |
| | Capacitive Touch Middleware (rm_touch)<br><br>This module supports the Capacitive Touch Sensing Unit (CTSU). It |

| | |
|---|---|
| | implements the Touch Middleware Interface. |
| | AWS Device Provisioning<br>AWS Device Provisioning example software. |
| | AWS MQTT<br>This module provides the AWS MQTT integration documentation. |
| | Wifi Middleware (rm_wifi_onchip_silex)<br>Wifi and Socket implementation using the Silex SX-ULPGN WiFi module on RA MCUs. |
| | AWS Secure Sockets<br>This module provides the AWS Secure Sockets implementation. |

## 4.2.1 High-Speed Analog Comparator (r_acmphs)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | R_ACMPHS_Open (comparator_ctrl_t *p_ctrl, comparator_cfg_t const *const p_cfg) |
| fsp_err_t | R_ACMPHS_OutputEnable (comparator_ctrl_t *const p_ctrl) |
| fsp_err_t | R_ACMPHS_InfoGet (comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info) |
| fsp_err_t | R_ACMPHS_StatusGet (comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status) |
| fsp_err_t | R_ACMPHS_Close (comparator_ctrl_t *const p_ctrl) |
| fsp_err_t | R_ACMPHS_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Driver for the ACMPHS peripheral on RA MCUs. This module implements the Comparator Interface.

# Overview

### Features

The ACMPHS HAL module supports the following features:

- Callback on rising edge, falling edge or both
- Configurable debounce filter
- Option for comparator output on VCOUT pin
- ELC event output

# Configuration

### Build Time Configurations for r_acmphs

The following build time configurations are defined in fsp_cfg/r_acmphs_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Analog > Comparator Driver on r_acmphs

This module can be added to the Stacks tab via New Stack > Driver > Analog > Comparator Driver on r_acmphs:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_comparator0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Select the hardware channel. |
| Trigger Edge Selector | • Rising<br>• Falling<br>• Both Edge | Both Edge | The trigger specifies when a comparator callback event should occur. Unused if the interrupt priority is disabled or the callback is NULL. |
| Noise Filter | • No Filter<br>• 8<br>• 16<br>• 32 | 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. |

| | | | |
|---|---|---|---|
| Maximum status retries (CMPMON) | Must be a valid non-negative integer between 2 and 32-bit maximum value | 1024 | Maximum number of status retries. |
| Output Polarity | • Not Inverted<br>• Inverted | Not Inverted | When enabled comparator output is inverted. This affects the output read from R_ACMPHS_StatusGet() , the pin output level, and the edge trigger. |
| Pin Output(VCOUT) | • Disabled<br>• Enabled | Disabled | Turn this on to include the output from this comparator on VCOUT. The comparator output on VCOUT is OR'd with output from all other ACMPHS and ACMPLP comparators. |
| Callback | Name must be a valid C symbol | NULL | Define this function in the application. It is called when the Trigger event occurs. |
| Comparator Interrupt Priority | MCU Specific Options | | Select the interrupt priority for the comparator interrupt. |
| Analog Input Voltage Source (IVCMP) | MCU Specific Options | | Select the Analog input voltage source. Channel mentioned in the options represents channel in ACMPHS |
| Reference Voltage Input Source (IVREF) | MCU Specific Options | | Select the Analog reference voltage source. Channel mentioned in the options represents channel in ACMPHS |

### Clock Configuration

The ACMPHS peripheral is clocked from PCLKB. You can set the PCLKB frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

### Pin Configuration

Comparator output can be enabled or disabled on each channel individually. The VCOUT pin is a logical OR of all comparator outputs.

The IVCMPn pins are used as comparator inputs. The IVREFn pins are used as comparator reference values.

# Usage Notes

### Noise Filter

When the noise filter is enabled, the ACMPHP0/ACMPHP1 signal is sampled three times based on the sampling clock selected. The filter clock frequency is determined by PCLKB and the comparator_filter_t setting.

### Output Polarity

If output polarity is configured as "Inverted" then the VCOUT signal will be inverted and the R_ACMPHS_StatusGet() will return an inverted status.

### Limitations

- Once the analog comparator is configured, the program must wait for the stabilization time to elapse before using the comparator.
- When the noise filter is not enabled the hardware requires software debouncing of the output (two consecutive equal values). This is automatically managed in R_ACMPHS_StatusGet but may result in delay or an API error in rare edge cases.
- Constraints apply on the simultaneous use of ACMPHS analog input and ADC analog input. Refer to the "Usage Notes" section in your MCU's User's Manual for the ADC unit(s) for more details.
- To allow ACMPHS0 to cancel Software Standby mode or enter Snooze, set the CSTEN bit to 1 and the CDFS bits to 00 in the CMPCTL0 register.

# Examples

### Basic Example

The following is a basic example of minimal use of the ACMPHS. The comparator is configured to trigger a callback when the input rises above the internal reference voltage (VREF). A GPIO output acts as the comparator input and is externally connected to the IVCMP input of the ACMPHS.

```
/* Connect this control pin to the VCMP input of the comparator. This can be any GPIO
pin
 * that is not input only. */
#define ACMPHS_EXAMPLE_CONTROL_PIN (BSP_IO_PORT_05_PIN_03)
#define ADC_PGA_BYPASS_VALUE (0x9999)
volatile uint32_t g_comparator_events = 0U;
/* This callback is called when a comparator event occurs. */
void acmphs_example_callback (comparator_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
    g_comparator_events++;
}
```

```
void acmphs_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /* Disable pin register write protection, if enabled */
 R_BSP_PinAccessEnable();
 /* Start with the VCMP pin low. This example assumes the comparator is configured to
trigger
  * when VCMP rises above VREF. */
    (void) R_BSP_PinWrite(ACMPHS_EXAMPLE_CONTROL_PIN, BSP_IO_LEVEL_LOW);
 /* Initialize the ACMPHS module */
    err = R_ACMPHS_Open(&g_comparator_ctrl, &g_comparator_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Bypass PGA on ADC unit 0.
  * (See Table 50.2 "Input source configuration of the ACMPHS" in the RA6M3 User's
Manual (R01UH0886EJ0100)) */
 R_BSP_MODULE_START(FSP_IP_ADC, 0);
    R_ADC0->ADPGACR   = ADC_PGA_BYPASS_VALUE;
    R_ADC0->ADPGADCR0 = 0;
 /* Wait for the minimum stabilization wait time before enabling output. */
 comparator_info_t info;
 R_ACMPHS_InfoGet(&g_comparator_ctrl, &info);
 R_BSP_SoftwareDelay(info.min_stabilization_wait_us, BSP_DELAY_UNITS_MICROSECONDS);
 /* Enable the comparator output */
    (void) R_ACMPHS_OutputEnable(&g_comparator_ctrl);
 /* Set the VCMP pin high. */
    (void) R_BSP_PinWrite(ACMPHS_EXAMPLE_CONTROL_PIN, BSP_IO_LEVEL_HIGH);
 while (0 == g_comparator_events)
    {
 /* Wait for interrupt. */
    }
 comparator_status_t status;
 /* Check status of comparator, Status will be COMPARATOR_STATE_OUTPUT_HIGH */
    (void) R_ACMPHS_StatusGet(&g_comparator_ctrl, &status);
```

```
}
```

## Function Documentation

### ◆ R_ACMPHS_Open()

fsp_err_t R_ACMPHS_Open ( comparator_ctrl_t *const *p_ctrl*, comparator_cfg_t const *const *p_cfg* )

Configures the comparator and starts operation. Callbacks and pin output are not active until outputEnable() is called. comparator_api_t::outputEnable() should be called after the output has stabilized. Implements comparator_api_t::open().

Comparator inputs must be configured in the application code prior to calling this function.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An input pointer is NULL |
| FSP_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. |
| FSP_ERR_ALREADY_OPEN | The control block is already open or the hardware lock is taken. |

### ◆ R_ACMPHS_OutputEnable()

fsp_err_t R_ACMPHS_OutputEnable ( comparator_ctrl_t *const *p_ctrl*)

Enables the comparator output, which can be polled using comparator_api_t::statusGet(). Also enables pin output and interrupts as configured during comparator_api_t::open(). Implements comparator_api_t::outputEnable().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Comparator output is enabled. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_ACMPHS_InfoGet()

fsp_err_t R_ACMPHS_InfoGet ( comparator_ctrl_t *const *p_ctrl*, comparator_info_t *const *p_info* )

Provides the minimum stabilization wait time in microseconds. Implements comparator_api_t::infoGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in p_info. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_ACMPHS_StatusGet()

fsp_err_t R_ACMPHS_StatusGet ( comparator_ctrl_t *const *p_ctrl*, comparator_status_t *const *p_status* )

Provides the operating status of the comparator. Implements comparator_api_t::statusGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operating status of the comparator is provided in p_status. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |
| FSP_ERR_TIMEOUT | The debounce filter is off and 2 consecutive matching values were not read within 1024 attempts. |

### ◆ R_ACMPHS_Close()

fsp_err_t R_ACMPHS_Close ( comparator_ctrl_t * *p_ctrl*)

Stops the comparator. Implements comparator_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Instance control block closed successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_ACMPHS_VersionGet()

| fsp_err_t R_ACMPHS_VersionGet ( fsp_version_t *const *p_version*) |
|---|
| Gets the API and code version. Implements comparator_api_t::versionGet(). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version information available in p_version. |
| FSP_ERR_ASSERTION | The parameter p_version is NULL. |

## 4.2.2 Low-Power Analog Comparator (r_acmplp)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_ACMPLP_Open (comparator_ctrl_t *const p_ctrl, comparator_cfg_t const *const p_cfg) |
| fsp_err_t | R_ACMPLP_OutputEnable (comparator_ctrl_t *const p_ctrl) |
| fsp_err_t | R_ACMPLP_InfoGet (comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info) |
| fsp_err_t | R_ACMPLP_StatusGet (comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status) |
| fsp_err_t | R_ACMPLP_Close (comparator_ctrl_t *const p_ctrl) |
| fsp_err_t | R_ACMPLP_VersionGet (fsp_version_t *const p_version) |

**Detailed Description**

Driver for the ACMPLP peripheral on RA MCUs. This module implements the Comparator Interface.

# Overview

**Features**

The ACMPLP HAL module supports the following features:

- Normal mode or window mode
- Callback on rising edge, falling edge or both

- Configurable debounce filter
- Option for comparator output on VCOUT pin
- ELC event output

# Configuration

### Build Time Configurations for r_acmplp

The following build time configurations are defined in fsp_cfg/r_acmplp_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Reference Voltage Selection (ACMPLP1) | • IVREF0<br>• IVREF1 | IVREF1 | Reference Voltage Selection for ACMPLP1. When set to IVREF0, configure the reference for ACMPLP channel 1 (if used) to one of the channel 0 sources. |

### Configurations for Driver > Analog > Comparator Driver on r_acmplp

This module can be added to the Stacks tab via New Stack > Driver > Analog > Comparator Driver on r_acmplp:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_comparator0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Select the hardware channel. |
| Mode | • Standard<br>• Window | Standard | In standard 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 | • Rising<br>• Falling<br>• Both Edge | 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 | • No sampling (bypass)<br>• Sampling at PCLKB<br>• Sampling at PCLKB/8<br>• Sampling at PCLKB/32 | No sampling (bypass) | Select the PCLK divisor for the hardware digital debounce filter. Larger divisors provide a longer debounce and take longer for the output to update. |
| Output Polarity | • Not Inverted<br>• Inverted | Not Inverted | When enabled comparator output is inverted. This affects the output read from R_ACMPLP_StatusGet(), the pin output level, and the edge trigger. |
| Pin Output (VCOUT) | • Off<br>• On | Off | Turn this on to include the output from this comparator on VCOUT. The comparator output on VCOUT is OR'd with output from all other ACMPHS and ACMPLP comparators. |
| Vref (Standard mode only) | • Enabled<br>• Disabled | Disabled | If reference voltage selection is enabled then internal reference voltage is used as comparator input |
| Callback | Name must be a valid C symbol | NULL | Define this function in the application. It is called when the Trigger event occurs. |
| Comparator Interrupt Priority | MCU Specific Options | | Select the interrupt priority for the comparator interrupt. |
| Analog Input Voltage Source (IVCMP) | MCU Specific Options | | Select the comparator input source. Only options for the configured channel are valid. |
| Reference Voltage Input Source (IVREF) | MCU Specific Options | | Select the comparator reference voltage source.<br><br>If channel 1 is seleected and the 'Reference Voltage Selection (ACMPLP1)' config option is set to |

IVREF0, select one of the Channel 0 options. In all other cases, only options for the configured channel are valid.

### Clock Configuration

The ACMPLP peripheral is clocked from PCLKB. You can set the PCLKB frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

### Pin Configuration

Comparator output can be enabled or disabled on each channel individually. The VCOUT pin is a logical OR of all comparator outputs.

The CMPINn pins are used as comparator inputs. The CMPREFn pins are used as comparator reference values.

## Usage Notes

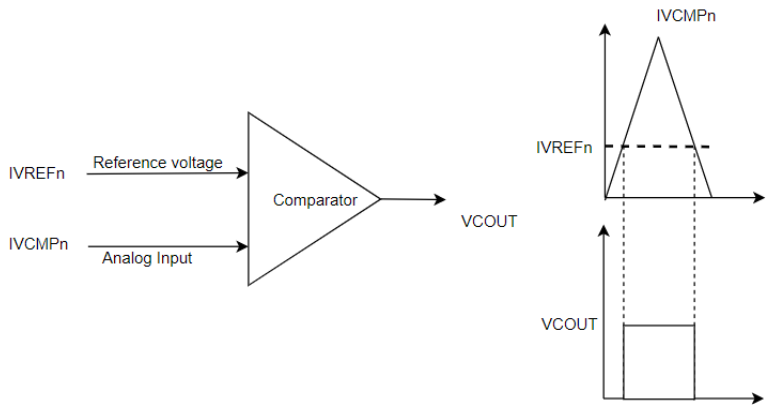



Figure 104: ACMPLP Standard Mode Operation

### Noise Filter

When the noise filter is enabled, the ACMPLP0/ACMPLP1 signal is sampled three times based on the sampling clock selected. The filter clock frequency is determined by PCLKB and the comparator_filter_t setting.

### Output Polarity

If output polarity is configured as "Inverted" then the VCOUT signal will be inverted and the R_ACMPLP_StatusGet() will return an inverted status.

### Window Mode

In window mode, the comparator indicates if the analog input voltage falls within the window (low and high reference voltage) or is outside the window.

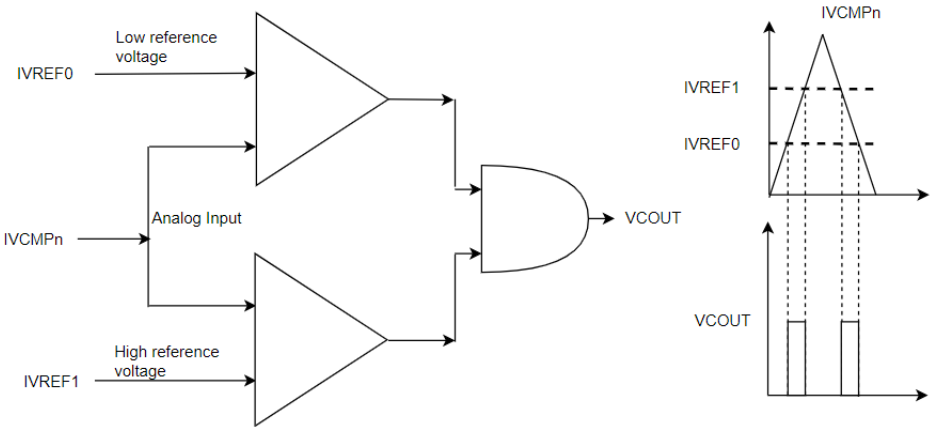



Figure 105: ACMPLP Window Mode Operation

### Limitations

- Once the analog comparator is configured, the program must wait for the stabilization time to elapse before using the comparator.
- Low speed is not supported by the ACMPLP driver.

# Examples

### Basic Example

The following is a basic example of minimal use of the ACMPLP. The comparator is configured to trigger a callback when the input rises above the internal reference voltage (VREF). A GPIO output acts as the comparator input and is externally connected to the CMPIN input of the ACMPLP.

```
/* Connect this control pin to the VCMP input of the comparator. This can be any GPIO
pin
 * that is not input only. */
#define ACMPLP_EXAMPLE_CONTROL_PIN (BSP_IO_PORT_04_PIN_08)
volatile uint32_t g_comparator_events = 0U;
/* This callback is called when a comparator event occurs. */
void acmplp_example_callback (comparator_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
    g_comparator_events++;
}
void acmplp_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /* Disable pin register write protection, if enabled */
```

```
R_BSP_PinAccessEnable();
/* Start with the VCMP pin low. This example assumes the comparator is configured to
trigger
 * when VCMP rises above VREF. */
   (void) R_BSP_PinWrite(ACMPLP_EXAMPLE_CONTROL_PIN, BSP_IO_LEVEL_LOW);
/* Initialize the ACMPLP module */
   err = R_ACMPLP_Open(&g_comparator_ctrl, &g_comparator_cfg);
/* Handle any errors. This function should be defined by the user. */
   handle_error(err);
/* Wait for the minimum stabilization wait time before enabling output. */
comparator_info_t info;
R_ACMPLP_InfoGet(&g_comparator_ctrl, &info);
R_BSP_SoftwareDelay(info.min_stabilization_wait_us, BSP_DELAY_UNITS_MICROSECONDS);
/* Enable the comparator output */
   (void) R_ACMPLP_OutputEnable(&g_comparator_ctrl);
/* Set VCMP low. */
   (void) R_BSP_PinWrite(ACMPLP_EXAMPLE_CONTROL_PIN, BSP_IO_LEVEL_HIGH);
while (0 == g_comparator_events)
   {
/* Wait for interrupt. */
   }
comparator_status_t status;
/* Check status of comparator, Status will be COMPARATOR_STATE_OUTPUT_HIGH */
   (void) R_ACMPLP_StatusGet(&g_comparator_ctrl, &status);
}
```

## Enumerations

| | |
|---|---|
| enum | acmplp_input_t |
| enum | acmplp_reference_t |

## Enumeration Type Documentation

### ◆ acmplp_input_t

| enum acmplp_input_t | |
|---|---|
| Enumerator | |
| ACMPLP_INPUT_AMPO | Only available on ra2a1. |
| ACMPLP_INPUT_CMPIN_1 | Only available on ra4m1. |

### ◆ acmplp_reference_t

| enum acmplp_reference_t | |
|---|---|
| Enumerator | |
| ACMPLP_REFERENCE_CMPREF_1 | Only available on ra4m1. |

## Function Documentation

### ◆ R_ACMPLP_Open()

fsp_err_t R_ACMPLP_Open ( comparator_ctrl_t *const *p_ctrl*, comparator_cfg_t const *const *p_cfg* )

Configures the comparator and starts operation. Callbacks and pin output are not active until outputEnable() is called. comparator_api_t::outputEnable() should be called after the output has stabilized. Implements comparator_api_t::open().

Comparator inputs must be configured in the application code prior to calling this function.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An input pointer is NULL |
| FSP_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. p_cfg->p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use callback function. |
| FSP_ERR_ALREADY_OPEN | The control block is already open or the hardware lock is taken. |

### ◆ R_ACMPLP_OutputEnable()

fsp_err_t R_ACMPLP_OutputEnable ( comparator_ctrl_t *const *p_ctrl*)

Enables the comparator output, which can be polled using comparator_api_t::statusGet(). Also enables pin output and interrupts as configured during comparator_api_t::open(). Implements comparator_api_t::outputEnable().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Comparator output is enabled. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_ACMPLP_InfoGet()

fsp_err_t R_ACMPLP_InfoGet ( comparator_ctrl_t *const *p_ctrl*, comparator_info_t *const *p_info* )

Provides the minimum stabilization wait time in microseconds. Implements comparator_api_t::infoGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in p_info. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_ACMPLP_StatusGet()

fsp_err_t R_ACMPLP_StatusGet ( comparator_ctrl_t *const *p_ctrl*, comparator_status_t *const *p_status* )

Provides the operating status of the comparator. Implements comparator_api_t::statusGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operating status of the comparator is provided in p_status. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_ACMPLP_Close()

fsp_err_t R_ACMPLP_Close ( comparator_ctrl_t * *p_ctrl*)

Stops the comparator. Implements comparator_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Instance control block closed successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_ACMPLP_VersionGet()

fsp_err_t R_ACMPLP_VersionGet ( fsp_version_t *const *p_version*)

Gets the API and code version. Implements comparator_api_t::versionGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version information available in p_version. |
| FSP_ERR_ASSERTION | The parameter p_version is NULL. |

## 4.2.3 Analog to Digital Converter (r_adc)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_ADC_Open (adc_ctrl_t *p_ctrl, adc_cfg_t const *const p_cfg) |
| fsp_err_t | R_ADC_ScanCfg (adc_ctrl_t *p_ctrl, void const *const p_extend) |
| fsp_err_t | R_ADC_InfoGet (adc_ctrl_t *p_ctrl, adc_info_t *p_adc_info) |
| fsp_err_t | R_ADC_ScanStart (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_ADC_ScanStop (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_ADC_StatusGet (adc_ctrl_t *p_ctrl, adc_status_t *p_status) |
| fsp_err_t | R_ADC_Read (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data) |

| | |
|---|---|
| fsp_err_t | R_ADC_Read32 (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data) |
| fsp_err_t | R_ADC_SampleStateCountSet (adc_ctrl_t *p_ctrl, adc_sample_state_t *p_sample) |
| fsp_err_t | R_ADC_Close (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_ADC_OffsetSet (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t offset) |
| fsp_err_t | R_ADC_Calibrate (adc_ctrl_t *const p_ctrl, void *const p_extend) |
| fsp_err_t | R_ADC_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Driver for the ADC12, ADC14, and ADC16 peripherals on RA MCUs. This module implements the ADC Interface.

# Overview

### Features

The ADC module supports the following features:

- 12, 14, or 16 bit maximum resolution depending on the MCU
- Configure scans to include:
  - Multiple analog channels
  - Temperature sensor channel
  - Voltage sensor channel
- Configurable scan start trigger:
  - Software scan triggers
  - Hardware scan triggers (timer expiration, for example)
  - External scan triggers from the ADTRGn port pins
- Configurable scan mode:
  - Single scan mode, where each trigger starts a single scan
  - Continuous scan mode, where all channels are scanned continuously
  - Group scan mode, where channels are grouped into group A and group B. The groups can be assigned different start triggers, and group A can be given priority over group B. When group A has priority over group B, a group A trigger suspends an ongoing group B scan.
- Supports adding and averaging converted samples
- Optional callback when scan completes
- Supports reading converted data
- Sample and hold support
- Double-trigger support

# Configuration

### Build Time Configurations for r_adc

The following build time configurations are defined in fsp_cfg/r_adc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Analog > ADC Driver on r_adc

This module can be added to the Stacks tab via New Stack > Driver > Analog > ADC Driver on r_adc:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General > Name | Name must be a valid C symbol | g_adc0 | Module name |
| General > Unit | Unit must be a non-negative integer | 0 | Specifies the ADC Unit to be used. |
| General > Resolution | MCU Specific Options | | Specifies the conversion resolution for this unit. |
| General > Alignment | MCU Specific Options | | Specifies the conversion result alignment. |
| General > Clear after read | • Off<br>• On | On | Specifies if the result register will be automatically cleared after the conversion result is read. |
| General > Mode | • Single Scan<br>• Continuous Scan<br>• Group Scan | Single Scan | Specifies the mode that this ADC unit is used in. |
| General > Double-trigger | • Disabled<br>• Enabled<br>• Enabled (extended mode) | Disabled | When enabled, the scan-end interrupt for Group A is only thrown on every second scan. Extended double-trigger mode (single-scan only) triggers on both ELC events, allowing (for example) a scan on two different timer compare match values.<br><br>In group mode Group B |

| | | | |
|---|---|---|---|
| | | | is unaffected. |
| Input > Sample and Hold > Sample and Hold Channels (Available only on selected MCUs) | • Channel 0<br>• Channel 1<br>• Channel 2 | | Specifies if this channel is included in the Sample and Hold Mask. |
| Input > Sample and Hold > Sample Hold States (Applies only to channels 0, 1, 2) | Must be a valid non-negative integer with configurable value 4 to 255 | 24 | Specifies the updated sample-and-hold count for the channel dedicated sample-and-hold circuit |
| Input > Channel Scan Mask (channel availability varies by MCU) | Refer to the RA Configuration tool for available options. | | In Normal mode of operation, this bitmask field specifies the channels that are enabled in that ADC unit. In group mode, this field specifies which channels belong to group A. |
| Input > Group B Scan Mask (channel availability varies by MCU) | Refer to the RA Configuration tool for available options. | | In group mode, this field specifies which channels belong to group B. |
| Input > Add/Average Count | • Disabled<br>• Add two samples<br>• Add three samples<br>• Add four samples<br>• Add sixteen samples<br>• Average two samples<br>• Average four samples | Disabled | Specifies if addition or averaging needs to be done for any of the channels in this unit. |
| Input > Reference Voltage control | MCU Specific Options | | Specify VREFH/VREFADC output voltage control. |
| Input > Addition/Averaging Mask (channel availability varies by MCU and unit) | Refer to the RA Configuration tool for available options. | | Select channels to include in the Addition/Averaging Mask |
| Interrupts > Normal/Group A Trigger | MCU Specific Options | | Specifies the trigger type to be used for this unit. |

| | | | |
|---|---|---|---|
| Interrupts > Group B Trigger | MCU Specific Options | | Specifies the trigger for Group B scanning in group scanning mode. This event is also used to trigger Group A in extended double-trigger mode. |
| Interrupts > Group Priority (Valid only in Group Scan Mode) | • Group A cannot interrupt Group B<br>• Group A can interrupt Group B; Group B scan restarts at next trigger<br>• Group A can interrupt Group B; Group B scan restarts immediately<br>• Group A can interrupt Group B; Group B scan restarts immediately and scans continuously | 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. |
| Interrupts > Callback | Name must be a valid C symbol | NULL | A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the ADC scan completes. |
| Interrupts > Scan End Interrupt Priority | MCU Specific Options | | Select scan end interrupt priority. |
| Interrupts > Scan End Group B Interrupt Priority | MCU Specific Options | | Select group B scan end interrupt priority. |

### Clock Configuration

The ADC clock is PCLKC if the MCU has PCLKC, or PCLKD otherwise.

The ADC clock must be at least 1 MHz when the ADC is used. Many MCUs also have PCLK ratio restrictions when the ADC is used. For details on PCLK ratio restrictions, reference the footnotes in the second table of the Clock Generation Circuit chapter of the MCU User's Manual (for example, Table 9.2 "Specifications of the clock generation circuit for the internal clocks" in the RA6M3 manual R01UH0886EJ0100).

### Pin Configuration

The ANxxx pins are analog input channels that can be used with the ADC.

ADTRG0 and ADTRG1 can be used to start scans with an external trigger for unit 0 and 1 respectively. When external triggers are used, ADC scans begin on the falling edge of the ADTRG pin.

# Usage Notes

### Sample Hold

Enabling the sample and hold functionality reduces the maximum scan frequency because the sample and hold time is added to each scan. Refer to the hardware manual for details on the sample and hold time.

### ADC Operational Modes

The driver supports three operation modes: single-scan, continuous-scan, and group-scan modes. In each mode, analog channels are converted in ascending order of channel number, followed by scans of the temperature sensor and voltage sensor if they are included in the mask of channels to scan.

### Single-scan Mode

In single scan mode, one or more specified channels are scanned once per trigger.

### Continuous-scan Mode

In continuous scan mode, a single trigger is required to start the scan. Scans continue until R_ADC_ScanStop() is called.

### Group-scan Mode

Group-scan mode allows the application to allocate channels to one of two groups (A and B). Conversion begins when the specified ELC start trigger for that group is received.

With the priority configuration parameter, you can optionally give group A priority over group B. If group A has priority over group B, a group B scan is interrupted when a group A scan trigger occurs. The following options exist for group B when group A has priority:

- To restart the interrupted group B scan after the group A scan completes.
- To wait for another group B trigger and forget the interrupted scan.
- To continuously scan group B and suspend scanning group B only when a group A trigger is received.

  *Note*

  *If this option is selected, group B scanning begins immediately after R_ADC_ScanCfg(). Group A scan triggers must be enabled by R_ADC_ScanStart() and can be disabled by R_ADC_ScanStop(). Group B scans can only be disabled by reconfiguring the group A priority to a different mode.*

### Double-triggering

When double-triggering is enabled a single channel is selected to be scanned twice before an interrupt is thrown. The first scan result when using double-triggering is always saved to the selected channel's data register. The second result is saved to the data duplexing register (ADC_CHANNEL_DUPLEX).

Double-triggering uses Group A; only one channel can be selected when enabled. No other scanning is possible on Group A while double-trigger mode is selected. In addition, any special ADC channels (such as temperature sensors or voltage references) are not valid double-trigger channels.

When extended double-triggering is enabled both ADC input events are routed to Group A. The interrupt is still thrown after every two scans regardless of the triggering event(s). While the first and second scan are saved to the selected ADC data register and the ADC duplexing register as before, scans associated with event A and B are additionally copied into duplexing register A and B, respectively (ADC_CHANNEL_DUPLEX_A and ADC_CHANNEL_DUPLEX_B).

### When Interrupts Are Not Enabled

If interrupts are not enabled, the R_ADC_StatusGet API can be used to poll the ADC to determine when the scan has completed. The read API function is used to access the converted ADC result. This applies to both normal scans and calibration scans for MCUs that support calibration.

### Sample-State Count Setting

The application program can modify the setting of the sample-state count for analog channels by calling the R_ADC_SampleStateCountSet() API function. The application program only needs to modify the sample-state count settings from their default values to increase the sampling time. This can be either because the impedance of the input signal is too high to secure sufficient sampling time under the default setting or if the ADCLK is too slow. To modify the sample-state count for a given channel, set the channel number and the number of states when calling the R_ADC_SampleStateCountSet() API function. Valid sample state counts are 7-255.

*Note*

*Although the hardware supports a minimum number of sample states of 5, some MCUs require 7 states, so the minimum is set to 7. At the lowest supported ADC conversion clock rate (1 MHz), these extra states will lead to, at worst case, a 2 microsecond increase in conversion time. At 60 MHz the extra states will add 33.4 ns to the conversion time.*

If the sample state count needs to be changed for multiple channels, the application program must call the R_ADC_SampleStateCountSet() API function repeatedly, with appropriately modified arguments for each channel.

If the ADCLK frequency changes, the sample states may need to be updated.

### Sample States for Temperature Sensor and Internal Voltage Reference

Sample states for the temperature sensor and the internal reference voltage are calculated during R_ADC_ScanCfg() based on the ADCLK frequency at the time. The sample states for the temperature sensor and internal voltage reference cannot be updated with R_ADC_SampleStateCountSet(). If the ADCLK frequency changes, call R_ADC_ScanCfg() before using the temperature sensor or internal reference voltage again to ensure the sampling time for the temperature sensor and internal voltage reference is optimal.

### Selecting Reference Voltage

The ADC16 can select VREFH0 or VREFADC as the high-potential reference voltage on selected MCU's. When using VREFADC stabilization time of 1500us is required after call for R_ADC_Open().

### 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 in the application program using the following formula, T = (Vs - V1)/slope + T1, where:

- T: Measured temperature (degrees C)
- Vs: Voltage output by the temperature sensor at the time of temperature measurement (Volts)
- T1: Temperature experimentally measured at one point (degrees 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 (degrees C)
- V2: Voltage output by the temperature sensor at the time of measurement of T2 (Volts)
- Slope: Temperature gradient of the temperature sensor (V/degrees C); slope = (V2 - V1)/(T2 - T1)

*Note*

*The slope value can be obtained from the hardware manual for each device in the Electrical Characteristics Chapter - TSN Characteristics Table, Temperature slope entry.*

### Usage Notes for ADC16

### Calibration

Calibration is required to use the ADC16 peripheral. When using this driver on an MCU that has ADC16, call R_ADC_Calibrate() after open, and prior to any other function.

### Range of ADC16 Results

The range of the ADC16 is from 0 (lowest) to 0x7FFF (highest) when used in single-ended mode. This driver only supports single ended mode.

## Examples

### Basic Example

This is a basic example of minimal use of the ADC in an application.

```
/* A channel configuration is generated by the RA Configuration editor based on the
options selected. If additional
 * configurations are desired additional adc_channel_cfg_t elements can be defined
and passed to R_ADC_ScanCfg. */
const adc_channel_cfg_t g_adc0_channel_cfg =
{
    .scan_mask          = ADC_MASK_CHANNEL_0 | ADC_MASK_CHANNEL_1,
    .scan_mask_group_b = 0,
    .priority_group_a   = (adc_group_a_t) 0,
    .add_mask          = 0,
    .sample_hold_mask   = 0,
    .sample_hold_states = 0,
```

```
};
void adc_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Enable channels. */
    err = R_ADC_ScanCfg(&g_adc0_ctrl, &g_adc0_channel_cfg);
    handle_error(err);
 /* In software trigger mode, start a scan by calling R_ADC_ScanStart(). In other
modes, enable external
  * triggers by calling R_ADC_ScanStart(). */
    (void) R_ADC_ScanStart(&g_adc0_ctrl);
 /* Wait for conversion to complete. */
 adc_status_t status;
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
    {
       (void) R_ADC_StatusGet(&g_adc0_ctrl, &status);
    }
 /* Read converted data. */
    uint16_t channel1_conversion_result;
    err = R_ADC_Read(&g_adc0_ctrl, ADC_CHANNEL_1, &channel1_conversion_result);
    handle_error(err);
}
```

### Temperature Sensor Example

This example shows how to calculate the MCU temperature using the ADC and the temperature sensor.

```
#define ADC_EXAMPLE_CALIBRATION_DATA_RA6M1 (0x7D5)
#define ADC_EXAMPLE_VCC_MICROVOLT (3300000)
```

```
#define ADC_EXAMPLE_TEMPERATURE_RESOLUTION (12U)
#define ADC_EXAMPLE_REFERENCE_CALIBRATION_TEMPERATURE (127)
void adc_temperature_example (void)
{
 /* The following example calculates the temperature on an RA6M1 device using the
data provided in the section
  * 44.3.1 "Preparation for Using the Temperature Sensor" of the RA6M1 manual
R01UH0884EJ0100. */
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Enable temperature sensor. */
    err = R_ADC_ScanCfg(&g_adc0_ctrl, &g_adc0_channel_cfg);
    handle_error(err);
 /* In software trigger mode, start a scan by calling R_ADC_ScanStart(). In other
modes, enable external
  * triggers by calling R_ADC_ScanStart(). */
    (void) R_ADC_ScanStart(&g_adc0_ctrl);
 /* Wait for conversion to complete. */
 adc_status_t status;
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
    {
       (void) R_ADC_StatusGet(&g_adc0_ctrl, &status);
    }
 /* Read converted data. */
    uint16_t temperature_conversion_result;
    err = R_ADC_Read(&g_adc0_ctrl, ADC_CHANNEL_TEMPERATURE,
&temperature_conversion_result);
    handle_error(err);
#if BSP_FEATURE_ADC_TSN_CALIBRATION_AVAILABLE
 /* Get Calibration data from the MCU. */
```

```
    int32_t    reference_calibration_data;
 adc_info_t adc_info;
    (void) R_ADC_InfoGet(&g_adc0_ctrl, &adc_info);
    reference_calibration_data = (int32_t) adc_info.calibration_data;
#else
 /* If the MCU does not provide calibration data, use the value in the hardware
manual or determine it
  * experimentally. */
    int32_t reference_calibration_data = ADC_EXAMPLE_CALIBRATION_DATA_RA6M1;
#endif
 /* NOTE: The slope of the temperature sensor varies from sensor to sensor. Renesas
recommends calculating
  * the slope of the temperature sensor experimentally.
  *
  * This example uses the typical slope provided in Table 52.38 "TSN characteristics"
in the RA6M1 manual
  * R01UM0011EU0050. */
    int32_t slope_uv_per_c = BSP_FEATURE_ADC_TSN_SLOPE;
 /* Formula for calculating temperature copied from section 44.3.1 "Preparation for
Using the Temperature Sensor"
  * of the RA6M1 manual R01UH0884EJ0100:
  *
  * In this MCU, the TSCDR register stores the temperature value (CAL127) of the
temperature sensor measured
  * under the condition Ta = Tj = 127 C and AVCC0 = 3.3 V. By using this value as the
sample measurement result
  * at the first point, preparation before using the temperature sensor can be
omitted.
  *
  * If V1 is calculated from CAL127,
  * V1 = 3.3 * CAL127 / 4096 [V]
  *
  * Using this, the measured temperature can be calculated according to the following
formula.
```

```
  *
  * T = (Vs - V1) / Slope + 127 [C]
  * T: Measured temperature (C)
  * Vs: Voltage output by the temperature sensor when the temperature is measured (V)
  * V1: Voltage output by the temperature sensor when Ta = Tj = 127 C and AVCC0 = 3.3
V (V)
  * Slope: Temperature slope given in Table 52.38 / 1000 (V/C)
  */
    int32_t v1_uv = (ADC_EXAMPLE_VCC_MICROVOLT >> ADC_EXAMPLE_TEMPERATURE_RESOLUTION)
*
                    reference_calibration_data;
    int32_t vs_uv = (ADC_EXAMPLE_VCC_MICROVOLT >> ADC_EXAMPLE_TEMPERATURE_RESOLUTION)
*
                    temperature_conversion_result;
    int32_t temperature_c = (vs_uv - v1_uv) / slope_uv_per_c +
ADC_EXAMPLE_REFERENCE_CALIBRATION_TEMPERATURE;
 /* Expect room temperature, break if temperature is outside the range of 20 C to 25
C. */
 if ((temperature_c < 20) || (temperature_c > 25))
    {
       __BKPT(0);
    }
}
```

### Double-Trigger Example

This example demonstrates reading data from a double-trigger scan. A flag is used to wait for a callback event. Two scans must occur before the callback is called. These results are read via R_ADC_Read using the selected channel enum value as well as ADC_CHANNEL_DUPLEX.

```
volatile bool scan_complete_flag = false;
void adc_callback (adc_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
    scan_complete_flag = true;
}
```

```
void adc_double_trigger_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the module. */
    err = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Enable double-trigger channel. */
    err = R_ADC_ScanCfg(&g_adc0_ctrl, &g_adc0_channel_cfg);
    handle_error(err);
 /* Enable scan triggering from ELC events. */
    (void) R_ADC_ScanStart(&g_adc0_ctrl);
 /* Wait for conversion to complete. Two scans must be triggered before a callback
occurs. */
    scan_complete_flag = false;
 while (!scan_complete_flag)
    {
 /* Wait for callback to set flag. */
    }
 /* Read converted data from both scans. */
    uint16_t channel1_conversion_result_0;
    uint16_t channel1_conversion_result_1;
    err = R_ADC_Read(&g_adc0_ctrl, ADC_CHANNEL_1, &channel1_conversion_result_0);
    handle_error(err);
    err = R_ADC_Read(&g_adc0_ctrl, ADC_CHANNEL_DUPLEX,
&channel1_conversion_result_1);
    handle_error(err);
}
```

## Data Structures

| | |
|---|---|
| struct | adc_sample_state_t |
| struct | adc_extended_cfg_t |
| struct | adc_channel_cfg_t |

| | |
|---|---|
| struct | adc_instance_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | adc_mask_t |
| enum | adc_add_t |
| enum | adc_clear_t |
| enum | adc_vref_control_t |
| enum | adc_sample_state_reg_t |
| enum | adc_group_a_t |
| enum | adc_double_trigger_t |

## Data Structure Documentation

### ◆ adc_sample_state_t

| struct adc_sample_state_t | | |
|---|---|---|
| ADC sample state configuration | | |
| Data Fields | | |
| 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. |

### ◆ adc_extended_cfg_t

| struct adc_extended_cfg_t | | |
|---|---|---|
| Extended configuration structure for ADC. | | |
| Data Fields | | |
| adc_add_t | add_average_count | Add or average samples. |
| adc_clear_t | clearing | Clear after read. |
| adc_trigger_t | trigger_group_b | Group B trigger source; valid only for group mode. |
| adc_double_trigger_t | double_trigger_mode | Double-trigger mode setting. |
| adc_vref_control_t | adc_vref_control | VREFADC output voltage control. |

### ◆ adc_channel_cfg_t

| struct adc_channel_cfg_t |
|---|

| ADC channel(s) configuration | | |
|---|---|---|
| Data Fields | | |
| uint32_t | scan_mask | Channels/bits: bit 0 is ch0; bit 15 is ch15. |
| uint32_t | scan_mask_group_b | Valid for group modes. |
| uint32_t | add_mask | Valid if add enabled in Open(). |
| adc_group_a_t | priority_group_a | Valid for group modes. |
| uint8_t | sample_hold_mask | Channels/bits 0-2. |
| uint8_t | sample_hold_states | Number of states to be used for sample and hold. Affects channels 0-2. |

### ◆ adc_instance_ctrl_t

| struct adc_instance_ctrl_t |
|---|
| ADC instance control block. DO NOT INITIALIZE. Initialized in adc_api_t::open(). |

## Enumeration Type Documentation

### ◆ adc_mask_t

| enum adc_mask_t | |
|---|---|
| For ADC Scan configuration adc_channel_cfg_t::scan_mask, adc_channel_cfg_t::scan_mask_group_b , adc_channel_cfg_t::add_mask and adc_channel_cfg_t::sample_hold_mask. Use bitwise OR to combine these masks for desired channels and sensors. | |
| Enumerator | |
| ADC_MASK_OFF | No channels selected. |
| ADC_MASK_CHANNEL_0 | Channel 0 mask. |
| ADC_MASK_CHANNEL_1 | Channel 1 mask. |
| ADC_MASK_CHANNEL_2 | Channel 2 mask. |
| ADC_MASK_CHANNEL_3 | Channel 3 mask. |
| ADC_MASK_CHANNEL_4 | Channel 4 mask. |
| ADC_MASK_CHANNEL_5 | Channel 5 mask. |
| ADC_MASK_CHANNEL_6 | Channel 6 mask. |
| ADC_MASK_CHANNEL_7 | Channel 7 mask. |

| | |
|---|---|
| ADC_MASK_CHANNEL_8 | Channel 8 mask. |
| ADC_MASK_CHANNEL_9 | Channel 9 mask. |
| ADC_MASK_CHANNEL_10 | Channel 10 mask. |
| ADC_MASK_CHANNEL_11 | Channel 11 mask. |
| ADC_MASK_CHANNEL_12 | Channel 12 mask. |
| ADC_MASK_CHANNEL_13 | Channel 13 mask. |
| ADC_MASK_CHANNEL_14 | Channel 14 mask. |
| ADC_MASK_CHANNEL_15 | Channel 15 mask. |
| ADC_MASK_CHANNEL_16 | Channel 16 mask. |
| ADC_MASK_CHANNEL_17 | Channel 17 mask. |
| ADC_MASK_CHANNEL_18 | Channel 18 mask. |
| ADC_MASK_CHANNEL_19 | Channel 19 mask. |
| ADC_MASK_CHANNEL_20 | Channel 20 mask. |
| ADC_MASK_CHANNEL_21 | Channel 21 mask. |
| ADC_MASK_CHANNEL_22 | Channel 22 mask. |
| ADC_MASK_CHANNEL_23 | Channel 23 mask. |
| ADC_MASK_CHANNEL_24 | Channel 24 mask. |
| ADC_MASK_CHANNEL_25 | Channel 25 mask. |
| ADC_MASK_CHANNEL_26 | Channel 26 mask. |
| ADC_MASK_CHANNEL_27 | Channel 27 mask. |
| ADC_MASK_TEMPERATURE | Temperature sensor channel mask. |
| ADC_MASK_VOLT | Voltage reference channel mask. |
| ADC_MASK_SENSORS | All sensor channel mask. |

### ◆ adc_add_t

| enum adc_add_t | |
|---|---|
| ADC data sample addition and averaging options | |
| Enumerator | |
| 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_EIGHT | Average eight samples. |
| ADC_ADD_AVERAGE_SIXTEEN | Add sixteen samples. |

### ◆ adc_clear_t

| enum adc_clear_t | |
|---|---|
| ADC clear after read definitions | |
| Enumerator | |
| ADC_CLEAR_AFTER_READ_OFF | Clear after read off. |
| ADC_CLEAR_AFTER_READ_ON | Clear after read on. |

### ◆ adc_vref_control_t

| enum adc_vref_control_t | |
|---|---|
| ADC VREFAMPCNT config options Reference Table 32.12 "VREFADC output voltage control list" in the RA2A1 manual R01UH0888EJ0100. | |
| Enumerator | |
| ADC_VREF_CONTROL_VREFH | VREFAMPCNT reset value. VREFADC Output voltage is Hi-Z. |
| ADC_VREF_CONTROL_1_5V_OUTPUT | BGR turn ON. VREFADC Output voltage is 1.5 V. |
| ADC_VREF_CONTROL_2_0V_OUTPUT | BGR turn ON. VREFADC Output voltage is 2.0 V. |
| ADC_VREF_CONTROL_2_5V_OUTPUT | BGR turn ON. VREFADC Output voltage is 2.5 V. |

### ◆ adc_sample_state_reg_t

| enum adc_sample_state_reg_t | |
|---|---|
| ADC sample state registers | |
| Enumerator | |
| 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. |
| 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_31 | Sample state register channel 16 to 31. |

### ◆ adc_group_a_t

| enum adc_group_a_t | |
|---|---|
| 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. | |
| Enumerator | |
| 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. |

### ◆ adc_double_trigger_t

| enum adc_double_trigger_t | |
|---|---|
| ADC double-trigger mode definitions | |
| Enumerator | |
| ADC_DOUBLE_TRIGGER_DISABLED | Double-triggering disabled. |
| ADC_DOUBLE_TRIGGER_ENABLED | Double-triggering enabled. |
| ADC_DOUBLE_TRIGGER_ENABLED_EXTENDED | Double-triggering enabled on both ADC ELC events. |

## Function Documentation

### ◆ R_ADC_Open()

fsp_err_t R_ADC_Open ( adc_ctrl_t * *p_ctrl*, adc_cfg_t const *const *p_cfg* )

Sets the operational mode, trigger sources, interrupt priority, and configurations for the peripheral as a whole. If interrupt is enabled, the function registers a callback function pointer for notifying the user whenever a scan has completed.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is ready for use. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_ALREADY_OPEN | The instance control structure has already been opened. |
| FSP_ERR_IRQ_BSP_DISABLED | A callback is provided, but the interrupt is not enabled. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The requested unit does not exist on this MCU. |
| FSP_ERR_INVALID_HW_CONDITION | The ADC clock must be at least 1 MHz |

### ◆ R_ADC_ScanCfg()

fsp_err_t R_ADC_ScanCfg ( adc_ctrl_t * *p_ctrl*, void const *const *p_extend* )

Configures the ADC scan parameters. Channel specific settings are set in this function. Pass a pointer to adc_channel_cfg_t to p_extend.

***Note***

*This starts group B scans if adc_channel_cfg_t::priority_group_a is set to ADC_GROUP_A_GROUP_B_CONTINUOUS_SCAN.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel specific settings applied. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_InfoGet()

fsp_err_t R_ADC_InfoGet ( adc_ctrl_t * *p_ctrl*, adc_info_t * *p_adc_info* )

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.

Also provides the temperature sensor slope and the calibration data for the sensor if available on this MCU. Otherwise, invalid calibration data of 0xFFFFFFFF will be returned.

***Note***

*In group mode, information is returned for group A only. Calculating information for group B is not currently supported.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in p_adc_info. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_ScanStart()

fsp_err_t R_ADC_ScanStart ( adc_ctrl_t * *p_ctrl*)

Starts a software scan or enables the hardware trigger for a scan depending on how the triggers were configured in the R_ADC_Open call. If the unit was configured for ELC or external hardware triggering, then this function allows the trigger signal 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.

**Precondition**

Call R_ADC_ScanCfg after R_ADC_Open before starting a scan.
On MCUs that support calibration, call R_ADC_Calibrate and wait for calibration to complete before starting a scan.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Scan started (software trigger) or hardware triggers enabled. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |
| FSP_ERR_IN_USE | Another scan is still in progress (software trigger). |

### ◆ R_ADC_ScanStop()

fsp_err_t R_ADC_ScanStop ( adc_ctrl_t * *p_ctrl*)

Stops the software scan or disables the unit from being triggered by the hardware trigger (ELC or external) based on what type of trigger the unit was configured for in the R_ADC_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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Scan stopped (software trigger) or hardware triggers disabled. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_StatusGet()

fsp_err_t R_ADC_StatusGet ( adc_ctrl_t * *p_ctrl*, adc_status_t * *p_status* )

Provides the status of any scan process that was started, including scans started by ELC or external triggers and calibration scans on MCUs that support calibration.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module status stored in the provided pointer p_status |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_Read()

fsp_err_t R_ADC_Read ( adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint16_t *const *p_data* )

Reads conversion results from a single channel or sensor.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read into provided p_data. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_Read32()

fsp_err_t R_ADC_Read32 ( adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint32_t *const *p_data* )

Reads conversion results from a single channel or sensor register into a 32-bit result.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read into provided p_data. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_SampleStateCountSet()

fsp_err_t R_ADC_SampleStateCountSet ( adc_ctrl_t * *p_ctrl*, adc_sample_state_t * *p_sample* )

Sets 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.

***Note***

*The sample states for the temperature and voltage sensor are set in R_ADC_ScanCfg.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Sample state count updated. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_Close()

fsp_err_t R_ADC_Close ( adc_ctrl_t * *p_ctrl*)

This function ends any scan in progress, disables interrupts, and removes power to the A/D peripheral.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module closed. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_OffsetSet()

fsp_err_t R_ADC_OffsetSet ( adc_ctrl_t *const *p_ctrl*, adc_channel_t const *reg_id*, int32_t *offset* )

adc_api_t::offsetSet is not supported on the ADC.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_ADC_Calibrate()

fsp_err_t R_ADC_Calibrate ( adc_ctrl_t *const *p_ctrl*, void *const *p_extend* )

Initiates calibration of the ADC on MCUs that require calibration. This function must be called before starting a scan on MCUs that require calibration.

Calibration is complete when the callback is called with ADC_EVENT_CALIBRATION_COMPLETE or when R_ADC_StatusGet returns ADC_STATUS_IDLE. Reference Figure 32.35 "Software flow and operation example of calibration operation." in the RA2A1 manual R01UH0888EJ0100.

ADC calibration time: 12 PCLKB + 774,930 ADCLK. (Reference Table 32.16 "Required calibration time (shown as the number of ADCLK and PCLKB cycles)" in the RA2A1 manual R01UH0888EJ0100. The lowest supported ADCLK is 1MHz.

Calibration will take a minimum of 24 milliseconds at 32 MHz PCLKB and ADCLK. This wait could take up to 780 milliseconds for a 1 MHz PCLKD (ADCLK).

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the instance control structure |
| [in] | p_extend | Unused argument. Pass NULL. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calibration successfully initiated. |
| FSP_ERR_INVALID_HW_CONDITION | A scan is in progress or hardware triggers are enabled. |
| FSP_ERR_UNSUPPORTED | Calibration not supported on this MCU. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_VersionGet()

| fsp_err_t R_ADC_VersionGet ( fsp_version_t *const *p_version*) |
|---|
| Retrieve the API version number. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version stored in the provided p_version. |
| FSP_ERR_ASSERTION | An input argument is invalid. |

## 4.2.4 Asynchronous General Purpose Timer (r_agt)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_AGT_Close (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_AGT_PeriodSet (timer_ctrl_t *const p_ctrl, uint32_t const period_counts) |
| fsp_err_t | R_AGT_DutyCycleSet (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin) |
| fsp_err_t | R_AGT_Reset (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_AGT_Start (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_AGT_Enable (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_AGT_Disable (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_AGT_InfoGet (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info) |
| fsp_err_t | R_AGT_StatusGet (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status) |
| fsp_err_t | R_AGT_Stop (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_AGT_Open (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg) |
| fsp_err_t | R_AGT_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Driver for the AGT peripheral on RA MCUs. This module implements the Timer Interface.

# Overview

### Features

The AGT module has the following features:

- Supports periodic mode, one-shot mode, and PWM mode.
- Signal can be output to a pin.
- Configurable period (counts per timer cycle).
- Configurable duty cycle in PWM mode.
- Configurable clock source, including PCLKB, LOCO, SUBCLK, and external sources input to AGTIO.
- Supports runtime reconfiguration of period.
- Supports runtime reconfiguration of duty cycle in PWM mode.
- Supports counting based on an external clock input to AGTIO.
- Supports debounce filter on AGTIO pins.
- Supports measuring pulse width or pulse period.
- APIs are provided to start, stop, and reset the counter.
- APIs are provided to get the current period, source clock frequency, and count direction.
- APIs are provided to get the current timer status and counter value.

### Selecting a Timer

RA MCUs have two timer peripherals: the General PWM Timer (GPT) and the Asynchronous General Purpose Timer (AGT). When selecting between them, consider these factors:

| | GPT | AGT |
|---|---|---|
| Low Power Modes | The GPT can operate in sleep mode. | The AGT can operate in all low power modes (when count source is LOCO or subclock). |
| Available Channels | The number of GPT channels is device specific. All currently supported MCUs have at least 7 GPT channels. | All MCUs have 2 AGT channels. |
| Timer Resolution | All MCUs have at least one 32-bit GPT timer. | The AGT timers are 16-bit timers. |
| Clock Source | The GPT runs off PCLKD with a configurable divider up to 1024. It can also be configured to count ELC events or external pulses. | The AGT runs off PCLKB, LOCO, or subclock with a configurable divider up to 8 for PCLKB or up to 128 for LOCO or subclock. |

# Configuration

### Build Time Configurations for r_agt

The following build time configurations are defined in fsp_cfg/r_agt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Pin Output Support | • Disabled<br>• Enabled | Disabled | If selected code for outputting a waveform to a pin is included in the build. |
| Pin Input Support | • Disabled<br>• Enabled | Disabled | Enable input support to use pulse width measurement mode, pulse period measurement mode, or input from P402, P402, or AGTIO. |

**Configurations for Driver > Timers > Timer Driver on r_agt**

This module can be added to the Stacks tab via New Stack > Driver > Timers > Timer Driver on r_agt:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General > Name | Name must be a valid C symbol | g_timer0 | Module name. |
| General > Channel | Available AGT Channels are 0 and 1 | 0 | Physical hardware channel. |
| General > Mode | • Periodic<br>• One-Shot<br>• PWM | Periodic | Mode selection. Note: One-shot mode is implemented in software. ISR's must be enabled for one shot even if callback is unused. |
| General > Period | Value must be non-negative | 0x10000 | Specify the timer period based on the selected unit.<br><br>When the unit is set to 'Raw Counts', setting the period to 0x10000 results in the maximum period at the lowest divisor (fastest timer tick). Set the period to 0x10000 for a free running timer, pulse |

| | | | |
|---|---|---|---|
| | | | width measurement or pulse period measurement. Setting the period higher will automatically select a higher divider; the period can be set up to 0x80000 when counting from PCLKB or 0x800000 when counting from LOCO/subclock, which will use a divider of 8 or 128 respectively with the maximum period. |
| General > Period Unit | • Raw Counts<br>• Nanoseconds<br>• Microseconds<br>• Milliseconds<br>• Seconds<br>• Hertz<br>• Kilohertz | Raw Counts | Unit of the period specified above |
| General > Count Source | • PCLKB<br>• LOCO<br>• SUBCLOCK<br>• AGT0 Underflow<br>• P402 Input<br>• P403 Input<br>• AGTIO Input | PCLKB | AGT counter clock source. NOTE: The divisor is calculated automatically based on the selected period. See agt_count_source_t documentation for details. |
| Output > Duty Cycle Percent (only applicable in PWM mode) | Value must be between 0 and 100 | 50 | Specify the timer duty cycle percent. Only used in PWM mode. |
| Output > AGTOA Output | • Disabled<br>• Start Level Low<br>• Start Level High | Disabled | Configure AGTOA output. |
| Output > AGTOB Output | • Disabled<br>• Start Level Low<br>• Start Level High | Disabled | Configure AGTOB output. |
| Output > AGTO Output | • Disabled<br>• Start Level Low<br>• Start Level High | Disabled | Configure AGTO output. |
| Input > Measurement Mode | • Measure Disabled<br>• Measure Low Level Pulse Width<br>• Measure High Level Pulse | Measure Disabled | Select if the AGT should be used to measure pulse width or pulse period. In high level pulse width measurement mode, the AGT counts when |

| | | | |
|---|---|---|---|
| | Width<br>• Measure Pulse Period | | AGTIO is high and starts counting immediately in the middle of a pulse if AGTIO is high when R_AGT_Start() is called. In low level pulse width measurement mode, the AGT counts when AGTIO is low and could start counting in the middle of a pulse if AGTIO is low when R_AGT_Start() is called. |
| Input > AGTIO Filter | • No Filter<br>• Filter sampled at PCLKB<br>• Filter sampled at PCLKB / 8<br>• Filter sampled at PCLKB / 32 | No Filter | Input filter, applies AGTIO in pulse period measurement, pulse width measurement, or event counter mode. The filter requires the signal to be at the same level for 3 successive reads at the specified filter frequency. |
| Input > Enable Pin | • Enable Pin Not Used<br>• Enable Pin Active Low<br>• Enable Pin Active High | Enable Pin Not Used | Select active edge for the AGTEE pin if used. Only applies if the count source is P402, P403 or AGTIO. |
| Input > Trigger Edge | • Trigger Edge Rising<br>• Trigger Edge Falling<br>• Trigger Edge Both | Trigger Edge Rising | Select the trigger edge. Applies if measurement mode is pulse period, or if the count source is P402, P403, or AGTIO. Do not select Trigger Edge Both with pulse period measurement. |
| Interrupts > Callback | Name must be a valid C symbol | NULL | A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the timer period elapses. |
| Interrupts > Underflow Interrupt Priority | MCU Specific Options | | Timer interrupt priority. |

**Clock Configuration**

The AGT clock is based on the PCLKB, LOCO, or Subclock frequency. You can set the clock frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

### Pin Configuration

This module can use the AGTOA and AGTOB pins as output pins for periodic, one-shot, or PWM signals.

For input capture, the input signal must be applied to the AGTIOn pin.

For event counting, the AGTEEn enable pin is optional.

### Timer Period

The RA Configuration editor will automatically calculate the period count value and source clock divider based on the selected period time, units, and clock speed.

When the selected unit is "Raw counts", the maximum allowed period setting varies depending on the selected clock source:

| Clock source | Maximum period (counts) |
|---|---|
| LOCO/Subclock | 0x800000 |
| PCLKB | 0x80000 |
| All other sources | 0x10000 |

*Note*

*Though the AGT is a 16-bit timer, because the period interrupt occurs when the counter underflows, setting the period register to 0 results in an effective period of 1 count. For this reason all user-provided raw count values reflect the actual number of period counts (not the raw register values).*

# Usage Notes

### Starting and Stopping the AGT

After starting or stopping the timer, AGT registers cannot be accessed until the AGT state is updated after 3 AGTCLK cycles. If another AGT function is called before the 3 AGTCLK period elapses, the function spins waiting for the AGT state to update. The required wait time after starting or stopping the timer can be determined using the frequency of AGTCLK, which is derived from timer_cfg_t::source_div and agt_extended_cfg_t::count_source.

The application is responsible for ensuring required clocks are started and stable before accessing MCU peripheral registers.

Warning

The subclock can take seconds to stabilize. The RA startup code does not wait for subclock stabilization unless the subclock is the main clock source. When running AGT or RTC off the subclock, the application must ensure the subclock is stable before starting operation.

### Low Power Modes

The AGT1 (channel 1 only) can be used to enter snooze mode or to wake the MCU from snooze, software standby, or deep software standby modes when a counter underflow occurs. The compare

match A and B events can also be used to wake from software standby or snooze modes.

### One-Shot Mode

The AGT timer does not support one-shot mode natively. One-shot mode is achieved by stopping the timer in the interrupt service routine before the callback is called. If the interrupt is not serviced before the timer period expires again, the timer generates more than one event. The callback is only called once in this case, but multiple events may be generated if the timer is linked to the Data Transfer Controller (r_dtc).

### One-Shot Mode Output

The output waveform in one-shot mode is one AGT clock cycle less than the configured period. The configured period must be at least 2 counts to generate an output pulse.

Examples of one-shot signals that can be generated by this module are shown below:

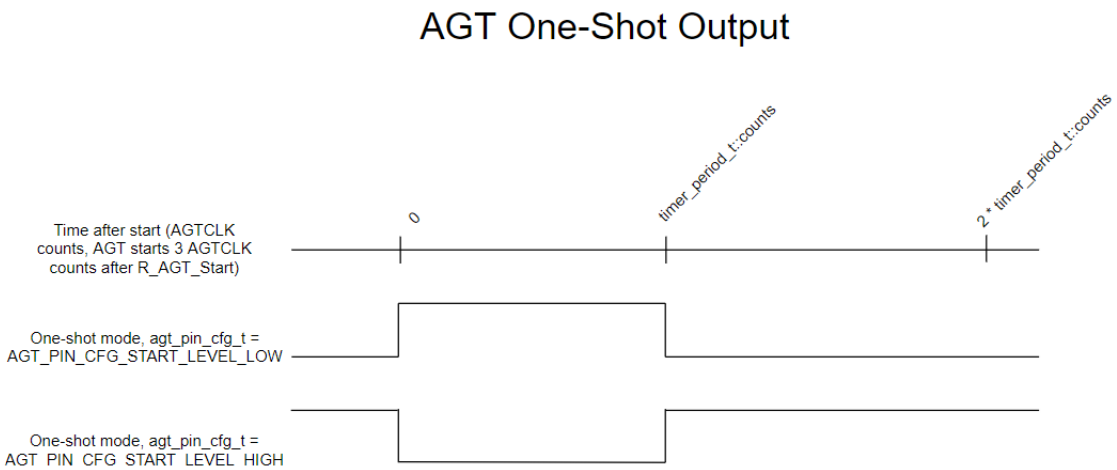



Figure 106: AGT One-Shot Output

### Periodic Output

The AGTOA or AGTOB pin toggles twice each time the timer expires in periodic mode. This is achieved by defining a PWM wave at a 50 percent duty cycle so that the period of the resulting square (from rising edge to rising edge) matches the period of the AGT timer. Since the periodic output is actually a PWM output, the time at the stop level is one cycle shorter than the time opposite the stop level for odd period values.

Examples of periodic signals that can be generated by this module are shown below:

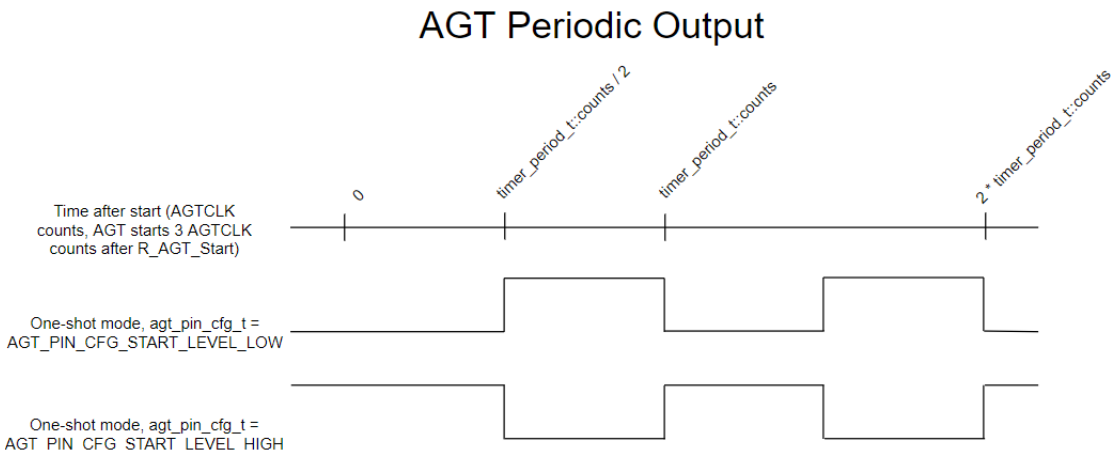



Figure 107: AGT Periodic Output

### PWM Output

This module does not support in phase PWM output. The PWM output signal is low at the beginning of the cycle and high at the end of the cycle.

Examples of PWM signals that can be generated by this module are shown below:

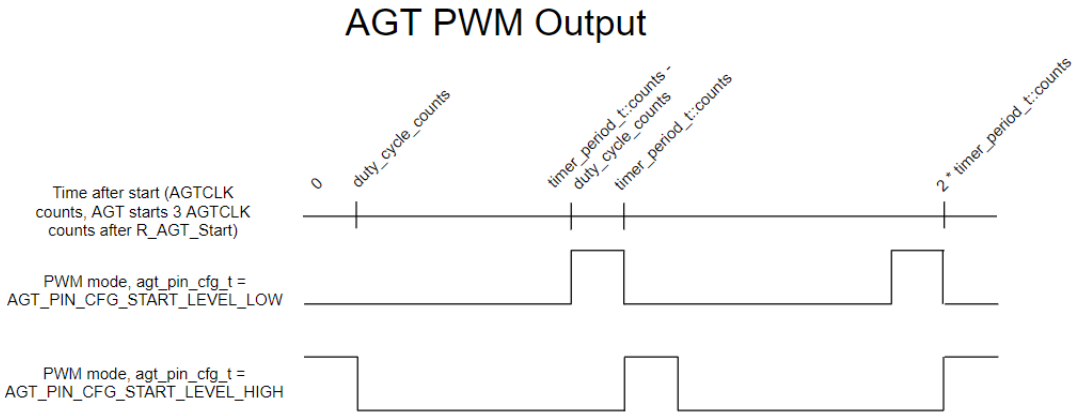



Figure 108: AGT PWM Output

### Triggering ELC Events with AGT

The AGT timer can trigger the start of other peripherals. The Event Link Controller (r_elc) guide provides a list of all available peripherals.

# Examples

### AGT Basic Example

This is a basic example of minimal use of the AGT in an application.

```
void agt_basic_example (void)
{
```

```
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Start the timer. */
    (void) R_AGT_Start(&g_timer0_ctrl);
}
```

### AGT Callback Example

This is an example of a timer callback.

```
/* Example callback called when timer expires. */
void timer_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* Add application code to be called periodically here. */
    }
}
```

### AGT Free Running Counter Example

To use the AGT as a free running counter, select periodic mode and set the the Period to 0xFFFF.

```
void agt_counter_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Start the timer. */
    (void) R_AGT_Start(&g_timer0_ctrl);
```

```
 /* (Optional) Stop the timer. */
    (void) R_AGT_Stop(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_AGT_StatusGet(&g_timer0_ctrl, &status);
}
```

### AGT Input Capture Example

This is an example of using the AGT to capture pulse width or pulse period measurements.

```
/* Example callback called when a capture occurs. */
uint64_t g_captured_time    = 0U;
uint32_t g_capture_overflows = 0U;
void timer_capture_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CAPTURE_A == p_args->event)
    {
 /* (Optional) Get the current period if not known. */
 timer_info_t info;
       (void) R_AGT_InfoGet(&g_timer0_ctrl, &info);
       uint32_t period = info.period_counts;
 /* Process capture from AGTIO. */
       g_captured_time    = ((uint64_t) period * g_capture_overflows) +
p_args->capture;
       g_capture_overflows = 0U;
    }
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* An overflow occurred during capture. This must be accounted for at the
application layer. */
       g_capture_overflows++;
    }
}
void agt_capture_example (void)
```

```
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Enable captures. Captured values arrive in the interrupt. */
    (void) R_AGT_Enable(&g_timer0_ctrl);
 /* (Optional) Disable captures. */
    (void) R_AGT_Disable(&g_timer0_ctrl);
}
```

### AGT Period Update Example

This an example of updating the period.

```
#define AGT_EXAMPLE_MSEC_PER_SEC (1000)
#define AGT_EXAMPLE_DESIRED_PERIOD_MSEC (20)
/* This example shows how to calculate a new period value at runtime. */
void agt_period_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Start the timer. */
    (void) R_AGT_Start(&g_timer0_ctrl);
 /* Get the source clock frequency (in Hz). There are several ways to do this in FSP:
  * - If LOCO or subclock is chosen in agt_extended_cfg_t::clock_source
  * - The source clock frequency is BSP_LOCO_HZ >> timer_cfg_t::source_div
  * - If PCLKB is chosen in agt_extended_cfg_t::clock_source and the PCLKB frequency
has not changed since reset,
  * - The source clock frequency is BSP_STARTUP_PCLKB_HZ >> timer_cfg_t::source_div
  * - Use the R_AGT_InfoGet function (it accounts for the clock source and divider).
```

```
  * - Calculate the current PCLKB frequency using
R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) and right shift
  * by timer_cfg_t::source_div.
  *
  * This example uses the last option (R_FSP_SystemClockHzGet).
  */
    uint32_t timer_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) >>
g_timer0_cfg.source_div;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX / pclkb_freq_hz. A cast to uint64_t is
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) (((uint64_t) timer_freq_hz * AGT_EXAMPLE_DESIRED_PERIOD_MSEC) /
AGT_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. This will return an error if parameter checking is
enabled and the calculated
  * period is larger than UINT16_MAX. */
    err = R_AGT_PeriodSet(&g_timer0_ctrl, period_counts);
    handle_error(err);
}
```

### AGT Duty Cycle Update Example

This an example of updating the duty cycle.

```
#define AGT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT (25)
#define AGT_EXAMPLE_MAX_PERCENT (100)
/* This example shows how to calculate a new duty cycle value at runtime. */
void agt_duty_cycle_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
```

```
    handle_error(err);
 /* Start the timer. */
    (void) R_AGT_Start(&g_timer0_ctrl);
 /* Get the current period setting. */
 timer_info_t info;
    (void) R_AGT_InfoGet(&g_timer0_ctrl, &info);
    uint32_t current_period_counts = info.period_counts;
 /* Calculate the desired duty cycle based on the current period. */
    uint32_t duty_cycle_counts = (current_period_counts *
AGT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                                AGT_EXAMPLE_MAX_PERCENT;
 /* Set the calculated duty cycle. */
    err = R_AGT_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts, AGT_OUTPUT_PIN_AGTOA
);
    handle_error(err);
}
```

### AGT Cascaded Timers Example

This an example of using AGT0 underflow as the count source for AGT1.

```
/* This example shows how use cascaded timers. The count source for AGT channel 1 is
set to AGT0 underflow. */
void agt_cascaded_timers_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the timers in any order. */
    err = R_AGT_Open(&g_timer_channel0_ctrl, &g_timer_channel0_cfg);
    handle_error(err);
    err = R_AGT_Open(&g_timer_channel1_ctrl, &g_timer_channel1_cfg);
    handle_error(err);
 /* Start AGT channel 1 first. */
    (void) R_AGT_Start(&g_timer_channel1_ctrl);
    (void) R_AGT_Start(&g_timer_channel0_ctrl);
 /* (Optional) Stop AGT channel 0 first. */
```

```
    (void) R_AGT_Stop(&g_timer_channel0_ctrl);
    (void) R_AGT_Stop(&g_timer_channel1_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_AGT_StatusGet(&g_timer_channel1_ctrl, &status);
}
```

## Data Structures

| | |
|---|---|
| struct | agt_instance_ctrl_t |
| struct | agt_extended_cfg_t |

## Enumerations

| | |
|---|---|
| enum | agt_clock_t |
| enum | agt_measure_t |
| enum | agt_agtio_filter_t |
| enum | agt_enable_pin_t |
| enum | agt_trigger_edge_t |
| enum | agt_output_pin_t |
| enum | agt_pin_cfg_t |

## Data Structure Documentation

### ◆ agt_instance_ctrl_t

| struct agt_instance_ctrl_t | | |
|---|---|---|
| Channel control block. DO NOT INITIALIZE. Initialization occurs when timer_api_t::open is called. | | |
| Data Fields | | |
| uint32_t | open | Whether or not channel is open. |
| const timer_cfg_t * | p_cfg | Pointer to initial configurations. |
| R_AGT0_Type * | p_reg | Base register for this channel. |
| uint32_t | period | Current timer period (counts) |

### ◆ agt_extended_cfg_t

| struct agt_extended_cfg_t |
|---|
| Optional AGT extension data structure. |

| Data Fields | | |
|---|---|---|
| agt_clock_t | count_source | AGT channel clock source. Valid values are: AGT_CLOCK_PCLKB, AGT_CLOCK_LOCO, AGT_CLOCK_FSUB. |
| union agt_extended_cfg_t | __unnamed__ | |
| agt_pin_cfg_t | agto: 3 | Configure AGTO pin.<br><br>*Note*<br>*AGTIO polarity is opposite AGTO* |
| agt_measure_t | measurement_mode | Measurement mode. |
| agt_agtio_filter_t | agtio_filter | Input filter for AGTIO. |
| agt_enable_pin_t | enable_pin | Enable pin (event counting only) |
| agt_trigger_edge_t | trigger_edge | Trigger edge to start pulse period measurement or count external event. |

## Enumeration Type Documentation

### ◆ agt_clock_t

| enum agt_clock_t | |
|---|---|
| Count source | |
| Enumerator | |
| AGT_CLOCK_PCLKB | PCLKB count source, division by 1, 2, or 8 allowed. |
| AGT_CLOCK_LOCO | LOCO count source, division by 1, 2, 4, 8, 16, 32, 64, or 128 allowed. |
| AGT_CLOCK_AGT0_UNDERFLOW | Underflow event signal from AGT0, division must be 1. |
| AGT_CLOCK_SUBCLOCK | Subclock count source, division by 1, 2, 4, 8, 16, 32, 64, or 128 allowed. |
| AGT_CLOCK_P402 | Counts events on P402, events are counted in deep software standby mode. |
| AGT_CLOCK_P403 | Counts events on P403, events are counted in deep software standby mode. |
| AGT_CLOCK_AGTIO | Counts events on AGTIOn, events are not counted in software standby modes. |

### ◆ agt_measure_t

| enum agt_measure_t | |
|---|---|
| Enable pin for event counting mode. | |
| Enumerator | |
| AGT_MEASURE_DISABLED | AGT used as a counter. |
| AGT_MEASURE_PULSE_WIDTH_LOW_LEVEL | AGT used to measure low level pulse width. |
| AGT_MEASURE_PULSE_WIDTH_HIGH_LEVEL | AGT used to measure high level pulse width. |
| AGT_MEASURE_PULSE_PERIOD | AGT used to measure pulse period. |

### ◆ agt_agtio_filter_t

| enum agt_agtio_filter_t | |
|---|---|
| Input filter, applies AGTIO in pulse period measurement, pulse width measurement, or event counter mode. The filter requires the signal to be at the same level for 3 successive reads at the specified filter frequency. | |
| Enumerator | |
| AGT_AGTIO_FILTER_NONE | No filter. |
| AGT_AGTIO_FILTER_PCLKB | Filter at PCLKB. |
| AGT_AGTIO_FILTER_PCLKB_DIV_8 | Filter at PCLKB / 8. |
| AGT_AGTIO_FILTER_PCLKB_DIV_32 | Filter at PCLKB / 32. |

### ◆ agt_enable_pin_t

| enum agt_enable_pin_t | |
|---|---|
| Enable pin for event counting mode. | |
| Enumerator | |
| AGT_ENABLE_PIN_NOT_USED | AGTEE is not used. |
| AGT_ENABLE_PIN_ACTIVE_LOW | Events are only counted when AGTEE is low. |
| AGT_ENABLE_PIN_ACTIVE_HIGH | Events are only counted when AGTEE is high. |

### ◆ agt_trigger_edge_t

| enum agt_trigger_edge_t | |
|---|---|
| Trigger edge for pulse period measurement mode and event counting mode. | |
| Enumerator | |
| AGT_TRIGGER_EDGE_RISING | Measurement starts or events are counted on rising edge. |
| AGT_TRIGGER_EDGE_FALLING | Measurement starts or events are counted on falling edge. |
| AGT_TRIGGER_EDGE_BOTH | Events are counted on both edges (n/a for pulse period mode) |

### ◆ agt_output_pin_t

| enum agt_output_pin_t | |
|---|---|
| Output pins, used to select which duty cycle to update in R_AGT_DutyCycleSet(). | |
| Enumerator | |
| AGT_OUTPUT_PIN_AGTOA | GTIOCA. |
| AGT_OUTPUT_PIN_AGTOB | GTIOCB. |

### ◆ agt_pin_cfg_t

| enum agt_pin_cfg_t | |
|---|---|
| Level of AGT pin | |
| Enumerator | |
| AGT_PIN_CFG_DISABLED | Not used as output pin. |
| AGT_PIN_CFG_START_LEVEL_LOW | Pin level low. |
| AGT_PIN_CFG_START_LEVEL_HIGH | Pin level high. |

## Function Documentation

### ◆ R_AGT_Close()

fsp_err_t R_AGT_Close ( timer_ctrl_t *const *p_ctrl*)

Stops counter, disables interrupts, disables output pins, and clears internal driver data. Implements timer_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer closed. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_AGT_PeriodSet()

fsp_err_t R_AGT_PeriodSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *period_counts* )

Updates period. The new period is updated immediately and the counter is reset to the maximum value. Implements timer_api_t::periodSet.

**Warning**

If periodic output is used, the duty cycle buffer registers are updated after the period buffer register. If this function is called while the timer is running and an AGT underflow occurs during processing, the duty cycle will not be the desired 50% duty cycle until the counter underflow after processing completes.
Stop the timer before calling this function if one-shot output is used.

Example:

```
 /* Get the source clock frequency (in Hz). There are several ways to do this in FSP:
  * - If LOCO or subclock is chosen in agt_extended_cfg_t::clock_source
  * - The source clock frequency is BSP_LOCO_HZ >> timer_cfg_t::source_div
  * - If PCLKB is chosen in agt_extended_cfg_t::clock_source and the PCLKB frequency
has not changed since reset,
  * - The source clock frequency is BSP_STARTUP_PCLKB_HZ >> timer_cfg_t::source_div
  * - Use the R_AGT_InfoGet function (it accounts for the clock source and divider).
  * - Calculate the current PCLKB frequency using
R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) and right shift
  * by timer_cfg_t::source_div.
  *
  * This example uses the last option (R_FSP_SystemClockHzGet).
  */
    uint32_t timer_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) >>
g_timer0_cfg.source_div;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX / pclkb_freq_hz. A cast to uint64_t is
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) (((uint64_t) timer_freq_hz * AGT_EXAMPLE_DESIRED_PERIOD_MSEC) /
AGT_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. This will return an error if parameter checking is
enabled and the calculated
```

```
 * period is larger than UINT16_MAX. */
   err = R_AGT_PeriodSet(&g_timer0_ctrl, period_counts);
   handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Period value updated. |
| FSP_ERR_ASSERTION | A required pointer was NULL, or the period was not in the valid range of 1 to 0xFFFF. |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_AGT_DutyCycleSet()

fsp_err_t R_AGT_DutyCycleSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *duty_cycle_counts*, uint32_t const *pin* )

Updates duty cycle. If the timer is counting, the new duty cycle is reflected after the next counter underflow. Implements timer_api_t::dutyCycleSet.

Example:

```
/* Get the current period setting. */
timer_info_t info;
   (void) R_AGT_InfoGet(&g_timer0_ctrl, &info);
   uint32_t current_period_counts = info.period_counts;
/* Calculate the desired duty cycle based on the current period. */
   uint32_t duty_cycle_counts = (current_period_counts *
AGT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                                AGT_EXAMPLE_MAX_PERCENT;
/* Set the calculated duty cycle. */
   err = R_AGT_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts, AGT_OUTPUT_PIN_AGTOA
);
   handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Duty cycle updated. |
| FSP_ERR_ASSERTION | A required pointer was NULL, or the pin was not AGT_AGTO_AGTOA or AGT_AGTO_AGTOB. |
| FSP_ERR_INVALID_ARGUMENT | Duty cycle was not in the valid range of 0 to period (counts) - 1 |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |
| FSP_ERR_UNSUPPORTED | AGT_CFG_OUTPUT_SUPPORT_ENABLE is 0. |

### ◆ R_AGT_Reset()

fsp_err_t R_AGT_Reset ( timer_ctrl_t *const *p_ctrl*)

Resets the counter value to the period minus one. Implements timer_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter reset. |
| FSP_ERR_ASSERTION | p_ctrl is NULL |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_AGT_Start()

fsp_err_t R_AGT_Start ( timer_ctrl_t *const *p_ctrl*)

Starts timer. Implements timer_api_t::start.

Example:

```
/* Start the timer. */
   (void) R_AGT_Start(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer started. |
| FSP_ERR_ASSERTION | p_ctrl is null. |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_AGT_Enable()

fsp_err_t R_AGT_Enable ( timer_ctrl_t *const *p_ctrl*)

Enables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::enable.

Example:

```
/* Enable captures. Captured values arrive in the interrupt. */

   (void) R_AGT_Enable(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | External events successfully enabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_AGT_Disable()

fsp_err_t R_AGT_Disable ( timer_ctrl_t *const *p_ctrl*)

Disables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::disable.

Example:

```
/* (Optional) Disable captures. */

   (void) R_AGT_Disable(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | External events successfully disabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_AGT_InfoGet()

fsp_err_t R_AGT_InfoGet ( timer_ctrl_t *const *p_ctrl*, timer_info_t *const *p_info* )

Gets timer information and store it in provided pointer p_info. Implements timer_api_t::infoGet.

Example:

```
/* (Optional) Get the current period if not known. */
timer_info_t info;
      (void) R_AGT_InfoGet(&g_timer0_ctrl, &info);
      uint32_t period = info.period_counts;
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Period, count direction, and frequency stored in p_info. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_AGT_StatusGet()

fsp_err_t R_AGT_StatusGet ( timer_ctrl_t *const *p_ctrl*, timer_status_t *const *p_status* )

Retrieves the current state and counter value stores them in p_status. Implements timer_api_t::statusGet.

Example:

```
/* Read the current counter value. Counter value is in status.counter. */
timer_status_t status;
   (void) R_AGT_StatusGet(&g_timer0_ctrl, &status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Current status and counter value provided in p_status. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_AGT_Stop()

fsp_err_t R_AGT_Stop ( timer_ctrl_t *const *p_ctrl*)

Stops the timer. Implements timer_api_t::stop.

Example:

```
/* (Optional) Stop the timer. */

   (void) R_AGT_Stop(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer stopped. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_AGT_Open()

fsp_err_t R_AGT_Open ( timer_ctrl_t *const *p_ctrl*, timer_cfg_t const *const *p_cfg* )

Initializes the AGT module instance. Implements timer_api_t::open.

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 agt_extended_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.

Example:

```
/* Initializes the module. */

   err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful and timer has started. |
| FSP_ERR_ASSERTION | A required input pointer is NULL or the period is not in the valid range of 1 to 0xFFFF. |
| FSP_ERR_ALREADY_OPEN | R_AGT_Open has already been called for this p_ctrl. |
| FSP_ERR_IRQ_BSP_DISABLED | A required interrupt has not been enabled in the vector table. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Requested channel number is not available on AGT. |

### ◆ R_AGT_VersionGet()

fsp_err_t R_AGT_VersionGet ( fsp_version_t *const *p_version*)

Sets driver version based on compile time macros. Implements timer_api_t::versionGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version in p_version. |
| FSP_ERR_ASSERTION | The parameter p_version is NULL. |

## 4.2.5 Bluetooth Low Energy Library (r_ble)

Modules

### Functions

| | |
|---|---|
| ble_status_t | R_BLE_Open (void)<br>Open the BLE protocol stack. More... |
| ble_status_t | R_BLE_Close (void)<br>Close the BLE protocol stack. More... |
| ble_status_t | R_BLE_Execute (void)<br>Execute the BLE task. More... |
| uint32_t | R_BLE_IsTaskFree (void)<br>Check the BLE task queue is free or not. More... |
| ble_status_t | R_BLE_SetEvent (ble_event_cb_t cb)<br>Set event. More... |
| uint32_t | R_BLE_GetVersion (void)<br>Get the BLE FIT module version. More... |
| uint32_t | R_BLE_GetLibType (void)<br>Get the type of BLE protocol stack library. More... |

### Detailed Description

Driver for the Radio peripheral on RA MCUs. This module implements the BLE Interface.

# Overview

The bluetooth low energy library (r_ble) provides an API to control the Radio peripheral. This module is configured via the QE for BLE.

### Features

- Common
  - Open/Close the BLE protocol stack.
  - Execute the BLE job.
  - Add an event in the BLE protocol stack internal queue.
- GAP
  - Initialization of the Host stack.
  - Start/Stop Advertising.
  - Start/Stop Scan.
  - Connect/Disconnect a link.
  - Initiate/Respond a pairing request.
- GATT Common
  - Get MTU size.
- GATT Server
  - Initialization of GATT Server.
  - Notification/Indication.
- GATT Client
  - Discovery services, characteristics.
  - Read/Write characteristic.
- L2CAP
  - Credit-based flow control transaction.
- Vendor Specific
  - DTM.
  - Set/Get transmit power.
  - Set/Get BD_ADDR.

### Target Devices

The Renesas Bluetooth Low Energy Library supports the following devices.

- RA4W1

# Configuration

### Clock Configuration

*Note*

*System clock (ICLK): 8 MHz or more*
*Peripheral module clock A (PCLKA): 8MHz or more*
*The BLE Protocol Stack is optimized for ICLK and PCLKA frequencies of 32 MHz.*
*It is recommended that the clock be set so that the ICLK and PCLKA frequencies are 32MHz in order to get the best performance from the BLE.*

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

Figure shows the software structure of the BLE FSP module.

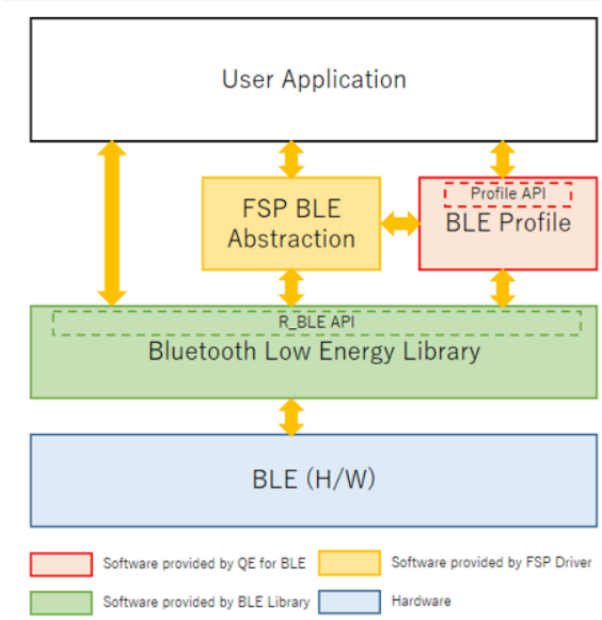



Figure 109: BLE software structure

The BLE FSP module consists of the BLE library.
The BLE Application uses the BLE functions via the R_BLE API provided by the BLE Library.
The QE for BLE generates the source codes (BLE base skeleton program) as a base for the BLE Application and the BLE Profile codes including the Profile API.

**Limitations**

Developers should be aware of the following limitations when using the ble:

**Modules**

| | |
|---|---|
| | GAP |
| | GATT_COMMON |
| | GATT_SERVER |
| | GATT_CLIENT |
| | L2CAP |
| | VS |

**Typedefs**

| | |
|---|---|
| typedef void(* | ble_event_cb_t) (void)<br>ble_event_cb_t is the callback function type for R_BLE_SetEvent(). More... |

## Typedef Documentation

### ◆ ble_event_cb_t

ble_event_cb_t

ble_event_cb_t is the callback function type for R_BLE_SetEvent().

**Parameters**

| | | |
|---|---|---|
| [in] | void | |

**Returns**

none

## Function Documentation

### ◆ R_BLE_Open()

ble_status_t R_BLE_Open ( void )

Open the BLE protocol stack.

This function should be called once before using the BLE protocol stack.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_Close()

ble_status_t R_BLE_Close ( void )

Close the BLE protocol stack.

This function should be called once to close the BLE protocol stack.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_Execute()

ble_status_t R_BLE_Execute ( void )

Execute the BLE task.

This handles all the task queued in the BLE protocol stack internal task queue and return. This function should be called repeatedly in the main loop.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_IsTaskFree()

uint32_t R_BLE_IsTaskFree ( void )

Check the BLE task queue is free or not.

This function returns the BLE task queue free status. When this function returns 0x0, call R_BLE_Execute() to execute the BLE task.

**Return values**

| | |
|---|---|
| 0x0 | BLE task queue is not free |
| 0x1 | BLE task queue is free |

### ◆ R_BLE_SetEvent()

ble_status_t R_BLE_SetEvent ( ble_event_cb_t *cb*)

Set event.

This function add an event in the BLE protocol stack internal queue. The event is handled in R_BLE_Execute just like Bluetooth event. This function is intended to be called in hardware interrupt context. Even if calling this function with the same cb before the cb is invoked, only one event is registered. The maximum number of the events can be registered at a time is eight.

**Parameters**

| | |
|---|---|
| cb | The callback for the event. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_ALREADY_IN_PROGRESS(0x000A) | The event already registered with the callback. |
| BLE_ERR_CONTEXT_FULL(0x000B) | No free slot for the event. |

### ◆ R_BLE_GetVersion()

| uint32_t R_BLE_GetVersion ( void ) |
|---|
| Get the BLE FIT module version.<br><br>This function returns the BLE FIT module version.<br>The major version(BLE_VERSION_MAJOR) is contained in the two most significant bytes, and the minor version(BLE_VERSION_MINOR) occupies the remaining two bytes.<br><br>**Return values**<br>BLE_VERSION_MAJOR \| BLE_VERSION_MINOR |

### ◆ R_BLE_GetLibType()

| uint32_t R_BLE_GetLibType ( void ) | |
|---|---|
| Get the type of BLE protocol stack library.<br><br>This function returns the type of BLE protocol stack library.<br><br>**Return values** | |
| BLE_LIB_ALL_FEATS(0x00) | All Features |
| BLE_LIB_BALANCE(0x01) | Balance |
| BLE_LIB_COMPACT(0x02) | Compact |

## 4.2.5.2 GATT_COMMON

Modules » Bluetooth Low Energy Library (r_ble)

### Functions

| | |
|---|---|
| ble_status_t | R_BLE_GATT_GetMtu (uint16_t conn_hdl, uint16_t *p_mtu)<br><br>This function gets the current MTU used in GATT communication.<br>More... |

### Detailed Description

### Function Documentation

### ◆ R_BLE_GATT_GetMtu()

ble_status_t R_BLE_GATT_GetMtu ( uint16_t *conn_hdl*, uint16_t * *p_mtu* )

This function gets the current MTU used in GATT communication.

Both GATT server and GATT Client can use this function.
The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the GATT Server or the GATT Client. |
| [in] | p_mtu | The Current MTU. Before MTU exchange, this parameter is 23 bytes. After MTU exchange, this parameter is the negotiated MTU. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The mtu parameter is NULL. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server or the GATT Client specified by conn_hdl was not found. |

## 4.2.6 Clock Frequency Accuracy Measurement Circuit (r_cac)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_CAC_Open (cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg) |
| fsp_err_t | R_CAC_StartMeasurement (cac_ctrl_t *const p_ctrl) |
| fsp_err_t | R_CAC_StopMeasurement (cac_ctrl_t *const p_ctrl) |
| fsp_err_t | R_CAC_Read (cac_ctrl_t *const p_ctrl, uint16_t *const p_counter) |
| fsp_err_t | R_CAC_Close (cac_ctrl_t *const p_ctrl) |
| fsp_err_t | R_CAC_VersionGet (fsp_version_t *const p_version) |

**Detailed Description**

Driver for the CAC peripheral on RA MCUs. This module implements the CAC Interface.

# Overview

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 measurement clock edges that occur between two edges of the reference clock.

### Features

- Supports clock frequency-measurement and monitoring based on a reference signal input
- Reference can be either an externally supplied clock source or an internal clock source
- 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.

# Configuration

### Build Time Configurations for r_cac

The following build time configurations are defined in fsp_cfg/r_cac_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Monitoring > Clock Accuracy Circuit Driver on r_cac

This module can be added to the Stacks tab via New Stack > Driver > Monitoring > Clock Accuracy Circuit Driver on r_cac:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_cac0 | Module name. |
| Reference clock divider | • 32<br>• 128<br>• 1024<br>• 8192 | 32 | Reference clock divider. |
| Reference clock source | • Main Oscillator<br>• Sub-clock<br>• HOCO<br>• MOCO<br>• LOCO<br>• PCLKB | Main Oscillator | Reference clock source. |

| | | | |
|---|---|---|---|
| | • IWDT<br>• External | | |
| Reference clock digital filter | • Disabled<br>• Sampling clock =Measuring freq<br>• Sampling clock =Measuring freq/4<br>• Sampling clock =Measuring freq/16 | Disabled | Reference clock digital filter. |
| Reference clock edge detect | • Rising<br>• Falling<br>• Both | Rising | Reference clock edge detection. |
| Measurement clock divider | • 1<br>• 4<br>• 8<br>• 32 | 1 | Measurement clock divider. |
| Measurement clock source | • Main Oscillator<br>• Sub-clock<br>• HOCO<br>• MOCO<br>• LOCO<br>• PCLKB<br>• IWDT | HOCO | Measurement clock source. |
| Upper Limit Threshold | Value must be a non-negative integer, between 0 to 65535 | 0 | Top end of allowable range for measurement completion. |
| Lower Limit Threshold | Value must be a non-negative integer, between 0 to 65535 | 0 | Bottom end of allowable range for measurement completion. |
| Frequency Error Interrupt Priority | MCU Specific Options | | CAC frequency error interrupt priority. |
| Measurement End Interrupt Priority | MCU Specific Options | | CAC measurement end interrupt priority. |
| Overflow Interrupt Priority | MCU Specific Options | | CAC overflow interrupt priority. |
| Callback | Name must be a valid C symbol | NULL | Function name for callback |

### Clock Configuration

The CAC measurement clock source can be configured as the following:

1. MAIN_OSC
2. SUBCLOCK
3. HOCO

4. MOCO
5. LOCO
6. PCLKB
7. IWDT

The CAC reference clock source can be configured as the following:

1. MAIN_OSC
2. SUBCLOCK
3. HOCO
4. MOCO
5. LOCO
6. PCLKB
7. IWDT
8. External Clock Source (CACREF)

### Pin Configuration

The CACREF pin can be configured to provide the reference clock for CAC measurements.

## Usage Notes

### Measurement Accuracy

The clock measurement result may be off by up to one pulse depending on the phase difference between the edge detection circuit, digital filter, and CACREF pin signal, if applicable.

### Frequency Error Interrupt

The frequency error interrupt is only triggered at the end of a CAC measurement. This means that there will be a measurement complete interrupt in addition to the frequency error interrupt.

## Examples

### Basic Example

This is a basic example of minimal use of the CAC in an application.

```
volatile uint32_t g_callback_complete;
void cac_basic_example ()
{
    g_callback_complete = 0;
 fsp_err_t err = R_CAC_Open(&g_cac_ctrl, &g_cac_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
    (void) R_CAC_StartMeasurement(&g_cac_ctrl);
 /* Wait for measurement to complete. */
 while (0 == g_callback_complete)
```

```
    {
    }
    uint16_t value;
 /* Read the CAC measurement. */
    (void) R_CAC_Read(&g_cac_ctrl, &value);
}
/* Called when measurement is completed. */
static void r_cac_callback (cac_callback_args_t * p_args)
{
 if (CAC_EVENT_MEASUREMENT_COMPLETE == p_args->event)
    {
       g_callback_complete = 1U;
    }
}
```

### Data Structures

| | |
|---|---|
| struct | cac_instance_ctrl_t |

## Data Structure Documentation

### ◆ cac_instance_ctrl_t

| struct cac_instance_ctrl_t |
|---|
| CAC instance control block. DO NOT INITIALIZE. |

## Function Documentation

### ◆ R_CAC_Open()

| fsp_err_t R_CAC_Open ( cac_ctrl_t *const *p_ctrl*, cac_cfg_t const *const *p_cfg* ) |
|---|
| The Open function configures the CAC based on the provided user configuration settings. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CAC is available and available for measurement(s). |
| FSP_ERR_ASSERTION | An argument is invalid. |
| FSP_ERR_ALREADY_OPEN | The CAC has already been opened. |

***Note***

*There is only a single CAC peripheral.*

### ◆ R_CAC_StartMeasurement()

fsp_err_t R_CAC_StartMeasurement ( cac_ctrl_t *const *p_ctrl*)

Start the CAC measurement process.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CAC measurement started. |
| FSP_ERR_ASSERTION | NULL provided for p_instance_ctrl or p_cfg. |
| FSP_ERR_NOT_OPEN | R_CAC_Open() has not been successfully called. |

### ◆ R_CAC_StopMeasurement()

fsp_err_t R_CAC_StopMeasurement ( cac_ctrl_t *const *p_ctrl*)

Stop the CAC measurement process.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CAC measuring has been stopped. |
| FSP_ERR_ASSERTION | NULL provided for p_instance_ctrl or p_cfg. |
| FSP_ERR_NOT_OPEN | R_CAC_Open() has not been successfully called. |

### ◆ R_CAC_Read()

fsp_err_t R_CAC_Read ( cac_ctrl_t *const *p_ctrl*, uint16_t *const *p_counter* )

Read and return the CAC status and counter registers.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CAC read successful. |
| FSP_ERR_ASSERTION | An argument is NULL. |
| FSP_ERR_NOT_OPEN | R_CAC_Open() has not been successfully called. |

### ◆ R_CAC_Close()

fsp_err_t R_CAC_Close ( cac_ctrl_t *const *p_ctrl*)

Release any resources that were allocated by the Open() or any subsequent CAC operations.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | NULL provided for p_instance_ctrl or p_cfg. |
| FSP_ERR_NOT_OPEN | R_CAC_Open() has not been successfully called. |

### ◆ R_CAC_VersionGet()

fsp_err_t R_CAC_VersionGet ( fsp_version_t *const *p_version*)

Get the API and code version information.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version info returned. |
| FSP_ERR_ASSERTION | An argument is NULL. |

## 4.2.7 Controller Area Network (r_can)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_CAN_Open (can_ctrl_t *const p_api_ctrl, can_cfg_t const *const p_cfg) |
| fsp_err_t | R_CAN_Close (can_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_CAN_Write (can_ctrl_t *const p_api_ctrl, uint32_t const mailbox, can_frame_t *const p_frame) |
| fsp_err_t | R_CAN_ModeTransition (can_ctrl_t *const p_api_ctrl, can_operation_mode_t operation_mode, can_test_mode_t test_mode) |
| fsp_err_t | R_CAN_InfoGet (can_ctrl_t *const p_api_ctrl, can_info_t *const p_info) |

fsp_err_t R_CAN_VersionGet (fsp_version_t *const version)

### Detailed Description

Driver for the CAN peripheral on RA MCUs. This module implements the CAN Interface.

# Overview

The Controller Area network (CAN) HAL module provides a high-level API for CAN applications and supports the CAN peripherals available on RA microcontroller hardware. A user-callback function must be defined that the driver will invoke when transmit, receive or error interrupts are received. The callback is passed a parameter which indicates the channel, mailbox and event as well as the received data (if available).

### Features

- Supports both standard (11-bit) and extended (29-bit) messaging formats
- Supports speeds upto 1 Mbps
- Support for bit timing configuration as defined in the CAN specification
- Supports 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
- Receive mailboxes can be configured to receive a range of IDs using mailbox masks
- Mailboxes can be configured with Overwrite or Overrun mode
- Supports a user-callback function when transmit, receive, or error interrupts are received

# Configuration

### Build Time Configurations for r_can

The following build time configurations are defined in fsp_cfg/r_can_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Connectivity > CAN Driver on r_can

This module can be added to the Stacks tab via New Stack > Driver > Connectivity > CAN Driver on r_can:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General > Name | Name must be a valid C symbol | g_can0 | Module name. |
| General > Channel | Channel should be 0 or 1 | 0 | Specify the CAN channel to use. |
| General > Clock Source | MCU Specific Options | | Select the CAN clock |

| | | | |
|---|---|---|---|
| | | | source. |
| General > Sample-Point (%) | Must be a valid integer between 0 and 100. | 75 | Sample-Point = (TSEG1 + 1) / (TSEG1 + TSEG2 + 1). |
| General > CAN Baud Rate (Hz) | Must be a valid integer configurable upto maximum 4MHz. | 500000 | Specify baud rate in Hz. |
| General > Overwrite/Overrrun Mode | • Overwrite Mode<br>• Overrrun Mode | Overwrite Mode | Select whether receive mailbox will be overwritten or overrun if data is not read in time. |
| General > Standard or Extended ID Mode | • Standard ID Mode<br>• Extended ID Mode | Standard ID Mode | Select whether the driver will use the CAN standard or extended IDs. |
| General > Number of Mailboxes | • 4 Mailboxes<br>• 8 Mailboxes<br>• 16 Mailboxes<br>• 32 Mailboxes | 32 Mailboxes | Select 4, 8, 16 or 32 mailboxes. |
| Interrupts > Callback | Name must be a valid C symbol | can_callback | A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time any interrupt occurs. |
| Interrupts > Interrupt Priority Level | MCU Specific Options | | Error/Receive/Transmit interrupt priority. |
| Input > Mailbox 0-3 Group > Mailbox ID > Mailbox 0 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0 | 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. |
| Input > Mailbox 0-3 Group > Mailbox ID > Mailbox 1 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 1 | Select the receive ID for mailbox 1, 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. |
| Input > Mailbox 0-3 Group > Mailbox ID > | Value must be decimal or HEX integer of | 2 | Select the receive ID for mailbox 2, between |

| | | | |
|---|---|---|---|
| Mailbox 2 ID | 0x1FFFFFFF or less. | | 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. |
| Input > Mailbox 0-3 Group > Mailbox ID > Mailbox 3 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 3 | Select the receive ID for mailbox 3, 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. |
| Input > Mailbox 0-3 Group > Mailbox Type > Mailbox 0 Type | • Receive Mailbox<br>• Transmit Mailbox | Transmit Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 0-3 Group > Mailbox Type > Mailbox 1 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 0-3 Group > Mailbox Type > Mailbox 2 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 0-3 Group > Mailbox Type > Mailbox 3 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 0-3 Group > Mailbox Frame Type > Mailbox 0 Frame Type | • Data Mailbox<br>• Remote Mailbox | Remote Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 0-3 Group > Mailbox Frame Type > Mailbox 1 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 0-3 Group > Mailbox Frame Type > Mailbox 2 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |

| | | | |
|---|---|---|---|
| Input > Mailbox 0-3 Group > Mailbox Frame Type > Mailbox 3 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 0-3 Group > Mailbox 0-3 Group Mask | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0x1FFFFFFF | Select the Mask for mailboxes 0-3. |
| Input > Mailbox 4-7 Group > Mailbox ID > Mailbox 4 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 4 | Select the receive ID for mailbox 4, 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. |
| Input > Mailbox 4-7 Group > Mailbox ID > Mailbox 5 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 5 | Select the receive ID for mailbox 5, 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. |
| Input > Mailbox 4-7 Group > Mailbox ID > Mailbox 6 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 6 | Select the receive ID for mailbox 6, 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. |
| Input > Mailbox 4-7 Group > Mailbox ID > Mailbox 7 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 7 | Select the receive ID for mailbox 7, 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. |
| Input > Mailbox 4-7 Group > Mailbox Type > Mailbox 4 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 4-7 Group > Mailbox Type | • Receive Mailbox | Receive Mailbox | Select whether the mailbox is used for |

| | | | |
|---|---|---|---|
| > Mailbox 5 Type | • Transmit Mailbox | | receive or transmit. |
| Input > Mailbox 4-7 Group > Mailbox Type > Mailbox 6 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 4-7 Group > Mailbox Type > Mailbox 7 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 4-7 Group > Mailbox Frame Type > Mailbox 4 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 4-7 Group > Mailbox Frame Type > Mailbox 5 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 4-7 Group > Mailbox Frame Type > Mailbox 6 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 4-7 Group > Mailbox Frame Type > Mailbox 7 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 4-7 Group > Mailbox 4-7 Group Mask | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0x1FFFFFFF | >Select the Mask for mailboxes 4-7. |
| Input > Mailbox 8-11 Group > Mailbox ID > Mailbox 8 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 8 | Select the receive ID for mailbox 8, 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. |
| Input > Mailbox 8-11 Group > Mailbox ID > Mailbox 9 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 9 | Select the receive ID for mailbox 9, 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. |
| Input > Mailbox 8-11 Group > Mailbox ID > Mailbox 10 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 10 | Select the receive ID for mailbox 10, 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. |
| Input > Mailbox 8-11 Group > Mailbox ID > Mailbox 11 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 11 | Select the receive ID for mailbox 11, 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. |
| Input > Mailbox 8-11 Group > Mailbox Type > Mailbox 8 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 8-11 Group > Mailbox Type > Mailbox 9 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 8-11 Group > Mailbox Type > Mailbox 10 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 8-11 Group > Mailbox Type > Mailbox 11 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 8-11 Group > Mailbox Frame Type > Mailbox 8 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 8-11 Group > Mailbox Frame | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to |

| | | | |
|---|---|---|---|
| Type > Mailbox 9 Frame Type | | | capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 8-11 Group > Mailbox Frame Type > Mailbox 10 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 8-11 Group > Mailbox Frame Type > Mailbox 11 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 8-11 Group > Mailbox 8-11 Group Mask | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0x1FFFFFFF | Select the Mask for mailboxes 8-11. |
| Input > Mailbox 12-15 Group > Mailbox ID > Mailbox 12 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 12 | Select the receive ID for mailbox 12, 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. |
| Input > Mailbox 12-15 Group > Mailbox ID > Mailbox 13 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 13 | Select the receive ID for mailbox 13, 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. |
| Input > Mailbox 12-15 Group > Mailbox ID > Mailbox 14 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 14 | Select the receive ID for mailbox 14, 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. |
| Input > Mailbox 12-15 Group > Mailbox ID > | Value must be decimal or HEX integer of | 15 | Select the receive ID for mailbox 15, |

| | | | |
|---|---|---|---|
| Mailbox 15 ID | 0x1FFFFFFF or less. | | 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. |
| Input > Mailbox 12-15 Group > Mailbox Type > Mailbox 12 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 12-15 Group > Mailbox Type > Mailbox 13 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 12-15 Group > Mailbox Type > Mailbox 14 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 12-15 Group > Mailbox Type > Mailbox 15 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 12-15 Group > Mailbox Frame Type > Mailbox 12 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 12-15 Group > Mailbox Frame Type > Mailbox 13 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 12-15 Group > Mailbox Frame Type > Mailbox 14 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 12-15 Group > Mailbox Frame Type > Mailbox 15 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 12-15 | Value must be decimal | 0x1FFFFFFF | Select the Mask for |

| | | | |
|---|---|---|---|
| Group > Mailbox 12-15 Group Mask | or HEX integer of 0x1FFFFFFF or less. | | mailboxes 12-15. |
| Input > Mailbox 16-19 Group > Mailbox ID > Mailbox 16 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 16 | Select the receive ID for mailbox 16, 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. |
| Input > Mailbox 16-19 Group > Mailbox ID > Mailbox 17 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 17 | Select the receive ID for mailbox 17, 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. |
| Input > Mailbox 16-19 Group > Mailbox ID > Mailbox 18 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 18 | Select the receive ID for mailbox 18, 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. |
| Input > Mailbox 16-19 Group > Mailbox ID > Mailbox 19 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 19 | Select the receive ID for mailbox 19, 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. |
| Input > Mailbox 16-19 Group > Mailbox Type > Mailbox 16 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 16-19 Group > Mailbox Type > Mailbox 17 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 16-19 | • Receive | Receive Mailbox | Select whether the |

| | | | |
|---|---|---|---|
| Group > Mailbox Type > Mailbox 18 Type | Mailbox<br>• Transmit Mailbox | | mailbox is used for receive or transmit. |
| Input > Mailbox 16-19 Group > Mailbox Type > Mailbox 19 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 16-19 Group > Mailbox Frame Type > Mailbox 16 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 16-19 Group > Mailbox Frame Type > Mailbox 17 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 16-19 Group > Mailbox Frame Type > Mailbox 18 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 16-19 Group > Mailbox Frame Type > Mailbox 19 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 16-19 Group > Mailbox 16-19 Group Mask | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0x1FFFFFFF | Select the Mask for mailboxes 16-19. |
| Input > Mailbox 20-23 Group > Mailbox ID > Mailbox 20 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 20 | Select the receive ID for mailbox 20, 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. |
| Input > Mailbox 20-23 Group > Mailbox ID > Mailbox 21 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 21 | Select the receive ID for mailbox 21, 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. |
| Input > Mailbox 20-23 Group > Mailbox ID > Mailbox 22 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 22 | Select the receive ID for mailbox 22, 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. |
| Input > Mailbox 20-23 Group > Mailbox ID > Mailbox 23 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 23 | Select the receive ID for mailbox 23, 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. |
| Input > Mailbox 20-23 Group > Mailbox Type > Mailbox 20 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 20-23 Group > Mailbox Type > Mailbox 21 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 20-23 Group > Mailbox Type > Mailbox 22 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 20-23 Group > Mailbox Type > Mailbox 23 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 20-23 Group > Mailbox Frame Type > Mailbox 20 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 20-23 Group > Mailbox Frame Type > Mailbox 21 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored |

| | | | |
|---|---|---|---|
| | | | for transmit mailboxes). |
| Input > Mailbox 20-23 Group > Mailbox Frame Type > Mailbox 22 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 20-23 Group > Mailbox Frame Type > Mailbox 23 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 20-23 Group > Mailbox 20-23 Group Mask | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0x1FFFFFFF | Select the Mask for mailboxes 20-23 |
| Input > Mailbox 24-27 Group > Mailbox ID > Mailbox 24 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 24 | Select the receive ID for mailbox 24, 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. |
| Input > Mailbox 24-27 Group > Mailbox ID > Mailbox 25 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 25 | Select the receive ID for mailbox 25, 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. |
| Input > Mailbox 24-27 Group > Mailbox ID > Mailbox 26 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 26 | Select the receive ID for mailbox 26, 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. |
| Input > Mailbox 24-27 Group > Mailbox ID > Mailbox 27 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 27 | Select the receive ID for mailbox 27, 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. |
| Input > Mailbox 24-27 Group > Mailbox Type > Mailbox 24 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 24-27 Group > Mailbox Type > Mailbox 25 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 24-27 Group > Mailbox Type > Mailbox 26 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 24-27 Group > Mailbox Type > Mailbox 27 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 24-27 Group > Mailbox Frame Type > Mailbox 24 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 24-27 Group > Mailbox Frame Type > Mailbox 25 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 24-27 Group > Mailbox Frame Type > Mailbox 26 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 24-27 Group > Mailbox Frame Type > Mailbox 27 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 24-27 Group > Mailbox 24-27 Group Mask | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0x1FFFFFFF | Select the Mask for mailboxes 24-27. |

| | | | |
|---|---|---|---|
| Input > Mailbox 28-31 Group > Mailbox ID > Mailbox 28 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 28 | Select the receive ID for mailbox 28, 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. |
| Input > Mailbox 28-31 Group > Mailbox ID > Mailbox 29 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 29 | Select the receive ID for mailbox 29, 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. |
| Input > Mailbox 28-31 Group > Mailbox ID > Mailbox 30 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 30 | Select the receive ID for mailbox 30, 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. |
| Input > Mailbox 28-31 Group > Mailbox ID > Mailbox 31 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 31 | Select the receive ID for mailbox 31, 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. |
| Input > Mailbox 28-31 Group > Mailbox Type > Mailbox 28 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 28-31 Group > Mailbox Type > Mailbox 29 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 28-31 Group > Mailbox Type > Mailbox 30 Type | • Receive Mailbox<br>• Transmit | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |

| | | | |
|---|---|---|---|
| | Mailbox | | |
| Input > Mailbox 28-31 Group > Mailbox Type > Mailbox 31 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 28-31 Group > Mailbox Frame Type > Mailbox 28 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 28-31 Group > Mailbox Frame Type > Mailbox 29 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 28-31 Group > Mailbox Frame Type > Mailbox 30 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 28-31 Group > Mailbox Frame Type > Mailbox 31 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Input > Mailbox 28-31 Group > Mailbox 28-31 Group Mask | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0x1FFFFFFF | Select the Mask for mailboxes 28-31. |

### Clock Configuration

The CAN peripheral uses the CANMCLK (main-clock oscillator) or PCLKB as its clock source (fCAN, CAN System Clock.) Using the PCLKB with the default of 60 MHz and the default CAN configuration will provide a CAN bit rate of 500 Kbit. To set the PCLKB frequency, use the **Clocks** tab of the RA Configuration editor. To change the clock frequency at run-time, use the CGC Interface. Refer to the CGC module guide for more information on configuring clocks.

- 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 RA6, RA4 and RA2 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 (fCANCLK = XTAL / Baud Rate Prescaler)
- For RA6 and RA4 MCUs, the source of the peripheral module clocks must be PLL for the CAN HAL module when the clock source is PCLKB.
- For RA4 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 RA2 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.

### Pin Configuration

The CAN 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. A CAN channel would consist of two pins - CRX and CTX for data transmission/reception.

## Usage Notes

### Bit Rate Calculation

The baudrate of the CAN peripheral is automatically set through the RA Configuration editor. For more details on how the baudrate is set refer to section 37.4 "Data Transfer Rate Configuration" of the RA6M3 User's Manual (R01UH0886EJ0100).

## Examples

### Basic Example

This is a basic example of minimal use of the CAN in an application.

```
can_frame_t g_can_tx_frame;
can_frame_t g_can_rx_frame;
volatile bool     g_rx_flag = false;
volatile bool     g_tx_flag = false;
volatile bool     g_err_flag = false;
volatile uint32_t g_rx_id;
void can_callback (can_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case CAN_EVENT_RX_COMPLETE:    /* Receive complete event. */
      {
            g_rx_flag = true;
            g_rx_id   = p_args->p_frame->id;
 /* Read received frame */
            memcpy(&g_can_rx_frame, p_args->p_frame, sizeof(can_frame_t));
 break;
       }
 case CAN_EVENT_TX_COMPLETE:    /* Transmit complete event. */
```

```
     {
           g_tx_flag = true;
 break;
      }
 case CAN_EVENT_ERR_BUS_OFF:          /* Bus error event. (bus off) */
 case CAN_EVENT_ERR_PASSIVE:          /* Bus error event. (error passive) */
 case CAN_EVENT_ERR_WARNING:          /* Bus error event. (error warning) */
 case CAN_EVENT_BUS_RECOVERY:         /* Bus error event. (bus recovery) */
 case CAN_EVENT_MAILBOX_MESSAGE_LOST: /* Overwrite/overrun error */
     {
 /* Set error flag */
           g_err_flag = true;
 break;
      }
 default:
     {
 break;
      }
    }
}
void basic_example (void)
{
 fsp_err_t err;
    uint32_t i;
    uint32_t timeout_ms = CAN_BUSY_DELAY;
 /* Initialize the CAN module */
    err = R_CAN_Open(&g_can0_ctrl, &g_can0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
    g_can_tx_frame.id               = CAN_DESTINATION_DEVICE_MAILBOX_NUMBER; /* CAN
Destination Device ID */
    g_can_tx_frame.type             = CAN_FRAME_TYPE_DATA;
    g_can_tx_frame.data_length_code = CAN_FRAME_TRANSMIT_DATA_BYTES;
 /* Write some data to the transmit frame */
```

```
 for (i = 0; i < sizeof(g_can_tx_frame.data); i++)
    {
       g_can_tx_frame.data[i] = (uint8_t) i;
    }
 /* Send data on the bus */
    g_tx_flag = false;
    g_err_flag = false;
    err = R_CAN_Write(&g_can0_ctrl, CAN_MAILBOX_NUMBER_31, &g_can_tx_frame);
    handle_error(err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((true != g_tx_flag) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (true == g_err_flag)
    {
       __BKPT(0);
    }
}
```

### External Loop-back Test

This example requires a 120 Ohm resistor connected across channel 0 CAN pins. The mailbox numbers are arbitrarily chosen.

```
void can_external_loopback_example (void)
{
 fsp_err_t            err;
    uint32_t             timeout_ms    = CAN_BUSY_DELAY;
 can_operation_mode_t operation_mode = CAN_OPERATION_MODE_NORMAL;
 can_test_mode_t      test_mode      = CAN_TEST_MODE_LOOPBACK_EXTERNAL;
 int      diff = 0;
    uint32_t i    = 0;
    err = R_CAN_Open(&g_can0_ctrl, &g_can0_cfg);
```

```
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
    err = R_CAN_ModeTransition(&g_can0_ctrl, operation_mode, test_mode);
    handle_error(err);
 /* Clear the data part of receive frame */
    memset(g_can_rx_frame.data, 0, CAN_FRAME_TRANSMIT_DATA_BYTES);
 /* CAN Destination Device ID, in this case it is the same device with another
mailbox */
    g_can_tx_frame.id               = CAN_MAILBOX_NUMBER_4;
    g_can_tx_frame.type             = CAN_FRAME_TYPE_DATA;
    g_can_tx_frame.data_length_code = CAN_FRAME_TRANSMIT_DATA_BYTES;
 /* Write some data to the transmit frame */
 for (i = 0; i < sizeof(g_can_tx_frame.data); i++)
    {
       g_can_tx_frame.data[i] = (uint8_t) i;
    }
 /* Send data on the bus */
    g_rx_flag = false;
    g_err_flag = false;
    err = R_CAN_Write(&g_can0_ctrl, CAN_MAILBOX_NUMBER_31, &g_can_tx_frame);
    handle_error(err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((true != g_rx_flag) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (true == g_err_flag)
    {
       __BKPT(0);
    }
 /* Verify received data */
    diff = memcmp(&g_can_rx_frame.data[0], &g_can_tx_frame.data[0],
CAN_FRAME_TRANSMIT_DATA_BYTES);
```

```
 if (0 != diff)
    {
       __BKPT(0);
    }
}
```

## Function Documentation

### ◆ R_CAN_Open()

fsp_err_t R_CAN_Open ( can_ctrl_t *const *p_api_ctrl*, can_cfg_t const *const *p_cfg* )

Open and configure the CAN channel for operation.

Example:

```
/* Initialize the CAN module */
   err = R_CAN_Open(&g_can0_ctrl, &g_can0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully |
| FSP_ERR_ALREADY_OPEN | Driver already open. |
| FSP_ERR_CAN_INIT_FAILED | Channel failed to initialize. |
| FSP_ERR_ASSERTION | Null pointer presented. |

### ◆ R_CAN_Close()

fsp_err_t R_CAN_Close ( can_ctrl_t *const *p_api_ctrl*)

Close the CAN channel.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel closed successfully. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_ASSERTION | Null pointer presented. |

### ◆ R_CAN_Write()

fsp_err_t R_CAN_Write ( can_ctrl_t *const *p_api_ctrl*, uint32_t *mailbox*, can_frame_t *const *p_frame* )

Write data to the CAN channel. Write up to eight bytes to the channel mailbox.

Example:

```
err = R_CAN_Write(&g_can0_ctrl, CAN_MAILBOX_NUMBER_31, &g_can_tx_frame);
handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_CAN_TRANSMIT_NOT_READY | Transmit in progress, cannot write data at this time. |
| FSP_ERR_CAN_RECEIVE_MAILBOX | Mailbox is setup for receive and cannot send. |
| FSP_ERR_INVALID_ARGUMENT | Data length or frame type invalid. |
| FSP_ERR_ASSERTION | Null pointer presented |

### ◆ R_CAN_ModeTransition()

fsp_err_t R_CAN_ModeTransition ( can_ctrl_t *const *p_api_ctrl*, can_operation_mode_t *operation_mode*, can_test_mode_t *test_mode* )

CAN Mode Transition is used to change CAN driver state.

Example:

```
err = R_CAN_ModeTransition(&g_can0_ctrl, operation_mode, test_mode);
handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_ASSERTION | Null pointer presented |

### ◆ R_CAN_InfoGet()

| fsp_err_t R_CAN_InfoGet ( can_ctrl_t *const *p_api_ctrl*, can_info_t *const *p_info* ) |
|---|
| Get CAN state and status information for the channel. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_ASSERTION | Null pointer presented |

### ◆ R_CAN_VersionGet()

| fsp_err_t R_CAN_VersionGet ( fsp_version_t *const *p_version*) |
|---|
| Get CAN module code and API versions. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Null pointer presented note This function is reentrant. |

## 4.2.8 Clock Generation Circuit (r_cgc)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_CGC_Open (cgc_ctrl_t *const p_ctrl, cgc_cfg_t const *const p_cfg) |
| fsp_err_t | R_CGC_ClocksCfg (cgc_ctrl_t *const p_ctrl, cgc_clocks_cfg_t const *const p_clock_cfg) |
| fsp_err_t | R_CGC_ClockStart (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_pll_cfg_t const *const p_pll_cfg) |
| fsp_err_t | R_CGC_ClockStop (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source) |
| fsp_err_t | R_CGC_ClockCheck (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source) |

| | |
|---|---|
| fsp_err_t | R_CGC_SystemClockSet (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_divider_cfg_t const *const p_divider_cfg) |
| fsp_err_t | R_CGC_SystemClockGet (cgc_ctrl_t *const p_ctrl, cgc_clock_t *const p_clock_source, cgc_divider_cfg_t *const p_divider_cfg) |
| fsp_err_t | R_CGC_OscStopDetectEnable (cgc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_CGC_OscStopDetectDisable (cgc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_CGC_OscStopStatusClear (cgc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_CGC_Close (cgc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_CGC_VersionGet (fsp_version_t *version) |

### Detailed Description

Driver for the CGC peripheral on RA MCUs. This module implements the CGC Interface.

*Note*

***This module is not required for the initial clock configuration. Initial clock settings are configurable on the **Clocks** tab of the RA Configuration editor. The initial clock settings are applied by the BSP during the startup process before main.***

# Overview

### Features

The CGC module supports runtime modifications of clock settings. Key features include the following:

- Supports changing the system clock source to any of the following options (provided they are supported on the MCU):
  - High-speed on-chip oscillator (HOCO)
  - Middle-speed on-chip oscillator (MOCO)
  - Low-speed on-chip oscillator (LOCO)
  - Main oscillator (external resonator or external clock input frequency)
  - Sub-clock oscillator (external resonator)
  - PLL (not available on all MCUs)
- When the system core clock frequency changes, the following things are updated:
  - The CMSIS standard global variable SystemCoreClock is updated to reflect the new clock frequency.
  - Wait states for ROM and RAM are adjusted to the minimum supported value for the new clock frequency.
  - The operating power control mode is updated to the minimum supported value for the new clock settings.
- Supports starting or stopping any of the system clock sources
- Supports changing dividers for the internal clocks
- Supports the oscillation stop detection feature

### Internal Clocks

The RA microcontrollers have up to seven internal clocks. Not all internal clocks exist on all MCUs. Each clock domain has its own divider that can be updated in R_CGC_SystemClockSet(). The dividers are subject to constraints described in the footnote of the table "Specifications of the Clock Generation Circuit for the internal clocks" in the hardware manual.

The internal clocks include:

- System clock (ICLK): core clock used for CPU, flash, internal SRAM, DTC, and DMAC
- PCLKA/PCLKB/PCLKC/PCLKD: Peripheral clocks, refer to the table "Specifications of the Clock Generation Circuit for the internal clocks" in the hardware manual to see which peripherals are controlled by which clocks.
- FCLK: Clock source for reading data flash and for programming/erasure of both code and data flash.
- BCLK: External bus clock

# Configuration

*Note*

*The initial clock settings are configurable on the **Clocks** tab of the RA Configuration editor.*
*There is a configuration to enable the HOCO on reset in the OFS1 settings on the BSP tab.*
*The following clock related settings are configurable in the RA Common section on the BSP tab:*

- *Main Oscillator Wait Time*
- *Main Oscillator Clock Source (external oscillator or crystal/resonator)*
- *Subclock Populated*
- *Subclock Drive*
- *Subclock Stabilization Time (ms)*

*The default stabilization times are determined based on development boards provided by Renesas, but are generally valid for most designs. Depending on the target board hardware configuration and requirements these values may need to be adjusted for reliability or startup speed.*

### Build Time Configurations for r_cgc

The following build time configurations are defined in fsp_cfg/r_cgc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > System > CGC Driver on r_cgc

This module can be added to the Stacks tab via New Stack > Driver > System > CGC Driver on r_cgc:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_cgc0 | Module name. |
| NMI Callback | Name must be a valid C symbol | NULL | A user callback function must be provided if oscillation |

stop detection is used. If this callback function is provided, it is called from the NMI handler if the main oscillator stops.

### Clock Configuration

This module is used to configure the system clocks. There are no module specific clock configurations required to use it.

### Pin Configuration

The CGC module controls the output of the CLOCKOUT signal.

If an external oscillator is used the XTAL and EXTAL pins must be configured accordingly. When running from an on chip oscillator there is no requirement for the main clock external oscillator. In this case, the XTAL and EXTAL pins can be set to a different function in the RA Configuration editor.

The functionality of the subclock external oscillator pins XCIN and XCOUT is fixed.

## Usage Notes

### NMI Interrupt

The CGC timer uses the NMI for oscillation stop detection of the main oscillator after R_CGC_OscStopDetectEnable is called. The NMI is enabled by default. No special configuration is required. When the NMI is triggered, the callback function registered during R_CGC_Open() is called.

### Starting or Stopping the Subclock

If the Subclock Populated property is set to Populated on the BSP configuration tab, then the subclock is started in the BSP startup routine. Otherwise, it is stopped in the BSP startup routine. Starting and stopping the subclock at runtime is not recommended since the stabilization requirements typically negate the negligible power savings.

The application is responsible for ensuring required clocks are started and stable before accessing MCU peripheral registers.

Warning

The subclock can take up to several seconds to stabilize. RA startup code does not wait for subclock stabilization unless the subclock is the main clock source. In this case the default wait time is 1000ms (1 second). When running AGT or RTC off the subclock, the application must ensure the subclock is stable before starting operation. Because there is no hardware stabilization status bit for the subclock R_CGC_ClockCheck cannot be used to optimize this wait.

Changing the subclock state during R_CGC_ClocksCfg() is not supported.

### Low Power Operation

If "Use Low Voltage Mode" is enabled in the BSP MCU specific properties (not available on all MCUs), the MCU is always in low voltage mode and no other power modes are considered. The following conditions must be met for the MCU to run in low voltage mode:

- Requires HOCO to be running, so HOCO cannot be stopped in low voltage mode
- Requires PLL to be stopped, so PLL APIs are not available in low voltage mode
- Requires ICLK <= 4 MHz
- If oscillation stop detection is used, dividers of 1 or 2 cannot be used for any clock

If "Use Low Voltage Mode" is not enabled, the MCU applies the lowest power mode by searching through the following list in order and applying the first power mode that is supported under the current conditions:

- Subosc-speed mode (lowest power)
  - Requires system clock to be LOCO or subclock
  - Requires MOCO, HOCO, main oscillator, and PLL (if present) to be stopped
  - Requires ICLK and FCLK dividers to be 1
- Low-speed mode
  - Requires PLL to be stopped
  - Requires ICLK <= 1 MHz
  - If oscillation stop detection is used, dividers of 1, 2, 4, or 8 cannot be used for any clock
- Middle-speed mode (not supported on all MCUs)
  - Requires ICLK <= 8 MHz
- High-speed mode
  - Default mode if no other operating mode is supported

Refer to the section "Function for Lower Operating Power Consumption" in the "Low Power Modes" chapter of the hardware manual for MCU specific information about operating power control modes.

When low voltage mode is not used, the following functions adjust the operating power control mode to ensure it remains within the hardware specification and to ensure the MCU is running at the optimal operating power control mode:

- R_CGC_ClockStart()
- R_CGC_ClockStop()
- R_CGC_SystemClockSet()
- R_CGC_OscStopDetectEnable()
- R_CGC_OscStopDetectDisable()

*Note*

*FSP APIs, including these APIs, are not thread safe. These APIs and any other user code that modifies the operating power control mode must not be allowed to interrupt each other. Proper care must be taken during application design if these APIs are used in threads or interrupts to ensure this constraint is met.*

No action is required by the user of these APIs. This section is provided for informational purposes only.

# Examples

### Basic Example

This is a basic example of minimal use of the CGC in an application.

```
void cgc_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
```

```
 /* Initializes the CGC module. */
    err = R_CGC_Open(&g_cgc0_ctrl, &g_cgc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Change the system clock to LOCO for power saving. */
 /* Start the LOCO. */
    err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_LOCO, NULL);
    handle_error(err);
 /* Wait for the LOCO stabilization wait time.
  *
  * NOTE: The MOCO, LOCO and subclock do not have stabilization status bits, so any
stabilization time must be
  * performed via a software wait when starting these oscillators. For all other
oscillators, R_CGC_ClockCheck can
  * be used to verify stabilization status.
  */
 R_BSP_SoftwareDelay(BSP_FEATURE_CGC_LOCO_STABILIZATION_MAX_US,
BSP_DELAY_UNITS_MICROSECONDS);
 /* Set divisors. Divisors for clocks that don't exist on the MCU are ignored. */
 cgc_divider_cfg_t dividers =
    {
 /* PCLKB is not used in this application, so select the maximum divisor for lowest
power. */
       .pclkb_div = CGC_SYS_CLOCK_DIV_64,
 /* PCLKD is not used in this application, so select the maximum divisor for lowest
power. */
       .pclkd_div = CGC_SYS_CLOCK_DIV_64,
 /* ICLK is the MCU clock, allow it to run as fast as the LOCO is capable. */
       .iclk_div = CGC_SYS_CLOCK_DIV_1,
 /* These clocks do not exist on some devices. If any clocks don't exist, set the
divider to 1. */
       .pclka_div = CGC_SYS_CLOCK_DIV_1,
       .pclkc_div = CGC_SYS_CLOCK_DIV_1,
       .fclk_div = CGC_SYS_CLOCK_DIV_1,
```

```
        .bclk_div = CGC_SYS_CLOCK_DIV_1,
    };
 /* Switch the system clock to LOCO. */
    err = R_CGC_SystemClockSet(&g_cgc0_ctrl, CGC_CLOCK_LOCO, &dividers);
    handle_error(err);
}
```

### Configuring Multiple Clocks

This example demonstrates switching to a new source clock and stopping the previous source clock in a single function call using R_CGC_ClocksCfg().

```
void cgc_clocks_cfg_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the CGC module. */
    err = R_CGC_Open(&g_cgc0_ctrl, &g_cgc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Change the system clock to PLL running from the main oscillator. */
 /* Assuming the system clock is MOCO, switch to HOCO. */
 cgc_clocks_cfg_t clocks_cfg;
    clocks_cfg.system_clock          = CGC_CLOCK_PLL;
    clocks_cfg.pll_state             = CGC_CLOCK_CHANGE_NONE;
    clocks_cfg.pll_cfg.source_clock = CGC_CLOCK_MAIN_OSC; // unused
    clocks_cfg.pll_cfg.multiplier    = CGC_PLL_MUL_10_0;   // unused
    clocks_cfg.pll_cfg.divider       = CGC_PLL_DIV_2;      // unused
    clocks_cfg.divider_cfg.iclk_div = CGC_SYS_CLOCK_DIV_1;
    clocks_cfg.divider_cfg.pclka_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.pclkb_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.pclkc_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.pclkd_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.bclk_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.fclk_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.mainosc_state         = CGC_CLOCK_CHANGE_NONE;
```

```
    clocks_cfg.hoco_state            = CGC_CLOCK_CHANGE_START;
    clocks_cfg.moco_state            = CGC_CLOCK_CHANGE_STOP;
    clocks_cfg.loco_state            = CGC_CLOCK_CHANGE_NONE;
    err = R_CGC_ClocksCfg(&g_cgc0_ctrl, &clocks_cfg);
    handle_error(err);
#if BSP_FEATURE_CGC_HAS_PLL
 /* Assuming the system clock is HOCO, switch to PLL running from main oscillator and
stop MOCO. */
    clocks_cfg.system_clock          = CGC_CLOCK_PLL;
    clocks_cfg.pll_state             = CGC_CLOCK_CHANGE_START;
    clocks_cfg.pll_cfg.source_clock = CGC_CLOCK_MAIN_OSC;
    clocks_cfg.pll_cfg.multiplier    = (cgc_pll_mul_t) BSP_CFG_PLL_MUL;
    clocks_cfg.pll_cfg.divider       = (cgc_pll_div_t) BSP_CFG_PLL_DIV;
    clocks_cfg.divider_cfg.iclk_div = CGC_SYS_CLOCK_DIV_1;
    clocks_cfg.divider_cfg.pclka_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.pclkb_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.pclkc_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.pclkd_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.bclk_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.fclk_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.mainosc_state         = CGC_CLOCK_CHANGE_START;
    clocks_cfg.hoco_state            = CGC_CLOCK_CHANGE_STOP;
    clocks_cfg.moco_state            = CGC_CLOCK_CHANGE_NONE;
    clocks_cfg.loco_state            = CGC_CLOCK_CHANGE_NONE;
    err = R_CGC_ClocksCfg(&g_cgc0_ctrl, &clocks_cfg);
    handle_error(err);
#endif
}
```

### Oscillation Stop Detection

This example demonstrates registering a callback for oscillation stop detection of the main oscillator.

```
/* Example callback called when oscillation stop is detected. */
void oscillation_stop_callback (cgc_callback_args_t * p_args)
```

```
{
 FSP_PARAMETER_NOT_USED(p_args);
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) If the MCU was running on the main oscillator, the MCU is now running
on MOCO. Switch clocks if
  * desired. This example shows switching to HOCO. */
    err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_HOCO, NULL);
    handle_error(err);
 do
    {
 /* Wait for HOCO to stabilize. */
       err = R_CGC_ClockCheck(&g_cgc0_ctrl, CGC_CLOCK_HOCO);
    } while (FSP_SUCCESS != err);
 cgc_divider_cfg_t dividers =
    {
       .pclkb_div = CGC_SYS_CLOCK_DIV_4,
       .pclkd_div = CGC_SYS_CLOCK_DIV_4,
       .iclk_div = CGC_SYS_CLOCK_DIV_1,
       .pclka_div = CGC_SYS_CLOCK_DIV_4,
       .pclkc_div = CGC_SYS_CLOCK_DIV_4,
       .fclk_div = CGC_SYS_CLOCK_DIV_4,
       .bclk_div = CGC_SYS_CLOCK_DIV_4,
    };
    err = R_CGC_SystemClockSet(&g_cgc0_ctrl, CGC_CLOCK_HOCO, &dividers);
    handle_error(err);
#if BSP_FEATURE_CGC_HAS_PLL
 /* (Optional) If the MCU was running on the PLL, the PLL is now in free-running
mode. Switch clocks if
  * desired. This example shows switching to the PLL running on HOCO. */
    err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_HOCO, NULL);
    handle_error(err);
 do
    {
 /* Wait for HOCO to stabilize. */
```

```
        err = R_CGC_ClockCheck(&g_cgc0_ctrl, CGC_CLOCK_HOCO);
    } while (FSP_SUCCESS != err);
 cgc_pll_cfg_t pll_cfg =
    {
        .source_clock = CGC_CLOCK_HOCO,
        .multiplier   = (cgc_pll_mul_t) BSP_CFG_PLL_MUL,
        .divider      = (cgc_pll_div_t) BSP_CFG_PLL_DIV,
    };
    err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_PLL, &pll_cfg);
    handle_error(err);
 do
    {
 /* Wait for PLL to stabilize. */
        err = R_CGC_ClockCheck(&g_cgc0_ctrl, CGC_CLOCK_PLL);
    } while (FSP_SUCCESS != err);
 cgc_divider_cfg_t pll_dividers =
    {
        .pclkb_div = CGC_SYS_CLOCK_DIV_4,
        .pclkd_div = CGC_SYS_CLOCK_DIV_4,
        .iclk_div = CGC_SYS_CLOCK_DIV_1,
        .pclka_div = CGC_SYS_CLOCK_DIV_4,
        .pclkc_div = CGC_SYS_CLOCK_DIV_4,
        .fclk_div = CGC_SYS_CLOCK_DIV_4,
        .bclk_div = CGC_SYS_CLOCK_DIV_4,
    };
    err = R_CGC_SystemClockSet(&g_cgc0_ctrl, CGC_CLOCK_PLL, &pll_dividers);
    handle_error(err);
#endif
 /* (Optional) Clear the error flag. Only clear this flag after switching the MCU
clock source away from the main
  * oscillator and if the main oscillator is stable again. */
    err = R_CGC_OscStopStatusClear(&g_cgc0_ctrl);
    handle_error(err);
}
```

```
void cgc_osc_stop_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the module. */
    err = R_CGC_Open(&g_cgc0_ctrl, &g_cgc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Enable oscillation stop detection. The main oscillator must be running at this
point. */
    err = R_CGC_OscStopDetectEnable(&g_cgc0_ctrl);
    handle_error(err);
 /* (Optional) Oscillation stop detection must be disabled before entering any low
power mode. */
    err = R_CGC_OscStopDetectDisable(&g_cgc0_ctrl);
    handle_error(err);
    __WFI();
 /* (Optional) Reenable oscillation stop detection after waking from low power mode.
*/
    err = R_CGC_OscStopDetectEnable(&g_cgc0_ctrl);
    handle_error(err);
}
```

## Data Structures

| | |
|---|---|
| struct | cgc_instance_ctrl_t |

## Data Structure Documentation

### ◆ cgc_instance_ctrl_t

| struct cgc_instance_ctrl_t |
|---|
| CGC private control block. DO NOT MODIFY. Initialization occurs when R_CGC_Open() is called. |

## Function Documentation

### ◆ R_CGC_Open()

fsp_err_t R_CGC_Open ( cgc_ctrl_t *const *p_ctrl*, cgc_cfg_t const *const *p_cfg* )

Initialize the CGC API. Implements cgc_api_t::open.

Example:

```
/* Initializes the CGC module. */
   err = R_CGC_Open(&g_cgc0_ctrl, &g_cgc0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CGC successfully initialized. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |

### ◆ R_CGC_ClocksCfg()

fsp_err_t R_CGC_ClocksCfg ( cgc_ctrl_t *const *p_ctrl*, cgc_clocks_cfg_t const *const *p_clock_cfg* )

Reconfigures all main system clocks. This API can be used for any of the following purposes:

- start or stop clocks
- change the system clock source
- configure the PLL multiplication and division ratios when starting the PLL
- change the system dividers

If the requested system clock source has a stabilization flag, this function blocks waiting for the stabilization flag of the requested system clock source to be set. If the requested system clock source was just started and it has no stabilization flag, this function blocks for the stabilization time required by the requested system clock source according to the Electrical Characteristics section of the hardware manual. If the requested system clock source has no stabilization flag and it is already running, it is assumed to be stable and this function will not block. If the requested system clock is the subclock, the subclock must be stable prior to calling this function.

The internal dividers (cgc_clocks_cfg_t::divider_cfg) are subject to constraints described in footnotes of the hardware manual table detailing specifications for the clock generation circuit for the internal clocks for the MCU. For example:

- RA6M3: see footnotes of Table 9.2 "Specifications of the clock generation circuit for the internal clocks" in the RA6M3 manual R01UH0886EJ0100
- RA2A1: see footnotes of Table 9.2 "Clock generation circuit specifications for the internal clocks" in the RA2A1 manual R01UH0888EJ0100

Do not attempt to stop the requested clock source or the source of the PLL if the PLL will be running after this operation completes.

Implements cgc_api_t::clocksCfg.

Example:

```
/* Assuming the system clock is MOCO, switch to HOCO. */
```

```
cgc_clocks_cfg_t clocks_cfg;
   clocks_cfg.system_clock          = CGC_CLOCK_PLL;
   clocks_cfg.pll_state             = CGC_CLOCK_CHANGE_NONE;
   clocks_cfg.pll_cfg.source_clock = CGC_CLOCK_MAIN_OSC; // unused
   clocks_cfg.pll_cfg.multiplier    = CGC_PLL_MUL_10_0;   // unused
   clocks_cfg.pll_cfg.divider       = CGC_PLL_DIV_2;      // unused
   clocks_cfg.divider_cfg.iclk_div = CGC_SYS_CLOCK_DIV_1;
   clocks_cfg.divider_cfg.pclka_div = CGC_SYS_CLOCK_DIV_4;
   clocks_cfg.divider_cfg.pclkb_div = CGC_SYS_CLOCK_DIV_4;
   clocks_cfg.divider_cfg.pclkc_div = CGC_SYS_CLOCK_DIV_4;
   clocks_cfg.divider_cfg.pclkd_div = CGC_SYS_CLOCK_DIV_4;
   clocks_cfg.divider_cfg.bclk_div = CGC_SYS_CLOCK_DIV_4;
   clocks_cfg.divider_cfg.fclk_div = CGC_SYS_CLOCK_DIV_4;
   clocks_cfg.mainosc_state         = CGC_CLOCK_CHANGE_NONE;
   clocks_cfg.hoco_state            = CGC_CLOCK_CHANGE_START;
   clocks_cfg.moco_state            = CGC_CLOCK_CHANGE_STOP;
   clocks_cfg.loco_state            = CGC_CLOCK_CHANGE_NONE;
   err = R_CGC_ClocksCfg(&g_cgc0_ctrl, &clocks_cfg);
   handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Clock configuration applied successfully. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_IN_USE | Attempt to stop the current system clock or the PLL source clock. |
| FSP_ERR_CLOCK_ACTIVE | PLL configuration cannot be changed while PLL is running. |
| FSP_ERR_OSC_STOP_DET_ENABLED | PLL multiplier must be less than 20 if oscillation stop detect is enabled and the input frequency is less than 12.5 MHz. |
| FSP_ERR_NOT_STABILIZED | PLL clock source is not stable. |
| FSP_ERR_PLL_SRC_INACTIVE | PLL clock source is not running. |

### ◆ R_CGC_ClockStart()

fsp_err_t R_CGC_ClockStart ( cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *clock_source*, cgc_pll_cfg_t const *const *p_pll_cfg* )

Start the specified clock if it is not currently active. The PLL configuration cannot be changed while the PLL is running. Implements cgc_api_t::clockStart.

The PLL source clock must be operating and stable prior to starting the PLL.

Example:

```
/* Start the LOCO. */

   err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_LOCO, NULL);

   handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Clock initialized successfully. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_STABILIZED | The clock source is not stabilized after being turned off or PLL clock source is not stable. |
| FSP_ERR_PLL_SRC_INACTIVE | PLL clock source is not running. |
| FSP_ERR_CLOCK_ACTIVE | PLL configuration cannot be changed while PLL is running. |
| FSP_ERR_OSC_STOP_DET_ENABLED | PLL multiplier must be less than 20 if oscillation stop detect is enabled and the input frequency is less than 12.5 MHz. |

### ◆ R_CGC_ClockStop()

fsp_err_t R_CGC_ClockStop ( cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *clock_source* )

Stop the specified clock if it is active. Implements cgc_api_t::clockStop.

Do not attempt to stop the current system clock source. Do not attempt to stop the source clock of the PLL if the PLL is running.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Clock stopped successfully. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_IN_USE | Attempt to stop the current system clock or the PLL source clock. |
| FSP_ERR_OSC_STOP_DET_ENABLED | Attempt to stop MOCO when Oscillation stop is enabled. |
| FSP_ERR_NOT_STABILIZED | Clock not stabilized after starting. |

### ◆ R_CGC_ClockCheck()

fsp_err_t R_CGC_ClockCheck ( cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *clock_source* )

Check the specified clock for stability. Implements cgc_api_t::clockCheck.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Clock is running and stable. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_STABILIZED | Clock not stabilized. |
| FSP_ERR_CLOCK_INACTIVE | Clock not turned on. |

### ◆ R_CGC_SystemClockSet()

fsp_err_t R_CGC_SystemClockSet ( cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *clock_source*, cgc_divider_cfg_t const *const *p_divider_cfg* )

Set the specified clock as the system clock and configure the internal dividers for ICLK, PCLKA, PCLKB, PCLKC, PCLKD, BCLK, and FCLK. Implements cgc_api_t::systemClockSet.

The requested clock source must be running and stable prior to calling this function. The internal dividers are subject to constraints described in the hardware manual table "Specifications of the Clock Generation Circuit for the internal clocks".

The internal dividers (p_divider_cfg) are subject to constraints described in footnotes of the hardware manual table detailing specifications for the clock generation circuit for the internal clocks for the MCU. For example:

- RA6M3: see footnotes of Table 9.2 "Specifications of the clock generation circuit for the internal clocks" in the RA6M3 manual R01UH0886EJ0100
- RA2A1: see footnotes of Table 9.2 "Clock generation circuit specifications for the internal clocks" in the RA2A1 manual R01UH0888EJ0100

This function also updates the RAM and ROM wait states, the operating power control mode, and the SystemCoreClock CMSIS global variable.

Example:

```
/* Set divisors. Divisors for clocks that don't exist on the MCU are ignored. */
cgc_divider_cfg_t dividers =
   {
/* PCLKB is not used in this application, so select the maximum divisor for lowest
power. */
      .pclkb_div = CGC_SYS_CLOCK_DIV_64,
/* PCLKD is not used in this application, so select the maximum divisor for lowest
power. */
      .pclkd_div = CGC_SYS_CLOCK_DIV_64,
/* ICLK is the MCU clock, allow it to run as fast as the LOCO is capable. */
      .iclk_div = CGC_SYS_CLOCK_DIV_1,
/* These clocks do not exist on some devices. If any clocks don't exist, set the
divider to 1. */
      .pclka_div = CGC_SYS_CLOCK_DIV_1,
      .pclkc_div = CGC_SYS_CLOCK_DIV_1,
      .fclk_div = CGC_SYS_CLOCK_DIV_1,
      .bclk_div = CGC_SYS_CLOCK_DIV_1,
   };
/* Switch the system clock to LOCO. */
```

```
err = R_CGC_SystemClockSet(&g_cgc0_ctrl, CGC_CLOCK_LOCO, &dividers);
handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation performed successfully. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CLOCK_INACTIVE | The specified clock source is inactive. |
| FSP_ERR_NOT_STABILIZED | The clock source has not stabilized |

### ◆ R_CGC_SystemClockGet()

fsp_err_t R_CGC_SystemClockGet ( cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *const *p_clock_source*, cgc_divider_cfg_t *const *p_divider_cfg* )

Return the current system clock source and configuration. Implements cgc_api_t::systemClockGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Parameters returned successfully. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ R_CGC_OscStopDetectEnable()

fsp_err_t R_CGC_OscStopDetectEnable ( cgc_ctrl_t *const *p_ctrl*)

Enable the oscillation stop detection for the main clock. Implements cgc_api_t::oscStopDetectEnable.

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.

Example:

```
/* Enable oscillation stop detection. The main oscillator must be running at this point. */
    err = R_CGC_OscStopDetectEnable(&g_cgc0_ctrl);
    handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation performed successfully. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_LOW_VOLTAGE_MODE | Settings not allowed in low voltage mode. |

### ◆ R_CGC_OscStopDetectDisable()

fsp_err_t R_CGC_OscStopDetectDisable ( cgc_ctrl_t *const *p_ctrl*)

Disable the oscillation stop detection for the main clock. Implements cgc_api_t::oscStopDetectDisable.

Example:

```
 /* (Optional) Oscillation stop detection must be disabled before entering any low
power mode. */
    err = R_CGC_OscStopDetectDisable(&g_cgc0_ctrl);
    handle_error(err);
    __WFI();
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation performed successfully. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_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. |

### ◆ R_CGC_OscStopStatusClear()

fsp_err_t R_CGC_OscStopStatusClear ( cgc_ctrl_t *const *p_ctrl*)

Clear the Oscillation Stop Detection Status register. This register is not cleared automatically if the stopped clock is restarted. Implements cgc_api_t::oscStopStatusClear.

After clearing the status, oscillation stop detection is no longer enabled.

This register cannot be cleared while the main oscillator is the system clock or the PLL source clock.

Example:

```
 /* (Optional) Clear the error flag. Only clear this flag after switching the MCU
clock source away from the main
  * oscillator and if the main oscillator is stable again. */
    err = R_CGC_OscStopStatusClear(&g_cgc0_ctrl);
    handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation performed successfully. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CLOCK_INACTIVE | Main oscillator must be running to clear the oscillation stop detection flag. |
| FSP_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. |
| FSP_ERR_INVALID_HW_CONDITION | Oscillation stop status was not cleared. Check preconditions and try again. |

### ◆ R_CGC_Close()

fsp_err_t R_CGC_Close ( cgc_ctrl_t *const *p_ctrl*)

Closes the CGC module. Implements cgc_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The module is successfully closed. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ R_CGC_VersionGet()

| fsp_err_t R_CGC_VersionGet ( fsp_version_t *const *p_version*) |
|---|
| Return the driver version. Implements cgc_api_t::versionGet.<br><br>**Return values**<br>FSP_SUCCESS — Module version provided in p_version.<br>FSP_ERR_ASSERTION — Invalid input argument. |

## 4.2.9 Cyclic Redundancy Check (CRC) Calculator (r_crc)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | R_CRC_Open (crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg) |
| fsp_err_t | R_CRC_Close (crc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_CRC_Calculate (crc_ctrl_t *const p_ctrl, crc_input_t *const p_crc_input, uint32_t *calculatedValue) |
| fsp_err_t | R_CRC_CalculatedValueGet (crc_ctrl_t *const p_ctrl, uint32_t *calculatedValue) |
| fsp_err_t | R_CRC_SnoopEnable (crc_ctrl_t *const p_ctrl, uint32_t crc_seed) |
| fsp_err_t | R_CRC_SnoopDisable (crc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_CRC_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Driver for the CRC peripheral on RA MCUs. This module implements the CRC Interface.

# Overview

The CRC module provides a API 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 industry-standard polynomials.

### Features

- CRC 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 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 module can calculate CRC with LSB first or MSB first bit order.

# Configuration

### Build Time Configurations for r_crc

The following build time configurations are defined in fsp_cfg/r_crc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Monitoring > CRC Driver on r_crc

This module can be added to the Stacks tab via New Stack > Driver > Monitoring > CRC Driver on r_crc:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_crc0 | Module name. |
| CRC Polynomial | • CRC-8<br>• CRC-16<br>• CRC-CCITT<br>• CRC-32<br>• CRC-32C | CRC-32C | Select the CRC polynomial. |
| Bit Order | • LSB<br>• MSB | MSB | Select the CRC bit order. |
| Snoop Address | Refer to the RA Configuration tool for available options. | NONE | Select the SCI register address CRC snoop |

### Clock Configuration

There is no clock configuration for the CRC module.

### Pin Configuration

This module does not use I/O pins.

## Usage Notes

### CRC Snoop

The CRC snoop function monitors reads from and writes to a specified I/O register address and performs CRC calculation on the data read from and written to the register address automatically. Instead of calling R_CRC_Calculate on a block of data, R_CRC_SnoopEnable is called to start monitoring reads/writes and R_CRC_CalculatedValueGet is used to obtain the current CRC.

*Note*

*Snoop mode is available for transmit/receive operations on SCI only.*

### Limitations

When using CRC32 polynomial functions the CRC module produces the same results as popular online CRC32 calculators, but it is important to remember a few important points.

- Online CRC32 calculators allow the input to be any number of bytes. The FSP CRC32 API function uses 32-bit words. This means the online calculations must be 'padded' to end on a 32-bit boundary.
- Online CRC32 calculators usually invert the output prior to presenting it as a result. It is up to the application program to include this step if needed.
- The seed value of 0xFFFFFFFF needs to be used by both the online calculator and the R_CRC module API (CRC32 polynomials)
- Make sure the bit orientation of the R_CRC CRC32 is set for LSB and that you have CRC32 selected and not CRC32C.
- Some online CRC tools XOR the final result with 0xFFFFFFFF.

## Examples

### Basic Example

This is a basic example of minimal use of the CRC module in an application.

```
void crc_example ()
{
    uint32_t length;
    uint32_t uint8_calculated_value;
    length = sizeof(g_data_8bit) / sizeof(g_data_8bit[0]);
 crc_input_t example_input =
    {
       .p_input_buffer = g_data_8bit,
       .num_bytes      = length,
```

```
        .crc_seed       = 0,
    };
 /* Open CRC module with 8 bit polynomial */
 R_CRC_Open(&crc_ctrl, &g_crc_test_cfg);
 /* 8-bit CRC calculation */
 R_CRC_Calculate(&crc_ctrl, &example_input, &uint8_calculated_value);
}
```

### Snoop Example

This example demonstrates CRC snoop operation.

```
void crc_snoop_example ()
{
 /* Open CRC module with 8 bit polynomial */
 R_CRC_Open(&crc_ctrl, &g_crc_test_cfg);
 /* Open SCI Driver */
 /* Configure Snoop address and enable snoop mode */
 R_CRC_SnoopEnable(&crc_ctrl, 0);
 /* Perfrom SCI read/Write operation depending on the SCI snoop address configure */
 /* Read CRC value */
 R_CRC_CalculatedValueGet(&crc_ctrl, &g_crc_buff);
}
```

## Data Structures

| | |
|---|---|
| struct | crc_instance_ctrl_t |

## Data Structure Documentation

### ◆ crc_instance_ctrl_t

| |
|---|
| struct crc_instance_ctrl_t |
| Driver instance control structure. |

## Function Documentation

### ◆ R_CRC_Open()

fsp_err_t R_CRC_Open ( crc_ctrl_t *const *p_ctrl*, crc_cfg_t const *const *p_cfg* )

Open the CRC driver module

Implements crc_api_t::open

Open the CRC driver module and initialize the driver control block according to the passed-in configuration structure.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration was successful. |
| FSP_ERR_ASSERTION | p_ctrl or p_cfg is NULL. |
| FSP_ERR_ALREADY_OPEN | Module already open |

### ◆ R_CRC_Close()

fsp_err_t R_CRC_Close ( crc_ctrl_t *const *p_ctrl*)

Close the CRC module driver.

Implements crc_api_t::close

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration was successful. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The driver is not opened. |

### ◆ R_CRC_Calculate()

fsp_err_t R_CRC_Calculate ( crc_ctrl_t *const *p_ctrl*, crc_input_t *const *p_crc_input*, uint32_t * *calculatedValue* )

Perform a CRC calculation on a block of 8-bit/32-bit (for 32-bit polynomial) data.

Implements crc_api_t::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

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calculation successful. |
| FSP_ERR_ASSERTION | Either p_ctrl, inputBuffer, or calculatedValue is NULL. |
| FSP_ERR_INVALID_ARGUMENT | length value is NULL. |
| FSP_ERR_NOT_OPEN | The driver is not opened. |

### ◆ R_CRC_CalculatedValueGet()

fsp_err_t R_CRC_CalculatedValueGet ( crc_ctrl_t *const *p_ctrl*, uint32_t * *calculatedValue* )

Return the current calculated value.

Implements crc_api_t::crcResultGet

CRC calculation operates on a running value. This function returns the current calculated value.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Return of calculated value successful. |
| FSP_ERR_ASSERTION | Either p_ctrl or calculatedValue is NULL. |
| FSP_ERR_NOT_OPEN | The driver is not opened. |

### ◆ R_CRC_SnoopEnable()

fsp_err_t R_CRC_SnoopEnable ( crc_ctrl_t *const *p_ctrl*, uint32_t *crc_seed* )

Configure the snoop channel and set the CRC seed.

Implements crc_api_t::snoopEnable

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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Snoop configured successfully. |
| FSP_ERR_ASSERTION | Pointer to control stucture is NULL |
| FSP_ERR_NOT_OPEN | The driver is not opened. |

### ◆ R_CRC_SnoopDisable()

fsp_err_t R_CRC_SnoopDisable ( crc_ctrl_t *const *p_ctrl*)

Disable snooping.

Implements crc_api_t::snoopDisable

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Snoop disabled. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The driver is not opened. |

### ◆ R_CRC_VersionGet()

fsp_err_t R_CRC_VersionGet ( fsp_version_t *const *p_version*)

Get the driver version based on compile time macros.

Implements crc_api_t::versionGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | p_version is NULL. |

## 4.2.10 Capacitive Touch Sensing Unit (r_ctsu)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | R_CTSU_Open (ctsu_ctrl_t *const p_ctrl, ctsu_cfg_t const *const p_cfg)<br>Opens and configures the CTSU driver module. Implements ctsu_api_t::open. More... |
| fsp_err_t | R_CTSU_ScanStart (ctsu_ctrl_t *const p_ctrl)<br>This function should be called each time a periodic timer expires. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Before starting the next scan, first get the data with R_CTSU_DataGet(). If a different control block scan should be run, check the scan is complete before executing. Implements ctsu_api_t::scanStart. More... |
| fsp_err_t | R_CTSU_DataGet (ctsu_ctrl_t *const p_ctrl, uint16_t *p_data)<br>This function gets the sensor values as scanned by the CTSU. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Implements ctsu_api_t::dataGet. More... |
| fsp_err_t | R_CTSU_Close (ctsu_ctrl_t *const p_ctrl)<br>Disables specified CTSU control block. Implements ctsu_api_t::close. More... |
| fsp_err_t | R_CTSU_VersionGet (fsp_version_t *const p_version)<br>Return CTSU HAL driver version. Implements ctsu_api_t::versionGet. More... |

### Detailed Description

This HAL driver supports the Capacitive Touch Sensing Unit (CTSU). It implements the CTSU Interface .

# Overview

The capacitive touch sensing unit HAL driver (r_ctsu) provides an API to control the CTSU peripheral. This module performs capacitance measurement based on various settings defined by the

configuration. This module is configured via the QE for Capacitive Touch.

### Features

- Supports both Self-capacitance multi scan mode and Mutual-capacitance full scan mode
- Scans may be started by software or an external trigger
- Returns measured capacitance data on scan completion
- Optional DTC support

# Configuration

*Note*

***This module is configured via the QE for Capacitive Touch. For information on how to use the QE tool, once the tool is installed click Help -> Help Contents in e2 studio and search for "QE".***

### Build Time Configurations for r_ctsu

The following build time configurations are defined in fsp_cfg/r_ctsu_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Support for using DTC | • Enabled<br>• Disabled | Disabled | Enable DTC support for the CTSU module. |
| Interrupt priority level | MCU Specific Options | | Priority level of all CTSU interrupt (CSTU_WR,CTSU_RD,CTSU_FN) |

### Configurations for Driver > CapTouch > CTSU Driver on r_ctsu

This module can be added to the Stacks tab via New Stack > Driver > CapTouch > CTSU Driver on r_ctsu:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Scan Start Trigger | MCU Specific Options | | CTSU Scan Start Trigger Select |

### Interrupt Configuration

The first R_CTSU_Open function call sets CTSU peripheral interrupts. The user should provide a callback function to be invoked at the end of the CTSU scan sequence. The callback argument will contain information about the scan status.

### Clock Configuration

The CTSU peripheral module uses PCLKB as its clock source. You can set the PCLKB frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

*Note*

*The CTSU Drive pulse will be calculated and set by the tooling depending on the selected transfer rate.*

### Pin Configuration

The TSn pins are sensor pins for the CTSU.

The TSCAP pin is used for an internal low-pass filter and must be connected to an external decoupling capacitor.

# Usage Notes

### CTSU

### Self-capacitance multi scan mode

In self-capacitance mode each TS pin is assigned to one touch button. Electrodes of multiple TS pins can be physically aligned to create slider or wheel interfaces.

- Scan Order
  - The hardware scans the specified pins in ascending order.
  - For example, if pins TS05, TS08, TS02, TS03, and TS06 are specified in your application, the hardware will scan them in the order TS02, TS03, TS05, TS06, TS08.
- Element
  - An element refers to the index of a pin within the scan order. Using the previous example, TS05 is element 2.
- Scan Time
  - Scanning is handled directly by the CTSU peripheral and does not utilize any main processor time.
  - It takes approximately 500us to scan a single sensor.
  - If DTC is not used additional overhead is required for the main processor to transfer data to/from registers when each sensor is scanned.

### Mutual-capacitance full scan mode

In mutual-capacitance mode each TS pin acts as either a 'row' or 'column' in an array of sensors. As a result, this mode uses fewer pins when more than five sensors are configured. Mutual-capacitance mode is ideal for applications where many touch sensors are required, like keypads, button matrices and touchpads.

As an example, consider a standard phone keypad comprised of a matrix of four rows and three columns.

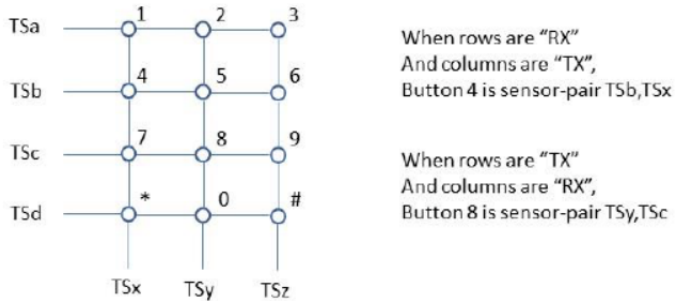



Figure 113: Mutual Button Image

In mutual capacitance mode only 7 pins are necessary to scan 12 buttons. In self mode, 12 pins would be required.

- Scan Order
  - The hardware scans the matrix by iterating over the TX pins first and the RX pins second.
  - For example, if pins TS10, TS11, and TS03 are specified as RX sensors and pins TS02, TS07, and TS04 are specified as TX sensors, the hardware will scan them in the following sensor-pair order:
    TS03-TS02, TS03-TS04, TS03-TS07, TS10-TS02, TS10-TS04, TS10-TS07, TS11-TS02, TS11-TS04, TS11-TS07
- Element
  - An element refers to the index of a sensor-pair within the scan order. Using the previous example, TS10-TS07 is element 5.
- Scan Time
  - Because mutual-capacitance scans two patterns for one element it takes twice as long as self-capacitance (1ms vs 0.5ms per element).

# Examples

### Basic Example

This is a basic example of minimal use of the CTSU in an application.

```
volatile bool g_scan_flag = false;
void ctsu_callback (ctsu_callback_args_t * p_args)
{
 if (CTSU_EVENT_SCAN_COMPLETE == p_args->event)
    {
       g_scan_flag = true;
    }
}
void ctsu_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint16_t data[CTSU_CFG_NUM_SELF_ELEMENTS];
    err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (true)
    {
       err = R_CTSU_ScanStart(&g_ctsu_ctrl);
      handle_error(err);
```

```
 while (!g_scan_flag)
      {
 /* Wait for scan end callback */
      }
       g_scan_flag = false;
       err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
 if (FSP_SUCCESS == err)
      {
 /* Application specific data processing. */
      }
    }
}
```

### Multi-configuration Example

This is a optional example of using both Self-capacitance and Mutual-capacitance configurations in the same project.

```
void ctsu_optional_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint16_t data[CTSU_CFG_NUM_SELF_ELEMENTS + (CTSU_CFG_NUM_MUTUAL_ELEMENTS * 2)];
    err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);
    handle_error(err);
    err = R_CTSU_Open(&g_ctsu_ctrl_mutual, &g_ctsu_cfg_mutual);
    handle_error(err);
 while (true)
    {
 R_CTSU_ScanStart(&g_ctsu_ctrl);
 while (!g_scan_flag)
      {
 /* Wait for scan end callback */
      }
       g_scan_flag = false;
 R_CTSU_ScanStart(&g_ctsu_ctrl_mutual);
```

```
while (!g_scan_flag)
     {
/* Wait for scan end callback */
     }
      g_scan_flag = false;
      err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
     handle_error(err);
if (FSP_SUCCESS == err)
     {
/* Application specific data processing. */
     }
      err = R_CTSU_DataGet(&g_ctsu_ctrl_mutual, data);
     handle_error(err);
if (FSP_SUCCESS == err)
     {
/* Application specific data processing. */
     }
   }
}
```

## Data Structures

| | |
|---|---|
| struct | ctsu_ctsuwr_t |
| struct | ctsu_self_buf_t |
| struct | ctsu_mutual_buf_t |
| struct | ctsu_correction_info_t |
| struct | ctsu_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | ctsu_state_t |
| enum | ctsu_tuning_t |
| enum | ctsu_correction_status_t |
| enum | ctsu_range_t |

## Data Structure Documentation

### ◆ ctsu_ctsuwr_t

| struct ctsu_ctsuwr_t | | |
|---|---|---|
| CTSUWR write register value | | |
| Data Fields | | |
| uint16_t | ctsussc | Copy from (ssdiv << 8) by Open API. |
| uint16_t | ctsuso0 | Copy from ((snum << 10) \| so) by Open API. |
| uint16_t | ctsuso1 | Copy from (sdpa << 8) by Open API. ICOG and RICOA is set recommend value. |

### ◆ ctsu_self_buf_t

| struct ctsu_self_buf_t | | |
|---|---|---|
| Scan buffer data formats (Self) | | |
| Data Fields | | |
| uint16_t | sen | Sensor counter data. |
| uint16_t | ref | Reference counter data (Not used) |

### ◆ ctsu_mutual_buf_t

| struct ctsu_mutual_buf_t | | |
|---|---|---|
| Scan buffer data formats (Mutual) | | |
| Data Fields | | |
| uint16_t | pri_sen | Primary sensor data. |
| uint16_t | pri_ref | Primary reference data (Not used) |
| uint16_t | snd_sen | Secondary sensor data. |
| uint16_t | snd_ref | Secondary reference data (Not used) |

### ◆ ctsu_correction_info_t

| struct ctsu_correction_info_t | | |
|---|---|---|
| Correction information | | |
| Data Fields | | |
| ctsu_correction_status_t | status | Correction status. |
| ctsu_ctsuwr_t | ctsuwr | Correction scan parameter. |

| | | |
|---|---|---|
| volatile ctsu_self_buf_t | scanbuf | Correction scan buffer. |
| uint16_t | first_val | 1st correction value |
| uint16_t | second_val | 2nd correction value |
| uint32_t | first_coefficient | 1st correction coefficient |
| uint32_t | second_coefficient | 2nd correction coefficient |
| uint32_t | ctsu_clock | CTSU clock [MHz]. |

### ◆ ctsu_instance_ctrl_t

| struct ctsu_instance_ctrl_t | |
|---|---|
| CTSU private control block. DO NOT MODIFY. Initialization occurs when R_CTSU_Open() is called. | |
| **Data Fields** | |
| uint32_t | open |
| | Whether or not driver is open. |
| | |
| ctsu_state_t | state |
| | CTSU run state. |
| | |
| ctsu_tuning_t | tuning |
| | CTSU Initial offset tuning status. |
| | |
| uint16_t | num_elements |
| | Number of elements to scan. |
| | |
| uint16_t | wr_index |
| | Word index into ctsuwr register array. |
| | |
| uint16_t | rd_index |
| | Word index into scan data buffer. |
| | |
| uint8_t * | p_tuning_complete |
| | Pointer to tuning completion flag of each element. |

| | |
|---|---|
| | g_ctsu_tuning_complete[] is set by Open API. |
| | |
| int32_t * | p_tuning_diff |
| | Pointer to difference from base value of each element. g_ctsu_tuning_diff[] is set by Open API. |
| | |
| uint16_t | average |
| | CTSU Moving average counter. |
| | |
| uint16_t | num_moving_average |
| | Copy from config by Open API. |
| | |
| uint8_t | ctsucr1 |
| | Copy from (atune1 << 3, md << 6) by Open API. CLK, ATUNE0, CSW, and PON is set by HAL driver. |
| | |
| ctsu_ctsuwr_t * | p_ctsuwr |
| | CTSUWR write register value. g_ctsu_ctsuwr[] is set by Open API. |
| | |
| ctsu_self_buf_t * | p_self_raw |
| | Pointer to Self raw data. g_ctsu_self_raw[] is set by Open API. |
| | |
| uint16_t * | p_self_work |
| | Pointer to Self work buffer. g_ctsu_self_work[] is set by Open API. |
| | |
| uint16_t * | p_self_data |
| | Pointer to Self moving average data. g_ctsu_self_data[] is set by Open API. |
| | |
| | |

| | |
|---|---|
| ctsu_mutual_buf_t * | p_mutual_raw |
| | Pointer to Mutual raw data. g_ctsu_mutual_raw[] is set by Open API. |
| | |
| uint16_t * | p_mutual_pri_work |
| | Pointer to Mutual primary work buffer. g_ctsu_mutual_pri_work[] is set by Open API. |
| | |
| uint16_t * | p_mutual_snd_work |
| | Pointer to Mutual secondary work buffer. g_ctsu_mutual_snd_work[] is set by Open API. |
| | |
| uint16_t * | p_mutual_pri_data |
| | Pointer to Mutual primary moving average data. g_ctsu_mutual_pri_data[] is set by Open API. |
| | |
| uint16_t * | p_mutual_snd_data |
| | Pointer to Mutual secondary moving average data. g_ctsu_mutual_snd_data[] is set by Open API. |
| | |
| ctsu_correction_info_t * | p_correction_info |
| | Pointer to correction info. |
| | |
| ctsu_cfg_t const * | p_ctsu_cfg |
| | Pointer to initial configurations. |
| | |
| IRQn_Type | write_irq |
| | Copy from config by Open API. CTSU_CTSUWR interrupt vector. |
| | |
| IRQn_Type | read_irq |
| | Copy from config by Open API. CTSU_CTSURD interrupt vector. |

| | |
|---|---|
| | |
| IRQn_Type | end_irq |
| | Copy from config by Open API. CTSU_CTSUFN interrupt vector. |
| | |
| void const * | p_context |
| | Placeholder for user data. |
| | |
| void(* | p_callback )(ctsu_callback_args_t *p_args) |
| | Callback provided when a CTSUFN occurs. |
| | |

## Enumeration Type Documentation

### ◆ ctsu_state_t

| enum ctsu_state_t | |
|---|---|
| CTSU run state | |
| Enumerator | |
| CTSU_STATE_INIT | Not open. |
| CTSU_STATE_IDLE | Opened. |
| CTSU_STATE_SCANNING | Scanning now. |
| CTSU_STATE_SCANNED | Scan end. |

### ◆ ctsu_tuning_t

| enum ctsu_tuning_t | |
|---|---|
| CTSU Initial offset tuning status | |
| Enumerator | |
| CTSU_TUNING_INCOMPLETE | Initial offset tuning incomplete. |
| CTSU_TUNING_COMPLETE | Initial offset tuning complete. |

### ◆ ctsu_correction_status_t

| enum ctsu_correction_status_t | |
|---|---|
| CTSU Correction status | |
| Enumerator | |
| CTSU_CORRECTION_INIT | Correction initial status. |
| CTSU_CORRECTION_RUN | Correction scan running. |
| CTSU_CORRECTION_COMPLETE | Correction complete. |
| CTSU_CORRECTION_ERROR | Correction error. |

### ◆ ctsu_range_t

| enum ctsu_range_t | |
|---|---|
| CTSU range definition | |
| Enumerator | |
| CTSU_RANGE_20UA | 20uA mode |
| CTSU_RANGE_40UA | 40uA mode |
| CTSU_RANGE_80UA | 80uA mode |
| CTSU_RANGE_160UA | 160uA mode |
| CTSU_RANGE_NUM | number of range |

## Function Documentation

### ◆ R_CTSU_Open()

fsp_err_t R_CTSU_Open ( ctsu_ctrl_t *const *p_ctrl*, ctsu_cfg_t const *const *p_cfg* )

Opens and configures the CTSU driver module. Implements ctsu_api_t::open.

Example:

```
err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CTSU successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Configuration parameter error. |

***Note***

*In the first Open, measurement for correction works, and it takes several tens of milliseconds.*

### ◆ R_CTSU_ScanStart()

fsp_err_t R_CTSU_ScanStart ( ctsu_ctrl_t *const *p_ctrl*)

This function should be called each time a periodic timer expires. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Before starting the next scan, first get the data with R_CTSU_DataGet(). If a different control block scan should be run, check the scan is complete before executing. Implements ctsu_api_t::scanStart.

Example:

```
while (true)
   {
      err = R_CTSU_ScanStart(&g_ctsu_ctrl);
     handle_error(err);
while (!g_scan_flag)
     {
/* Wait for scan end callback */
     }
      g_scan_flag = false;
      err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
if (FSP_SUCCESS == err)
     {
/* Application specific data processing. */
     }
   }
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CTSU successfully configured. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CTSU_SCANNING | Scanning this instance or other. |
| FSP_ERR_CTSU_NOT_GET_DATA | The previous data has not been retrieved by DataGet. |

### ◆ R_CTSU_DataGet()

fsp_err_t R_CTSU_DataGet ( ctsu_ctrl_t *const *p_ctrl*, uint16_t * *p_data* )

This function gets the sensor values as scanned by the CTSU. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Implements ctsu_api_t::dataGet.

Example:

```
while (true)
   {
      err = R_CTSU_ScanStart(&g_ctsu_ctrl);
     handle_error(err);
while (!g_scan_flag)
     {
/* Wait for scan end callback */
     }
      g_scan_flag = false;
      err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
if (FSP_SUCCESS == err)
     {
/* Application specific data processing. */
     }
   }
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CTSU successfully configured. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CTSU_SCANNING | Scanning this instance. |
| FSP_ERR_CTSU_INCOMPLETE_TUNING | Incomplete initial offset tuning. |

### ◆ R_CTSU_Close()

fsp_err_t R_CTSU_Close ( ctsu_ctrl_t *const *p_ctrl*)

Disables specified CTSU control block. Implements ctsu_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CTSU successfully configured. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ R_CTSU_VersionGet()

fsp_err_t R_CTSU_VersionGet ( fsp_version_t *const *p_version*)

Return CTSU HAL driver version. Implements ctsu_api_t::versionGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version information successfully read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter |

## 4.2.11 Digital to Analog Converter (r_dac)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_DAC_Open (dac_ctrl_t *p_api_ctrl, dac_cfg_t const *const p_cfg) |
| fsp_err_t | R_DAC_Write (dac_ctrl_t *p_api_ctrl, uint16_t value) |
| fsp_err_t | R_DAC_Start (dac_ctrl_t *p_api_ctrl) |
| fsp_err_t | R_DAC_Stop (dac_ctrl_t *p_api_ctrl) |
| fsp_err_t | R_DAC_Close (dac_ctrl_t *p_api_ctrl) |
| fsp_err_t | R_DAC_VersionGet (fsp_version_t *p_version) |

**Detailed Description**

Driver for the DAC12 peripheral on RA MCUs. This module implements the DAC Interface.

# Overview

### Features

The DAC module outputs one of 4096 voltage levels between the positive and negative reference voltages.

- Supports setting left-justified or right-justified 12-bit value format for the 16-bit input data registers
- Supports output amplifiers on selected MCUs
- Supports charge pump on selected MCUs
- Supports synchronization with the Analog-to-Digital Converter (ADC) module

# Configuration

*Note*

*For MCUs supporting more than one channel, the following configuration options are shared by all the DAC channels:*

- *Synchronize with ADC*
- *Data Format*
- *Charge Pump*

### Build Time Configurations for r_dac

The following build time configurations are defined in fsp_cfg/r_dac_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Analog > DAC Driver on r_dac

This module can be added to the Stacks tab via New Stack > Driver > Analog > DAC Driver on r_dac:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_dac0 | Module name. |
| Channel | Value must be an integer greater than or equal to 0 | 0 | Specify the hardware channel. |
| Synchronize with ADC | • Enabled<br>• Disabled | Disabled | Enable DA/AD synchronization. |
| Data Format | • Right Justified<br>• Left Justified | Right Justified | Specify the DAC data format. |

| | | |
|---|---|---|
| Output Amplifier | MCU Specific Options | Enable the DAC output amplifier. |
| Charge Pump (Requires MOCO active) | MCU Specific Options | Enable the DAC charge pump. |
| ELC Trigger Source | MCU Specific Options | ELC event source that will trigger the DAC to start a conversion. |

### Clock Configuration

The DAC peripheral module uses PCLKB as its clock source.

### Pin Configuration

The DAn pins are used as analog outputs. Each DAC channel has one output pin.

The AVCC0 and AVSS0 pins are power and ground supply pins for the DAC and ADC.

The VREFH and VREFL pins are top and ground voltage reference pins for the DAC and ADC.

## Usage Notes

### Charge Pump

The charge pump must be enabled when using DAC pin output while operating at $AV_{CC}$ < 2.7V.

*Note*

*The MOCO must be running to use the charge pump.*
*If the DAC output is to be routed to an internal signal, do not enable the charge pump.*

### Synchronization with ADC

When ADC synchronization is enabled and an ADC conversion is in progress, if a DAC conversion is started it will automatically be delayed until after the ADC conversion is complete.

### Limitations

- For MCUs supporting ADC unit 1:
  - Once synchronization between DAC and ADC unit 1 is turned on during R_DAC_Open synchronization cannot be turned off by the driver. In order to desynchronize DAC with ADC unit 1, manually clear DAADSCR.DAADST to 0 when the ADCSR.ADST bit is 0 and ADC unit 1 is halted.
  - The DAC module can only be synchronized with ADC unit 1.
  - For MCUs having more than 1 DAC channel, both channels are synchronized with ADC unit 1 if synchronization is enabled.

## Examples

### Basic Example

This is a basic example of minimal use of the R_DAC in an application. This example shows how this driver can be used for basic Digital to Analog Conversion operations.

```
void basic_example (void)
{
 fsp_err_t err;
    uint16_t value;
 /* Pin configuration: Output enable DA0 as Analog. */
 /* Initialize the DAC channel */
    err = R_DAC_Open(&g_dac_ctrl, &g_dac_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
    value = (uint16_t) DAC_EXAMPLE_VALUE_ABC;
    err = R_DAC_Write(&g_dac_ctrl, value);
    handle_error(err);
    err = R_DAC_Start(&g_dac_ctrl);
    handle_error(err);
}
```

## Data Structures

| | |
|---|---|
| struct | dac_instance_ctrl_t |
| struct | dac_extended_cfg_t |

## Data Structure Documentation

### ◆ dac_instance_ctrl_t

| |
|---|
| struct dac_instance_ctrl_t |
| DAC instance control block. |

### ◆ dac_extended_cfg_t

| | | |
|---|---|---|
| struct dac_extended_cfg_t | | |
| DAC extended configuration | | |
| Data Fields | | |
| bool | enable_charge_pump | Enable DAC charge pump available on selected MCUs. |
| bool | output_amplifier_enabled | Output amplifier enable available on selected MCUs. |
| dac_data_format_t | data_format | Data format. |

## Function Documentation

### ◆ R_DAC_Open()

fsp_err_t R_DAC_Open ( dac_ctrl_t * *p_api_ctrl*, dac_cfg_t const *const *p_cfg* )

Perform required initialization described in hardware manual. Implements dac_api_t::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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel was successfully opened. |
| FSP_ERR_ASSERTION | Parameter check failure due to one or more reasons below:<br>1. One or both of the following parameters may be NULL: p_api_ctrl or p_cfg<br>2. data_format value in p_cfg is out of range.<br>3. Extended configuration structure is set to NULL for MCU supporting charge pump. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Channel ID requested in p_cfg may not available on the devices. |
| FSP_ERR_ALREADY_OPEN | The control structure is already opened. |

### ◆ R_DAC_Write()

fsp_err_t R_DAC_Write ( dac_ctrl_t * *p_api_ctrl*, uint16_t *value* )

Write data to the D/A converter and enable the output if it has not been enabled.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data is successfully written to the D/A Converter. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Channel associated with p_ctrl has not been opened. |

### ◆ R_DAC_Start()

fsp_err_t R_DAC_Start ( dac_ctrl_t * *p_api_ctrl*)

Start the D/A conversion output if it has not been started.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel is started successfully. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_IN_USE | Attempt to re-start a channel. |
| FSP_ERR_NOT_OPEN | Channel associated with p_ctrl has not been opened. |

### ◆ R_DAC_Stop()

fsp_err_t R_DAC_Stop ( dac_ctrl_t * *p_api_ctrl*)

Stop the D/A conversion and disable the output signal.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The control is successfully stopped. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Channel associated with p_ctrl has not been opened. |

### ◆ R_DAC_Close()

fsp_err_t R_DAC_Close ( dac_ctrl_t * *p_api_ctrl*)

Stop the D/A conversion, stop output, and close the DAC channel.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel is successfully closed. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Channel associated with p_ctrl has not been opened. |

### ◆ R_DAC_VersionGet()

| fsp_err_t R_DAC_VersionGet ( fsp_version_t * *p_version*) |
|---|
| Get version and store it in provided pointer p_version. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully retrieved version information. |
| FSP_ERR_ASSERTION | p_version is NULL. |

## 4.2.12 Digital to Analog Converter (r_dac8)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_DAC8_Open (dac_ctrl_t *const p_ctrl, dac_cfg_t const *const p_cfg) |
| fsp_err_t | R_DAC8_Close (dac_ctrl_t *const p_ctrl) |
| fsp_err_t | R_DAC8_Write (dac_ctrl_t *const p_ctrl, uint16_t value) |
| fsp_err_t | R_DAC8_Start (dac_ctrl_t *const p_ctrl) |
| fsp_err_t | R_DAC8_Stop (dac_ctrl_t *const p_ctrl) |
| fsp_err_t | R_DAC8_VersionGet (fsp_version_t *p_version) |

**Detailed Description**

Driver for the DAC8 peripheral on RA MCUs. This module implements the DAC Interface.

# Overview

**Features**

The DAC8 module outputs one of 256 voltage levels between the positive and negative reference voltages. DAC8 on selected MCUs have below features

- Charge pump control
- Synchronization with the Analog-to-Digital Converter (ADC) module
- Multiple Operation Modes
  - Normal

- Real-Time (Event Link)

# Configuration

*Note*

*For MCUs supporting more than one channel, the following configuration options are shared by all the DAC8 channels:*

- *Synchronize with ADC*
- *Charge Pump*

### Build Time Configurations for r_dac8

The following build time configurations are defined in fsp_cfg/r_dac8_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Analog > DAC8 Driver on r_dac8

This module can be added to the Stacks tab via New Stack > Driver > Analog > DAC8 Driver on r_dac8:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_dac8_0 | Module name. |
| Channel | Value must be an integer greater than or equal to 0 | 0 | Specify the hardware channel. |
| D/A A/D Synchronous Conversion | • Enabled<br>• Disabled | Disabled | Synchronize the DAC8 update with the ADC to reduce interference with A/D conversions. |
| DAC Mode | • Normal Mode<br>• Real-time (Event Link) Mode | Normal Mode | Select the DAC operating mode |
| Real-time Trigger Event | MCU Specific Options | | Specify the event used to trigger conversion in Real-time mode. This setting is only valid when Real-time mode is enabled. |
| Charge Pump (Requires | • Enabled | Enabled | Enable the DAC charge |

MOCO active) • Disabled pump.

### Clock Configuration

The DAC8 peripheral module uses the PCLKB as its clock source.

### Pin Configuration

The DA8_n pins are used as analog outputs. Each DAC8 channel has one output pin.

The AVCC0 and AVSS0 pins are power and ground supply and reference pins for the DAC8.

## Usage Notes

### Charge Pump

The charge pump must be enabled when using DAC8 pin output while operating at $AV_{CC}$ < 2.7V.

*Note*

*The MOCO must be running to use the charge pump.*
*If DAC8 output is to be routed to an internal signal, do not enable the charge pump.*

### Synchronization with ADC

When ADC synchronization is enabled and an ADC conversion is in progress, if a DAC8 conversion is started it will automatically be delayed until after the ADC conversion is complete.

### Real-time Mode

When Real-time mode is selected, the DAC8 will perform a conversion each time the selected ELC event is received.

### Limitations

- Synchronization between DAC8 and ADC is activated when calling R_DAC8_Open. At this point synchronization cannot be deactivated by the driver. In order to desynchronize DAC8 with ADC, manually clear DACADSCR.DACADST to 0 while the ADCSR.ADST bit is 0 and the ADC is halted.
- For MCUs having more than 1 DAC8 channel, both channels are synchronized with ADC if synchronization is enabled.

## Examples

### Basic Example

This is a basic example of minimal use of the R_DAC8 in an application. This example shows how this driver can be used for basic 8 bit Digital to Analog Conversion operations.

```
dac8_instance_ctrl_t g_dac8_ctrl;

dac_cfg_t g_dac8_cfg =

{

    .channel            = 0U,
```

```
    .ad_da_synchronized = false,
    .p_extend           = &g_dac8_cfg_extend
};
void basic_example (void)
{
 fsp_err_t err;
    uint16_t value;
 /* Pin configuration: Output enable DA8_0(RA2A1) as Analog. */
 /* Initialize the DAC8 channel */
    err = R_DAC8_Open(&g_dac8_ctrl, &g_dac8_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
    value = (uint8_t) DAC8_EXAMPLE_VALUE_ABC;
 /* Write value to DAC module */
    err = R_DAC8_Write(&g_dac8_ctrl, value);
    handle_error(err);
 /* Start DAC8 conversion */
    err = R_DAC8_Start(&g_dac8_ctrl);
    handle_error(err);
}
```

### Data Structures

| | |
|---|---|
| struct | dac8_instance_ctrl_t |
| struct | dac8_extended_cfg_t |

### Enumerations

| | |
|---|---|
| enum | dac8_mode_t |

## Data Structure Documentation

### ◆ dac8_instance_ctrl_t

| |
|---|
| struct dac8_instance_ctrl_t |
| DAC8 instance control block. DO NOT INITIALIZE. |

### ◆ dac8_extended_cfg_t

| |
|---|
| struct dac8_extended_cfg_t |

| DAC8 extended configuration | | |
|---|---|---|
| Data Fields | | |
| bool | enable_charge_pump | Enable DAC charge pump. |
| dac8_mode_t | dac_mode | DAC mode. |

## Enumeration Type Documentation

### ◆ dac8_mode_t

| enum dac8_mode_t | |
|---|---|
| Enumerator | |
| DAC8_MODE_NORMAL | DAC Normal mode. |
| DAC8_MODE_REAL_TIME | DAC Real-time (event link) mode. |

## Function Documentation

### ◆ R_DAC8_Open()

fsp_err_t R_DAC8_Open ( dac_ctrl_t *const *p_ctrl*, dac_cfg_t const *const *p_cfg* )

Perform required initialization described in hardware manual.

Implements dac_api_t::open.

Configures a single DAC channel. 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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel was successfully opened. |
| FSP_ERR_ASSERTION | One or both of the following parameters may be NULL: p_ctrl or p_cfg |
| FSP_ERR_ALREADY_OPEN | The instance control structure has already been opened. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | An invalid channel was requested. |
| FSP_ERR_NOT_ENABLED | Setting DACADSCR is not enabled when ADCSR.ADST = 0. |

***Note***

*This function is reentrant for different channels. It is not reentrant for the same channel.*

### ◆ R_DAC8_Close()

fsp_err_t R_DAC8_Close ( dac_ctrl_t *const *p_ctrl*)

Stop the D/A conversion, stop output, and close the DAC channel.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel is successfully closed. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Channel associated with p_instance_ctrl has not been opened. |

### ◆ R_DAC8_Write()

fsp_err_t R_DAC8_Write ( dac_ctrl_t *const *p_ctrl*, uint16_t *value* )

Write data to the D/A converter.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data is successfully written to the D/A Converter. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Channel associated with p_instance_ctrl has not been opened. |
| FSP_ERR_OVERFLOW | Data overflow when data value exceeds 8-bit limit. |

### ◆ R_DAC8_Start()

fsp_err_t R_DAC8_Start ( dac_ctrl_t *const *p_ctrl*)

Start the D/A conversion output.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel is started successfully. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Channel associated with p_instance_ctrl has not been opened. |
| FSP_ERR_IN_USE | Attempt to re-start a channel. |

### ◆ R_DAC8_Stop()

| fsp_err_t R_DAC8_Stop ( dac_ctrl_t *const *p_ctrl*) |
|---|
| Stop the D/A conversion and disable the output signal. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The control is successfully stopped. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Channel associated with p_instance_ctrl has not been opened. |

### ◆ R_DAC8_VersionGet()

| fsp_err_t R_DAC8_VersionGet ( fsp_version_t * *p_version*) |
|---|
| Get version and store it in provided pointer p_version. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully retrieved version information. |
| FSP_ERR_ASSERTION | p_version is NULL. |

## 4.2.13 Direct Memory Access Controller (r_dmac)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_DMAC_Open (transfer_ctrl_t *const p_api_ctrl, transfer_cfg_t const *const p_cfg) |
| fsp_err_t | R_DMAC_Reconfigure (transfer_ctrl_t *const p_api_ctrl, transfer_info_t *p_info) |
| fsp_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) |
| fsp_err_t | R_DMAC_SoftwareStart (transfer_ctrl_t *const p_api_ctrl, transfer_start_mode_t mode) |
| fsp_err_t | R_DMAC_SoftwareStop (transfer_ctrl_t *const p_api_ctrl) |

| | |
|---|---|
| fsp_err_t | R_DMAC_Enable (transfer_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_DMAC_Disable (transfer_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_DMAC_InfoGet (transfer_ctrl_t *const p_api_ctrl, transfer_properties_t *const p_info) |
| fsp_err_t | R_DMAC_Close (transfer_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_DMAC_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Driver for the DMAC peripheral on RA MCUs. This module implements the Transfer Interface.

# Overview

The Direct Memory Access Controller (DMAC) transfers data from one memory location to another without using the CPU.

### Features

- Supports multiple transfer modes
  - Normal transfer
  - Repeat transfer
  - Block transfer
- Address increment, decrement, fixed, or offset modes
- Triggered by ELC events
  - Some exceptions apply, see the Event table in the Event Numbers section of the Interrupt Controller Unit chapter of the hardware manual
- Supports 1, 2, and 4 byte data units

# Configuration

### Build Time Configurations for r_dmac

The following build time configurations are defined in fsp_cfg/r_dmac_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Transfer > Transfer Driver on r_dmac

This module can be added to the Stacks tab via New Stack > Driver > Transfer > Transfer Driver on r_dmac :

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_transfer0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Specify the hardware channel. |
| Mode | • Normal<br>• Repeat<br>• Block | Normal | Select the transfer mode. Normal: One transfer per activation, transfer ends after Number of Transfers; Repeat: One transfer per activation, Repeat Area address reset after Number of Transfers, transfer ends after Number of Blocks; Block: Number of Blocks per activation, Repeat Area address reset after Number of Transfers, transfer ends after Number of Blocks. |
| Transfer Size | • 1 Byte<br>• 2 Bytes<br>• 4 Bytes | 2 Bytes | Select the transfer size. |
| Destination Address Mode | • Fixed<br>• Offset addition<br>• Incremented<br>• Decremented | Fixed | Select the address mode for the destination. |
| Source Address Mode | • Fixed<br>• Offset addition<br>• Incremented<br>• Decremented | Fixed | Select the address mode for the source. |
| Repeat Area (Unused in Normal Mode) | • Destination<br>• Source | Source | Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode. |
| Destination Pointer | Manual Entry | NULL | Specify the transfer destination pointer. |
| Source Pointer | Manual Entry | NULL | Specify the transfer source pointer. |
| Number of Transfers | Value must be a non- | 1 | Specify the number of |

| | | | |
|---|---|---|---|
| | negative integer | | transfers. |
| Number of Blocks (Valid only in Repeat and Block Mode) | Value must be a non-negative integer | 0 | Specify the number of blocks to transfer in Repeat or Block mode. |
| Activation Source | MCU Specific Options | | Select the DMAC transfer start event. If no ELC event is chosen then software start can be used. |
| Callback | Name must be a valid C symbol | NULL | A user callback that is called at the end of the transfer. |
| Transfer End Interrupt Priority | MCU Specific Options | | Select the transfer end interrupt priority. |
| Interrupt Frequency | • Interrupt after all transfers have completed<br>• Interrupt after each block, or repeat size is transfered | Interrupt after all transfers have completed | Select to have interrupt after each transfer or after last transfer. |
| Offset value (Valid only when address mode is \'Offset\') | Value must be a 24 bit signed integer. | 1 | Offset value * transfer size is added to the address after each transfer. |

### Clock Configuration

The DMAC peripheral module uses ICLK as the clock source. The ICLK frequency is set by using the **Clocks** tab of the RA Configuration editor prior to a build, or by using the CGC module at run-time.

### Pin Configuration

This module does not use I/O pins.

## Usage Notes

### Transfer Modes

The DMAC Module supports three modes of operation.

- **Normal Mode** - In normal mode, a single data unit is transfered every time the configured ELC event is received by the DMAC channel. A data unit can be 1-byte, 2-bytes, or 4-bytes. The source and destination addresses can be fixed, increment, decrement, or add an offset to the next data unit after each transfer. A 16-bit counter decrements after each transfer. When the counter reaches 0, transfers will no longer be triggered by the ELC event and the CPU can be interrupted to signal that all transfers have finished.
- **Repeat Mode** - Repeat mode works the same way as normal mode, however the length is limited to an integer in the range[1,1024]. When the transfer counter reaches 0, the counter is reset to its configured value, the repeat area (source or destination address) resets to its starting address and the block count remaining will decrement by 1. When the

block count reaches 0, transfers will no longer be triggered by the ELC event and the CPU may be interrupted to signal that all transfers have finished.
- **Block Mode** - In block mode, the amount of data units transfered by each interrupt can be set to an integer in the range [1,1024]. The number of blocks to transfer can also be configured to a 16-bit number. After each block transfer the repeat area (source or destination address) will reset to the original address and the other address will be incremented or decremented to the next block.

### Selecting the DTC or DMAC

The Transfer API is implemented by both DTC and the DMAC so that applications can switch between the DTC and the DMAC. When selecting between them, consider these factors:

| | DTC | DMAC |
|---|---|---|
| Repeat Mode | • Repeats forever<br>• Max repeat size is 256 x 4 bytes | • Configurable number of repeats<br>• Max repeat size is 1024 x 4 bytes |
| Block Mode | • Max block size is 256 x 4 bytes | • Max block size is 1024 x 4 bytes |
| Channels | • One instance per interrupt | • MCU specific (8 channels or less) |
| Chained Transfers | • Supported | • Not Supported |
| Software Trigger | • Must use the software ELC event | • Has support for software trigger without using software ELC event<br>• Supports TRANSFER_START_MODE_SINGLE and TRANSFER_START_MODE_REPEAT |
| Offset Address Mode | • Not supported | • Supported |

### Interrupts

The DTC and DMAC interrupts behave differently. The DTC uses the configured IELSR event IRQ as the interrupt source whereas each DMAC channel has its own IRQ.

The transfer_info_t::irq setting also behaves a little differently depending on which mode is selected.

## Normal Mode

| | DTC | DMAC |
|---|---|---|
| TRANSFER_IRQ_EACH | Interrupt after each transfer | N/A |
| TRANSFER_IRQ_END | Interrupt after last transfer | Interrupt after last transfer |

## Repeat Mode

| | DTC | DMAC |
|---|---|---|
| TRANSFER_IRQ_EACH | Interrupt after each transfer | Interrupt after each repeat |

| | | |
|---|---|---|
| TRANSFER_IRQ_END | Interrupt after each repeat | Interrupt after last transfer |

### Block Mode

| | DTC | DMAC |
|---|---|---|
| TRANSFER_IRQ_EACH | Interrupt after each block | Interrupt after each block |
| TRANSFER_IRQ_END | Interrupt after last block | Interrupt after last block |

#### Additional Considerations

- The DTC requires a moderate amount of RAM (one transfer_info_t struct per open instance + DTC_VECTOR_TABLE_SIZE).
- The DTC stores transfer information in RAM and writes back to RAM after each transfer whereas the DMAC stores all transfer information in registers.
- When transfers are configured for more than one activation source, the DTC must fetch the transfer info from RAM on each interrupt. This can cause a higher latency between transfers.

#### Offset Address Mode

When the source or destination mode is configured to offset mode, a configurable offset is added to the source or destination pointer after each transfer. The offset is a signed 24 bit number.

## Examples

#### Basic Example

This is a basic example of minimal use of the DMAC in an application. In this case, one or more events have been routed to the DMAC for handling so it only needs to be enabled to start accepting transfers.

```
void dmac_minimal_example (void)
{
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_DMAC_Open(&g_transfer_ctrl, &g_transfer_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Enable the DMAC so that it responds to transfer requests. */
    err = R_DMAC_Enable(&g_transfer_ctrl);
    handle_error(err);
}
```

#### CRC32 Example

In this example the DMAC is used to feed the CRC peripheral to perform a CRC32 operation.

```
volatile bool g_transfer_complete = false;
void dmac_callback (dmac_callback_args_t * cb_data)
{
 FSP_PARAMETER_NOT_USED(cb_data);
    g_transfer_complete = true;
}
void dmac_crc_example (void)
{
    uint8_t p_src[TRANSFER_LENGTH];
 /* Initialize p_src to [ABC..OP] */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       p_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Set transfer source address to p_src */
    g_transfer_cfg.p_info->p_src = (void *) p_src;
 /* Set transfer destination address to the CRC data input register */
    g_transfer_cfg.p_info->p_dest = (void *) &R_CRC->CRCDIR;
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_DMAC_Open(&g_transfer_ctrl, &g_transfer_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Enable DMAC transfers. */
    (void) R_DMAC_Enable(&g_transfer_ctrl);
 /* Open the CRC module. */
    err = R_CRC_Open(&g_crc_ctrl, &g_crc_cfg);
    handle_error(err);
 /* Clear the transfer complete flag. */
    g_transfer_complete = false;
 /* Trigger the transfer using software. */
    err = R_DMAC_SoftwareStart(&g_transfer_ctrl, TRANSFER_START_MODE_SINGLE);
    handle_error(err);
 while (!g_transfer_complete)
    {
```

```
/* Wait for transfer complete interrupt */
   }
/* Get CRC result and perform final XOR. */
   uint32_t crc32;
   (void) R_CRC_CalculatedValueGet(&g_crc_ctrl, &crc32);
   crc32 ^= CRC32_FINAL_XOR_VALUE;
/* Verify that the CRC32 is calculated correctly. */
/* CRC32("ABCD...NOP") = 0xE0E8FF4D. */
const uint32_t expected_crc32 = 0xE0E8FF4D;
if (expected_crc32 != crc32)
   {
/* Handle any CRC errors. This function should be defined by the user. */
      handle_crc_error();
   }
}
```

## Data Structures

| | |
|---|---|
| struct | dmac_instance_ctrl_t |
| struct | dmac_callback_args_t |
| struct | dmac_extended_cfg_t |

## Macros

| | |
|---|---|
| #define | DMAC_MAX_NORMAL_TRANSFER_LENGTH |
| #define | DMAC_MAX_REPEAT_TRANSFER_LENGTH |
| #define | DMAC_MAX_BLOCK_TRANSFER_LENGTH |
| #define | DMAC_MAX_REPEAT_COUNT |
| #define | DMAC_MAX_BLOCK_COUNT |

## Data Structure Documentation

### ◆ dmac_instance_ctrl_t

| struct dmac_instance_ctrl_t |
|---|
| Control block used by driver. DO NOT INITIALIZE - this structure will be initialized in transfer_api_t::open. |

### ◆ dmac_callback_args_t

| struct dmac_callback_args_t | | |
|---|---|---|
| Callback function parameter data. | | |
| Data Fields | | |
| void const * | p_context | Placeholder for user data. Set in r_transfer_t::open function in transfer_cfg_t. |

### ◆ dmac_extended_cfg_t

| struct dmac_extended_cfg_t | |
|---|---|
| DMAC transfer configuration extension. This extension is required. | |
| **Data Fields** | |
| uint8_t | channel |
| | Channel number, does not apply to all HAL drivers. |
| | |
| IRQn_Type | irq |
| | DMAC interrupt number. |
| | |
| uint8_t | ipl |
| | DMAC interrupt priority. |
| | |
| int32_t | offset |
| | Offset value used with transfer_addr_mode_t::TRANSFER_ADDR_MODE_OFFSET. |
| | |
| elc_event_t | activation_source |
| | |
| void(* | p_callback )(dmac_callback_args_t *cb_data) |
| | |
| void const * | p_context |
| | |

## Field Documentation

### ◆ activation_source

elc_event_t dmac_extended_cfg_t::activation_source

Select which event will trigger the transfer.

***Note***

*Select ELC_EVENT_NONE for software activation in order to use softwareStart and softwareStart to trigger transfers.*

### ◆ p_callback

void(* dmac_extended_cfg_t::p_callback) (dmac_callback_args_t *cb_data)

Callback for transfer end interrupt.

### ◆ p_context

void const* dmac_extended_cfg_t::p_context

Placeholder for user data. Passed to the user p_callback in dmac_callback_args_t.

## Macro Definition Documentation

### ◆ DMAC_MAX_NORMAL_TRANSFER_LENGTH

#define DMAC_MAX_NORMAL_TRANSFER_LENGTH

Max configurable number of transfers in TRANSFER_MODE_NORMAL.

### ◆ DMAC_MAX_REPEAT_TRANSFER_LENGTH

#define DMAC_MAX_REPEAT_TRANSFER_LENGTH

Max number of transfers per repeat for TRANSFER_MODE_REPEAT.

### ◆ DMAC_MAX_BLOCK_TRANSFER_LENGTH

#define DMAC_MAX_BLOCK_TRANSFER_LENGTH

Max number of transfers per block in TRANSFER_MODE_BLOCK

### ◆ DMAC_MAX_REPEAT_COUNT

#define DMAC_MAX_REPEAT_COUNT

Max configurable number of repeats to trasnfer in TRANSFER_MODE_REPEAT

### ◆ DMAC_MAX_BLOCK_COUNT

| #define DMAC_MAX_BLOCK_COUNT |
|---|
| Max configurable number of blocks to transfer in TRANSFER_MODE_BLOCK |

## Function Documentation

### ◆ R_DMAC_Open()

fsp_err_t R_DMAC_Open ( transfer_ctrl_t *const *p_api_ctrl*, transfer_cfg_t const *const *p_cfg* )

Configure a DMAC channel.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful open. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The configured channel is invalid. |
| FSP_ERR_IRQ_BSP_DISABLED | The IRQ associated with the activation source is not enabled in the BSP. |
| FSP_ERR_ALREADY_OPEN | The control structure is already opened. |

### ◆ R_DMAC_Reconfigure()

fsp_err_t R_DMAC_Reconfigure ( transfer_ctrl_t *const *p_api_ctrl*, transfer_info_t * *p_info* )

Reconfigure the transfer with new transfer info.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfer is configured and will start when trigger occurs. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_ENABLED | DMAC is not enabled. The current configuration must not be valid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DMAC_Open to initialize the control block. |

### ◆ R_DMAC_Reset()

fsp_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* )

Reset transfer source, destination, and number of transfers.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfer reset successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_ENABLED | DMAC is not enabled. The current configuration must not be valid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DMAC_Open to initialize the control block. |

### ◆ R_DMAC_SoftwareStart()

fsp_err_t R_DMAC_SoftwareStart ( transfer_ctrl_t *const *p_api_ctrl*, transfer_start_mode_t *mode* )

If the mode is TRANSFER_START_MODE_SINGLE initiate a single transfer with software. If the mode is TRANSFER_START_MODE_REPEAT continue triggering transfers until all of the transfers are completed.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfer started written successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DMAC_Open to initialize the control block. |
| FSP_ERR_UNSUPPORTED | Handle was not configured for software activation. |

### ◆ R_DMAC_SoftwareStop()

fsp_err_t R_DMAC_SoftwareStop ( transfer_ctrl_t *const *p_api_ctrl*)

Stop software transfers if they were started with TRANSFER_START_MODE_REPEAT.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfer stopped written successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DMAC_Open to initialize the control block. |

### ◆ R_DMAC_Enable()

fsp_err_t R_DMAC_Enable ( transfer_ctrl_t *const *p_api_ctrl*)

Enable transfers for the configured activation source.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter value written successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DMAC_Open to initialize the control block. |

### ◆ R_DMAC_Disable()

fsp_err_t R_DMAC_Disable ( transfer_ctrl_t *const *p_api_ctrl*)

Disable transfers so that they are no longer triggered by the activation source.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter value written successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DMAC_Open to initialize the control block. |

### ◆ R_DMAC_InfoGet()

fsp_err_t R_DMAC_InfoGet ( transfer_ctrl_t *const *p_api_ctrl*, transfer_properties_t *const *p_info* )

Set driver specific information in provided pointer.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information has been written to p_info. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DMAC_Open to initialize the control block. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |

### ◆ R_DMAC_Close()

fsp_err_t R_DMAC_Close ( transfer_ctrl_t *const *p_api_ctrl*)

Disable transfer and clean up internal data. Implements transfer_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DMAC_Open to initialize the control block. |

### ◆ R_DMAC_VersionGet()

fsp_err_t R_DMAC_VersionGet ( fsp_version_t *const *p_version*)

Set driver version based on compile time macros.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |

## 4.2.14 Data Operation Circuit (r_doc)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | R_DOC_Open (doc_ctrl_t *const p_api_ctrl, doc_cfg_t const *const p_cfg) |
| fsp_err_t | R_DOC_Close (doc_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_DOC_StatusGet (doc_ctrl_t *const p_api_ctrl, doc_status_t *const p_status) |
| fsp_err_t | R_DOC_Write (doc_ctrl_t *const p_api_ctrl, uint16_t data) |
| fsp_err_t | R_DOC_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Driver for the DOC peripheral on RA MCUs. This module implements the DOC Interface.

# Overview

### Features

The DOC HAL module peripheral is used to compare, add or subtract 16-bit data and can detect the following events:

- A match or mismatch between data values
- Overflow of an addition operation
- Underflow of a subtraction operation

A user-defined callback can be created to inform the CPU when any of above events occur.

# Configuration

### Build Time Configurations for r_doc

The following build time configurations are defined in fsp_cfg/r_doc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Monitoring > Data Operation Circuit Driver on r_doc

This module can be added to the Stacks tab via New Stack > Driver > Monitoring > Data Operation Circuit Driver on r_doc:

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_doc0 | Module name. |
| Event | • Comparison mismatch<br>• Comparison match<br>• Addition overflow<br>• Subtraction underflow | Comparison mismatch | Select the event that will trigger the DOC interrupt. |
| Reference/Initial Data | Value must be a 16 bit integer between 0 and 65535 | 0 | Enter Initial Value for Addition/Subtraction or enter reference value for comparison. |
| Callback | Name must be a valid C symbol | NULL | A user callback function must be provided. This will be called from the interrupt service routine (ISR) when the configured DOC event occurs. |
| DOC Interrupt Priority | MCU Specific Options | | Select the DOC interrupt priority. |

### Clock Configuration

The DOC HAL module does not require a specific clock configuration.

### Pin Configuration

The DOC HAL module does not require and specific pin configurations.

## Usage Notes

### DMAC/DTC Integration

DOC can be used with Direct Memory Access Controller (r_dmac) or Data Transfer Controller (r_dtc) to write to the input register without CPU intervention. DMAC is more useful for most DOC applications because it can be started directly from software. To write DOC input data with DTC/DMAC, set transfer_info_t::p_dest to R_DOC->DODIR.

## Examples

### Basic Example

This is a basic example of minimal use of the R_DOC in an application. This example shows how this driver can be used for continuous 16 bit addition operation while reading the result at every overflow event.

```
#define DOC_EXAMPLE_VALUE 0xF000
```

```
uint32_t g_callback_event_counter = 0;
/* This callback is called when DOC overflow event occurs. It is registered in
doc_cfg_t when R_DOC_Open is
 * called. */
void doc_callback (doc_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
    g_callback_event_counter++;
}
void basic_example (void)
{
 fsp_err_t    err;
 doc_status_t result;
 /* Initialize the DOC module for addition with initial value specified in
doc_cfg_t::doc_data. */
    err = R_DOC_Open(&g_doc_ctrl, &g_doc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Write data to the DOC Data Input Register and read the result of addition from
status register when an
  * interrupt occurs. */
 for (int i = 0; i < 5; i++)
    {
       err = R_DOC_Write(&g_doc_ctrl, DOC_EXAMPLE_VALUE);
      handle_error(err);
 if (g_callback_event_counter >= 1)
      {
 /* Read the result of the operation */
            err = R_DOC_StatusGet(&g_doc_ctrl, &result);
      handle_error(err);
       }
    }
}
```

## Function Documentation

### ◆ R_DOC_Open()

fsp_err_t R_DOC_Open ( doc_ctrl_t *const *p_api_ctrl*, doc_cfg_t const *const *p_cfg* )

Opens and configures the Data Operation Circuit (DOC) in comparison, addition or subtraction mode and sets initial data for addition or subtraction, or reference data for comparison.

Example:

```
/* Initialize the DOC module for addition with initial value specified in
doc_cfg_t::doc_data. */
    err = R_DOC_Open(&g_doc_ctrl, &g_doc_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | DOC successfully configured. |
| FSP_ERR_ALREADY_OPEN | Module already open. |
| FSP_ERR_ASSERTION | One or more pointers point to NULL or callback is NULL or the interrupt vector is invalid. |

### ◆ R_DOC_Close()

fsp_err_t R_DOC_Close ( doc_ctrl_t *const *p_api_ctrl*)

Closes the module driver. Enables module stop mode.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module successfully closed. |
| FSP_ERR_NOT_OPEN | Driver not open. |
| FSP_ERR_ASSERTION | Pointer pointing to NULL. |

***Note***

*This function will disable the DOC interrupt in the NVIC.*

### ◆ R_DOC_StatusGet()

fsp_err_t R_DOC_StatusGet ( doc_ctrl_t *const *p_api_ctrl*, doc_status_t *const *p_status* )

Returns the result of addition/subtraction.

Example:

```
/* Read the result of the operation */
            err = R_DOC_StatusGet(&g_doc_ctrl, &result);
      handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Status successfully read. |
| FSP_ERR_NOT_OPEN | Driver not open. |
| FSP_ERR_ASSERTION | One or more pointers point to NULL. |

### ◆ R_DOC_Write()

fsp_err_t R_DOC_Write ( doc_ctrl_t *const *p_api_ctrl*, uint16_t *data* )

Writes to the DODIR - DOC Input Register.

Example:

```
       err = R_DOC_Write(&g_doc_ctrl, DOC_EXAMPLE_VALUE);
      handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Values successfully written to the registers. |
| FSP_ERR_NOT_OPEN | Driver not open. |
| FSP_ERR_ASSERTION | One or more pointers point to NULL. |

### ◆ R_DOC_VersionGet()

fsp_err_t R_DOC_VersionGet ( fsp_version_t *const *p_version*)

Returns DOC HAL driver version.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version information successfully read. |
| FSP_ERR_ASSERTION | Pointer pointing to NULL. |

### 4.2.15 D/AVE 2D Port Interface (r_drw)

Modules

Driver for the DRW peripheral on RA MCUs. This module is a port of D/AVE 2D.

# Overview

*Note*

***The D/AVE 2D Port Interface (D1 layer) is a HAL layer for the D/AVE D2 layer API and does not provide any interfaces to the user. Consult the TES Dave2D Driver Documentation for further information on using the D2 API.***

*For cross-platform compatibility purposes the D1 and D2 APIs are not bound by the FSP coding guidelines for function names and general module functionality.*

# Configuration

### Build Time Configurations for r_drw

The following build time configurations are defined in fsp_cfg/r_drw_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Allow Indirect Mode | • Enabled<br>• Disabled | Enabled | Enable indirect mode to allow no-copy mode for d2_adddlist (see the TES Dave2D Driver Documentation for details). |
| Memory Allocation | • Default<br>• Custom | Default | Set Memory Allocation to Default to use built-in dynamic memory allocation for the D2 heap. This will use an RTOS heap if configured; otherwise, standard C malloc and free will be used.<br>Set to Custom to define your own allocation scheme for the D2 heap. In this case, the developer will need to define the following functions: |

| | | | |
|---|---|---|---|
| | | | void * d1_malloc(size_t size)<br>void d1_free(void * ptr) |

### Configurations for Driver > Graphics > D/AVE 2D Port Interface on r_drw

This module can be added to the Stacks tab via New Stack > Driver > Graphics > D/AVE 2D Port Interface on r_drw:

| Configuration | Options | Default | Description |
|---|---|---|---|
| D2 Device Handle Name | Name must be a valid C symbol | d2_handle0 | Set the name for the d2_device handle used when calling D2 layer functions. |
| DRW Interrupt Priority | MCU Specific Options | | Select the DRW_INT (display list completion) interrupt priority. |

### Heap Size

The D1 port layer allows the D2 driver to allocate memory as needed. There are three ways the driver can accomplish this:

1. Allocate memory using the main heap
2. Allocate memory using a heap provided by an RTOS
3. Allocate memory via user-provided functions

When the "Memory Allocation" configuration option is set to "Default" the driver will use an RTOS implementation if available and the main heap otherwise. Setting the option to "Custom" allows the user to define their own scheme using the following prototypes:

```
void * d1_malloc(size_t size);

void d1_free(void * ptr);
```

Warning

If there is no RTOS-based allocation scheme the main heap will be used. Be sure that it is enabled by setting the "Heap size (bytes)" property under RA Common on the **BSP** tab of the RA Configuration editor.

*Note*

*It is recommended to add 32KB of additional heap space for the D2 driver until the actual usage can be determined in your application.*

### Interrupt

The D1 port includes one interrupt to handle various events like display list completion or bus error. This interrupt is managed internally by the D2 driver and no callback function is available.

# Usage Notes

### Limitations

Developers should be aware of the following limitations when using the DRW engine:

- The DRW module supports two additional interrupt types - bus error and render complete. These interrupts are not needed for D2 layer operation and thus are not supported.
- If the DRW module is stopped during rendering the render will continue once the module is started again. If this behavior is undesirable in your application it is recommended to call d2_flushframe before stopping the peripheral.

## 4.2.16 Data Transfer Controller (r_dtc)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | R_DTC_Open (transfer_ctrl_t *const p_api_ctrl, transfer_cfg_t const *const p_cfg) |
| fsp_err_t | R_DTC_Reconfigure (transfer_ctrl_t *const p_api_ctrl, transfer_info_t *p_info) |
| fsp_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) |
| fsp_err_t | R_DTC_SoftwareStart (transfer_ctrl_t *const p_api_ctrl, transfer_start_mode_t mode) |
| fsp_err_t | R_DTC_SoftwareStop (transfer_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_DTC_Enable (transfer_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_DTC_Disable (transfer_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_DTC_InfoGet (transfer_ctrl_t *const p_api_ctrl, transfer_properties_t *const p_properties) |
| fsp_err_t | R_DTC_Close (transfer_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_DTC_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Driver for the DTC peripheral on RA MCUs. This module implements the Transfer Interface.

# Overview

The Data Transfer Controller (DTC) transfers data from one memory location to another without using the CPU.

The DTC uses a RAM based vector table. Each entry in the vector table corresponds to an entry in the ISR vector table. When the DTC is triggered by an interrupt, it reads the DTC vector table, fetches the transfer information, and then executes the transfer. After the transfer is executed, the DTC writes the updated transfer info back to the location pointed to by the DTC vector table.

**Features**

- Supports multiple transfer modes
  - Normal transfer
  - Repeat transfer
  - Block transfer
- Chain transfers
- Address increment, decrement or fixed modes
- Can be triggered by any event that has reserved a slot in the interrupt vector table.
  - Some exceptions apply, see the Event table in the Event Numbers section of the Interrupt Controller Unit chapter of the hardware manual
- Supports 1, 2, and 4 byte data units

# Configuration

**Build Time Configurations for r_dtc**

The following build time configurations are defined in fsp_cfg/r_dtc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Linker section to keep DTC vector table | Manual Entry | .fsp_dtc_vector_table | Section to place the DTC vector table. |

**Configurations for Driver > Transfer > Transfer Driver on r_dtc**

This module can be added to the Stacks tab via New Stack > Driver > Transfer > Transfer Driver on r_dtc :

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_transfer0 | Module name. |
| Mode | • Normal<br>• Repeat<br>• Block | Normal | Select the transfer mode. Select the transfer mode. Normal: |

| | | | |
|---|---|---|---|
| | | | One transfer per activation, transfer ends after Number of Transfers; Repeat: One transfer per activation, Repeat Area address reset after Number of Transfers, transfer repeats until stopped; Block: Number of Blocks per activation, Repeat Area address reset after Number of Transfers, transfer ends after Number of Blocks. |
| Transfer Size | • 1 Byte<br>• 2 Bytes<br>• 4 Bytes | 2 Bytes | Select the transfer size. |
| Destination Address Mode | • Fixed<br>• Incremented<br>• Decremented | Fixed | Select the address mode for the destination. |
| Source Address Mode | • Fixed<br>• Incremented<br>• Decremented | Fixed | Select the address mode for the source. |
| Repeat Area (Unused in Normal Mode) | • Destination<br>• Source | Source | Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode. |
| Interrupt Frequency | • After all transfers have completed<br>• After each transfer | After all transfers have completed | Select to have interrupt after each transfer or after last transfer. |
| Number of Transfers | Value must be a non-negative integer | 0 | Specify the number of transfers. |
| Number of Blocks (Valid only in Block Mode) | Must be a valid non-negative integer with a maximum configurable value of 65536. Applicable only in Block Mode. | 0 | Specify the number of blocks to transfer in Block mode. |
| Activation Source | MCU Specific Options | | Select the DTC transfer start event. |

**Clock Configuration**

The DTC peripheral module uses ICLK as the clock source. The ICLK frequency is set by using the **Clocks** tab of the RA Configuration editor prior to a build or by using the CGC module at runtime.

### Pin Configuration

This module does not use I/O pins.

## Usage Notes

### Transfer Modes

The DTC Module supports three modes of operation.

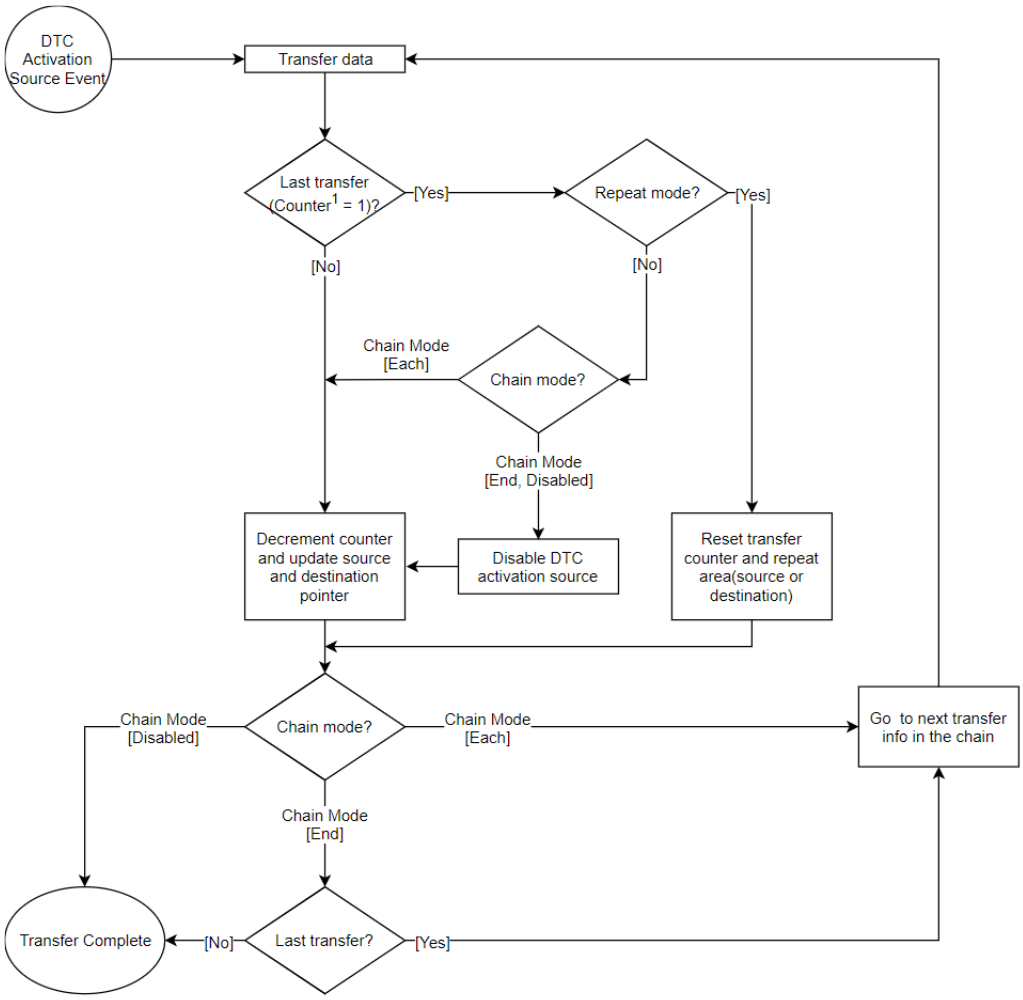
- **Normal Mode** - In normal mode, a single data unit is transfered every time an interrupt is received by the DTC. A data unit can be 1-byte, 2-bytes, or 4-bytes. The source and destination addresses can be fixed, increment or decrement to the next data unit after each transfer. A 16-bit counter (length) decrements after each transfer. When the counter reaches 0, transfers will no longer be triggered by the interrupt source and the CPU can be interrupted to signal that all transfers have finished.
- **Repeat Mode** - Repeat mode works the same way as normal mode, however the length is limited to an integer in the range[1,256]. When the tranfer counter reaches 0, the counter is reset to its configured value and the repeat area (source or destination address) resets to its starting address and transfers will still be triggered by the interrupt.
- **Block Mode** - In block mode, the amount of data units transfered by each interrupt can be set to an integer in the range [1,256]. The number of blocks to transfer can also be configured to a 16-bit number. After each block transfer the repeat area (source or destination address) will reset to the original address and the other address will be incremented or decremented to the next block.

*Note*

*1. The source and destination address of the transfer must be aligned to the configured data unit.*
*2. In normal mode the length can be set to [0,65535]. When the length is set to 0, than the transaction will execute 65536 transfers not 0.*
*3. In block mode, num_blocks can be set to [0,65535]. When the length is set to 0, than the transaction will execute 65536 transfers not 0.*

### Chaining Transfers

Multiple transfers can be configured for the same interrupt source by specifying an array of transfer_info_t structs instead of just passing a pointer to one. In this configuration, every transfer_info_t struct must be configured for a chain mode except for the last one. There are two types of chain mode; CHAIN_MODE_EACH and CHAIN_MODE_END. If a transfer is configured in CHAIN_MODE_EACH then it triggers the next transfer in the chain after it completes each transfer. If a transfer is configured in CHAIN_MODE_END then it triggers the next transfer in the chain after it completes its last transfer.




1. **Counter** refers to transfer_info_t::**length** in normal and repeat mode and transfer_info_t::**num_blocks** in block mode.

Figure 114: DTC Transfer Flowchart

### Selecting the DTC or DMAC

The Transfer API is implemented by both DTC and the DMAC so that applications can switch between the DTC and the DMAC. When selecting between them, consider these factors:

| | DTC | DMAC |
|---|---|---|
| Repeat Mode | • Repeats forever<br>• Max repeat size is 256 x 4 bytes | • Configurable number of repeats<br>• Max repeat size is 1024 x 4 bytes |
| Block Mode | • Max block size is 256 x 4 bytes | • Max block size is 1024 x 4 bytes |
| Channels | • One instance per interrupt | • MCU specific (8 channels or less) |
| Chained Transfers | • Supported | • Not Supported |
| Software Trigger | • Must use the software ELC event | • Has support for software trigger without using |

| | | |
|---|---|---|
| | | software ELC event<br>• Supports TRANSFER_START_MODE_SINGLE and TRANSFER_START_MODE_REPEAT |
| Offset Address Mode | • Not supported | • Supported |

**Additional Considerations**

- The DTC requires a moderate amount of RAM (one transfer_info_t struct per open instance + DTC_VECTOR_TABLE_SIZE).
- The DTC stores transfer information in RAM and writes back to RAM after each transfer whereas the DMAC stores all transfer information in registers.
- When transfers are configured for more than one activation source, the DTC must fetch the transfer info from RAM on each interrupt. This can cause a higher latency between transfers.
- The DTC interrupts the CPU using the activation source's IRQ. Each DMAC channel has its own IRQ.

**Interrupts**

The DTC and DMAC interrupts behave differently. The DTC uses the configured IELSR event IRQ as the interrupt source whereas each DMAC channel has its own IRQ.

The transfer_info_t::irq setting also behaves a little differently depending on which mode is selected.

## Normal Mode

| | DTC | DMAC |
|---|---|---|
| TRANSFER_IRQ_EACH | Interrupt after each transfer | N/A |
| TRANSFER_IRQ_END | Interrupt after last transfer | Interrupt after last transfer |

## Repeat Mode

| | DTC | DMAC |
|---|---|---|
| TRANSFER_IRQ_EACH | Interrupt after each transfer | Interrupt after each repeat |
| TRANSFER_IRQ_END | Interrupt after each repeat | Interrupt after last transfer |

## Block Mode

| | DTC | DMAC |
|---|---|---|
| TRANSFER_IRQ_EACH | Interrupt after each block | Interrupt after each block |
| TRANSFER_IRQ_END | Interrupt after last block | Interrupt after last block |

*Note*

*DTC_VECTOR_TABLE_SIZE = (ICU_NVIC_IRQ_SOURCES x 4) Bytes*

**Peripheral Interrupts and DTC**

When an interrupt is configured to trigger DTC transfers, the peripheral ISR will trigger on the

following conditions:

- Each transfer completed (transfer_info_t::irq = TRANSFER_IRQ_EACH)
- Last transfer completed (transfer_info_t::irq = TRANSFER_IRQ_END)

For example, if SCI1_RXI is configured to trigger DTC transfers and a SCI1_RXI event occurs, the interrupt will not fire until the DTC transfer is completed. If the DTC transfer_info_t::irq is configured to only interrupt on the last transfer, than no RXI interrupts will occur until the last transfer is completed.

*Note*

*1. The DTC activation source must be enabled in the NVIC in order to trigger DTC transfers (Modules that are designed to integrate the R_DTC module will automatically handle this).*
*2. The DTC prioritizes activation sources by granting the smaller interrupt vector numbers higher priority. The priority of interrupts to the CPU is determined by the NVIC priority.*

### Low Power Modes

DTCST must be set to 0 before transitioning to any of the following:

- Module-stop state
- Software Standby mode without Snooze mode transition
- Deep Software Standby mode

*Note*

*1. R_LPM Module stops the DTC before entering deep software standby mode and software standby without snooze mode transition.*
*2. For more information see 18.9 and 18.10 in the RA6M3 manual R01UH0886EJ0100.*

### Limitations

Developers should be aware of the following limitations when using the DTC:

- If the DTC is configured to service many different activation sources, the system could run in to performance issues due to memory contention. To address this issue, it is reccomended that the DTC vector table and transfer information be moved to their own dedicated memory area (Ex: SRAM0, SRAM1, SRAMHS). This allows memory accesses from different BUS Masters (CPU, DTC, DMAC, EDMAC and Graphics IPs) to occur in parallel.

## Examples

### Basic Example

This is a basic example of minimal use of the DTC in an application.

```
void dtc_minimal_example (void)
{
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_DTC_Open(&g_transfer_ctrl, &g_transfer_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
```

```
/* Enable the DTC to handle incoming transfer requests. */
    err = R_DTC_Enable(&g_transfer_ctrl);
    handle_error(err);
}
```

## Data Structures

| | |
|---|---|
| struct | dtc_extended_cfg_t |
| struct | dtc_instance_ctrl_t |

## Macros

| | |
|---|---|
| #define | DTC_MAX_NORMAL_TRANSFER_LENGTH |
| #define | DTC_MAX_REPEAT_TRANSFER_LENGTH |
| #define | DTC_MAX_BLOCK_TRANSFER_LENGTH |
| #define | DTC_MAX_BLOCK_COUNT |

## Data Structure Documentation

### ◆ dtc_extended_cfg_t

| struct dtc_extended_cfg_t | | |
|---|---|---|
| DTC transfer configuration extension. This extension is required. | | |
| Data Fields | | |
| IRQn_Type | activation_source | Select which IRQ will trigger the transfer. |

### ◆ dtc_instance_ctrl_t

| struct dtc_instance_ctrl_t |
|---|
| Control block used by driver. DO NOT INITIALIZE - this structure will be initialized in transfer_api_t::open. |

## Macro Definition Documentation

### ◆ DTC_MAX_NORMAL_TRANSFER_LENGTH

| #define DTC_MAX_NORMAL_TRANSFER_LENGTH |
|---|
| Max configurable number of transfers in NORMAL MODE |

### ◆ DTC_MAX_REPEAT_TRANSFER_LENGTH

| #define DTC_MAX_REPEAT_TRANSFER_LENGTH |
|---|
| Max number of transfers per repeat for REPEAT MODE |

### ◆ DTC_MAX_BLOCK_TRANSFER_LENGTH

| #define DTC_MAX_BLOCK_TRANSFER_LENGTH |
|---|
| Max number of transfers per block in BLOCK MODE |

### ◆ DTC_MAX_BLOCK_COUNT

| #define DTC_MAX_BLOCK_COUNT |
|---|
| Max configurable number of blocks to transfer in BLOCK MODE |

## Function Documentation

### ◆ R_DTC_Open()

fsp_err_t R_DTC_Open ( transfer_ctrl_t *const *p_api_ctrl*, transfer_cfg_t const *const *p_cfg* )

Configure the vector table if it hasn't been configured, enable the Module and copy the pointer to the transfer info into the DTC vector table. Implements transfer_api_t::open.

Example:

```
/* Open the transfer instance with initial configuration. */
fsp_err_t err = R_DTC_Open(&g_transfer_ctrl, &g_transfer_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful open. Transfer transfer info pointer copied to DTC Vector table. Module started. DTC vector table configured. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_UNSUPPORTED | Address Mode Offset is selected. |
| FSP_ERR_ALREADY_OPEN | The control structure is already opened. |
| FSP_ERR_IN_USE | The index for this IRQ in the DTC vector table is already configured. |
| FSP_ERR_IRQ_BSP_DISABLED | The IRQ associated with the activation source is not enabled in the BSP. |

### ◆ R_DTC_Reconfigure()

fsp_err_t R_DTC_Reconfigure ( transfer_ctrl_t *const *p_api_ctrl*, transfer_info_t * *p_info* )

Copy pointer to transfer info into the DTC vector table and enable transfer in ICU. Implements transfer_api_t::reconfigure.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfer is configured and will start when trigger occurs. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DTC_Open to initialize the control block. |
| FSP_ERR_NOT_ENABLED | Transfer source address is NULL or is not aligned corrrectly. Transfer destination address is NULL or is not aligned corrrectly. |

***Note***

*p_info must persist until all transfers are completed.*

### ◆ R_DTC_Reset()

fsp_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* )

Reset transfer source, destination, and number of transfers. Implements transfer_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfer reset successfully (transfers are enabled). |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DTC_Open to initialize the control block. |
| FSP_ERR_NOT_ENABLED | Transfer source address is NULL or is not aligned corrrectly. Transfer destination address is NULL or is not aligned corrrectly. |

### ◆ R_DTC_SoftwareStart()

fsp_err_t R_DTC_SoftwareStart ( transfer_ctrl_t *const *p_api_ctrl*, transfer_start_mode_t *mode* )

Placeholder for unsupported softwareStart function. Implements transfer_api_t::softwareStart.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | DTC software start is not supported. |

### ◆ R_DTC_SoftwareStop()

fsp_err_t R_DTC_SoftwareStop ( transfer_ctrl_t *const *p_api_ctrl*)

Placeholder for unsupported softwareStop function. Implements transfer_api_t::softwareStop.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | DTC software stop is not supported. |

### ◆ R_DTC_Enable()

fsp_err_t R_DTC_Enable ( transfer_ctrl_t *const *p_api_ctrl*)

Enable transfers on this activation source. Implements transfer_api_t::enable.

Example:

```
/* Enable the DTC to handle incoming transfer requests. */
   err = R_DTC_Enable(&g_transfer_ctrl);
   handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfers will be triggered by the activation source |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_UNSUPPORTED | Address Mode Offset is selected. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DTC_Open to initialize the control block. |

### ◆ R_DTC_Disable()

fsp_err_t R_DTC_Disable ( transfer_ctrl_t *const *p_api_ctrl*)

Disable transfer on this activation source. Implements transfer_api_t::disable.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfers will not occur on activation events. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DTC_Open to initialize the control block. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |

### ◆ R_DTC_InfoGet()

fsp_err_t R_DTC_InfoGet ( transfer_ctrl_t *const *p_api_ctrl*, transfer_properties_t *const *p_properties* )

Provides information about this transfer. Implements transfer_api_t::infoGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | p_info updated with current instance information. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DTC_Open to initialize the control block. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |

### ◆ R_DTC_Close()

fsp_err_t R_DTC_Close ( transfer_ctrl_t *const *p_api_ctrl*)

Disables DTC activation in the ICU, then clears transfer data from the DTC vector table. Implements transfer_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DTC_Open to initialize the control block. |

### ◆ R_DTC_VersionGet()

| fsp_err_t R_DTC_VersionGet ( fsp_version_t *const *p_version*) |
|---|
| Get the driver version based on compile time macros. Implements transfer_api_t::versionGet.<br><br>**Return values**<br>FSP_SUCCESS — Version information written to p_version.<br>FSP_ERR_ASSERTION — An input parameter is invalid. |

## 4.2.17 Event Link Controller (r_elc)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_ELC_Open (elc_ctrl_t *const p_ctrl, elc_cfg_t const *const p_cfg) |
| fsp_err_t | R_ELC_Close (elc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_ELC_SoftwareEventGenerate (elc_ctrl_t *const p_ctrl, elc_software_event_t event_number) |
| fsp_err_t | R_ELC_LinkSet (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral, elc_event_t signal) |
| fsp_err_t | R_ELC_LinkBreak (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral) |
| fsp_err_t | R_ELC_Enable (elc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_ELC_Disable (elc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_ELC_VersionGet (fsp_version_t *const p_version) |

**Detailed Description**

Driver for the ELC peripheral on RA MCUs. This module implements the ELC Interface.

# Overview

The event link controller (ELC) uses the event requests generated by various peripheral modules as source signals to connect (link) them to different modules, allowing direct cooperation between the modules without central processing unit (CPU) intervention. The conceptual diagram below illustrates

a potential setup where a pin interrupt triggers a timer which later triggers an ADC conversion and CTSU scan, while at the same time a serial communication interrupt automatically starts a data transfer. These tasks would be automatically handled without the need for polling or interrupt management.

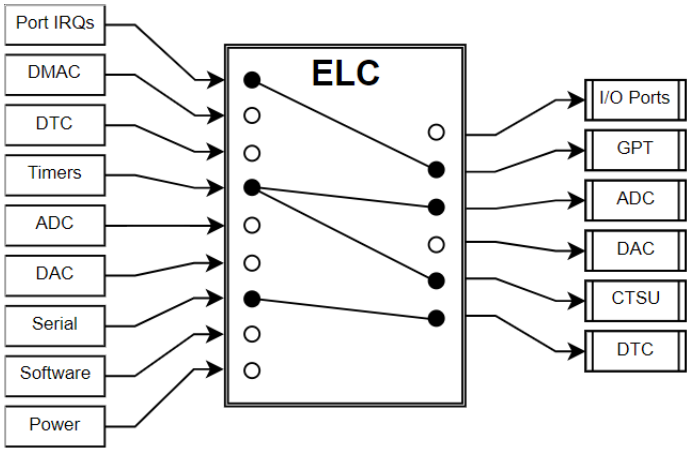



Figure 115: Event Link Controller Conceptual Diagram

In essence, the ELC is an array of multiplexers to route a wide variety of interrupt signals to a subset of peripheral functions. Events are linked by setting the multiplexer for the desired function to the desired signal (through R_ELC_LinkSet). The diagram below illustrates one peripheral output of the ELC. In this example, a conversion start is triggered for ADC0 Group A when the GPT0 counter overflows:

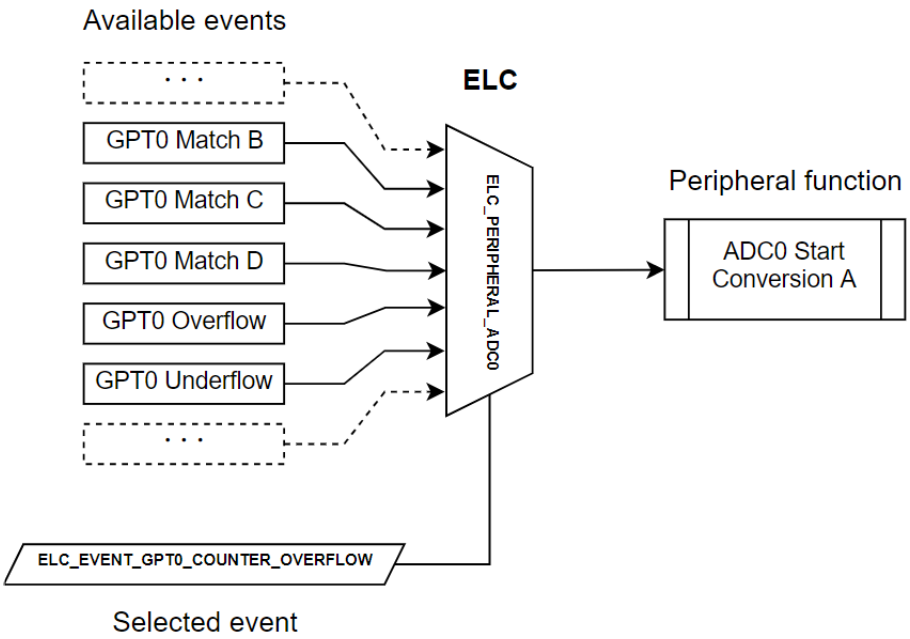



Figure 116: ELC Example

### Features

The ELC HAL module can perform the following functions:

- Initialize the ELC to a pre-defined set of links
- Create an event link between two blocks
- Break an event link between two blocks
- Generate one of two software events that interrupt the CPU
- Globally enable or disable event links

A variety of functions can be activated via events, including:

- General-purpose timer (GPT) control
- ADC and DAC conversion start
- Synchronized I/O port output (ports 1-4 only)
- Capacitive touch unit (CTSU) measurement activation

*Note*

*The available sources and peripherals may differ between devices. A full list of selectable peripherals and events is available in the User's Manual for your device.*
*Some peripherals have specific settings related to ELC event generation and/or reception. Details on how to enable event functionality for each peripheral are located in the usage notes for the related module(s) as well as in the User's Manual for your device.*

# Configuration

*Note*

*Event links will be automatically generated based on the selections made in module properties. To view the currently linked events check the Event Links tab in the RA Configuration editor.*
*Calling R_ELC_Open followed by R_ELC_Enable will automatically link all events shown in the Event Links tab.*

To manually link an event to a peripheral at runtime perform the following steps:

1. Configure the operation of the destination peripheral (including any configuration necessary to receive events)
2. Use R_ELC_LinkSet to set the desired event link to the peripheral
3. Use R_ELC_Enable to enable transmission of event signals
4. Configure the signaling module to output the desired event (typically an interrupt)

To disable the event, either use R_ELC_LinkBreak to clear the link for a specific event or R_ELC_Disable to globally disable event linking.

*Note*

*The ELC module needs no pin, clocking or interrupt configuration; it is merely a mechanism to connect signals between peripherals. However, when linking I/O Ports via the ELC the relevant I/O pins need to be configured as inputs or outputs.*

### Build Time Configurations for r_elc

The following build time configurations are defined in fsp_cfg/r_elc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > System > ELC Driver on r_elc

This module can be added to the Stacks tab via New Stack > Driver > System > ELC Driver on r_elc:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | ELC instance name must be g_elc to match elc_cfg_t data structure created in elc_data.c | g_elc | Module name. |

# Usage Notes

### Limitations

Developers should be aware of the following limitations when using the ELC:

- To link events it is necessary for the ELC and the related modules to be enabled. The ELC cannot operate if the related modules are in the module stop state or the MCU is in a low power consumption mode for which the module is stopped.
- If two modules are linked across clock domains there may be a 1 to 2 cycle delay between event signaling and reception. The delay timing is based on the frequency of the slowest clock.

# Examples

### Basic Example

Below is a basic example of minimal use of event linking in an application.

```
/* This struct is automatically generated based on the events configured by
peripherals in the RA Configuration editor. */
static const elc_cfg_t g_elc_cfg =
{
    .link[ELC_PERIPHERAL_GPT_A]   = ELC_EVENT_ICU_IRQ0,
    .link[ELC_PERIPHERAL_IOPORT1] = ELC_EVENT_GPT0_COUNTER_OVERFLOW
};
void elc_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the software and sets the links defined in the control structure. */
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Create or modify a link between a peripheral function and an event source. */
```

```
    err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_ADC0,
ELC_EVENT_GPT0_COUNTER_OVERFLOW);
    handle_error(err);
 /* Globally enable event linking in the ELC. */
    err = R_ELC_Enable(&g_elc_ctrl);
    handle_error(err);
}
```

### Software-Generated Events

This example demonstrates how to use a software-generated event to signal a peripheral. This can be useful when the desired event source is not supported by the ELC hardware.

```
/* Interrupt handler for peripheral event not supported by the ELC */
void peripheral_isr (void)
{
 fsp_err_t err;
 /* Generate an event signal through software to the linked peripheral. */
    err = R_ELC_SoftwareEventGenerate(&g_elc_ctrl, ELC_SOFTWARE_EVENT_0);
    handle_error(err);
}
void elc_software_event (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the module. */
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Link ADC0 conversion start to software event 0. */
    err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_ADC0,
ELC_EVENT_ELC_SOFTWARE_EVENT_0);
    handle_error(err);
 while (true)
    {
 /* Application code here. */
```

```
    }
}
```

## Data Structures

| | |
|---|---|
| struct | elc_instance_ctrl_t |

## Data Structure Documentation

### ◆ elc_instance_ctrl_t

| struct elc_instance_ctrl_t |
|---|
| ELC private control block. DO NOT MODIFY. Initialization occurs when R_ELC_Open() is called. |

## Function Documentation

### ◆ R_ELC_Open()

| fsp_err_t R_ELC_Open ( elc_ctrl_t *const *p_ctrl*, elc_cfg_t const *const *p_cfg* ) |
|---|

Initialize all the links in the Event Link Controller. Implements elc_api_t::open

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 or clear an individual link use R_ELC_LinkSet and R_ELC_LinkBreak respectively.

Example:

```
/* Initializes the software and sets the links defined in the control structure. */

   err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful |
| FSP_ERR_ASSERTION | p_ctrl or p_cfg was NULL |
| FSP_ERR_ALREADY_OPEN | The module is currently open |

### ◆ R_ELC_Close()

fsp_err_t R_ELC_Close ( elc_ctrl_t *const *p_ctrl*)

Globally disable ELC linking. Implements elc_api_t::close

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The ELC was successfully disabled |
| FSP_ERR_ASSERTION | p_ctrl was NULL |
| FSP_ERR_NOT_OPEN | The module has not been opened |

### ◆ R_ELC_SoftwareEventGenerate()

fsp_err_t R_ELC_SoftwareEventGenerate ( elc_ctrl_t *const *p_ctrl*, elc_software_event_t *event_number* )

Generate a software event in the Event Link Controller. Implements elc_api_t::softwareEventGenerate

Example:

```
/* Generate an event signal through software to the linked peripheral. */
   err = R_ELC_SoftwareEventGenerate(&g_elc_ctrl, ELC_SOFTWARE_EVENT_0);
   handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful |
| FSP_ERR_ASSERTION | Invalid event number or p_ctrl was NULL |
| FSP_ERR_NOT_OPEN | The module has not been opened |

### ◆ R_ELC_LinkSet()

fsp_err_t R_ELC_LinkSet ( elc_ctrl_t *const *p_ctrl*, elc_peripheral_t *peripheral*, elc_event_t *signal* )

Create a single event link. Implements elc_api_t::linkSet

Example:

```
/* Create or modify a link between a peripheral function and an event source. */
   err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_ADC0,
ELC_EVENT_GPT0_COUNTER_OVERFLOW);
   handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful |
| FSP_ERR_ASSERTION | p_ctrl was NULL |
| FSP_ERR_NOT_OPEN | The module has not been opened |

### ◆ R_ELC_LinkBreak()

fsp_err_t R_ELC_LinkBreak ( elc_ctrl_t *const *p_ctrl*, elc_peripheral_t *peripheral* )

Break an event link. Implements elc_api_t::linkBreak

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Event link broken |
| FSP_ERR_ASSERTION | p_ctrl was NULL |
| FSP_ERR_NOT_OPEN | The module has not been opened |

### ◆ R_ELC_Enable()

fsp_err_t R_ELC_Enable ( elc_ctrl_t *const *p_ctrl*)

Enable the operation of the Event Link Controller. Implements elc_api_t::enable

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ELC enabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL |
| FSP_ERR_NOT_OPEN | The module has not been opened |

### ◆ R_ELC_Disable()

fsp_err_t R_ELC_Disable ( elc_ctrl_t *const *p_ctrl*)

Disable the operation of the Event Link Controller. Implements elc_api_t::disable

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ELC disabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL |
| FSP_ERR_NOT_OPEN | The module has not been opened |

### ◆ R_ELC_VersionGet()

fsp_err_t R_ELC_VersionGet ( fsp_version_t *const *p_version*)

Get the driver version based on compile time macros. Implements elc_api_t::versionGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | p_version is NULL. |

## 4.2.18 Ethernet (r_ether)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_ETHER_Open (ether_ctrl_t *const p_ctrl, ether_cfg_t const *const p_cfg)<br>After ETHERC, EDMAC and PHY-LSI are reset in software, an auto negotiation of PHY-LSI is begun. Afterwards, the link signal change interrupt is permitted. Implements ether_api_t::open. More... |
| fsp_err_t | R_ETHER_Close (ether_ctrl_t *const p_ctrl)<br>Disables interrupts. Removes power and releases hardware lock. Implements ether_api_t::close. More... |
| fsp_err_t | R_ETHER_Read (ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t *const length_bytes) |

| | |
|---|---|
| | Receive Ethernet frame. Receives data to the location specified by the pointer to the receive buffer. In zero copy mode, the address of the receive buffer is returned. In non zero copy mode, the received data in the internal buffer is copied to the pointer passed by the argument. Implements ether_api_t::read. More... |
| fsp_err_t | R_ETHER_BufferRelease (ether_ctrl_t *const p_ctrl)<br>Move to the next buffer in the circular receive buffer list. Implements ether_api_t::bufferRelease. More... |
| fsp_err_t | R_ETHER_Write (ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t const frame_length)<br>Transmit Ethernet frame. Transmits data from the location specified by the pointer to the transmit buffer, with the data size equal to the specified frame length. In the non zero copy mode, transmits data after being copied to the internal buffer. Implements ether_api_t::write. More... |
| fsp_err_t | R_ETHER_LinkProcess (ether_ctrl_t *const p_ctrl)<br>The Link up processing, the Link down processing, and the magic packet detection processing are executed. Implements ether_api_t::linkProcess. More... |
| fsp_err_t | R_ETHER_WakeOnLANEnable (ether_ctrl_t *const p_ctrl)<br>The setting of ETHERC is changed from normal sending and receiving mode to magic packet detection mode. Implements ether_api_t::wakeOnLANEnable. More... |
| fsp_err_t | R_ETHER_VersionGet (fsp_version_t *const p_version)<br>Provides API and code version in the user provided pointer. Implements ether_api_t::versionGet. More... |

### Detailed Description

Driver for the Ethernet peripheral on RA MCUs. This module implements the Ethernet Interface.

# Overview

This module performs Ethernet frame transmission and reception using an Ethernet controller and an Ethernet DMA controller.

### Features

The Ethernet module supports the following features:

- Transmit/receive processing
- Optional zero-copy buffering
- Callback function with returned event code
- Magic packet detection mode support
- Auto negotiation support
- Flow control support
- Multicast filtering support
- Broadcast filtering support
- Promiscuous mode support

# Configuration

### Build Time Configurations for r_ether

The following build time configurations are defined in fsp_cfg/r_ether_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| The polarity of the link signal output by the PHY-LSI | • Fall -> Rise<br>• Rise -> Fall | Fall -> Rise | Specify the polarity of the link signal output by the PHY-LSI. When 0 is specified, link-up and link-down correspond respectively to the fall and rise of the LINKSTA signal. When 1 is specified, link-up and link-down correspond respectively to the rise and fall of the LINKSTA signal. |
| The link status is detected by LINKSTA signal | • Unused<br>• Used | Unused | Use LINKSTA signal for detect link status changes 0 = unused (use PHY-LSI status register) 1 = use (use LINKSTA signal) |

### Configurations for Driver > Network > Ethernet Driver on r_ether

This module can be added to the Stacks tab via New Stack > Driver > Network > Ethernet Driver on r_ether:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General > Name | Name must be a valid | g_ether0 | Module name. |

| | | | |
|---|---|---|---|
| | C symbol | | |
| General > Channel | 0 | 0 | Select the ether channel number. |
| General > MAC address | Must be a valid MAC address | 00:11:22:33:44:55 | MAC address of this channel. |
| General > Zero-copy Mode | • Disable<br>• Enable | Disable | Enable or disable zero-copy mode. |
| General > Flow control functionality | • Disable<br>• Enable | Disable | Enable or disable flow control. |
| Filters > Multicast Mode | • Disable<br>• Enable | Enable | Enable or disable multicast frame reception. |
| Filters > Promiscuous Mode | • Disable<br>• Enable | Disable | Enable this option to receive packets addressed to other NICs. |
| Filters > Broadcast filter | Must be a valid non-negative integer with maximum configurable value of 65535. | 0 | Limit of the number of broadcast frames received continuously |
| Buffers > Number of TX buffer | Must be an integer from 1 to 8 | 1 | Number of transmit buffers |
| Buffers > Number of RX buffer | Must be an integer from 1 to 8 | 1 | Number of receive buffers |
| Buffers > Buffer size | Must be at least 1514 which is the maximum Ethernet frame size | 1514 | Size of Ethernet buffer |
| Interrupts > Interrupt priority | MCU Specific Options | | Select the EDMAC interrupt priority. |
| Interrupts > Callback | Name must be a valid C symbol | NULL | Callback provided when an ISR occurs |

### Interrupt Configuration

The first R_ETHER_Open function call sets EINT interrupts. The user could provide callback function which would be invoked when EINT interrupt handler has been completed. The callback arguments will contain information about a channel number, the ETHERC and EDMAC status, the event code, and a pointer to the user defined context.

### Callback Configuration

The user could provide callback function which would be invoked when either a magic packet or a link signal change is detected. When the callback function is called, a variable in which the channel number for which the detection occurred and a constant shown in Table 2.4 are stored is passed as an argument. If the value of this argument is to be used outside the callback function, its value

should be copied into, for example, a global variable.

### Clock Configuration

The ETHER clock is derived from the following peripheral clock on each device.

| MCU | Peripheral Clock |
|---|---|
| RA6M2 | PCLKA |
| RA6M3 | PCLKA |

*Note*

1. *When using ETHERC, the PCLKA frequency is in the range 12.5 MHz <= PCLKA <= 120 MHz.*
2. *When using ETHERC, PCLKA = ICLK.*

### Pin Configuration

To use the Ethernet module, input/output signals of the peripheral function have to be allocated to pins with the multi-function pin controller (MPC). Please perform the pin setting before calling the R_ETHER_Open function.

## Usage Notes

### Ethernet Frame Format

The Ethernet module supports the Ethernet II/IEEE 802.3 frame format.

### Frame Format for Data Transmission and Reception

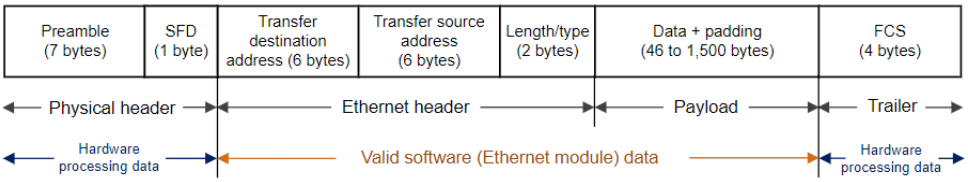



Figure 117: Frame Format Image

The preamble and SFD signal the start of an Ethernet frame. The FCS contains the CRC of the Ethernet frame and is calculated on the transmitting side. When data is received the CRC value of the frame is calculated in hardware, and the Ethernet frame is discarded if the values do not match. When the hardware determines that the data is normal, the valid range of receive data is: (transmission destination address) + (transmission source address) + (length/type) + (data).

### PAUSE Frame Format

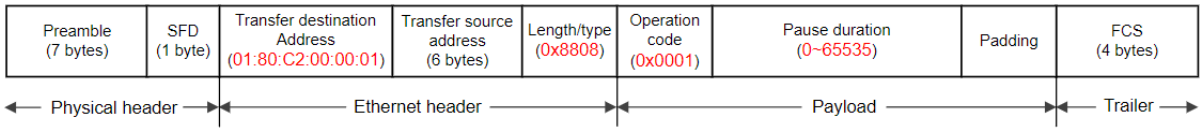



Figure 118: Pause Frame Format Image

The transmission destination address is specified as 01:80:C2:00:00:01 (a multicast address reserved for PAUSE frames). At the start of the payload the length/type is specified as 0x8808 and the operation code as 0x0001. The pause duration in the payload is specified by the value of the automatic PAUSE (AP) bits in the automatic PAUSE frame setting register (APR), or the manual PAUSE time setting (MP) bits in the manual PAUSE frame setting register (MPR).

### Magic Packet Frame Format

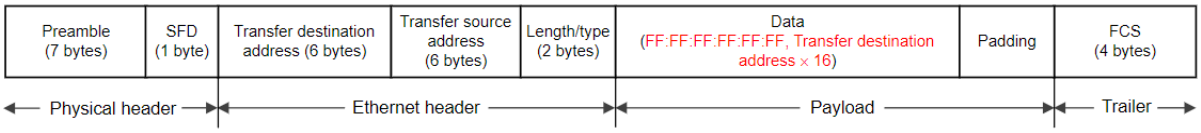



Figure 119: Magic Packet Frame Format Image

In a Magic Packet, the value FF:FF:FF:FF:FF:FF followed by the transmission destination address repeated 16 times is inserted somewhere in the Ethernet frame data.

### Limitations

The Ethernet Driver has several alignment constraints:

- 16-byte alignment for the descriptor
- 32-byte aligned write buffer for R_ETHER_Write when zero copy mode is enabled

# Examples

### ETHER Basic Example

This is a basic example of minimal use of the ETHER in an application.

*Note*

*In this example zero-copy mode is disabled and there are no restrictions on buffer alignment.*

```
#define ETHER_EXAMPLE_MAXIMUM_ETHERNET_FRAME_SIZE (1514)

#define ETHER_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE (60)

#define ETHER_EXAMPLE_SOURCE_MAC_ADDRESS 0x74, 0x90, 0x50, 0x00, 0x79, 0x01

#define ETHER_EXAMPLE_DESTINATION_MAC_ADDRESS 0x74, 0x90, 0x50, 0x00, 0x79, 0x02

#define ETHER_EXAMPLE_FRAME_TYPE 0x00, 0x2E

#define ETHER_EXAMPLE_PAYLOAD 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \


/* Receive data buffer */
```

```
static uint8_t gp_read_buffer[ETHER_EXAMPLE_MAXIMUM_ETHERNET_FRAME_SIZE] = {0};
/* Transmit data buffer */
static uint8_t gp_send_data[ETHER_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE] =
{
    ETHER_EXAMPLE_DESTINATION_MAC_ADDRESS, /* Destination MAC address */
    ETHER_EXAMPLE_SOURCE_MAC_ADDRESS,      /* Source MAC address */
    ETHER_EXAMPLE_FRAME_TYPE,             /* Type field */
    ETHER_EXAMPLE_PAYLOAD                  /* Payload value (46byte) */
};
void ether_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Source MAC Address */
 static uint8_t mac_address_source[6] = {ETHER_EXAMPLE_SOURCE_MAC_ADDRESS};
    uint32_t read_data_size = 0;
    g_ether0_cfg.p_mac_address = mac_address_source;
 /* Open the ether instance with initial configuration. */
    err = R_ETHER_Open(&g_ether0_ctrl, &g_ether0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 do
    {
 /* When the Ethernet link status read from the PHY-LSI Basic Status register is link-
up,
  * Initializes the module and make auto negotiation. */
       err = R_ETHER_LinkProcess(&g_ether0_ctrl);
    } while (FSP_SUCCESS != err);
 /* Transmission is non-blocking. */
 /* User data copy to internal buffer and is transferred by DMA in the background. */
    err = R_ETHER_Write(&g_ether0_ctrl, (void *) gp_send_data, sizeof(gp_send_data));
    handle_error(err);
 /* received data copy to user buffer from internal buffer. */
    err = R_ETHER_Read(&g_ether0_ctrl, (void *) gp_read_buffer, &read_data_size);
    handle_error(err);
```

```
 /* Disable transmission and receive function and close the ether instance. */
 R_ETHER_Close(&g_ether0_ctrl);
}
```

### ETHER Advanced Example

The example demonstrates using send and receive function in zero copy mode. Transmit buffers must be 32-byte aligned and the receive buffer must be released once its contents have been used.

```
#define ETHER_EXAMPLE_FLAG_ON (1U)
#define ETHER_EXAMPLE_FLAG_OFF (0U)
#define ETHER_EXAMPLE_ETHER_ISR_EE_FR_MASK (1UL << 18)
#define ETHER_EXAMPLE_ETHER_ISR_EE_TC_MASK (1UL << 21)
#define ETHER_EXAMPLE_ETHER_ISR_EC_MPD_MASK (1UL << 1)
#define ETHER_EXAMPLE_ALIGNMENT_32_BYTE (32)
static volatile uint32_t g_example_receive_complete = 0;
static volatile uint32_t g_example_transfer_complete = 0;
static volatile uint32_t g_example_magic_packet_done = 0;
/* The data buffer must be 32-byte aligned when using zero copy mode. */
static uint8_t gp_send_data_nocopy[ETHER_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE]
BSP_ALIGN_VARIABLE(32) =
{
    ETHER_EXAMPLE_DESTINATION_MAC_ADDRESS, /* Destination MAC address */
    ETHER_EXAMPLE_SOURCE_MAC_ADDRESS,      /* Source MAC address */
    ETHER_EXAMPLE_FRAME_TYPE,             /* Type field */
    ETHER_EXAMPLE_PAYLOAD                  /* Payload value (46byte) */
};
void ether_example_callback (ether_callback_args_t * p_args) {
 switch (p_args->event)
    {
 case ETHER_EVENT_INTERRUPT:
      {
 if (ETHER_EXAMPLE_ETHER_ISR_EC_MPD_MASK == (p_args->status_ecsr &
ETHER_EXAMPLE_ETHER_ISR_EC_MPD_MASK))
      {
```

```
                g_example_magic_packet_done = ETHER_EXAMPLE_FLAG_ON;
            }
 if (ETHER_EXAMPLE_ETHER_ISR_EE_TC_MASK == (p_args->status_eesr &
ETHER_EXAMPLE_ETHER_ISR_EE_TC_MASK))
      {
                g_example_transfer_complete = ETHER_EXAMPLE_FLAG_ON;
            }
 if (ETHER_EXAMPLE_ETHER_ISR_EE_FR_MASK == (p_args->status_eesr &
ETHER_EXAMPLE_ETHER_ISR_EE_FR_MASK))
      {
                g_example_receive_complete = ETHER_EXAMPLE_FLAG_ON;
            }
 break;
       }
 default:
      {
       }
    }
}
void ether_advanced_example (void) {
 fsp_err_t err = FSP_SUCCESS;
 /* Source MAC Address */
 static uint8_t mac_address_source[6] = {ETHER_EXAMPLE_SOURCE_MAC_ADDRESS};
 static uint8_t * p_read_buffer_nocopy;
    uint32_t         read_data_size = 0;
    g_ether0_cfg.p_mac_address = mac_address_source;
    g_ether0_cfg.zerocopy   = ETHER_ZEROCOPY_ENABLE;
    g_ether0_cfg.p_callback = (void (*)(ether_callback_args_t
*))ether_example_callback;
 /* Open the ether instance with initial configuration. */
    err = R_ETHER_Open(&g_ether0_ctrl, &g_ether0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 do
```

```
    {
 /* When the Ethernet link status read from the PHY-LSI Basic Status register is link-
up,
  * Initializes the module and make auto negotiation. */
       err = R_ETHER_LinkProcess(&g_ether0_ctrl);
    } while (FSP_SUCCESS != err);
 /* Set user buffer to TX descriptor and enable transmission. */
    err = R_ETHER_Write(&g_ether0_ctrl, (void *) gp_send_data_nocopy, sizeof
(gp_send_data_nocopy));
 if (FSP_SUCCESS == err)
    {
 /* Wait for the transmission to complete. */
 /* Data array should not change in zero copy mode until transfer complete. */
 while (ETHER_EXAMPLE_FLAG_ON != g_example_transfer_complete)
      {
            ;
       }
    }
 /* Get receive buffer from RX descriptor. */
    err = R_ETHER_Read(&g_ether0_ctrl, (void *) &p_read_buffer_nocopy,
&read_data_size);
    handle_error(err);
 /* Process received data here */
 /* Release receive buffer to RX descriptor. */
    err = R_ETHER_BufferRelease(&g_ether0_ctrl);
    handle_error(err);
 /* Disable transmission and receive function and close the ether instance. */
 R_ETHER_Close(&g_ether0_ctrl);
}
```

## Data Structures

| | |
|---|---|
| struct | ether_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | ether_previous_link_status_t |

| | |
|---|---|
| enum | ether_link_change_t |
| enum | ether_magic_packet_t |
| enum | ether_link_establish_status_t |

## Data Structure Documentation

### ◆ ether_instance_ctrl_t

| struct ether_instance_ctrl_t | | |
|---|---|---|
| ETHER control block. DO NOT INITIALIZE. Initialization occurs when ether_api_t::open is called. | | |
| Data Fields | | |
| uint32_t | open | Used to determine if the channel is configured. |
| ether_cfg_t const * | p_ether_cfg | Pointer to initial configurations. |
| ether_instance_descriptor_t * | p_rx_descriptor | Pointer to the currently referenced transmit descriptor. |
| ether_instance_descriptor_t * | p_tx_descriptor | Pointer to the currently referenced receive descriptor. |
| void * | p_reg_etherc | Base register of ethernet controller for this channel. |
| void * | p_reg_edmac | Base register of EDMA controller for this channel. |
| ether_previous_link_status_t | previous_link_status | Previous link status. |
| ether_link_change_t | link_change | status of link change |
| ether_magic_packet_t | magic_packet | status of magic packet detection |
| ether_link_establish_status_t | link_establish_status | Current Link status. |

## Enumeration Type Documentation

### ◆ ether_previous_link_status_t

| enum ether_previous_link_status_t | |
|---|---|
| Enumerator | |
| ETHER_PREVIOUS_LINK_STATUS_DOWN | Previous link status is down. |
| ETHER_PREVIOUS_LINK_STATUS_UP | Previous link status is up. |

### ◆ ether_link_change_t

| enum ether_link_change_t | |
|---|---|
| Enumerator | |
| ETHER_LINK_CHANGE_NO_CHANGE | Link status is no change. |
| ETHER_LINK_CHANGE_LINK_DOWN | Link status changes to down. |
| ETHER_LINK_CHANGE_LINK_UP | Link status changes to up. |

### ◆ ether_magic_packet_t

| enum ether_magic_packet_t | |
|---|---|
| Enumerator | |
| ETHER_MAGIC_PACKET_NOT_DETECTED | Magic packet is not detected. |
| ETHER_MAGIC_PACKET_DETECTED | Magic packet is detected. |

### ◆ ether_link_establish_status_t

| enum ether_link_establish_status_t | |
|---|---|
| Enumerator | |
| ETHER_LINK_ESTABLISH_STATUS_DOWN | Link establish status is down. |
| ETHER_LINK_ESTABLISH_STATUS_UP | Link establish status is up. |

## Function Documentation

### ◆ R_ETHER_Open()

fsp_err_t R_ETHER_Open ( ether_ctrl_t *const *p_ctrl*, ether_cfg_t const *const *p_cfg* )

After ETHERC, EDMAC and PHY-LSI are reset in software, an auto negotiation of PHY-LSI is begun. Afterwards, the link signal change interrupt is permitted. Implements ether_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |
| FSP_ERR_ETHER_ERROR_PHY_COMMUNICATION | Initialization of PHY-LSI failed. |
| FSP_ERR_INVALID_CHANNEL | Invalid channel number is given. |
| FSP_ERR_INVALID_POINTER | Pointer to MAC address is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Interrupt is not enabled. |
| FSP_ERR_ETHER_PHY_ERROR_LINK | Initialization of PHY-LSI failed. |

### ◆ R_ETHER_Close()

fsp_err_t R_ETHER_Close ( ether_ctrl_t *const *p_ctrl*)

Disables interrupts. Removes power and releases hardware lock. Implements ether_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_ETHER_Read()

fsp_err_t R_ETHER_Read ( ether_ctrl_t *const *p_ctrl*, void *const *p_buffer*, uint32_t *const *length_bytes* )

Receive Ethernet frame. Receives data to the location specified by the pointer to the receive buffer. In zero copy mode, the address of the receive buffer is returned. In non zero copy mode, the received data in the internal buffer is copied to the pointer passed by the argument. Implements ether_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ETHER_ERROR_NO_DATA | There is no data in receive buffer. |
| FSP_ERR_ETHER_ERROR_LINK | Auto-negotiation is not completed, and reception is not enabled. |
| FSP_ERR_ETHER_ERROR_MAGIC_PACKET_MODE | As a Magic Packet is being detected, transmission and reception is not enabled. |
| FSP_ERR_ETHER_ERROR_FILTERING | Multicast Frame filter is enable, and Multicast Address Frame is received. |
| FSP_ERR_INVALID_POINTER | Value of the pointer is NULL. |

### ◆ R_ETHER_BufferRelease()

fsp_err_t R_ETHER_BufferRelease ( ether_ctrl_t *const *p_ctrl*)

Move to the next buffer in the circular receive buffer list. Implements ether_api_t::bufferRelease.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_ETHER_ERROR_LINK | Auto-negotiation is not completed, and reception is not enabled. |
| FSP_ERR_ETHER_ERROR_MAGIC_PACKET_MODE | As a Magic Packet is being detected, transmission and reception is not enabled. |

### ◆ R_ETHER_Write()

fsp_err_t R_ETHER_Write ( ether_ctrl_t *const *p_ctrl*, void *const *p_buffer*, uint32_t const *frame_length* )

Transmit Ethernet frame. Transmits data from the location specified by the pointer to the transmit buffer, with the data size equal to the specified frame length. In the non zero copy mode, transmits data after being copied to the internal buffer. Implements ether_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ETHER_ERROR_LINK | Auto-negotiation is not completed, and reception is not enabled. |
| FSP_ERR_ETHER_ERROR_MAGIC_PACKET_MODE | As a Magic Packet is being detected, transmission and reception is not enabled. |
| FSP_ERR_ETHER_ERROR_TRANSMIT_BUFFER_FULL | Transmit buffer is not empty. |
| FSP_ERR_INVALID_POINTER | Value of the pointer is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Value of the send frame size is out of range. |

### ◆ R_ETHER_LinkProcess()

fsp_err_t R_ETHER_LinkProcess ( ether_ctrl_t *const *p_ctrl*)

The Link up processing, the Link down processing, and the magic packet detection processing are executed. Implements ether_api_t::linkProcess.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Link is up. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ETHER_ERROR_LINK | Link is down. |
| FSP_ERR_ETHER_ERROR_PHY_COMMUNICATION | When reopening the PHY interface initialization of the PHY-LSI failed. |
| FSP_ERR_ALREADY_OPEN | When reopening the PHY interface it was already opened. |
| FSP_ERR_INVALID_CHANNEL | When reopening the PHY interface an invalid channel was passed. |
| FSP_ERR_INVALID_POINTER | When reopening the PHY interface the MAC address pointer was NULL. |
| FSP_ERR_INVALID_ARGUMENT | When reopening the PHY interface the interrupt was not enabled. |
| FSP_ERR_ETHER_PHY_ERROR_LINK | Initialization of the PHY-LSI failed. |

### ◆ R_ETHER_WakeOnLANEnable()

fsp_err_t R_ETHER_WakeOnLANEnable ( ether_ctrl_t *const *p_ctrl*)

The setting of ETHERC is changed from normal sending and receiving mode to magic packet detection mode. Implements ether_api_t::wakeOnLANEnable.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ETHER_ERROR_LINK | Auto-negotiation is not completed, and reception is not enabled. |
| FSP_ERR_ETHER_PHY_ERROR_LINK | Initialization of PHY-LSI failed. |

### ◆ R_ETHER_VersionGet()

| __INLINE fsp_err_t R_ETHER_VersionGet ( fsp_version_t *const *p_version*) |
|---|
| Provides API and code version in the user provided pointer. Implements ether_api_t::versionGet.<br><br>**Return values**<br>FSP_SUCCESS — Version information stored in provided p_version.<br>FSP_ERR_ASSERTION — p_version is NULL. |

## 4.2.19 Ethernet PHY (r_ether_phy)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_ETHER_PHY_Open (ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg)<br><br>Resets Ethernet PHY device. Implements ether_phy_api_t::open. More... |
| fsp_err_t | R_ETHER_PHY_Close (ether_phy_ctrl_t *const p_ctrl)<br><br>Close Ethernet PHY device. Implements ether_phy_api_t::close. More... |
| fsp_err_t | R_ETHER_PHY_StartAutoNegotiate (ether_phy_ctrl_t *const p_ctrl)<br><br>Starts auto-negotiate. Implements ether_phy_api_t::startAutoNegotiate. More... |
| fsp_err_t | R_ETHER_PHY_LinkPartnerAbilityGet (ether_phy_ctrl_t *const p_ctrl, uint32_t *const p_line_speed_duplex, uint32_t *const p_local_pause, uint32_t *const p_partner_pause)<br><br>Reports the other side's physical capability. Implements ether_phy_api_t::linkPartnerAbilityGet. More... |
| fsp_err_t | R_ETHER_PHY_LinkStatusGet (ether_phy_ctrl_t *const p_ctrl)<br><br>Returns the status of the physical link. Implements |

| | |
|---|---|
| | ether_phy_api_t::linkStatusGet. More... |
| fsp_err_t | R_ETHER_PHY_VersionGet (fsp_version_t *const p_version)<br>Provides API and code version in the user provided pointer. Implements ether_phy_api_t::versionGet. More... |

### Detailed Description

The Ethernet PHY module (r_ether_phy) provides an API for standard Ethernet PHY communications applications that use the ETHERC peripheral. It implements the Ethernet PHY Interface.

# Overview

The Ethernet PHY module is used to setup and manage an external Ethernet PHY device for use with the on-chip Ethernet Controller (ETHERC) peripheral. It performs auto-negotiation to determine the optimal connection parameters between link partners. Once initialized the connection between the external PHY and the onboard controller is automatically managed in hardware.

### Features

The Ethernet PHY module supports the following features:

- Auto negotiation support
- Flow control support
- Link status check support

# Configuration

### Build Time Configurations for r_ether_phy

The following build time configurations are defined in fsp_cfg/r_ether_phy_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Select PHY | • Default<br>• Other<br>• KSZ8091RNB<br>• KSZ8041<br>• DP83620 | Default | Select PHY chip to use. Selecting 'Default' will automatically choose the correct option when using a Renesas development board. |
| Use Reference Clock | • Default<br>• Enabled<br>• Disabled | Enabled | Select whether to use the RMII reference clock. Selecting 'Default' will |

| | | | |
|---|---|---|---|
| | | | automatically choose the correct option when using a Renesas development board. |

### Configurations for Driver > Network > Ethernet Driver on r_ether_phy

This module can be added to the Stacks tab via New Stack > Driver > Network > Ethernet Driver on r_ether_phy:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_ether_phy0 | Module name. |
| Channel | • 0<br>• 1 | 0 | Select the Ethernet controller channel number. |
| PHY-LSI Address | Specify a value between 0 and 31. | 0 | Specify the address of the PHY-LSI used. |
| PHY-LSI Reset Completion Timeout | Specify a value between 1 and 0xFFFFFFFF. | 0x00020000 | Specify the number of times to read the PHY-LSI control register while waiting for reset completion. This value should be adjusted experimentally based on the PHY-LSI used. |
| MII/RMII Register Access Wait-time | Specify a value between 1 and 0x7FFFFFFF. | 8 | Specify the bit timing for MII/RMII register accesses during PHY initialization. This value should be adjusted experimentally based on the PHY-LSI used. |
| Flow Control | • Disable<br>• Enable | Disable | Select whether to enable or disable flow control. |

# Usage Notes

*Note*

*See the example below for details on how to initialize the Ethernet PHY module.*

### Limitations

- The r_ether_phy module may need to be customized for PHY devices other than the ones currently supported (KSZ8091RNB, KSZ8041 and DP83620). Use the existing code as a starting point for creating a custom implementation.

# Examples

### ETHER PHY Basic Example

This is a basic example of minimal use of the ETHER PHY in an application.

```
void ether_phy_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    g_ether_phy0_ctrl.open   = 0U;
    g_ether_phy0_cfg.channel = 0;
 /* Initializes the module. */
    err = R_ETHER_PHY_Open(&g_ether_phy0_ctrl, &g_ether_phy0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Start auto negotiation. */
    err = R_ETHER_PHY_StartAutoNegotiate(&g_ether_phy0_ctrl);
    handle_error(err);
 /* Polling until link is established. */
 while (FSP_SUCCESS != R_ETHER_PHY_LinkStatusGet(&g_ether_phy0_ctrl))
    {
 /* Do nothing */
    }
 /* Get link partner ability from phy interface. */
    err = R_ETHER_PHY_LinkPartnerAbilityGet(&g_ether_phy0_ctrl,
                                            &g_ether_phy0_line_speed_duplex,
                                            &g_ether_phy0_local_pause,
                                            &g_ether_phy0_partner_pause);
    handle_error(err);
 /* Check current link status. */
    err = R_ETHER_PHY_LinkStatusGet(&g_ether_phy0_ctrl);
    handle_error(err);
}
```

### Data Structures

| | |
|---|---|
| struct | ether_phy_instance_ctrl_t |

## Data Structure Documentation

### ◆ ether_phy_instance_ctrl_t

| struct ether_phy_instance_ctrl_t | | |
|---|---|---|
| ETHER PHY control block. DO NOT INITIALIZE. Initialization occurs when ether_phy_api_t::open is called. | | |
| Data Fields | | |
| uint32_t | open | Used to determine if the channel is configured. |
| ether_phy_cfg_t const * | p_ether_phy_cfg | Pointer to initial configurations. |
| volatile uint32_t * | p_reg_pir | Pointer to ETHERC peripheral registers. |
| uint32_t | local_advertise | Capabilities bitmap for local advertising. |

## Function Documentation

### ◆ R_ETHER_PHY_Open()

fsp_err_t R_ETHER_PHY_Open ( ether_phy_ctrl_t *const *p_ctrl*, ether_phy_cfg_t const *const *p_cfg* )

Resets Ethernet PHY device. Implements ether_phy_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER_PHY control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |
| FSP_ERR_INVALID_CHANNEL | Invalid channel number is given. |
| FSP_ERR_INVALID_POINTER | Pointer to p_cfg is NULL. |
| FSP_ERR_TIMEOUT | PHY-LSI Reset wait timeout. |

### ◆ R_ETHER_PHY_Close()

fsp_err_t R_ETHER_PHY_Close ( ether_phy_ctrl_t *const *p_ctrl*)

Close Ethernet PHY device. Implements ether_phy_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to ETHER_PHY control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_ETHER_PHY_StartAutoNegotiate()

fsp_err_t R_ETHER_PHY_StartAutoNegotiate ( ether_phy_ctrl_t *const *p_ctrl*)

Starts auto-negotiate. Implements ether_phy_api_t::startAutoNegotiate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ETHER_PHY successfully starts auto-negotiate. |
| FSP_ERR_ASSERTION | Pointer to ETHER_PHY control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_ETHER_PHY_LinkPartnerAbilityGet()

fsp_err_t R_ETHER_PHY_LinkPartnerAbilityGet ( ether_phy_ctrl_t *const *p_ctrl*, uint32_t *const *p_line_speed_duplex*, uint32_t *const *p_local_pause*, uint32_t *const *p_partner_pause* )

Reports the other side's physical capability. Implements ether_phy_api_t::linkPartnerAbilityGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ETHER_PHY successfully get link partner ability. |
| FSP_ERR_ASSERTION | Pointer to ETHER_PHY control block is NULL. |
| FSP_ERR_INVALID_POINTER | Pointer to arguments are NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_ETHER_PHY_ERROR_LINK | PHY-LSI is not link up. |
| FSP_ERR_ETHER_PHY_NOT_READY | The auto-negotiation isn't completed |

### ◆ R_ETHER_PHY_LinkStatusGet()

fsp_err_t R_ETHER_PHY_LinkStatusGet ( ether_phy_ctrl_t *const *p_ctrl*)

Returns the status of the physical link. Implements ether_phy_api_t::linkStatusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ETHER_PHY successfully get link partner ability. |
| FSP_ERR_ASSERTION | Pointer to ETHER_PHY control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_ETHER_PHY_ERROR_LINK | PHY-LSI is not link up. |

### ◆ R_ETHER_PHY_VersionGet()

__INLINE fsp_err_t R_ETHER_PHY_VersionGet ( fsp_version_t *const *p_version*)

Provides API and code version in the user provided pointer. Implements ether_phy_api_t::versionGet.

**Parameters**

| | | |
|---|---|---|
| [in] | p_version | Version number set here |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version information stored in provided p_version. |
| FSP_ERR_ASSERTION | p_version is NULL. |

## 4.2.20 High-Performance Flash Driver (r_flash_hp)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_FLASH_HP_Open (flash_ctrl_t *const p_api_ctrl, flash_cfg_t const *const p_cfg) |
| fsp_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) |
| fsp_err_t | R_FLASH_HP_Erase (flash_ctrl_t *const p_api_ctrl, uint32_t const |

| | |
|---|---|
| | address, uint32_t const num_blocks) |
| fsp_err_t | R_FLASH_HP_BlankCheck (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t num_bytes, flash_result_t *blank_check_result) |
| fsp_err_t | R_FLASH_HP_Close (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_HP_StatusGet (flash_ctrl_t *const p_api_ctrl, flash_status_t *const p_status) |
| fsp_err_t | R_FLASH_HP_AccessWindowSet (flash_ctrl_t *const p_api_ctrl, uint32_t const start_addr, uint32_t const end_addr) |
| fsp_err_t | R_FLASH_HP_AccessWindowClear (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_HP_IdCodeSet (flash_ctrl_t *const p_api_ctrl, uint8_t const *const p_id_code, flash_id_code_mode_t mode) |
| fsp_err_t | R_FLASH_HP_Reset (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_HP_UpdateFlashClockFreq (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_HP_StartUpAreaSelect (flash_ctrl_t *const p_api_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary) |
| fsp_err_t | R_FLASH_HP_VersionGet (fsp_version_t *const p_version) |
| fsp_err_t | R_FLASH_HP_InfoGet (flash_ctrl_t *const p_api_ctrl, flash_info_t *const p_info) |

### Detailed Description

Driver for the flash memory on RA high-performance MCUs. This module implements the Flash Interface.

# Overview

The Flash HAL module APIs allow an application to write, erase and blank check both the data and ROM flash areas that reside within the MCU. The amount of flash memory available varies across MCU parts.

### Features

The R_FLASH_HP module has the following key features:

- Blocking and non-blocking erasing, writing and blank-checking of data flash.
- Blocking erasing, writing and blank-checking of code flash.
- Callback functions for completion of non-blocking data flash operations.
- Access window (write protection) for ROM Flash, allowing only specified areas of code flash

to be erased or written.
- Boot block-swapping.
- ID code programming support.

# Configuration

### Build Time Configurations for r_flash_hp

The following build time configurations are defined in fsp_cfg/r_flash_hp_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Code Flash Programming Enable | • Enabled<br>• Disabled | Disabled | Controls whether or not code-flash programming is enabled. Disabling reduces the amount of ROM and RAM used by the API. |
| Data Flash Programming Enable | • Enabled<br>• Disabled | Enabled | Controls whether or not data-flash programming is enabled. Disabling reduces the amount of ROM used by the API. |

### Configurations for Driver > Storage > Flash Driver on r_flash_hp

This module can be added to the Stacks tab via New Stack > Driver > Storage > Flash Driver on r_flash_hp:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_flash0 | Module name. |
| Data Flash Background Operation | • Enabled<br>• Disabled | Enabled | Enabling allows Flash API calls that reference data-flash to return immediately, with the operation continuing in the background. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be specified. Callback function called when a dataflash BGO operation completes or |

| | | |
|---|---|---|
| | | errors. |
| Flash Ready Interrupt Priority | MCU Specific Options | Select the flash ready interrupt priority. |
| Flash Error Interrupt Priority | MCU Specific Options | Select the flash error interrupt priority. |

### Clock Configuration

Flash uses FCLK as the clock source depending on the MCU. When writing and erasing the clock source must be at least 4 MHz.

### Pin Configuration

This module does not use I/O pins.

## Usage Notes

Warning

It is highly recommended that the developer reviews sections 5 and 6 of the Flash Memory section of the target MCUs Hardware User's Manual prior to using the r_flash_hp module. In particular, understanding ID Code and Access Window functionality can help avoid unrecoverable flash scenarios.

### Data Flash Background Operation (BGO) Precautions

When using the data flash BGO (Background Operation) mode, you can still access the user ROM, RAM and external memory. 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

Code flash cannot be accessed while writing, erasing or blank checking code flash. Code flash cannot be accessed while modifying the access window, selecting the startup area or setting the ID code. In order to support modifying code flash all supporting code must reside in RAM. This is only done when code flash programming is enabled. 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 code flash. If an interrupt occurs during the code flash operation, then code flash 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 multi-threaded environment, threads running from code flash cannot become active while a code flash operation is in progress.

### Flash Clock (FCLK)

The flash clock source is the clock used by the Flash peripheral in performing all Flash operations. As part of the flash_api_t::open function the Flash clock source is checked will return FSP_ERR_FCLK if it is invalid. Once the Flash API has been opened, if the flash clock source frequency is changed, the flash_api_t::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 (for example, flash_api_t::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 flash_api_t::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 the Flash Ready Interrupt and Flash Error Interrupts must be enabled (assigned a priority).*

### Limitations

- Write operations must be aligned on page boundaries and must be a multiple of the page boundary size.
- Erase operations will erase the entire block the provided address resides in.
- Data flash is better suited for storing data as it can be erased and written to while code is still executing from code flash. Data flash is also guaranteed for a larger number of reprogramming/erasure cycles than code flash.
- Read values of erased data flash blocks are not guaranteed to be 0xFF. Blank check should be used to determine if memory has been erased but not yet programmed.

## Examples

### High-Performance Flash Basic Example

This is a basic example of erasing and writing to data flash and code flash.

```
#define FLASH_DF_BLOCK_0 0x40100000U /* 64 B: 0x40100000 - 0x4010003F */
#define FLASH_CF_BLOCK_8 0x00010000  /* 32 KB: 0x00010000 - 0x00017FFF */
#define FLASH_DATA_BLOCK_SIZE (1024)
#define FLASH_HP_EXAMPLE_WRITE_SIZE 32
uint8_t       g_dest[TRANSFER_LENGTH];
uint8_t       g_src[TRANSFER_LENGTH];
flash_result_t blank_check_result;
void r_flash_hp_basic_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the flash hp instance. */
 fsp_err_t err = R_FLASH_HP_Open(&g_flash_ctrl, &g_flash_cfg);
```

```
    handle_error(err);
 /* Erase 1 block of data flash starting at block 0. */
    err = R_FLASH_HP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
    handle_error(err);
 /* Check if block 0 is erased. */
    err = R_FLASH_HP_BlankCheck(&g_flash_ctrl, FLASH_DF_BLOCK_0,
FLASH_DATA_BLOCK_SIZE, &blank_check_result);
    handle_error(err);
 /* Verify the previously erased area is blank */
 if (FLASH_RESULT_NOT_BLANK == blank_check_result)
    {
       handle_error(FSP_ERR_BLANK_CHECK_FAILED);
    }
 /* Write 32 bytes to the first block of data flash. */
    err = R_FLASH_HP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_HP_EXAMPLE_WRITE_SIZE);
    handle_error(err);
 if (0 != memcmp(g_src, (uint8_t *) FLASH_DF_BLOCK_0, FLASH_HP_EXAMPLE_WRITE_SIZE))
    {
       handle_error(FSP_ERR_WRITE_FAILED);
    }
 /* Disable interrupts to prevent vector table access while code flash is in P/E
mode. */
    __disable_irq();
 /* Erase 1 block of code flash starting at block 10. */
    err = R_FLASH_HP_Erase(&g_flash_ctrl, FLASH_CF_BLOCK_8, 1);
    handle_error(err);
 /* Write 32 bytes to the first block of data flash. */
    err = R_FLASH_HP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_CF_BLOCK_8,
FLASH_HP_EXAMPLE_WRITE_SIZE);
    handle_error(err);
 /* Enable interrupts after code flash operations are complete. */
    __enable_irq();
 if (0 != memcmp(g_src, (uint8_t *) FLASH_CF_BLOCK_8, FLASH_HP_EXAMPLE_WRITE_SIZE))
```

```
    {
       handle_error(FSP_ERR_WRITE_FAILED);
    }
}
```

### High-Performance Flash Advanced Example

This example demonstrates using BGO to do non-blocking operations on the data flash.

```
bool interrupt_called;
flash_event_t flash_event;
static flash_cfg_t g_flash_bgo_example_cfg =
{
    .p_callback    = flash_callback,
    .p_context      = 0,
    .p_extend       = NULL,
    .data_flash_bgo = true,
    .ipl            = 5,
    .irq            = BSP_VECTOR_FLASH_HP_FRDYI_ISR,
};
void r_flash_hp_bgo_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the flash hp instance. */
 fsp_err_t err = R_FLASH_HP_Open(&g_flash_ctrl, &g_flash_bgo_example_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
    interrupt_called = false;
 /* Erase 1 block of data flash starting at block 0. */
    err = R_FLASH_HP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
    handle_error(err);
```

```
while (!interrupt_called)
   {
      ;
   }
if (FLASH_EVENT_ERASE_COMPLETE != flash_event)
   {
      handle_error(FSP_ERR_ERASE_FAILED);
   }
   interrupt_called = false;
/* Write 32 bytes to the first block of data flash. */
   err = R_FLASH_HP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_HP_EXAMPLE_WRITE_SIZE);
   handle_error(err);
flash_status_t status;
/* Wait until the current flash operation completes. */
do
   {
      err = R_FLASH_HP_StatusGet(&g_flash_ctrl, &status);
   } while ((FSP_SUCCESS == err) && (FLASH_STATUS_BUSY == status));
/* If the interrupt wasn't called process the error. */
if (!interrupt_called)
   {
      handle_error(FSP_ERR_WRITE_FAILED);
   }
/* If the event wasn't a write complete process the error. */
if (FLASH_EVENT_WRITE_COMPLETE != flash_event)
   {
      handle_error(FSP_ERR_WRITE_FAILED);
   }
/* Verify the data was written correctly. */
if (0 != memcmp(g_src, (uint8_t *) FLASH_DF_BLOCK_0, FLASH_HP_EXAMPLE_WRITE_SIZE))
   {
      handle_error(FSP_ERR_WRITE_FAILED);
   }
```

```
}
void flash_callback (flash_callback_args_t * p_args)
{
    interrupt_called = true;
    flash_event      = p_args->event;
}
```

### Data Structures

| | |
|---|---|
| struct | flash_hp_instance_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | flash_bgo_operation_t |

## Data Structure Documentation

### ◆ flash_hp_instance_ctrl_t

| | |
|---|---|
| struct flash_hp_instance_ctrl_t | |
| Flash HP instance control block. DO NOT INITIALIZE. | |
| **Data Fields** | |
| uint32_t | opened |
| | To check whether api has been opened or not. |
| | |
| flash_cfg_t const * | p_cfg |
| | User Callback function. |
| | |
| flash_bgo_operation_t | current_operation |
| | Operation in progress, for example, FLASH_OPERATION_CF_ERASE. |
| | |

## Enumeration Type Documentation

### ◆ flash_bgo_operation_t

| |
|---|
| enum flash_bgo_operation_t |
| Possible Flash operation states |

## Function Documentation

### ◆ R_FLASH_HP_Open()

fsp_err_t R_FLASH_HP_Open ( flash_ctrl_t *const *p_api_ctrl*, flash_cfg_t const *const *p_cfg* )

Initializes the high performance flash peripheral. Implements flash_api_t::open.

The Open function initializes the Flash.

Example:

```
/* Open the flash hp instance. */
fsp_err_t err = R_FLASH_HP_Open(&g_flash_ctrl, &g_flash_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful and timer has started. |
| FSP_ERR_ALREADY_OPEN | The flash control block is already open. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl or p_cfg. |
| FSP_ERR_IRQ_BSP_DISABLED | Caller is requesting BGO but the Flash interrupts are not enabled. |
| FSP_ERR_FCLK | FCLK must be a minimum of 4 MHz for Flash operations. |

### ◆ R_FLASH_HP_Write()

fsp_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* )

Writes to the specified Code or Data Flash memory area. Implements flash_api_t::write.

Example:

```
/* Write 32 bytes to the first block of data flash. */
   err = R_FLASH_HP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_HP_EXAMPLE_WRITE_SIZE);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation successful. If BGO is enabled this means the operation was started successfully. |
| FSP_ERR_IN_USE | The Flash peripheral is busy with a prior on-going transaction. |
| FSP_ERR_NOT_OPEN | The Flash API is not Open. |
| FSP_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. |
| FSP_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. |
| FSP_ERR_TIMEOUT | Timed out waiting for FCU operation to complete. |
| FSP_ERR_INVALID_SIZE | Number of bytes provided was not a multiple of the programming size or exceeded the maximum range. |
| FSP_ERR_INVALID_ADDRESS | Invalid address was input or address not on programming boundary. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit P/E mode. |

### ◆ R_FLASH_HP_Erase()

fsp_err_t R_FLASH_HP_Erase ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t const *num_blocks* )

Erases the specified Code or Data Flash blocks. Implements flash_api_t::erase by the block_erase_address.

***Note***

*Code flash may contain blocks of different sizes. When erasing code flash it is important to take this into consideration to prevent erasing a larger address space than desired.*

Example:

```
/* Erase 1 block of data flash starting at block 0. */

   err = R_FLASH_HP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful open. |
| FSP_ERR_INVALID_BLOCKS | Invalid number of blocks specified |
| FSP_ERR_INVALID_ADDRESS | Invalid address specified. If the address is in code flash then code flash programming must be enabled. |
| FSP_ERR_IN_USE | Other flash operation in progress, or API not initialized |
| FSP_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. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | The Flash API is not Open. |
| FSP_ERR_ERASE_FAILED | Status is indicating a Erase error. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit P/E mode. |

### ◆ R_FLASH_HP_BlankCheck()

fsp_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* )

Performs a blank check on the specified address area. Implements flash_api_t::blankCheck.

Example:

```
/* Check if block 0 is erased. */
   err = R_FLASH_HP_BlankCheck(&g_flash_ctrl, FLASH_DF_BLOCK_0,
FLASH_DATA_BLOCK_SIZE, &blank_check_result);
   handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Blank check operation completed with result in p_blank_check_result, or blank check started and in-progess (BGO mode). |
| FSP_ERR_INVALID_ADDRESS | Invalid data flash address was input. |
| FSP_ERR_INVALID_SIZE | 'num_bytes' was either too large or not aligned for the CF/DF boundary size. |
| FSP_ERR_IN_USE | Other flash operation in progress or API not initialized. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_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. |
| FSP_ERR_NOT_OPEN | The Flash API is not Open. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit P/E mode. |
| FSP_ERR_BLANK_CHECK_FAILED | Blank check operation failed. |

### ◆ R_FLASH_HP_Close()

fsp_err_t R_FLASH_HP_Close ( flash_ctrl_t *const *p_api_ctrl*)

Releases any resources that were allocated by the Open() or any subsequent Flash operations. Implements flash_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl or p_cfg. |

### ◆ R_FLASH_HP_StatusGet()

fsp_err_t R_FLASH_HP_StatusGet ( flash_ctrl_t *const *p_api_ctrl*, flash_status_t *const *p_status* )

Query the FLASH peripheral for its status. Implements flash_api_t::statusGet.

Example:

```
flash_status_t status;
/* Wait until the current flash operation completes. */
do
   {
      err = R_FLASH_HP_StatusGet(&g_flash_ctrl, &status);
   } while ((FSP_SUCCESS == err) && (FLASH_STATUS_BUSY == status));
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | FLASH peripheral is ready to use. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_NOT_OPEN | The Flash API is not Open. |

### ◆ R_FLASH_HP_AccessWindowSet()

fsp_err_t R_FLASH_HP_AccessWindowSet ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *start_addr*, uint32_t const *end_addr* )

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.

***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 flash_api_t::accessWindowSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Access window successfully configured. |
| FSP_ERR_INVALID_ADDRESS | Invalid settings for start_addr and/or end_addr. |
| FSP_ERR_IN_USE | FLASH peripheral is busy with a prior operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_UNSUPPORTED | Code Flash Programming is not enabled. |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit Code Flash P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of having received an illegal command. |

### ◆ R_FLASH_HP_AccessWindowClear()

fsp_err_t R_FLASH_HP_AccessWindowClear ( flash_ctrl_t *const *p_api_ctrl*)

Remove any access window that is currently configured in the Code Flash. Subsequent to this call all Code Flash is writable. Implements flash_api_t::accessWindowClear.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Access window successfully removed. |
| FSP_ERR_IN_USE | FLASH peripheral is busy with a prior operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_UNSUPPORTED | Code Flash Programming is not enabled. |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit Code Flash P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of having received an illegal command. |

### ◆ R_FLASH_HP_IdCodeSet()

fsp_err_t R_FLASH_HP_IdCodeSet ( flash_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_id_code*, flash_id_code_mode_t *mode* )

Implements flash_api_t::idCodeSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ID Code successfully configured. |
| FSP_ERR_IN_USE | FLASH peripheral is busy with a prior operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_UNSUPPORTED | Code Flash Programming is not enabled. |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit Code Flash P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of having received an illegal command. |

### ◆ R_FLASH_HP_Reset()

fsp_err_t R_FLASH_HP_Reset ( flash_ctrl_t *const *p_api_ctrl*)

Reset the FLASH peripheral. Implements flash_api_t::reset.

No attempt is made to check if the flash is busy before executing the reset since the assumption is that a reset will terminate any existing operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Flash circuit successfully reset. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of having received an illegal command. |

### ◆ R_FLASH_HP_UpdateFlashClockFreq()

fsp_err_t R_FLASH_HP_UpdateFlashClockFreq ( flash_ctrl_t *const *p_api_ctrl*)

Indicate to the already open Flash API that the FCLK has changed. Implements flash_api_t::updateFlashClockFreq.

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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Start-up area successfully toggled. |
| FSP_ERR_IN_USE | Flash is busy with an on-going operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_FCLK | FCLK is not within the acceptable range. |

### ◆ R_FLASH_HP_StartUpAreaSelect()

fsp_err_t R_FLASH_HP_StartUpAreaSelect ( flash_ctrl_t *const *p_api_ctrl*, flash_startup_area_swap_t *swap_type*, bool *is_temporary* )

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 flash_api_t::startupAreaSelect.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Start-up area successfully toggled. |
| FSP_ERR_IN_USE | FLASH peripheral is busy with a prior operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_UNSUPPORTED | Code Flash Programming is not enabled. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit Code Flash P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of having received an illegal command. |

### ◆ R_FLASH_HP_VersionGet()

fsp_err_t R_FLASH_HP_VersionGet ( fsp_version_t *const *p_version*)

This function gets FLASH HAL driver version

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation performed successfully |
| FSP_ERR_ASSERTION | Null pointer |

### ◆ R_FLASH_HP_InfoGet()

fsp_err_t R_FLASH_HP_InfoGet ( flash_ctrl_t *const *p_api_ctrl*, flash_info_t *const *p_info* )

Returns the information about the flash regions. Implements flash_api_t::infoGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful retrieved the request information. |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl or p_info. |

## 4.2.21 Low-Power Flash Driver (r_flash_lp)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_FLASH_LP_Open (flash_ctrl_t *const p_api_ctrl, flash_cfg_t const *const p_cfg) |
| fsp_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) |
| fsp_err_t | R_FLASH_LP_Erase (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t const num_blocks) |
| fsp_err_t | R_FLASH_LP_BlankCheck (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t num_bytes, flash_result_t *blank_check_result) |
| fsp_err_t | R_FLASH_LP_Close (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_LP_StatusGet (flash_ctrl_t *const p_api_ctrl, flash_status_t *const p_status) |
| fsp_err_t | R_FLASH_LP_AccessWindowSet (flash_ctrl_t *const p_api_ctrl, uint32_t const start_addr, uint32_t const end_addr) |
| fsp_err_t | R_FLASH_LP_AccessWindowClear (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_LP_IdCodeSet (flash_ctrl_t *const p_api_ctrl, uint8_t const *const p_id_code, flash_id_code_mode_t mode) |

| | |
|---|---|
| fsp_err_t | R_FLASH_LP_Reset (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_LP_StartUpAreaSelect (flash_ctrl_t *const p_api_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary) |
| fsp_err_t | R_FLASH_LP_UpdateFlashClockFreq (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_LP_VersionGet (fsp_version_t *const p_version) |
| fsp_err_t | R_FLASH_LP_InfoGet (flash_ctrl_t *const p_api_ctrl, flash_info_t *const p_info) |

### Detailed Description

Driver for the flash memory on RA low-power MCUs. This module implements the Flash Interface.

# Overview

The Flash HAL module APIs allow an application to write, erase and blank check both the data and code flash areas that reside within the MCU. The amount of flash memory available varies across MCU parts.

### Features

The Low-Power Flash HAL module has the following key features:

- Blocking and non-blocking erasing, writing and blank-checking of data flash.
- Blocking erasing, writing and blank checking of code flash.
- Callback functions for completion of non-blocking data flash operations.
- Access window (write protection) for code flash, allowing only specified areas of code flash to be erased or written.
- Boot block-swapping.
- ID code programming support.

# Configuration

### Build Time Configurations for r_flash_lp

The following build time configurations are defined in fsp_cfg/r_flash_lp_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Code Flash Programming | • Enabled<br>• Disabled | Disabled | Controls whether or not code-flash programming is enabled. Disabling |

| | | | |
|---|---|---|---|
| | | | reduces the amount of ROM and RAM used by the API. |
| Data Flash Programming | • Enabled<br>• Disabled | Enabled | Controls whether or not data-flash programming is enabled. Disabling reduces the amount of ROM used by the API. |

**Configurations for Driver > Storage > Flash Driver on r_flash_lp**

This module can be added to the Stacks tab via New Stack > Driver > Storage > Flash Driver on r_flash_lp:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_flash0 | Module name. |
| Data Flash Background Operation | • Enabled<br>• Disabled | Enabled | Enabling allows Flash API calls that reference data-flash to return immediately, with the operation continuing in the background. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be specified. Callback function called when a dataflash BGO operation completes or errors. |
| Flash Ready Interrupt Priority | MCU Specific Options | | Select the flash ready interrupt priority. |

**Clock Configuration**

Flash either uses FCLK or ICLK as the clock source depending on the MCU. When writing and erasing the clock source must be at least 4 MHz.

**Pin Configuration**

This module does not use I/O pins.

# Usage Notes

Warning

It is highly recommended that the developer reviews sections 5 and 6 of the Flash Memory section of the target MCUs Hardware User's Manual prior to using the r_flash_lp module. In particular, understanding ID Code and Access Window functionality can help avoid unrecoverable flash scenarios.

### Data Flash Background Operation (BGO) Precautions

When using the data flash BGO, the code flash, 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

Code flash cannot be accessed while writing, erasing or blank checking code flash. Code flash cannot be accessed while modifying the access window, selecting the startup area or setting the ID code. In order to support modifying code flash all supporting code must reside in RAM. This is only done when code flash programming is enabled. 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 code flash. If an interrupt occurs during the code flash operation, then code flash 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 multi-threaded environment, threads running from code flash cannot become active while a code flash operation is in progress.

### Flash Clock Source

The flash clock source is the clock used by the Flash peripheral in performing all Flash operations. As part of the flash_api_t::open function the Flash clock source is checked will return FSP_ERR_FCLK if it is invalid. Once the Flash API has been opened, if the flash clock source frequency is changed, the flash_api_t::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 (for example, flash_api_t::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 flash_api_t::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 the Flash Ready Interrupt and Flash Error Interrupts must be enabled (assigned a priority).*

### Limitations

- Write operations must be aligned on page boundaries and must be a multiple of the page boundary size.
- Erase operations will erase the entire block the provided address resides in.
- Data flash is better suited for storing data as it can be erased and written to while code is still executing from code flash. Data flash is also guaranteed for a larger number of reprogramming/erasure cycles than code flash.
- Read values of erased blocks are not guaranteed to be 0xFF. Blank check should be used to determine if memory has been erased but not yet programmed.

# Examples

### Low-Power Flash Basic Example

This is a basic example of erasing and writing to data flash and code flash.

```
#define FLASH_DF_BLOCK_0 0x40100000U /* 1 KB: 0x40100000 - 0x401003FF */
#define FLASH_CF_BLOCK_10 0x00005000  /* 2 KB: 0x00005000 - 0x000057FF */
#define FLASH_DATA_BLOCK_SIZE (1024)
#define FLASH_LP_EXAMPLE_WRITE_SIZE 32
uint8_t       g_dest[TRANSFER_LENGTH];
uint8_t       g_src[TRANSFER_LENGTH];
flash_result_t blank_check_result;
void R_FLASH_LP_basic_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the flash lp instance. */
 fsp_err_t err = R_FLASH_LP_Open(&g_flash_ctrl, &g_flash_cfg);
    handle_error(err);
 /* Erase 1 block of data flash starting at block 0. */
    err = R_FLASH_LP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
    handle_error(err);
 /* Check if block 0 is erased. */
    err = R_FLASH_LP_BlankCheck(&g_flash_ctrl, FLASH_DF_BLOCK_0,
FLASH_DATA_BLOCK_SIZE, &blank_check_result);
    handle_error(err);
 /* Verify the previously erased area is blank */
 if (FLASH_RESULT_NOT_BLANK == blank_check_result)
    {
       handle_error(FSP_ERR_BLANK_CHECK_FAILED);
    }
 /* Write 32 bytes to the first block of data flash. */
```

```
    err = R_FLASH_LP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_LP_EXAMPLE_WRITE_SIZE);
    handle_error(err);
 if (0 != memcmp(g_src, (uint8_t *) FLASH_DF_BLOCK_0, FLASH_LP_EXAMPLE_WRITE_SIZE))
    {
       handle_error(FSP_ERR_WRITE_FAILED);
    }
 /* Disable interrupts to prevent vector table access while code flash is in P/E
mode. */
    __disable_irq();
 /* Erase 1 block of code flash starting at block 10. */
    err = R_FLASH_LP_Erase(&g_flash_ctrl, FLASH_CF_BLOCK_10, 1);
    handle_error(err);
 /* Write 32 bytes to the first block of data flash. */
    err = R_FLASH_LP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_CF_BLOCK_10,
FLASH_LP_EXAMPLE_WRITE_SIZE);
    handle_error(err);
 /* Enable interrupts after code flash operations are complete. */
    __enable_irq();
 if (0 != memcmp(g_src, (uint8_t *) FLASH_CF_BLOCK_10, FLASH_LP_EXAMPLE_WRITE_SIZE))
    {
       handle_error(FSP_ERR_WRITE_FAILED);
    }
}
```

### Low-Power Flash Advanced Example

This example demonstrates using BGO to do non-blocking operations on the data flash.

```
bool interrupt_called;
flash_event_t flash_event;
static flash_cfg_t g_flash_bgo_example_cfg =
{
    .p_callback    = flash_callback,
    .p_context      = 0,
```

```
    .p_extend       = NULL,
    .data_flash_bgo = true,
    .ipl            = 5,
    .irq            = BSP_VECTOR_FLASH_LP_FRDYI_ISR,
};
void R_FLASH_LP_bgo_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the flash lp instance. */
 fsp_err_t err = R_FLASH_LP_Open(&g_flash_ctrl, &g_flash_bgo_example_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
    interrupt_called = false;
 /* Erase 1 block of data flash starting at block 0. */
    err = R_FLASH_LP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
    handle_error(err);
 while (!interrupt_called)
    {
       ;
    }
 if (FLASH_EVENT_ERASE_COMPLETE != flash_event)
    {
       handle_error(FSP_ERR_ERASE_FAILED);
    }
    interrupt_called = false;
 /* Write 32 bytes to the first block of data flash. */
    err = R_FLASH_LP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_LP_EXAMPLE_WRITE_SIZE);
    handle_error(err);
 flash_status_t status;
```

```
 /* Wait until the current flash operation completes. */
 do
    {
       err = R_FLASH_LP_StatusGet(&g_flash_ctrl, &status);
    } while ((FSP_SUCCESS == err) && (FLASH_STATUS_BUSY == status));
 /* If the interrupt wasn't called process the error. */
 if (!interrupt_called)
    {
       handle_error(FSP_ERR_WRITE_FAILED);
    }
 /* If the event wasn't a write complete process the error. */
 if (FLASH_EVENT_WRITE_COMPLETE != flash_event)
    {
       handle_error(FSP_ERR_WRITE_FAILED);
    }
 /* Verify the data was written correctly. */
 if (0 != memcmp(g_src, (uint8_t *) FLASH_DF_BLOCK_0, FLASH_LP_EXAMPLE_WRITE_SIZE))
    {
       handle_error(FSP_ERR_WRITE_FAILED);
    }
}
void flash_callback (flash_callback_args_t * p_args)
{
    interrupt_called = true;
    flash_event      = p_args->event;
}
```

## Data Structures

| | |
|---|---|
| struct | flash_lp_instance_ctrl_t |

## Data Structure Documentation

### ◆ flash_lp_instance_ctrl_t

| struct flash_lp_instance_ctrl_t |
|---|
| Flash instance control block. DO NOT INITIALIZE. Initialization occurs when R_FLASH_LP_Open() is called. |

## Function Documentation

### ◆ R_FLASH_LP_Open()

fsp_err_t R_FLASH_LP_Open ( flash_ctrl_t *const *p_api_ctrl*, flash_cfg_t const *const *p_cfg* )

Initialize the Low Power flash peripheral. Implements flash_api_t::open.

The Open function initializes the Flash.

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.

Example:

```
/* Open the flash lp instance. */

fsp_err_t err = R_FLASH_LP_Open(&g_flash_ctrl, &g_flash_cfg);
```

***Note***

*Providing a callback function in the supplied p_cfg->callback field automatically configures the Flash for Data Flash to operate in non-blocking background operation (BGO) mode.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful and timer has started. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl, p_cfg or p_callback if BGO is enabled. |
| FSP_ERR_IRQ_BSP_DISABLED | Caller is requesting BGO but the Flash interrupts are not enabled. |
| FSP_ERR_FCLK | FCLK must be a minimum of 4 MHz for Flash operations. |
| FSP_ERR_ALREADY_OPEN | Flash Open() has already been called. |
| FSP_ERR_TIMEOUT | Failed to exit P/E mode after configuring flash. |

### ◆ R_FLASH_LP_Write()

fsp_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* )

Write to the specified Code or Data Flash memory area. Implements flash_api_t::write.

Example:

```
/* Write 32 bytes to the first block of data flash. */
   err = R_FLASH_LP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_LP_EXAMPLE_WRITE_SIZE);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation successful. If BGO is enabled this means the operation was started successfully. |
| FSP_ERR_IN_USE | The Flash peripheral is busy with a prior on-going transaction. |
| FSP_ERR_NOT_OPEN | The Flash API is not Open. |
| FSP_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. |
| FSP_ERR_TIMEOUT | Timed out waiting for FCU operation to complete. |
| FSP_ERR_INVALID_SIZE | Number of bytes provided was not a multiple of the programming size or exceeded the maximum range. |
| FSP_ERR_INVALID_ADDRESS | Invalid address was input or address not on programming boundary. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |

### ◆ R_FLASH_LP_Erase()

fsp_err_t R_FLASH_LP_Erase ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t const *num_blocks* )

Erase the specified Code or Data Flash blocks. Implements flash_api_t::erase.

Example:

```
/* Erase 1 block of data flash starting at block 0. */

   err = R_FLASH_LP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful open. |
| FSP_ERR_INVALID_BLOCKS | Invalid number of blocks specified |
| FSP_ERR_INVALID_ADDRESS | Invalid address specified |
| FSP_ERR_IN_USE | Other flash operation in progress, or API not initialized |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | The Flash API is not Open. |
| FSP_ERR_TIMEOUT | Timed out waiting for FCU to be ready. |
| FSP_ERR_ERASE_FAILED | Status is indicating a Erase error. |

### ◆ R_FLASH_LP_BlankCheck()

fsp_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* )

Perform a blank check on the specified address area. Implements flash_api_t::blankCheck.

Example:

```
/* Check if block 0 is erased. */
   err = R_FLASH_LP_BlankCheck(&g_flash_ctrl, FLASH_DF_BLOCK_0,
FLASH_DATA_BLOCK_SIZE, &blank_check_result);
   handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Blankcheck operation completed with result in p_blank_check_result, or blankcheck started and in-progess (BGO mode). |
| FSP_ERR_INVALID_ADDRESS | Invalid data flash address was input |
| FSP_ERR_INVALID_SIZE | 'num_bytes' was either too large or not aligned for the CF/DF boundary size. |
| FSP_ERR_IN_USE | Flash is busy with an on-going operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_BLANK_CHECK_FAILED | An error occurred during blank checking. |

### ◆ R_FLASH_LP_Close()

fsp_err_t R_FLASH_LP_Close ( flash_ctrl_t *const *p_api_ctrl*)

Release any resources that were allocated by the Flash API. Implements flash_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl or p_cfg. |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_IN_USE | The flash is currently in P/E mode. |

### ◆ R_FLASH_LP_StatusGet()

fsp_err_t R_FLASH_LP_StatusGet ( flash_ctrl_t *const *p_api_ctrl*, flash_status_t *const *p_status* )

Query the FLASH for its status. Implements flash_api_t::statusGet.

Example:

```
flash_status_t status;

/* Wait until the current flash operation completes. */

do

   {

      err = R_FLASH_LP_StatusGet(&g_flash_ctrl, &status);

   } while ((FSP_SUCCESS == err) && (FLASH_STATUS_BUSY == status));
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Flash is ready and available to accept commands. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |

### ◆ R_FLASH_LP_AccessWindowSet()

fsp_err_t R_FLASH_LP_AccessWindowSet ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *start_addr*, uint32_t const *end_addr* )

Configure an access window for the Code Flash memory. Implements flash_api_t::accessWindowSet .

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 (inclusive) –> last block (exclusive). 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.

**Parameters**

| | | |
|---|---|---|
| | p_api_ctrl | The p api control |
| [in] | start_addr | The start address |
| [in] | end_addr | The end address |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Access window successfully configured. |
| FSP_ERR_INVALID_ADDRESS | Invalid settings for start_addr and/or end_addr. |
| FSP_ERR_IN_USE | FLASH peripheral is busy with a prior operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_UNSUPPORTED | Code Flash Programming is not enabled. |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |

### ◆ R_FLASH_LP_AccessWindowClear()

fsp_err_t R_FLASH_LP_AccessWindowClear ( flash_ctrl_t *const *p_api_ctrl*)

Remove any access window that is configured in the Code Flash. Implements flash_api_t::accessWindowClear. On successful return from this call all Code Flash is writable.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Access window successfully removed. |
| FSP_ERR_IN_USE | FLASH peripheral is busy with a prior operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_UNSUPPORTED | Code Flash Programming is not enabled. |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |

### ◆ R_FLASH_LP_IdCodeSet()

fsp_err_t R_FLASH_LP_IdCodeSet ( flash_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_id_code*, flash_id_code_mode_t *mode* )

Write the ID code provided to the id code registers. Implements flash_api_t::idCodeSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ID code successfully configured. |
| FSP_ERR_IN_USE | FLASH peripheral is busy with a prior operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_UNSUPPORTED | Code Flash Programming is not enabled. |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_TIMEOUT | Timed out waiting for completion of extra command. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |

### ◆ R_FLASH_LP_Reset()

fsp_err_t R_FLASH_LP_Reset ( flash_ctrl_t *const *p_api_ctrl*)

Reset the FLASH peripheral. Implements flash_api_t::reset.

No attempt is made to check if the flash is busy before executing the reset since the assumption is that a reset will terminate any existing operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Flash circuit successfully reset. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |

### ◆ R_FLASH_LP_StartUpAreaSelect()

fsp_err_t R_FLASH_LP_StartUpAreaSelect ( flash_ctrl_t *const *p_api_ctrl*, flash_startup_area_swap_t *swap_type*, bool *is_temporary* )

Select which block is used as the startup area block. Implements flash_api_t::startupAreaSelect.

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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Start-up area successfully toggled. |
| FSP_ERR_IN_USE | Flash is busy with an on-going operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_UNSUPPORTED | Code Flash Programming is not enabled. Cannot set FLASH_STARTUP_AREA_BTFLG when the temporary flag is false. |

### ◆ R_FLASH_LP_UpdateFlashClockFreq()

fsp_err_t R_FLASH_LP_UpdateFlashClockFreq ( flash_ctrl_t *const *p_api_ctrl*)

Indicate to the already open Flash API that the FCLK has changed. Implements flash_api_t::updateFlashClockFreq.

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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Start-up area successfully toggled. |
| FSP_ERR_IN_USE | Flash is busy with an on-going operation. |
| FSP_ERR_FCLK | Invalid flash clock source frequency. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |

### ◆ R_FLASH_LP_VersionGet()

fsp_err_t R_FLASH_LP_VersionGet ( fsp_version_t *const *p_version*)

Get Flash LP driver version.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation performed successfully |
| FSP_ERR_ASSERTION | Null Pointer |

### ◆ R_FLASH_LP_InfoGet()

fsp_err_t R_FLASH_LP_InfoGet ( flash_ctrl_t *const *p_api_ctrl*, flash_info_t *const *p_info* )

Returns the information about the flash regions. Implements flash_api_t::infoGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful retrieved the request information. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl or p_info. |
| FSP_ERR_NOT_OPEN | The flash is not open. |

## 4.2.22 Graphics LCD Controller (r_glcdc)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | R_GLCDC_Open (display_ctrl_t *const p_api_ctrl, display_cfg_t const *const p_cfg) |
| fsp_err_t | R_GLCDC_Close (display_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_GLCDC_Start (display_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_GLCDC_Stop (display_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_GLCDC_LayerChange (display_ctrl_t const *const p_api_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t layer) |
| fsp_err_t | R_GLCDC_BufferChange (display_ctrl_t const *const p_api_ctrl, uint8_t *const framebuffer, display_frame_layer_t layer) |
| fsp_err_t | R_GLCDC_ColorCorrection (display_ctrl_t const *const p_api_ctrl, display_correction_t const *const p_correction) |
| fsp_err_t | R_GLCDC_ClutUpdate (display_ctrl_t const *const p_api_ctrl, display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t layer) |
| fsp_err_t | R_GLCDC_ClutEdit (display_ctrl_t const *const p_api_ctrl, display_frame_layer_t layer, uint8_t index, uint32_t color) |
| fsp_err_t | R_GLCDC_StatusGet (display_ctrl_t const *const p_api_ctrl, display_status_t *const status) |
| fsp_err_t | R_GLCDC_VersionGet (fsp_version_t *p_version) |

### Detailed Description

Driver for the GLCDC peripheral on RA MCUs. This module implements the Display Interface.

# Overview

The GLCDC is a multi-stage graphics output peripheral designed to automatically generate timing and data signals for LCD panels. As part of its internal pipeline the two internal graphics layers can

be repositioned, alpha blended, color corrected, dithered and converted to and from a wide variety of pixel formats.

### Features

The following features are available:

| Feature | Options |
|---|---|
| Input color formats | ARGB8888, ARGB4444, ARGB1555, RGB888 (32-bit), RGB565, CLUT 8bpp, CLUT 4bpp, CLUT 1bpp |
| Output color formats | RGB888, RGB666, RGB565, Serial RGB888 (8-bit parallel) |
| Correction processes | Alpha blending, positioning, brightness and contrast, gamma correction, dithering |
| Timing signals | Dot clock, Vsync, Hsync, Vertical and horizontal data enable (DE) |
| Maximum resolution | Up to 1020 x 1008 pixels (dependent on sync signal width) |
| Maximum dot clock | 60MHz for serial RGB mode, 54MHz otherwise |
| Internal clock divisors | 1-9, 12, 16, 24, 32 |
| Interrupts | Vsync (line detect), Layer 1 underflow, Layer 2 underflow |
| Other functions | Byte-order and endianness control, line repeat function |

## Configuration

### Build Time Configurations for r_glcdc

The following build time configurations are defined in fsp_cfg/r_glcdc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected, code for parameter checking is included in the build. |
| Color Correction | • On<br>• Off | Off | If selected, code to adjust brightness, contrast and gamma settings is included in the build. When disabled all color correction |

| | | | |
|---|---|---|---|
| | | | configuration options are ignored. |

### Configurations for Driver > Graphics > Display Driver on r_glcdc

This module can be added to the Stacks tab via New Stack > Driver > Graphics > Display Driver on r_glcdc:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General > Name | Name must be a valid C symbol | g_display0 | Module name. |
| Interrupts > Callback Function | Name must be a valid C symbol | NULL | A user callback function can be defined here. |
| Interrupts > Line Detect Interrupt Priority | MCU Specific Options | | Select the line detect (Vsync) interrupt priority. |
| Interrupts > Underflow 1 Interrupt Priority | MCU Specific Options | | Select the underflow interrupt priority for layer 1. |
| Interrupts > Underflow 2 Interrupt Priority | MCU Specific Options | | Select the underflow interrupt priority for layer 2. |
| Input > Graphics Layer 1 > General > Enabled | • Yes<br>• No | Yes | Specify Used if the graphics layer 1 is used. If so a framebuffer will be automatically generated based on the specified height and horizontal stride. |
| Input > Graphics Layer 1 > General > Horizontal size | Value must be between 16 and 1016 | 480 | Specify the number of horizontal pixels. |
| Input > Graphics Layer 1 > General > Vertical size | Value must be between 16 and 1020 | 272 | Specify the number of vertical pixels. |
| Input > Graphics Layer 1 > General > Horizontal position | Must be a valid non-negative integer with a maximum configurable value of 4091 | 0 | Specify the horizontal offset in pixels of the graphics layer from the background layer. |
| Input > Graphics Layer 1 > General > Vertical position | Must be a valid non-negative integer with a maximum configurable value of 4094 | 0 | Specify the vertical offset in pixels of the graphics layer from the background layer. |
| Input > Graphics Layer | • ARGB8888 | RGB565 (16-bit) | Specify the graphics |

| | | | |
|---|---|---|---|
| 1 > General > Color format | (32-bit)<br>• RGB888 (32-bit)<br>• RGB565 (16-bit)<br>• ARGB1555 (16-bit)<br>• ARGB4444 (16-bit)<br>• CLUT8 (8-bit)<br>• CLUT4 (4-bit)<br>• CLUT1 (1-bit) | | layer Input format. If selecting CLUT formats, you must write the CLUT table data before starting output. |
| Input > Graphics Layer 1 > General > Line descending mode | • Enabled<br>• Disabled | Disabled | Select Used if the framebuffer starts from the bottom of the line. |
| Input > Graphics Layer 1 > Background Color > Alpha | Value must be between 0 and 255 | 255 | Based on the alpha value, either the graphics Layer 2 (foreground graphics layer) is blended into the graphics Layer 1 (background graphics layer) or the graphics Layer 1 is blended into the monochrome background layer. |
| Input > Graphics Layer 1 > Background Color > Red | Value must be between 0 and 255 | 255 | Red component of the background color for layer 1. |
| Input > Graphics Layer 1 > Background Color > Green | Value must be between 0 and 255 | 255 | Green component of the background color for layer 1. |
| Input > Graphics Layer 1 > Background Color > Blue | Value must be between 0 and 255 | 255 | Blue component of the background color for layer 1. |
| Input > Graphics Layer 1 > Framebuffer > Framebuffer name | This property must be a valid C symbol | fb_background | Specify the name for the framebuffer for Layer 1. |
| Input > Graphics Layer 1 > Framebuffer > Number of framebuffers | Must be a valid non-negative integer with a maximum configurable value of 65535 | 2 | Number of framebuffers allocated for Graphics Layer 1. |
| Input > Graphics Layer 1 > Framebuffer > Section for framebuffer allocation | Manual Entry | .bss | Specify the section in which to allocate the framebuffer. |
| Input > Graphics Layer 1 > Line Repeat > Enable | • On<br>• Off | Off | Select On if the display will be repeated from a smaller section of the framebuffer. |
| Input > Graphics Layer | Must be a valid non- | 0 | Specify the number of |

| | | | |
|---|---|---|---|
| 1 > Line Repeat > Repeat count | negative integer with a maximum configurable value of 65535 i.e (vertical size) x (lines repeat times) must be equal to the panel vertical size | | times the image is repeated. |
| Input > Graphics Layer 1 > Fading > Mode | • None<br>• Fade-in<br>• Fade-out | None | Select the fade method. |
| Input > Graphics Layer 1 > Fading > Speed | Value must be between 0 and 255 | 0 | Specify the number of frames for the fading transition to complete. |
| Input > Graphics Layer 2 > General > Enabled | • Yes<br>• No | No | Specify Used if the graphics layer 2 is used. If so a framebuffer will be automatically generated based on the specified height and horizontal stride. |
| Input > Graphics Layer 2 > General > Horizontal size | Value must be between 16 and 1016 | 480 | Specify the number of horizontal pixels. |
| Input > Graphics Layer 2 > General > Vertical size | Value must be between 16 and 1020 | 272 | Specify the number of vertical pixels. |
| Input > Graphics Layer 2 > General > Horizontal position | Must be a valid non-negative integer with a maximum configurable value of 4091 | 0 | Specify the horizontal offset in pixels of the graphics layer from the background layer. |
| Input > Graphics Layer 2 > General > Vertical position | Must be a valid non-negative integer with a maximum configurable value of 4094 | 0 | Specify the vertical offset in pixels of the graphics layer from the background layer. |
| Input > Graphics Layer 2 > General > Color format | • ARGB8888 (32-bit)<br>• RGB888 (32-bit)<br>• RGB565 (16-bit)<br>• ARGB1555 (16-bit)<br>• ARGB4444 (16-bit)<br>• CLUT8 (8-bit)<br>• CLUT4 (4-bit)<br>• CLUT1 (1-bit) | RGB565 (16-bit) | Specify the graphics layer Input format. If selecting CLUT formats, you must write the CLUT table data before starting output. |
| Input > Graphics Layer 2 > General > Line descending mode | • Enabled<br>• Disabled | Disabled | Select Used if the framebuffer starts from the bottom of the line. |

| | | | |
|---|---|---|---|
| Input > Graphics Layer 2 > Background Color > Alpha | Value must be between 0 and 255 | 255 | Based on the alpha value, either the graphics Layer 2 (foreground graphics layer) is blended into the graphics Layer 1 (background graphics layer) or the graphics Layer 1 is blended into the monochrome background layer. |
| Input > Graphics Layer 2 > Background Color > Red | Value must be between 0 and 255 | 255 | Red component of the background color for layer 2. |
| Input > Graphics Layer 2 > Background Color > Green | Value must be between 0 and 255 | 255 | Green component of the background color for layer 2. |
| Input > Graphics Layer 2 > Background Color > Blue | Value must be between 0 and 255 | 255 | Blue component of the background color for layer 2. |
| Input > Graphics Layer 2 > Framebuffer > Framebuffer name | This property must be a valid C symbol | fb_foreground | Specify the name for the framebuffer for Layer 2. |
| Input > Graphics Layer 2 > Framebuffer > Number of framebuffers | Must be a valid non-negative integer with a maximum configurable value of 65535 | 2 | Number of framebuffers allocated for Graphics Layer 2. |
| Input > Graphics Layer 2 > Framebuffer > Section for framebuffer allocation | Manual Entry | .bss | Specify the section in which to allocate the framebuffer. |
| Input > Graphics Layer 2 > Line Repeat > Enable | • On<br>• Off | Off | Select On if the display will be repeated from a smaller section of the framebuffer. |
| Input > Graphics Layer 2 > Line Repeat > Repeat count | Must be a valid non-negative integer with a maximum configurable value of 65535 i.e (vertical size) x (lines repeat times) must be equal to the panel vertical size | 0 | Specify the number of times the image is repeated. |
| Input > Graphics Layer 2 > Fading > Mode | • None<br>• Fade-in<br>• Fade-out | None | Select the fade method. |
| Input > Graphics Layer 2 > Fading > Speed | Value must be between 0 and 255 | 0 | Specify the number of frames for the fading transition to complete. |

| | | | |
|---|---|---|---|
| Output > Timing > Horizontal total cycles | Value must be between 24 and 1024 | 525 | 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 |
| Output > Timing > Horizontal active video cycles | Value must be between 16 and 1016 | 480 | Specify the number of active video cycles in a horizontal line (including front and back porch). Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. |
| Output > Timing > Horizontal back porch cycles | Value must be between 6 and 1006 | 40 | 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. |
| Output > Timing > Horizontal sync signal cycles | Value must be between 0 and 1023 | 1 | 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. |
| Output > Timing > Horizontal sync signal polarity | • Low active<br>• High active | Low active | Select the polarity of Hsync signal to match your system. |
| Output > Timing > Vertical total lines | Value must be between 20 and 1024 | 316 | Specify number of total lines in a frame (including front and back porch). |
| Output > Timing > Vertical active video lines | Value must be between 16 and 1020 | 272 | Specify the number of active video lines in a frame. |
| Output > Timing > Vertical back porch lines | Value must be between 3 and 1007 | 8 | 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. |
| Output > Timing > Vertical sync signal lines | Value must be between 0 and 1023 | 1 | Specify the Vsync signal assertion lines in a frame. |
| Output > Timing > Vertical sync signal polarity | • Low active<br>• High active | Low active | Select the polarity of Vsync signal to match to your system. |
| Output > Timing > Data Enable Signal Polarity | • Low active<br>• High active | High active | Select the polarity of Data Enable signal to match to your system. |
| Output > Timing > Sync edge | • Rising edge<br>• Falling edge | Rising edge | Select the polarity of Sync signals to match to your system. |
| Output > Format > Color format | • 24bits RGB888<br>• 18bits RGB666<br>• 16bits RGB565<br>• 8bits serial | 16bits RGB565 | Specify the graphics layer output format to match to your LCD panel. |
| Output > Format > Color order | • RGB<br>• BGR | RGB | Select data order for output signal to LCD panel. |
| Output > Format > Endian | • Little endian<br>• Big endian | Little endian | Select data endianness for output signal to LCD panel. |
| Output > Background > Alpha | Value must be between 0 and 255 | 255 | Alpha component of the background color. |
| Output > Background > Red | Value must be between 0 and 255 | 0 | Red component of the background color. |
| Output > Background > Green | Value must be between 0 and 255 | 0 | Green component of the background color. |
| Output > Background > Blue | Value must be between 0 and 255 | 0 | Blue component of the background color. |
| CLUT > Enabled | • Yes<br>• No | No | Specify Used if selecting CLUT formats for a graphics layer input format. If used, a buffer (CLUT_buffer) will be automatically generated based on the selected pixel width. |
| CLUT > Size | Must be a valid non-negative integer with a maximum configurable value of 256 | 256 | Specify the number of entries for the CLUT source data buffer. Each entry consumes 4 bytes (1 word). |

| | | | |
|---|---|---|---|
| TCON > Hsync pin select | • Not used<br>• LCD_TCON0<br>• LCD_TCON1<br>• LCD_TCON2<br>• LCD_TCON3 | LCD_TCON0 | Select the TCON pin used for the Hsync signal to match to your system. |
| TCON > Vsync pin select | • Not used<br>• LCD_TCON0<br>• LCD_TCON1<br>• LCD_TCON2<br>• LCD_TCON3 | LCD_TCON1 | Select TCON pin used for Vsync signal to match to your system. |
| TCON > Data enable (DE) pin select | • Not used<br>• LCD_TCON0<br>• LCD_TCON1<br>• LCD_TCON2<br>• LCD_TCON3 | LCD_TCON2 | Select TCON pin used for DataEnable signal to match to your system. |
| TCON > Panel clock source | • Internal clock (GLCDCLK)<br>• External clock (LCD_EXTCLK) | Internal clock (GLCDCLK) | Choose between an internal GLCDCLK generated from PCLKA or an external clock provided to the LCD_EXTCLK pin. |
| TCON > Panel clock division ratio | Refer to the RA Configuration tool for available options. | 1/24 | Select the clock source divider value. |
| Color Correction > Brightness > Enabled | • Yes<br>• No | No | Enable brightness color correction. |
| Color Correction > Brightness > Red channel | Value must be between 0 and 1023 | 512 | Red component of the brightness calibration. This value is divided by 512 to determine gain. |
| Color Correction > Brightness > Green channel | Value must be between 0 and 1023 | 512 | Green component of the brightness calibration. This value is divided by 512 to determine gain. |
| Color Correction > Brightness > Blue channel | Value must be between 0 and 1023 | 512 | Blue component of the brightness calibration. This value is divided by 512 to determine gain. |
| Color Correction > Contrast > Enabled | • Yes<br>• No | No | Enable contrast color correction. |
| Color Correction > Contrast > Red channel gain | Value must be between 0 and 255 | 128 | Red component of the contrast calibration. This value is divided by 128 to determine gain. |
| Color Correction > Contrast > Green channel gain | Value must be between 0 and 255 | 128 | Green component of the contrast calibration. This value |

| | | | |
|---|---|---|---|
| | | | is divided by 128 to determine gain. |
| Color Correction > Contrast > Blue channel gain | Value must be between 0 and 255 | 128 | Blue component of the contrast calibration. This value is divided by 128 to determine gain. |
| Color Correction > Gamma > Red | • On<br>• Off | Off | Enable gamma color correction for the red channel. |
| Color Correction > Gamma > Green | • On<br>• Off | Off | Enable gamma color correction for the green channel. |
| Color Correction > Gamma > Blue | • On<br>• Off | Off | Enable gamma color correction for the blue channel. |
| Color Correction > Process order | • Brightness/contrast first<br>• Gamma first | Brightness/contrast first | Select the color correction processing order. |
| Dithering > Enabled | • Yes<br>• No | No | Enable dithering to reduce the effect of color banding. |
| Dithering > Mode | • Truncate<br>• Round off<br>• 2x2 Pattern | Truncate | Select the dithering mode. |
| Dithering > Pattern A | • Pattern 00<br>• Pattern 01<br>• Pattern 10<br>• Pattern 11 | Pattern 11 | Select the dithering pattern. |
| Dithering > Pattern B | • Pattern 00<br>• Pattern 01<br>• Pattern 10<br>• Pattern 11 | Pattern 11 | Select the dithering pattern. |
| Dithering > Pattern C | • Pattern 00<br>• Pattern 01<br>• Pattern 10<br>• Pattern 11 | Pattern 11 | Select the dithering pattern. |
| Dithering > Pattern D | • Pattern 00<br>• Pattern 01<br>• Pattern 10<br>• Pattern 11 | Pattern 11 | Select the dithering pattern. |

### Clock Configuration

The peripheral clock for this module is PCLKA.

The dot clock is typically generated from the PLL with a maximum output frequency of 54 MHz in most pixel formats (60 MHz for serial RGB). Optionally, a clock signal can be provided to the LCD_EXTCLK pin for finer framerate control (60 MHz maximum input). With either clock source

dividers of 1-9, 12, 16, 24 and 32 may be used. Clocks must be initialized and settled prior to starting this module.

### Pin Configuration

This module controls a variety of pins necessary for LCD data and timing signal output:

| Pin Name | Function | Notes |
|---|---|---|
| LCD_EXTCLK | External clock signal input | The maximum input clock frequency is 60MHz. |
| LCD_CLK | Dot clock output | The maximum output frequency is 54MHz (60MHz in serial RGB mode). |
| LCD_DATAn | Pixel data output lines | Pin assignment and color order is based on the output block configuration. See the RA6M3 User's Manual (R01UH0886EJ0100) section 58.1.4 "Output Control for Data Format" for details. |
| LCD_TCONn | Panel timing signal output | These pins can be configured to output vertical and horizontal synchronization and data valid signals. |

*Note*

*There are two banks of pins listed for the GLCDC in the RA6M3 User's Manual (_A and _B). In most cases the _B bank will be used as _A conflicts with SDRAM pins. In either case, it is generally recommended to only use pins from only one bank at a time as this allows for superior signal routing both inside and outside the package. If _A and _B pins must be mixed be sure to note the timing precision penalty detailed in Table 60.33 in in the RA6M3 User's Manual.*

## Usage Notes

### Overview

The GLCDC peripheral is a combination of several sub-peripherals that form a pixel data processing pipeline. Each block passes pixel data to the next but otherwise they are disconnected from one another - in other words, changing timing block parameters does not affect the output generation block configuration and vice versa.

### Initial Configuration

During R_GLCDC_Open all configured parameters are set in the GLCDC peripheral fully preparing it for operation. Once opened, calling R_GLCDC_Start is typically all that is needed for basic operation. Background generation, timing and output parameters are not configurable at runtime, though layer control and color correction options can be altered.

### Framebuffer Allocation

The framebuffer should be allocated in the highest-speed region available (excluding SRAMHS)

without displacing the stack, heap and other program-critical structures. While the RA6M3 does contain a relatively large 640K of on-chip SRAM, for many screen sizes and color depths SDRAM will be required. Regardless of the placement two rules must be followed to ensure correct operation of the GLCDC:

- The framebuffer must be aligned on a 64-byte boundary
- The horizontal stride of the buffer must be a multiple of 64 bytes

*Note*

*Framebuffers allocated through the RA Configuraton tool automatically follow the alignment and size requirements.*

If your framebuffer will be placed into internal SRAM please note the following best practices:

- The framebuffer should ideally not be placed in the SRAMHS block of SRAM as there is no speed advantage for doing so. In particular, it is important to ensure the framebuffer does not push the stack or any heaps outside of SRAMHS to preserve CPU performance.
- It is recommended to not cross the boundary between SRAM0 and SRAM1 with a single framebuffer for performance reasons.
- If double-buffering is desired (and possible within SRAM), place one framebuffer in SRAM0 and the other in SRAM1.

If you are using SRAM for the framebuffer, to ensure correct placement you will need to edit the linker script to add new sections. Below is an example of the required edits in the GCC and IAR formats:

### GCC Linker

```
/*
  Linker File for RA6M3 MCU
*/
/* Linker script to configure memory regions. */
MEMORY
{
  FLASH (rx)         : ORIGIN = 0x00000000, LENGTH = 0x0200000 /*   2M */
  RAM (rwx)          : ORIGIN = 0x1FFE0000, LENGTH = 0x00A0000 /* 640K */
  FB0 (rwx)          : ORIGIN = 0x20000000, LENGTH = 0x0080000 /* 512K */ // Section
for framebuffer 0 (or only framebuffer)
  FB1 (rwx)          : ORIGIN = 0x20040000, LENGTH = 0x0040000 /* 256K */ // Section
for framebuffer 1
  DATA_FLASH (rx)    : ORIGIN = 0x40100000, LENGTH = 0x0010000 /* 64K */
  QSPI_FLASH (rx)    : ORIGIN = 0x60000000, LENGTH = 0x4000000 /* 64M */
  SDRAM (rwx)       : ORIGIN = 0x90000000, LENGTH = 0x2000000 /* 32M */
  ID_CODE (rx)       : ORIGIN = 0x0100A150, LENGTH = 0x10    /* 16 bytes */
}
```

```
// ...
    .noinit (NOLOAD):
    {
       . = ALIGN(4);
       __noinit_start = .;
       KEEP(*(.noinit*))
       __noinit_end = .;
    } > RAM
 /* Place framebuffer sections first, then the rest of RAM */
    .fb0 :
    {
       . = ALIGN(64);
       __fb0_start = .;
       *(.fb0*);
       __fb0_end = .;
    } > FB0
    .fb1 :
    {
       . = ALIGN(64);
       __fb1_start = .;
       *(.fb1*);
       __fb1_end = .;
    } > FB1
    .bss :
    {
       . = ALIGN(4);
       __bss_start__ = .;
       *(.bss*)
       *(COMMON)
       . = ALIGN(4);
       __bss_end__ = .;
    } > RAM
// ...
```

### IAR Linker

*Note*

*The IAR linker does not place items correctly when sections overlap. As a result, it is advised to place your framebuffer(s) as high as possible in the SRAM region in the linker script to maximize the RAM available for everything else. The below is a general case that should be used unedited only if RAM usage (excluding framebuffers) is less than 128K.*

```
/* ... */
/*-Memory Regions-*/
define symbol region_VECT_start    = 0x00000000;
define symbol region_VECT_end      = 0x000003FF;
define symbol region_ROMREG_start = 0x00000400;
define symbol region_ROMREG_end    = 0x000004FF;
define symbol region_FLASH_start   = 0x00000500;
define symbol region_FLASH_end    = 0x001FFFFF;
define symbol region_RAM_start    = 0x1FFE0000;
define symbol region_RAM_end       = 0x1FFFFFFF; /* RAM limited to SRAMHS */
define symbol region_FB0_start    = 0x20000000;
define symbol region_FB0_end       = 0x2003FFFF; /* SRAM0 dedicated to framebuffer 0
*/
define symbol region_FB1_start    = 0x20040000;
define symbol region_FB1_end       = 0x2007FFFF; /* SRAM1 dedicated to framebuffer 1
*/
define symbol region_DF_start      = 0x40100000;
define symbol region_DF_end       = 0x4010FFFF;
define symbol region_SDRAM_start   = 0x90000000;
define symbol region_SDRAM_end    = 0x91FFFFFF;
define symbol region_QSPI_start    = 0x60000000;
define symbol region_QSPI_end      = 0x63FFFFFF;
/* ... */
define memory mem with size    = 4G;
define region VECT_region       = mem:[from region_VECT_start    to region_VECT_end];
define region ROMREG_region    = mem:[from region_ROMREG_start to region_ROMREG_end];
define region FLASH_region      = mem:[from region_FLASH_start   to
region_FLASH_end];
define region RAM_region       = mem:[from region_RAM_start    to region_RAM_end];
```

```
define region FB0_region       = mem:[from region_FB0_start    to region_FB0_end]; /*
Define framebuffer 0 region */
define region FB1_region       = mem:[from region_FB1_start    to region_FB1_end]; /*
Define framebuffer 1 region */
define region DF_region         = mem:[from region_DF_start      to region_DF_end];
define region SDRAM_region      = mem:[from region_SDRAM_start   to
region_SDRAM_end];
define region QSPI_region       = mem:[from region_QSPI_start    to region_QSPI_end];
/* ... */
define block START_OF_RAM with fixed order { rw section .fsp_dtc_vector_table,
                                             block RAM_CODE };
place at start of RAM_region { block START_OF_RAM };
/* Place framebuffer sections first, then the rest of RAM */
place in FB0_region { rw section .fb0 };
place in FB1_region { rw section .fb1 };
place in RAM_region     { rw,
                         rw section .noinit,
                         rw section .bss,
                         rw section .data,
                         rw section HEAP,
                         rw section .stack };
```

### Graphics Layers and Timing Parameters

The GLCDC synthesizes graphics data through two configurable graphics layers onto a background layer. The background is used as a solid-color canvas upon which to composite data from the graphics layers. The two graphics layers are blended on top of each other (Layer 2 above Layer 1) and overlaid on the background layer based on their individual configuration. The placement of the layers (as well as LCD timing parameters) are detailed in Figure 1. The colors of the dimensions indicate which element of the display_cfg_t struct is being referenced - for example, the width of the background layer would be [display_cfg].output.htiming.display_cyc as shown in the figure below.

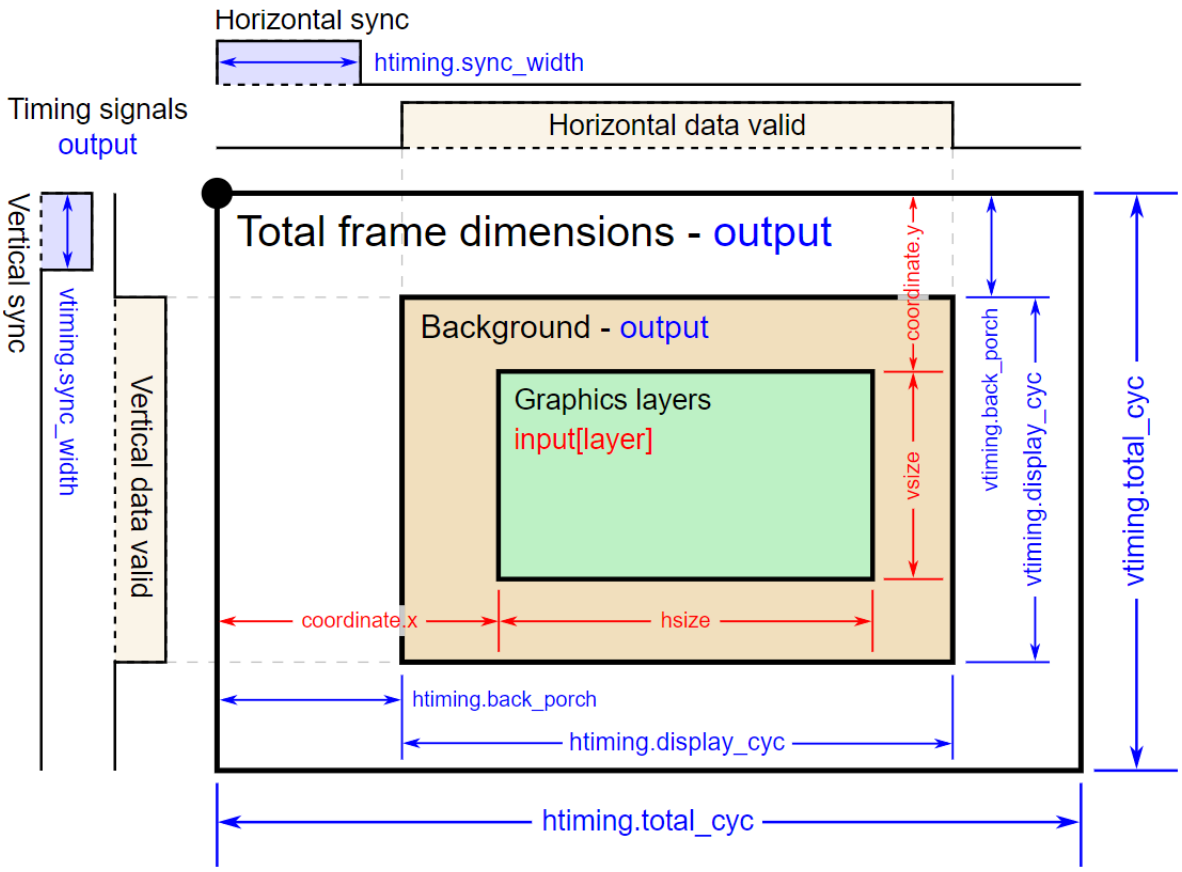



Figure 120: GLCDC layers and timing

*Note*

*The data enable signal (if configured) is the logical AND of the horizontal and vertical data valid signals. In the GLCDC layers and timing figure, only one graphics layer is shown for simplicity. Additionally, in most applications the graphics layer(s) will be the same dimensions as the background layer.*

### Runtime Configuration Options

*Note*

*All runtime configurations detailed below are also automatically configured during R_GLCDC_Open based on the options selected in the RA Configuration editor.*

### Blend processing

Control of layer positioning, alpha blending and fading is possible at runtime via R_GLCDC_LayerChange. This function takes a display_runtime_cfg_t parameter which contains the same input and layer elements as the display_cfg_t control block. Refer to the documentation for display_runtime_cfg_t as well as the Examples below to see what options are configurable.

### Brightness and contrast

Brightness and contrast correction can be controlled through R_GLCDC_ColorCorrection. The display_correction_t parameter is used to control enabling, disabling and gain values for both corrections as shown below:

```
display_correction_t correction;
/* Brightness values are 0-1023 with +512 offset being neutral */
   correction.brightness.r = 512;
   correction.brightness.g = 512;
   correction.brightness.b = 512;
/* Contrast values are 0-255 representing gain of 0-2 (128 is gain of 1) */
   correction.contrast.r = 128;
   correction.contrast.g = 128;
   correction.contrast.b = 128;
/* Brightness and contrast correction can be enabled or disabled independent of one
another */
   correction.brightness.enable = true;
   correction.contrast.enable   = true;
/* Enable correction */
R_GLCDC_ColorCorrection(&g_disp_ctrl, &correction);
```

### Color Look-Up Table (CLUT) Modes

The GLCDC supports 1-, 4- and 8-bit color look-up table (CLUT) formats for input pixel data. By using these modes the framebuffer size in memory can be reduced significantly, allowing even high-resolution displays to be buffered in on-chip SRAM. To enable CLUT modes for a layer the color format must be set to a CLUT mode (either at startup or through R_GLCDC_LayerChange) in addition to filling the CLUT as appropriate via R_GLCDC_ClutUpdate as shown below:

```
/* Basic 4-bit (16-color) CLUT definition */
   uint32_t clut_4[16] =
   {
      0xFF000000,                  // Black
      0xFFFFFFFF,                  // White
      0xFF0000FF,                  // Blue
      0xFF0080FF,                  // Turquoise
      0xFF00FFFF,                  // Cyan
      0xFF00FF80,                  // Mint Green
      0xFF00FF00,                  // Green
      0xFF80FF00,                  // Lime Green
      0xFFFFFF00,                  // Yellow
      0xFFFF8000,                  // Orange
```

```
      0xFFFF0000,                    // Red
      0xFFFF0080,                    // Pink
      0xFFFF00FF,                    // Magenta
      0xFF8000FF,                    // Purple
      0xFF808080,                    // Gray
      0x00000000                      // Transparent
   };
/* Define the CLUT configuration */
display_clut_cfg_t clut_cfg =
   {
      .start = 0,
      .size   = 16,
      .p_base = clut_4
   };
/* Update the CLUT in the GLCDC */
R_GLCDC_ClutUpdate(&g_disp_ctrl, &clut_cfg, DISPLAY_FRAME_LAYER_1);
```

*Note*

*If individual elements of the CLUT must be changed or if elements must be changed one at a time (for instance, when using emWin) it is recommended to use R_GLCDC_ClutEdit to avoid repeated memcpy operations.*

### Other Configuration Options

### Gamma correction

Gamma correction is performed based on a gain curved defined in the RA Configuration editor. Each point on the curve is defined by a threshold and a gain value - each gain value represents a multiplier from 0x-2x (set as 0-2047) that sets the Y-value of the slope of the gain curve, while each threshold interval sets the X-value respectively. For a more detailed explanation refer to the RA6M3 User's Manual (R01UH0886EJ0100) Figure 58.12 "Calculation of gamma correction value" and the related description above it.

When setting threshold values three rules must be followed:

- Each threshold value must be greater than the previous value
- Threshold values must be greater than zero and less than 1024
- Threshold values can equal the previous value only if they are 1023 (maximum)

*Note*

*Gamma correction can only be applied via R_GLCDC_Open.*

### Dithering

Dithering is a method of pixel blending that allows for smoother transitions between colors when

using a limited palette. A full description of dithering is outside the scope of this document. For more information on the pattern settings and how to configure them refer to the RA6M3 User's Manual (R01UH0886EJ0100) Figure 58.13 "Configuration of dither correction block" and Figure 58.14 "Addition value selection method for 2x2 pattern dither".

### Bus Utilization

*Note*

***The data provided in this section consists of estimates only. Experimentation is necessary to obtain real-world performance data on any platform.***

While the GLCDC is very flexible in size and color depth of displays there are considerations to be made in the tradeoff between color depth, framerate and bus utilization. Below is a table showing estimates of the load at various resolutions, framerates and color depths based on a PLL frequency of 120MHz (default) and an effective SDRAM throughput of 60 MB/sec. Bus utilization percentages are provided for the following use cases:

- Static image display (**GLCDC only**): One read
- Redrawing one framebuffer every display frame (**minimal redraw**): One write, one read
- Blitting one buffer to another then redrawing the entire buffer every display frame (**worst case**): Two writes, three reads

| Name | Width | Height | Input color depth (bits) | Framerate (FPS) | Buffer size (bytes) | SRAM use | SRAM bus (GLCDC only) | SDRAM bus (GLCDC only) | SRAM bus (minimal redraw) | SDRAM bus (minimal redraw) | SRAM bus (worst case) | SDRAM bus (worst case) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HQVGA | 240 | 160 | 8 | 60 | 38400 | 6% | 1% | 4% | 2% | 8% | 5% | 19% |
| HQVGA | 240 | 160 | 16 | 60 | 76800 | 12% | 2% | 8% | 4% | 15% | 10% | 38% |
| QVGA | 320 | 240 | 16 | 60 | 153600 | 23% | 4% | 15% | 8% | 31% | 19% | 77% |
| WQVGA | 400 | 240 | 8 | 60 | 96000 | 15% | 2% | 10% | 5% | 19% | 12% | 48% |
| WQVGA | 400 | 240 | 16 | 60 | 192000 | 29% | 5% | 19% | 10% | 38% | 24% | 96% |
| HVGA | 480 | 320 | 16 | 60 | 307200 | 47% | 8% | 31% | 15% | 61% | 38% | 154% |
| VGA | 640 | 480 | 16 | 30 | 614400 | — | — | 31% | — | 61% | — | 154% |
| WVGA | 800 | 480 | 8 | 60 | 384000 | 59% | 10% | 38% | 19% | 77% | 48% | 192% |
| WVGA | 800 | 480 | 16 | 30 | 768000 | — | — | 38% | — | 77% | — | 192% |
| WVGA | 800 | 480 | 32 | 15 | 1536000 | — | — | 38% | — | 77% | — | 192% |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FWVGA | 960 | 480 | 8 | 30 | 460800 | 70% | 6% | 23% | 12% | 46% | 29% | 115% |
| FWVGA | 960 | 480 | 16 | 30 | 921600 | — | — | 46% | — | 92% | — | 230% |
| qHD | 960 | 540 | 8 | 30 | 518400 | 79% | 6% | 26% | 13% | 52% | 32% | 130% |

*Note*

*Bus utilization values over 100% indicate that the bandwidth for that bus is exceeded in that scenario and GLCDC underflow and/or dropped frames may result depending on the bus priority setting.* ***It is recommended to avoid these scenarios if at all possible by reducing the buffer drawing rate, number of draw/copy operations or the input color depth.*** *Relaxing vertical timing (increasing total line count) or increasing the clock divider are the easiest ways to increase the time per frame.*

### Limitations

Developers should be aware of the following limitations when using the GLCDC API:

- Due to a limitation of the GLCDC hardware, if the horizontal back porch is less than the number of pixels in a graphics burst read (64 bytes) for a layer and the layer is positioned at a negative X-value then the layer X-position will be locked to the nearest 64-byte boundary, rounded toward zero.
- The GLCDC peripheral offers a chroma-key function that can be used to perform a green-screen-like color replacement. This functionality is not exposed through the GLCDC API. See the descriptions for GRn.AB7 through .AB9 in the RA6M3 User's Manual for further details.
- Use of R_GLCDC_ClutUpdate and R_GLCDC_ClutEdit may not be mixed on the same frame.

## Examples

### Basic Example

This is a basic example showing the minimum code required to initialize and start the GLCDC module. If the entire display can be drawn within the vertical blanking period no further code may be necessary.

```
void glcdc_init (void)
{
 fsp_err_t err;
 // Open the GLCDC driver
    err = R_GLCDC_Open(&g_disp_ctrl, &g_disp_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 // Start display output
    err = R_GLCDC_Start(&g_disp_ctrl);
    handle_error(err);
}
```

### Layer Transitions

This example demonstrates how to set up and execute both a sliding and fading layer transition. This is most useful in static image transition scenarios as switching between two actively-drawing graphics layers may require up to four framebuffers to eliminate tearing.

```
volatile uint32_t g_vsync_count = 0;
/* Callback function for GLCDC interrupts */
static void glcdc_callback (display_callback_args_t * p_args)
{
 if (p_args->event == DISPLAY_EVENT_LINE_DETECTION)
    {
       g_vsync_count++;
    }
}
/* Simple wait that returns 1 if no Vsync happened within the timeout period */
uint8_t vsync_wait (void)
{
    uint32_t timeout_timer = GLCDC_VSYNC_TIMEOUT;
    g_vsync_count = 0;
 while (!g_vsync_count && --timeout_timer)
    {
 /* Spin here until DISPLAY_EVENT_LINE_DETECTION callback or timeout */
    }
 return timeout_timer ? 0 : 1;
}
/* Initiate a fade on Layer 2
 *
 * Parameters:
 * direction True for fade in, false for fade out
 * speed number of frames over which to fade
 */
void glcdc_layer_transition_fade (display_runtime_cfg_t * disp_rt_cfg, bool
direction, uint16_t speed)
{
```

```
 fsp_err_t err;
 if (direction)
    {
 /* Set the runtime struct to the desired buffer */
       disp_rt_cfg->input.p_base       = (uint32_t *) g_framebuffer_1;
       disp_rt_cfg->layer.fade_control = DISPLAY_FADE_CONTROL_FADEIN;
    }
 else
    {
       disp_rt_cfg->layer.fade_control = DISPLAY_FADE_CONTROL_FADEOUT;
    }
 /* Ensure speed is at least 1 frame */
 if (!speed)
    {
       speed = 1;
    }
 /* Set the fade speed to the desired change in alpha per frame */
    disp_rt_cfg->layer.fade_speed = UINT8_MAX / speed;
 /* Initiate the fade (will start on the next Vsync) */
    err = R_GLCDC_LayerChange(&g_disp_ctrl, disp_rt_cfg, DISPLAY_FRAME_LAYER_2);
    handle_error(err);
}
/* Slide Layer 1 out to the left while sliding Layer 2 in from the right */
void glcdc_layer_transition_sliding (display_runtime_cfg_t * disp_rt_cfg_in,
display_runtime_cfg_t * disp_rt_cfg_out)
{
 fsp_err_t err;
 /* Set the config for the incoming layer to be just out of bounds on the right side
*/
    disp_rt_cfg_in->input.p_base       = (uint32_t *) g_framebuffer_1;
    disp_rt_cfg_in->layer.coordinate.x = DISPLAY_WIDTH;
 /* Move layer 1 out and layer 2 in at a fixed rate of 4 pixels per frame */
 for (int32_t x = disp_rt_cfg_in->layer.coordinate.x; x >= 0; x -= 4)
    {
```

```
 /* Wait for a Vsync before starting */
       vsync_wait();
 /* Set the X-coordinate of both layers then update them */
       disp_rt_cfg_out->layer.coordinate.x = (int16_t) (x - DISPLAY_WIDTH);
       disp_rt_cfg_in->layer.coordinate.x = (int16_t) x;
       err = R_GLCDC_LayerChange(&g_disp_ctrl, disp_rt_cfg_out, DISPLAY_FRAME_LAYER_1
);
      handle_error(err);
       err = R_GLCDC_LayerChange(&g_disp_ctrl, disp_rt_cfg_in, DISPLAY_FRAME_LAYER_2
);
      handle_error(err);
    }
}
```

### Double-Buffering

Using a double-buffer allows one to be output to the LCD while the other is being drawn to memory, eliminating tearing and in some cases reducing bus load. The following is a basic example showing integration of the line detect (Vsync) interrupt to set the timing for buffer swapping and drawing.

```
/* User-defined function to draw the current display to a framebuffer */
void display_draw (uint8_t * framebuffer)
{
 FSP_PARAMETER_NOT_USED(framebuffer);
 /* Draw buffer here */
}
/* This function is an example of a basic double-buffered display thread */
void display_thread (void)
{
    uint8_t * p_framebuffer = NULL;
 fsp_err_t err;
 /* Initialize and start the R_GLCDC module */
    glcdc_init();
 while (1)
    {
 /* Swap the active framebuffer */
```

```
        p_framebuffer = (p_framebuffer == g_framebuffer_0) ? g_framebuffer_1 :
g_framebuffer_0;
 /* Draw the new framebuffer now */
        display_draw(p_framebuffer);
 /* Now that the framebuffer is ready, update the GLCDC buffer pointer on the next
Vsync */
        err = R_GLCDC_BufferChange(&g_disp_ctrl, p_framebuffer, DISPLAY_FRAME_LAYER_1
);
      handle_error(err);
 /* Wait for a Vsync event */
        vsync_wait();
    }
}
```

## Data Structures

| | |
|---|---|
| struct | glcdc_instance_ctrl_t |
| struct | glcdc_extended_cfg_t |

## Enumerations

| | |
|---|---|
| enum | glcdc_clk_src_t |
| enum | glcdc_panel_clk_div_t |
| enum | glcdc_tcon_pin_t |
| enum | glcdc_bus_arbitration_t |
| enum | glcdc_correction_proc_order_t |
| enum | glcdc_tcon_signal_select_t |
| enum | glcdc_clut_plane_t |
| enum | glcdc_dithering_mode_t |
| enum | glcdc_dithering_pattern_t |
| enum | glcdc_input_interface_format_t |
| enum | glcdc_output_interface_format_t |

| | |
|---|---|
| enum | glcdc_dithering_output_format_t |

## Data Structure Documentation

### ◆ glcdc_instance_ctrl_t

| struct glcdc_instance_ctrl_t |
|---|
| Display control block. DO NOT INITIALIZE. |

### ◆ glcdc_extended_cfg_t

| struct glcdc_extended_cfg_t | | |
|---|---|---|
| GLCDC hardware specific configuration | | |
| Data Fields | | |
| glcdc_tcon_pin_t | tcon_hsync | GLCDC TCON output pin select. |
| glcdc_tcon_pin_t | tcon_vsync | GLCDC TCON output pin select. |
| glcdc_tcon_pin_t | tcon_de | GLCDC TCON output pin select. |
| glcdc_correction_proc_order_t | correction_proc_order | Correction control route select. |
| glcdc_clk_src_t | clksrc | Clock Source selection. |
| glcdc_panel_clk_div_t | clock_div_ratio | Clock divide ratio for dot clock. |
| glcdc_dithering_mode_t | dithering_mode | Dithering mode. |
| glcdc_dithering_pattern_t | dithering_pattern_A | Dithering pattern A. |
| glcdc_dithering_pattern_t | dithering_pattern_B | Dithering pattern B. |
| glcdc_dithering_pattern_t | dithering_pattern_C | Dithering pattern C. |
| glcdc_dithering_pattern_t | dithering_pattern_D | Dithering pattern D. |

## Enumeration Type Documentation

### ◆ glcdc_clk_src_t

| enum glcdc_clk_src_t | |
|---|---|
| Clock source select | |
| Enumerator | |
| GLCDC_CLK_SRC_INTERNAL | Internal. |
| GLCDC_CLK_SRC_EXTERNAL | External. |

### ◆ glcdc_panel_clk_div_t

| enum glcdc_panel_clk_div_t | |
|---|---|
| Clock frequency division ratio | |
| Enumerator | |
| GLCDC_PANEL_CLK_DIVISOR_1 | Division Ratio 1/1. |
| GLCDC_PANEL_CLK_DIVISOR_2 | Division Ratio 1/2. |
| GLCDC_PANEL_CLK_DIVISOR_3 | Division Ratio 1/3. |
| GLCDC_PANEL_CLK_DIVISOR_4 | Division Ratio 1/4. |
| GLCDC_PANEL_CLK_DIVISOR_5 | Division Ratio 1/5. |
| GLCDC_PANEL_CLK_DIVISOR_6 | Division Ratio 1/6. |
| GLCDC_PANEL_CLK_DIVISOR_7 | Division Ratio 1/7. |
| GLCDC_PANEL_CLK_DIVISOR_8 | Division Ratio 1/8. |
| GLCDC_PANEL_CLK_DIVISOR_9 | Division Ratio 1/9. |
| GLCDC_PANEL_CLK_DIVISOR_12 | Division Ratio 1/12. |
| GLCDC_PANEL_CLK_DIVISOR_16 | Division Ratio 1/16. |
| GLCDC_PANEL_CLK_DIVISOR_24 | Division Ratio 1/24. |
| GLCDC_PANEL_CLK_DIVISOR_32 | Division Ratio 1/32. |

### ◆ glcdc_tcon_pin_t

| enum glcdc_tcon_pin_t | |
|---|---|
| LCD TCON output pin select | |
| Enumerator | |
| GLCDC_TCON_PIN_NONE | No output. |
| GLCDC_TCON_PIN_0 | LCD_TCON0. |
| GLCDC_TCON_PIN_1 | LCD_TCON1. |
| GLCDC_TCON_PIN_2 | LCD_TCON2. |
| GLCDC_TCON_PIN_3 | LCD_TCON3. |

### ◆ glcdc_bus_arbitration_t

| enum glcdc_bus_arbitration_t | |
|---|---|
| Bus Arbitration setting | |
| Enumerator | |
| GLCDC_BUS_ARBITRATION_ROUNDROBIN | Round robin. |
| GLCDC_BUS_ARBITRATION_FIX_PRIORITY | Fixed. |

### ◆ glcdc_correction_proc_order_t

| enum glcdc_correction_proc_order_t | |
|---|---|
| Correction circuit sequence control | |
| Enumerator | |
| GLCDC_CORRECTION_PROC_ORDER_BRIGHTNESS_CONTRAST2GAMMA | Brightness -> contrast -> gamma correction. |
| GLCDC_CORRECTION_PROC_ORDER_GAMMA2BRIGHTNESS_CONTRAST | Gamma correction -> brightness -> contrast. |

### ◆ glcdc_tcon_signal_select_t

| enum glcdc_tcon_signal_select_t | |
|---|---|
| Timing signals for driving the LCD panel | |
| Enumerator | |
| GLCDC_TCON_SIGNAL_SELECT_STVA_VS | STVA/VS. |
| GLCDC_TCON_SIGNAL_SELECT_STVB_VE | STVB/VE. |
| GLCDC_TCON_SIGNAL_SELECT_STHA_HS | STH/SP/HS. |
| GLCDC_TCON_SIGNAL_SELECT_STHB_HE | STB/LP/HE. |
| GLCDC_TCON_SIGNAL_SELECT_DE | DE. |

### ◆ glcdc_clut_plane_t

| enum glcdc_clut_plane_t | |
|---|---|
| Clock phase adjustment for serial RGB output | |
| Enumerator | |
| GLCDC_CLUT_PLANE_0 | GLCDC CLUT plane 0. |
| GLCDC_CLUT_PLANE_1 | GLCDC CLUT plane 1. |

### ◆ glcdc_dithering_mode_t

| enum glcdc_dithering_mode_t | |
|---|---|
| Dithering mode | |
| Enumerator | |
| GLCDC_DITHERING_MODE_TRUNCATE | No dithering (truncate) |
| GLCDC_DITHERING_MODE_ROUND_OFF | Dithering with round off. |
| GLCDC_DITHERING_MODE_2X2PATTERN | Dithering with 2x2 pattern. |

### ◆ glcdc_dithering_pattern_t

| enum glcdc_dithering_pattern_t | |
|---|---|
| Dithering mode | |
| Enumerator | |
| GLCDC_DITHERING_PATTERN_00 | 2x2 pattern '00' |
| GLCDC_DITHERING_PATTERN_01 | 2x2 pattern '01' |
| GLCDC_DITHERING_PATTERN_10 | 2x2 pattern '10' |
| GLCDC_DITHERING_PATTERN_11 | 2x2 pattern '11' |

### ◆ glcdc_input_interface_format_t

| enum glcdc_input_interface_format_t | |
|---|---|
| Output interface format | |
| Enumerator | |
| GLCDC_INPUT_INTERFACE_FORMAT_RGB565 | Input interface format RGB565. |
| GLCDC_INPUT_INTERFACE_FORMAT_RGB888 | Input interface format RGB888. |
| GLCDC_INPUT_INTERFACE_FORMAT_ARGB1555 | Input interface format ARGB1555. |
| GLCDC_INPUT_INTERFACE_FORMAT_ARGB4444 | Input interface format ARGB4444. |
| GLCDC_INPUT_INTERFACE_FORMAT_ARGB8888 | Input interface format ARGB8888. |
| GLCDC_INPUT_INTERFACE_FORMAT_CLUT8 | Input interface format CLUT8. |
| GLCDC_INPUT_INTERFACE_FORMAT_CLUT4 | Input interface format CLUT4. |
| GLCDC_INPUT_INTERFACE_FORMAT_CLUT1 | Input interface format CLUT1. |

### ◆ glcdc_output_interface_format_t

| enum glcdc_output_interface_format_t | |
|---|---|
| Output interface format | |
| Enumerator | |
| GLCDC_OUTPUT_INTERFACE_FORMAT_RGB888 | Output interface format RGB888. |
| GLCDC_OUTPUT_INTERFACE_FORMAT_RGB666 | Output interface format RGB666. |
| GLCDC_OUTPUT_INTERFACE_FORMAT_RGB565 | Output interface format RGB565. |
| GLCDC_OUTPUT_INTERFACE_FORMAT_SERIAL_RGB | Output interface format Serial RGB. |

### ◆ glcdc_dithering_output_format_t

| enum glcdc_dithering_output_format_t | |
|---|---|
| Dithering output format | |
| Enumerator | |
| GLCDC_DITHERING_OUTPUT_FORMAT_RGB888 | Dithering output format RGB888. |
| GLCDC_DITHERING_OUTPUT_FORMAT_RGB666 | Dithering output format RGB666. |
| GLCDC_DITHERING_OUTPUT_FORMAT_RGB565 | Dithering output format RGB565. |

## Function Documentation

### ◆ R_GLCDC_Open()

fsp_err_t R_GLCDC_Open ( display_ctrl_t *const *p_api_ctrl*, display_cfg_t const *const *p_cfg* )

Open GLCDC module. Implements display_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was opened successfully. |
| FSP_ERR_ALREADY_OPEN | Device was already open. |
| FSP_ERR_ASSERTION | Pointer to the control block or the configuration structure is NULL. |
| FSP_ERR_CLOCK_GENERATION | Dot clock cannot be generated from clock source. |
| FSP_ERR_INVALID_TIMING_SETTING | Invalid panel timing parameter. |
| FSP_ERR_INVALID_LAYER_SETTING | Invalid layer setting found. |
| FSP_ERR_INVALID_ALIGNMENT | Input buffer alignment invalid. |
| FSP_ERR_INVALID_GAMMA_SETTING | Invalid gamma correction setting found |
| FSP_ERR_INVALID_BRIGHTNESS_SETTING | Invalid brightness 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.*

### ◆ R_GLCDC_Close()

fsp_err_t R_GLCDC_Close ( display_ctrl_t *const *p_api_ctrl*)

Close GLCDC module. Implements display_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was closed successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL. |
| FSP_ERR_NOT_OPEN | The function call is performed when the driver state is not equal to DISPLAY_STATE_CLOSED. |
| FSP_ERR_INVALID_UPDATE_TIMING | A function call is performed when the GLCDC 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.*

### ◆ R_GLCDC_Start()

fsp_err_t R_GLCDC_Start ( display_ctrl_t *const *p_api_ctrl*)

Start GLCDC module. Implements display_api_t::start.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was started successfully. |
| FSP_ERR_NOT_OPEN | GLCDC module has not been opened. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL. |

***Note***

*This API can be called when the driver is not in DISPLAY_STATE_OPENED status.*

### ◆ R_GLCDC_Stop()

fsp_err_t R_GLCDC_Stop ( display_ctrl_t *const *p_api_ctrl*)

Stop GLCDC module. Implements display_api_t::stop.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was stopped successfully |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL |
| FSP_ERR_INVALID_MODE | Function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING. |
| FSP_ERR_INVALID_UPDATE_TIMING | The function call is performed while the GLCDC 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.*

### ◆ R_GLCDC_LayerChange()

fsp_err_t R_GLCDC_LayerChange ( display_ctrl_t const *const *p_api_ctrl*, display_runtime_cfg_t const *const *p_cfg*, display_frame_layer_t *layer* )

Change layer parameters of GLCDC module at runtime. Implements display_api_t::layerChange.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Changed layer parameters of GLCDC module successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block or the configuration structure is NULL. |
| FSP_ERR_INVALID_MODE | A function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING. |
| FSP_ERR_INVALID_UPDATE_TIMING | A function call is performed while the GLCDC 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.*

### ◆ R_GLCDC_BufferChange()

fsp_err_t R_GLCDC_BufferChange ( display_ctrl_t const *const *p_api_ctrl*, uint8_t *const *framebuffer*, display_frame_layer_t *layer* )

Change the framebuffer pointer for a layer. Implements display_api_t::bufferChange.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Changed layer parameters of GLCDC module successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL. |
| FSP_ERR_INVALID_MODE | A function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING. |
| FSP_ERR_INVALID_ALIGNMENT | The framebuffer pointer is not 64-byte aligned. |
| FSP_ERR_INVALID_UPDATE_TIMING | A function call is performed while the GLCDC is updating register values internally. |

***Note***

*This API can be called when the driver is in DISPLAY_STATE_OPENED state or higher. 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.*

### ◆ R_GLCDC_ColorCorrection()

fsp_err_t R_GLCDC_ColorCorrection ( display_ctrl_t const *const *p_api_ctrl*, display_correction_t const *const *p_correction* )

Perform color correction through the GLCDC module. Implements display_api_t::correction.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Color correction by GLCDC module was performed successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block or the display correction structure is NULL. |
| FSP_ERR_INVALID_MODE | Function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING. |
| FSP_ERR_INVALID_UPDATE_TIMING | A function call is performed while the GLCDC is updating registers internally. |
| FSP_ERR_INVALID_BRIGHTNESS_SETTING | Invalid brightness correction setting found |

***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.*

### ◆ R_GLCDC_ClutUpdate()

fsp_err_t R_GLCDC_ClutUpdate ( display_ctrl_t const *const *p_api_ctrl*, display_clut_cfg_t const *const *p_clut_cfg*, display_frame_layer_t *layer* )

Write an entire color look-up table (CLUT) in the GLCDC module. Implements display_api_t::clut.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CLUT written successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block or CLUT source data is NULL. |
| FSP_ERR_INVALID_UPDATE_TIMING | R_GLCDC_ClutEdit was already used to edit the specified CLUT this frame. |
| FSP_ERR_INVALID_CLUT_ACCESS | Illegal CLUT entry or size is specified. |

***Note***

*This API can be called any time. The written data will be used after the next vertical sync event.*

### ◆ R_GLCDC_ClutEdit()

fsp_err_t R_GLCDC_ClutEdit ( display_ctrl_t const *const *p_api_ctrl*, display_frame_layer_t *layer*, uint8_t *index*, uint32_t *color* )

Update an element of a color look-up table (CLUT) in the GLCDC module. Implements display_api_t::clutEdit.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CLUT element updated successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL. |

***Note***

*This API can be called any time. The written data will be used after the next vertical sync event.*

### ◆ R_GLCDC_StatusGet()

fsp_err_t R_GLCDC_StatusGet ( display_ctrl_t const *const *p_api_ctrl*, display_status_t *const *p_status* )

Get status of GLCDC module. Implements display_api_t::statusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got status successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block or the status structure is NULL. |

***Note***

*The GLCDC 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_PENDING, instead of DISPLAY_FADE_STATUS_NOT_UNDERWAY.*

### ◆ R_GLCDC_VersionGet()

fsp_err_t R_GLCDC_VersionGet ( fsp_version_t * *p_version*)

Get version of R_GLCDC module. Implements display_api_t::versionGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got version information successfully. |

***Note***

*This function is re-entrant.*

## 4.2.23 General PWM Timer (r_gpt)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | R_GPT_Open (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg) |
| fsp_err_t | R_GPT_Stop (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_Start (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_Reset (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_Enable (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_Disable (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_PeriodSet (timer_ctrl_t *const p_ctrl, uint32_t const period_counts) |
| fsp_err_t | R_GPT_DutyCycleSet (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin) |
| fsp_err_t | R_GPT_InfoGet (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info) |
| fsp_err_t | R_GPT_StatusGet (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status) |
| fsp_err_t | R_GPT_CounterSet (timer_ctrl_t *const p_ctrl, uint32_t counter) |
| fsp_err_t | R_GPT_OutputEnable (timer_ctrl_t *const p_ctrl, gpt_io_pin_t pin) |
| fsp_err_t | R_GPT_OutputDisable (timer_ctrl_t *const p_ctrl, gpt_io_pin_t pin) |
| fsp_err_t | R_GPT_AdcTriggerSet (timer_ctrl_t *const p_ctrl, gpt_adc_compare_match_t which_compare_match, uint32_t compare_match_value) |
| fsp_err_t | R_GPT_Close (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Driver for the GPT32 and GPT16 peripherals on RA MCUs. This module implements the Timer

Interface.

# Overview

The GPT module can be used to count events, measure external input signals, generate a periodic interrupt, or output a periodic or PWM signal to a GTIOC pin.

This module supports the GPT peripherals GPT32EH, GPT32E, GPT32, and GPT16. GPT16 is a 16-bit timer. The other peripherals (GPT32EH, GPT32E, and GPT32) are 32-bit timers. The 32-bit timers are all treated the same in this module from the API perspective.

### Features

The GPT module has the following features:

- Supports periodic mode, one-shot mode, and PWM mode.
- Supports count source of PCLK, GTETRG pins, GTIOC pins, or ELC events.
- Supports debounce filter on GTIOC pins.
- Signal can be output to a pin.
- Configurable period (counts per timer cycle).
- Configurable duty cycle in PWM mode.
- Supports runtime reconfiguration of period.
- Supports runtime reconfiguration of duty cycle in PWM mode.
- APIs are provided to start, stop, and reset the counter.
- APIs are provided to get the current period, source clock frequency, and count direction.
- APIs are provided to get the current timer status and counter value.
- Supports start, stop, clear, count up, count down, and capture by external sources from GTETRG pins, GTIOC pins, or ELC events.
- Supports symmetric and asymmetric PWM waveform generation.
- Supports automatic addition of dead time.
- Supports generating ELC events to start an ADC scan at a compare match value (see Event Link Controller (r_elc)) and updating the compare match value.
- Supports linking with a POEG channel to automatically disable GPT output when an error condition is detected.
- Supports setting the counter value while the timer is stopped.
- Supports enabling and disabling output pins.
- Supports skipping up to seven overflow/underflow (crest/trough) interrupts at a time

### Selecting a Timer

RA MCUs have two timer peripherals: the General PWM Timer (GPT) and the Asynchronous General Purpose Timer (AGT). When selecting between them, consider these factors:

| | GPT | AGT |
|---|---|---|
| Low Power Modes | The GPT can operate in sleep mode. | The AGT can operate in all low power modes. |
| Available Channels | The number of GPT channels is device specific. All currently supported MCUs have at least 7 GPT channels. | All MCUs have 2 AGT channels. |
| Timer Resolution | All MCUs have at least one 32-bit GPT timer. | The AGT timers are 16-bit timers. |

| | | |
|---|---|---|
| Clock Source | The GPT runs off PCLKD with a configurable divider up to 1024. It can also be configured to count ELC events or external pulses. | The AGT runs off PCLKB, LOCO, or subclock. |

# Configuration

### Build Time Configurations for r_gpt

The following build time configurations are defined in fsp_cfg/r_gpt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Pin Output Support | • Disabled<br>• Enabled<br>• Enabled with Extra Features | Disabled | If selected code for outputting a waveform to a pin is included in the build. |
| Write Protect Enable | • Enabled<br>• Disabled | Disabled | If selected write protection is applied to all GPT channels. |

### Configurations for Driver > Timers > Timer Driver on r_gpt

This module can be added to the Stacks tab via New Stack > Driver > Timers > Timer Driver on r_gpt:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General > Name | Name must be a valid C symbol | g_timer0 | Module name. |
| General > Channel | Channel number must exist on this MCU | 0 | Specify the hardware channel. |
| General > Mode | • Periodic<br>• One-Shot<br>• PWM<br>• Triangle-Wave Symmetric PWM<br>• Triangle-Wave Asymmetric PWM | Periodic | Mode selection. Periodic: Generates periodic interrupts or square waves. One-shot: Generate a single interrupt or a pulse wave. Note: One-shot mode is implemented in software. ISRs must be enabled for one-shot even if callback is |

| | | | |
|---|---|---|---|
| | | | unused.<br>PWM: Generates basic PWM waveforms.<br>Triangle-Wave Symmetric PWM: Generates symmetric PWM waveforms with duty cycle determined by compare match set during a crest interrupt and updated at the next trough.<br>Triangle-Wave Asymmetric PWM: Generates asymmetric PWM waveforms with duty cycle determined by compare match set during a crest/trough interrupt and updated at the next trough/crest. |
| General > Period | Value must be a non-negative integer less than or equal to 0x40000000000 | 0x100000000 | Specify the timer period in units selected below. Setting the period to 0x100000000 raw counts results in the maximum period. Set the period to 0x100000000 raw counts for a free running timer or an input capture configuration. The period can be set up to 0x40000000000, which will use a divider of 1024 with the maximum period. |
| General > Period Unit | • Raw Counts<br>• Nanoseconds<br>• Microseconds<br>• Milliseconds<br>• Seconds<br>• Hertz<br>• Kilohertz | Raw Counts | Unit of the period specified above |
| Output > Duty Cycle Percent (only applicable in PWM mode) | Value must be between 0 and 100 | 50 | Specify the timer duty cycle percent. Only used in PWM mode. |
| Output > Duty Cycle Range (only applicable | • Shortest: 2 PCLK, Longest: | Shortest: 2 PCLK, Longest: (Period - 1) | Select the duty cycle range. Due to hardware |

| | | | |
|---|---|---|---|
| in PWM mode) | (Period - 1) PCLK<br>• Shortest: 1 PCLK, Longest: (Period - 2) PCLK | PCLK | limitations, one PCLK cycle is added before the output pin toggles after the duty cycle is reached. This extra clock cycle is added to the ON time (if Shortest: 2 PCLK is selected) or the OFF time (if Shortest: 1 PCLK is selected) based on this configuration. |
| Output > GTIOCA Output Enabled | • True<br>• False | False | Enable the output of GTIOCA on a pin. |
| Output > GTIOCA Stop Level | • Pin Level Low<br>• Pin Level High | Pin Level Low | Select the behavior of the output pin when the timer is stopped. |
| Output > GTIOCB Output Enabled | • True<br>• False | False | Enable the output of GTIOCB on a pin. |
| Output > GTIOCB Stop Level | • Pin Level Low<br>• Pin Level High | Pin Level Low | Select the behavior of the output pin when the timer is stopped. |
| Input > Count Up Source | MCU Specific Options | | Select external source that will increment the counter. If any count up source is selected, the timer will count the external sources only. It will not count PCLKD cycles. |
| Input > Count Down Source | MCU Specific Options | | Select external source that will decrement the counter. If any count down source is selected, the timer will count the external sources only. It will not count PCLKD cycles. |
| Input > Start Source | MCU Specific Options | | Select external source that will start the timer.<br><br>For pulse width measurement, set the Start Source and the Clear Source to the trigger edge (the edge to start the measurement), and set the Stop Source and Capture Source (either A or B) to the opposite |

| | | | |
|---|---|---|---|
| | | | edge (the edge to stop the measurement).<br><br>For pulse period measurement, set the Start Source, the Clear Source, and the Capture Source (either A or B) to the trigger edge (the edge to start the measurement). |
| Input > Stop Source | MCU Specific Options | | Select external source that will stop the timer. |
| Input > Clear Source | MCU Specific Options | | Select external source that will clear the timer. |
| Input > Capture A Source | MCU Specific Options | | Select external source that will trigger a capture A event. |
| Input > Capture B Source | MCU Specific Options | | Select external source that will trigger a capture B event. |
| Input > Noise Filter A Sampling Clock Select | • No Filter<br>• Filter PCLKD / 1<br>• Filter PCLKD / 4<br>• Filter PCLKD / 16<br>• Filter PCLKD / 64 | No Filter | Select the input filter for GTIOCA. |
| Input > Noise Filter B Sampling Clock Select | • No Filter<br>• Filter PCLKD / 1<br>• Filter PCLKD / 4<br>• Filter PCLKD / 16<br>• Filter PCLKD / 64 | No Filter | Select the input filter for GTIOCB. |
| Interrupts > Callback | Name must be a valid C symbol | NULL | A user callback function can be specified here. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses |
| Interrupts > Overflow/Crest Interrupt Priority | MCU Specific Options | | Select the overflow interrupt priority. This is the crest interrupt for triangle-wave PWM. |

| | | | |
|---|---|---|---|
| Interrupts > Capture A Interrupt Priority | MCU Specific Options | | Select the interrupt priority for capture A. |
| Interrupts > Capture B Interrupt Priority | MCU Specific Options | | Select the interrupt priority for capture B. |
| Interrupts > Trough Interrupt Priority | MCU Specific Options | | Select the interrupt priority for the trough interrupt (triangle-wave PWM only). |
| Extra Features > Output Disable > POEG Link | • POEG Channel 0<br>• POEG Channel 1<br>• POEG Channel 2<br>• POEG Channel 3 | POEG Channel 0 | Select which POEG to link this GPT channel to. |
| Extra Features > Output Disable > Output Disable POEG Trigger | • Dead Time Error<br>• GTIOCA and GTIOCB High Level<br>• GTIOCA and GTIOCB Low Level | | Select which errors send an output disable trigger to POEG. Dead time error is only available on GPT32E and GPT32EH variants. |
| Extra Features > Output Disable > GTIOCA Disable Setting | • Disable Prohibited<br>• Set Hi Z<br>• Level Low<br>• Level High | Disable Prohibited | Select the disable setting for GTIOCA. |
| Extra Features > Output Disable > GTIOCB Disable Setting | • Disable Prohibited<br>• Set Hi Z<br>• Level Low<br>• Level High | Disable Prohibited | Select the disable setting for GTIOCB. |
| Extra Features > ADC Trigger > Start Event Trigger (GPTE/GPTEH only) | • Trigger Event A/D Converter Start Request A During Up Counting<br>• Trigger Event A/D Converter Start Request A During Down Counting<br>• Trigger Event A/D Converter Start Request B During Up Counting<br>• Trigger Event A/D Converter | | Select which A/D converter start request interrupts to generate and at which point in the cycle to generate them. This value only applies to the GPT32E and GPT32EH variants. |

| | | | |
|---|---|---|---|
| | Start Request B During Down Counting | | |
| Extra Features > Dead Time > Dead Time Count Up (Raw Counts) | Must be a valid non-negative integer with a maximum configurable value of 65535. | 0 | Select the dead time to apply during up counting. This value also applies during down counting for the GPT32 and GPT16 variants. |
| Extra Features > Dead Time > Dead Time Count Down (Raw Counts) (GPTE/GPTEH only) | Must be a valid non-negative integer with a maximum configurable value of 65535. | 0 | Select the dead time to apply during down counting. This value only applies to the GPT32E and GPT32EH variants. |
| Extra Features > ADC Trigger (GPTE/GPTEH only) > ADC A Compare Match (Raw Counts) | Must be a valid non-negative integer with a maximum configurable value of 65535. | 0 | Select the compare match value that generates a GPTn AD TRIG A event. This value only applies to the GPT32E and GPT32EH variants. |
| Extra Features > ADC Trigger (GPTE/GPTEH only) > ADC B Compare Match (Raw Counts) | Must be a valid non-negative integer with a maximum configurable value of 65535. | 0 | Select the compare match value that generates a GPTn AD TRIG B event. This value only applies to the GPT32E and GPT32EH variants. |
| Extra Features > Interrupt Skipping (GPTE/GPTEH only) > Interrupt to Count | • None<br>• Overflow and Underflow (sawtooth)<br>• Crest (triangle)<br>• Trough (triangle) | None | Select the count source for interrupt skipping. The interrupt skip counter increments after each source event. All crest/overflow and trough/underflow interrupts are skipped when the interrupt skip counter is non-zero. This value only applies to the GPT32E and GPT32EH variants. |
| Extra Features > Interrupt Skipping (GPTE/GPTEH only) > Interrupt Skip Count | • 0<br>• 1<br>• 2<br>• 3<br>• 4<br>• 5<br>• 6 | 0 | Select the number of interrupts to skip. This value only applies to the GPT32E and GPT32EH variants. |

| | | | |
|---|---|---|---|
| | • 7 | | |
| Extra Features > Interrupt Skipping (GPTE/GPTEH only) > Skip ADC Events | • None<br>• ADC A Compare Match<br>• ADC B Compare Match<br>• ADC A and B Compare Match | module.driver.timer.interrupt_skip.adc.none | Select ADC events to suppress when the interrupt skip count is not zero. This value only applies to the GPT32E and GPT32EH variants. |
| Extra Features > Extra Features | • Enabled<br>• Disabled | Disabled | Select whether to enable extra features on this channel. |

### Clock Configuration

The GPT clock is based on the PCLKD frequency. You can set the PCLKD frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

### Pin Configuration

This module can use GTETRGA, GTETRGB, GTETRGC, GTETRGD, GTIOCA and GTIOCB pins as count sources.

This module can use GTIOCA and GTIOCB pins as output pins for periodic or PWM signals.

This module can use GTIOCA and GTIOCB as input pins to measure input signals.

## Usage Notes

### Maximum Period for GPT32

The RA Configuration editor will automatically calculate the period count value and source clock divider based on the selected period time, units and clock speed.

When the selected period unit is "Raw counts", the maximum period setting is 0x40000000000 on a 32-bit timer or 0x0x4000000 on a 16-bit timer. This will configure the timer with the maximum period and a count clock divisor of 128.

*Note*

> *When manually changing the timer period counts the maximum value for a 32-bit GPT is 0x100000000. This number overflows the 32-bit value for timer_cfg_t::period_counts. To configure the timer for the maximum period, set timer_cfg_t::period_counts to 0.*

### Updating Period and Duty Cycle

The period and duty cycle are updated after the next counter overflow after calling R_GPT_PeriodSet() or R_GPT_DutyCycleSet(). To force them to update before the next counter overflow, call R_GPT_Reset() while the counter is running.

### One-Shot Mode

The GPT timer does not support one-shot mode natively. One-shot mode is achieved by stopping the timer in the interrupt service routine before the callback is called. If the interrupt is not serviced before the timer period expires again, the timer generates more than one event. The callback is only

called once in this case, but multiple events may be generated if the timer is linked to the Data Transfer Controller (r_dtc).

### One-Shot Mode Output

The output waveform in one-shot mode is one PCLKD cycle less than the configured period. The configured period must be at least 2 counts to generate an output pulse.

Examples of one-shot signals that can be generated by this module are shown below:

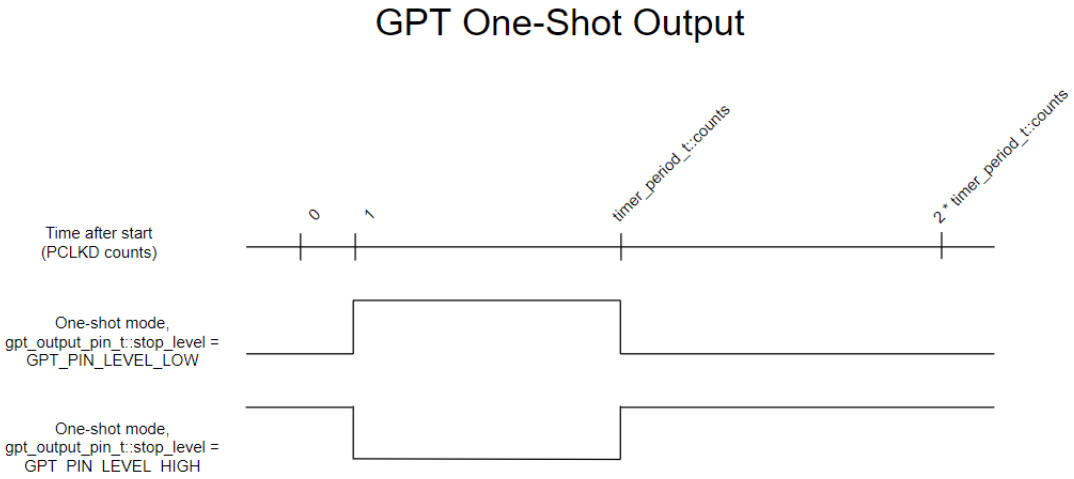



Figure 121: GPT One-Shot Output

### Periodic Output

The GTIOC pin toggles twice each time the timer expires in periodic mode. This is achieved by defining a PWM wave at a 50 percent duty cycle so that the period of the resulting square wave (from rising edge to rising edge) matches the period of the GPT timer. Since the periodic output is actually a PWM output, the time at the stop level is one cycle shorter than the time opposite the stop level for odd period values.

Examples of periodic signals that can be generated by this module are shown below:

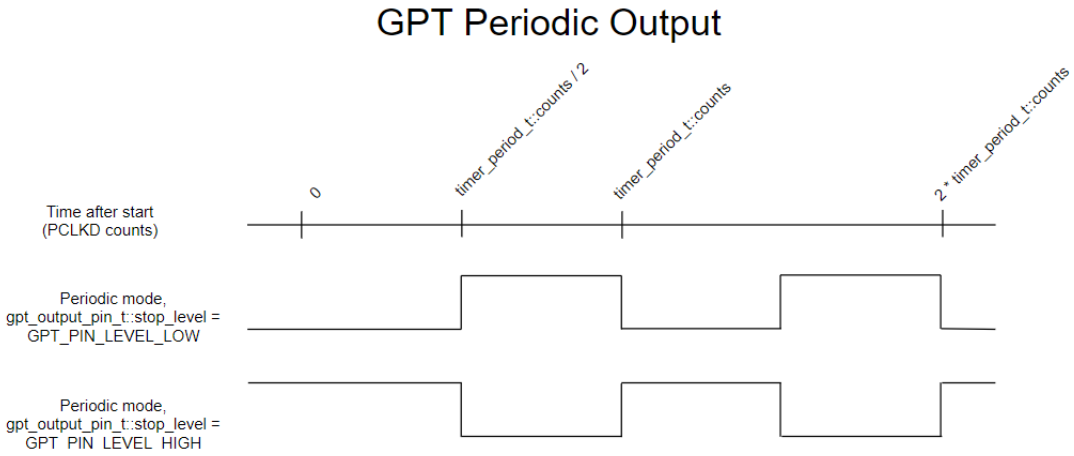



Figure 122: GPT Periodic Output

### PWM Output

The PWM output signal is high at the beginning of the cycle and low at the end of the cycle. If gpt_extended_cfg_t::shortest_pwm_signal is set to GPT_SHORTEST_LEVEL_ON, the PWM output signal is low at the beginning of the cycle and high at the end of the cycle.

Examples of PWM signals that can be generated by this module are shown below:

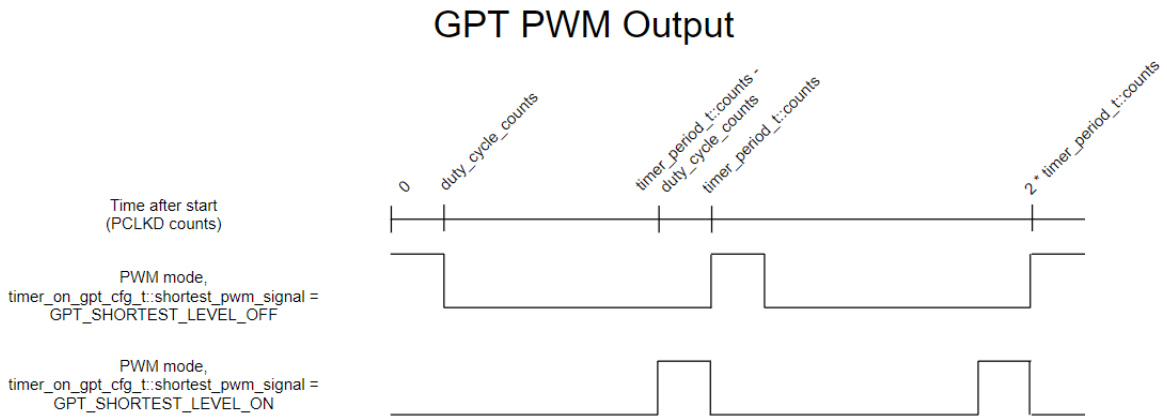



Figure 123: GPT PWM Output

### Triangle-Wave PWM Output

Examples of PWM signals that can be generated by this module are shown below. The duty_cycle_counts can be modified using R_GPT_DutyCycleSet() in the crest interrupt and updated at the following trough for symmetric PWM or modified in both the crest/trough interrupts and updated at the following trough/crest for asymmetric PWM.

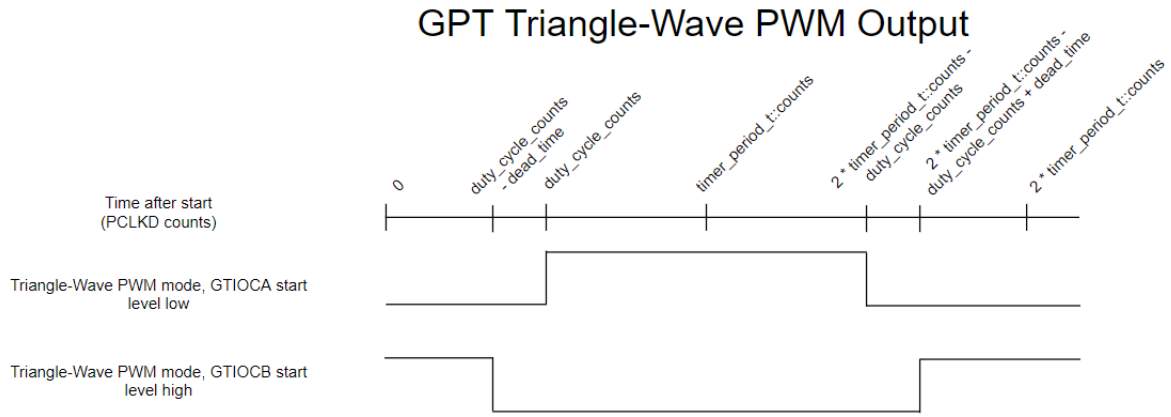



Figure 124: GPT Triangle-Wave PWM Output

### Event Counting

Event counting can be done by selecting up or down counting sources from GTETRG pins, ELC events, or GTIOC pins. In event counting mode, the GPT counter is not affected by PCLKD.

*Note*

*In event counting mode, the application must call R_GPT_Start() to enable event counting. The counter will not change after calling R_GPT_Start() until an event occurs.*

### Pulse Measurement

If the capture edge occurs before the start edge in pulse measurement, the first capture is invalid (0).

### Controlling GPT with GTETRG Edges

The GPT timer can be configured to stop, start, clear, count up, or count down when a GTETRG rising or falling edge occurs.

*Note*

*The GTETRG pins are shared by all GPT channels.*
*GTETRG pins require POEG to be on (example code for this is provided in GPT Free Running Counter Example).*
*If input filtering is required on the GTETRG pins, that must also be handled outside this module.*

### Controlling GPT with ELC Events

The GPT timer can be configured to stop, start, clear, count up, or count down when an ELC event occurs.

*Note*

*The configurable ELC GPT sources are shared by all GPT channels.*
*The event links for the ELC must be configured outside this module.*

### Triggering ELC Events with GPT

The GPT timer can trigger the start of other peripherals. The Event Link Controller (r_elc) guide provides a list of all available peripherals.

### Enabling External Sources for Start, Stop, Clear, or Capture

R_GPT_Enable() must be called when external sources are used for start, stop, clear, or capture.

### Interrupt Skipping

When an interrupt skipping source is selected a hardware counter will increment each time the selected event occurs. Each interrupt past the first (up to the specified skip count) will be suppressed. If ADC events are selected for skipping they will also be suppressed except during the timer period leading to the selected interrupt skipping event (see below diagram).

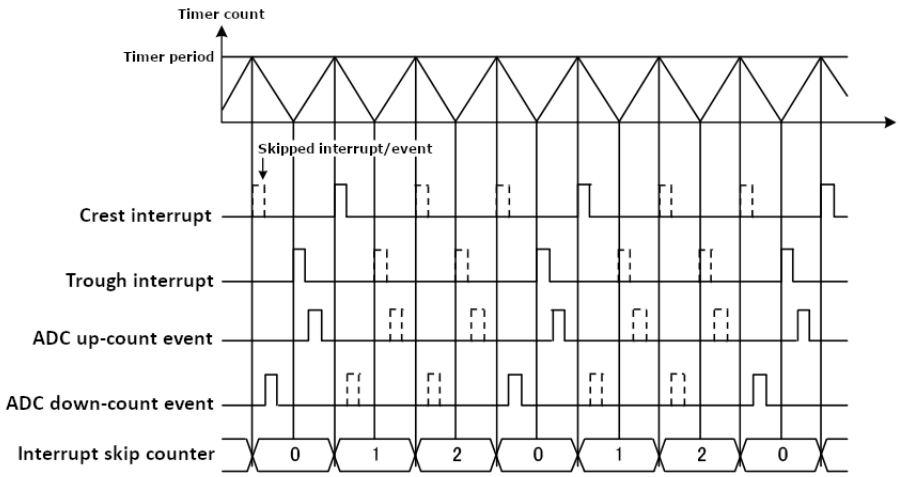



Figure 125: Crest interrupt skipping in triangle-wave PWM modes (skip count 2)

# Examples

### GPT Basic Example

This is a basic example of minimal use of the GPT in an application.

```
void gpt_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
}
```

### GPT Callback Example

This is an example of a timer callback.

```
/* Example callback called when timer expires. */
void timer_callback (timer_callback_args_t * p_args)
```

```
{
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* Add application code to be called periodically here. */
    }
}
```

### GPT Free Running Counter Example

To use the GPT as a free running counter, select periodic mode and set the the Period to 0xFFFFFFFF for a 32-bit timer or 0xFFFF for a 16-bit timer.

```
void gpt_counter_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) If event count mode is used to count edges on a GTETRG pin, POEG must
be started to use GTETRG.
  * Reference Note 1 of Table 23.2 "GPT functions" in the RA6M3 manual
R01UH0886EJ0100. */
 R_BSP_MODULE_START(FSP_IP_POEG, 0U);
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
 /* (Optional) Stop the timer. */
    (void) R_GPT_Stop(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_GPT_StatusGet(&g_timer0_ctrl, &status);
}
```

### GPT Input Capture Example

This is an example of using the GPT to capture pulse width or pulse period measurements.

```
/* Example callback called when a capture occurs. */
uint64_t g_captured_time    = 0U;
uint32_t g_capture_overflows = 0U;
void timer_capture_callback (timer_callback_args_t * p_args)
{
 if ((TIMER_EVENT_CAPTURE_A == p_args->event) || (TIMER_EVENT_CAPTURE_B ==
p_args->event))
    {
 /* (Optional) Get the current period if not known. */
 timer_info_t info;
       (void) R_GPT_InfoGet(&g_timer0_ctrl, &info);
       uint64_t period = info.period_counts;
 /* The maximum period is one more than the maximum 32-bit number, but will be
reflected as 0 in
  * timer_info_t::period_counts. */
 if (0U == period)
      {
            period = UINT32_MAX + 1U;
      }
       g_captured_time    = (period * g_capture_overflows) + p_args->capture;
       g_capture_overflows = 0U;
    }
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* An overflow occurred during capture. This must be accounted for at the
application layer. */
       g_capture_overflows++;
    }
}
void gpt_capture_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
```

```
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Enable captures. Captured values arrive in the interrupt. */
    (void) R_GPT_Enable(&g_timer0_ctrl);
 /* (Optional) Disable captures. */
    (void) R_GPT_Disable(&g_timer0_ctrl);
}
```

### GPT Period Update Example

This an example of updating the period.

```
#define GPT_EXAMPLE_MSEC_PER_SEC (1000)
#define GPT_EXAMPLE_DESIRED_PERIOD_MSEC (20)
/* This example shows how to calculate a new period value at runtime. */
void gpt_period_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
 /* Get the source clock frequency (in Hz). There are 3 ways to do this in FSP:
  * - If the PCLKD frequency has not changed since reset, the source clock frequency
is
  * BSP_STARTUP_PCLKD_HZ >> timer_cfg_t::source_div
  * - Use the R_GPT_InfoGet function (it accounts for the divider).
  * - Calculate the current PCLKD frequency using
R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKD) and right shift
  * by timer_cfg_t::source_div.
  *
  * This example uses the 3rd option (R_FSP_SystemClockHzGet).
  */
```

```
    uint32_t pclkd_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKD) >>
g_timer0_cfg.source_div;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX / pclkd_freq_hz. A cast to uint64_t is
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) (((uint64_t) pclkd_freq_hz * GPT_EXAMPLE_DESIRED_PERIOD_MSEC) /
GPT_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. */
    err = R_GPT_PeriodSet(&g_timer0_ctrl, period_counts);
    handle_error(err);
}
```

### GPT Duty Cycle Update Example

This an example of updating the duty cycle.

```
#define GPT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT (25)
#define GPT_EXAMPLE_MAX_PERCENT (100)
/* This example shows how to calculate a new duty cycle value at runtime. */
void gpt_duty_cycle_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
 /* Get the current period setting. */
 timer_info_t info;
    (void) R_GPT_InfoGet(&g_timer0_ctrl, &info);
    uint32_t current_period_counts = info.period_counts;
 /* Calculate the desired duty cycle based on the current period. Note that if the
```

```
period could be larger than
  * UINT32_MAX / 100, this calculation could overflow. A cast to uint64_t is used to
prevent this. The cast is
  * not required for 16-bit timers. */
    uint32_t duty_cycle_counts =
       (uint32_t) (((uint64_t) current_period_counts *
GPT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                   GPT_EXAMPLE_MAX_PERCENT);
 /* Set the calculated duty cycle. */
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts, GPT_IO_PIN_GTIOCB);
    handle_error(err);
}
```

### GPT A/D Converter Start Request Example

This is an example of using the GPT to start the ADC at a configurable A/D converter compare match value.

```
#if ((1U << GPT_EXAMPLE_CHANNEL) & (BSP_FEATURE_GPTEH_CHANNEL_MASK |
BSP_FEATURE_GPTE_CHANNEL_MASK))
/* This example shows how to configure the GPT to generate an A/D start request at an
A/D start request compare
 * match value. This example can only be used with GPTE or GPTEH variants. */
void gpt_adc_start_request_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize and configure the ELC. */
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Configure the ELC to start a scan on ADC unit 0 when GPT channel 0. Note: This is
typically configured in
  * g_elc_cfg and already set during R_ELC_Open. */
    err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_ADC0, ELC_EVENT_GPT0_AD_TRIG_A);
    handle_error(err);
```

```
 /* Globally enable ELC events. */
    err = R_ELC_Enable(&g_elc_ctrl);
    handle_error(err);
 /* Initialize the ADC to start a scan based on an ELC event trigger. Set
adc_cfg_t::trigger to
  * ADC_TRIGGER_SYNC_ELC. */
    err = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
    handle_error(err);
    err = R_ADC_ScanCfg(&g_adc0_ctrl, &g_adc0_channel_cfg);
    handle_error(err);
 /* Enable ELC triggers by calling R_ADC_ScanStart(). */
    (void) R_ADC_ScanStart(&g_adc0_ctrl);
 /* Initializes the GPT module. Configure gpt_extended_pwm_cfg_t::adc_trigger to set
when the A/D start request
  * is generated. Set gpt_extended_pwm_cfg_t::adc_a_compare_match to set the desired
compare match value. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
    handle_error(err);
 /* Start the timer. A/D converter start request events are generated each time the
counter is equal to the
  * A/D start request compare match value. */
    (void) R_GPT_Start(&g_timer0_ctrl);
}
#endif
```

## Data Structures

| | |
|---|---|
| struct | gpt_output_pin_t |
| struct | gpt_instance_ctrl_t |
| struct | gpt_extended_pwm_cfg_t |
| struct | gpt_extended_cfg_t |

## Enumerations

| | |
|---|---|
| enum | gpt_io_pin_t |
| enum | gpt_pin_level_t |

| | |
|---|---|
| enum | gpt_shortest_level_t |
| enum | gpt_source_t |
| enum | gpt_capture_filter_t |
| enum | gpt_adc_trigger_t |
| enum | gpt_poeg_link_t |
| enum | gpt_output_disable_t |
| enum | gpt_gtioc_disable_t |
| enum | gpt_adc_compare_match_t |
| enum | gpt_interrupt_skip_source_t |
| enum | gpt_interrupt_skip_count_t |
| enum | gpt_interrupt_skip_adc_t |

## Data Structure Documentation

### ◆ gpt_output_pin_t

| struct gpt_output_pin_t | | |
|---|---|---|
| Configurations for output pins. | | |
| Data Fields | | |
| 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. |

### ◆ gpt_instance_ctrl_t

| struct gpt_instance_ctrl_t |
|---|
| Channel control block. DO NOT INITIALIZE. Initialization occurs when timer_api_t::open is called. |

### ◆ gpt_extended_pwm_cfg_t

| struct gpt_extended_pwm_cfg_t | | |
|---|---|---|
| GPT extension for advanced PWM features. | | |
| Data Fields | | |
| uint8_t | trough_ipl | Trough interrupt priority. |

| | | |
|---|---|---|
| IRQn_Type | trough_irq | Trough interrupt. |
| gpt_poeg_link_t | poeg_link | Select which POEG channel controls output disable for this GPT channel. |
| gpt_output_disable_t | output_disable | Select which trigger sources request output disable from POEG. |
| gpt_adc_trigger_t | adc_trigger | Select trigger sources to start A/D conversion. |
| uint32_t | dead_time_count_up | Set a dead time value for counting up. |
| uint32_t | dead_time_count_down | Set a dead time value for counting down (available on GPT32E and GPT32EH only) |
| uint32_t | adc_a_compare_match | Select the compare match value used to trigger an A/D conversion start request using ELC_EVENT_GPT<channel>_AD_TRIG_A. |
| uint32_t | adc_b_compare_match | Select the compare match value used to trigger an A/D conversion start request using ELC_EVENT_GPT<channel>_AD_TRIG_B. |
| gpt_interrupt_skip_source_t | interrupt_skip_source | Interrupt source to count for interrupt skipping. |
| gpt_interrupt_skip_count_t | interrupt_skip_count | Number of interrupts to skip between events. |
| gpt_interrupt_skip_adc_t | interrupt_skip_adc | ADC events to skip when interrupt skipping is enabled. |
| gpt_gtioc_disable_t | gtioca_disable_setting | Select how to configure GTIOCA when output is disabled. |
| gpt_gtioc_disable_t | gtiocb_disable_setting | Select how to configure GTIOCB when output is disabled. |

### ◆ gpt_extended_cfg_t

| struct gpt_extended_cfg_t | | |
|---|---|---|
| GPT extension configures the output pins for GPT. | | |
| Data Fields | | |
| gpt_output_pin_t | gtioca | Configuration for GPT I/O pin A. |
| gpt_output_pin_t | gtiocb | Configuration for GPT I/O pin B. |
| gpt_shortest_level_t | shortest_pwm_signal | Shortest PWM signal level. |
| gpt_source_t | start_source | Event sources that trigger the |

| | | |
|---|---|---|
| | | timer to start. |
| gpt_source_t | stop_source | Event sources that trigger the timer to stop. |
| gpt_source_t | clear_source | Event sources that trigger the timer to clear. |
| gpt_source_t | capture_a_source | Event sources that trigger capture of GTIOCA. |
| gpt_source_t | capture_b_source | Event sources that trigger capture of GTIOCB. |
| gpt_source_t | count_up_source | Event sources that trigger a single up count. If GPT_SOURCE_NONE is selected for both count_up_source and count_down_source, then the timer count source is PCLK. |
| gpt_source_t | count_down_source | Event sources that trigger a single down count. If GPT_SOURCE_NONE is selected for both count_up_source and count_down_source, then the timer count source is PCLK. |
| gpt_capture_filter_t | capture_filter_gtioca | |
| gpt_capture_filter_t | capture_filter_gtiocb | |
| uint8_t | capture_a_ipl | Capture A interrupt priority. |
| uint8_t | capture_b_ipl | Capture B interrupt priority. |
| IRQn_Type | capture_a_irq | Capture A interrupt. |
| IRQn_Type | capture_b_irq | Capture B interrupt. |
| gpt_extended_pwm_cfg_t const * | p_pwm_cfg | Advanced PWM features, optional. |

## Enumeration Type Documentation

### ◆ gpt_io_pin_t

| enum gpt_io_pin_t | |
|---|---|
| Input/Output pins, used to select which duty cycle to update in R_GPT_DutyCycleSet(). | |
| Enumerator | |
| GPT_IO_PIN_GTIOCA | GTIOCA. |
| GPT_IO_PIN_GTIOCB | GTIOCB. |
| GPT_IO_PIN_GTIOCA_AND_GTIOCB | GTIOCA and GTIOCB. |

### ◆ gpt_pin_level_t

| enum gpt_pin_level_t | |
|---|---|
| Level of GPT pin | |
| Enumerator | |
| GPT_PIN_LEVEL_LOW | Pin level low. |
| GPT_PIN_LEVEL_HIGH | Pin level high. |

### ◆ gpt_shortest_level_t

| enum gpt_shortest_level_t | |
|---|---|
| GPT PWM shortest pin level | |
| Enumerator | |
| GPT_SHORTEST_LEVEL_OFF | 1 extra PCLK in ON time. Minimum ON time will be limited to 2 PCLK raw counts. |
| GPT_SHORTEST_LEVEL_ON | 1 extra PCLK in OFF time. Minimum ON time will be limited to 1 PCLK raw counts. |

### ◆ gpt_source_t

| enum gpt_source_t | |
|---|---|
| Sources can be used to start the timer, stop the timer, count up, or count down. These enumerations represent a bitmask. Multiple sources can be ORed together. | |
| Enumerator | |
| GPT_SOURCE_NONE | No active event sources. |
| GPT_SOURCE_GTETRGA_RISING | Action performed on GTETRGA rising edge. |
| GPT_SOURCE_GTETRGA_FALLING | Action performed on GTETRGA falling edge. |
| GPT_SOURCE_GTETRGB_RISING | Action performed on GTETRGB rising edge. |
| GPT_SOURCE_GTETRGB_FALLING | Action performed on GTETRGB falling edge. |
| GPT_SOURCE_GTETRGC_RISING | Action performed on GTETRGC rising edge. |
| GPT_SOURCE_GTETRGC_FALLING | Action performed on GTETRGC falling edge. |
| GPT_SOURCE_GTETRGD_RISING | |

| | |
|---|---|
| | Action performed on GTETRGB rising edge. |
| GPT_SOURCE_GTETRGD_FALLING | Action performed on GTETRGB falling edge. |
| GPT_SOURCE_GTIOCA_RISING_WHILE_GTIOCB_LOW | Action performed when GTIOCA input rises while GTIOCB is low. |
| GPT_SOURCE_GTIOCA_RISING_WHILE_GTIOCB_HIGH | Action performed when GTIOCA input rises while GTIOCB is high. |
| GPT_SOURCE_GTIOCA_FALLING_WHILE_GTIOCB_LOW | Action performed when GTIOCA input falls while GTIOCB is low. |
| GPT_SOURCE_GTIOCA_FALLING_WHILE_GTIOCB_HIGH | Action performed when GTIOCA input falls while GTIOCB is high. |
| GPT_SOURCE_GTIOCB_RISING_WHILE_GTIOCA_LOW | Action performed when GTIOCB input rises while GTIOCA is low. |
| GPT_SOURCE_GTIOCB_RISING_WHILE_GTIOCA_HIGH | Action performed when GTIOCB input rises while GTIOCA is high. |
| GPT_SOURCE_GTIOCB_FALLING_WHILE_GTIOCA_LOW | Action performed when GTIOCB input falls while GTIOCA is low. |
| GPT_SOURCE_GTIOCB_FALLING_WHILE_GTIOCA_HIGH | Action performed when GTIOCB input falls while GTIOCA is high. |
| GPT_SOURCE_GPT_A | Action performed on ELC GPTA event. |
| GPT_SOURCE_GPT_B | Action performed on ELC GPTB event. |
| GPT_SOURCE_GPT_C | Action performed on ELC GPTC event. |
| GPT_SOURCE_GPT_D | Action performed on ELC GPTD event. |
| GPT_SOURCE_GPT_E | Action performed on ELC GPTE event. |
| GPT_SOURCE_GPT_F | Action performed on ELC GPTF event. |
| GPT_SOURCE_GPT_G | Action performed on ELC GPTG event. |
| GPT_SOURCE_GPT_H | Action performed on ELC GPTH event. |

### ◆ gpt_capture_filter_t

| enum gpt_capture_filter_t | |
|---|---|
| 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. | |
| Enumerator | |
| GPT_CAPTURE_FILTER_NONE | None - no filtering. |
| GPT_CAPTURE_FILTER_PCLKD_DIV_1 | PCLK/1 - fast sampling. |
| GPT_CAPTURE_FILTER_PCLKD_DIV_4 | PCLK/4. |
| GPT_CAPTURE_FILTER_PCLKD_DIV_16 | PCLK/16. |
| GPT_CAPTURE_FILTER_PCLKD_DIV_64 | PCLK/64 - slow sampling. |

### ◆ gpt_adc_trigger_t

| enum gpt_adc_trigger_t | |
|---|---|
| Trigger options to start A/D conversion. | |
| Enumerator | |
| GPT_ADC_TRIGGER_NONE | None - no output disable request. |
| GPT_ADC_TRIGGER_UP_COUNT_START_ADC_A | Request A/D conversion from ADC unit 0 at up counting compare match of gpt_extended_pwm_cfg_t::adc_a_compare_match. |
| GPT_ADC_TRIGGER_DOWN_COUNT_START_ADC_A | Request A/D conversion from ADC unit 0 at down counting compare match of gpt_extended_pwm_cfg_t::adc_a_compare_match. |
| GPT_ADC_TRIGGER_UP_COUNT_START_ADC_B | Request A/D conversion from ADC unit 1 at up counting compare match of gpt_extended_pwm_cfg_t::adc_b_compare_match. |
| GPT_ADC_TRIGGER_DOWN_COUNT_START_ADC_B | Request A/D conversion from ADC unit 1 at down counting compare match of gpt_extended_pwm_cfg_t::adc_b_compare_match. |

### ◆ gpt_poeg_link_t

| enum gpt_poeg_link_t | |
|---|---|
| POEG channel to link to this channel. | |
| Enumerator | |
| GPT_POEG_LINK_POEG0 | Link this GPT channel to POEG channel 0 (GTETRGA) |
| GPT_POEG_LINK_POEG1 | Link this GPT channel to POEG channel 1 (GTETRGB) |
| GPT_POEG_LINK_POEG2 | Link this GPT channel to POEG channel 2 (GTETRGC) |
| GPT_POEG_LINK_POEG3 | Link this GPT channel to POEG channel 3 (GTETRGD) |

### ◆ gpt_output_disable_t

| enum gpt_output_disable_t | |
|---|---|
| Select trigger to send output disable request to POEG. | |
| Enumerator | |
| GPT_OUTPUT_DISABLE_NONE | None - no output disable request. |
| GPT_OUTPUT_DISABLE_DEAD_TIME_ERROR | Request output disable if a dead time error occurs. |
| GPT_OUTPUT_DISABLE_GTIOCA_GTIOCB_HIGH | Request output disable if GTIOCA and GTIOCB are high at the same time. |
| GPT_OUTPUT_DISABLE_GTIOCA_GTIOCB_LOW | Request output disable if GTIOCA and GTIOCB are low at the same time. |

### ◆ gpt_gtioc_disable_t

| enum gpt_gtioc_disable_t | |
|---|---|
| Disable level options for GTIOC pins. | |
| Enumerator | |
| GPT_GTIOC_DISABLE_PROHIBITED | Do not allow output disable. |
| GPT_GTIOC_DISABLE_SET_HI_Z | Set GTIOC to high impedance when output is disabled. |
| GPT_GTIOC_DISABLE_LEVEL_LOW | Set GTIOC level low when output is disabled. |
| GPT_GTIOC_DISABLE_LEVEL_HIGH | Set GTIOC level high when output is disabled. |

### ◆ gpt_adc_compare_match_t

| enum gpt_adc_compare_match_t | |
|---|---|
| Trigger options to start A/D conversion. | |
| Enumerator | |
| GPT_ADC_COMPARE_MATCH_ADC_A | Set A/D conversion start request value for GPT A/D converter start request A. |
| GPT_ADC_COMPARE_MATCH_ADC_B | Set A/D conversion start request value for GPT A/D converter start request B. |

### ◆ gpt_interrupt_skip_source_t

| enum gpt_interrupt_skip_source_t | |
|---|---|
| Interrupt skipping modes | |
| Enumerator | |
| GPT_INTERRUPT_SKIP_SOURCE_NONE | Do not skip interrupts. |
| GPT_INTERRUPT_SKIP_SOURCE_OVERFLOW_UNDERFLOW | Count and skip overflow and underflow interrupts. |
| GPT_INTERRUPT_SKIP_SOURCE_CREST | Count crest interrupts for interrupt skipping. Skip the number of crest and trough interrupts configured in gpt_interrupt_skip_count_t. When the interrupt does fire, the trough interrupt fires before the crest interrupt. |
| GPT_INTERRUPT_SKIP_SOURCE_TROUGH | Count trough interrupts for interrupt skipping. Skip the number of crest and trough interrupts configured in gpt_interrupt_skip_count_t. When the interrupt does fire, the crest interrupt fires before the trough interrupt. |

### ◆ gpt_interrupt_skip_count_t

| enum gpt_interrupt_skip_count_t | |
|---|---|
| Number of interrupts to skip between events | |
| Enumerator | |
| GPT_INTERRUPT_SKIP_COUNT_0 | Do not skip interrupts. |
| GPT_INTERRUPT_SKIP_COUNT_1 | Skip one interrupt. |
| GPT_INTERRUPT_SKIP_COUNT_2 | Skip two interrupts. |
| GPT_INTERRUPT_SKIP_COUNT_3 | Skip three interrupts. |
| GPT_INTERRUPT_SKIP_COUNT_4 | Skip four interrupts. |
| GPT_INTERRUPT_SKIP_COUNT_5 | Skip five interrupts. |
| GPT_INTERRUPT_SKIP_COUNT_6 | Skip six interrupts. |
| GPT_INTERRUPT_SKIP_COUNT_7 | Skip seven interrupts. |

### ◆ gpt_interrupt_skip_adc_t

| enum gpt_interrupt_skip_adc_t | |
|---|---|
| ADC events to skip during interrupt skipping | |
| Enumerator | |
| GPT_INTERRUPT_SKIP_ADC_NONE | Do not skip ADC events. |
| GPT_INTERRUPT_SKIP_ADC_A | Skip ADC A events. |
| GPT_INTERRUPT_SKIP_ADC_B | Skip ADC B events. |
| GPT_INTERRUPT_SKIP_ADC_A_AND_B | Skip ADC A and B events. |

## Function Documentation

### ◆ R_GPT_Open()

fsp_err_t R_GPT_Open ( timer_ctrl_t *const *p_ctrl*, timer_cfg_t const *const *p_cfg* )

Initializes the timer module and applies configurations. Implements timer_api_t::open.

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 gpt_extended_cfg_t extension parameter.

Example:

```
/* Initializes the module. */
   err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful and timer has started. |
| FSP_ERR_ASSERTION | A required input pointer is NULL or the source divider is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_IRQ_BSP_DISABLED | timer_cfg_t::mode is TIMER_MODE_ONE_SHOT or timer_cfg_t::p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use one-shot mode or callback. |
| FSP_ERR_INVALID_MODE | Triangle wave PWM is only supported if GPT_CFG_OUTPUT_SUPPORT_ENABLE is 2. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel requested in the p_cfg parameter is not available on this device. |

### ◆ R_GPT_Stop()

fsp_err_t R_GPT_Stop ( timer_ctrl_t *const *p_ctrl*)

Stops timer. Implements timer_api_t::stop.

Example:

```
/* (Optional) Stop the timer. */
   (void) R_GPT_Stop(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer successfully stopped. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_Start()

fsp_err_t R_GPT_Start ( timer_ctrl_t *const *p_ctrl*)

Starts timer. Implements timer_api_t::start.

Example:

```
/* Start the timer. */
   (void) R_GPT_Start(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer successfully started. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_Reset()

fsp_err_t R_GPT_Reset ( timer_ctrl_t *const *p_ctrl*)

Resets the counter value to 0. Implements timer_api_t::reset.

***Note***

*This function also updates to the new period if no counter overflow has occurred since the last call to R_GPT_PeriodSet().*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter value written successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_Enable()

fsp_err_t R_GPT_Enable ( timer_ctrl_t *const *p_ctrl*)

Enables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::enable.

Example:

```
/* Enable captures. Captured values arrive in the interrupt. */
   (void) R_GPT_Enable(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | External events successfully enabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_Disable()

fsp_err_t R_GPT_Disable ( timer_ctrl_t *const *p_ctrl*)

Disables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::disable.

***Note***

*The timer could be running after R_GPT_Disable(). To ensure it is stopped, call R_GPT_Stop().*

Example:

```
/* (Optional) Disable captures. */
   (void) R_GPT_Disable(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | External events successfully disabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_PeriodSet()

fsp_err_t R_GPT_PeriodSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *period_counts* )

Sets period value provided. If the timer is running, the period will be updated after the next counter overflow. If the timer is stopped, this function resets the counter and updates the period. Implements timer_api_t::periodSet.

**Warning**

If periodic output is used, the duty cycle buffer registers are updated after the period buffer register. If this function is called while the timer is running and a GPT overflow occurs during processing, the duty cycle will not be the desired 50% duty cycle until the counter overflow after processing completes.

Example:

```
/* Get the source clock frequency (in Hz). There are 3 ways to do this in FSP:
 * - If the PCLKD frequency has not changed since reset, the source clock frequency
is
 * BSP_STARTUP_PCLKD_HZ >> timer_cfg_t::source_div
 * - Use the R_GPT_InfoGet function (it accounts for the divider).
 * - Calculate the current PCLKD frequency using
R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKD) and right shift
 * by timer_cfg_t::source_div.
 *
```

```
 * This example uses the 3rd option (R_FSP_SystemClockHzGet).
 */
   uint32_t pclkd_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKD) >>
g_timer0_cfg.source_div;
/* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
 * desired period is larger than UINT32_MAX / pclkd_freq_hz. A cast to uint64_t is
used to prevent this. */
   uint32_t period_counts =
      (uint32_t) (((uint64_t) pclkd_freq_hz * GPT_EXAMPLE_DESIRED_PERIOD_MSEC) /
GPT_EXAMPLE_MSEC_PER_SEC);
/* Set the calculated period. */
   err = R_GPT_PeriodSet(&g_timer0_ctrl, period_counts);
   handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Period value written successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_DutyCycleSet()

fsp_err_t R_GPT_DutyCycleSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *duty_cycle_counts*, uint32_t const *pin* )

Sets duty cycle on requested pin. Implements timer_api_t::dutyCycleSet.

Duty cycle is updated in the buffer register. The updated duty cycle is reflected after the next cycle end (counter overflow).

Example:

```
/* Get the current period setting. */
timer_info_t info;
   (void) R_GPT_InfoGet(&g_timer0_ctrl, &info);
   uint32_t current_period_counts = info.period_counts;
/* Calculate the desired duty cycle based on the current period. Note that if the
period could be larger than
 * UINT32_MAX / 100, this calculation could overflow. A cast to uint64_t is used to
```

```
prevent this. The cast is
  * not required for 16-bit timers. */
    uint32_t duty_cycle_counts =
       (uint32_t) (((uint64_t) current_period_counts *
GPT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                   GPT_EXAMPLE_MAX_PERCENT);
 /* Set the calculated duty cycle. */
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts, GPT_IO_PIN_GTIOCB);
    handle_error(err);
```

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block. |
| [in] | duty_cycle_counts | Duty cycle to set in counts. |
| [in] | pin | Use gpt_io_pin_t to select GPT_IO_PIN_GTIOCA or GPT_IO_PIN_GTIOCB |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Duty cycle updated successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL or the pin is not one of gpt_io_pin_t |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_INVALID_ARGUMENT | Duty cycle is larger than period. |
| FSP_ERR_UNSUPPORTED | GPT_CFG_OUTPUT_SUPPORT_ENABLE is 0. |

### ◆ R_GPT_InfoGet()

fsp_err_t R_GPT_InfoGet ( timer_ctrl_t *const *p_ctrl*, timer_info_t *const *p_info* )

Get timer information and store it in provided pointer p_info. Implements timer_api_t::infoGet.

Example:

```
/* (Optional) Get the current period if not known. */
timer_info_t info;
      (void) R_GPT_InfoGet(&g_timer0_ctrl, &info);
      uint64_t period = info.period_counts;
/* The maximum period is one more than the maximum 32-bit number, but will be
reflected as 0 in
 * timer_info_t::period_counts. */
if (0U == period)
     {
           period = UINT32_MAX + 1U;
     }
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Period, count direction, frequency, and ELC event written to caller's structure successfully. |
| FSP_ERR_ASSERTION | p_ctrl or p_info was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_StatusGet()

fsp_err_t R_GPT_StatusGet ( timer_ctrl_t *const *p_ctrl*, timer_status_t *const *p_status* )

Get current timer status and store it in provided pointer p_status. Implements timer_api_t::statusGet.

Example:

```
/* Read the current counter value. Counter value is in status.counter. */
timer_status_t status;
   (void) R_GPT_StatusGet(&g_timer0_ctrl, &status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Current timer state and counter value set successfully. |
| FSP_ERR_ASSERTION | p_ctrl or p_status was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_CounterSet()

fsp_err_t R_GPT_CounterSet ( timer_ctrl_t *const *p_ctrl*, uint32_t *counter* )

Set counter value.

***Note***

*Do not call this API while the counter is counting. The counter value can only be updated while the counter is stopped.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter value updated. |
| FSP_ERR_ASSERTION | p_ctrl or p_status was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_IN_USE | The timer is running. Stop the timer before calling this function. |

### ◆ R_GPT_OutputEnable()

fsp_err_t R_GPT_OutputEnable ( timer_ctrl_t *const *p_ctrl*, gpt_io_pin_t *pin* )

Enable output for GTIOCA and/or GTIOCB.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Output is enabled. |
| FSP_ERR_ASSERTION | p_ctrl or p_status was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_OutputDisable()

fsp_err_t R_GPT_OutputDisable ( timer_ctrl_t *const *p_ctrl*, gpt_io_pin_t *pin* )

Disable output for GTIOCA and/or GTIOCB.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Output is disabled. |
| FSP_ERR_ASSERTION | p_ctrl or p_status was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_AdcTriggerSet()

fsp_err_t R_GPT_AdcTriggerSet ( timer_ctrl_t *const *p_ctrl*, gpt_adc_compare_match_t *which_compare_match*, uint32_t *compare_match_value* )

Set A/D converter start request compare match value.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter value updated. |
| FSP_ERR_ASSERTION | p_ctrl or p_status was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_Close()

| fsp_err_t R_GPT_Close ( timer_ctrl_t *const *p_ctrl*) |
|---|
| Stops counter, disables output pins, and clears internal driver data. Implements timer_api_t::close. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_VersionGet()

| fsp_err_t R_GPT_VersionGet ( fsp_version_t *const *p_version*) |
|---|
| Sets driver version based on compile time macros. Implements timer_api_t::versionGet. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version stored in p_version. |
| FSP_ERR_ASSERTION | p_version was NULL. |

## 4.2.24 General PWM Timer Three-Phase Motor Control Driver (r_gpt_three_phase)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_GPT_THREE_PHASE_Open (three_phase_ctrl_t *const p_ctrl, three_phase_cfg_t const *const p_cfg) |
| fsp_err_t | R_GPT_THREE_PHASE_Stop (three_phase_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_THREE_PHASE_Start (three_phase_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_THREE_PHASE_Reset (three_phase_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_THREE_PHASE_DutyCycleSet (three_phase_ctrl_t *const p_ctrl, three_phase_duty_cycle_t *const p_duty_cycle) |
| fsp_err_t | R_GPT_THREE_PHASE_Close (three_phase_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | R_GPT_THREE_PHASE_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Driver for 3-phase motor control using the GPT peripheral on RA MCUs. This module implements the Three-Phase Interface.

# Overview

The General PWM Timer (GPT) Three-Phase driver provides basic functionality for synchronously starting and stopping three PWM channels for use in 3-phase motor control applications. A function is additionally provided to allow setting duty cycle values for all three channels, optionally with double-buffering.

### Features

The GPT Three-Phase driver provides the following functions:

- Synchronize configuration of three GPT channels
- Synchronously start, stop and reset all three GPT channels
- Set duty cycle on all three channels with one function

# Configuration

### Build Time Configurations for r_gpt_three_phase

The following build time configurations are defined in fsp_cfg/r_gpt_three_phase_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Timers > Three-Phase PWM Driver on r_gpt_three_phase

This module can be added to the Stacks tab via New Stack > Driver > Timers > Three-Phase PWM Driver on r_gpt_three_phase:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General > Name | Name must be a valid C symbol | g_three_phase0 | Module name. |
| General > Mode | • Triangle-Wave Symmetric PWM<br>• Triangle-Wave | Triangle-Wave Symmetric PWM | Mode selection. Triangle-Wave Symmetric PWM: Generates symmetric |

| | | | |
|---|---|---|---|
| | Asymmetric PWM | | PWM waveforms with duty cycle determined by compare match set during a crest interrupt and updated at the next trough. Triangle-Wave Asymmetric PWM: Generates asymmetric PWM waveforms with duty cycle determined by compare match set during a crest/trough interrupt and updated at the next trough/crest. |
| General > Period | Value must be a non-negative integer less than or equal to 0x40000000000 | 15 | Specify the timer period in units selected below. Setting the period to 0x100000000 raw counts results in the maximum period. Set the period to 0x100000000 raw counts for a free running timer or an input capture configuration. The period can be set up to 0x40000000000, which will use a divider of 1024 with the maximum period. |
| General > Period Unit | • Raw Counts<br>• Nanoseconds<br>• Microseconds<br>• Milliseconds<br>• Seconds<br>• Hertz<br>• Kilohertz | Kilohertz | Unit of the period specified above |
| General > GPT U-Channel | Channel number must exist on the device | 0 | Specify the GPT channel for U signal output. |
| General > GPT V-Channel | Channel number must exist on the device | 1 | Specify the GPT channel for V signal output. |
| General > GPT W-Channel | Channel number must exist on the device | 2 | Specify the GPT channel for W signal output. |
| General > Buffer Mode | • Single Buffer<br>• Double Buffer | Single Buffer | When Double Buffer is selected the |

| | | | |
|---|---|---|---|
| | | | 'duty_buffer' array in three_phase_duty_cycle_t is used as a buffer for the 'duty' array. This allows setting the duty cycle for the next two crest/trough events in asymmetric mode with only one call to R_GPT_THREE_PHASE_DutyCycleSet. |
| General > GTIOCA Stop Level | • Pin Level Low<br>• Pin Level High<br>• Pin Level Retained | Pin Level Low | Select the behavior of the output pin when the timer is stopped. |
| General > GTIOCB Stop Level | • Pin Level Low<br>• Pin Level High<br>• Pin Level Retained | Pin Level Low | Select the behavior of the output pin when the timer is stopped. |
| Dead Time > Dead Time Count Up (Raw Counts) | Must be an integer greater than or equal to 0 | 0 | Select the dead time to apply during up counting. This value also applies during down counting for the GPT32 and GPT16 variants. |
| Dead Time > Dead Time Count Down (Raw Counts) (GPTE/GPTEH only) | Must be an integer greater than or equal to 0 | 0 | Select the dead time to apply during down counting. This value only applies to the GPT32E and GPT32EH variants. |

### Clock Configuration

Please refer to the General PWM Timer (r_gpt) section for more information.

### Pin Configuration

Please refer to the General PWM Timer (r_gpt) section for more information.

## Usage Notes

Warning

Be sure the GTIOCA/B stop level and dead time values are set appropriately for your application before attempting to drive a motor. Failure to do so may result in damage to the motor drive circuitry and/or the motor itself if the timer is stopped by software.

### Initial Setup

The following should be configured once the GPT Three-Phase module has been added to a project:

1. Set "Pin Output Support" in one of the GPT submodules to "Enabled with Extra Features"
2. Configure common settings in the GPT Three-Phase module properties
3. Set the crest and trough interrupt priority and callback function in **one** of the three GPT submodules (if desired)
4. Set the "Extra Features -> Output Disable" settings in each GPT submodule as needed for your application

*Note*

*Because all three modules are operated synchronously with the same period interrupts only need to be enabled in one of the three GPT modules.*

### Buffer Modes

There are two buffering modes available for duty cycle values - single- and double-buffered. In single buffer mode only the values specified in the duty array element of three_phase_duty_cycle_t are used by R_GPT_THREE_PHASE_DutyCycleSet. At the next trough or crest event the output duty cycle will be internally updated to the set values.

In double buffer mode the duty_buffer array values are used as buffer values for the duty elements. Once passed to R_GPT_THREE_PHASE_DutyCycleSet, the next trough or crest event will update the output duty cycle to the values specified in duty as before. However, at the following crest or trough event the output duty cycle will be updated to the values in duty_buffer. This allows the duty cycle for both sides of an asymmetric PWM waveform to be set at only one trough or crest event per period instead of at every event.

## Examples

### GPT Three-Phase Basic Example

This is a basic example of minimal use of the GPT Three-Phase module in an application. The duty cycle is updated at every timer trough with the previously loaded buffer value, then the duty cycle buffer is reloaded in the trough interrupt callback.

```
void gpt_callback (timer_callback_args_t * p_args)
{
 fsp_err_t                err;
 three_phase_duty_cycle_t duty_cycle;
 if (TIMER_EVENT_TROUGH == p_args->event)
    {
 /* Update duty cycle values (example) */
       duty_cycle.duty[THREE_PHASE_CHANNEL_U] =
get_duty_counts(THREE_PHASE_CHANNEL_U);
       duty_cycle.duty[THREE_PHASE_CHANNEL_V] =
get_duty_counts(THREE_PHASE_CHANNEL_V);
       duty_cycle.duty[THREE_PHASE_CHANNEL_W] =
get_duty_counts(THREE_PHASE_CHANNEL_W);
```

```
 /* Update duty cycle values */
       err = R_GPT_THREE_PHASE_DutyCycleSet(&g_gpt_three_phase_ctrl, &duty_cycle);
       handle_error(err);
    }
 else
    {
 /* Handle crest event. */
    }
}
void gpt_three_phase_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_THREE_PHASE_Open(&g_gpt_three_phase_ctrl, &g_gpt_three_phase_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Start the timer. */
    (void) R_GPT_THREE_PHASE_Start(&g_gpt_three_phase_ctrl);
}
```

## Data Structures

| | |
|---|---|
| struct | gpt_three_phase_instance_ctrl_t |

## Data Structure Documentation

### ◆ gpt_three_phase_instance_ctrl_t

| struct gpt_three_phase_instance_ctrl_t |
|---|
| Channel control block. DO NOT INITIALIZE. Initialization occurs when three_phase_api_t::open is called. |

## Function Documentation

### ◆ R_GPT_THREE_PHASE_Open()

fsp_err_t R_GPT_THREE_PHASE_Open ( three_phase_ctrl_t *const *p_ctrl*, three_phase_cfg_t const *const *p_cfg* )

Initializes the 3-phase timer module (and associated timers) and applies configurations. Implements three_phase_api_t::open.

Example:

```
/* Initializes the module. */
   err = R_GPT_THREE_PHASE_Open(&g_gpt_three_phase_ctrl, &g_gpt_three_phase_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful. |
| FSP_ERR_ASSERTION | A required input pointer is NULL. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |

### ◆ R_GPT_THREE_PHASE_Stop()

fsp_err_t R_GPT_THREE_PHASE_Stop ( three_phase_ctrl_t *const *p_ctrl*)

Stops all timers synchronously. Implements three_phase_api_t::stop.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timers successfully stopped. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_THREE_PHASE_Start()

fsp_err_t R_GPT_THREE_PHASE_Start ( three_phase_ctrl_t *const *p_ctrl*)

Starts all timers synchronously. Implements three_phase_api_t::start.

Example:

```
/* Start the timer. */
   (void) R_GPT_THREE_PHASE_Start(&g_gpt_three_phase_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timers successfully started. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_THREE_PHASE_Reset()

fsp_err_t R_GPT_THREE_PHASE_Reset ( three_phase_ctrl_t *const *p_ctrl*)

Resets the counter values to 0. Implements three_phase_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counters were reset successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_THREE_PHASE_DutyCycleSet()

fsp_err_t R_GPT_THREE_PHASE_DutyCycleSet ( three_phase_ctrl_t *const *p_ctrl*, three_phase_duty_cycle_t *const *p_duty_cycle* )

Sets duty cycle for all three timers. Implements three_phase_api_t::dutyCycleSet.

In symmetric PWM mode duty cycle values are reflected after the next trough. In asymmetric PWM mode values are reflected at the next trough OR crest, whichever comes first.

When double-buffering is enabled the values in three_phase_duty_cycle_t::duty_buffer are set to the double-buffer registers. When values are reflected the first time the single buffer values (three_phase_duty_cycle_t::duty) are used. On the second reflection the duty_buffer values are used. In asymmetric PWM mode this enables both count-up and count-down PWM values to be set at trough (or crest) exclusively.

Example:

```
/* Update duty cycle values */
      err = R_GPT_THREE_PHASE_DutyCycleSet(&g_gpt_three_phase_ctrl, &duty_cycle);
      handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Duty cycle updated successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_INVALID_ARGUMENT | One or more duty cycle count values was outside the range 0..(period - 1). |

### ◆ R_GPT_THREE_PHASE_Close()

fsp_err_t R_GPT_THREE_PHASE_Close ( three_phase_ctrl_t *const *p_ctrl*)

Stops counters, disables output pins, and clears internal driver data. Implements three_phase_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_THREE_PHASE_VersionGet()

fsp_err_t R_GPT_THREE_PHASE_VersionGet ( fsp_version_t *const *p_version*)

Sets driver version based on compile time macros. Implements three_phase_api_t::versionGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version stored in p_version. |
| FSP_ERR_ASSERTION | p_version was NULL. |

## 4.2.25 Interrupt Controller Unit (r_icu)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_ICU_ExternalIrqOpen (external_irq_ctrl_t *const p_api_ctrl, external_irq_cfg_t const *const p_cfg) |
| fsp_err_t | R_ICU_ExternalIrqEnable (external_irq_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_ICU_ExternalIrqDisable (external_irq_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_ICU_ExternalIrqVersionGet (fsp_version_t *const p_version) |
| fsp_err_t | R_ICU_ExternalIrqClose (external_irq_ctrl_t *const p_api_ctrl) |

**Detailed Description**

Driver for the ICU peripheral on RA MCUs. This module implements the External IRQ Interface.

# Overview

The Interrupt Controller Unit (ICU) controls which event signals are linked to the NVIC, DTC, and DMAC modules. The R_ICU software module only implements the External IRQ Interface. The external_irq interface is for configuring interrupts to fire when a trigger condition is detected on an external IRQ pin.

*Note*

*Multiple instances are used when more than one external interrupt is needed. Configure each instance with different channels and properties as needed for the specific interrupt.*

### Features

- Supports configuring interrupts for IRQ pins on the target MCUs
  - Enabling and disabling interrupt generation.
  - Configuring interrupt trigger on rising edge, falling edge, both edges, or low level signal.
  - Enabling and disabling the IRQ noise filter.
- Supports configuring a user callback function, which will be invoked by the HAL module when an external pin interrupt is generated.

# Configuration

### Build Time Configurations for r_icu

The following build time configurations are defined in fsp_cfg/r_icu_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Input > External IRQ Driver on r_icu

This module can be added to the Stacks tab via New Stack > Driver > Input > External IRQ Driver on r_icu:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_external_irq0 | Module name. |
| Channel | Value must be an integer between 0 and 15 | 0 | Specify the hardware channel. |
| Trigger | • Falling | Rising | Select the signal edge |

| | | | |
|---|---|---|---|
| | • Rising<br>• Both Edges<br>• Low Level | | or state that triggers an interrupt. |
| Digital Filtering | • Enabled<br>• Disabled | Disabled | Select if the digital noise filter should be enabled. |
| Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled) | • PCLK / 1<br>• PCLK / 8<br>• PCLK / 32<br>• PCLK / 64 | PCLK / 64 | Select the clock divider for the digital noise filter. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided here. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the IRQn triggers |
| Pin Interrupt Priority | MCU Specific Options | | Select the PIN interrupt priority. |

### Clock Configuration

The ICU peripheral module doesn't require any specific clock settings.

*Note*

*The digital filter uses PCLKB as the clock source for sampling the IRQ pin.*

### Pin Configuration

The pin for the external interrupt channel must be configured as an input with IRQ Input Enabled.

## Usage Notes

### Digital Filter

The digital filter is used to reject trigger conditions that are too short. The trigger condition must be longer than three periods of the filter clock. The filter clock frequency is determined by PCLKB and the external_irq_pclk_div_t setting.

MIN_PULSE_WIDTH = EXTERNAL_IRQ_PCLKB_DIV / PCLKB_FREQUENCY * 3

### DMAC/DTC

When using an External IRQ pin to trigger a DMAC/DTC transfer, the External IRQ pin must be opened before the transfer instance is opened.

## Examples

### Basic Example

This is a basic example of minimal use of the ICU in an application.

```
#define ICU_IRQN_PIN BSP_IO_PORT_02_PIN_06
#define ICU_IRQN 6
/* Called from icu_irq_isr */
void external_irq_callback (external_irq_callback_args_t * p_args)
{
    (void) p_args;
    g_external_irq_complete = 1;
}
void simple_example ()
{
 /* Example Configuration */
 external_irq_cfg_t icu_cfg =
    {
       .channel       = ICU_IRQN,
       .trigger       = EXTERNAL_IRQ_TRIG_RISING,
       .filter_enable = false,
       .pclk_div      = EXTERNAL_IRQ_PCLK_DIV_BY_1,
       .p_callback    = external_irq_callback,
       .p_context    = 0,
       .ipl          = 0,
       .irq          = (IRQn_Type) 0,
    };
 /* Configure the external interrupt. */
 fsp_err_t err = R_ICU_ExternalIrqOpen(&g_icu_ctrl, &icu_cfg);
    handle_error(err);
 /* Enable the external interrupt. */
 /* Enable not required when used with ELC or DMAC. */
    err = R_ICU_ExternalIrqEnable(&g_icu_ctrl);
    handle_error(err);
 while (0 == g_external_irq_complete)
    {
 /* Wait for interrupt. */
    }
```

```
}
```

## Data Structures

| | |
|---|---|
| struct | icu_instance_ctrl_t |

## Data Structure Documentation

### ◆ icu_instance_ctrl_t

struct icu_instance_ctrl_t

ICU private control block. DO NOT MODIFY. Initialization occurs when R_ICU_ExternalIrqOpen is called.

**Data Fields**

| | |
|---|---|
| uint32_t | open |
| | Used to determine if channel control block is in use. |
| | |
| IRQn_Type | irq |
| | NVIC interrupt number. |
| | |
| uint8_t | channel |
| | Channel. |
| | |
| void(* | p_callback )(external_irq_callback_args_t *p_args) |
| | |
| void const * | p_context |
| | |

## Field Documentation

### ◆ p_callback

void(* icu_instance_ctrl_t::p_callback) (external_irq_callback_args_t *p_args)

Callback provided when a external IRQ ISR occurs. Set to NULL for no CPU interrupt.

### ◆ p_context

void const* icu_instance_ctrl_t::p_context

Placeholder for user data. Passed to the user callback in external_irq_callback_args_t.

## Function Documentation

### ◆ R_ICU_ExternalIrqOpen()

fsp_err_t R_ICU_ExternalIrqOpen ( external_irq_ctrl_t *const *p_api_ctrl*, external_irq_cfg_t const *const *p_cfg* )

Configure an IRQ input pin for use with the external interrupt interface. Implements external_irq_api_t::open.

The Open function is responsible for preparing an external IRQ pin for operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | One of the following is invalid:<br>• p_ctrl or p_cfg is NULL |
| FSP_ERR_ALREADY_OPEN | The channel specified has already been opened. No configurations were changed. Call the associated Close function to reconfigure the channel. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel requested in p_cfg is not available on the device selected in r_bsp_cfg.h. |
| FSP_ERR_INVALID_ARGUMENT | p_cfg->p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use callback function. |

***Note***

*This function is reentrant for different channels. It is not reentrant for the same channel.*

### ◆ R_ICU_ExternalIrqEnable()

fsp_err_t R_ICU_ExternalIrqEnable ( external_irq_ctrl_t *const *p_api_ctrl*)

Enable external interrupt for specified channel at NVIC. Implements external_irq_api_t::enable.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Interrupt Enabled successfully. |
| FSP_ERR_ASSERTION | The p_ctrl parameter was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |
| FSP_ERR_IRQ_BSP_DISABLED | Requested IRQ is not defined in this system |

### ◆ R_ICU_ExternalIrqDisable()

fsp_err_t R_ICU_ExternalIrqDisable ( external_irq_ctrl_t *const *p_api_ctrl*)

Disable external interrupt for specified channel at NVIC. Implements external_irq_api_t::disable.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Interrupt disabled successfully. |
| FSP_ERR_ASSERTION | The p_ctrl parameter was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |
| FSP_ERR_IRQ_BSP_DISABLED | Requested IRQ is not defined in this system |

### ◆ R_ICU_ExternalIrqVersionGet()

fsp_err_t R_ICU_ExternalIrqVersionGet ( fsp_version_t *const *p_version*)

Set driver version based on compile time macros. Implements external_irq_api_t::versionGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | The parameter p_version is NULL. |

### ◆ R_ICU_ExternalIrqClose()

fsp_err_t R_ICU_ExternalIrqClose ( external_irq_ctrl_t *const *p_api_ctrl*)

Close the external interrupt channel. Implements external_irq_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |

## 4.2.26 I2C Master on IIC (r_iic_master)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | R_IIC_MASTER_Open (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_cfg_t const *const p_cfg) |
| fsp_err_t | R_IIC_MASTER_Read (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart) |
| fsp_err_t | R_IIC_MASTER_Write (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart) |
| fsp_err_t | R_IIC_MASTER_Abort (i2c_master_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_IIC_MASTER_SlaveAddressSet (i2c_master_ctrl_t *const p_api_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode) |
| fsp_err_t | R_IIC_MASTER_Close (i2c_master_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_IIC_MASTER_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Driver for the IIC peripheral on RA MCUs. This module implements the I2C Master Interface.

# Overview

The I2C master on IIC HAL module supports transactions with an I2C Slave device. Callbacks must be provided which are invoked when a transmit or receive operation has completed. The callback argument will contain information about the transaction status, bytes transferred and a pointer to the user defined context.

### Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100-kHz transaction rate.
  - Fast Mode Support with up to 400-kHz transaction rate.
  - Fast Mode Plus Support with up to 1-MHz transaction rate.
- I2C Master Read from a slave device.
- I2C Master Write to a slave device.
- Abort any in-progress transactions.
- Set the address of the slave device.
- Non-blocking behavior is achieved by the use of callbacks.
- Additional build-time features
  - Optional (build time) DTC support for read and write respectively.
  - Optional (build time) support for 10-bit slave addressing.

# Configuration

### Build Time Configurations for r_iic_master

The following build time configurations are defined in fsp_cfg/r_iic_master_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| DTC on Transmission and Reception | • Enabled<br>• Disabled | Disabled | If enabled, DTC instances will be included in the build for both transmission and reception. |
| 10-bit slave addressing | • Enabled<br>• Disabled | Disabled | If enabled, the driver will support 10-bit slave addressing mode along with the default 7-bit slave addressing mode. |

**Configurations for Driver > Connectivity > I2C Master Driver on r_iic_master**

This module can be added to the Stacks tab via New Stack > Driver > Connectivity > I2C Master Driver on r_iic_master:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_i2c_master0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Specify the IIC channel. |
| Rate | • Standard<br>• Fast-mode<br>• Fast-mode plus | Standard | Select the transfer rate. |
| Rise Time (ns) | Value must be a non-negative integer | 120 | Set the rise time (tr) in nanoseconds. |
| Fall Time (ns) | Value must be a non-negative integer | 120 | Set the fall time (tf) in nanoseconds. |
| Duty Cycle (%) | Value must be an integer between 0 and 100 | 50 | Set the SCL duty cycle. |
| Slave Address | Value must be non-negative | 0x00 | Specify the slave address. |
| Address Mode | • 7-Bit<br>• 10-Bit | 7-Bit | Select the slave address mode. Ensure 10-bit slave addressing is enabled in the configuration to use 10-Bit setting here. |

| | | | |
|---|---|---|---|
| Timeout Mode | • Short Mode<br>• Long Mode | Short Mode | Select the timeout mode to detect bus hang. |
| Callback | Name must be a valid C symbol | i2c_master_callback | A user callback function must be provided. This will be called from the interrupt service routine (ISR) upon IIC transaction completion reporting the transaction status. |
| Interrupt Priority Level | MCU Specific Options | | Select the interrupt priority level. This is set for TXI, RXI, TEI and ERI interrupts. |

### Clock Configuration

The IIC peripheral module uses the PCLKB as its clock source. The actual I2C transfer rate will be calculated and set by the tooling 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.

### 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. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

## Usage Notes

### Interrupt Configuration

- The IIC 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.
- Set equal priority levels for all the interrupts mentioned above. Setting the interrupts to different priority levels could result in improper operation.

### IIC Master Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate. 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 by the tool.

### Enabling DTC with the IIC

- DTC transfer support is configurable and is disabled from the build by default. IIC driver provides two DTC instances for transmission and reception respectively. The DTC instances can be enabled individually during configuration.
- For further details on DTC please refer Data Transfer Controller (r_dtc)

### Multiple Devices on the Bus

- A single IIC instance can be used to communicate with multiple slave devices on the same channel by using the SlaveAddressSet 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.

### Restart

- IIC master can hold the the bus after an I2C transaction by issuing Restart. This will mimic a stop followed by start condition.

## Examples

### Basic Example

This is a basic example of minimal use of the r_iic_master in an application. This example shows how this driver can be used for basic read and write operations.

```
iic_master_instance_ctrl_t g_i2c_device_ctrl_1;
i2c_master_cfg_t g_i2c_device_cfg_1 =
{
    .channel       = I2C_CHANNEL,
    .rate          = I2C_MASTER_RATE_FAST,
    .slave         = I2C_SLAVE_EEPROM,
    .addr_mode    = I2C_MASTER_ADDR_MODE_7BIT,
    .p_callback    = i2c_callback,    // Callback
    .p_context    = &g_i2c_device_ctrl_1,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_master_cfg_extend
};
void i2c_callback (i2c_master_callback_args_t * p_args)
{
    g_i2c_callback_event = p_args->event;
}
void basic_example (void)
{
 fsp_err_t err;
```

```
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
 /* Initialize the IIC module */
    err = R_IIC_MASTER_Open(&g_i2c_device_ctrl_1, &g_i2c_device_cfg_1);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_tx_buffer[i] = (uint8_t) i;
    }
 /* Send data to I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_IIC_MASTER_Write(&g_i2c_device_ctrl_1, &g_i2c_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    handle_error(err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Read data back from the I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_MASTER_Read(&g_i2c_device_ctrl_1, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    handle_error(err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
```

```
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
        timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
        __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_tx_buffer, g_i2c_rx_buffer, I2C_BUFFER_SIZE_BYTES))
    {
        __BKPT(0);
    }
}
```

### Multiple Slave devices on the same channel (bus)

This example demonstrates how a single IIC driver can be used to communicate with different slave devices which are on the same channel.

*Note*

*The callback function from the first example applies to this example as well.*

```
iic_master_instance_ctrl_t g_i2c_device_ctrl_2;
i2c_master_cfg_t g_i2c_device_cfg_2 =
{
    .channel       = I2C_CHANNEL,
    .rate          = I2C_MASTER_RATE_STANDARD,
    .slave         = I2C_SLAVE_TEMP_SENSOR,
    .addr_mode    = I2C_MASTER_ADDR_MODE_7BIT,
    .p_callback    = i2c_callback,    // Callback
    .p_context    = &g_i2c_device_ctrl_2,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_master_cfg_extend
};
void single_channel_multi_slave (void)
```

```
{
 fsp_err_t err;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_MASTER_Open(&g_i2c_device_ctrl_2, &g_i2c_device_cfg_2);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Clear the recieve buffer */
    memset(g_i2c_rx_buffer, '0', I2C_BUFFER_SIZE_BYTES);
 /* Read data from I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_IIC_MASTER_Read(&g_i2c_device_ctrl_2, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    handle_error(err);
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Send data to I2C slave on the same channel */
    err = R_IIC_MASTER_SlaveAddressSet(&g_i2c_device_ctrl_2,
I2C_SLAVE_DISPLAY_ADAPTER, I2C_MASTER_ADDR_MODE_7BIT);
    handle_error(err);
    g_i2c_tx_buffer[0]   = 0xAA;       // NOLINT
    g_i2c_tx_buffer[1]   = 0xBB;       // NOLINT
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms           = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_MASTER_Write(&g_i2c_device_ctrl_2, &g_i2c_tx_buffer[0], 2U, false);
    handle_error(err);
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
```

```
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
}
```

## Data Structures

| | |
|---|---|
| struct | iic_master_clock_settings_t |
| struct | iic_master_instance_ctrl_t |
| struct | iic_master_extended_cfg_t |

## Enumerations

| | |
|---|---|
| enum | iic_master_timeout_mode_t |

## Data Structure Documentation

### ◆ iic_master_clock_settings_t

| struct iic_master_clock_settings_t | | |
|---|---|---|
| I2C clock settings | | |
| Data Fields | | |
| uint8_t | cks_value | Internal Reference Clock Select. |
| uint8_t | brh_value | High-level period of SCL clock. |
| uint8_t | brl_value | Low-level period of SCL clock. |

### ◆ iic_master_instance_ctrl_t

| struct iic_master_instance_ctrl_t |
|---|
| I2C control structure. DO NOT INITIALIZE. |

### ◆ iic_master_extended_cfg_t

| struct iic_master_extended_cfg_t | | |
|---|---|---|
| R_IIC extended configuration | | |
| Data Fields | | |
| iic_master_timeout_mode_t | timeout_mode | Timeout Detection Time Select: |

| | | |
|---|---|---|
| | | Long Mode = 0 and Short Mode = 1. |
| iic_master_clock_settings_t | clock_settings | I2C Clock settings. |

## Enumeration Type Documentation

### ◆ iic_master_timeout_mode_t

| enum iic_master_timeout_mode_t | |
|---|---|
| I2C Timeout mode parameter definition | |
| Enumerator | |
| IIC_MASTER_TIMEOUT_MODE_LONG | Timeout Detection Time Select: Long Mode -> TMOS = 0. |
| IIC_MASTER_TIMEOUT_MODE_SHORT | Timeout Detection Time Select: Short Mode -> TMOS = 1. |

## Function Documentation

### ◆ R_IIC_MASTER_Open()

fsp_err_t R_IIC_MASTER_Open ( i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_cfg_t const *const *p_cfg* )

Opens the I2C device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Requested clock rate was set exactly. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_ASSERTION | Parameter check failure due to one or more reasons below:<br>1. p_api_ctrl or p_cfg is NULL.<br>2. extended parameter is NULL.<br>3. Callback parameter is NULL.<br>4. Set the rate to fast mode plus on a channel which does not support it.<br>5. Invalid IRQ number assigned |

### ◆ R_IIC_MASTER_Read()

fsp_err_t R_IIC_MASTER_Read ( i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, bool const *restart* )

Performs a read from the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_RX_COMPLETE in the callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | p_api_ctrl, p_dest or bytes is NULL. |
| FSP_ERR_INVALID_SIZE | Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer. |
| FSP_ERR_IN_USE | Bus busy condition. Another transfer was in progress. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IIC_MASTER_Open to initialize the control block. |

### ◆ R_IIC_MASTER_Write()

fsp_err_t R_IIC_MASTER_Write ( i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*, bool const *restart* )

Performs a write to the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_TX_COMPLETE in the callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | p_api_ctrl or p_src is NULL. |
| FSP_ERR_INVALID_SIZE | Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer. |
| FSP_ERR_IN_USE | Bus busy condition. Another transfer was in progress. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IIC_MASTER_Open to initialize the control block. |

### ◆ R_IIC_MASTER_Abort()

fsp_err_t R_IIC_MASTER_Abort ( i2c_master_ctrl_t *const *p_api_ctrl*)

Safely aborts any in-progress transfer and forces the IIC peripheral into ready state.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel was reset successfully. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IIC_MASTER_Open to initialize the control block. |

***Note***

*A callback will not be invoked in case an in-progress transfer gets aborted by calling this API.*

### ◆ R_IIC_MASTER_SlaveAddressSet()

fsp_err_t R_IIC_MASTER_SlaveAddressSet ( i2c_master_ctrl_t *const *p_api_ctrl*, uint32_t const *slave*, i2c_master_addr_mode_t const *addr_mode* )

Sets address and addressing mode of the slave device. This function is used to set the device address and addressing mode of the slave without reconfiguring the entire bus.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Address of the slave is set correctly. |
| FSP_ERR_ASSERTION | Pointer to control structure is NULL. |
| FSP_ERR_IN_USE | Another transfer was in-progress. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IIC_MASTER_Open to initialize the control block. |

### ◆ R_IIC_MASTER_Close()

fsp_err_t R_IIC_MASTER_Close ( i2c_master_ctrl_t *const *p_api_ctrl*)

Closes the I2C device. May power down IIC peripheral. This function will safely terminate any in-progress I2C transfers.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device closed without issue. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IIC_MASTER_Open to initialize the control block. |

***Note***

*A callback will not be invoked in case an in-progress transfer gets aborted by calling this API.*

### ◆ R_IIC_MASTER_VersionGet()

fsp_err_t R_IIC_MASTER_VersionGet ( fsp_version_t *const *p_version*)

Gets version information and stores it in the provided version structure.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful version get. |
| FSP_ERR_ASSERTION | p_version is NULL. |

## 4.2.27 I2C Slave on IIC (r_iic_slave)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_IIC_SLAVE_Open (i2c_slave_ctrl_t *const p_api_ctrl, i2c_slave_cfg_t const *const p_cfg) |
| fsp_err_t | R_IIC_SLAVE_Read (i2c_slave_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_IIC_SLAVE_Write (i2c_slave_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes) |
| fsp_err_t | R_IIC_SLAVE_Close (i2c_slave_ctrl_t *const p_api_ctrl) |

fsp_err_t R_IIC_SLAVE_VersionGet (fsp_version_t *const p_version)

### Detailed Description

Driver for the IIC peripheral on RA MCUs. This module implements the I2C Slave Interface.

# Overview

### Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100-kHz transaction rate.
  - Fast Mode Support with up to 400-kHz transaction rate.
  - Fast Mode Plus Support with up to 1-MHz transaction rate.
- Reads data written by master device.
- Write data which is read by master device.
- Can accept 0x00 as slave address.
- Can be assigned a 10-bit address.
- Clock stretching is supported and can be implemented via callbacks.
- Provides Transmission/Reception transaction size in the callback.
- I2C Slave can notify the following events via callbacks: Transmission/Reception Request, Transmission/Reception Request for more data, Transmission/Reception Completion, Error Condition.

# Configuration

### Build Time Configurations for r_iic_slave

The following build time configurations are defined in fsp_cfg/r_iic_slave_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Connectivity > I2C Slave Driver on r_iic_slave

This module can be added to the Stacks tab via New Stack > Driver > Connectivity > I2C Slave Driver on r_iic_slave:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_i2c_slave0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Specify the IIC channel. |
| Rate | • Standard | Standard | Select the transfer |

| | | | |
|---|---|---|---|
| | • Fast-mode<br>• Fast-mode plus | | rate. |
| Internal Reference Clock | • PCLKB / 1<br>• PCLKB / 2<br>• PCLKB / 4<br>• PCLKB / 8<br>• PCLKB / 16<br>• PCLKB / 32<br>• PCLKB / 64<br>• PCLKB / 128 | PCLKB / 1 | Select the internal reference clock for IIC slave. The internal reference clock is used only to determine the clock frequency of the noise filter samples. |
| Digital Filter | • Disabled<br>• 1 Reference Clock Cycle<br>• 2 Reference Clock Cycles<br>• 3 Reference Clock Cycles<br>• 4 Reference Clock Cycles | 3 Reference Clock Cycles | Select the number of digital filter stages for IIC Slave. |
| Slave Address | Value must be non-negative | 0x00 | Specify the slave address. |
| General Call | • Enabled<br>• Disabled | Disabled | Allows the slave to respond to general call address: 0x00. |
| Address Mode | • 7-Bit<br>• 10-Bit | 7-Bit | Select the slave address mode. |
| Callback | Name must be a valid C symbol | i2c_slave_callback | A user callback function must be provided. This will be called from the interrupt service routine (ISR) to report I2C Slave transaction events and status. |
| Interrupt Priority Level | MCU Specific Options | | Select the interrupt priority level. This is set for TXI, RXI, TEI and ERI interrupts. |

### Clock Configuration

The IIC peripheral module uses the PCLKB as its clock source. The actual I2C transfer rate will be calculated and set by the tooling depending on the selected transfer rate. If the PCLKB is configured in such a manner that the selected transfer rate cannot be achieved, an error will be returned.

### 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. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

# Usage Notes

### Interrupt Configuration

- The IIC error (EEI), receive buffer full (RXI), transmit buffer empty (TXI) and transmit end (TEI) interrupts for the selected channel must be enabled in the properties of the selected device.

### Callback

- A callback function must be provided which will be invoked for the cases below:
    - An I2C Master initiates a transmission or reception: I2C_SLAVE_EVENT_TX_REQUEST; I2C_SLAVE_EVENT_RX_REQUEST
    - A Transmission or reception has been completed: I2C_SLAVE_EVENT_TX_COMPLETE; I2C_SLAVE_EVENT_RX_COMPLETE
    - An I2C Master is requesting to read or write more data: I2C_SLAVE_EVENT_TX_MORE_REQUEST; I2C_SLAVE_EVENT_RX_MORE_REQUEST
    - Error conditions: I2C_SLAVE_EVENT_ABORTED
    - An I2C Master initiates a general call by passing 0x00 as slave address: I2C_SLAVE_EVENT_GENERAL_CALL
- The callback arguments will contain information about the transaction status/events, bytes transferred and a pointer to the user defined context.
- Clock stretching is enabled by the use of callbacks. This means that the IIC slave can hold the clock line SCL LOW to force the I2C Master into a wait state.
- The table below shows I2C Slave event handling expected in user code:

| IIC Slave Callback Event | IIC Slave API expected to be called |
|---|---|
| I2C_SLAVE_EVENT_ABORTED | Handle event based on application |
| I2C_SLAVE_EVENT_RX_COMPLETE | Handle event based on application |
| I2C_SLAVE_EVENT_TX_COMPLETE | Handle event based on application |
| I2C_SLAVE_EVENT_RX_REQUEST | R_IIC_SLAVE_Read API. If the slave is a Write Only device call this API with 0 bytes to send a NACK to the master. |
| I2C_SLAVE_EVENT_TX_REQUEST | R_IIC_SLAVE_Write API |
| I2C_SLAVE_EVENT_RX_MORE_REQUEST | R_IIC_SLAVE_Read API. If the slave cannot read any more data call this API with 0 bytes to send a NACK to the master. |
| I2C_SLAVE_EVENT_TX_MORE_REQUEST | R_IIC_SLAVE_Write API |
| I2C_SLAVE_EVENT_GENERAL_CALL | R_IIC_SLAVE_Read |

- If parameter checking is enabled and R_IIC_SLAVE_Read API is not called for I2C_SLAVE_EVENT_RX_REQUEST and/or I2C_SLAVE_EVENT_RX_MORE_REQUEST, the slave will send a NACK to the master and would eventually timeout.
- R_IIC_SLAVE_Write API is not called for I2C_SLAVE_EVENT_TX_REQUEST and/or I2C_SLAVE_EVENT_TX_MORE_REQUEST:
    - Slave timeout is less than Master timeout: The slave will timeout and release the bus causing the master to read 0xFF for every remaining byte.

- Slave timeout is more than Master timeout: The master will timeout first followed by the slave.

### IIC Slave Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate. 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.

## Examples

### Basic Example

This is a basic example of minimal use of the R_IIC_SLAVE in an application. This example shows how this driver can be used for basic read and write operations.

```
iic_master_instance_ctrl_t g_i2c_master_ctrl;
i2c_master_cfg_t g_i2c_master_cfg =
{
    .channel       = I2C_MASTER_CHANNEL_2,
    .rate          = I2C_MASTER_RATE_STANDARD,
    .slave         = I2C_7BIT_ADDR_IIC_SLAVE,
    .addr_mode    = I2C_MASTER_ADDR_MODE_7BIT,
    .p_callback    = i2c_master_callback, // Callback
    .p_context    = &g_i2c_master_ctrl,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_master_cfg_extend_standard_mode
};
iic_slave_instance_ctrl_t g_i2c_slave_ctrl;
i2c_slave_cfg_t g_i2c_slave_cfg =
{
    .channel    = I2C_SLAVE_CHANNEL_0,
    .rate       = I2C_SLAVE_RATE_STANDARD,
    .slave      = I2C_7BIT_ADDR_IIC_SLAVE,
    .addr_mode = I2C_SLAVE_ADDR_MODE_7BIT,
    .p_callback = i2c_slave_callback, // Callback
    .p_context = &g_i2c_slave_ctrl,
    .p_extend   = &g_iic_slave_cfg_extend_standard_mode
};
```

```
void i2c_master_callback (i2c_master_callback_args_t * p_args)
{
    g_i2c_master_callback_event = p_args->event;
}
void i2c_slave_callback (i2c_slave_callback_args_t * p_args)
{
    g_i2c_slave_callback_event = p_args->event;
 if ((p_args->event == I2C_SLAVE_EVENT_RX_COMPLETE) || (p_args->event ==
I2C_SLAVE_EVENT_TX_COMPLETE))
    {
 /* Transaction Successful */
    }
 else if ((p_args->event == I2C_SLAVE_EVENT_RX_REQUEST) || (p_args->event ==
I2C_SLAVE_EVENT_RX_MORE_REQUEST))
    {
 /* Read from Master */
       err = R_IIC_SLAVE_Read(&g_i2c_slave_ctrl, g_i2c_slave_buffer,
g_slave_transfer_length);
       handle_error(err);
    }
 else if ((p_args->event == I2C_SLAVE_EVENT_TX_REQUEST) || (p_args->event ==
I2C_SLAVE_EVENT_TX_MORE_REQUEST))
    {
 /* Write to master */
       err = R_IIC_SLAVE_Write(&g_i2c_slave_ctrl, g_i2c_slave_buffer,
g_slave_transfer_length);
       handle_error(err);
    }
 else
    {
 /* Error Event - reported through g_i2c_slave_callback_event */
    }
}
void basic_example (void)
```

```
{
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    g_slave_transfer_length = I2C_BUFFER_SIZE_BYTES;
 /* Pin connections:
  * Channel 0 SDA <--> Channel 2 SDA
  * Channel 0 SCL <--> Channel 2 SCL
  */
 /* Initialize the IIC Slave module */
    err = R_IIC_SLAVE_Open(&g_i2c_slave_ctrl, &g_i2c_slave_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Initialize the IIC Master module */
    err = R_IIC_MASTER_Open(&g_i2c_master_ctrl, &g_i2c_master_cfg);
    handle_error(err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_master_tx_buffer[i] = (uint8_t) i;
    }
 /* Send data to I2C slave */
    g_i2c_master_callback_event = I2C_MASTER_EVENT_ABORTED;
    g_i2c_slave_callback_event = I2C_SLAVE_EVENT_ABORTED;
    err = R_IIC_MASTER_Write(&g_i2c_master_ctrl, &g_i2c_master_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    handle_error(err);
 /* Since there is nothing else to do, block until Callback triggers
  * The Slave Callback will call the R_IIC_SLAVE_Read API to service the Master Write
Request.
  */
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_master_callback_event ||
 I2C_SLAVE_EVENT_RX_COMPLETE != g_i2c_slave_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
```

```
        timeout_ms--;
    }
 if ((I2C_MASTER_EVENT_ABORTED == g_i2c_master_callback_event) ||
        (I2C_SLAVE_EVENT_ABORTED == g_i2c_slave_callback_event))
    {
        __BKPT(0);
    }
 /* Read data back from the I2C slave */
    g_i2c_master_callback_event = I2C_MASTER_EVENT_ABORTED;
    g_i2c_slave_callback_event = I2C_SLAVE_EVENT_ABORTED;
    timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_MASTER_Read(&g_i2c_master_ctrl, &g_i2c_master_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    handle_error(err);
 /* Since there is nothing else to do, block until Callback triggers
  * The Slave Callback will call the R_IIC_SLAVE_Write API to service the Master Read
Request.
  */
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_master_callback_event ||
 I2C_SLAVE_EVENT_TX_COMPLETE != g_i2c_slave_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
        timeout_ms--;
    }
 if ((I2C_MASTER_EVENT_ABORTED == g_i2c_master_callback_event) ||
        (I2C_SLAVE_EVENT_ABORTED == g_i2c_slave_callback_event))
    {
        __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_master_tx_buffer, g_i2c_master_rx_buffer,
I2C_BUFFER_SIZE_BYTES))
    {
        __BKPT(0);
```

```
    }
}
```

## Data Structures

| | |
|---|---|
| struct | iic_slave_clock_settings_t |
| struct | iic_slave_extended_cfg_t |

## Data Structure Documentation

### ◆ iic_slave_clock_settings_t

| struct iic_slave_clock_settings_t | | |
|---|---|---|
| I2C clock settings | | |
| Data Fields | | |
| uint8_t | cks_value | Internal Reference Clock Select. |
| uint8_t | brl_value | Low-level period of SCL clock. |
| uint8_t | digital_filter_stages | Number of digital filter stages based on brl_value. |

### ◆ iic_slave_extended_cfg_t

| struct iic_slave_extended_cfg_t | | |
|---|---|---|
| R_IIC_SLAVE extended configuration | | |
| Data Fields | | |
| iic_slave_clock_settings_t | clock_settings | I2C Clock settings. |

## Function Documentation

### ◆ R_IIC_SLAVE_Open()

fsp_err_t R_IIC_SLAVE_Open ( i2c_slave_ctrl_t *const *p_api_ctrl*, i2c_slave_cfg_t const *const *p_cfg* )

Opens the I2C slave device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | I2C slave device opened successfully. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_ASSERTION | Parameter check failure due to one or more reasons below:<br>1. p_api_ctrl or p_cfg is NULL.<br>2. extended parameter is NULL.<br>3. Callback parameter is NULL.<br>4. Set the rate to fast mode plus on a channel which does not support it.<br>5. Invalid IRQ number assigned |

### ◆ R_IIC_SLAVE_Read()

fsp_err_t R_IIC_SLAVE_Read ( i2c_slave_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Performs a read from the I2C Master device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C slave read operation will begin. The caller will be notified when the operation has finished by an I2C_SLAVE_EVENT_RX_COMPLETE in the callback. In case the master continues to write more data, an I2C_SLAVE_EVENT_RX_MORE_REQUEST will be issued via callback. In case of errors, an I2C_SLAVE_EVENT_ABORTED will be issued via callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue |
| FSP_ERR_ASSERTION | p_api_ctrl, bytes or p_dest is NULL. |
| FSP_ERR_IN_USE | Another transfer was in progress. |
| FSP_ERR_NOT_OPEN | Device is not open. |

### ◆ R_IIC_SLAVE_Write()

fsp_err_t R_IIC_SLAVE_Write ( i2c_slave_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes* )

Performs a write to the I2C Master device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C slave write operation will begin. The caller will be notified when the operation has finished by an I2C_SLAVE_EVENT_TX_COMPLETE in the callback. In case the master continues to read more data, an I2C_SLAVE_EVENT_TX_MORE_REQUEST will be issued via callback. In case of errors, an I2C_SLAVE_EVENT_ABORTED will be issued via callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | p_api_ctrl or p_src is NULL. |
| FSP_ERR_IN_USE | Another transfer was in progress. |
| FSP_ERR_NOT_OPEN | Device is not open. |

### ◆ R_IIC_SLAVE_Close()

fsp_err_t R_IIC_SLAVE_Close ( i2c_slave_ctrl_t *const *p_api_ctrl*)

Closes the I2C device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device closed successfully. |
| FSP_ERR_NOT_OPEN | Device not opened. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |

### ◆ R_IIC_SLAVE_VersionGet()

fsp_err_t R_IIC_SLAVE_VersionGet ( fsp_version_t *const *p_version*)

Gets version information and stores it in the provided version structure.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful version get. |
| FSP_ERR_ASSERTION | p_version is NULL. |

## 4.2.28 I/O Ports (r_ioport)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | R_IOPORT_Open (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg) |
| fsp_err_t | R_IOPORT_Close (ioport_ctrl_t *const p_ctrl) |
| fsp_err_t | R_IOPORT_PinsCfg (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg) |
| fsp_err_t | R_IOPORT_PinCfg (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, uint32_t cfg) |
| fsp_err_t | R_IOPORT_PinEventInputRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_event) |
| fsp_err_t | R_IOPORT_PinEventOutputWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t pin_value) |
| fsp_err_t | R_IOPORT_PinRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_value) |
| fsp_err_t | R_IOPORT_PinWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t level) |
| fsp_err_t | R_IOPORT_PortDirectionSet (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t direction_values, ioport_size_t mask) |
| fsp_err_t | R_IOPORT_PortEventInputRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *event_data) |
| fsp_err_t | R_IOPORT_PortEventOutputWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t event_data, ioport_size_t mask_value) |
| fsp_err_t | R_IOPORT_PortRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_port_value) |
| fsp_err_t | R_IOPORT_PortWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t value, ioport_size_t mask) |
| fsp_err_t | R_IOPORT_EthernetModeCfg (ioport_ctrl_t *const p_ctrl, ioport_ethernet_channel_t channel, ioport_ethernet_mode_t mode) |
| fsp_err_t | R_IOPORT_VersionGet (fsp_version_t *p_data) |

### Detailed Description

Driver for the I/O Ports peripheral on RA MCUs. This module implements the I/O Port Interface.

# Overview

The I/O port pins operate as general I/O port pins, I/O pins for peripheral modules, interrupt input pins, analog I/O, port group function for the ELC, or bus control pins.

### Features

The IOPORT HAL module can configure the following pin settings:

- Pin direction
- Default output state
- 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
- Peripheral connection

The module also provides the following functionality:

- Read/write GPIO pins/ports
- Sets event output data
- Reads event input data

# Configuration

The I/O PORT HAL module must be configured by the user for the desired operation. The operating state of an I/O pin can be set via the RA Configuraton tool. When the project is built a pin configuration file is created. The BSP will automatically configure the MCU IO ports accordingly at startup using the same API functions mentioned in this document.

### Build Time Configurations for r_ioport

The following build time configurations are defined in fsp_cfg/r_ioport_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > System > I/O Port Driver on r_ioport

This module can be added to the Stacks tab via New Stack > Driver > System > I/O Port Driver on r_ioport:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_ioport | Module name. |
| Port 1 ELC Trigger Source | MCU Specific Options | | ELC source that will trigger PORT1 |
| Port 2 ELC Trigger Source | MCU Specific Options | | ELC source that will trigger PORT2 |
| Port 3 ELC Trigger Source | MCU Specific Options | | ELC source that will trigger PORT3 |
| Port 4 ELC Trigger Source | MCU Specific Options | | ELC source that will trigger PORT4 |

### Clock Configuration

The I/O PORT HAL module does not require a specific clock configuration.

### Pin Configuration

The IOPORT module is used for configuring pins.

## Usage Notes

### Port Group Function for ELC

Depending on pin configuration, the IOPORT module can perform automatic reads and writes on ports 1-4 on receipt of an ELC event.

When an event is received by a port, the state of the input pins on the port is saved in a hardware register. Simultaneously, the state of output pins on the port is set or cleared based on settings configured by the user. The functions R_IOPORT_PinEventInputRead and R_IOPORT_PortEventInputRead allow reading the last event input state of a pin or port, and event-triggered pin output can be configured through R_IOPORT_PinEventOutputWrite and R_IOPORT_PortEventOutputWrite.

In addition, each pin on ports 1-4 can be configured to trigger an ELC event on rising, falling or both edges. This event can be used to activate other modules when the pin changes state.

*Note*

*The number of ELC-aware ports vary across MCUs. Refer to the Hardware User's Manual for your device for more details.*

## Examples

### Basic Example

This is a basic example of minimal use of the IOPORT in an application.

```
void basic_example ()
```

```
{
 bsp_io_level_t readLevel;
 fsp_err_t      err;
 /* Initialize the IOPORT module and configure the pins
  * Note: The default pin configuration name in the RA Configuraton tool is
g_bsp_pin_cfg */
    err = R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Call R_IOPORT_PinsCfg if the configuration was not part of initial configurations
made in open */
    err = R_IOPORT_PinsCfg(&g_ioport_ctrl, &g_runtime_pin_cfg);
    handle_error(err);
 /* Set Pin 00 of Port 06 to High */
    err = R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_06_PIN_00, BSP_IO_LEVEL_HIGH
);
    handle_error(err);
 /* Read Pin 00 of Port 06*/
    err = R_IOPORT_PinRead(&g_ioport_ctrl, BSP_IO_PORT_06_PIN_00, &readLevel);
    handle_error(err);
}
```

### Blinky Example

This example uses IOPORT to configure and toggle a pin to blink an LED.

```
void blinky_example ()
{
 fsp_err_t err;
 /* Initialize the IOPORT module and configure the pins */
    err = R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Configure Pin as output
  * Call the R_IOPORT_PinCfg if the configuration was not part of initial
```

```
configurations made in open */
    err = R_IOPORT_PinCfg(&g_ioport_ctrl, BSP_IO_PORT_06_PIN_00,
BSP_IO_DIRECTION_OUTPUT);
    handle_error(err);
 bsp_io_level_t level = BSP_IO_LEVEL_LOW;
 while (1)
    {
 /* Determine the next state of the LEDs */
 if (BSP_IO_LEVEL_LOW == level)
      {
            level = BSP_IO_LEVEL_HIGH;
       }
 else
      {
            level = BSP_IO_LEVEL_LOW;
       }
 /* Update LED on RA6M3-PK */
       err = R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_06_PIN_00, level);
      handle_error(err);
 /* Delay */
 R_BSP_SoftwareDelay(100, BSP_DELAY_UNITS_MILLISECONDS); // NOLINT
    }
}
```

### ELC Example

This is an example of using IOPORT with ELC events. The ELC event system allows the captured data to be stored when it occurs and then read back at a later time.

```
static elc_instance_ctrl_t g_elc_ctrl;
static elc_cfg_t g_elc_cfg;
void ioport_elc_example ()
{
 bsp_io_level_t eventValue;
 fsp_err_t      err;
```

```
 /* Initializes the software and sets the links defined in the control structure. */
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Create or modify a link between a peripheral function and an event source. */
    err = R_ELC_LinkSet(&g_elc_ctrl, (elc_peripheral_t) ELC_PERIPHERAL_IOPORT2,
ELC_EVENT_ELC_SOFTWARE_EVENT_0);
    handle_error(err);
 /* Globally enable event linking in the ELC. */
    err = R_ELC_Enable(&g_elc_ctrl);
    handle_error(err);
 /* Initialize the IOPORT module and configure the pins */
    err = R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
    handle_error(err);
 /* Call the R_IOPORT_PinCfg if the configuration was not part of initial
configurations made in open */
    err = R_IOPORT_PinCfg(&g_ioport_ctrl, BSP_IO_PORT_02_PIN_00,
BSP_IO_DIRECTION_INPUT);
    handle_error(err);
 /* Generate an event signal through software to the linked peripheral. */
    err = R_ELC_SoftwareEventGenerate(&g_elc_ctrl, ELC_SOFTWARE_EVENT_0);
    handle_error(err);
 /* Read Pin Event Input. The data(BSP_IO_LEVEL_HIGH/ BSP_IO_LEVEL_LOW) from
BSP_IO_PORT_02_PIN_00 is read into the
  * EIDR bit */
    err = R_IOPORT_PinEventInputRead(&g_ioport_ctrl, BSP_IO_PORT_02_PIN_00,
&eventValue);
    handle_error(err);
}
```

## Data Structures

| | |
|---|---|
| struct | ioport_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | ioport_port_pin_t |

## Data Structure Documentation

### ◆ ioport_instance_ctrl_t

| struct ioport_instance_ctrl_t |
|---|
| IOPORT private control block. DO NOT MODIFY. Initialization occurs when R_IOPORT_Open() is called. |

## Enumeration Type Documentation

### ◆ ioport_port_pin_t

| enum ioport_port_pin_t | |
|---|---|
| Superset list of all possible IO port pins. | |
| Enumerator | |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

## Function Documentation

### ◆ R_IOPORT_Open()

fsp_err_t R_IOPORT_Open ( ioport_ctrl_t *const *p_ctrl*, const ioport_cfg_t * *p_cfg* )

Initializes internal driver data, then calls pin configuration function to configure pins.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Pin configuration data written to PFS register(s) |
| FSP_ERR_ASSERTION | NULL pointer |
| FSP_ERR_ALREADY_OPEN | Module is already open. |

### ◆ R_IOPORT_Close()

fsp_err_t R_IOPORT_Close ( ioport_ctrl_t *const *p_ctrl*)

Resets IOPORT registers. Implements ioport_api_t::close

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The IOPORT was successfully uninitialized |
| FSP_ERR_ASSERTION | p_ctrl was NULL |
| FSP_ERR_NOT_OPEN | The module has not been opened |

### ◆ R_IOPORT_PinsCfg()

fsp_err_t R_IOPORT_PinsCfg ( ioport_ctrl_t *const *p_ctrl*, const ioport_cfg_t * *p_cfg* )

Configures the functions of multiple pins by loading configuration data into pin PFS registers. Implements ioport_api_t::pinsCfg.

This function initializes the supplied list of PmnPFS registers with the supplied values. This data can be generated by the Pins tab of the RA Configuration editor or manually by the developer. Different pin configurations can be loaded for different situations such as low power modes and testing.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Pin configuration data written to PFS register(s) |
| FSP_ERR_NOT_OPEN | The module has not been opened |
| FSP_ERR_ASSERTION | NULL pointer |

### ◆ R_IOPORT_PinCfg()

fsp_err_t R_IOPORT_PinCfg ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, uint32_t *cfg* )

Configures the settings of a pin. Implements ioport_api_t::pinCfg.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Pin configured |
| FSP_ERR_NOT_OPEN | The module has not been opened |
| FSP_ERR_ASSERTION | NULL pointer |

***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.*

### ◆ R_IOPORT_PinEventInputRead()

fsp_err_t R_IOPORT_PinEventInputRead ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, bsp_io_level_t * *p_pin_event* )

Reads the value of the event input data of a specific pin. Implements ioport_api_t::pinEventInputRead.

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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Pin read |
| FSP_ERR_ASSERTION | NULL pointer |
| FSP_ERR_NOT_OPEN | The module has not been opened |
| FSP_ERR_INVALID_ARGUMENT | Port is not valid ELC PORT. |

***Note***

*This function is re-entrant.*

### ◆ R_IOPORT_PinEventOutputWrite()

fsp_err_t R_IOPORT_PinEventOutputWrite ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, bsp_io_level_t *pin_value* )

This function writes the event output data value to a pin. Implements ioport_api_t::pinEventOutputWrite.

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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Pin event data written |
| FSP_ERR_INVALID_ARGUMENT | Port or Pin or value not valid |
| FSP_ERR_NOT_OPEN | The module has not been opened |
| FSP_ERR_ASSERTION | NULL pointer |

***Note***

*This function is re-entrant for different ports.*

### ◆ R_IOPORT_PinRead()

fsp_err_t R_IOPORT_PinRead ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, bsp_io_level_t * *p_pin_value* )

Reads the level on a pin. Implements ioport_api_t::pinRead.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Pin read |
| FSP_ERR_ASSERTION | NULL pointer |
| FSP_ERR_NOT_OPEN | The module has not been opened |

***Note***

*This function is re-entrant for different pins.*

### ◆ R_IOPORT_PinWrite()

fsp_err_t R_IOPORT_PinWrite ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, bsp_io_level_t *level* )

Sets a pin's output either high or low. Implements ioport_api_t::pinWrite.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Pin written to |
| FSP_ERR_INVALID_ARGUMENT | The pin and/or level not valid |
| FSP_ERR_NOT_OPEN | The module has not been opene |
| FSP_ERR_ASSERTION | NULL pointerd |

***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.*

### ◆ R_IOPORT_PortDirectionSet()

fsp_err_t R_IOPORT_PortDirectionSet ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t *direction_values*, ioport_size_t *mask* )

Sets the direction of individual pins on a port. Implements ioport_api_t::portDirectionSet().

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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Port direction updated |
| FSP_ERR_INVALID_ARGUMENT | The port and/or mask not valid |
| FSP_ERR_NOT_OPEN | The module has not been opened |
| FSP_ERR_ASSERTION | NULL pointer |

***Note***

*This function is re-entrant for different ports.*

### ◆ R_IOPORT_PortEventInputRead()

fsp_err_t R_IOPORT_PortEventInputRead ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t * *p_event_data* )

Reads the value of the event input data. Implements ioport_api_t::portEventInputRead().

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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Port read |
| FSP_ERR_INVALID_ARGUMENT | Port not a valid ELC port |
| FSP_ERR_ASSERTION | NULL pointer |
| FSP_ERR_NOT_OPEN | The module has not been opened |

***Note***

*This function is re-entrant for different ports.*

### ◆ R_IOPORT_PortEventOutputWrite()

fsp_err_t R_IOPORT_PortEventOutputWrite ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t *event_data*, ioport_size_t *mask_value* )

This function writes the set and reset event output data for a port. Implements ioport_api_t::portEventOutputWrite.

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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Port event data written |
| FSP_ERR_INVALID_ARGUMENT | Port or Mask not valid |
| FSP_ERR_NOT_OPEN | The module has not been opened |
| FSP_ERR_ASSERTION | NULL pointer |

***Note***

*This function is re-entrant for different ports.*

### ◆ R_IOPORT_PortRead()

fsp_err_t R_IOPORT_PortRead ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t * *p_port_value* )

Reads the value on an IO port. Implements ioport_api_t::portRead.

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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Port read |
| FSP_ERR_ASSERTION | NULL pointer |
| FSP_ERR_NOT_OPEN | The module has not been opened |

***Note***

*This function is re-entrant for different ports.*

### ◆ R_IOPORT_PortWrite()

fsp_err_t R_IOPORT_PortWrite ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t *value*, ioport_size_t *mask* )

Writes to multiple pins on a port. Implements ioport_api_t::portWrite.

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. For example, value = 0xFFFF, mask = 0x0003 results in only bits 0 and 1 being updated.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Port written to |
| FSP_ERR_INVALID_ARGUMENT | The port and/or mask not valid |
| FSP_ERR_NOT_OPEN | The module has not been opened |
| FSP_ERR_ASSERTION | NULL pointerd |

***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.*

### ◆ R_IOPORT_EthernetModeCfg()

fsp_err_t R_IOPORT_EthernetModeCfg ( ioport_ctrl_t *const *p_ctrl*, ioport_ethernet_channel_t *channel*, ioport_ethernet_mode_t *mode* )

Configures Ethernet channel PHY mode. Implements ioport_api_t::pinEthernetModeCfg.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Ethernet PHY mode set |
| FSP_ERR_INVALID_ARGUMENT | Channel or mode not valid |
| FSP_ERR_UNSUPPORTED | Ethernet configuration not supported on this device. |
| FSP_ERR_NOT_OPEN | The module has not been opened |
| FSP_ERR_ASSERTION | NULL pointer |

***Note***

*This function is not re-entrant.*

### ◆ R_IOPORT_VersionGet()

| fsp_err_t R_IOPORT_VersionGet ( fsp_version_t * *p_data*) |
|---|
| Returns IOPort HAL driver version. Implements ioport_api_t::versionGet.<br><br>**Return values**<br>FSP_SUCCESS — Version information read<br>FSP_ERR_ASSERTION — The parameter p_data is NULL<br><br>***Note***<br>*This function is reentrant.* |

## 4.2.29 Independent Watchdog Timer (r_iwdt)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | R_IWDT_Refresh (wdt_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_IWDT_Open (wdt_ctrl_t *const p_api_ctrl, wdt_cfg_t const *const p_cfg) |
| fsp_err_t | R_IWDT_StatusClear (wdt_ctrl_t *const p_api_ctrl, const wdt_status_t status) |
| fsp_err_t | R_IWDT_StatusGet (wdt_ctrl_t *const p_api_ctrl, wdt_status_t *const p_status) |
| fsp_err_t | R_IWDT_CounterGet (wdt_ctrl_t *const p_api_ctrl, uint32_t *const p_count) |
| fsp_err_t | R_IWDT_TimeoutGet (wdt_ctrl_t *const p_api_ctrl, wdt_timeout_values_t *const p_timeout) |
| fsp_err_t | R_IWDT_VersionGet (fsp_version_t *const p_data) |

### Detailed Description

Driver for the IWDT peripheral on RA MCUs. This module implements the WDT Interface.

# Overview

The independent watchdog timer is used to recover from unexpected errors in an application. The timer must be refreshed periodically in the permitted count window by the application. 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.

### Features

The IWDT HAL module has the following key features:

- 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 IWDT begins counting at reset.

### Selecting a Watchdog

RA MCUs have two watchdog peripherals: the watchdog timer (WDT) and the independent watchdog timer (IWDT). When selecting between them, consider these factors:

| | WDT | IWDT |
|---|---|---|
| Start Mode | The WDT can be started from the application (register start mode) or configured by hardware to start automatically (auto start mode). | The IWDT can only be configured by hardware to start automatically. |
| Clock Source | The WDT runs off a peripheral clock. | The IWDT has its own clock source which improves safety. |

# Configuration

### Build Time Configurations for r_iwdt

The following build time configurations are defined in fsp_cfg/r_iwdt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Monitoring > Watchdog Driver on r_iwdt

This module can be added to the Stacks tab via New Stack > Driver > Monitoring > Watchdog Driver on r_iwdt:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid | g_wdt0 | Module name. |

| | | | |
|---|---|---|---|
| | C symbol | | |
| NMI callback | Name must be a valid C symbol | NULL | A user callback function can be provided here. If this callback function is provided, it is called from the interrupt service routine (ISR) when the watchdog triggers. |

*Note*

*The IWDT has additional configurable settings in the OFS0 register in the **BSP** tab properties window. These settings include the following:*

- *Start Mode*
- *Timeout Period*
- *Dedicated Clock Frequency Divisor*
- *Window End Position*
- *Window Start Position*
- *Reset Interrupt Request Select*
- *Stop Control*

*Review the OFS0 properties window to see additional details.*

### Clock Configuration

The IWDT clock is based on the IWDTCLK frequency. You can set the IWDTCLK frequency divider using the **BSP** tab of the RA Configuration editor.

### Pin Configuration

This module does not use I/O pins.

## Usage Notes

### NMI Interrupt

The independent watchdog timer uses the NMI, which is enabled by default. No special configuration is required. When the NMI is triggered, the callback function registered during open is called.

### Period Calculation

The IWDT operates from IWDTCLK. With a IWDTCLK of 15000 Hz, the maximum time from the last refresh to device reset or NMI generation will be just below 35 seconds as detailed below.

IWDTCLK = 15000 Hz
Clock division ratio = IWDTCLK / 256
Timeout period = 2048 cycles
WDT clock frequency = 15000 Hz / 256 = 58.59 Hz
Cycle time = 1 / 58.59 Hz = 17.067 ms
Timeout = 17.067 ms x 2048 cycles = 34.95 seconds

### Limitations

Developers should be aware of the following limitations when using the IWDT:

- When using a J-Link debugger the IWDT counter does not count and therefore will not reset the device or generate an NMI. To enable the watchdog to count and generate a reset or NMI while debugging, add this line of code in the application:

```
/* (Optional) Enable the IWDT to count and generate NMI or reset when the
 * debugger is connected. */
   R_DEBUG->DBGSTOPCR_b.DBGSTOP_IWDT = 0;
```

- If the IWDT is configured to stop the counter in low power mode, then your application must restart the watchdog by calling R_IWDT_Refresh() after the MCU wakes from low power mode.

# Examples

### IWDT Basic Example

This is a basic example of minimal use of the IWDT in an application.

```
void iwdt_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* In auto start mode, the IWDT starts counting immediately when the MCU is powered
on. */
 /* Initializes the module. */
    err = R_IWDT_Open(&g_iwdt0_ctrl, &g_iwdt0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (true)
    {
 /* Application work here. */
 /* Refresh before the counter underflows to prevent reset or NMI based on the
setting. */
       (void) R_IWDT_Refresh(&g_iwdt0_ctrl);
    }
}
```

### IWDT Advanced Example

This example demonstrates using a start window and gives an example callback to handle an NMI generated by an underflow or refresh error.

```
#define IWDT_TIMEOUT_COUNTS (2048U)
#define IWDT_MAX_COUNTER (IWDT_TIMEOUT_COUNTS - 1U)
#define IWDT_START_WINDOW_75 ((IWDT_MAX_COUNTER * 3) / 4)
/* Example callback called when a watchdog NMI occurs. */
void iwdt_callback (wdt_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) Determine the source of the NMI. */
 wdt_status_t status = WDT_STATUS_NO_ERROR;
    err = R_IWDT_StatusGet(&g_iwdt0_ctrl, &status);
    handle_error(err);
 /* (Optional) Log source of NMI and any other debug information. */
 /* (Optional) Clear the error flags. */
    err = R_IWDT_StatusClear(&g_iwdt0_ctrl, status);
    handle_error(err);
 /* (Optional) Issue a software reset to reset the MCU. */
    __NVIC_SystemReset();
}
void iwdt_advanced_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) Enable the IWDT to count and generate NMI or reset when the
  * debugger is connected. */
    R_DEBUG->DBGSTOPCR_b.DBGSTOP_IWDT = 0;
 /* (Optional) Check if the IWDTRF flag is set to know if the system is
  * recovering from a IWDT reset. */
 if (R_SYSTEM->RSTSR1_b.IWDTRF)
    {
 /* Clear the flag. */
       R_SYSTEM->RSTSR1 = 0U;
    }
 /* Open the module. */
    err = R_IWDT_Open(&g_iwdt0_ctrl, &g_iwdt0_cfg);
```

```
/* Handle any errors. This function should be defined by the user. */
   handle_error(err);
/* Initialize other application code. */
/* Do not call R_IWDT_Refresh() in auto start mode unless the
 * counter is in the acceptable refresh window. */
   (void) R_IWDT_Refresh(&g_iwdt0_ctrl);
while (true)
   {
/* Application work here. */
/* (Optional) If there is a chance the application takes less time than
 * the start window, verify the IWDT counter is past the start window
 * before refreshing the IWDT. */
      uint32_t iwdt_counter = 0U;
do
     {
/* Read the current IWDT counter value. */
           err = R_IWDT_CounterGet(&g_iwdt0_ctrl, &iwdt_counter);
     handle_error(err);
     } while (iwdt_counter >= IWDT_START_WINDOW_75);
/* Refresh before the counter underflows to prevent reset or NMI. */
      (void) R_IWDT_Refresh(&g_iwdt0_ctrl);
   }
}
```

## Data Structures

| | |
|---|---|
| struct | iwdt_instance_ctrl_t |

## Data Structure Documentation

### ◆ iwdt_instance_ctrl_t

| struct iwdt_instance_ctrl_t | |
|---|---|
| IWDT control block. DO NOT INITIALIZE. Initialization occurs when wdt_api_t::open is called. | |
| **Data Fields** | |
| uint32_t | 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 )(wdt_callback_args_t *p_args) |
| | Callback provided when a WDT NMI ISR occurs. |
| | |

## Function Documentation

### ◆ R_IWDT_Refresh()

fsp_err_t R_IWDT_Refresh ( wdt_ctrl_t *const *p_api_ctrl*)

Refresh the Independent Watchdog Timer. 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.

Example:

```
 /* Refresh before the counter underflows to prevent reset or NMI based on the
setting. */
       (void) R_IWDT_Refresh(&g_iwdt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | IWDT successfully refreshed. |
| FSP_ERR_ASSERTION | One or more parameters are NULL pointers. |
| FSP_ERR_NOT_OPEN | The driver has not been opened. Perform R_IWDT_Open() first. |

### ◆ R_IWDT_Open()

fsp_err_t R_IWDT_Open ( wdt_ctrl_t *const *p_api_ctrl*, wdt_cfg_t const *const *p_cfg* )

Register the IWDT NMI callback.

Example:

```
/* Initializes the module. */
   err = R_IWDT_Open(&g_iwdt0_ctrl, &g_iwdt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | IWDT successfully configured. |
| FSP_ERR_ASSERTION | Null Pointer. |
| FSP_ERR_NOT_ENABLED | An attempt to open the IWDT when the OFS0 register is not configured for auto-start mode. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |

### ◆ R_IWDT_StatusClear()

fsp_err_t R_IWDT_StatusClear ( wdt_ctrl_t *const *p_api_ctrl*, const wdt_status_t *status* )

Clear the IWDT status and error flags. Implements wdt_api_t::statusClear.

Example:

```
/* (Optional) Clear the error flags. */
   err = R_IWDT_StatusClear(&g_iwdt0_ctrl, status);
   handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | IWDT flag(s) successfully cleared. |
| FSP_ERR_ASSERTION | Null pointer as a parameter. |
| FSP_ERR_NOT_OPEN | The driver has not been opened. Perform R_IWDT_Open() first. |

### ◆ R_IWDT_StatusGet()

fsp_err_t R_IWDT_StatusGet ( wdt_ctrl_t *const *p_api_ctrl*, wdt_status_t *const *p_status* )

Read the IWDT status flags. 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.

Indicates both status and error conditions.

Example:

```
/* (Optional) Determine the source of the NMI. */
wdt_status_t status = WDT_STATUS_NO_ERROR;
   err = R_IWDT_StatusGet(&g_iwdt0_ctrl, &status);
   handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | IWDT status successfully read. |
| FSP_ERR_ASSERTION | Null pointer as a parameter. |
| FSP_ERR_NOT_OPEN | The driver has not been opened. Perform R_IWDT_Open() first. |

### ◆ R_IWDT_CounterGet()

fsp_err_t R_IWDT_CounterGet ( wdt_ctrl_t *const *p_api_ctrl*, uint32_t *const *p_count* )

Read the current count value of the IWDT. Implements wdt_api_t::counterGet.

Example:

```
/* Read the current IWDT counter value. */
           err = R_IWDT_CounterGet(&g_iwdt0_ctrl, &iwdt_counter);
     handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | IWDT current count successfully read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | The driver has not been opened. Perform R_IWDT_Open() first. |

### ◆ R_IWDT_TimeoutGet()

fsp_err_t R_IWDT_TimeoutGet ( wdt_ctrl_t *const *p_api_ctrl*, wdt_timeout_values_t *const *p_timeout* )

Read timeout information for the watchdog timer. Implements wdt_api_t::timeoutGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | IWDT timeout information retrieved successfully. |
| FSP_ERR_ASSERTION | One or more parameters are NULL pointers. |
| FSP_ERR_NOT_OPEN | The driver has not been opened. Perform R_IWDT_Open() first. |

### ◆ R_IWDT_VersionGet()

fsp_err_t R_IWDT_VersionGet ( fsp_version_t *const *p_data*)

Return IWDT HAL driver version. Implements wdt_api_t::versionGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Call successful. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |

## 4.2.30 JPEG Codec (r_jpeg)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_JPEG_Open (jpeg_ctrl_t *const p_api_ctrl, jpeg_cfg_t const *const p_cfg) |
| fsp_err_t | R_JPEG_OutputBufferSet (jpeg_ctrl_t *p_api_ctrl, void *output_buffer, uint32_t output_buffer_size) |
| fsp_err_t | R_JPEG_InputBufferSet (jpeg_ctrl_t *constp_api_ctrl, void *p_data_buffer, uint32_t data_buffer_size) |
| fsp_err_t | R_JPEG_StatusGet (jpeg_ctrl_t *p_api_ctrl, jpeg_status_t *p_status) |

| | |
|---|---|
| fsp_err_t | R_JPEG_Close (jpeg_ctrl_t *p_api_ctrl) |
| fsp_err_t | R_JPEG_VersionGet (fsp_version_t *p_version) |
| fsp_err_t | R_JPEG_EncodeImageSizeSet (jpeg_ctrl_t *const p_api_ctrl, jpeg_encode_image_size_t *p_image_size) |
| fsp_err_t | R_JPEG_DecodeLinesDecodedGet (jpeg_ctrl_t *const p_api_ctrl, uint32_t *const p_lines) |
| fsp_err_t | R_JPEG_DecodeHorizontalStrideSet (jpeg_ctrl_t *p_api_ctrl, uint32_t horizontal_stride) |
| fsp_err_t | R_JPEG_DecodeImageSizeGet (jpeg_ctrl_t *p_api_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size) |
| fsp_err_t | R_JPEG_DecodeImageSubsampleSet (jpeg_ctrl_t *const p_api_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample) |
| fsp_err_t | R_JPEG_DecodePixelFormatGet (jpeg_ctrl_t *p_api_ctrl, jpeg_color_space_t *p_color_space) |
| fsp_err_t | R_JPEG_ModeSet (jpeg_ctrl_t *const p_api_ctrl, jpeg_mode_t mode) |

### Detailed Description

Driver for the JPEG peripheral on RA MCUs. This module implements the JPEG Codec Interface.

# Overview

The JPEG Codec is a hardware block providing accelerated JPEG image encode and decode functionality independent of the CPU. Images can optionally be partially processed facilitating streaming applications.

### Features

The JPEG Codec provides a number of options useful in a variety of applications:

- Basic encoding and decoding
- Streaming input and/or output
- Decoding JPEGs of unknown size
- Shrink (sub-sample) an image during the decoding process
- Rearrange input and output byte order (byte, word and/or longword swap)
- JPEG error detection

The specifications for the codec are as follows:

| Feature | Options |
|---|---|
| Decompression input formats | Baseline JPEG Y'CbCr 4:4:4, 4:2:2, 4:2:0 and 4:1:1 |
| Decompression output formats | ARGB8888, RGB565 |
| Compression input formats | Raw Y'CbCr 4:2:2 only |
| Compression output formats | Baseline JPEG Y'CbCr 4:2:2 only |
| Byte reordering | Byte, halfword and/or word swapping on input and output |
| Interrupt sources | Image size acquired, input/output data pause, decode complete, error |
| Compatible image sizes | See Minimum Coded Unit (MCU) below |

# Configuration

### Build Time Configurations for r_jpeg

The following build time configurations are defined in fsp_cfg/r_jpeg_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected, code for parameter checking is included in the build. |
| Decode Support | • Enabled<br>• Disabled | Enabled | If selected, code for decoding JPEG images is included in the build. |
| Encode Support | • Enabled<br>• Disabled | Disabled | If selected, code for encoding JPEG images is included in the build. |

### Configurations for Driver > Graphics > JPEG Codec Driver on r_jpeg

This module can be added to the Stacks tab via New Stack > Driver > Graphics > JPEG Codec Driver on r_jpeg:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General > Name | Name must be a valid C symbol | g_jpeg0 | Module name. |
| General > Default mode | • Decode<br>• Encode | Decode | Set the mode to use when calling R_JPEG_Open. This parameter is only used when both Encode and Decode support are enabled. |

| | | | |
|---|---|---|---|
| Decode > Input byte order | MCU Specific Options | | Select the byte order of the input data for decoding. |
| Decode > Output byte order | MCU Specific Options | | Select the byte order of the output data for decoding. |
| Decode > Output color format | • ARGB8888 (32-bit)<br>• RGB565 (16-bit) | RGB565 (16-bit) | Select the output pixel format for decode operations. |
| Decode > Output alpha (ARGB8888 only) | Value must be an 8-bit integer (0-255) | 255 | Specify the alpha value to apply to each output pixel when ARGB8888 format is chosen. |
| Decode > Callback | Name must be a valid C symbol | NULL | If a callback function is provided it will be called from the interrupt service routine (ISR) each time a related IRQ triggers. |
| Encode > Horizontal resolution | Value cannot be greater than 65535 and must be a non-negative integer divisible by 16 | 480 | Horizontal resolution of the raw image (in pixels). This value can be configured at runtime via R_JPEG_ImageSizeSet. |
| Encode > Vertical resolution | Value cannot be greater than 65535 and must be a non-negative integer divisible by 8 | 272 | Vertical resolution of the raw image. This value can be configured at runtime via R_JPEG_ImageSizeSet. |
| Encode > Horizontal stride | Value cannot be greater than 65535 and must be a non-negative integer | 480 | Horizontal stride of the raw image buffer (in pixels). This value can be configured at runtime via R_JPEG_ImageSizeSet. |
| Encode > Input byte order | MCU Specific Options | | Select the byte order of the input data for encoding. |
| Encode > Output byte order | MCU Specific Options | | Select the byte order of the output data for encoding. |
| Encode > Reset interval | Value cannot be greater than 65535 and must be a non-negative integer | 512 | Set the number of MCUs between RST markers. A value of 0 will disable DRI and RST marker output. |

| | | | |
|---|---|---|---|
| Encode > Quality factor | Value must be between 1 and 100 and must be an integer | 50 | Set the quality factor for encoding (1-100). Lower values produce smaller images at the cost of image quality. |
| Encode > Callback | Name must be a valid C symbol | NULL | If a callback function is provided it will be called from the interrupt service routine (ISR) each time a related IRQ triggers. |
| Interrupts > Decode Process Interrupt Priority | MCU Specific Options | | Select the decompression interrupt priority. |
| Interrupts > Data Transfer Interrupt Priority | MCU Specific Options | | Select the data transfer interrupt priority. |

### Clock Configuration

The peripheral clock for this module is PCLKA. No clocks are provided by this module.

### Pin Configuration

This module does not have any input or output pin connections.

## Usage Notes

### Overview

The JPEG Codec contains both decode and encode hardware. While these two functions are largely independent in configuration only one can be used at a time.

To switch from decode to encode mode (or vice versa) use R_JPEG_ModeSet while the JPEG Codec is idle.

### Status

The status value (jpeg_status_t) provided by the callback and by R_JPEG_StatusGet is a bitfield that encompasses all potential status indication conditions. One or more statuses can be set simultaneously.

### Decoding Process

JPEG decoding can be performed in several ways depending on the application:

- To perform the simplest decode operation where all dimensions are known:
  - Set the input buffer, stride and output buffer then wait for a callback with status JPEG_STATUS_OPERATION_COMPLETE.
- To pause after decoding the JPEG header (in order to acquire image dimensions and secure an output buffer):
  - Call R_JPEG_InputBufferSet before setting the output buffer and wait for a callback with status JPEG_STATUS_IMAGE_SIZE_READY.

- To decode a partial JPEG image then pause until the next chunk is available:
  - Specify a size smaller than the full JPEG data when calling R_JPEG_InputBufferSet.
- To pause decoding once an output buffer is filled:
  - Specify a size smaller than the full decoded image when calling R_JPEG_OutputBufferSet.

The flowchart below illustrates the steps necessary to handle any decode operation. The statuses given in blue are part of jpeg_status_t with the JPEG_DECODE_STATUS prefix omitted.

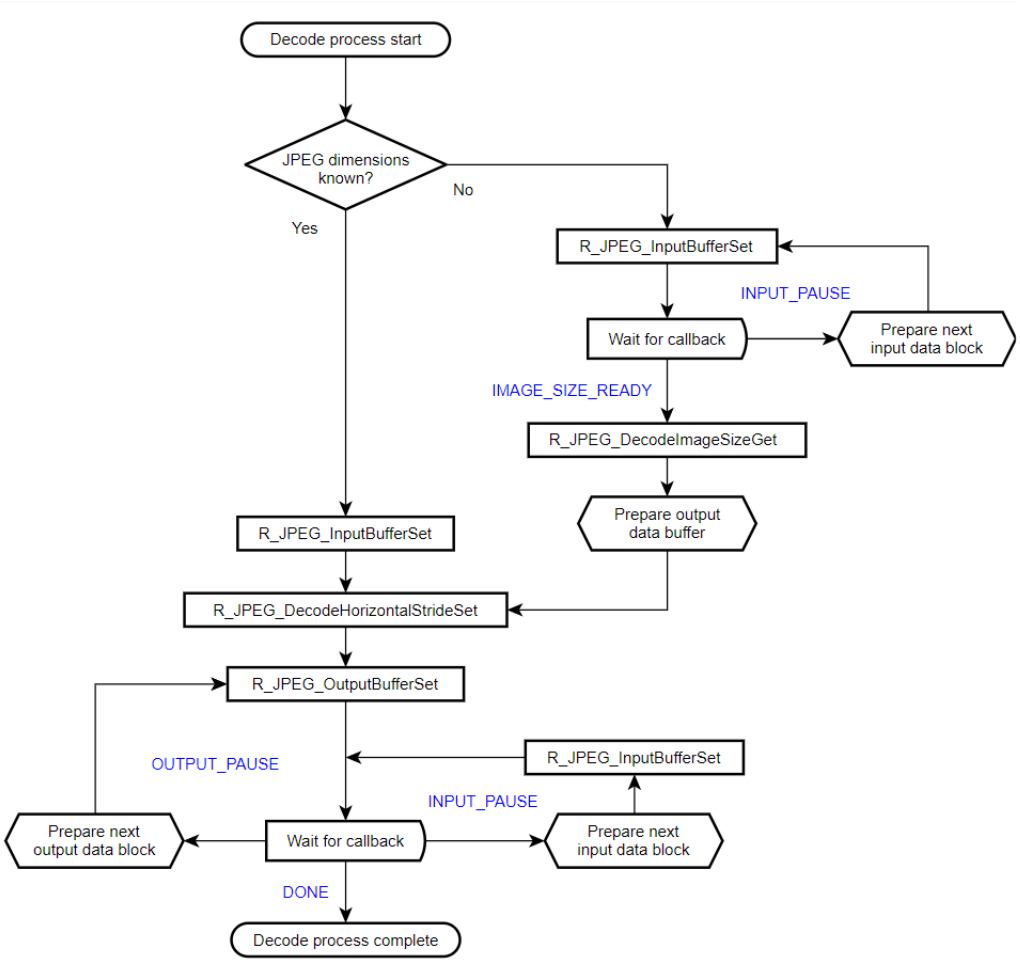



Figure 126: JPEG Decode Operational Flow

### Encoding Process

As compared to decoding, encoding is fairly straightforward. The only option available is to stream input data if desired. The flowchart below details the steps needed to compress an image.

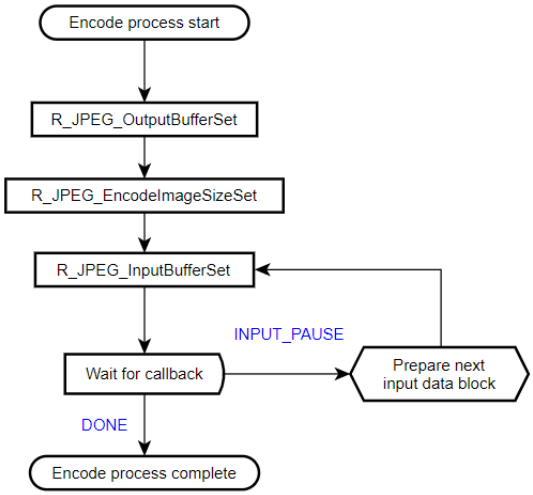



Figure 127: JPEG Encode Operational Flow

### Handling Failed Operations

If an encode or decode operation fails or times out while the codec is running, the peripheral must be reset before it is used again. To reset the JPEG Codec simply close and re-open the module by calling R_JPEG_Close followed by R_JPEG_Open.

### Limitations

Developers should be aware of the following limitations when using the JPEG API.

### Minimum Coded Unit (MCU)

The JPEG Codec can only correctly process images that are an even increment of minimum coded units (MCUs). In other words, depending on the format the width and height of an image to be encoded or decoded must be divisible by the following:

| Format | Horizontal | Vertical |
|---|---|---|
| Y'CbCr 4:4:4 | 8 pixels | 8 lines |
| Y'CbCr 4:2:2 | 16 pixels | 8 lines |
| Y'CbCr 4:1:1 | 32 pixels | 8 lines |
| Y'CbCr 4:2:0 | 16 pixels | 16 lines |

### Encoding Input Format

The encoding unit only supports Y'CbCr 4:2:2 input. Raw RGB888 data can be converted to this format as follows:

```
y =       (0.299000f * r) + (0.587000f * g) + (0.114000f * b);
cb = 128 - (0.168736f * r) - (0.331264f * g) + (0.500000f * b);
cr = 128 + (0.500000f * r) - (0.418688f * g) - (0.081312f * b);
```

While these equations are mathematically simple they do use the floating-point unit. To speed things up we can multiply the coefficients by 256/256...

```
y =       ((76.5440f * r) + (150.272f * g) + (29.1840f * b)) / 256;
cb = 128 - ((43.1964f * r) - (84.8036f * g) + (128.000f * b)) / 256;
cr = 128 + ((128.000f * r) - (107.184f * g) - (20.8159f * b)) / 256;
```

...which allows the formulas to be calculated entirely with shifts and addition (coefficients rounded to the nearest integer):

```
y =       (   (r << 6) + (r << 3) + (r << 2) + r
            + (g << 7) + (g << 4) + (g << 2) + (g << 1)
            + (b << 4) + (b << 3) + (b << 2) + b
           ) >> 8;
cb = 128 - (   (r << 5) + (r << 3) + (r << 1) + r
            + (g << 6) + (g << 4) + (g << 2) + g
            - (b << 7)
           ) >> 8;
cr = 128 + (   (r << 7)
            - (g << 6) - (g << 5) - (g << 3) - (g << 1) - g
            - (b << 4) - (b << 2) - b)
           ) >> 8;
```

To compose the final Y'CbCr 4:2:2 data the chroma of every two pixels must be averaged. **In addition, the JPEG Codec expects chrominance values to be in the range -127..127 instead of the standard 1..255.**

```
cb = (uint8_t) ((int8_t) ((cb0 + cb1 + 1) >> 1) - 128);
cr = (uint8_t) ((int8_t) ((cr0 + cr1 + 1) >> 1) - 128);
```

Finally, the below equation composes two 4:2:2 output pixels at a time with standard byte order (JPEG_DATA_ORDER_NORMAL):

```
out = y0 + (cb << 8) + (y1 << 16) + (cr << 24);
```

*Note*

*RGB565 pixels must be upscaled to RGB888 before using the above formulas. Refer to the below example on Y'CbCr Conversion for implementation details.*

# Examples

### Basic Decode Example

This is a basic example showing the minimum code required to initialize the JPEG Codec and decode an image.

```
void jpeg_decode_basic (void)
{
 fsp_err_t err;
 /* Open JPEG Codec */
    err = R_JPEG_Open(&g_jpeg_ctrl, &g_jpeg_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Set input buffer */
    err = R_JPEG_InputBufferSet(&g_jpeg_ctrl, JPEG_PTR, JPEG_SIZE_BYTES);
    handle_error(err);
 /* Set horizontal stride of output buffer */
    err = R_JPEG_DecodeHorizontalStrideSet(&g_jpeg_ctrl, JPEG_HSIZE);
    handle_error(err);
 /* Set output buffer */
    err = R_JPEG_OutputBufferSet(&g_jpeg_ctrl, decode_buffer, sizeof(decode_buffer));
    handle_error(err);
 /* Wait for decode completion */
 jpeg_status_t status = (jpeg_status_t) 0;
 while (!(status & (JPEG_STATUS_OPERATION_COMPLETE | JPEG_STATUS_ERROR)))
    {
       err = R_JPEG_StatusGet(&g_jpeg_ctrl, &status);
      handle_error(err);
    }
}
```

### Streaming Input/Output Example

In this example JPEG data is read in 512-byte chunks. Decoding is paused when a chunk is read and

once the JPEG header is decoded. The image is decoded 16 lines at a time.

*Note*

*Streaming is always bypassed when a given buffer's size encompasses the entire input or output image, respectively. Though this example decodes via smaller chunks the input and output data are still contiguous for ease of demonstration. Refer to the comments for further insight as to how to implement streaming with different JPEG/output buffer size combinations.*

```
#define JPEG_INPUT_SIZE_BYTES 512U
/* JPEG Codec status */
static volatile jpeg_status_t g_jpeg_status = JPEG_STATUS_NONE;
/* JPEG event flag */
static volatile uint8_t jpeg_event = 0;
/* Callback function for JPEG decode interrupts */
void jpeg_decode_callback (jpeg_callback_args_t * p_args)
{
 /* Get JPEG Codec status */
    g_jpeg_status = p_args->status;
 /* Set JPEG flag */
    jpeg_event = 1;
}
/* Simple wait that returns 1 if no event happened within the timeout period */
static uint8_t jpeg_event_wait (void)
{
    uint32_t timeout_timer = JPEG_EVENT_TIMEOUT;
 while (!jpeg_event && --timeout_timer)
    {
 /* Spin here until an event callback or timeout */
    }
    jpeg_event = 0;
 return timeout_timer ? 0 : 1;
}
/* Decode a JPEG image to a buffer using streaming input and output */
void jpeg_decode_streaming (void)
{
    uint8_t    * p_jpeg = (uint8_t *) JPEG_PTR;
 jpeg_status_t status = (jpeg_status_t) 0;
```

```
   uint8_t       timeout = 0;
fsp_err_t    err;
/* Number of input bytes to read at a time */
   uint32_t input_bytes = JPEG_INPUT_SIZE_BYTES;
/* Open JPEG unit and start decode */
   err = R_JPEG_Open(&g_jpeg_ctrl, &g_jpeg_cfg);
/* Handle any errors. This function should be defined by the user. */
   handle_error(err);
while (!(status & JPEG_STATUS_ERROR) && !timeout)
   {
/* Set the input buffer to read `input_bytes` bytes at a time */
      err = R_JPEG_InputBufferSet(&g_jpeg_ctrl, p_jpeg, input_bytes);
     handle_error(err);
/* This delay is required for streaming input mode to function correctly.
 * (Without this delay the JPEG Codec will not correctly locate markers in the file
header.) */
R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MICROSECONDS);
/* Wait for a callback */
      timeout = jpeg_event_wait();
/* Get the status from the callback */
      status = g_jpeg_status;
/* Break if the header has finished decoding */
if (status & JPEG_STATUS_IMAGE_SIZE_READY)
     {
break;
     }
/* Move pointer to next block of input data (if needed) */
      p_jpeg = (uint8_t *) ((uint32_t) p_jpeg + input_bytes);
   }
/* Get image size */
   uint16_t horizontal;
   uint16_t vertical;
   err = R_JPEG_DecodeImageSizeGet(&g_jpeg_ctrl, &horizontal, &vertical);
   handle_error(err);
```

```
/* Prepare output data buffer here if needed (already allocated in this example) */
   uint8_t * p_output = decode_buffer;
/* Set horizontal stride */
   err = R_JPEG_DecodeHorizontalStrideSet(&g_jpeg_ctrl, horizontal);
   handle_error(err);
/* Calculate the number of bytes that will fit in the buffer (16 lines in this
example) */
   uint32_t output_size = horizontal * 16U * 4U;
/* Start decoding by setting the output buffer */
   err = R_JPEG_OutputBufferSet(&g_jpeg_ctrl, p_output, output_size);
   handle_error(err);
while (!(status & JPEG_STATUS_ERROR) && !timeout)
   {
/* Wait for a callback */
      timeout = jpeg_event_wait();
/* Get the status from the callback */
      status = g_jpeg_status;
/* Break if decoding is complete */
if (status & JPEG_STATUS_OPERATION_COMPLETE)
     {
break;
     }
if (status & JPEG_STATUS_OUTPUT_PAUSE)
     {
/* Draw the JPEG work buffer to the framebuffer here (if needed) */
/* Move pointer to next block of output data (if needed) */
           p_output += output_size;
/* Set the output buffer to the next 16-line block */
     err = R_JPEG_OutputBufferSet(&g_jpeg_ctrl, p_output, output_size);
     handle_error(err);
     }
if (status & JPEG_STATUS_INPUT_PAUSE)
     {
/* Get next block of input data */
```

```
        p_jpeg = (uint8_t *) ((uint32_t) p_jpeg + input_bytes);
 /* Set the new input buffer pointer */
        err = R_JPEG_InputBufferSet(&g_jpeg_ctrl, p_jpeg, input_bytes);
      handle_error(err);
      }
    }
 /* Close driver to allow encode operations if needed */
    err = R_JPEG_Close(&g_jpeg_ctrl);
    handle_error(err);
}
```

### Encode Example

This is a basic example showing the minimum code required to initialize the JPEG Codec and encode an image.

*Note*

*This example assumes image dimensions are provided in the configuration. If this is not the case, R_JPEG_EncodeImageSizeSet must be used to set the size before calling R_JPEG_InputBufferSet.*

```
void jpeg_encode_basic (void)
{
 fsp_err_t err;
 /* Open JPEG Codec */
    err = R_JPEG_Open(&g_jpeg_ctrl, &g_jpeg_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Set output buffer */
    err = R_JPEG_OutputBufferSet(&g_jpeg_ctrl, jpeg_buffer, sizeof(jpeg_buffer));
    handle_error(err);
 /* Set input buffer */
    err = R_JPEG_InputBufferSet(&g_jpeg_ctrl, RAW_YCBCR_IMAGE_PTR, IMAGE_SIZE_BYTES);
    handle_error(err);
 /* Wait for decode completion */
 jpeg_status_t status = (jpeg_status_t) 0;
 while (!(status & JPEG_STATUS_OPERATION_COMPLETE))
    {
```

```
        err = R_JPEG_StatusGet(&g_jpeg_ctrl, &status);
      handle_error(err);
    }
}
```

### Streaming Encode Example

In this example the raw input data is provided in smaller chunks. This can help significantly reduce buffer size and improve throughput when streaming in raw data from an outside source.

```
/* Callback function for JPEG encode interrupts */
void jpeg_encode_callback (jpeg_callback_args_t * p_args)
{
 /* Get JPEG Codec status */
    g_jpeg_status = p_args->status;
 /* Set JPEG flag */
    jpeg_event = 1;
}
void jpeg_encode_streaming (void)
{
    uint8_t   timeout = 0;
    uint8_t * p_chunk = (uint8_t *) RAW_YCBCR_IMAGE_PTR;
 fsp_err_t err;
 /* Open JPEG Codec */
    err = R_JPEG_Open(&g_jpeg_ctrl, &g_jpeg_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Set output buffer */
    err = R_JPEG_OutputBufferSet(&g_jpeg_ctrl, jpeg_buffer, sizeof(jpeg_buffer));
    handle_error(err);
 /* Set the image size */
 jpeg_encode_image_size_t image_size;
    image_size.horizontal_resolution    = X_RESOLUTION;
    image_size.vertical_resolution      = Y_RESOLUTION;
    image_size.horizontal_stride_pixels = H_STRIDE;
```

```
    err = R_JPEG_EncodeImageSizeSet(&g_jpeg_ctrl, &image_size);
    handle_error(err);
 /* Calculate the size of the input data chunk (16 lines in this example) */
    uint32_t chunk_size = H_STRIDE * 16U * YCBCR_BYTES_PER_PIXEL;
 while (!timeout)
    {
 /* Set the input buffer */
       err = R_JPEG_InputBufferSet(&g_jpeg_ctrl, p_chunk, chunk_size);
      handle_error(err);
 /* Wait for a callback */
       timeout = jpeg_event_wait();
 if (g_jpeg_status & JPEG_STATUS_OPERATION_COMPLETE)
      {
 /* Encode complete */
 break;
      }
 if (g_jpeg_status & JPEG_STATUS_INPUT_PAUSE)
      {
 /* Load next block of input data here (if needed) */
            p_chunk += chunk_size;
      }
    }
}
```

### Y'CbCr Conversion

The below function is provided as a reference for how to convert RGB values to Y'CbCr for use with the JPEG Codec.

*Note*

*This function is only partially optimized for clarity. Further appllication-specific size- or speed-based optimizations should be considered when implementing in an actual project.*

```
#define RGB565_G_MASK 0x07E0
#define RGB565_B_MASK 0x001F
#define C_0 128
typedef enum e_pixel_format
```

```
{
    PIXEL_FORMAT_ARGB8888,
    PIXEL_FORMAT_RGB565
} pixel_format_t;
/* 5-bit to 8-bit LUT */
const uint8_t lut_32[] =
{
    0,   8,   16, 25, 33, 41, 49, 58,
    66, 74, 82, 90, 99, 107, 115, 123,
    132, 140, 148, 156, 165, 173, 181, 189,
    197, 206, 214, 222, 230, 239, 247, 255
};
/* 6-bit to 8-bit LUT */
const uint8_t lut_64[] =
{
    0,   4,   8,   12, 16, 20, 24, 28,
    32, 36, 40, 45, 49, 53, 57, 61,
    65, 69, 73, 77, 81, 85, 89, 93,
    97, 101, 105, 109, 113, 117, 121, 125,
    130, 134, 138, 142, 146, 150, 154, 158,
    162, 166, 170, 174, 178, 182, 186, 190,
    194, 198, 202, 206, 210, 215, 219, 223,
    227, 231, 235, 239, 243, 247, 251, 255
};
void bitmap_rgb2ycbcr(uint32_t * out, uint8_t * in, uint32_t len, pixel_format_t
format);
/*********************************************************************************
***********************************
 * Convert an RGB buffer to Y'CbCr 4:2:2.
 *
 * NOTE: The width (in pixels) of the image to be converted must be divisible by 2.
 *
 * Parameters:
 * out Pointer to output buffer
```

```
 * in Pointer to input buffer
 * len Length of input buffer (in pixels)
 * format Input buffer format (ARGB8888 or RGB565)
 ********************************************************************************
*********************************/
void bitmap_rgb2ycbcr (uint32_t * out, uint8_t * in, uint32_t len, pixel_format_t
format)
{
    uint16_t in0;
    uint16_t in1;
    uint32_t r0;
    uint32_t g0;
    uint32_t b0;
    uint32_t r1;
    uint32_t g1;
    uint32_t b1;
    uint8_t y0;
    uint8_t y1;
    uint8_t cb0;
    uint8_t cr0;
    uint8_t cb1;
    uint8_t cr1;
 /* Divide length by 2 as we're working with two pixels at a time */
    len >>= 1;
 /* Perform the conversion */
 while (len)
    {
 /* Get R, G and B channel values */
 if (format == PIXEL_FORMAT_RGB565)
      {
 /* Get next two 16-bit values */
            in0 = *((uint16_t *) in);
            in += 2;
            in1 = *((uint16_t *) in);
```

```
            in += 2;
/* Decompose into individual channels */
            r0 = in0 >> 11;
            g0 = (in0 & RGB565_G_MASK) >> 5;
            b0 = in0 & RGB565_B_MASK;
            r1 = in1 >> 11;
            g1 = (in1 & RGB565_G_MASK) >> 5;
            b1 = in1 & RGB565_B_MASK;
      }
else
      {
/* Get each ARGB8888 channel in sequence, skipping alpha */
            b0 = *in++;
            g0 = *in++;
            r0 = *in++;
            in++;
            b1 = *in++;
            g1 = *in++;
            r1 = *in++;
            in++;
      }
/* Convert RGB565 data to RGB888 */
if (PIXEL_FORMAT_RGB565 == format)
      {
            r0 = lut_32[r0];
            g0 = lut_64[g0];
            b0 = lut_32[b0];
            r1 = lut_32[r1];
            g1 = lut_64[g1];
            b1 = lut_32[b1];
      }
/* Calculate Y'CbCr 4:4:4 values for the two pixels */
/* Algorithm based on method shown here: https://sistenix.com/rgb2ycbcr.html */
/* Original coefficients from https://en.wikipedia.org/wiki/YCbCr#JPEG_conversion */
```

```
        y0 = (uint8_t) (((r0 << 6) + (r0 << 3) + (r0 << 2) + r0 +
                         (g0 << 7) + (g0 << 4) + (g0 << 2) + (g0 << 1) +
                         (b0 << 4) + (b0 << 3) + (b0 << 2) + b0
                         ) >> 8);
        cb0 = (uint8_t) (C_0 - (((r0 << 5) + (r0 << 3) + (r0 << 1) + r0 +
                                 (g0 << 6) + (g0 << 4) + (g0 << 2) + g0 -
                                 (b0 << 7)
                                 ) >> 8));
        cr0 = (uint8_t) (C_0 + (((r0 << 7) -
                                 (g0 << 6) - (g0 << 5) - (g0 << 3) - (g0 << 1) - g0 -
                                 (b0 << 4) - (b0 << 2) - b0
                                 ) >> 8));
        y1 = (uint8_t) (((r1 << 6) + (r1 << 3) + (r1 << 2) + r1 +
                         (g1 << 7) + (g1 << 4) + (g1 << 2) + (g1 << 1) +
                         (b1 << 4) + (b1 << 3) + (b1 << 2) + b1
                         ) >> 8);
        cb1 = (uint8_t) (C_0 - (((r1 << 5) + (r1 << 3) + (r1 << 1) + r1 +
                                 (g1 << 6) + (g1 << 4) + (g1 << 2) + g1 -
                                 (b1 << 7)
                                 ) >> 8));
        cr1 = (uint8_t) (C_0 + (((r1 << 7) -
                                 (g1 << 6) - (g1 << 5) - (g1 << 3) - (g1 << 1) - g1 -
                                 (b1 << 4) - (b1 << 2) - b1
                                 ) >> 8));
 /* The above code is based on the floating point method shown here: */
 // y0 = (uint8_t) ((0.299F * (float) r0) + (0.587F * (float) g0) + (0.114F * (float)
b0));
 // y1 = (uint8_t) ((0.299F * (float) r1) + (0.587F * (float) g1) + (0.114F * (float)
b1));
 // cb0 = (uint8_t) (128.0F - (0.168736F * (float) r0) - (0.331264F * (float) g0) +
(0.5F * (float) b0));
 // cb1 = (uint8_t) (128.0F - (0.168736F * (float) r1) - (0.331264F * (float) g1) +
(0.5F * (float) b1));
 // cr0 = (uint8_t) (128.0F + (0.5F * (float) r0) - (0.418688F * (float) g0) -
```

```
(0.081312F * (float) b0));
 // cr1 = (uint8_t) (128.0F + (0.5F * (float) r1) - (0.418688F * (float) g1) -
(0.081312F * (float) b1));
 /* NOTE: The JPEG Codec expects signed instead of unsigned chrominance values. */
 /* Convert chrominance to -127..127 instead of 1..255 */
       cb0 = (uint8_t) ((int8_t) ((cb0 + cb1 + 1) >> 1) - C_0);
       cr0 = (uint8_t) ((int8_t) ((cr0 + cr1 + 1) >> 1) - C_0);
 /* Convert the two 4:4:4 values into 4:2:2 by averaging the chroma, then write to
output */
       *out++ = (uint32_t) (y0 + (cb0 << 8) + (y1 << 16) + (cr0 << 24));
       len--;
    }
}
```

## Data Structures

| | |
|---|---|
| struct | jpeg_instance_ctrl_t |

## Data Structure Documentation

### ◆ jpeg_instance_ctrl_t

| struct jpeg_instance_ctrl_t | | |
|---|---|---|
| JPEG Codec module control block. DO NOT INITIALIZE. Initialization occurs when jpep_api_t::open is called. | | |
| Data Fields | | |
| uint32_t | open | JPEG Codec driver status. |
| jpeg_status_t | status | JPEG Codec operational status. |
| fsp_err_t | error_code | JPEG Codec error code (if any). |
| jpeg_mode_t | mode | Current mode (decode or encode). |
| uint32_t | horizontal_stride_bytes | Horizontal Stride settings. |
| uint32_t | output_buffer_size | Output buffer size. |
| jpeg_cfg_t const * | p_cfg | JPEG Decode configuration struct. |
| void const * | p_extend | JPEG Codec hardware dependent configuration */. |
| jpeg_decode_pixel_format_t | pixel_format | Pixel format. |
| uint16_t | total_lines_decoded | Track the number of lines |

| | | |
|---|---|---|
| | | decoded so far. |
| jpeg_decode_subsample_t | horizontal_subsample | Horizontal sub-sample setting. |
| uint16_t | lines_to_encode | Number of lines to encode. |
| uint16_t | vertical_resolution | vertical size |
| uint16_t | total_lines_encoded | Number of lines encoded. |

## Function Documentation

### ◆ R_JPEG_Open()

fsp_err_t R_JPEG_Open ( jpeg_ctrl_t *const *p_api_ctrl*, jpeg_cfg_t const *const *p_cfg* )

Initialize the JPEG Codec module.

***Note***

*This function configures the JPEG Codec for operation and sets up the registers for data format and pixel format based on user-supplied configuration parameters. Interrupts are enabled to support callbacks.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | JPEG Codec module is properly configured and is ready to take input data. |
| FSP_ERR_ALREADY_OPEN | JPEG Codec is already open. |
| FSP_ERR_ASSERTION | Pointer to the control block or the configuration structure is NULL. |
| FSP_ERR_IRQ_BSP_DISABLED | JEDI interrupt does not have an IRQ number. |
| FSP_ERR_INVALID_ARGUMENT | (Encode only) Quality factor, horizontal resolution and/or vertical resolution are invalid. |
| FSP_ERR_INVALID_ALIGNMENT | (Encode only) The horizontal resolution (at 16bpp) is not divisible by 8 bytes. |

### ◆ R_JPEG_OutputBufferSet()

fsp_err_t R_JPEG_OutputBufferSet ( jpeg_ctrl_t * *p_api_ctrl*, void * *p_output_buffer*, uint32_t *output_buffer_size* )

Assign a buffer to the JPEG Codec for storing output data.

***Note***

*In Decode mode, 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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The output buffer is properly assigned to JPEG codec device. |
| FSP_ERR_ASSERTION | Pointer to the control block or output_buffer is NULL or output_buffer_size is 0. |
| FSP_ERR_INVALID_ALIGNMENT | Buffer starting address is not 8-byte aligned. |
| FSP_ERR_NOT_OPEN | JPEG not opened. |
| FSP_ERR_JPEG_UNSUPPORTED_IMAGE_SIZE | The number of horizontal pixels exceeds horizontal memory stride. |
| FSP_ERR_JPEG_BUFFERSIZE_NOT_ENOUGH | Invalid buffer size. |
| FSP_ERR_IN_USE | The output buffer cannot be changed during codec operation. |

### ◆ R_JPEG_InputBufferSet()

fsp_err_t R_JPEG_InputBufferSet ( jpeg_ctrl_t *const *p_api_ctrl*, void * *p_data_buffer*, uint32_t *data_buffer_size* )

Assign an input data buffer to the JPEG codec for processing.

***Note***

*After the amount of data is processed, the JPEG driver triggers a callback function with the flag JPEG_PRV_OPERATION_INPUT_PAUSE set. The application supplies the next chunk of data to the driver so processing 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.*

If zero is provided for the decode data buffer size the JPEG Codec will never pause for more input data and will continue to read until either an image has been fully decoded or an error condition occurs.

***Note***

*When encoding images the minimum data buffer size is 8 lines by 16 Y'CbCr 4:2:2 pixels (256 bytes). This corresponds to one minimum coded unit (MCU) of the resulting JPEG output.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The input data buffer is properly assigned to JPEG Codec device. |
| FSP_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. |
| FSP_ERR_INVALID_ALIGNMENT | Buffer starting address is not 8-byte aligned. |
| FSP_ERR_NOT_OPEN | JPEG not opened. |
| FSP_ERR_IN_USE | The input buffer cannot be changed while the codec is running. |
| FSP_ERR_INVALID_CALL | In encode mode the output buffer must be set first. |
| FSP_ERR_JPEG_IMAGE_SIZE_ERROR | The buffer size is smaller than the minimum coded unit (MCU). |

### ◆ R_JPEG_StatusGet()

fsp_err_t R_JPEG_StatusGet ( jpeg_ctrl_t * *p_api_ctrl*, jpeg_status_t * *p_status* )

Get the status of the JPEG codec. This function can also be used to poll the device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The status information is successfully retrieved. |
| FSP_ERR_ASSERTION | Pointer to the control block or p_status is NULL. |
| FSP_ERR_NOT_OPEN | JPEG is not opened. |

### ◆ R_JPEG_Close()

fsp_err_t R_JPEG_Close ( jpeg_ctrl_t * *p_api_ctrl*)

Cancel an outstanding JPEG codec operation and close the device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The input data buffer is properly assigned to JPEG Codec device. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL. |
| FSP_ERR_NOT_OPEN | JPEG not opened. |

### ◆ R_JPEG_VersionGet()

fsp_err_t R_JPEG_VersionGet ( fsp_version_t * *p_version*)

Get the version of the JPEG Codec driver.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version number returned successfully. |
| FSP_ERR_ASSERTION | The parameter p_version is NULL. |

### ◆ R_JPEG_EncodeImageSizeSet()

fsp_err_t R_JPEG_EncodeImageSizeSet ( jpeg_ctrl_t *const *p_api_ctrl*, jpeg_encode_image_size_t * *p_image_size* )

Set the image dimensions for an encode operation.

***Note***

*Image dimensions must be set before setting the input buffer.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Image size was successfully written to the JPEG Codec. |
| FSP_ERR_ASSERTION | Pointer to the control block or p_image_size is NULL. |
| FSP_ERR_INVALID_ALIGNMENT | Horizontal stride is not 8-byte aligned. |
| FSP_ERR_INVALID_ARGUMENT | Horizontal or vertical resolution is invalid or zero. |
| FSP_ERR_NOT_OPEN | JPEG not opened. |
| FSP_ERR_IN_USE | Image parameters cannot be changed while the codec is running. |

### ◆ R_JPEG_DecodeLinesDecodedGet()

fsp_err_t R_JPEG_DecodeLinesDecodedGet ( jpeg_ctrl_t * *p_api_ctrl*, uint32_t * *p_lines* )

Returns the number of lines decoded into the output buffer.

***Note***

*Use this function to retrieve the number of image lines written to the output buffer after a partial decode operation. 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.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Line count successfully returned. |
| FSP_ERR_ASSERTION | Pointer to the control block or p_lines is NULL. |
| FSP_ERR_NOT_OPEN | JPEG not opened. |

### ◆ R_JPEG_DecodeHorizontalStrideSet()

fsp_err_t R_JPEG_DecodeHorizontalStrideSet ( jpeg_ctrl_t * *p_api_ctrl*, uint32_t *horizontal_stride* )

Configure horizontal stride setting for decode operations.

***Note***

*If the image size is known prior to the open call and/or the output buffer stride is constant, pass the horizontal stride value in the jpeg_cfg_t structure. Otherwise, after the image size becomes available use this function to set the output buffer horizontal stride value.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Horizontal stride value is properly configured. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL. |
| FSP_ERR_INVALID_ALIGNMENT | Horizontal stride is zero or is not 8-byte aligned. |
| FSP_ERR_NOT_OPEN | JPEG not opened. |

### ◆ R_JPEG_DecodeImageSizeGet()

fsp_err_t R_JPEG_DecodeImageSizeGet ( jpeg_ctrl_t * *p_api_ctrl*, uint16_t * *p_horizontal_size*, uint16_t * *p_vertical_size* )

Obtain the size of an image being decoded.

**Return values**

| | |
|---|---|
| FSP_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. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL and/or size is not ready. |
| FSP_ERR_NOT_OPEN | JPEG is not opened. |

### ◆ R_JPEG_DecodeImageSubsampleSet()

fsp_err_t R_JPEG_DecodeImageSubsampleSet ( jpeg_ctrl_t *const *p_api_ctrl*, jpeg_decode_subsample_t *horizontal_subsample*, jpeg_decode_subsample_t *vertical_subsample* )

Configure horizontal and vertical subsampling.

***Note***

*This function can be used to scale the output of decoded image data.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Horizontal stride value is properly configured. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL. |
| FSP_ERR_NOT_OPEN | JPEG not opened. |

### ◆ R_JPEG_DecodePixelFormatGet()

fsp_err_t R_JPEG_DecodePixelFormatGet ( jpeg_ctrl_t * *p_api_ctrl*, jpeg_color_space_t * *p_color_space* )

Get the color format of the JPEG being decoded.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The color format was successfully retrieved. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL. |
| FSP_ERR_NOT_OPEN | JPEG is not opened. |

### ◆ R_JPEG_ModeSet()

fsp_err_t R_JPEG_ModeSet ( jpeg_ctrl_t *const *p_api_ctrl*, jpeg_mode_t *mode* )

Switch between encode and decode mode (or vice-versa).

***Note***

*The codec must not be idle in order to switch modes.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Mode changed successfully. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_IN_USE | JPEG Codec is currently in use. |
| FSP_ERR_INVALID_ARGUMENT | (Encode only) Quality factor, horizontal resolution and/or vertical resolution are invalid. |
| FSP_ERR_INVALID_ALIGNMENT | (Encode only) The horizontal resolution (at 16bpp) is not divisible by 8 bytes. |

## 4.2.31 Key Interrupt (r_kint)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | R_KINT_Open (keymatrix_ctrl_t *const p_api_ctrl, keymatrix_cfg_t const *const p_cfg) |
| fsp_err_t | R_KINT_Enable (keymatrix_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_KINT_Disable (keymatrix_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_KINT_Close (keymatrix_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_KINT_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Driver for the KINT peripheral on RA MCUs. This module implements the Key Matrix Interface.

# Overview

The KINT 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.

### Features

- Detect rising or falling edges on KINT channels
- Callback for notifying the application when edges are detected on the configured channels

# Configuration

### Build Time Configurations for r_kint

The following build time configurations are defined in fsp_cfg/r_kint_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking Enable | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Input > Key Matrix Driver on r_kint

This module can be added to the Stacks tab via New Stack > Driver > Input > Key Matrix Driver on r_kint:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General > Name | Name must be a valid C symbol | g_kint0 | Module name. |
| Input > Key Interrupt Flag Mask | • Channel 0<br>• Channel 1<br>• Channel 2<br>• Channel 3<br>• Channel 4<br>• Channel 5<br>• Channel 6<br>• Channel 7 | | Select channels to enable. |
| Interrupts > Trigger Type | • Falling Edge<br>• Rising Edge | Rising Edge | Specifies 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. |

| | | | |
|---|---|---|---|
| Interrupts > Callback | Name must be a valid C symbol | kint_callback | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the IRQ triggers. |
| Interrupts > Key Interrupt Priority | MCU Specific Options | | Select the key interrupt priority. |

### Clock Configuration

The KINT peripheral runs on PCLKB.

### Pin Configuration

The KRn pins are key switch matrix row input pins.

## Usage Notes

### Connecting a Switch Matrix

The KINT module is designed to scan the rows of a switch matrix where each row is connected to a number of columns through switches. A periodic timer (or other mechanism) sets one column pin high at a time. Any switches that are pressed on the driven column cause a rising (or falling) edge on the row pin (KRn) causing an interrupt.

*Note*

*In applications where multiple keys may be pressed at the same time it is recommended to put a diode inline with each switch to prevent ghosting.*

### Handling Multiple Pins

When an edge is detected on multiple pins at the same time, a single IRQ will be generated. A mask of all the pins that detected an edge will be passed to the callback.

## Examples

### Basic Example

This is a basic example of minimal use of the KINT in an application.

```
static volatile uint32_t g_channel_mask;
static volatile uint32_t g_kint_edge_detected = 0U;
/* Called from key_int_isr */
void r_kint_callback (keymatrix_callback_args_t * p_args)
{
    g_channel_mask       = p_args->channel_mask;
```

```
    g_kint_edge_detected = 1U;
}
void r_kint_example ()
{
 /* Configure the KINT. */
 fsp_err_t err = R_KINT_Open(&g_kint_ctrl, &g_kint_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Enable the KINT. */
    err = R_KINT_Enable(&g_kint_ctrl);
    handle_error(err);
 while (0 == g_kint_edge_detected)
    {
 /* Wait for interrupt. */
    }
}
```

## Data Structures

| | |
|---|---|
| struct | kint_instance_ctrl_t |

## Data Structure Documentation

### ◆ kint_instance_ctrl_t

| struct kint_instance_ctrl_t |
|---|
| Channel instance control block. DO NOT INITIALIZE. Initialization occurs when keymatrix_api_t::open is called. |

## Function Documentation

### ◆ R_KINT_Open()

fsp_err_t R_KINT_Open ( keymatrix_ctrl_t *const *p_api_ctrl*, keymatrix_cfg_t const *const *p_cfg* )

Configure all the Key Input (KINT) channels and provides a handle for use with the rest of the KINT API functions. Implements keymatrix_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful. |
| FSP_ERR_ASSERTION | One of the following parameters may be NULL: p_cfg, or p_ctrl or the callback. |
| FSP_ERR_ALREADY_OPEN | The module has already been opened. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel mask is invalid. |

### ◆ R_KINT_Enable()

fsp_err_t R_KINT_Enable ( keymatrix_ctrl_t *const *p_api_ctrl*)

This function enables interrupts for the KINT peripheral after clearing any pending requests. Implements keymatrix_api_t::enable.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Interrupt disabled successfully. |
| FSP_ERR_ASSERTION | The p_ctrl parameter was null. |
| FSP_ERR_NOT_OPEN | The peripheral is not opened. |

### ◆ R_KINT_Disable()

fsp_err_t R_KINT_Disable ( keymatrix_ctrl_t *const *p_api_ctrl*)

This function disables interrupts for the KINT peripheral. Implements keymatrix_api_t::disable.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Interrupt disabled successfully. |
| FSP_ERR_ASSERTION | The p_ctrl parameter was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |

### ◆ R_KINT_Close()

| fsp_err_t R_KINT_Close ( keymatrix_ctrl_t *const *p_api_ctrl*) |
|---|
| Clear the KINT configuration and disable the KINT IRQ. Implements keymatrix_api_t::close. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The module is not opened. |

### ◆ R_KINT_VersionGet()

| fsp_err_t R_KINT_VersionGet ( fsp_version_t *const *p_version*) |
|---|
| Set driver version based on compile time macros. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful return. |
| FSP_ERR_ASSERTION | The parameter p_version is NULL. |

## 4.2.32 Low Power Modes (r_lpm)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_LPM_Open (lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg) |
| fsp_err_t | R_LPM_Close (lpm_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_LPM_LowPowerReconfigure (lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg) |
| fsp_err_t | R_LPM_LowPowerModeEnter (lpm_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_LPM_VersionGet (fsp_version_t *const p_version) |
| fsp_err_t | R_LPM_IoKeepClear (lpm_ctrl_t *const p_api_ctrl) |

### Detailed Description

Driver for the LPM peripheral on RA MCUs. This module implements the Low Power Modes Interface.

# Overview

The low power modes driver is used to configure and place the device into the desired low power mode. Various sources can be configured to wake from standby, request snooze mode, end snooze mode or end deep standby mode.

### Features

The LPM HAL module has the following key features:

- Supports the follwowing low power modes:
  - Deep Software Standby mode (On supported MCUs)
  - Software Standby mode
  - Sleep mode
  - Snooze mode
- Supports reducing power consumption when in deep software standby mode through internal power supply control and by resetting the states of I/O ports.
- Supports disabling and enabling the MCU's other hardware peripherals

# Configuration

### Build Time Configurations for r_lpm

The following build time configurations are defined in fsp_cfg/r_lpm_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Power > Low Power Modes Driver on r_lpm

This module can be added to the Stacks tab via New Stack > Driver > Power > Low Power Modes Driver on r_lpm:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General > Name | Name must be a valid C symbol | g_lpm0 | Module name. |
| General > Low Power Mode | MCU Specific Options | | Power mode to be entered. |
| General > Output port state in standby and deep standby | MCU Specific Options | | Select the state of output pins during standby. Applies to address output, data output, and other bus |

| | | | |
|---|---|---|---|
| | | | control output pins. |
| Standby Options > Wake Sources | MCU Specific Options | | Enable wake from standby from these Sources. |
| Standby Options > Snooze Request Source | MCU Specific Options | | Select the event that will enter snooze. |
| Standby Options > Snooze End Sources | MCU Specific Options | | Enable wake from snooze from these sources. |
| Standby Options > DTC state in Snooze Mode | • Disabled<br>• Enabled | Disabled | Enable wake from snooze from this source. |
| Standby Options > Snooze Cancel Source | MCU Specific Options | | Select an interrupt source to cancel snooze. |
| Deep Standby Options > Maintain or reset the IO port states on exit from deep standby mode | MCU Specific Options | | Select the state of the IO Pins after exiting deep standby mode. |
| Deep Standby Options > Internal power supply control in deep standby mode | MCU Specific Options | | Select the state of the internal power supply in deep standby mode. |
| Deep Standby Options > Cancel Sources | MCU Specific Options | | Enable wake from deep standby using these sources. |
| Deep Standby Options > Cancel Edges | MCU Specific Options | | Falling edge trigger is default. Select sources to enable wake from deep standby with rising edge. |

### Clock Configuration

This module does not have any selectable clock sources.

### Pin Configuration

This module does not use I/O pins.

## Usage Notes

### Sleep Mode

At power on, by default sleep is set 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 RTOS scheduler.

### Software Standby Mode

In software-standby mode, the CPU, as well as most of the on-chip peripheral functions and all of the internal oscillators, are stopped. The contents of the CPU internal registers and SRAM data, the states of on-chip peripheral functions, and I/O Ports are all 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 requires an interrupt or event be configured and enabled to wake up.

### Snooze Mode

Snooze mode can be used with some MCU peripherals to execute basic tasks while keeping the MCU in a low-power state. Many core peripherals and all clocks can be selected to run during Snooze, allowing for more flexible low-power configuration than Software Standby mode. To enable Snooze, select "Software Standby mode with Snooze mode enabled" for the "Low Power Mode" configuration option. Snooze mode settings (including entry/exit sources) are available under "Standby Options".

### Deep Software Standby Mode

Deep Software Standby Mode is only available on some MCU devices. 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 the wakeup sources configurable in the "Deep Standby Options" configuration group.

The Reset Status Registers can be used to determine if the reset occured after coming out of deep sofware standby. For example, R_SYSTEM->RSTSR0_b.DPSRSTF is set to 1 after a deep software standby reset.

I/O Port Retention can be enabled to maintain I/O port configuration across a deep software standby reset. Retention can be cancelled through the R_LPM_IoKeepClear API.

### Limitations

Developers should be aware of the following limitations when using the LPM:

- Flash stop (code flash disable) is not supported. See the section "Flash Operation Control Register (FLSTOP)" of the RA2/RA4 Family 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 RA4 Hardware User's Manual.
- Only one Snooze Request Source can be used at a time.
- When using Snooze mode with SCI0 RXD as the snooze source the system clock must be HOCO and the MOCO, Main Oscillator and PLL clocks must be turned off.
- 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. When attached, the debugger will prevent the MCU from entering standby modes.
- 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 module properties. See the "Wakeup Timing and Duration" table in Electrical Characteristics for more information.

# Examples

### LPM Sleep Example

This is a basic example of minimal use of the LPM in an application. The LPM instance is opened and the configured low-power mode is entered.

```
void r_lpm_sleep (void)
{
 fsp_err_t err = R_LPM_Open(&g_lpm_ctrl, &g_lpm_cfg_sleep);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
    err = R_LPM_LowPowerModeEnter(&g_lpm_ctrl);
    handle_error(err);
}
```

### LPM Deep Software Standby Example

```
void r_lpm_deep_software_standby (void)
{
 fsp_err_t err;
    err = R_LPM_Open(&g_lpm_ctrl, &g_lpm_cfg_deep_software_standby);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Check the Deep Software Standby Reset Flag. */
 if (1U == R_SYSTEM->RSTSR0_b.DPSRSTF)
    {
 /* Clear the IOKEEP bit to allow I/O Port use. */
       err = R_LPM_IoKeepClear(&g_lpm_ctrl);
      handle_error(err);
    }
 /* Add user code here. */
 /* Reconfigure the module to set the IOKEEP bit before entering deep software
standby. */
```

```
    err = R_LPM_LowPowerReconfigure(&g_lpm_ctrl, &g_lpm_cfg_deep_software_standby);
    handle_error(err);
    err = R_LPM_LowPowerModeEnter(&g_lpm_ctrl);
 /* Code after R_LPM_LowPowerModeEnter when using Deep Software Standby never be
executed.
  * Deep software standby exits by resetting the MCU. */
    handle_error(err);
}
```

## Data Structures

| | |
|---|---|
| struct | lpm_instance_ctrl_t |

## Data Structure Documentation

### ◆ lpm_instance_ctrl_t

| struct lpm_instance_ctrl_t |
|---|
| LPM private control block. DO NOT MODIFY. Initialization occurs when R_LPM_Open() is called. |

## Function Documentation

### ◆ R_LPM_Open()

fsp_err_t R_LPM_Open ( lpm_ctrl_t *const *p_api_ctrl*, lpm_cfg_t const *const *p_cfg* )

Perform any necessary initialization

**Return values**

| | |
|---|---|
| FSP_SUCCESS | LPM instance opened |
| FSP_ERR_ASSERTION | Null Pointer |
| FSP_ERR_ALREADY_OPEN | LPM instance is already open |
| FSP_ERR_UNSUPPORTED | This MCU does not support Deep Software Standby |
| FSP_ERR_INVALID_ARGUMENT | One of the following:<br>• Invalid snooze entry source<br>• Invalid snooze end sources |
| FSP_ERR_INVALID_MODE | One of the following:<br>• Invalid low power mode<br>• Invalid DTC option for snooze mode<br>• Invalid deep standby end sources<br>• Invalid deep standby end sources edges<br>• Invalid power supply option for deep standby<br>• Invalid IO port option for deep standby<br>• Invalid output port state setting for standby or deep standby<br>• Invalid sources for wake from standby mode<br>• Invalid power supply option for standby<br>• Invalid IO port option for standby<br>• Invalid standby end sources<br>• Invalid standby end sources edges |

### ◆ R_LPM_Close()

fsp_err_t R_LPM_Close ( lpm_ctrl_t *const *p_api_ctrl*)

Close the LPM Instance

**Return values**

| | |
|---|---|
| FSP_SUCCESS | LPM Software lock initialized |
| FSP_ERR_NOT_OPEN | LPM instance is not open |
| FSP_ERR_ASSERTION | Null Pointer |

### ◆ R_LPM_LowPowerReconfigure()

fsp_err_t R_LPM_LowPowerReconfigure ( lpm_ctrl_t *const *p_api_ctrl*, lpm_cfg_t const *const *p_cfg* )

Configure a low power mode

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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Low power mode successfuly applied |
| FSP_ERR_ASSERTION | Null Pointer |
| FSP_ERR_NOT_OPEN | LPM instance is not open |
| FSP_ERR_UNSUPPORTED | This MCU does not support Deep Software Standby |
| FSP_ERR_INVALID_ARGUMENT | One of the following:<br>• Invalid snooze entry source<br>• Invalid snooze end sources |
| FSP_ERR_INVALID_MODE | One of the following:<br>• Invalid low power mode<br>• Invalid DTC option for snooze mode<br>• Invalid deep standby end sources<br>• Invalid deep standby end sources edges<br>• Invalid power supply option for deep standby<br>• Invalid IO port option for deep standby<br>• Invalid output port state setting for standby or deep standby<br>• Invalid sources for wake from standby mode<br>• Invalid power supply option for standby<br>• Invalid IO port option for standby<br>• Invalid standby end sources<br>• Invalid standby end sources edges |

### ◆ R_LPM_LowPowerModeEnter()

fsp_err_t R_LPM_LowPowerModeEnter ( lpm_ctrl_t *const *p_api_ctrl*)

Enter low power mode (sleep/standby/deep standby) using WFI macro.

Function will return after waking from low power mode.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | LPM instance is not open |
| FSP_ERR_INVALID_MODE | One of the following:<br>• HOCO was not system clock when using snooze mode with SCI0/RXD0.<br>• HOCO was not stable when using snooze mode with SCI0/RXD0.<br>• MOCO was running when using snooze mode with SCI0/RXD0.<br>• MAIN OSCILLATOR was running when using snooze mode with SCI0/RXD0.<br>• PLL was running when using snooze mode with SCI0/RXD0.<br>• Unable to disable ocillator stop detect when using standby or deep standby. |

### ◆ R_LPM_VersionGet()

fsp_err_t R_LPM_VersionGet ( fsp_version_t *const *p_version*)

Get the driver version based on compile time macros.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | p_version is NULL. |

### ◆ R_LPM_IoKeepClear()

| fsp_err_t R_LPM_IoKeepClear ( lpm_ctrl_t *const *p_api_ctrl*) |
|---|
| Clear the IOKEEP bit after deep software standby |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | DPSBYCR_b.IOKEEP bit cleared Successfully. |
| FSP_ERR_UNSUPPORTED | Deep standby mode not supported on this MCU. |

## 4.2.33 Low Voltage Detection (r_lvd)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_LVD_Open (lvd_ctrl_t *const p_api_ctrl, lvd_cfg_t const *const p_cfg) |
| fsp_err_t | R_LVD_Close (lvd_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_LVD_StatusGet (lvd_ctrl_t *const p_api_ctrl, lvd_status_t *p_lvd_status) |
| fsp_err_t | R_LVD_StatusClear (lvd_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_LVD_VersionGet (fsp_version_t *const p_version) |

**Detailed Description**

Driver for the LVD peripheral on RA MCUs. This module implements the Low Voltage Detection Interface.

# Overview

The Low Voltage Detection module configures the voltage monitors to detect when $V_{CC}$ crosses a specified threshold.

**Features**

The LVD HAL module supports the following functions:

- Two run-time configurable voltage monitors (Voltage Monitor 1, Voltage Monitor 2)

- Configurable voltage threshold
- Digital filter (Available on specific MCUs)
- Support for both interrupt or polling
    - NMI or maskable interrupt can be configured
- Rising, falling, or both edge event detection
- Support for resetting the MCU when $V_{CC}$ falls below configured threshold.

# Configuration

### Build Time Configurations for r_lvd

The following build time configurations are defined in fsp_cfg/r_lvd_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Power > Low Voltage Detection Driver on r_lvd

This module can be added to the Stacks tab via New Stack > Driver > Power > Low Voltage Detection Driver on r_lvd:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_lvd | Module name. |
| Monitor Number | • 1<br>• 2 | 1 | Select the LVD monitor. |
| Digital Filter Setting | MCU Specific Options | | Enable the digital filter and select the digital filter clock divider. |
| Voltage Threshold | MCU Specific Options | | Select the low voltage detection threshold. |
| Detection Response | • Maskable interrupt<br>• Non-maskable interrupt<br>• Reset MCU (Only available for falling edge)<br>• No response (Voltage monitor status will be polled) | No response (Voltage monitor status will be polled) | Select what happens when the voltage crosses the threshold voltage. |
| Voltage Slope | • Falling voltage<br>• Rising voltage | Falling voltage | Select detection on rising voltage, falling |

| | | | |
|---|---|---|---|
| | • Rising or falling voltage | | voltage or both. |
| Negation Delay | • Delay from reset<br>• Delay from voltage returning to normal range | Delay from reset | Negation of the monitor signal can either be delayed from the reset event or from voltage returning to normal range. |
| Monitor Interrupt Callback | Name must be a valid C symbol. | NULL | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the IRQ triggers. |
| LVD Monitor Interrupt Priority | MCU Specific Options | | Select the LVD Monitor interrupt priority. |

### Clock Configuration

The LOCO clock must be enabled in order to use the digital filter.

### Pin Configuration

This module does not use I/O pins.

## Usage Notes

### Startup Edge Detection

If $V_{CC}$ is below the threshold prior to configuring the voltage monitor for falling edge detection, the monitor will immediately detect the a falling edge condition. If $V_{CC}$ is above the threshold prior to configuring the monitor for rising edge detection, the monitor will not detect a rising edge condition until $V_{CC}$ falls below the threshold and then rises above it again.

### Voltage Monitor 0

The LVD HAL module only supports configuring voltage monitor 1 and voltage monitor 2. Voltage monitor 0 can be configured by setting the appropriate bits in the OFS1 register. This means that voltage monitor 0 settings cannot be changed at runtime.

Voltage monitor 0 supports the following features

- Configurable Voltage Threshold ($V_{DET0}$)
- Reset the device when $V_{CC}$ falls below $V_{DET0}$

### Limitations

- The digital filter must be disabled when using voltage monitors in Software Standby or Deep Software Standby.
- Deep Software Standby mode is not possible if the voltage monitor is configured to reset

the MCU.
- When the detection response is set to reset, only voltage falling edge detection is possible.

# Examples

### Basic Example

This is a basic example of minimal use of the LVD in an application.

```
void basic_example (void)
{
 fsp_err_t err = R_LVD_Open(&g_lvd_ctrl, &g_lvd_cfg);
    handle_error(err);
 while (1)
    {
 lvd_status_t status;
       err = R_LVD_StatusGet(&g_lvd_ctrl, &status);
      handle_error(err);
 if (LVD_THRESHOLD_CROSSING_DETECTED == status.crossing_detected)
      {
            err = R_LVD_StatusClear(&g_lvd_ctrl);
      handle_error(err);
 /* Do something */
       }
    }
}
```

### Interrupt Example

This is a basic example of using a LVD instance that is configured to generate an interrupt.

```
void interrupt_example (void)
{
 fsp_err_t err = R_LVD_Open(&g_lvd_ctrl, &g_lvd_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
```

```
 /* Application Process */
 /* Application will be interrupted when Vcc crosses the configured threshold. */
    }
}
/* Called when Vcc crosses configured threshold. */
void lvd_callback (lvd_callback_args_t * p_args)
{
 if (LVD_CURRENT_STATE_BELOW_THRESHOLD == p_args->current_state)
    {
 /* Do Something */
    }
}
```

### Reset Example

This is a basic example of using a LVD instance that is configured to reset the MCU.

```
void reset_example (void)
{
 if (1U == R_SYSTEM->RSTSR0_b.LVD1RF)
    {
 /* The system is coming out of reset because Vcc crossed configured voltage
threshold. */
 /* Clear Voltage Monitor 1 Reset Detect Flag. */
       R_SYSTEM->RSTSR0_b.LVD1RF = 0;
    }
 fsp_err_t err = R_LVD_Open(&g_lvd_ctrl, &g_lvd_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
 /* Application Process */
 /* Application will reset when Vcc crosses the configured threshold. */
    }
}
```

## Data Structures

| | |
|---|---|
| struct | lvd_instance_ctrl_t |

## Data Structure Documentation

### ◆ lvd_instance_ctrl_t

| |
|---|
| struct lvd_instance_ctrl_t |
| LVD instance control structure |

## Function Documentation

### ◆ R_LVD_Open()

fsp_err_t R_LVD_Open ( lvd_ctrl_t *const *p_api_ctrl*, lvd_cfg_t const *const *p_cfg* )

Initializes a voltage monitor and detector according to the passed-in configuration structure.

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to the control structure for the driver instance |
| [in] | p_cfg | Pointer to the configuration structure for the driver instance |

***Note***

*Digital filter is not to be used with standby modes.*
*Startup time can take on the order of milliseconds for some configurations.*

Example:

```
fsp_err_t err = R_LVD_Open(&g_lvd_ctrl, &g_lvd_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful |
| FSP_ERR_ASSERTION | Requested configuration was invalid |
| FSP_ERR_ALREADY_OPEN | The instance was already opened |
| FSP_ERR_IN_USE | Another instance is already using the desired monitor |
| FSP_ERR_UNSUPPORTED | Digital filter was enabled on a device that does not support it |

### ◆ R_LVD_Close()

fsp_err_t R_LVD_Close ( lvd_ctrl_t *const *p_api_ctrl*)

Disables the LVD peripheral. Closes the driver instance.

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to the control block structure for the driver instance |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful |
| FSP_ERR_ASSERTION | An argument was NULL |
| FSP_ERR_NOT_OPEN | Driver is not open |

### ◆ R_LVD_StatusGet()

fsp_err_t R_LVD_StatusGet ( lvd_ctrl_t *const *p_api_ctrl*, lvd_status_t * *p_lvd_status* )

Get the current state of the monitor (threshold crossing detected, voltage currently above or below threshold).

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to the control structure for the driver instance |
| [out] | p_lvd_status | Pointer to status structure |

Example:

```
err = R_LVD_StatusGet(&g_lvd_ctrl, &status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful |
| FSP_ERR_ASSERTION | An argument was NULL |
| FSP_ERR_NOT_OPEN | Driver is not open |

### ◆ R_LVD_StatusClear()

fsp_err_t R_LVD_StatusClear ( lvd_ctrl_t *const *p_api_ctrl*)

Clears the latched status of the monitor.

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to the control structure for the driver instance |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful |
| FSP_ERR_ASSERTION | An argument was NULL |
| FSP_ERR_NOT_OPEN | Driver is not open |

### ◆ R_LVD_VersionGet()

fsp_err_t R_LVD_VersionGet ( fsp_version_t *const *p_version*)

Returns the LVD driver version based on compile time macros.

**Parameters**

| | | |
|---|---|---|
| [in] | p_version | Pointer to the version structure |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful |
| FSP_ERR_ASSERTION | p_version was NULL |

## 4.2.34 Operational Amplifier (r_opamp)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_OPAMP_Open (opamp_ctrl_t *const p_api_ctrl, opamp_cfg_t const *const p_cfg) |
| fsp_err_t | R_OPAMP_InfoGet (opamp_ctrl_t *const p_api_ctrl, opamp_info_t *const p_info) |

| | |
|---|---|
| fsp_err_t | R_OPAMP_Start (opamp_ctrl_t *const p_api_ctrl, uint32_t const channel_mask) |
| fsp_err_t | R_OPAMP_Stop (opamp_ctrl_t *const p_api_ctrl, uint32_t const channel_mask) |
| fsp_err_t | R_OPAMP_StatusGet (opamp_ctrl_t *const p_api_ctrl, opamp_status_t *const p_status) |
| fsp_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) |
| fsp_err_t | R_OPAMP_Close (opamp_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_OPAMP_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Driver for the OPAMP peripheral on RA MCUs. This module implements the OPAMP Interface.

# Overview

The OPAMP HAL module provides a high level API for signal amplification applications and supports the OPAMP peripheral available on RA MCUs.

### 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)

# Configuration

### Build Time Configurations for r_opamp

The following build time configurations are defined in fsp_cfg/r_opamp_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Analog > Operational Amplifier Driver on r_opamp

This module can be added to the Stacks tab via New Stack > Driver > Analog > Operational

Amplifier Driver on r_opamp:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_opamp0 | 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 OPAMPs 1 and 3 if configured for AGT Start<br>• 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<br>• AGT1 Compare Match Starts all OPAMPs configured for AGT Start | AGT1 Compare Match Starts all OPAMPs configured for AGT Start | Configure which AGT channel event triggers which 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<br>• Middle Speed<br>• High Speed | High Speed | Configure the op-amp based on power or speed requirements. This setting affects the minimum required stabilization time. Middle speed is not available for all MCUs. |
| Trigger Channel 0 | • Software Start Software Stop<br>• AGT Start Software Stop<br>• AGT Start ADC Stop | 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. |
| Trigger Channel 1 | • Software Start | Software Start | Select the event |

| | | | |
|---|---|---|---|
| | Software Stop<br>• AGT Start Software Stop<br>• AGT Start ADC Stop | Software Stop | 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<br>• AGT Start Software Stop<br>• AGT Start ADC Stop | 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<br>• AGT Start Software Stop<br>• AGT Start ADC Stop | 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. |
| OPAMP AMPOS0 | MCU Specific Options | | Select output to connect to AMP0O pin |
| OPAMP AMPPS0 | MCU Specific Options | | Select input to connect to AMP0+ pin |
| OPAMP AMPMS0 | MCU Specific Options | | Select input to connect to AMP0- pin |
| OPAMP AMPPS1 | MCU Specific Options | | Select input to connect to AMP1+ pin |
| OPAMP AMPMS1 | MCU Specific Options | | Select input to connect to AMP1- pin |
| OPAMP AMPPS2 | MCU Specific Options | | Select input to connect to AMP2+ pin |
| OPAMP AMPMS2 | MCU Specific Options | | Select input to connect to AMP2- pin |

### Clock Configuration

The OPAMP runs on PCLKB.

### Pin Configuration

To use the OPAMP HAL module, the port pins for the channels receiving the analog input must be set as inputs on the **Pins** tab of the RA Configuration editor.

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

# Usage Notes

### Trimming the OPAMP

- On MCUs that support trimming, the op-amp trim register is set to the factory default after the Open API is called.
- This function allows the application to trim the operational amplifier to a user setting, which overwrites the factory default trim values.
- Supported on selected 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 the trim API with the 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.

### Examples

### Basic Example

This is a basic example of minimal use of the R_OPAMP in an application. The example demonstrates configuring OPAMP channel 0 for high speed mode, starting the OPAMP and reading the status of the OPAMP channel running. It also verifies that the stabilization wait time is the expected time for selected power mode

```
#define OPAMP_EXAMPLE_CHANNEL (0U)
void basic_example (void)
{
 fsp_err_t err;
 /* Initialize the OPAMP module. */
    err = R_OPAMP_Open(&g_opamp_ctrl, &g_opamp_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Start the OPAMP module. */
    err = R_OPAMP_Start(&g_opamp_ctrl, 1 << OPAMP_EXAMPLE_CHANNEL);
    handle_error(err);
 /* Look up the required stabilization wait time. */
 opamp_info_t info;
    err = R_OPAMP_InfoGet(&g_opamp_ctrl, &info);
    handle_error(err);
 /* Wait for the OPAMP to stabilize. */
 R_BSP_SoftwareDelay(info.min_stabilization_wait_us, BSP_DELAY_UNITS_MICROSECONDS);
}
```

### Trim Example

This example demonstrates the typical trimming procedure for opamp channel 0 using R_OPAMP_Trim() API.

```
#ifndef OPAMP_EXAMPLE_CHANNEL
 #define OPAMP_EXAMPLE_CHANNEL (0U)
#endif
#ifndef OPAMP_EXAMPLE_ADC_CHANNEL
 #define OPAMP_EXAMPLE_ADC_CHANNEL (ADC_CHANNEL_2)
#endif
#define ADC_SCAN_END_DELAY (100U)
#define OPAMP_TRIM_LOOP_COUNT (5)
```

```
#define ADC_SCAN_END_MAX_TIMEOUT (0xFFFF)
uint32_t          g_callback_event_counter = 0;
opamp_trim_args_t trim_args_ch =
{
    .channel = OPAMP_EXAMPLE_CHANNEL,
    .input   = OPAMP_TRIM_INPUT_PCH
};
/* This callback is called when ADC Scan Complete event is generated. */
void adc_callback (adc_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
    g_callback_event_counter++;
}
void trimming_example (void)
{
 fsp_err_t err;
 /* On RA2A1, configure negative feedback and put DAC12 signal on AMP0+ Pin. */
    g_opamp_cfg_extend.plus_input_select_opamp0 = OPAMP_PLUS_INPUT_AMPPS7;
    g_opamp_cfg_extend.minus_input_select_opamp0 = OPAMP_MINUS_INPUT_AMPMS7;
 /* Initialize the OPAMP module. */
    err = R_OPAMP_Open(&g_opamp_ctrl, &g_opamp_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Start the OPAMP module. */
    err = R_OPAMP_Start(&g_opamp_ctrl, 1 << OPAMP_EXAMPLE_CHANNEL);
    handle_error(err);
 /* Look up the required stabilization wait time. */
 opamp_info_t info;
    err = R_OPAMP_InfoGet(&g_opamp_ctrl, &info);
    handle_error(err);
 /* Wait for the OPAMP to stabilize. */
 R_BSP_SoftwareDelay(info.min_stabilization_wait_us, BSP_DELAY_UNITS_MICROSECONDS);
 /* Call trim() for the Pch (+) side input */
    trim_procedure(&trim_args_ch);
```

```
    handle_error(err);
    trim_args_ch.input = OPAMP_TRIM_INPUT_NCH;
 /* Call trim() for the Nch (-) side input */
    trim_procedure(&trim_args_ch);
}
void trim_procedure (opamp_trim_args_t * trim_args)
{
 fsp_err_t err;
 /* Call trim() for the selected channel and input with command OPAMP_TRIM_CMD_START.
*/
    err = R_OPAMP_Trim(&g_opamp_ctrl, OPAMP_TRIM_CMD_START, trim_args);
    handle_error(err);
 /* Measure the fixed voltage connected to the channel input using the SAR ADC and
save the value
  * (referred to as result_a later in this procedure). */
 /* Reset the ADC callback counter */
    g_callback_event_counter = 0;
    err = R_ADC_ScanStart(&g_adc_ctrl);
    handle_error(err);
 /* Wait for ADC scan complete flag */
    uint32_t timeout = ADC_SCAN_END_MAX_TIMEOUT;
 while (g_callback_event_counter == 0 && timeout != 0)
    {
       timeout--;
    }
 if (0 == timeout)
    {
       err = FSP_ERR_TIMEOUT;
      handle_error(err);
    }
    uint16_t result_a;
    err = R_ADC_Read(&g_adc_ctrl, OPAMP_EXAMPLE_ADC_CHANNEL, &result_a);
    handle_error(err);
 /* Iterate over the following loop 5 times: */
```

```
 /* Call trim() with command OPAMP_TRIM_CMD_NEXT_STEP for the selected channel and
given input. */
    uint8_t count = OPAMP_TRIM_LOOP_COUNT;
 while (count > 0)
    {
       count--;
       err = R_OPAMP_Trim(&g_opamp_ctrl, OPAMP_TRIM_CMD_NEXT_STEP, trim_args);
      handle_error(err);
 /* Reset the ADC callback counter */
      g_callback_event_counter = 0;
 /* Read converted value after trim completes. */
      err = R_ADC_ScanStart(&g_adc_ctrl);
      handle_error(err);
 /* Wait for ADC scan complete flag */
       timeout = ADC_SCAN_END_MAX_TIMEOUT;
 while (g_callback_event_counter == 0 && timeout != 0)
      {
         timeout--;
      }
 if (0 == timeout)
      {
         err = FSP_ERR_TIMEOUT;
      handle_error(err);
      }
       uint16_t result_b;
       err = R_ADC_Read(&g_adc_ctrl, OPAMP_EXAMPLE_ADC_CHANNEL, &result_b);
      handle_error(err);
 /* Measure the op-amp output using the SAR ADC (referred to as result_b in the next
step). */
 /* If result_a <= result_b, call trim() for the selected channel and input with
command OPAMP_TRIM_CMD_CLEAR_BIT. */
 if (result_a <= result_b)
      {
            err = R_OPAMP_Trim(&g_opamp_ctrl, OPAMP_TRIM_CMD_CLEAR_BIT, trim_args);
```

```
        handle_error(err);
        }
    }
}
```

### Data Structures

| | |
|---|---|
| struct | opamp_extended_cfg_t |
| struct | opamp_instance_ctrl_t |

### Macros

| | |
|---|---|
| #define | OPAMP_CODE_VERSION_MAJOR |

### Enumerations

| | |
|---|---|
| enum | opamp_trigger_t |
| enum | opamp_agt_link_t |
| enum | opamp_mode_t |
| enum | opamp_plus_input_t |
| enum | opamp_minus_input_t |
| enum | opamp_output_t |

### Variables

| | |
|---|---|
| const opamp_api_t | g_opamp_on_opamp |

## Data Structure Documentation

### ◆ opamp_extended_cfg_t

| struct opamp_extended_cfg_t | | |
|---|---|---|
| OPAMP configuration extension. This extension is required and must be provided in opamp_cfg_t::p_extend. | | |
| Data Fields | | |
| 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. |
| opamp_plus_input_t | plus_input_select_opamp0 | OPAMP0+ connection. |
| opamp_minus_input_t | minus_input_select_opamp0 | OPAMP0- connection. |
| opamp_output_t | output_select_opamp0 | OPAMP0O connection. |
| opamp_plus_input_t | plus_input_select_opamp1 | OPAMP1+ connection. |
| opamp_minus_input_t | minus_input_select_opamp1 | OPAMP1- connection. |
| opamp_plus_input_t | plus_input_select_opamp2 | OPAMP2+ connection. |
| opamp_minus_input_t | minus_input_select_opamp2 | OPAMP2- connection. |

### ◆ opamp_instance_ctrl_t

| |
|---|
| struct opamp_instance_ctrl_t |
| OPAMP instance control block. DO NOT INITIALIZE. Initialized in opamp_api_t::open(). |

## Macro Definition Documentation

### ◆ OPAMP_CODE_VERSION_MAJOR

| |
|---|
| #define OPAMP_CODE_VERSION_MAJOR |
| Version of code that implements the API defined in this file |

## Enumeration Type Documentation

### ◆ opamp_trigger_t

| enum opamp_trigger_t | |
|---|---|
| Start and stop trigger for the op-amp. | |
| Enumerator | |
| 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. |

### ◆ opamp_agt_link_t

| enum opamp_agt_link_t | |
|---|---|
| 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. | |
| Enumerator | |
| 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. |

### ◆ opamp_mode_t

| enum opamp_mode_t | |
|---|---|
| Op-amp mode. | |
| Enumerator | |
| 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. |

### ◆ opamp_plus_input_t

| enum opamp_plus_input_t | |
|---|---|
| Options to connect AMPnPS pins. | |
| Enumerator | |
| OPAMP_PLUS_INPUT_NONE | No Connection. |
| OPAMP_PLUS_INPUT_AMPPS0 | Set AMPPS0. See hardware manual for channel specific options. |
| OPAMP_PLUS_INPUT_AMPPS1 | Set AMPPS1. See hardware manual for channel specific options. |
| OPAMP_PLUS_INPUT_AMPPS2 | Set AMPPS2. See hardware manual for channel specific options. |
| OPAMP_PLUS_INPUT_AMPPS3 | Set AMPPS3. See hardware manual for channel specific options. |
| OPAMP_PLUS_INPUT_AMPPS7 | Set AMPPS7. See hardware manual for channel specific options. |

### ◆ opamp_minus_input_t

| enum opamp_minus_input_t | |
|---|---|
| Options to connect AMPnMS pins. | |
| Enumerator | |
| OPAMP_MINUS_INPUT_NONE | No Connection. |
| OPAMP_MINUS_INPUT_AMPMS0 | Set AMPMS0. See hardware manual for channel specific options. |
| OPAMP_MINUS_INPUT_AMPMS1 | Set AMPMS1. See hardware manual for channel specific options. |
| OPAMP_MINUS_INPUT_AMPMS2 | Set AMPMS2. See hardware manual for channel specific options. |
| OPAMP_MINUS_INPUT_AMPMS3 | Set AMPMS3. See hardware manual for channel specific options. |
| OPAMP_MINUS_INPUT_AMPMS4 | Set AMPMS4. See hardware manual for channel specific options. |
| OPAMP_MINUS_INPUT_AMPMS7 | Set AMPMS7. See hardware manual for channel specific options. |

### ◆ opamp_output_t

| enum opamp_output_t | |
|---|---|
| Options to connect AMP0OS pin. | |
| Enumerator | |
| OPAMP_OUTPUT_NONE | No Connection. |
| OPAMP_OUTPUT_AMPOS0 | Set AMPOS0. See hardware manual for channel specific options. |
| OPAMP_OUTPUT_AMPOS1 | Set AMPOS1. See hardware manual for channel specific options. |
| OPAMP_OUTPUT_AMPOS2 | Set AMPOS2. See hardware manual for channel specific options. |
| OPAMP_OUTPUT_AMPOS3 | Set AMPOS3. See hardware manual for channel specific options. |

## Function Documentation

### ◆ R_OPAMP_Open()

fsp_err_t R_OPAMP_Open ( opamp_ctrl_t *const *p_api_ctrl*, opamp_cfg_t const *const *p_cfg* )

Applies power to the OPAMP and initializes the hardware based on the user configuration. Implements opamp_api_t::open.

The op-amp is not operational until the opamp_api_t::start is called. If the op-amp is configured to start after AGT compare match, the op-amp is not operational until opamp_api_t::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 opamp_api_t::open and before opamp_api_t::start.

Example:

```
/* Initialize the OPAMP module. */

   err = R_OPAMP_Open(&g_opamp_ctrl, &g_opamp_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration successful. |
| FSP_ERR_ASSERTION | An input pointer is NULL. |
| FSP_ERR_ALREADY_OPEN | Control block is already opened. |
| FSP_ERR_INVALID_ARGUMENT | An attempt to configure OPAMP in middle speed mode on MCU that does not support middle speed mode. |

### ◆ R_OPAMP_InfoGet()

fsp_err_t R_OPAMP_InfoGet ( opamp_ctrl_t *const *p_api_ctrl*, opamp_info_t *const *p_info* )

Provides the minimum stabilization wait time in microseconds. Implements opamp_api_t::infoGet.

- Example:

```
/* Look up the required stabilization wait time. */
opamp_info_t info;
   err = R_OPAMP_InfoGet(&g_opamp_ctrl, &info);
   handle_error(err);
/* Wait for the OPAMP to stabilize. */
R_BSP_SoftwareDelay(info.min_stabilization_wait_us,
BSP_DELAY_UNITS_MICROSECONDS);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | information on opamp_power_mode stored in p_info. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_OPAMP_Start()

fsp_err_t R_OPAMP_Start ( opamp_ctrl_t *const *p_api_ctrl*, uint32_t const *channel_mask* )

If the OPAMP is configured for hardware triggers, enables hardware triggers. Otherwise, starts the op-amp. Implements opamp_api_t::start.

Some MCUs have switches that must be set before starting the op-amp. These switches must be set in the application code after opamp_api_t::open and before opamp_api_t::start.

Example:

```
/* Start the OPAMP module. */
   err = R_OPAMP_Start(&g_opamp_ctrl, 1 << OPAMP_EXAMPLE_CHANNEL);
   handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Op-amp started or hardware triggers enabled successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |
| FSP_ERR_INVALID_ARGUMENT | channel_mask includes a channel that does not exist on this MCU. |

### ◆ R_OPAMP_Stop()

fsp_err_t R_OPAMP_Stop ( opamp_ctrl_t *const *p_api_ctrl*, uint32_t const *channel_mask* )

Stops the op-amp. If the OPAMP is configured for hardware triggers, disables hardware triggers. Implements opamp_api_t::stop.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Op-amp stopped or hardware triggers disabled successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |
| FSP_ERR_INVALID_ARGUMENT | channel_mask includes a channel that does not exist on this MCU. |

### ◆ R_OPAMP_StatusGet()

fsp_err_t R_OPAMP_StatusGet ( opamp_ctrl_t *const *p_api_ctrl*, opamp_status_t *const *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. Implements opamp_api_t::statusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operating status of each op-amp provided in p_status. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_OPAMP_Trim()

fsp_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* )

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 opamp_api_t::trim.

Reference: Section 37.9 "User Offset Trimming" RA2A1 hardware manual R01UM0008EU0130. 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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Conversion result in p_data. |
| FSP_ERR_UNSUPPORTED | Trimming is not supported on this MCU. |
| FSP_ERR_INVALID_STATE | The command is not valid in the current state of the trim state machine. |
| FSP_ERR_INVALID_ARGUMENT | The requested channel is not operating or the trim procedure is not in progress for this channel/input combination. |
| FSP_ERR_INVALID_MODE | Trim is not allowed in low power mode. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_OPAMP_Close()

fsp_err_t R_OPAMP_Close ( opamp_ctrl_t *const *p_api_ctrl*)

Stops the op-amps. Implements opamp_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Instance control block closed successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_OPAMP_VersionGet()

fsp_err_t R_OPAMP_VersionGet ( fsp_version_t *const *p_version*)

Gets the API and code version. Implements opamp_api_t::versionGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version information available in p_version. |
| FSP_ERR_ASSERTION | The parameter p_version is NULL. |

## Variable Documentation

### ◆ g_opamp_on_opamp

| const opamp_api_t g_opamp_on_opamp |
|---|
| OPAMP Implementation of OPAMP interface. |

## 4.2.35 Port Output Enable for GPT (r_poeg)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | R_POEG_Open (poeg_ctrl_t *const p_ctrl, poeg_cfg_t const *const p_cfg) |
| fsp_err_t | R_POEG_StatusGet (poeg_ctrl_t *const p_ctrl, poeg_status_t *const p_status) |
| fsp_err_t | R_POEG_OutputDisable (poeg_ctrl_t *const p_ctrl) |
| fsp_err_t | R_POEG_Reset (poeg_ctrl_t *const p_ctrl) |
| fsp_err_t | R_POEG_Close (poeg_ctrl_t *const p_ctrl) |
| fsp_err_t | R_POEG_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Driver for the POEG peripheral on RA MCUs. This module implements the POEG Interface.

# Overview

The POEG module can be used to configure events to disable GPT GTIOC output pins.

### Features

The POEG module has the following features:

- Supports disabling GPT output pins based on GTETRG input pin level.
- Supports disabling GPT output pins based on comparator crossing events (configurable in the High-Speed Analog Comparator (r_acmphs) driver).
- Supports disabling GPT output pins when GTIOC pins are the same level (configurable in the General PWM Timer (r_gpt) driver).
- Supports disabling GPT output pins when main oscillator stop is detected.
- Supports disabling GPT output pins by software API.
- Supports notifying the application when GPT output pins are disabled by POEG.
- Supports resetting POEG status.

# Configuration

### Build Time Configurations for r_poeg

The following build time configurations are defined in fsp_cfg/r_poeg_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Timers > Port Output Enable for GPT on r_poeg

This module can be added to the Stacks tab via New Stack > Driver > Timers > Port Output Enable for GPT on r_poeg:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General > Name | Name must be a valid C symbol | g_poeg0 | Module name. |
| General > Channel | Must be a valid POEG channel | 0 | Specify the hardware channel. |
| General > Trigger | MCU Specific Options | | Select the trigger sources that will enable POEG. Software disable is always supported. This configuration can only be set once after reset. It cannot be modified after the initial setting. |
| Input > GTETRG Polarity | • Active High<br>• Active Low | Active High | Select the polarity of the GTETRG pin. Only applicable if GTETRG pin is selected under Trigger. |
| Input > GTETRG Noise Filter | • Disabled<br>• PCLKB/1<br>• PCLKB/8<br>• PCLKB/32<br>• PCLKB/128 | Disabled | Configure the noise filter for the GTETRG pin. Only applicable if GTETRG pin is selected under Trigger. |
| Interrupts > Callback | Name must be a valid C symbol | NULL | A user callback function can be specified here. If this callback function is provided, it will be called from the |

| | | |
|---|---|---|
| | | interrupt service routine (ISR) when GPT output pins are disabled by POEG. |
| Interrupts > Interrupt Priority | MCU Specific Options | Select the POEG interrupt priority. |

### Clock Configuration

The POEG clock is based on the PCLKB frequency.

### Pin Configuration

This module can use GTETRGA, GTETRGB, GTETRGC, or GTETRGD as an input signal to disable GPT output pins.

## Usage Notes

### POEG GTETRG Pin and Channel

The POEG channel number corresponds to the GTETRG input pin that can be used with the channel. GTETRGA must be used with POEG channel 0, GTETRGB must be used with POEG channel 1, etc.

### Limitations

The user should be aware of the following limitations when using POEG:

- The POEG trigger source can only be set once per channel. Modifying the POEG trigger source after it is set is not allowed by the hardware.
- The POEG cannot be disabled using this API. The interrupt is disabled in R_POEG_Close(), but the POEG will still disable the GPT output pins if a trigger is detected even if the module is closed.

## Examples

### POEG Basic Example

This is a basic example of minimal use of the POEG in an application.

```
void poeg_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the POEG. */
    err = R_POEG_Open(&g_poeg0_ctrl, &g_poeg0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
}
```

### POEG Callback Example

This is an example of a using the POEG callback to restore GPT output operation.

```
/* Example callback called when POEG disables GPT output pins. */
void poeg_callback (poeg_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
 /* (Optional) Determine the cause of the POEG event. */
 poeg_status_t status;
    (void) R_POEG_StatusGet(&g_poeg0_ctrl, &status);
 /* Correct the cause of the POEG event before resetting POEG. */
 /* Reset the POEG before exiting the callback. */
    (void) R_POEG_Reset(&g_poeg0_ctrl);
 /* Wait for the status to clear after reset before exiting the callback to ensure
the interrupt does not fire
  * again. */
 do
    {
      (void) R_POEG_StatusGet(&g_poeg0_ctrl, &status);
    } while (POEG_STATE_NO_DISABLE_REQUEST != status.state);
 /* Alternatively, if the POEG cannot be reset, disable the POEG interrupt to prevent
it from firing continuously.
  * Update the 0 in the macro below to match the POEG channel number. */
    NVIC_DisableIRQ(VECTOR_NUMBER_POEG0_EVENT);
}
```

### Data Structures

| | |
|---|---|
| struct | poeg_instance_ctrl_t |

### Data Structure Documentation

#### ◆ poeg_instance_ctrl_t

| |
|---|
| struct poeg_instance_ctrl_t |
| Channel control block. DO NOT INITIALIZE. Initialization occurs when poeg_api_t::open is called. |

### Function Documentation

### ◆ R_POEG_Open()

fsp_err_t R_POEG_Open ( poeg_ctrl_t *const *p_ctrl*, poeg_cfg_t const *const *p_cfg* )

Initializes the POEG module and applies configurations. Implements poeg_api_t::open.

***Note***

*The poeg_cfg_t::trigger setting can only be configured once after reset. Reopening with a different trigger configuration is not possible.*

Example:

```
/* Initializes the POEG. */

   err = R_POEG_Open(&g_poeg0_ctrl, &g_poeg0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful. |
| FSP_ERR_ASSERTION | A required input pointer is NULL or the source divider is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_IRQ_BSP_DISABLED | poeg_cfg_t::p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use callback. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel requested in the p_cfg parameter is not available on this device. |

### ◆ R_POEG_StatusGet()

fsp_err_t R_POEG_StatusGet ( poeg_ctrl_t *const *p_ctrl*, poeg_status_t *const *p_status* )

Get current POEG status and store it in provided pointer p_status. Implements poeg_api_t::statusGet.

Example:

```
/* (Optional) Determine the cause of the POEG event. */

poeg_status_t status;

   (void) R_POEG_StatusGet(&g_poeg0_ctrl, &status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Current POEG state stored successfully. |
| FSP_ERR_ASSERTION | p_ctrl or p_status was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_POEG_OutputDisable()

fsp_err_t R_POEG_OutputDisable ( poeg_ctrl_t *const *p_ctrl*)

Disables GPT output pins. Implements poeg_api_t::outputDisable.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | GPT output pins successfully disabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_POEG_Reset()

fsp_err_t R_POEG_Reset ( poeg_ctrl_t *const *p_ctrl*)

Resets status flags. Implements poeg_api_t::reset.

***Note***

*Status flags are only reset if the original POEG trigger is resolved. Check the status using R_POEG_StatusGet after calling this function to verify the status is cleared.*

Example:

```
/* Correct the cause of the POEG event before resetting POEG. */
/* Reset the POEG before exiting the callback. */
   (void) R_POEG_Reset(&g_poeg0_ctrl);
/* Wait for the status to clear after reset before exiting the callback to ensure
the interrupt does not fire
 * again. */
do
   {
     (void) R_POEG_StatusGet(&g_poeg0_ctrl, &status);
   } while (POEG_STATE_NO_DISABLE_REQUEST != status.state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function attempted to clear status flags. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_POEG_Close()

fsp_err_t R_POEG_Close ( poeg_ctrl_t *const *p_ctrl*)

Disables POEG interrupt. Implements poeg_api_t::close.

***Note***

*This function does not disable the POEG.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_POEG_VersionGet()

fsp_err_t R_POEG_VersionGet ( fsp_version_t *const *p_version*)

Sets driver version based on compile time macros. Implements poeg_api_t::versionGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version stored in p_version. |
| FSP_ERR_ASSERTION | p_version was NULL. |

## 4.2.36 Quad Serial Peripheral Interface Flash (r_qspi)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_QSPI_Open (spi_flash_ctrl_t *p_ctrl, spi_flash_cfg_t const *const p_cfg) |
| fsp_err_t | R_QSPI_Close (spi_flash_ctrl_t *p_ctrl) |
| fsp_err_t | R_QSPI_DirectWrite (spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, uint32_t const bytes, bool const read_after_write) |
| fsp_err_t | R_QSPI_DirectRead (spi_flash_ctrl_t *p_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_QSPI_SpiProtocolSet (spi_flash_ctrl_t *p_ctrl, spi_flash_protocol_t |

| | |
|---|---|
| | spi_protocol) |
| fsp_err_t | R_QSPI_XipEnter (spi_flash_ctrl_t *p_ctrl) |
| fsp_err_t | R_QSPI_XipExit (spi_flash_ctrl_t *p_ctrl) |
| fsp_err_t | R_QSPI_Write (spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint32_t byte_count) |
| fsp_err_t | R_QSPI_Erase (spi_flash_ctrl_t *p_ctrl, uint8_t *const p_device_address, uint32_t byte_count) |
| fsp_err_t | R_QSPI_StatusGet (spi_flash_ctrl_t *p_ctrl, spi_flash_status_t *const p_status) |
| fsp_err_t | R_QSPI_BankSet (spi_flash_ctrl_t *p_ctrl, uint32_t bank) |
| fsp_err_t | R_QSPI_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Driver for the QSPI peripheral on RA MCUs. This module implements the SPI Flash Interface.

# Overview

### Features

The QSPI driver has the following key features:

- Memory mapped read access to the QSPI flash
- Programming the QSPI flash device
- Erasing the QSPI flash device
- Sending device specific commands and reading back responses
- Entering and exiting QPI mode
- Entering and exiting XIP mode
- 3 or 4 byte addressing

# Configuration

### Build Time Configurations for r_qspi

The following build time configurations are defined in driver/r_qspi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking Enable | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Support Multiple Line | • Enabled | Disabled | If selected code for |

| | | | |
|---|---|---|---|
| Program in Extended SPI Mode | • Disabled | | programming on multiple lines in extended SPI mode is included in the build. |

**Configurations for Driver > Storage > QSPI Driver on r_qspi**

This module can be added to the Stacks tab via New Stack > Driver > Storage > QSPI Driver on r_qspi:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General > Name | Name must be a valid C symbol | g_qspi0 | Module name. |
| General > SPI Protocol | • Extended SPI<br>• QPI | Extended SPI | Select the initial SPI protocol. SPI protocol can be changed in R_QSPI_Direct(). |
| General > Address Bytes | • 3<br>• 4<br>• 4 with 4-byte read code | 3 | Select the number of address bytes. Selecting '4 with 4-byte read code' converts the default read code determined in Read Mode to the 4-byte version. If 4-byte mode is selected without using 4-byte commands, the application must issue the EN4B command using R_QSPI_Direct(). |
| General > Read Mode | • Standard<br>• Fast Read<br>• Fast Read Dual Output<br>• Fast Read Dual I/O<br>• Fast Read Quad Output<br>• Fast Read Quad I/O | Fast Read Quad I/O | Select the read mode for memory mapped access. |
| General > Dummy Clocks for Fast Read | Refer to the RA Configuration tool for available options. | Default | Select the number of dummy clocks for fast read operations. Default is 6 clocks for Fast Read Quad I/O, 4 clocks for Fast Read Dual I/O, and 8 clocks for other fast read instructions including Fast Read Quad |

| | | | |
|---|---|---|---|
| | | | Output, Fast Read Dual Output, and Fast Read |
| General > Page Size Bytes | Must be an integer greater than 0 | 256 | The maximum number of bytes allowed for a single write. |
| Command Definitions > Page Program Command | Must be an 8-bit QSPI command | 0x02 | The command to program a page. If 'Support Multiple Line Program in Extended SPI Mode' is Enabled, this command must use the same number of data lines as the selected read mode. |
| Command Definitions > Page Program Address Lines | • 1<br>• 2<br>• 4 | 1 | Select the number of lines to use for the address bytes during write operations. This can be determined by referencing the datasheet for the external QSPI. It should either be 1 or match the number of data lines used for memory mapped fast read operations. |
| Command Definitions > Write Enable Command | Must be an 8-bit QSPI command | 0x06 | The command to enable write. |
| Command Definitions > Status Command | Must be an 8-bit QSPI command | 0x05 | The command to query the status of a write or erase command. |
| Command Definitions > Write Status Bit | Must be an integer between 0 and 7 | 0 | Which bit contains the write in progress status returned from the Write Status Command. |
| Command Definitions > Sector Erase Command | Must be an 8-bit QSPI command | 0x20 | The command to erase a sector. Set Sector Erase Size to 0 if unused. |
| Command Definitions > Sector Erase Size | Must be an integer greater than or equal to 0 | 4096 | The sector erase size. Set Sector Erase Size to 0 if Sector Erase is not supported. |
| Command Definitions > Block Erase Command | Must be an 8-bit QSPI command | 0xD8 | The command to erase a block. Set Block Erase Size to 0 if |

| | | | |
|---|---|---|---|
| | | | unused. |
| Command Definitions > Block Erase Size | Must be an integer greater than or equal to 0 | 65536 | The block erase size. Set Block Erase Size to 0 if Block Erase is not supported. |
| Command Definitions > Block Erase 32KB Command | Must be an 8-bit QSPI command | 0x52 | The command to erase a 32KB block. Set Block Erase Size to 0 if unused. |
| Command Definitions > Block Erase 32KB Size | Must be an integer greater than or equal to 0 | 32768 | The block erase 32KB size. Set Block Erase 32KB Size to 0 if Block Erase 32KB is not supported. |
| Command Definitions > Chip Erase Command | Must be an 8-bit QSPI command | 0xC7 | The command to erase the entire chip. Set Chip Erase Command to 0 if unused. |
| Command Definitions > XIP Enter M7-M0 | Must be an 8-bit QSPI command | 0x20 | How to set M7-M0 to enter XIP mode. |
| Command Definitions > XIP Exit M7-M0 | Must be an 8-bit QSPI command | 0xFF | How to set M7-M0 exit XIP mode. |
| Bus Timing > QSPKCLK Divisor | Refer to the RA Configuration tool for available options. | 2 | Select the divisor to apply to PCLK to get QSPCLK. |
| Bus Timing > Minimum QSSL Deselect Cycles | Refer to the RA Configuration tool for available options. | 4 QSPCLK | Define the minimum number of QSPCLK cycles for QSSL to remain high beween operations. |

### Clock Configuration

The QSPI clock is derived from PCLKA.

### Pin Configuration

The following pins are available to connect to an external QSPI device:

- QSPCLK: QSPI clock output
- QSSL: QSPI slave select
- QIO0: Data 0 I/O
- QIO1: Data 1 I/O
- QIO2: Data 2 I/O
- QIO3: Data 3 I/O

*Note*

*It is recommended to configure the pins with IOPORT_CFG_DRIVE_HIGH.*

# Usage Notes

### QSPI Memory Mapped Access

After R_QSPI_Open() completes successfully, the QSPI flash device contents are mapped to address 0x60000000 and can be read like on-chip flash.

### Limitations

Developers should be aware of the following limitations when using the QSPI driver:

- Only P305-P310 are currently supported by the J-Link driver to flash the QSPI.
- The default J-Link downloader requires the device to be in extended SPI mode (not QPI mode).

# Examples

### Basic Example

This is a basic example of minimal use of the QSPI in an application.

```
#define QSPI_EXAMPLE_DATA_LENGTH (1024)
uint8_t g_dest[QSPI_EXAMPLE_DATA_LENGTH];
/* Place data in the .qspi_flash section to flash it during programming. */
const uint8_t g_src[QSPI_EXAMPLE_DATA_LENGTH] BSP_PLACE_IN_SECTION(".qspi_flash") =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
/* Place code in the .code_in_qspi section to flash it during programming. */
void r_qspi_example_function(void) BSP_PLACE_IN_SECTION(".code_in_qspi")
__attribute__((noinline));
void r_qspi_example_function (void)
{
 /* Add code here. */
}
void r_qspi_basic_example (void)
{
 /* Open the QSPI instance. */
 fsp_err_t err = R_QSPI_Open(&g_qspi0_ctrl, &g_qspi0_cfg);
    handle_error(err);
 /* (Optional) Send device specific initialization commands. */
    r_qspi_example_init();
 /* After R_QSPI_Open() and any required device specific intiialization, data can be
```

```
read directly from the QSPI flash. */
    memcpy(&g_dest[0], &g_src[0], QSPI_EXAMPLE_DATA_LENGTH);
 /* After R_QSPI_Open() and any required device specific intiialization, functions in
the QSPI flash can be called. */
    r_qspi_example_function();
}
```

### Initialization Command Structure Example

This is an example of the types of commands that can be used to initialize the QSPI.

```
#define QSPI_COMMAND_WRITE_ENABLE (0x06U)
#define QSPI_COMMAND_WRITE_STATUS_REGISTER (0x01U)
#define QSPI_COMMAND_ENTER_QPI_MODE (0x38U)
#define QSPI_EXAMPLE_STATUS_REGISTER_1 (0x40)
#define QSPI_EXAMPLE_STATUS_REGISTER_2 (0x00)
static void r_qspi_example_init (void)
{
 /* Write status registers */
 /* Write one byte to enable writing to the status register, then deassert QSSL. */
    uint8_t   data[4];
 fsp_err_t err;
    data[0] = QSPI_COMMAND_WRITE_ENABLE;
    err    = R_QSPI_DirectWrite(&g_qspi0_ctrl, &data[0], 1, false);
    handle_error(err);
 /* Write 3 bytes, including the write status register command followed by values for
both status registers. In the
  * status registers, set QE to 1 and other bits to their default setting. After all
data is written, deassert the
  * QSSL line. */
    data[0] = QSPI_COMMAND_WRITE_STATUS_REGISTER;
    data[1] = QSPI_EXAMPLE_STATUS_REGISTER_1;
    data[2] = QSPI_EXAMPLE_STATUS_REGISTER_2;
    err    = R_QSPI_DirectWrite(&g_qspi0_ctrl, &data[0], 3, false);
    handle_error(err);
```

```
 /* Wait for status register to update. */
 spi_flash_status_t status;
 do
    {
       (void) R_QSPI_StatusGet(&g_qspi0_ctrl, &status);
    } while (true == status.write_in_progress);
 /* Write one byte to enter QSPI mode, then deassert QSSL. After entering QPI mode on
the device, change the SPI
  * protocol to QPI mode on the MCU peripheral. */
    data[0] = QSPI_COMMAND_ENTER_QPI_MODE;
    err    = R_QSPI_DirectWrite(&g_qspi0_ctrl, &data[0], 1, false);
    handle_error(err);
    (void) R_QSPI_SpiProtocolSet(&g_qspi0_ctrl, SPI_FLASH_PROTOCOL_QPI);
}
```

### Reading Status Register Example (R_QSPI_DirectWrite, R_QSPI_DirectRead)

This is an example of using R_QSPI_DirectWrite followed by R_QSPI_DirectRead to send the read status register command and read back the status register from the device.

```
#define QSPI_COMMAND_READ_STATUS_REGISTER (0x05U)
void r_qspi_direct_example (void)
{
 /* Read a status register. */
 /* Write one byte to read the status register. Do not deassert QSSL. */
    uint8_t   data;
 fsp_err_t err;
    data = QSPI_COMMAND_READ_STATUS_REGISTER;
    err = R_QSPI_DirectWrite(&g_qspi0_ctrl, &data, 1, true);
    handle_error(err);
 /* Read one byte. After all data is read, deassert the QSSL line. */
    err = R_QSPI_DirectRead(&g_qspi0_ctrl, &data, 1);
    handle_error(err);
 /* Status register contents are available in variable 'data'. */
}
```

### Querying Device Size Example (R_QSPI_DirectWrite, R_QSPI_DirectRead)

This is an example of using R_QSPI_DirectWrite followed by R_QSPI_DirectRead to query the device size.

```
#define QSPI_EXAMPLE_COMMAND_READ_ID (0x9F)
#define QSPI_EXAMPLE_COMMAND_READ_SFDP (0x5A)
void r_qspi_size_example (void)
{
 /* Many QSPI devices support more than one way to query the device size. Consult the
datasheet for your
  * QSPI device to determine which of these methods are supported (if any). */
    uint32_t device_size_bytes;
 fsp_err_t err;
#ifdef QSPI_EXAMPLE_COMMAND_READ_ID
 /* This example shows how to get the device size by reading the manufacturer ID. */
    uint8_t data[4];
    data[0] = QSPI_EXAMPLE_COMMAND_READ_ID;
    err    = R_QSPI_DirectWrite(&g_qspi0_ctrl, &data[0], 1, true);
    handle_error(err);
 /* Read 3 bytes. The third byte often represents the size of the QSPI, where the
size of the QSPI = 2 ^ N. */
    err = R_QSPI_DirectRead(&g_qspi0_ctrl, &data[0], 3);
    handle_error(err);
    device_size_bytes = 1U << data[2];
 FSP_PARAMETER_NOT_USED(device_size_bytes);
#endif
#ifdef QSPI_EXAMPLE_COMMAND_READ_SFDP
 /* Read the JEDEC SFDP header to locate the JEDEC flash parameters table. Reference
JESD216 "Serial Flash
  * Discoverable Parameters (SFDP)". */
 /* Send the standard 0x5A command followed by 3 address bytes (SFDP header is at
address 0). */
    uint8_t buffer[16];
```

```
    memset(&buffer[0], 0, sizeof(buffer));
    buffer[0] = QSPI_EXAMPLE_COMMAND_READ_SFDP;
    err       = R_QSPI_DirectWrite(&g_qspi0_ctrl, &buffer[0], 4, true);
    handle_error(err);
 /* Read out 16 bytes (1 dummy byte followed by 15 data bytes). */
    err = R_QSPI_DirectRead(&g_qspi0_ctrl, &buffer[0], 16);
    handle_error(err);
 /* Read the JEDEC flash parameters to locate the memory size. */
 /* Send the standard 0x5A command followed by 3 address bytes (located in big endian
order at offset 0xC-0xE).
  * These bytes are accessed at 0xD-0xF because the first byte read is a dummy byte.
*/
    buffer[0] = QSPI_EXAMPLE_COMMAND_READ_SFDP;
    buffer[1] = buffer[0xF];
    buffer[2] = buffer[0xE];
    buffer[3] = buffer[0xD];
    err       = R_QSPI_DirectWrite(&g_qspi0_ctrl, &buffer[0], 4, true);
    handle_error(err);
 /* Read out 9 bytes (1 dummy byte followed by 8 data bytes). */
    err = R_QSPI_DirectRead(&g_qspi0_ctrl, &buffer[0], 9);
    handle_error(err);
 /* Read the memory density (located in big endian order at offset 0x4-0x7). These
bytes are accessed at 0x5-0x8
  * because the first byte read is a dummy byte. */
    uint32_t memory_density = (uint32_t) ((buffer[8] << 24) | (buffer[7] << 16) |
(buffer[6] << 8) | buffer[5]);
 if ((1U << 31) & memory_density)
    {
 /* For densities 4 gigabits and above, bit-31 is set to 1b. The field 30:0 defines
'N' where the density is
  * computed as 2^N bits (N must be >= 32). This code subtracts 3 from N to divide by
8 to get the size in
  * bytes instead of bits. */
       device_size_bytes = 1U << ((memory_density & ~(1U << 31)) - 3U);
```

```
    }
 else
    {
 /* For densities 2 gigabits or less, bit-31 is set to 0b. The field 30:0 defines the
size in bits. This
  * code divides the memory density by 8 to get the size in bytes instead of bits. */
       device_size_bytes = (memory_density / 8) + 1;
    }
 FSP_PARAMETER_NOT_USED(device_size_bytes);
#endif
}
```

## Data Structures

| | |
|---|---|
| struct | qspi_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | qspi_qssl_min_high_level_t |
| enum | qspi_qspclk_div_t |

## Data Structure Documentation

### ◆ qspi_instance_ctrl_t

| |
|---|
| struct qspi_instance_ctrl_t |
| Instance control block. DO NOT INITIALIZE. Initialization occurs when spi_flash_api_t::open is called |

## Enumeration Type Documentation

### ◆ qspi_qssl_min_high_level_t

| enum qspi_qssl_min_high_level_t | |
|---|---|
| Enumerator | |
| QSPI_QSSL_MIN_HIGH_LEVEL_1_QSPCLK | QSSL deselected for at least 1 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_2_QSPCLK | QSSL deselected for at least 2 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_3_QSPCLK | QSSL deselected for at least 3 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_4_QSPCLK | QSSL deselected for at least 4 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_5_QSPCLK | QSSL deselected for at least 5 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_6_QSPCLK | QSSL deselected for at least 6 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_7_QSPCLK | QSSL deselected for at least 7 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_8_QSPCLK | QSSL deselected for at least 8 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_9_QSPCLK | QSSL deselected for at least 9 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_10_QSPCLK | QSSL deselected for at least 10 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_11_QSPCLK | QSSL deselected for at least 11 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_12_QSPCLK | QSSL deselected for at least 12 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_13_QSPCLK | QSSL deselected for at least 13 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_14_QSPCLK | QSSL deselected for at least 14 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_15_QSPCLK | QSSL deselected for at least 15 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_16_QSPCLK | QSSL deselected for at least 16 QSPCLK. |

### ◆ qspi_qspclk_div_t

| enum qspi_qspclk_div_t | |
|---|---|
| Enumerator | |
| QSPI_QSPCLK_DIV_2 | QSPCLK = PCLK / 2. |
| QSPI_QSPCLK_DIV_3 | QSPCLK = PCLK / 3. |
| QSPI_QSPCLK_DIV_4 | QSPCLK = PCLK / 4. |
| QSPI_QSPCLK_DIV_5 | QSPCLK = PCLK / 5. |
| QSPI_QSPCLK_DIV_6 | QSPCLK = PCLK / 6. |
| QSPI_QSPCLK_DIV_7 | QSPCLK = PCLK / 7. |
| QSPI_QSPCLK_DIV_8 | QSPCLK = PCLK / 8. |
| QSPI_QSPCLK_DIV_9 | QSPCLK = PCLK / 9. |
| QSPI_QSPCLK_DIV_10 | QSPCLK = PCLK / 10. |
| QSPI_QSPCLK_DIV_11 | QSPCLK = PCLK / 11. |
| QSPI_QSPCLK_DIV_12 | QSPCLK = PCLK / 12. |
| QSPI_QSPCLK_DIV_13 | QSPCLK = PCLK / 13. |
| QSPI_QSPCLK_DIV_14 | QSPCLK = PCLK / 14. |
| QSPI_QSPCLK_DIV_15 | QSPCLK = PCLK / 15. |
| QSPI_QSPCLK_DIV_16 | QSPCLK = PCLK / 16. |
| QSPI_QSPCLK_DIV_17 | QSPCLK = PCLK / 17. |
| QSPI_QSPCLK_DIV_18 | QSPCLK = PCLK / 18. |
| QSPI_QSPCLK_DIV_19 | QSPCLK = PCLK / 19. |
| QSPI_QSPCLK_DIV_20 | QSPCLK = PCLK / 20. |
| QSPI_QSPCLK_DIV_22 | QSPCLK = PCLK / 22. |
| QSPI_QSPCLK_DIV_24 | QSPCLK = PCLK / 24. |
| QSPI_QSPCLK_DIV_26 | |

| | |
|---|---|
| | QSPCLK = PCLK / 26. |
| QSPI_QSPCLK_DIV_28 | QSPCLK = PCLK / 28. |
| QSPI_QSPCLK_DIV_30 | QSPCLK = PCLK / 30. |
| QSPI_QSPCLK_DIV_32 | QSPCLK = PCLK / 32. |
| QSPI_QSPCLK_DIV_34 | QSPCLK = PCLK / 34. |
| QSPI_QSPCLK_DIV_36 | QSPCLK = PCLK / 36. |
| QSPI_QSPCLK_DIV_38 | QSPCLK = PCLK / 38. |
| QSPI_QSPCLK_DIV_40 | QSPCLK = PCLK / 40. |
| QSPI_QSPCLK_DIV_42 | QSPCLK = PCLK / 42. |
| QSPI_QSPCLK_DIV_44 | QSPCLK = PCLK / 44. |
| QSPI_QSPCLK_DIV_46 | QSPCLK = PCLK / 46. |
| QSPI_QSPCLK_DIV_48 | QSPCLK = PCLK / 48. |

## Function Documentation

### ◆ R_QSPI_Open()

fsp_err_t R_QSPI_Open ( spi_flash_ctrl_t * *p_ctrl*, spi_flash_cfg_t const *const *p_cfg* )

Open the QSPI driver module. After the driver is open, the QSPI can be accessed like internal flash memory starting at address 0x60000000.

Implements spi_flash_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration was successful. |
| FSP_ERR_ASSERTION | The parameter p_instance_ctrl or p_cfg is NULL. |
| FSP_ERR_ALREADY_OPEN | Driver has already been opened with the same p_instance_ctrl. |

### ◆ R_QSPI_Close()

fsp_err_t R_QSPI_Close ( spi_flash_ctrl_t * *p_ctrl*)

Close the QSPI driver module.

Implements spi_flash_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration was successful. |
| FSP_ERR_ASSERTION | p_instance_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |

### ◆ R_QSPI_DirectWrite()

fsp_err_t R_QSPI_DirectWrite ( spi_flash_ctrl_t * *p_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes*, bool const *read_after_write* )

Writes raw data directly to the QSPI.

Implements spi_flash_api_t::directWrite.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The flash was programmed successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_INVALID_MODE | This function can't be called when XIP mode is enabled. |
| FSP_ERR_DEVICE_BUSY | The device is busy. |

### ◆ R_QSPI_DirectRead()

fsp_err_t R_QSPI_DirectRead ( spi_flash_ctrl_t * *p_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Reads raw data directly from the QSPI. This API can only be called after R_QSPI_DirectWrite with read_after_write set to true.

Implements spi_flash_api_t::directRead.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The flash was programmed successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_INVALID_MODE | This function must be called after R_QSPI_DirectWrite with read_after_write set to true. |

### ◆ R_QSPI_SpiProtocolSet()

fsp_err_t R_QSPI_SpiProtocolSet ( spi_flash_ctrl_t * *p_ctrl*, spi_flash_protocol_t *spi_protocol* )

Sets the SPI protocol.

Implements spi_flash_api_t::spiProtocolSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | SPI protocol updated on MCU peripheral. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |

### ◆ R_QSPI_XipEnter()

fsp_err_t R_QSPI_XipEnter ( spi_flash_ctrl_t * *p_ctrl*)

Enters XIP (execute in place) mode.

Implements spi_flash_api_t::xipEnter.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The flash was programmed successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |

### ◆ R_QSPI_XipExit()

fsp_err_t R_QSPI_XipExit ( spi_flash_ctrl_t * *p_ctrl*)

Exits XIP (execute in place) mode.

Implements spi_flash_api_t::xipExit.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The flash was programmed successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |

### ◆ R_QSPI_Write()

fsp_err_t R_QSPI_Write ( spi_flash_ctrl_t * *p_ctrl*, uint8_t const *const *p_src*, uint8_t *const *p_dest*, uint32_t *byte_count* )

Program a page of data to the flash.

Implements spi_flash_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The flash was programmed successfully. |
| FSP_ERR_ASSERTION | p_instance_ctrl, p_dest or p_src is NULL, or byte_count crosses a page boundary. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_INVALID_MODE | This function can't be called when XIP mode is enabled. |
| FSP_ERR_DEVICE_BUSY | The device is busy. |

### ◆ R_QSPI_Erase()

fsp_err_t R_QSPI_Erase ( spi_flash_ctrl_t * *p_ctrl*, uint8_t *const *p_device_address*, uint32_t *byte_count* )

Erase a block or sector of flash. The byte_count must exactly match one of the erase sizes defined in spi_flash_cfg_t. For chip erase, byte_count must be SPI_FLASH_ERASE_SIZE_CHIP_ERASE.

Implements spi_flash_api_t::erase.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The command to erase the flash was executed successfully. |
| FSP_ERR_ASSERTION | p_instance_ctrl or p_device_address is NULL, or byte_count doesn't match an erase size defined in spi_flash_cfg_t, or device is in XIP mode. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_INVALID_MODE | This function can't be called when XIP mode is enabled. |
| FSP_ERR_DEVICE_BUSY | The device is busy. |

### ◆ R_QSPI_StatusGet()

fsp_err_t R_QSPI_StatusGet ( spi_flash_ctrl_t * *p_ctrl*, spi_flash_status_t *const *p_status* )

Gets the write or erase status of the flash.

Implements spi_flash_api_t::statusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The write status is in p_status. |
| FSP_ERR_ASSERTION | p_instance_ctrl or p_status is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_INVALID_MODE | This function can't be called when XIP mode is enabled. |

### ◆ R_QSPI_BankSet()

fsp_err_t R_QSPI_BankSet ( spi_flash_ctrl_t * *p_ctrl*, uint32_t *bank* )

Selects the bank to access. A bank is a 64MB sliding access window into the QSPI device flash memory space. To access chip address 0x4000000, select bank 1, then read from internal flash address 0x60000000. To access chip address 0x8001000, select bank 2, then read from internal flash address 0x60001000.

This function is not required for memory devices less than or equal to 512 Mb (64MB).

Implements spi_flash_api_t::bankSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Bank successfully selected. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |

### ◆ R_QSPI_VersionGet()

fsp_err_t R_QSPI_VersionGet ( fsp_version_t *const *p_version*)

Get the driver version based on compile time macros.

Implements spi_flash_api_t::versionGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | p_version is NULL. |

## 4.2.37 Realtime Clock (r_rtc)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_RTC_Open (rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg) |
| fsp_err_t | R_RTC_Close (rtc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_RTC_CalendarTimeSet (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time) |

| | |
|---|---|
| fsp_err_t | R_RTC_CalendarTimeGet (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time) |
| fsp_err_t | R_RTC_CalendarAlarmSet (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm) |
| fsp_err_t | R_RTC_CalendarAlarmGet (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm) |
| fsp_err_t | R_RTC_PeriodicIrqRateSet (rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t const rate) |
| fsp_err_t | R_RTC_ErrorAdjustmentSet (rtc_ctrl_t *const p_ctrl, rtc_error_adjustment_cfg_t const *const err_adj_cfg) |
| fsp_err_t | R_RTC_InfoGet (rtc_ctrl_t *const p_ctrl, rtc_info_t *const p_rtc_info) |
| fsp_err_t | R_RTC_VersionGet (fsp_version_t *version) |

### Detailed Description

Driver for the RTC peripheral on RA MCUs. This module implements the RTC Interface.

# Overview

The RTC HAL module configures the RTC module and controls clock, calendar and alarm functions. A callback can be used to respond to the alarm and periodic interrupt.

### Features

- RTC time and date get and set.
- RTC time and date alarm get and set.
- RTC alarm and periodic event notification.

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 is used internally when reading time from the RTC calender to get accurant time readings.

*Note*

*See section "23.3.5 Reading 64-Hz Counter and Time" of the RA6M3 manual R01UH0886EJ0100 for more details.*

A user-defined callback function can be registered (in the rtc_api_t::open API call) and will be called from the interrupt service routine (ISR) for alarm and periodic interrupt. 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.

### Date and Time validation

"Parameter Checking" needs to be enabled 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 (Time Error Adjustment Function)

The time error adjustment function is used to correct errors, running fast or slow, in the time caused by variation in the precision of oscillation by the sub-clock oscillator. Because 32,768 cycles of the sub-clock oscillator constitute 1 second of operation when the sub-clock oscillator is selected, the clock runs fast if the sub-clock frequency is high and slow if the sub-clock frequency is low. The time error adjustment functions include:

- Automatic adjustment
- Adjustment by software

The error adjustment is reset every time RTC is reconfigured or time is set.

*Note*

*RTC driver configurations do not do error adjustment internally while initiliazing the driver. Application must make calls to the error adjustment api's for desired adjustment. See section 26.3.8 "Time Error Adjustment Function" of the RA6M3 manual R01UH0886EJ0100) for more details on this feature*

# Configuration

### Build Time Configurations for r_rtc

The following build time configurations are defined in fsp_cfg/r_rtc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Timers > RTC Driver on r_rtc

This module can be added to the Stacks tab via New Stack > Driver > Timers > RTC Driver on r_rtc:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rtc0 | Module name. |
| Clock Source | • Sub-Clock<br>• LOCO | LOCO | Select the RTC clock source. |
| Frequency Comparision Value (LOCO) | Value must be a positive integer between 7 and 511 | 255 | Frequency comparison value when using LOCO |

| | | | |
|---|---|---|---|
| Automatic Adjustment Mode | • Enabled<br>• Disabled | Enabled | Enable/ Disable the Error Adjustment mode |
| Automatic Adjustment Period | • 10 Seconds<br>• 1 Minute<br>• NONE | 10 Seconds | Select the Error Adjustment Period for Automatic Adjustment |
| Adjustment Type (Plus-Minus) | • NONE<br>• Addition<br>• Subtraction | NONE | Select the Error Adjustment type |
| Error Adjustment Value | Value must be a positive integer less than equal to 63 | 0 | Specify the Adjustment Value (the number of sub-clock cycles) from the prescaler |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |
| Alarm Interrupt Priority | MCU Specific Options | | Select the alarm interrupt priority. |
| Period Interrupt Priority | MCU Specific Options | | Select the period interrupt priority. |
| Carry Interrupt Priority | MCU Specific Options | | Select the carry interrupt priority. |

*Note*

*See 23.2.20 Frequency Register (RFRH/RFRL) of the RA6M3 manual R01UH0886EJ0100) for more details*

**Interrupt Configuration**

To activate interrupts for the RTC module, the desired interrupts must be enabled, The underlying implementation will be expected to handle any interrupts it can support and notify higher layers via callback.

**Clock Configuration**

The RTC HAL module can use the following clock sources:

- LOCO (Low Speed On-Chip Oscillator) with less accuracy
- Sub-clock oscillator with increased accuracy

The LOCO is the default selection during configuration.

**Pin Configuration**

This module does not use I/O pins.

# Usage Notes

### System Initialization

- RTC driver does not start the sub-clock. The application is responsible for ensuring required clocks are started and stable before accessing MCU peripheral registers.

Warning

The subclock can take seconds to stabilize. The RA startup code does not wait for subclock stabilization unless the subclock is the main clock source. When running AGT or RTC off the subclock, the application must ensure the subclock is stable before starting operation.

- Carry interrupt priority must be set to avoid incorrect time returned from calendarTimeGet API during roll-over.
- Even when only running in Periodic Interrupt mode R_RTC_CalendarTimeSet must be called successfully to start the RTC.

### Limitations

Developers should be aware of the following limitations when using the RTC: Below features are not supported by the driver

- Binary-count mode
- The R_RTC_CalendarTimeGet() cannot be used from an interrupt that has higher priority than the carry interrupt. Also, it must not be called with interrupts disabled globally, as this API internally uses carry interrupt for its processing. API may return incorrect time if this is done.

# Examples

### RTC Basic Example

This is a basic example of minimal use of the RTC in an application.

```
/* rtc_time_t is an alias for the C Standard time.h struct 'tm' */
rtc_time_t set_time =
{
    .tm_sec = 10,
    .tm_min = 11,
    .tm_hour = 12,
    .tm_mday = 6,
    .tm_wday = 3,
    .tm_mon = 11,
    .tm_year = YEARS_SINCE_1900,
};
rtc_time_t get_time;
void rtc_example ()
{
```

```
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the RTC module */
    err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Set the calendar time */
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time);
 /* Get the calendar time */
 R_RTC_CalendarTimeGet(&g_rtc0_ctrl, &get_time);
}
```

### RTC Periodic interrupt example

This is an example of periodic interrupt in RTC.

```
void rtc_periodic_irq_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the RTC module*/
    err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* R_RTC_CalendarTimeSet must be called at least once to start the RTC */
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time);
 /* Set the periodic interrupt rate to 1 second */
 R_RTC_PeriodicIrqRateSet(&g_rtc0_ctrl, RTC_PERIODIC_IRQ_SELECT_1_SECOND);
 /* Wait for the periodic interrupt */
 while (1)
    {
 /* Wait for interrupt */
    }
}
```

### RTC Alarm interrupt example

This is an example of alarm interrupt in RTC.

```
void rtc_alarm_irq_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /*Initialize the RTC module*/
    err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time1.time);
 R_RTC_CalendarAlarmSet(&g_rtc0_ctrl, &set_time1);
 /* Wait for the Alarm interrupt */
 while (1)
    {
 /* Wait for interrupt */
    }
}
```

### RTC Error Adjustment example

This is an example of modifying error adjustment in RTC.

```
void rtc_erroradj_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /*Initialize the RTC module*/
 R_RTC_Open(&g_rtc0_ctrl, &g_rtc1_cfg);
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time1.time);
 /* Modify Error Adjustment after RTC is running */
    err = R_RTC_ErrorAdjustmentSet(&g_rtc0_ctrl, &err_cfg2);
    handle_error(err);
}
```

## Data Structures

| | |
|---|---|
| struct | rtc_instance_ctrl_t |

## Data Structure Documentation

### ◆ rtc_instance_ctrl_t

| struct rtc_instance_ctrl_t | | |
|---|---|---|
| Channel control block. DO NOT INITIALIZE. Initialization occurs when rtc_api_t::open is called | | |
| Data Fields | | |
| uint32_t | open | Whether or not driver is open. |
| const rtc_cfg_t * | p_cfg | Pointer to initial configurations. |
| volatile bool | carry_isr_triggered | Was the carry isr triggered. |

## Function Documentation

### ◆ R_RTC_Open()

fsp_err_t R_RTC_Open ( rtc_ctrl_t *const *p_ctrl*, rtc_cfg_t const *const *p_cfg* )

Opens and configures the RTC driver module. Implements rtc_api_t::open. Configuration includes clock source, and interrupt callback function.

Example:

```
/* Initialize the RTC module */
   err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful and RTC has started. |
| FSP_ERR_ASSERTION | Invalid p_ctrl or p_cfg pointer. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_INVALID_ARGUMENT | Invalid time parameter field. |

### ◆ R_RTC_Close()

fsp_err_t R_RTC_Close ( rtc_ctrl_t *const *p_ctrl*)

Close the RTC driver. Implements rtc_api_t::close

**Return values**

| | |
|---|---|
| FSP_SUCCESS | De-Initialization was successful and RTC driver closed. |
| FSP_ERR_ASSERTION | Invalid p_ctrl. |
| FSP_ERR_NOT_OPEN | Driver not open already for close. |

### ◆ R_RTC_CalendarTimeSet()

fsp_err_t R_RTC_CalendarTimeSet ( rtc_ctrl_t *const *p_ctrl*, rtc_time_t *const *p_time* )

Set the calendar time.

Implements rtc_api_t::calendarTimeSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calendar time set operation was successful. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |
| FSP_ERR_INVALID_ARGUMENT | Invalid time parameter field. |

### ◆ R_RTC_CalendarTimeGet()

fsp_err_t R_RTC_CalendarTimeGet ( rtc_ctrl_t *const *p_ctrl*, rtc_time_t *const *p_time* )

Get the calendar time.

**Warning**

Do not call this function from a critical section or from an interrupt with higher priority than the carry interrupt, or the time returned may be inaccurate.

Implements rtc_api_t::calendarTimeGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calendar time get operation was successful. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |
| FSP_ERR_IRQ_BSP_DISABLED | User IRQ parameter not valid |

### ◆ R_RTC_CalendarAlarmSet()

fsp_err_t R_RTC_CalendarAlarmSet ( rtc_ctrl_t *const *p_ctrl*, rtc_alarm_time_t *const *p_alarm* )

Set the calendar alarm time.

Implements rtc_api_t::calendarAlarmSet.

**Precondition**

The calendar counter must be running before the alarm can be set.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calendar alarm time set operation was successful. |
| FSP_ERR_INVALID_ARGUMENT | Invalid time parameter field. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |
| FSP_ERR_IRQ_BSP_DISABLED | User IRQ parameter not valid |

### ◆ R_RTC_CalendarAlarmGet()

fsp_err_t R_RTC_CalendarAlarmGet ( rtc_ctrl_t *const *p_ctrl*, rtc_alarm_time_t *const *p_alarm* )

Get the calendar alarm time.

Implements rtc_api_t::calendarAlarmGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calendar alarm time get operation was successful. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |

### ◆ R_RTC_PeriodicIrqRateSet()

fsp_err_t R_RTC_PeriodicIrqRateSet ( rtc_ctrl_t *const *p_ctrl*, rtc_periodic_irq_select_t const *rate* )

Set the periodic interrupt rate and enable periodic interrupt.

Implements rtc_api_t::periodicIrqRateSet

***Note***

*To start the RTC R_RTC_CalendarTimeSet must be called at least once.*

Example:

```
/* Set the periodic interrupt rate to 1 second */
R_RTC_PeriodicIrqRateSet(&g_rtc0_ctrl, RTC_PERIODIC_IRQ_SELECT_1_SECOND);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The periodic interrupt rate was successfully set. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |
| FSP_ERR_IRQ_BSP_DISABLED | User IRQ parameter not valid |

### ◆ R_RTC_ErrorAdjustmentSet()

fsp_err_t R_RTC_ErrorAdjustmentSet ( rtc_ctrl_t *const *p_ctrl*, rtc_error_adjustment_cfg_t const *const *err_adj_cfg* )

This function sets time error adjustment

Implements rtc_api_t::errorAdjustmentSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Time error adjustment successful. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open for operation. |
| FSP_ERR_UNSUPPORTED | The clock source is not sub-clock. |
| FSP_ERR_INVALID_ARGUMENT | Invalid error adjustment value. |

### ◆ R_RTC_InfoGet()

fsp_err_t R_RTC_InfoGet ( rtc_ctrl_t *const *p_ctrl*, rtc_info_t *const *p_rtc_info* )

Set RTC clock source and running status information ad store it in provided pointer p_rtc_info

Implements rtc_api_t::infoGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Get information Successful. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |

### ◆ R_RTC_VersionGet()

fsp_err_t R_RTC_VersionGet ( fsp_version_t * *p_version*)

Get driver version based on compile time macros.

Implements rtc_api_t::versionGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | The parameter p_version is NULL. |

## 4.2.38 Serial Communications Interface (SCI) I2C (r_sci_i2c)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_SCI_I2C_VersionGet (fsp_version_t *const p_version) |
| fsp_err_t | R_SCI_I2C_Open (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_cfg_t const *const p_cfg) |
| fsp_err_t | R_SCI_I2C_Close (i2c_master_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_I2C_Read (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart) |
| fsp_err_t | R_SCI_I2C_Write (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const |

| | |
|---|---|
| | p_src, uint32_t const bytes, bool const restart) |
| fsp_err_t | R_SCI_I2C_Abort (i2c_master_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_I2C_SlaveAddressSet (i2c_master_ctrl_t *const p_api_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode) |

### Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the I2C Master Interface.

# Overview

The Simple I2C master on SCI HAL module supports transactions with an I2C Slave device. Callbacks must be provided which would be invoked when a transmission or receive has been completed. The callback arguments will contain information about the transaction status, bytes transferred and a pointer to the user defined context.

### Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100 kHz transaction rate.
  - Fast Mode Support with up to 400 kHz transaction rate.
- SDA Delay in nanoseconds can be specified as a part of the configuration.
- I2C Master Read from a slave device.
- I2C Master Write to a slave device.
- Abort any in-progress transactions.
- Set the address of the slave device.
- Non-blocking behavior is achieved by the use of callbacks.
- Additional build-time features
  - Optional (build time) DTC support for read and write respectively.
  - Optional (build time) support for 10-bit slave addressing.

# Configuration

### Build Time Configurations for r_sci_i2c

The following build time configurations are defined in fsp_cfg/r_sci_i2c_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| DTC on Transmission and Reception | • Enabled<br>• Disabled | Disabled | If enabled, DTC instances will be included in the build for both transmission and reception. |
| 10-bit slave addressing | • Enabled | Disabled | If enabled, the driver |

| | | |
|---|---|---|
| | • Disabled | will support 10-bit slave addressing mode along with the default 7-bit slave addressing mode. |

**Configurations for Driver > Connectivity > I2C Master Driver on r_sci_i2c**

This module can be added to the Stacks tab via New Stack > Driver > Connectivity > I2C Master Driver on r_sci_i2c:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_i2c0 | Module name. |
| Channel | Value must be an integer between 0 and 9 | 0 | Select the SCI channel. |
| Slave Address | Value must be a hex value | 0x00 | Specify the slave address. |
| Address Mode | • 7-Bit<br>• 10-Bit | 7-Bit | Select the address mode. |
| Rate | • Standard<br>• Fast-mode | Standard | Select the I2C data rate. |
| SDA Output Delay (nano seconds) | Must be a valid non-negative integer with maximum configurable value of 300 | 300 | Specify the SDA output delay in nanoseconds. |
| Noise filter setting | • Use clock signal divided by 1 with noise filter<br>• Use clock signal divided by 2 with noise filter<br>• Use clock signal divided by 4 with noise filter<br>• Use clock signal divided by 8 with noise filter | Use clock signal divided by 1 with noise filter | Select the sampling clock for the digital noise filter |
| Bit Rate Modulation | • Enable<br>• Disable | Enable | Enabling bitrate modulation reduces the percent error of the actual bitrate with respect to the requested baud rate. It does this by modulating the number of cycles per clock output pulse, so the |

| | | | |
|---|---|---|---|
| | | | clock is no longer a square wave. |
| Callback | Name must be a valid C symbol | sci_i2c_master_callback | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |
| Interrupt Priority Level | MCU Specific Options | | Select the interrupt priority level. This is set for TXI, RXI (if used), TEI interrupts. |
| RX Interrupt Priority Level [Only used when DTC is enabled] | MCU Specific Options | | Select the interrupt priority level. This is set for RXI only when DTC is enabled. |

### Clock Configuration

The SCI I2C peripheral module uses either PCLKA or PCLKB (depending on the MCU) as its clock source. The actual I2C transfer rate will be calculated and set by the tooling depending on the selected transfer rate and the SDA delay. If the PCLK is configured in such a manner that the selected internal rate cannot be achieved, an error will be returned.

### Pin Configuration

The SCI I2C 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. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

## Usage Notes

### Interrupt Configuration

- 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.
- Set equal priority levels for all the interrupts mentioned above. Setting the interrupts to different priority levels could result in improper operation.

### SCI I2C Master Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate and SDA Delay. The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current PCLK settings is calculated and used.
- If a valid clock rate could not be calculated, an error is returned by the tool.

### Enabling DTC with the SCI I2C

- DTC transfer support is configurable and is disabled from the build by default. SCI I2C driver provides two DTC instances for transmission and reception respectively.

- For further details on DTC please refer Data Transfer Controller (r_dtc)

### Multiple Devices on the Bus

- A single SCI I2C instance can be used to communicate with multiple slave devices on the same channel by using the SlaveAddressSet API.

### Restart

- SCI I2C master can hold the the bus after an I2C transaction by issuing Restart. This will mimic a stop followed by start condition.

## Examples

### Basic Example

This is a basic example of minimal use of the r_sci_i2c in an application. This example shows how this driver can be used for basic read and write operations.

```
void basic_example (void)
{
 fsp_err_t err;
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
 /* Initialize the IIC module */
    err = R_SCI_I2C_Open(&g_i2c_device_ctrl_1, &g_i2c_device_cfg_1);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_tx_buffer[i] = (uint8_t) i;
    }
 /* Send data to I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_SCI_I2C_Write(&g_i2c_device_ctrl_1, &g_i2c_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    handle_error(err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
```

```
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Read data back from the I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SCI_I2C_Read(&g_i2c_device_ctrl_1, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    handle_error(err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_tx_buffer, g_i2c_rx_buffer, I2C_BUFFER_SIZE_BYTES))
    {
       __BKPT(0);
    }
}
```

### Multiple Slave devices on the same channel (bus)

This example demonstrates how a single SCI I2C driver can be used to communicate with different slave devices which are on the same channel.

```
void single_channel_multi_slave (void)
{
 fsp_err_t err;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SCI_I2C_Open(&g_i2c_device_ctrl_2, &g_i2c_device_cfg_2);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Clear the recieve buffer */
    memset(g_i2c_rx_buffer, '0', I2C_BUFFER_SIZE_BYTES);
 /* Read data from I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_SCI_I2C_Read(&g_i2c_device_ctrl_2, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    handle_error(err);
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Send data to I2C slave on the same channel */
    err = R_SCI_I2C_SlaveAddressSet(&g_i2c_device_ctrl_2, I2C_SLAVE_DISPLAY_ADAPTER,
I2C_MASTER_ADDR_MODE_7BIT);
    handle_error(err);
    g_i2c_tx_buffer[0]   = (uint8_t) I2C_EXAMPLE_DATA_1;
    g_i2c_tx_buffer[1]   = (uint8_t) I2C_EXAMPLE_DATA_2;
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms           = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SCI_I2C_Write(&g_i2c_device_ctrl_2, &g_i2c_tx_buffer[0], 2U, false);
    handle_error(err);
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
```

```
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
        timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
        __BKPT(0);
    }
}
```

## Data Structures

| | |
|---|---|
| struct | sci_i2c_clock_settings_t |
| struct | sci_i2c_instance_ctrl_t |
| struct | sci_i2c_extended_cfg_t |

## Data Structure Documentation

### ◆ sci_i2c_clock_settings_t

| struct sci_i2c_clock_settings_t | | |
|---|---|---|
| I2C clock settings | | |
| Data Fields | | |
| bool | bitrate_modulation | Bit-rate Modulation Function enable or disable. |
| uint8_t | brr_value | Bit rate register settings. |
| uint8_t | clk_divisor_value | Clock Select settings. |
| uint8_t | mddr_value | Modulation Duty Register settings. |
| uint8_t | cycles_value | SDA Delay Output Cycles Select. |
| uint8_t | snfr_value | Noise Filter Setting Register value. |

### ◆ sci_i2c_instance_ctrl_t

| struct sci_i2c_instance_ctrl_t |
|---|
| I2C control structure. DO NOT INITIALIZE. |

### ◆ sci_i2c_extended_cfg_t

| struct sci_i2c_extended_cfg_t | | |
|---|---|---|
| SCI I2C extended configuration | | |
| Data Fields | | |
| sci_i2c_clock_settings_t | clock_settings | I2C Clock settings. |

## Function Documentation

### ◆ R_SCI_I2C_VersionGet()

fsp_err_t R_SCI_I2C_VersionGet ( fsp_version_t *const *p_version*)

Sets driver version based on compile time macros.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful version get. |
| FSP_ERR_ASSERTION | The parameter p_version is NULL. |

### ◆ R_SCI_I2C_Open()

fsp_err_t R_SCI_I2C_Open ( i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_cfg_t const *const *p_cfg* )

Opens the I2C device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Requested clock rate was set exactly. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_ASSERTION | Parameter check failure due to one or more reasons below:<br>1. p_api_ctrl or p_cfg is NULL.<br>2. extended parameter is NULL.<br>3. Callback parameter is NULL.<br>4. Clock rate requested is greater than 400KHz<br>5. Invalid IRQ number assigned |

### ◆ R_SCI_I2C_Close()

fsp_err_t R_SCI_I2C_Close ( i2c_master_ctrl_t *const *p_api_ctrl*)

Closes the I2C device. Power down I2C peripheral.

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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device closed without issue. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Device was not even opened. |

### ◆ R_SCI_I2C_Read()

fsp_err_t R_SCI_I2C_Read ( i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, bool const *restart* )

Performs a read from the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_RX_COMPLETE in the callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | The parameter p_ctrl, p_dest is NULL, bytes is 0. |
| FSP_ERR_INVALID_SIZE | Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer. |
| FSP_ERR_NOT_OPEN | Device was not even opened. |

### ◆ R_SCI_I2C_Write()

fsp_err_t R_SCI_I2C_Write ( i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*, bool const *restart* )

Performs a write to the I2C device.

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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | p_ctrl, p_src is NULL. |
| FSP_ERR_INVALID_SIZE | Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer. |
| FSP_ERR_NOT_OPEN | Device was not even opened. |

### ◆ R_SCI_I2C_Abort()

fsp_err_t R_SCI_I2C_Abort ( i2c_master_ctrl_t *const *p_api_ctrl*)

Aborts any in-progress transfer and forces the I2C peripheral into a ready state.

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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transaction was aborted without issue. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Device was not even opened. |

### ◆ R_SCI_I2C_SlaveAddressSet()

fsp_err_t R_SCI_I2C_SlaveAddressSet ( i2c_master_ctrl_t *const *p_api_ctrl*, uint32_t const *slave*, i2c_master_addr_mode_t const *addr_mode* )

Sets address and addressing mode of the slave device.

This function is used to set the device address and addressing mode of the slave without reconfiguring the entire bus.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Address of the slave is set correctly. |
| FSP_ERR_ASSERTION | p_ctrl or address is NULL. |
| FSP_ERR_NOT_OPEN | Device was not even opened. |
| FSP_ERR_IN_USE | An I2C Transaction is in progress. |

## 4.2.39 Serial Communications Interface (SCI) SPI (r_sci_spi)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | R_SCI_SPI_Open (spi_ctrl_t *p_api_ctrl, spi_cfg_t const *const p_cfg) |
| fsp_err_t | R_SCI_SPI_Read (spi_ctrl_t *const p_api_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_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) |
| fsp_err_t | R_SCI_SPI_WriteRead (spi_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SCI_SPI_Close (spi_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_SPI_VersionGet (fsp_version_t *p_version) |
| fsp_err_t | R_SCI_SPI_CalculateBitrate (uint32_t bitrate, sci_spi_div_setting_t *sclk_div, bool use_mddr) |

### Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the SPI Interface.

# Overview

### Features

- Standard SPI Modes
  - Master or Slave Mode
  - Clock Polarity (CPOL)
    - CPOL=0 SCLK is low when idle
    - CPOL=1 SCLK is high when idle
  - Clock Phase (CPHA)
    - CPHA=0 Data Sampled on the even edge of SCLK
    - CPHA=1 Data Sampled on the odd edge of SCLK
  - MSB/LSB first
- Configurable bit rate
- DTC Support
- Callback Events
  - Transfer Complete
  - RX Overflow Error (The SCI shift register is copied to the data register before previous data was read)

# Configuration

### Build Time Configurations for r_sci_spi

The following build time configurations are defined in fsp_cfg/r_sci_spi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| DTC Support | • Enabled<br>• Disabled | Enabled | If support for transfering data using the DTC will be compiled in. |

### Configurations for Driver > Connectivity > SPI Driver on r_sci_spi

This module can be added to the Stacks tab via New Stack > Driver > Connectivity > SPI Driver on r_sci_spi:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_spi0 | Module name. |
| Channel | Value must be an integer between 0 and 9 | 0 | Select the SCI channel. |
| Operating Mode | • Master | Master | Select the SPI |

| | | | |
|---|---|---|---|
| | • Slave | | operating mode. |
| Clock Phase | • Data sampling on odd edge, data variation on even edge<br>• Data sampling on even edge, data variation on odd edge | Data sampling on odd edge, data variation on even edge | Select the clock edge to sample data. |
| Clock Polarity | • Low when idle<br>• High when idle | Low when idle | Select clock level when idle. |
| Mode Fault Error | • Enable<br>• Disable | Disable | Detect master/slave mode conflicts. |
| Bit Order | • MSB First<br>• LSB First | MSB First | Select the data bit order. |
| Callback | Name must be a valid C symbol | sci_spi_callback | A user callback function that is called from the sci spi interrupts when a transfer is completed or an error has occurred. |
| Receive Interrupt Priority | MCU Specific Options | | Select the receive interrupt priority. |
| Transmit Interrupt Priority | MCU Specific Options | | Select the transmit interrupt priority. |
| Transmit End Interrupt Priority | MCU Specific Options | | Select the transmit end interrupt priority. |
| Error Interrupt Priority | MCU Specific Options | | Select the error interrupt priority. |
| Bitrate | Must be a valid non-negative integer with maximum configurable value of 30000000 | 30000000 | Enter the desired bitrate. |
| Bitrate Modulation | • Disabled<br>• Enabled | Disabled | Enabling bitrate modulation reduces the percent error of the actual bitrate with respect to the requested baud rate. It does this by modulating the number of cycles per clock output pulse, so the clock is no longer a square wave. |

**Clock Configuration**

The SCI_SPI clock is derived from the following peripheral clock on each device.

| MCU | Peripheral Clock |
|---|---|
| RA2A1 | PCLKB |
| RA4M1 | PCLKA |
| RA6M1 | PCLKA |
| RA6M2 | PCLKA |
| RA6M3 | PCLKA |

### Pin Configuration

This module uses SCIn_MOSI, SCIn_MISO, SCIn_SPCK, and SCIn_SS pins to communicate with on board devices.

*Note*

*At high bit rates, it might be necessary to configure the pins with IOPORT_CFG_DRIVE_HIGH.*

## Usage Notes

### Transfer Complete Event

The transfer complete event is triggered when all of the data has been transfered. In slave mode if the SS pin is de-asserted then no transfer complete event is generated until the SS pin is asserted and the remaining data is transferred.

### Performance

At high bit rates, interrupts may not be able to service transfers fast enough. In master mode this means there will be a delay between each data frame. In slave mode this could result in RX Overflow errors.

In order to improve performance at high bit rates, it is recommended that the instance be configured to service transfers using the DTC.

### Transmit From RXI Interrupt

After every byte, the SCI SPI peripheral generates a transmit buffer empty interrupt and a receive buffer full interrupt. Whenever possible, the SCI_SPI module handles both interrupts in the receive buffer full interrupt. This improves performance when the DTC is not being used.

### Slave Select Pin

- In master mode the slave select pin must be driven in software.
- In slave mode the hardware handles the slave select pin and will only transfer data when the SS pin is low.

### Bit Rate Modulation

Depending on the peripheral clock frequency, the desired bit rate may not be achievable. With bit rate modulation, the device can remove a configurable number of input clock pulses to the internal bit rate counter in order to create the desired bit rate. This has the effect of changing the period of

individual bits in order to achieve the desired average bit rate. For more information see section 34.9 Bit Rate Modulation Function in the RA6M3 manual.

# Examples

### Basic Example

This is a basic example of minimal use of the SCI_SPI in an application.

```
static volatile bool g_transfer_complete = false;
static void r_sci_spi_callback (spi_callback_args_t * p_args)
{
 if (SPI_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_transfer_complete = true;
    }
}
void sci_spi_basic_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
 /* Configure Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Configure Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the SPI module. */
    err = R_SCI_SPI_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err = R_SCI_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    handle_error(err);
```

```
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Wait for minimum time required between transfers. */
 R_BSP_SoftwareDelay(SSL_NEXT_ACCESS_DELAY, BSP_DELAY_UNITS_MICROSECONDS);
 /* Assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err = R_SCI_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    handle_error(err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
}
```

## Function Documentation

### ◆ R_SCI_SPI_Open()

fsp_err_t R_SCI_SPI_Open ( spi_ctrl_t * *p_api_ctrl*, spi_cfg_t const *const *p_cfg* )

Initialize a channel for SPI communication mode. Implements spi_api_t::open.

This function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enables the clock for the SCI channel.
- Initializes the associated registers with default value and the user-configurable options.
- Provides the channel handle for use with other API functions.

**Parameters**

| | |
|---|---|
| p_api_ctrl | Pointer to the control structure. |
| p_cfg | Pointer to a configuration structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel initialized successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid or NULL. |
| FSP_ERR_ALREADY_OPEN | The instance has already been opened. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel number is invalid. |

### ◆ R_SCI_SPI_Read()

fsp_err_t R_SCI_SPI_Read ( spi_ctrl_t *const *p_api_ctrl*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

Receive data from an SPI device. Implements spi_api_t::read.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission by writing data to the TXD register.
- 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.

**Parameters**

| | | |
|---|---|---|
| | p_api_ctrl | Pointer to the control structure. |
| | p_dest | Pointer to the destination buffer. |
| [in] | length | The number of bytes to transfer. |
| [in] | bit_width | Invalid for SCI_SPI (Set to SPI_BIT_WIDTH_8_BITS). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Read operation successfully completed. |
| FSP_ERR_ASSERTION | One of the following invalid parameters passed:<br>• Pointer p_api_ctrl is NULL<br>• Bit width is not 8 bits<br>• Length is equal to 0<br>• Pointer to destination is NULL |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_UNSUPPORTED | The given bit_width is not supported. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

### ◆ R_SCI_SPI_Write()

fsp_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* )

Transmit data to a SPI device. Implements spi_api_t::write.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable interrupts.
- Start data transmission with data via transmit buffer empty interrupt.
- Copy data from source buffer to the SPI data register for transmission.
- Complete data transmission via transmit buffer empty interrupt.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

**Parameters**

| | | |
|---|---|---|
| | p_api_ctrl | Pointer to the control structure. |
| | p_src | Pointer to the source buffer. |
| [in] | length | The number of bytes to transfer. |
| [in] | bit_width | Invalid for SCI_SPI (Set to SPI_BIT_WIDTH_8_BITS). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write operation successfully completed. |
| FSP_ERR_ASSERTION | One of the following invalid parameters passed:<br>• Pointer p_api_ctrl is NULL<br>• Pointer to source is NULL<br>• Length is equal to 0<br>• Bit width is not equal to 8 bits |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_UNSUPPORTED | The given bit_width is not supported. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

### ◆ R_SCI_SPI_WriteRead()

fsp_err_t R_SCI_SPI_WriteRead ( spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

Simultaneously transmit data to SPI device while receiving data from SPI device (full duplex). Implements spi_api_t::writeRead.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission using transmit buffer empty interrupt (or by writing to the TDR register).
- Copy data from source buffer to the SPI data register for transmission.
- Receive data from receive buffer full interrupt and copy data to the destination buffer.
- Complete data transmission and reception via transmit end interrupt.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

**Parameters**

| | | |
|---|---|---|
| | p_api_ctrl | Pointer to the control structure. |
| | p_src | Pointer to the source buffer. |
| | p_dest | Pointer to the destination buffer. |
| [in] | length | The number of bytes to transfer. |
| [in] | bit_width | Invalid for SCI_SPI (Set to SPI_BIT_WIDTH_8_BITS). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write operation successfully completed. |
| FSP_ERR_ASSERTION | One of the following invalid parameters passed:<br>• Pointer p_api_ctrl is NULL<br>• Pointer to source is NULL<br>• Pointer to destination is NULL<br>• Length is equal to 0<br>• Bit width is not equal to 8 bits |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_UNSUPPORTED | The given bit_width is not supported. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

### ◆ R_SCI_SPI_Close()

fsp_err_t R_SCI_SPI_Close ( spi_ctrl_t *const *p_api_ctrl*)

Disable the SCI channel and set the instance as not open. Implements spi_api_t::close.

**Parameters**

| | |
|---|---|
| p_api_ctrl | Pointer to an opened instance. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |

### ◆ R_SCI_SPI_VersionGet()

fsp_err_t R_SCI_SPI_VersionGet ( fsp_version_t * *p_version*)

Get the version information of the underlying driver. Implements spi_api_t::versionGet.

**Parameters**

| | |
|---|---|
| p_version | Pointer to version structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful version get. |
| FSP_ERR_ASSERTION | The parameter p_version is NULL. |

### ◆ R_SCI_SPI_CalculateBitrate()

fsp_err_t R_SCI_SPI_CalculateBitrate ( uint32_t *bitrate*, sci_spi_div_setting_t * *sclk_div*, bool *use_mddr* )

Calculate the register settings required to achieve the desired bitrate.

**Parameters**

| | | |
|---|---|---|
| [in] | bitrate | bitrate [bps]. For example, 250,000; 500,00; 2,500,000 (max), etc. |
| | sclk_div | Pointer to sci_spi_div_setting_t used to configure baudrate settings. |
| [in] | use_mddr | Calculate the divider settings for use with MDDR. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Baud rate is set successfully. |
| FSP_ERR_ASSERTION | Baud rate is not achievable. |

***Note***

*The application must pause for 1 bit time after the BRR register is loaded before transmitting/receiving to allow time for the clock to settle.*

## 4.2.40 Serial Communications Interface (SCI) UART (r_sci_uart)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_SCI_UART_Open (uart_ctrl_t *const p_api_ctrl, uart_cfg_t const *const p_cfg) |
| fsp_err_t | R_SCI_UART_Read (uart_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_SCI_UART_Write (uart_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint32_t const bytes) |
| fsp_err_t | R_SCI_UART_BaudSet (uart_ctrl_t *const p_api_ctrl, void const *const p_baud_setting) |
| fsp_err_t | R_SCI_UART_InfoGet (uart_ctrl_t *const p_api_ctrl, uart_info_t *const p_info) |

| | |
|---|---|
| fsp_err_t | R_SCI_UART_Close (uart_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_UART_VersionGet (fsp_version_t *p_version) |
| fsp_err_t | R_SCI_UART_Abort (uart_ctrl_t *const p_api_ctrl, uart_dir_t communication_to_abort) |
| fsp_err_t | R_SCI_UART_BaudCalculate (uint32_t baudrate, bool bitrate_modulation, uint32_t baud_rate_error_x_1000, baud_setting_t *const p_baud_setting) |

### Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the UART Interface.

# Overview

### Features

The SCI UART module supports the following features:

- Full-duplex UART communication
- Interrupt-driven data transmission and reception
- Invoking the user-callback function with an event code (RX/TX complete, TX data empty, RX char, error, etc)
- Baud-rate change at run-time
- Bit rate modulation and noise cancellation
- RS232 CTS/RTS hardware flow control (with an associated pin)
- RS485 Half/Full Duplex flow control
- Integration with the DTC transfer module
- Abort in-progress read/write operations
- FIFO support on supported channels

# Configuration

### Build Time Configurations for r_sci_uart

The following build time configurations are defined in fsp_cfg/r_sci_uart_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| FIFO Support | • Enable<br>• Disable | Disable | Enable FIFO support for the SCI_UART module. |
| DTC Support | • Enable<br>• Disable | Disable | Enable DTC support for the SCI_UART module. |

| | | | |
|---|---|---|---|
| RS232/RS485 Flow Control Support | • Enable<br>• Disable | Disable | Enable RS232 and RS485 flow control support using a user provided pin. |

**Configurations for Driver > Connectivity > UART Driver on r_sci_uart**

This module can be added to the Stacks tab via New Stack > Driver > Connectivity > UART Driver on r_sci_uart:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General > Name | Name must be a valid C symbol | g_uart0 | Module name. |
| General > Channel | Value must be an integer between 0 and 9 | 0 | Select the SCI channel. |
| General > Data Bits | • 8bits<br>• 7bits<br>• 9bits | 8bits | Select the number of bits per word. |
| General > Parity | • None<br>• Odd<br>• Even | None | Select the parity mode. |
| General > Stop Bits | • 1bit<br>• 2bits | 1bit | Select the number of stop bits. |
| Baud > Baud Rate | Value must be an integer greater than 0 | 115200 | Enter the desired baud rate. |
| Baud > Baud Rate Modulation | • Disabled<br>• Enabled | Disabled | Enabling baud rate modulation reduces the percent error of the actual baud rate with respect to the requested baud rate. It does this by modulating the number of cycles per clock, so some bits are slightly longer than others. |
| Baud > Max Error (%) | Must be a valid non negative integer with a maximum configurable value of 100 | 5 | Maximum percent error allowed during baud calculation. This is used by the algorithm to determine whether or not to consider using less accurate alternative register settings.<br><br>NOTE: The baud calculation does not |

| | | | |
|---|---|---|---|
| | | | show an error in the tool if this percent error was not achieved. The calculated percent error is recorded in a comment in the generated baud_setting_t structure. |
| Flow Control > CTS/RTS Selection | • RTS (CTS is disabled)<br>• CTS (Note that RTS is available when enabling External RTS Operation mode which uses 1 GPIO pin) | 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. |
| Flow Control > UART Communication Mode | • RS232<br>• RS485 Half Duplex<br>• RS485 Full Duplex | RS232 | Select the UART communication mode as either RS232 or RS485. |
| Flow Control > Pin Control | • Enabled<br>• Disabled | Disabled | Enables pin control for external RTS in RS232 mode RS485 mode. |
| Flow Control > RTS Port | Refer to the RA Configuration tool for available options. | Disabled | Specify the flow control pin port for the MCU. |
| Flow Control > RTS Pin | Refer to the RA Configuration tool for available options. | Disabled | Specify the flow control pin for the MCU. |
| Extra > Clock Source | • Internal Clock<br>• Internal Clock With Output on SCK<br>• External Clock 8x baud rate<br>• External Clock 16x baud rate | Internal Clock | Selection of the clock source to be used in the baud-rate clock generator. When internal clock is used the baud rate can be output on the SCK pin. |
| Extra > Start bit detection | • Falling Edge<br>• Low Level | Falling Edge | Start bit detected as falling edge or low level. |
| Extra > Noise Filter | • Enable<br>• Disable | Disable | Enable the digital noise filter on RXDn pin. The digital noise filter block in SCI consists of two-stage flipflop circuits. |
| Extra > Receive FIFO | • One | Max | Unused if the channel |

| | | | |
|---|---|---|---|
| Trigger Level | • Max | | has no FIFO or if DTC is used for reception. Set to One to get a callback immediately when each byte is received. Set to Max to get a callback when FIFO is full or after 15 bit times with no data (fewer interrupts). |
| Interrupts > Callback | Name must be a valid C symbol | user_uart_callback | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |
| Interrupts > Receive Interrupt Priority | MCU Specific Options | | Select the receive interrupt priority. |
| Interrupts > Transmit Data Empty Interrupt Priority | MCU Specific Options | | Select the transmit interrupt priority. |
| Interrupts > Transmit End Interrupt Priority | MCU Specific Options | | Select the transmit end interrupt priority. |
| Interrupts > Error Interrupt Priority | MCU Specific Options | | Select the error interrupt priority. |

### Clock Configuration

The SCI clock is derived from the following peripheral clock on each device.

| MCU | Peripheral Clock |
|---|---|
| RA2A1 | PCLKB |
| RA4M1 | PCLKA |
| RA6M1 | PCLKA |
| RA6M2 | PCLKA |
| RA6M3 | PCLKA |

The clock source for the baud-rate clock generator can be selected from the internal clock, the external clock times 8 or the external clock times 16. The external clock is supplied to the SCK pin.

### Pin Configuration

This module uses TXD and RXD to communicate to external devices. CTS or RTS can be controlled by the hardware. If both are desired a GPIO pin can be used for RTS. When the internal clock is the source for the baud-rate generator the SCK pin can be used to output a clock with the same frequency as the bit rate.

# Usage Notes

### Limitations

- Transfer size must be less than or equal to 64K bytes if DTC interface is used for transfer. uart_api_t::infoGet API can be used to get the max transfer size allowed.
- Reception is still enabled after uart_api_t::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.
- When using 9-bit reception with DTC, clear the upper 7 bits of data before processing the read data. The upper 7 bits contain status flags that are part of the register used to read data in 9-bit mode.

# Examples

### SCI UART Example

```
uint8_t g_dest[TRANSFER_LENGTH];
uint8_t g_src[TRANSFER_LENGTH];
uint8_t g_out_of_band_received[TRANSFER_LENGTH];
uint32_t g_transfer_complete = 0;
uint32_t g_receive_complete = 0;
uint32_t g_out_of_band_index = 0;
void r_sci_uart_basic_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_SCI_UART_Open(&g_uart0_ctrl, &g_uart0_cfg);
    handle_error(err);
    err = R_SCI_UART_Read(&g_uart0_ctrl, g_dest, TRANSFER_LENGTH);
    handle_error(err);
    err = R_SCI_UART_Write(&g_uart0_ctrl, g_src, TRANSFER_LENGTH);
    handle_error(err);
 while (!g_transfer_complete)
    {
    }
```

```
 while (!g_receive_complete)
    {
    }
}
void example_callback (uart_callback_args_t * p_args)
{
 /* Handle the UART event */
 switch (p_args->event)
    {
 /* Received a character */
 case UART_EVENT_RX_CHAR:
      {
 /* Only put the next character in the receive buffer if there is space for it */
 if (sizeof(g_out_of_band_received) > g_out_of_band_index)
      {
 /* Write either the next one or two bytes depending on the receive data size */
 if (UART_DATA_BITS_8 >= g_uart0_cfg.data_bits)
      {
                    g_out_of_band_received[g_out_of_band_index++] = (uint8_t)
p_args->data;
      }
 else
      {
                    uint16_t * p_dest = (uint16_t *)
&g_out_of_band_received[g_out_of_band_index];
                    *p_dest              = (uint16_t) p_args->data;
                    g_out_of_band_index += 2;
      }
      }
 break;
      }
 /* Receive complete */
 case UART_EVENT_RX_COMPLETE:
      {
```

```
            g_receive_complete = 1;
 break;
      }
 /* Transmit complete */
 case UART_EVENT_TX_COMPLETE:
      {
            g_transfer_complete = 1;
 break;
      }
 default:
      {
      }
    }
}
```

### SCI UART Baud Set Example

```
#define SCI_UART_BAUDRATE_19200 (19200)
void r_sci_uart_baud_example (void)
{
 baud_setting_t baud_setting;
    uint32_t       baud_rate                = SCI_UART_BAUDRATE_19200;
 bool          enable_bitrate_modulation = false;
    uint32_t       error_rate_x_1000         = 5;
 fsp_err_t err = R_SCI_UART_BaudCalculate(baud_rate, enable_bitrate_modulation,
error_rate_x_1000, &baud_setting);
    handle_error(err);
    err = R_SCI_UART_BaudSet(&g_uart0_ctrl, (void *) &baud_setting);
    handle_error(err);
}
```

### Data Structures

| | |
|---|---|
| struct | sci_uart_instance_ctrl_t |
| struct | baud_setting_t |

| | |
|---|---|
| struct | sci_uart_extended_cfg_t |

### Enumerations

| | |
|---|---|
| enum | sci_clk_src_t |
| enum | uart_mode_t |
| enum | sci_uart_rx_fifo_trigger_t |
| enum | sci_uart_start_bit_detect_t |
| enum | sci_uart_noise_cancellation_t |
| enum | sci_uart_ctsrts_config_t |

## Data Structure Documentation

### ◆ sci_uart_instance_ctrl_t

| |
|---|
| struct sci_uart_instance_ctrl_t |
| UART instance control block. |

### ◆ baud_setting_t

| struct baud_setting_t | | |
|---|---|---|
| Register settings to acheive a desired baud rate and modulation duty. | | |
| Data Fields | | |
| union baud_setting_t | __unnamed__ | |
| uint8_t | cks: 2 | CKS value to get divisor (CKS = N) |
| uint8_t | brr | Bit Rate Register setting. |
| uint8_t | mddr | Modulation Duty Register setting. |

### ◆ sci_uart_extended_cfg_t

| struct sci_uart_extended_cfg_t | | |
|---|---|---|
| UART on SCI device Configuration | | |
| Data Fields | | |
| sci_clk_src_t | clock | The source clock for the baud-rate generator. If internal optionally output baud rate on SCK. |
| sci_uart_start_bit_detect_t | rx_edge_start | Start reception on falling edge. |
| | | |

| | | |
|---|---|---|
| sci_uart_noise_cancellation_t | noise_cancel | Noise cancellation setting. |
| baud_setting_t * | p_baud_setting | Register settings for a desired baud rate. |
| sci_uart_rx_fifo_trigger_t | rx_fifo_trigger | Receive FIFO trigger level, unused if channel has no FIFO or if DTC is used. |
| uart_mode_t | uart_mode | UART communication mode selection. |
| bsp_io_port_pin_t | flow_control_pin | UART Driver Enable pin. |
| sci_uart_ctsrts_config_t | ctsrts_en | CTS/RTS function of the SSn pin. |

## Enumeration Type Documentation

### ◆ sci_clk_src_t

| enum sci_clk_src_t | |
|---|---|
| Enumeration for SCI clock source | |
| Enumerator | |
| SCI_UART_CLOCK_INT | Use internal clock for baud generation. |
| SCI_UART_CLOCK_INT_WITH_BAUDRATE_OUTPUT | Use internal clock for baud generation and output on SCK. |
| SCI_UART_CLOCK_EXT8X | Use external clock 8x baud rate. |
| SCI_UART_CLOCK_EXT16X | Use external clock 16x baud rate. |

### ◆ uart_mode_t

| enum uart_mode_t | |
|---|---|
| UART communication mode definition | |
| Enumerator | |
| UART_MODE_RS232 | Enables RS232 communication mode. |
| UART_MODE_RS485_HD | Enables RS485 half duplex communication mode. |
| UART_MODE_RS485_FD | Enables RS485 full duplex communication mode. |

### ◆ sci_uart_rx_fifo_trigger_t

| enum sci_uart_rx_fifo_trigger_t | |
|---|---|
| Receive FIFO trigger configuration. | |
| Enumerator | |
| 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) |

### ◆ sci_uart_start_bit_detect_t

| enum sci_uart_start_bit_detect_t | |
|---|---|
| Asynchronous Start Bit Edge Detection configuration. | |
| Enumerator | |
| SCI_UART_START_BIT_LOW_LEVEL | Detect low level on RXDn pin as start bit. |
| SCI_UART_START_BIT_FALLING_EDGE | Detect falling level on RXDn pin as start bit. |

### ◆ sci_uart_noise_cancellation_t

| enum sci_uart_noise_cancellation_t | |
|---|---|
| Noise cancellation configuration. | |
| Enumerator | |
| SCI_UART_NOISE_CANCELLATION_DISABLE | Disable noise cancellation. |
| SCI_UART_NOISE_CANCELLATION_ENABLE | Enable noise cancellation. |

### ◆ sci_uart_ctsrts_config_t

| enum sci_uart_ctsrts_config_t | |
|---|---|
| CTS/RTS function of the SSn pin. | |
| Enumerator | |
| SCI_UART_CTSRTS_RTS_OUTPUT | Disable CTS function (RTS output function is enabled) |
| SCI_UART_CTSRTS_CTS_INPUT | Enable CTS function. |

## Function Documentation

### ◆ R_SCI_UART_Open()

fsp_err_t R_SCI_UART_Open ( uart_ctrl_t *const *p_api_ctrl*, uart_cfg_t const *const *p_cfg* )

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. Implements uart_api_t::open

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to UART control block or configuration structure is NULL. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The requested channel does not exist on this MCU. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::open

### ◆ R_SCI_UART_Read()

fsp_err_t R_SCI_UART_Read ( uart_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Receives user specified number of bytes into destination buffer pointer. Implements uart_api_t::read

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data reception successfully ends. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used |
| FSP_ERR_INVALID_ARGUMENT | Destination address or data size is not valid for 9-bit mode. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_IN_USE | A previous read operation is still in progress. |
| FSP_ERR_UNSUPPORTED | SCI_UART_CFG_RX_ENABLE is set to 0 |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

***Note***

*If 9-bit data length is specified at R_SCI_UART_Open call, p_dest must be aligned 16-bit boundary.*

### ◆ R_SCI_UART_Write()

fsp_err_t R_SCI_UART_Write ( uart_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes* )

Transmits user specified number of bytes from the source buffer pointer. Implements uart_api_t::write

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data transmission finished successfully. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used |
| FSP_ERR_INVALID_ARGUMENT | Source address or data size is not valid for 9-bit mode. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_IN_USE | A UART transmission is in progress |
| FSP_ERR_UNSUPPORTED | SCI_UART_CFG_TX_ENABLE is set to 0 |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

***Note***

*If 9-bit data length is specified at R_SCI_UART_Open call, p_src must be aligned on a 16-bit boundary.*

### ◆ R_SCI_UART_BaudSet()

fsp_err_t R_SCI_UART_BaudSet ( uart_ctrl_t *const *p_api_ctrl*, void const *const *p_baud_setting* )

Updates the baud rate using the clock selected in Open. p_baud_setting is a pointer to a baud_setting_t structure. Implements uart_api_t::baudSet

**Warning**

This terminates any in-progress transmission.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Baud rate was successfully changed. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL or the UART is not configured to use the internal clock. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_SCI_UART_InfoGet()

fsp_err_t R_SCI_UART_InfoGet ( uart_ctrl_t *const *p_api_ctrl*, uart_info_t *const *p_info* )

Provides the driver information, including the maximum number of bytes that can be received or transmitted at a time. Implements uart_api_t::infoGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in provided p_info. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_SCI_UART_Close()

fsp_err_t R_SCI_UART_Close ( uart_ctrl_t *const *p_api_ctrl*)

Aborts any in progress transfers. Disables interrupts, receiver, and transmitter. Closes lower level transfer drivers if used. Removes power. Implements uart_api_t::close

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_SCI_UART_VersionGet()

fsp_err_t R_SCI_UART_VersionGet ( fsp_version_t * *p_version*)

Provides API and code version in the user provided pointer. Implements uart_api_t::versionGet

**Parameters**

| | | |
|---|---|---|
| [in] | p_version | Version number set here |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version information stored in provided p_version. |
| FSP_ERR_ASSERTION | p_version is NULL. |

### ◆ R_SCI_UART_Abort()

fsp_err_t R_SCI_UART_Abort ( uart_ctrl_t *const *p_api_ctrl*, uart_dir_t *communication_to_abort* )

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. Implements uart_api_t::communicationAbort

**Return values**

| | |
|---|---|
| FSP_SUCCESS | UART transaction aborted successfully. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_UNSUPPORTED | The requested Abort direction is unsupported. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::disable

### ◆ R_SCI_UART_BaudCalculate()

fsp_err_t R_SCI_UART_BaudCalculate ( uint32_t *baudrate*, bool *bitrate_modulation*, uint32_t *baud_rate_error_x_1000*, baud_setting_t *const *p_baud_setting* )

Calculates baud rate register settings. Evaluates and determines the best possible settings set to the baud rate related registers.

**Parameters**

| | | |
|---|---|---|
| [in] | baudrate | Baud rate [bps]. For example, 19200, 57600, 115200, etc. |
| [in] | bitrate_modulation | Enable bitrate modulation |
| [in] | baud_rate_error_x_1000 | <baud_rate_percent_error> x 1000 required for module to function. Absolute max baud_rate_error is 15000 (15%). |
| [out] | p_baud_setting | Baud setting information stored here if successful |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Baud rate is set successfully |
| FSP_ERR_ASSERTION | Null pointer |
| FSP_ERR_INVALID_ARGUMENT | Baud rate is '0', source clock frequency could not be read, or error in calculated baud rate is larger than 10%. |

## 4.2.41 Sigma Delta Analog to Digital Converter (r_sdadc)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_SDADC_Open (adc_ctrl_t *p_ctrl, adc_cfg_t const *const p_cfg) |
| fsp_err_t | R_SDADC_ScanCfg (adc_ctrl_t *p_ctrl, void const *const p_extend) |
| fsp_err_t | R_SDADC_InfoGet (adc_ctrl_t *p_ctrl, adc_info_t *p_adc_info) |
| fsp_err_t | R_SDADC_ScanStart (adc_ctrl_t *p_ctrl) |

| | |
|---|---|
| fsp_err_t | R_SDADC_ScanStop (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_SDADC_StatusGet (adc_ctrl_t *p_ctrl, adc_status_t *p_status) |
| fsp_err_t | R_SDADC_Read (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data) |
| fsp_err_t | R_SDADC_Read32 (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data) |
| fsp_err_t | R_SDADC_OffsetSet (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t const offset) |
| fsp_err_t | R_SDADC_Calibrate (adc_ctrl_t *const p_ctrl, void *const p_extend) |
| fsp_err_t | R_SDADC_Close (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_SDADC_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Driver for the SDADC24 peripheral on RA MCUs. This module implements the ADC Interface.

# Overview

### Features

The SDADC module supports the following features:

- 24 bit maximum resolution
- Configure scans to include:
  - Multiple analog channels
  - Outputs of OPAMP0 (P side) and OPAMP1 (N side) of SDADC channel 4
- Configurable scan start trigger:
  - Software scan triggers
  - Hardware scan triggers (timer expiration, for example)
- Configurable scan mode:
  - Single scan mode, where each trigger starts a single scan
  - Continuous scan mode, where all channels are scanned continuously
- Supports averaging converted samples
- Optional callback when single conversion, entire scan, or calibration completes
- Supports reading converted data
- Sample and hold support

### Selecting an ADC

All RA MCUs have an Analog to Digital Converter (r_adc). Only select RA MCUs have an SDADC. When selecting between them, consider these factors. Refer to the hardware manual for details.

| | ADC | SDADC |
|---|---|---|

| | | |
|---|---|---|
| Availability | Available on all RA MCUs. | Available on select RA MCUs. |
| Resolution | The ADC has a maximum resolution of 12, 14, or 16 bits depending on the MCU. | The SDADC has a maximum accuracy of 24 bits. |
| Number of Channels | The ADC has more channels than the SDADC. | The SDADC 5 channels, one of which is tied to OPAMP0 and OPAMP1. |
| Frequency | The ADC sampling time is shorter (more samples per second). | The SDADC sampling time is longer (fewer samples per second). |
| Settling Time | The ADC does not have a settling time when switching between channels. | The SDADC requires a settling time when switching between channels. |

# Configuration

### Build Time Configurations for r_sdadc

The following build time configurations are defined in fsp_cfg/r_sdadc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Analog > ADC Driver on r_sdadc

This module can be added to the Stacks tab via New Stack > Driver > Analog > ADC Driver on r_sdadc:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_adc0 | Module name. |
| Mode | • Single Scan<br>• Continuous Scan | 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<br>• 24 Bit | 24 Bit | Select 24-bit or 16-bit resolution. |
| Alignment | • Right<br>• Left | Right | Select left or right alignment. |
| Trigger | MCU Specific Options | | 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<br>• External | Internal | Vref can be source internally and output on the SBIAS pin, or Vref can be input from VREFI. |
| Vref Voltage | • 0.8 V<br>• 1.0 V<br>• 1.2 V<br>• 1.4 V<br>• 1.6 V<br>• 1.8 V<br>• 2.0 V<br>• 2.2 V<br>• 2.4 V | 1.0 V | Select Vref voltage. If Vref is input externally, the voltage on VREFI must match the voltage selected within 3%. |
| Callback | Name must be a valid C symbol | NULL | Enter the name of the callback function to be called when conversion completes or a scan ends. |
| Conversion End Interrupt Priority | MCU Specific Options | | [Required] Select the interrupt priority for the conversion end interrupt. |
| Scan End Interrupt Priority | MCU Specific Options | | [Optional] Select the interrupt priority for the scan end interrupt. |
| Calibration End Interrupt Priority | MCU Specific Options | | [Optional] Select the interrupt priority for the calibration end interrupt. |

**Configurations for Driver > Analog > SDADC Channel Configuration on r_sdadc**

This module can be added to the Stacks tab via New Stack > Driver > Analog > SDADC Channel Configuration on r_sdadc:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Input | • Differential<br>• Single Ended | Differential | Select differential or single-ended input. |
| Stage 1 Gain | • 1<br>• 2<br>• 3<br>• 4<br>• 8 | 1 | Select the gain for stage 1 of the PGA. Must be 1 for single-ended input. |
| Stage 2 Gain | • 1<br>• 2<br>• 4<br>• 8 | 1 | Select the gain for stage 2 of the PGA. Must be 1 for single-ended input. |
| Oversampling | • 64<br>• 128<br>• 256<br>• 512<br>• 1024<br>• 2048 | 256 | Select the oversampling ratio for the PGA. Must be 256 for single-ended input. |
| Polarity (Valid for Single-Ended Input Only) | • Positive<br>• Negative | 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)<br>• Average 8<br>• Average 16<br>• Average 32<br>• Average 64 | Do Not Average (Interrupt after Each Conversion) | 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<br>• Result Inverted | 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^resolution - 1. |
| Number of Conversions Per Scan | Refer to the RA Configuration tool for available options. | 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.

### Clock Configuration

The SDADC clock clock is configurable on the clocks tab.

The SDADC clock must be 4 MHz when the SDADC is used.

### Pin Configuration

The ANSDnP (n = 0-3) pins are analog input channels that can be used with the SDADC.

## Usage Notes

### Scan Procedure

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 FSP 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, SDADC performs the number of conversions requested. These are performed consecutively. There is a settling time associated with switching channels. Performing all of the requested conversions for each channel at a time avoids this settling time after the first conversion.

   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 R_SDADC_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.

### When Interrupts Are Not Enabled

If interrupts are not enabled, the R_SDADC_StatusGet() API can be used to poll the SDADC to determine when the scan has completed. The R_SDADC_Read() API function is used to access the converted SDADC result. This applies to both normal scans and calibration scans.

### Calibration

Calibration is required to use the SDADC if any channel is configured for differential mode. Call R_SDADC_Calibrate() after open, and prior to any other function, then wait for a calibration complete

event before using the SDADC. R_SDADC_Calibrate() should not be called if all channels are configured for single-ended mode.

# Examples

### Basic Example

This is a basic example of minimal use of the SDADC in an application.

```
void sdadc_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_SDADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Calibrate all differential channels. */
 sdadc_calibrate_args_t calibrate_args;
    calibrate_args.mode    = SDADC_CALIBRATION_INTERNAL_GAIN_OFFSET;
    calibrate_args.channel = ADC_CHANNEL_0;
    err = R_SDADC_Calibrate(&g_adc0_ctrl, &calibrate_args);
    handle_error(err);
 /* Wait for calibration to complete. */
 adc_status_t status;
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
    {
 R_SDADC_StatusGet(&g_adc0_ctrl, &status);
    }
 /* In software trigger mode, start a scan by calling R_SDADC_ScanStart(). In other
modes, enable external
  * triggers by calling R_SDADC_ScanStart(). */
    (void) R_SDADC_ScanStart(&g_adc0_ctrl);
 /* Wait for conversion to complete. */
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
    {
```

```
 R_SDADC_StatusGet(&g_adc0_ctrl, &status);
    }
 /* Read converted data. */
    uint32_t channel1_conversion_result;
 R_SDADC_Read32(&g_adc0_ctrl, ADC_CHANNEL_1, &channel1_conversion_result);
}
```

### Using DTC or DMAC with the SDADC

If desired, the DTC or DMAC can be used to store each conversion result in a circular buffer. An example configuration is below.

```
/* Example DTC transfer settings to used with SDADC. */
/* The transfer length should match the total number of conversions per scan. This
example assumes the SDADC is
 * configured to scan channel 1 three times, then channel 2 and channel 4 once, for a
total of 5 conversions. */
#define SDADC_EXAMPLE_TRANSFER_LENGTH (5)
uint32_t g_sdadc_example_buffer[SDADC_EXAMPLE_TRANSFER_LENGTH];
transfer_info_t g_sdadc_transfer_info =
{
    .dest_addr_mode = TRANSFER_ADDR_MODE_INCREMENTED,
    .repeat_area    = TRANSFER_REPEAT_AREA_DESTINATION,
    .irq            = TRANSFER_IRQ_END,
    .chain_mode    = TRANSFER_CHAIN_MODE_DISABLED,
    .src_addr_mode = TRANSFER_ADDR_MODE_FIXED,
    .mode          = TRANSFER_MODE_REPEAT,
 /* NOTE: The data transferred will contain a 24-bit converted value in bits 23:0.
Bit 24 contains a status flag
  * indicating if the result overflowed or not. Bits 27:25 contain the channel number
+ 1. The settings for
  * resolution and alignment and ignored when DTC or DMAC is used. */
    .size          = TRANSFER_SIZE_4_BYTE,
 /* NOTE: It is strongly recommended to enable averaging on all channels or no
channels when using DTC with SDADC
```

```
  * because the result register is different when averaging is used. If averaging is
enabled on all channels,
  * set transfer_info_t::p_src to &R_SDADC->ADAR. */
    .p_src = (void const *) &R_SDADC0->ADCR,
    .p_dest = &g_sdadc_example_buffer[0],
    .length = SDADC_EXAMPLE_TRANSFER_LENGTH,
};
void sdadc_dtc_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_SDADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Calibrate all differential channels. */
 sdadc_calibrate_args_t calibrate_args;
    calibrate_args.mode    = SDADC_CALIBRATION_INTERNAL_GAIN_OFFSET;
    calibrate_args.channel = ADC_CHANNEL_0;
    err = R_SDADC_Calibrate(&g_adc0_ctrl, &calibrate_args);
    handle_error(err);
 /* Wait for calibration to complete. */
 adc_status_t status;
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
    {
 R_SDADC_StatusGet(&g_adc0_ctrl, &status);
    }
 /* In software trigger mode, start a scan by calling R_SDADC_ScanStart(). In other
modes, enable external
  * triggers by calling R_SDADC_ScanStart(). */
    (void) R_SDADC_ScanStart(&g_adc0_ctrl);
 /* After each conversion, the converted data is transferred to the next index in
g_sdadc_example_buffer. After
  * the entire scan completes, the index in g_sdadc_example_buffer resets. The data
```

```
in g_sdadc_example_buffer
  * is:
  * - g_sdadc_example_buffer[0] = SDADC channel 1 conversion 0
  * - g_sdadc_example_buffer[1] = SDADC channel 1 conversion 1
  * - g_sdadc_example_buffer[2] = SDADC channel 1 conversion 2
  * - g_sdadc_example_buffer[3] = SDADC channel 2 conversion 0
  * - g_sdadc_example_buffer[4] = SDADC channel 4 conversion 0
  *//* At any point in the application after the first scan completes, the most
recent data for channel 2 can be read
  * from the buffer like this. Shifting removes the unrelated bits in the result
register and propagates the sign
  * bit so the value can be interpreted as a signed result. This assumes channel 2 is
configured in differential
  * mode. */
    int32_t channel_2_data = (int32_t) (g_sdadc_example_buffer[3] << 8) >> 8;
 FSP_PARAMETER_NOT_USED(channel_2_data);
}
```

## Data Structures

| | |
|---|---|
| struct | sdadc_calibrate_args_t |
| struct | sdadc_channel_cfg_t |
| struct | sdadc_scan_cfg_t |
| struct | sdadc_extended_cfg_t |
| struct | sdadc_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | sdadc_vref_src_t |
| enum | sdadc_vref_voltage_t |
| enum | sdadc_channel_input_t |
| enum | sdadc_channel_stage_1_gain_t |
| enum | sdadc_channel_stage_2_gain_t |
| enum | sdadc_channel_oversampling_t |

| | |
|---|---|
| enum | sdadc_channel_polarity_t |
| enum | sdadc_channel_average_t |
| enum | sdadc_channel_inversion_t |
| enum | sdadc_channel_count_formula_t |
| enum | sdadc_calibration_t |

## Data Structure Documentation

### ◆ sdadc_calibrate_args_t

| struct sdadc_calibrate_args_t | | |
|---|---|---|
| Structure to pass to the adc_api_t::calibrate p_extend argument. | | |
| Data Fields | | |
| adc_channel_t | channel | Which channel to calibrate. |
| sdadc_calibration_t | mode | Calibration mode. |

### ◆ sdadc_channel_cfg_t

| struct sdadc_channel_cfg_t |
|---|
| SDADC per channel configuration. |

### ◆ sdadc_scan_cfg_t

| struct sdadc_scan_cfg_t | | |
|---|---|---|
| SDADC active channel configuration | | |
| Data Fields | | |
| uint32_t | scan_mask | Channels/bits: bit 0 is ch0; bit 15 is ch15. |

### ◆ sdadc_extended_cfg_t

| struct sdadc_extended_cfg_t | | |
|---|---|---|
| SDADC configuration extension. This extension is required and must be provided in adc_cfg_t::p_extend. | | |
| Data Fields | | |
| uint8_t | conv_end_ipl | Conversion end interrupt priority. |
| IRQn_Type | conv_end_irq | |
| 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. |

### ◆ sdadc_instance_ctrl_t

| struct sdadc_instance_ctrl_t |
|---|
| ADC instance control block. DO NOT INITIALIZE. Initialized in adc_api_t::open(). |

## Enumeration Type Documentation

### ◆ sdadc_vref_src_t

| enum sdadc_vref_src_t | |
|---|---|
| Source of Vref. | |
| Enumerator | |
| 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. |

### ◆ sdadc_vref_voltage_t

| enum sdadc_vref_voltage_t | |
|---|---|
| Voltage of Vref. | |
| Enumerator | |
| SDADC_VREF_VOLTAGE_800_MV | Vref is 0.8 V. |
| 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) |

### ◆ sdadc_channel_input_t

| enum sdadc_channel_input_t | |
|---|---|
| Per channel input mode. | |
| Enumerator | |
| SDADC_CHANNEL_INPUT_DIFFERENTIAL | Differential input. |
| SDADC_CHANNEL_INPUT_SINGLE_ENDED | Single-ended input. |

### ◆ sdadc_channel_stage_1_gain_t

| enum sdadc_channel_stage_1_gain_t | |
|---|---|
| Per channel stage 1 gain options. | |
| Enumerator | |
| SDADC_CHANNEL_STAGE_1_GAIN_1 | Gain of 1. |
| SDADC_CHANNEL_STAGE_1_GAIN_2 | Gain of 2. |
| 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. |

### ◆ sdadc_channel_stage_2_gain_t

| enum sdadc_channel_stage_2_gain_t | |
|---|---|
| Per channel stage 2 gain options. | |
| Enumerator | |
| 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. |

### ◆ sdadc_channel_oversampling_t

| enum sdadc_channel_oversampling_t | |
|---|---|
| Per channel oversampling ratio. | |
| Enumerator | |
| 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. |
| SDADC_CHANNEL_OVERSAMPLING_2048 | Oversampling ratio of 2048. |

### ◆ sdadc_channel_polarity_t

| enum sdadc_channel_polarity_t | |
|---|---|
| Per channel polarity, valid for single-ended input only. | |
| Enumerator | |
| SDADC_CHANNEL_POLARITY_POSITIVE | Positive-side single-ended input. |
| SDADC_CHANNEL_POLARITY_NEGATIVE | Negative-side single-ended input. |

### ◆ sdadc_channel_average_t

| enum sdadc_channel_average_t | |
|---|---|
| Per channel number of conversions to average before conversion end callback. | |
| Enumerator | |
| 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. |

### ◆ sdadc_channel_inversion_t

| enum sdadc_channel_inversion_t | |
|---|---|
| Per channel polarity, valid for negative-side single-ended input only. | |
| Enumerator | |
| SDADC_CHANNEL_INVERSION_OFF | Do not invert conversion result. |
| SDADC_CHANNEL_INVERSION_ON | Invert conversion result. |

### ◆ sdadc_channel_count_formula_t

| enum sdadc_channel_count_formula_t | |
|---|---|
| Select a formula to specify the number of conversions. The following symbols are used in the formulas:<br>• N: Number of conversions<br>• n: sdadc_channel_cfg_t::coefficient_n, do not set to 0 if m is 0<br>• m: sdadc_channel_cfg_t::coefficient_m, do not set to 0 if n is 0<br>Either m or n must be non-zero. | |
| Enumerator | |
| SDADC_CHANNEL_COUNT_FORMULA_EXPONENTIAL | N = 32 * (2 ^ n - 1) + m * 2 ^ n. |
| SDADC_CHANNEL_COUNT_FORMULA_LINEAR | N = (32 * n) + m. |

### ◆ sdadc_calibration_t

| enum sdadc_calibration_t | |
|---|---|
| Calibration mode. | |
| Enumerator | |
| SDADC_CALIBRATION_INTERNAL_GAIN_OFFSET | Use internal reference to calibrate offset and gain. |
| SDADC_CALIBRATION_EXTERNAL_OFFSET | Use external reference to calibrate offset. |
| SDADC_CALIBRATION_EXTERNAL_GAIN | Use external reference to calibrate gain. |

## Function Documentation

### ◆ R_SDADC_Open()

fsp_err_t R_SDADC_Open ( adc_ctrl_t * *p_ctrl*, adc_cfg_t const *const *p_cfg* )

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. R_SDADC_Calibrate() must be called after this function before using the SDADC if any channels are used in differential mode. Implements adc_api_t::open().

***Note***

*This function delays at least 2 ms as required by the SDADC power on procedure.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration successful. |
| FSP_ERR_ASSERTION | An input pointer is NULL or an input parameter is invalid. |
| FSP_ERR_ALREADY_OPEN | Control block is already open. |
| FSP_ERR_IRQ_BSP_DISABLED | A required interrupt is disabled |

### ◆ R_SDADC_ScanCfg()

fsp_err_t R_SDADC_ScanCfg ( adc_ctrl_t * *p_ctrl*, void const *const *p_extend* )

Configures the enabled channels of the ADC. Pass a pointer to sdadc_scan_cfg_t to p_extend. Implements adc_api_t::scanCfg().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in p_adc_info. |
| FSP_ERR_ASSERTION | An input pointer is NULL or an input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SDADC_InfoGet()

fsp_err_t R_SDADC_InfoGet ( adc_ctrl_t * *p_ctrl*, adc_info_t * *p_adc_info* )

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 adc_api_t::infoGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in p_adc_info. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SDADC_ScanStart()

fsp_err_t R_SDADC_ScanStart ( adc_ctrl_t * *p_ctrl*)

If the SDADC is configured for hardware triggers, enables hardware triggers. Otherwise, starts a scan. Implements adc_api_t::scanStart().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Scan started or hardware triggers enabled successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |
| FSP_ERR_IN_USE | A conversion or calibration is in progress. |

### ◆ R_SDADC_ScanStop()

fsp_err_t R_SDADC_ScanStop ( adc_ctrl_t * *p_ctrl*)

If the SDADC is configured for hardware triggers, disables hardware triggers. Otherwise, stops any in-progress scan started by software. Implements adc_api_t::scanStop().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Scan stopped or hardware triggers disabled successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SDADC_StatusGet()

fsp_err_t R_SDADC_StatusGet ( adc_ctrl_t * *p_ctrl*, adc_status_t * *p_status* )

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 adc_api_t::scanStatusGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | No software scan or calibration is in progress. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SDADC_Read()

fsp_err_t R_SDADC_Read ( adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint16_t *const *p_data* )

Reads the most recent conversion result from a channel. Truncates 24-bit results to the upper 16 bits. Implements adc_api_t::read().

***Note***

*The result stored in p_data is signed when the SDADC channel is configured in differential mode.*
*Do not use this API if the conversion end interrupt (SDADC0_ADI) is used to trigger the DTC unless the interrupt mode is set to TRANSFER_IRQ_EACH.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Conversion result in p_data. |
| FSP_ERR_ASSERTION | An input pointer was NULL or an input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SDADC_Read32()

fsp_err_t R_SDADC_Read32 ( adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint32_t *const *p_data* )

Reads the most recent conversion result from a channel. Implements adc_api_t::read32().

***Note***

*The result stored in p_data is signed when the SDADC channel is configured in differential mode. When the SDADC is configured for 24-bit resolution and right alignment, the sign bit is bit 23, and the upper 8 bits are 0. When the SDADC is configured for 16-bit resolution and right alignment, the sign bit is bit 15, and the upper 16 bits are 0.*
*Do not use this API if the conversion end interrupt (SDADC0_ADI) is used to trigger the DTC unless the interrupt mode is set to TRANSFER_IRQ_EACH.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Conversion result in p_data. |
| FSP_ERR_ASSERTION | An input pointer was NULL or an input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SDADC_OffsetSet()

fsp_err_t R_SDADC_OffsetSet ( adc_ctrl_t *const *p_ctrl*, adc_channel_t const *reg_id*, int32_t const *offset* )

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 adc_api_t::offsetSet().

Note: The offset is cleared if adc_api_t::calibrate() is called. The offset can be re-applied if necessary after the the callback with event ADC_EVENT_CALIBRATION_COMPLETE is called.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | See p_instance_ctrl in adc_api_t::offsetSet(). |
| [in] | reg_id | See reg_id in adc_api_t::offsetSet(). |
| [in] | offset | Must be between -15 and 15, offset (mV) = 10.9376 mV * offset_steps / stage 1 gain. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Offset updated successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL or an input parameter was invalid. |
| FSP_ERR_IN_USE | A conversion or calibration is in progress. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SDADC_Calibrate()

fsp_err_t R_SDADC_Calibrate ( adc_ctrl_t *const *p_ctrl*, void *const *p_extend* )

Requires sdadc_calibrate_args_t passed to p_extend. Calibrates the specified channel. Calibration is not required or supported for single-ended mode. Calibration must be completed for differential mode before using the SDADC. A callback with the event ADC_EVENT_CALIBRATION_COMPLETE is called when calibration completes. Implements adc_api_t::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 adc_api_t::offsetSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calibration began successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_IN_USE | A conversion or calibration is in progress. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SDADC_Close()

fsp_err_t R_SDADC_Close ( adc_ctrl_t * *p_ctrl*)

Stops any scan in progress, disables interrupts, and powers down the SDADC peripheral. Implements adc_api_t::close().

***Note***

*This function delays at least 3 us as required by the SDADC24 stop procedure.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Instance control block closed successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SDADC_VersionGet()

| fsp_err_t R_SDADC_VersionGet ( fsp_version_t *const *p_version*) |
|---|
| Gets the API and code version. Implements adc_api_t::versionGet(). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version information available in p_version. |
| FSP_ERR_ASSERTION | The parameter p_version is NULL. |

## 4.2.42 SD/MMC Host Interface (r_sdhi)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_SDHI_Open (sdmmc_ctrl_t *const p_api_ctrl, sdmmc_cfg_t const *const p_cfg) |
| fsp_err_t | R_SDHI_MediaInit (sdmmc_ctrl_t *const p_api_ctrl, sdmmc_device_t *const p_device) |
| fsp_err_t | R_SDHI_Read (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count) |
| fsp_err_t | R_SDHI_Write (sdmmc_ctrl_t *const p_api_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count) |
| fsp_err_t | R_SDHI_ReadIo (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address) |
| fsp_err_t | R_SDHI_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) |
| fsp_err_t | R_SDHI_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) |
| fsp_err_t | R_SDHI_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) |

| | |
|---|---|
| fsp_err_t | R_SDHI_IoIntEnable (sdmmc_ctrl_t *const p_api_ctrl, bool enable) |
| fsp_err_t | R_SDHI_StatusGet (sdmmc_ctrl_t *const p_api_ctrl, sdmmc_status_t *const p_status) |
| fsp_err_t | R_SDHI_Erase (sdmmc_ctrl_t *const p_api_ctrl, uint32_t const start_sector, uint32_t const sector_count) |
| fsp_err_t | R_SDHI_Close (sdmmc_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SDHI_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Driver for the SD/MMC Host Interface (SDHI) peripheral on RA MCUs. This module implements the SD/MMC Interface.

# Overview

### Features

- Supports the following memory devices: SDSC (SD Standard Capacity), SDHC (SD High Capacity), and SDXC (SD Extended Capacity)
    - Supports reading, writing and erasing SD memory devices
    - Supports 1-bit or 4-bit bus
    - Supports detection of device write protection (SD cards only)
- Automatically configures the clock to the maximum clock rate supported by both host (MCU) and device
- Supports hardware acceleration using DMAC or DTC
- Supports callback notification when an operation completes or an error occurs

# Configuration

### Build Time Configurations for r_sdhi

The following build time configurations are defined in fsp_cfg/r_sdhi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking Enable | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Unaligned Access Support | • Disabled<br>• Enabled | Enabled | If enabled, code for supporting buffers that are not aligned on a 4-byte boundary is included in the build. Only disable this if all buffers passed to the |

| | | | |
|---|---|---|---|
| | | | driver are 4-byte aligned. |

**Configurations for Driver > Storage > SD/MMC Driver on r_sdhi**

This module can be added to the Stacks tab via New Stack > Driver > Storage > SD/MMC Driver on r_sdhi:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_sdmmc0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Select the channel. |
| Bus Width | MCU Specific Options | | Select the bus width. |
| Block Size | Value must be an integer between 1 and 512 | 512 | Select the media block size. Must be 512 for SD cards or eMMC devices. Must be 1-512 for SDIO. |
| Card Detection | • Not Used<br>• CD Pin | CD Pin | Select the card detection method. |
| Write Protection | • Not Used<br>• WP Pin | WP Pin | Select whether or not to use the write protect pin. Select Not Used if the MCU or device does not have a write protect pin. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |
| Access Interrupt Priority | MCU Specific Options | | Select the access interrupt priority. |
| Card Interrupt Priority | MCU Specific Options | | Select the card interrupt priority. |
| DTC Interrupt Priority | MCU Specific Options | | Select the DTC interrupt priority. |

**Interrupt Configurations:**

The following interrupts are required to use the r_sdhi module:

Using SD/MMC with DTC:

- Access Interrupt
- DTC Interrupt

Using SD/MMC with DMAC:

- Access Interrupt
- DMAC Interrupt (in DMAC instance)

The Card interrupt is optional and only available on MCU packages that have the SDnCD pin (n = channel number).

### Clock Configuration

The SDMMC MCU peripheral (SDHI) uses the PCLKA for its clock source. The SDMMC driver selects the optimal built-in divider based on the PCLKA frequency and the maximum clock rate allowed by the device obtained at media initialization.

### Pin Configuration

The SDMMC driver supports the following pins (n = channel number):

- SDnCLK
- SDnCMD
- SDnDAT0
- SDnDAT1
- SDnDAT2
- SDnDAT3
- SDnCD (not available on all MCUs)
- SDnWP

The drive capacity for each pin should be set to "Medium" or "High" for most hardware designs. This can be configured in the **Pins** tab of the RA Configuration editor by selecting the pin under Pin Selection -> Ports.

## Usage Notes

### Card Detection

When Card Detection is configured to "CD Pin" in the RA Configuration editor, card detection is enabled during R_SDHI_Open().

R_SDHI_StatusGet() can be called to retrieve the current status of the card (including whether a card is present). If the Card Interrupt Priority is enabled, a callback is called when a card is inserted or removed.

If a card is removed and reinserted, R_SDHI_MediaInit() must be called before reading from the card or writing to the card.

### DMA Request 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 DMA Request interrupt. This blocks all other interrupts that are a lower or equal priority to the access interrupt until the software copy is complete.

### Timing Notes for R_SDHI_MediaInit

The R_SDHI_MediaInit() 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.

### Limitations

Developers should be aware of the following limitations when using the SDHI:

### Blocking Calls

The following functions block execution until the response is received for at least one command:

- R_SDHI_MediaInit
- R_SDHI_ReadIo
- R_SDHI_WriteIo
- R_SDHI_ReadIoExt
- R_SDHI_WriteIoExt
- R_SDHI_Erase

Once the function returns the status of the operation can be determined via R_SDHI_StatusGet or through receipt of a callback.

*Note*

*Due to the variability in clocking configurations it is recommended to determine blocking delays experimentally on the target system.*

### 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_sdhi 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 and a software copy is used to move data to the destination buffer.

## Examples

### Basic Example

This is a basic example of minimal use of the r_sdhi in an application.

```
uint8_t g_dest[SDHI_MAX_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint8_t g_src[SDHI_MAX_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint32_t g_transfer_complete = 0;
void r_sdhi_basic_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < SDHI_MAX_BLOCK_SIZE; i++)
```

```
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the SDHI driver. */
 fsp_err_t err = R_SDHI_Open(&g_sdmmc0_ctrl, &g_sdmmc0_cfg);
    handle_error(err);
 /* A device shall be ready to accept the first command within 1ms from detecting VDD
min. Reference section 6.4.1.1
  * "Power Up Time of Card" in the SD Physical Layer Simplified Specification Version
6.00. */
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
 /* Initialize the SD card. This should not be done until the card is plugged in for
SD devices. */
    err = R_SDHI_MediaInit(&g_sdmmc0_ctrl, NULL);
    handle_error(err);
    err = R_SDHI_Write(&g_sdmmc0_ctrl, g_src, 3, 1);
    handle_error(err);
 while (!g_transfer_complete)
    {
 /* Wait for transfer. */
    }
    err = R_SDHI_Read(&g_sdmmc0_ctrl, g_dest, 3, 1);
    handle_error(err);
 while (!g_transfer_complete)
    {
 /* Wait for transfer. */
    }
}
/* The callback is called when a transfer completes. */
void r_sdhi_example_callback (sdmmc_callback_args_t * p_args)
{
 if (SDMMC_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_transfer_complete = 1;
```

```
    }
}
```

### Card Detection Example

This is an example of using SDHI when the card may not be plugged in. The card detection interrupt must be enabled to use this example.

```
bool g_card_inserted = false;
void r_sdhi_card_detect_example (void)
{
 /* Open the SDHI driver. This enables the card detection interrupt. */
 fsp_err_t err = R_SDHI_Open(&g_sdmmc0_ctrl, &g_sdmmc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Check if card is inserted. */
 sdmmc_status_t status;
    err = R_SDHI_StatusGet(&g_sdmmc0_ctrl, &status);
    handle_error(err);
 if (!status.card_inserted)
    {
 while (!g_card_inserted)
      {
 /* Wait for a card insertion interrupt. */
      }
    }
 /* A device shall be ready to accept the first command within 1ms from detecting VDD
min. Reference section 6.4.1.1
  * "Power Up Time of Card" in the SD Physical Layer Simplified Specification Version
6.00. */
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
 /* Initialize the SD card after card insertion is detected. */
    err = R_SDHI_MediaInit(&g_sdmmc0_ctrl, NULL);
    handle_error(err);
}
```

```
/* The callback is called when a card detection event occurs if the card detection
interrupt is enabled. */
void r_sdhi_card_detect_example_callback (sdmmc_callback_args_t * p_args)
{
 if (SDMMC_EVENT_CARD_INSERTED == p_args->event)
    {
       g_card_inserted = true;
    }
 if (SDMMC_EVENT_CARD_REMOVED == p_args->event)
    {
       g_card_inserted = false;
    }
}
```

## Function Documentation

### ◆ R_SDHI_Open()

fsp_err_t R_SDHI_Open ( sdmmc_ctrl_t *const *p_api_ctrl*, sdmmc_cfg_t const *const *p_cfg* )

Opens the driver. Resets SDHI, and enables card detection interrupts if card detection is enabled. R_SDHI_MediaInit must be called after this function before any other functions can be used.

Implements sdmmc_api_t::open().

Example:

```
/* Open the SDHI driver. */
fsp_err_t err = R_SDHI_Open(&g_sdmmc0_ctrl, &g_sdmmc0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is now open. |
| FSP_ERR_ASSERTION | Null Pointer or block size is not in the valid range of 1-512. Block size must be 512 bytes for SD cards and eMMC devices. It is configurable for SDIO only. |
| FSP_ERR_ALREADY_OPEN | Driver has already been opened with this instance of the control structure. |
| FSP_ERR_IRQ_BSP_DISABLED | Access interrupt is not enabled. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Requested channel does not exist on this MCU. |

### ◆ R_SDHI_MediaInit()

fsp_err_t R_SDHI_MediaInit ( sdmmc_ctrl_t *const *p_api_ctrl*, sdmmc_device_t *const *p_device* )

Initializes the SDHI hardware and completes identification and configuration for the SD or eMMC device. This procedure requires several sequential commands. This function blocks until all identification and configuration commands are complete.

Implements sdmmc_api_t::mediaInit().

Example:

```
 /* A device shall be ready to accept the first command within 1ms from detecting VDD min. Reference section 6.4.1.1
  * "Power Up Time of Card" in the SD Physical Layer Simplified Specification Version 6.00. */
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
 /* Initialize the SD card. This should not be done until the card is plugged in for SD devices. */
    err = R_SDHI_MediaInit(&g_sdmmc0_ctrl, NULL);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is now ready for read/write access. |
| FSP_ERR_ASSERTION | Null Pointer or block size is not in the valid range of 1-512. Block size must be 512 bytes for SD cards and eMMC devices. It is configurable for SDIO only. |
| FSP_ERR_NOT_OPEN | Driver has not been initialized. |
| FSP_ERR_CARD_INIT_FAILED | Device was not identified as an SD card, eMMC device, or SDIO card. |
| FSP_ERR_RESPONSE | Device did not respond or responded with an error. |
| FSP_ERR_DEVICE_BUSY | Device is holding DAT0 low (device is busy) or another operation is ongoing. |

### ◆ R_SDHI_Read()

fsp_err_t R_SDHI_Read ( sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *start_sector*, uint32_t const *sector_count* )

Reads data from an SD or eMMC device. Up to 0x10000 sectors can be read at a time. Implements sdmmc_api_t::read().

A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the read data is available.

Example:

```
err = R_SDHI_Read(&g_sdmmc0_ctrl, g_dest, 3, 1);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read successfully. |
| FSP_ERR_ASSERTION | NULL pointer. |
| FSP_ERR_NOT_OPEN | Driver has not been initialized. |
| FSP_ERR_CARD_NOT_INITIALIZED | Card was unplugged. |
| FSP_ERR_DEVICE_BUSY | Driver is busy with a previous operation. |

### ◆ R_SDHI_Write()

fsp_err_t R_SDHI_Write ( sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_source*, uint32_t const *start_sector*, uint32_t const *sector_count* )

Writes data to an SD or eMMC device. Up to 0x10000 sectors can be written at a time. Implements sdmmc_api_t::write().

A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the all data has been written and the device is no longer holding DAT0 low to indicate it is busy.

Example:

```
err = R_SDHI_Write(&g_sdmmc0_ctrl, g_src, 3, 1);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Card write finished successfully. |
| FSP_ERR_ASSERTION | Handle or Source address is NULL. |
| FSP_ERR_NOT_OPEN | Driver has not been initialized. |
| FSP_ERR_CARD_NOT_INITIALIZED | Card was unplugged. |
| FSP_ERR_DEVICE_BUSY | Driver is busy with a previous operation. |
| FSP_ERR_CARD_WRITE_PROTECTED | SD card is Write Protected. |
| FSP_ERR_WRITE_FAILED | Write operation failed. |

### ◆ R_SDHI_ReadIo()

fsp_err_t R_SDHI_ReadIo ( sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_data*, uint32_t const *function*, uint32_t const *address* )

The Read function reads a one byte register from an SDIO card. Implements sdmmc_api_t::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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read successfully. |
| FSP_ERR_ASSERTION | NULL pointer. |
| FSP_ERR_NOT_OPEN | Driver has not been initialized. |
| FSP_ERR_CARD_NOT_INITIALIZED | Card was unplugged. |
| FSP_ERR_UNSUPPORTED | SDIO support disabled in SDHI_CFG_SDIO_SUPPORT_ENABLE. |
| FSP_ERR_RESPONSE | Device did not respond or responded with an error. |
| FSP_ERR_DEVICE_BUSY | Device is holding DAT0 low (device is busy) or another operation is ongoing. |

### ◆ R_SDHI_WriteIo()

fsp_err_t R_SDHI_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* )

Writes a one byte register to an SDIO card. Implements sdmmc_api_t::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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Card write finished successfully. |
| FSP_ERR_ASSERTION | Handle or Source address is NULL. |
| FSP_ERR_NOT_OPEN | Driver has not been initialized. |
| FSP_ERR_CARD_NOT_INITIALIZED | Card was unplugged. |
| FSP_ERR_WRITE_FAILED | Write operation failed. |
| FSP_ERR_UNSUPPORTED | SDIO support disabled in SDHI_CFG_SDIO_SUPPORT_ENABLE. |
| FSP_ERR_RESPONSE | Device did not respond or responded with an error. |
| FSP_ERR_DEVICE_BUSY | Device is holding DAT0 low (device is busy) or another operation is ongoing. |

### ◆ R_SDHI_ReadIoExt()

fsp_err_t R_SDHI_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* )

Reads data from an SDIO card function. Implements sdmmc_api_t::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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read successfully. |
| FSP_ERR_ASSERTION | NULL pointer, or count is not in the valid range of 1-512 for byte mode or 1-511 for block mode. |
| FSP_ERR_NOT_OPEN | Driver has not been initialized. |
| FSP_ERR_CARD_NOT_INITIALIZED | Card was unplugged. |
| FSP_ERR_DEVICE_BUSY | Driver is busy with a previous operation. |
| FSP_ERR_UNSUPPORTED | SDIO support disabled in SDHI_CFG_SDIO_SUPPORT_ENABLE. |

### ◆ R_SDHI_WriteIoExt()

fsp_err_t R_SDHI_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* )

Writes data to an SDIO card function. Implements sdmmc_api_t::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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Card write finished successfully. |
| FSP_ERR_ASSERTION | NULL pointer, or count is not in the valid range of 1-512 for byte mode or 1-511 for block mode. |
| FSP_ERR_NOT_OPEN | Driver has not been initialized. |
| FSP_ERR_CARD_NOT_INITIALIZED | Card was unplugged. |
| FSP_ERR_DEVICE_BUSY | Driver is busy with a previous operation. |
| FSP_ERR_WRITE_FAILED | Write operation failed. |
| FSP_ERR_UNSUPPORTED | SDIO support disabled in SDHI_CFG_SDIO_SUPPORT_ENABLE. |

### ◆ R_SDHI_IoIntEnable()

fsp_err_t R_SDHI_IoIntEnable ( sdmmc_ctrl_t *const *p_api_ctrl*, bool *enable* )

Enables or disables the SDIO Interrupt. Implements sdmmc_api_t::ioIntEnable().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Card enabled or disabled SDIO interrupts successfully. |
| FSP_ERR_NOT_OPEN | Driver has not been initialized. |
| FSP_ERR_ASSERTION | NULL pointer. |
| FSP_ERR_DEVICE_BUSY | Driver is busy with a previous operation. |
| FSP_ERR_UNSUPPORTED | SDIO support disabled in SDHI_CFG_SDIO_SUPPORT_ENABLE. |

### ◆ R_SDHI_StatusGet()

fsp_err_t R_SDHI_StatusGet ( sdmmc_ctrl_t *const *p_api_ctrl*, sdmmc_status_t *const *p_status* )

Provides driver status. Implements sdmmc_api_t::statusGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Status stored in p_status. |
| FSP_ERR_ASSERTION | NULL pointer. |
| FSP_ERR_NOT_OPEN | Driver has not been initialized. |

### ◆ R_SDHI_Erase()

fsp_err_t R_SDHI_Erase ( sdmmc_ctrl_t *const *p_api_ctrl*, uint32_t const *start_sector*, uint32_t const *sector_count* )

Erases sectors of an SD card or eMMC device. Implements sdmmc_api_t::erase().

This function blocks until the erase command is sent. Poll the status to determine when erase is complete.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Erase operation requested. |
| FSP_ERR_ASSERTION | A required pointer is NULL or an argument is invalid. |
| FSP_ERR_NOT_OPEN | Driver has not been initialized. |
| FSP_ERR_CARD_NOT_INITIALIZED | Card was unplugged. |
| FSP_ERR_CARD_WRITE_PROTECTED | SD card is Write Protected. |
| FSP_ERR_RESPONSE | Device did not respond or responded with an error. |
| FSP_ERR_DEVICE_BUSY | Device is holding DAT0 low (device is busy) or another operation is ongoing. |

### ◆ R_SDHI_Close()

fsp_err_t R_SDHI_Close ( sdmmc_ctrl_t *const *p_api_ctrl*)

Closes an open SD/MMC device. Implements sdmmc_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Driver has not been initialized. |

### ◆ R_SDHI_VersionGet()

fsp_err_t R_SDHI_VersionGet ( fsp_version_t *const *p_version*)

Returns the version of the firmware and API. Implements sdmmc_api_t::versionGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed successfully. |
| FSP_ERR_ASSERTION | Null Pointer. |

## 4.2.43 Segment LCD Controller (r_slcdc)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_SLCDC_Open (slcdc_ctrl_t *const p_ctrl, slcdc_cfg_t const *const p_cfg) |
| fsp_err_t | R_SLCDC_Write (slcdc_ctrl_t *const p_ctrl, uint8_t const start_segment, uint8_t const *p_data, uint8_t const segment_count) |
| fsp_err_t | R_SLCDC_Modify (slcdc_ctrl_t *const p_ctrl, uint8_t const segment_number, uint8_t const data_mask, uint8_t const data) |
| fsp_err_t | R_SLCDC_Start (slcdc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_SLCDC_Stop (slcdc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_SLCDC_SetContrast (slcdc_ctrl_t *const p_ctrl, slcdc_contrast_t |

| | |
|---|---|
| | const contrast) |
| fsp_err_t | R_SLCDC_SetDisplayArea (slcdc_ctrl_t *const p_ctrl, slcdc_display_area_t const display_area) |
| fsp_err_t | R_SLCDC_Close (slcdc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_SLCDC_VersionGet (fsp_version_t *p_version) |

### Detailed Description

Driver for the SLCDC peripheral on RA MCUs. This module implements the SLCDC Interface.

# Overview

The segment LCD controller (SLCDC) utilizes two to four reference voltages to provide AC signals for driving traditional segment LCD panels. Depending on the LCD and MCU package, up to 272 segments can be driven. A built-in link to the RTC allows for up to 152 segments to switch between two patterns at regular intervals. An on-chip boost driver can be used to provide configurable reference voltages up to 5.25V allowing for simple contrast adjustment.

### Features

The SLCDC module can perform the following functions:

- Initialize, start and stop the SLCDC
- Set and modify the output pattern
- Blink between two patterns based on a periodic RTC interrupt signal
- Adjust display contrast (only when using internal voltage boosting)

# Configuration

### Build Time Configurations for r_slcdc

The following build time configurations are defined in fsp_cfg/r_slcdc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Driver > Graphics > Segment LCD Driver on r_slcdc

This module can be added to the Stacks tab via New Stack > Driver > Graphics > Segment LCD Driver on r_slcdc:

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| General > Name | Name must be a valid C symbol | g_slcdc0 | Module Name |
| Clock > Source | • LOCO<br>• SOSC<br>• MOSC<br>• HOCO | HOCO | Select the clock source. |
| Clock > Divisor | Refer to the RA Configuration tool for available options. | (HOCO/MOSC) 16384 | Select the clock divisor. |
| Output > Bias method | • 1/2 bias<br>• 1/3 bias<br>• 1/4 bias | 1/2 bias | Select the bias method. This determines the number of voltage levels used to create the waveforms. |
| Output > Timeslice | • Static<br>• 2-slice<br>• 3-slice<br>• 4-slice<br>• 8-slice | Static | Select the LCD time slice. The number of slices should match the number of common (COM) pins for your LCD panel. |
| Output > Waveform | • Waveform A<br>• Waveform B | Waveform A | Select the LCD waveform. |
| Output > Drive method | • External resistance division<br>• Internal voltage boosting<br>• Capacitor split | External resistance division | Select the LCD drive method. |
| Output > Default contrast | Refer to the RA Configuration tool for available options. | 0 | Select the default contrast level. |

### Valid Configurations

Though there are many setting combinations only a limited subset are supported by the SLCDC peripheral hardware:

| Waveform | Slices | Bias | External Resistance | Internal Boost | Capacitor Split |
|---|---|---|---|---|---|
| A | 8 | 1/4 | Available | Available | — |
| A | 4 | 1/3 | Available | Available | Available |
| A | 3 | 1/3 | Available | Available | Available |
| A | 3 | 1/2 | Available | — | — |
| A | 2 | 1/2 | Available | — | — |
| A | Static | — | Available | — | — |
| B | 8 | 1/4 | Available | Available | Available |

| | | | | | |
|---|---|---|---|---|---|
| B | 4 | 1/3 | Available | Available | — |

### Clock Configuration

The SLCDC clock can be sourced from the main clock (MOSC), sub-clock (SOSC), HOCO or LOCO. Dividers of 4 to 1024 are available for SOSC/LOCO and 256 to 524288 for MOSC/HOCO. It is recommended to adjust the divisor such that the resulting clock provides a frame frequency of 32-128 Hz.

*Note*

*Make sure your desired source clock is enabled and running before starting SLCDC output.*
*Do not set the segment LCD clock over 512 Hz when using internal boost or capacitor split modes.*

### Pin Configuration

This module controls a variety of pins necessary for segment LCD voltage generation and signal output:

| Pin Name | Function | Notes |
|---|---|---|
| SEGn | Segment data output | Connect these signals to the segment pins of the LCD. |
| COMn | Common signal output | Connect these signals to the common pins of the LCD. |
| VLn | Voltage reference | These pins should be connected to passive components based on the selected drive method (see section 45.7 "Supplying LCD Drive Voltages VL1, VL2, VL3, and VL4" in the RA4M1 User's Manual (R01UH0887EJ0100)). |
| CAPH, CAPL | Drive voltage generator capacitor | Connect a nonpolar 0.47uF capacitor across these pins when using internal boost or capacitor split modes. This pin is not needed when using resistance division. |

### Interrupt Configuration

The SLCDC provides no interrupt signals.

*Note*

*Blinking output timing is driven directly from the RTC periodic interrupt. Once the interrupt is enabled setting the display to SLCDC_DISP_BLINK will swap between A- and B-pattern each time it occurs. The ELC is not required for this functionality.*

## Usage Notes

### Limitations

Developers should be aware of the following limitations when using the SLCDC:

- Different packages provide different numbers of segment pins. Check the User's Manual for your device to confirm availability and mapping of segment signals.
- When using internal boost mode a delay of 5ms is required between calling R_SLCDC_Open and R_SLCDC_Start to allow the boost circuit to charge.
- When using the internal boost or capacitor split method do not set the segment LCD clock higher than 512 Hz.

# Examples

### Basic Example

Below is a basic example of minimal use of the SLCDC in an application. The SLCDC driver is initialized, output is started and a pattern is written to the segment registers.

```
void slcdc_init (void)
{
 fsp_err_t err;
 /* Open SLCDC driver */
    err = R_SLCDC_Open(&g_slcdc_ctrl, &g_slcdc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* When using internal boost mode this delay is required to allow the boost circuit
to charge. See RA4M1 User's
  * Manual (R01UH0887EJ0100) 8.2.18 "Segment LCD Source Clock Control Register
(SLCDSCKCR)" for details. */
 R_BSP_SoftwareDelay(5, BSP_DELAY_UNITS_MILLISECONDS);
 /* Start SLCDC output */
    err = R_SLCDC_Start(&g_slcdc_ctrl);
    handle_error(err);
 /* Write pattern to display */
    err = R_SLCDC_Write(&g_slcdc_ctrl, 0, segment_data, NUM_SEGMENTS);
    handle_error(err);
}
```

*Note*

*While the SLCDC is running, pattern data is constantly being output. No latching or buffering is required when writing or reading segment data.*

### Blinking Output

This example demonstrates how to set up blinking output using the RTC periodic interrupt. In this example it is assumed that the SLCDC has already been started.

```
void slcdc_blink (void)
{
 fsp_err_t err;
 /* Open RTC and set time/date */
    err = R_RTC_Open(&r_rtc_ctrl, &r_rtc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
    err = R_RTC_CalendarTimeSet(&r_rtc_ctrl, &g_rtc_time);
    handle_error(err);
 /* Set RTC periodic interrupt to 2 Hz (display blink cycle will be 1 Hz) */
    err = R_RTC_PeriodicIrqRateSet(&r_rtc_ctrl,
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_2_SECOND);
    handle_error(err);
 /* Set display to blink */
    err = R_SLCDC_SetDisplayArea(&g_slcdc_ctrl, SLCDC_DISP_BLINK);
    handle_error(err);
 /* Display will now continuously blink */
}
```

## Data Structures

| | |
|---|---|
| struct | slcdc_instance_ctrl_t |

## Data Structure Documentation

### ◆ slcdc_instance_ctrl_t

| |
|---|
| struct slcdc_instance_ctrl_t |
| SLCDC control block. DO NOT INITIALIZE. Initialization occurs when slcdc_api_t::open is called |

## Function Documentation

### ◆ R_SLCDC_Open()

fsp_err_t R_SLCDC_Open ( slcdc_ctrl_t *const *p_ctrl*, slcdc_cfg_t const *const *p_cfg* )

Opens the SLCDC driver. Implements slcdc_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was opened successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block or the configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_UNSUPPORTED | Invalid display mode. |

### ◆ R_SLCDC_Write()

fsp_err_t R_SLCDC_Write ( slcdc_ctrl_t *const *p_ctrl*, uint8_t const *start_segment*, uint8_t const * *p_data*, uint8_t const *segment_count* )

Writes a sequence of display data to the segment data registers. Implements slcdc_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data was written successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block or data is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Segment index is (or will be) out of range. |
| FSP_ERR_NOT_OPEN | Device is not opened or initialized. |

### ◆ R_SLCDC_Modify()

fsp_err_t R_SLCDC_Modify ( slcdc_ctrl_t *const *p_ctrl*, uint8_t const *segment*, uint8_t const *data*, uint8_t const *data_mask* )

Modifies a single segment register based on a mask and the desired data. Implements slcdc_api_t::modify.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was opened successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block structure is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Invalid parameter in the argument. |
| FSP_ERR_NOT_OPEN | Device is not opened or initialized |

### ◆ R_SLCDC_Start()

fsp_err_t R_SLCDC_Start ( slcdc_ctrl_t *const *p_ctrl*)

Starts output of LCD signals. Implements slcdc_api_t::start.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was opened successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block structure is NULL. |
| FSP_ERR_NOT_OPEN | Device is not opened or initialized |

### ◆ R_SLCDC_Stop()

fsp_err_t R_SLCDC_Stop ( slcdc_ctrl_t *const *p_ctrl*)

Stops output of LCD signals. Implements slcdc_api_t::stop.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was opened successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block structure is NULL. |
| FSP_ERR_NOT_OPEN | Device is not opened or initialized |

### ◆ R_SLCDC_SetContrast()

fsp_err_t R_SLCDC_SetContrast ( slcdc_ctrl_t *const *p_ctrl*, slcdc_contrast_t const *contrast* )

Sets contrast to the specified level. Implements slcdc_api_t::setContrast.

***Note***

*Contrast can be adjusted when the SLCDC is operating in internal boost mode only. The range of values is 0-5 when 1/4 bias setting is used and 0-15 otherwise. See RA4M1 User's Manual (R01UH0887EJ0100) section 45.2.4 "LCD Boost Level Control Register (VLCD)" for voltage levels at each setting.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was opened successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block structure is NULL. |
| FSP_ERR_NOT_OPEN | Device is not opened or initialized |
| FSP_ERR_UNSUPPORTED | Unsupported operation |

### ◆ R_SLCDC_SetDisplayArea()

fsp_err_t R_SLCDC_SetDisplayArea ( slcdc_ctrl_t *const *p_ctrl*, slcdc_display_area_t const *display_area* )

Sets output to Waveform A, Waveform B or blinking output. Implements slcdc_api_t::setDisplayArea .

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was opened successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block structure is NULL. |
| FSP_ERR_UNSUPPORTED | Pattern selection has no effect in 8-time-slice mode. |
| FSP_ERR_NOT_OPEN | Device is not opened or initialized. |

### ◆ R_SLCDC_Close()

fsp_err_t R_SLCDC_Close ( slcdc_ctrl_t *const *p_ctrl*)

Closes the SLCDC driver. Implements slcdc_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was closed successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block structure is NULL. |
| FSP_ERR_NOT_OPEN | Device is not opened or initialized |

### ◆ R_SLCDC_VersionGet()

fsp_err_t R_SLCDC_VersionGet ( fsp_version_t *const *p_version*)

Retrieve the API version number. Implements slcdc_api_t::versionGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful return. |
| FSP_ERR_ASSERTION | p_version is NULL. |

## 4.2.44 Serial Peripheral Interface (r_spi)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | R_SPI_Open (spi_ctrl_t *p_api_ctrl, spi_cfg_t const *const p_cfg) |
| fsp_err_t | R_SPI_Read (spi_ctrl_t *const p_api_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SPI_Write (spi_ctrl_t *const p_api_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SPI_WriteRead (spi_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SPI_Close (spi_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SPI_VersionGet (fsp_version_t *p_version) |
| fsp_err_t | R_SPI_CalculateBitrate (uint32_t bitrate, rspck_div_setting_t *spck_div) |

### Detailed Description

Driver for the SPI peripheral on RA MCUs. This module implements the SPI Interface.

# Overview

### Features

- Standard SPI Modes
  - Master or Slave Mode
  - Clock Polarity (CPOL)
    - CPOL=0 SCLK is low when idle
    - CPOL=1 SCLK is high when idle
  - Clock Phase (CPHA)
    - CPHA=0 Data Sampled on the even edge of SCLK (Master Mode Only)
    - CPHA=1 Data Sampled on the odd edge of SCLK
  - MSB/LSB first
  - 8-Bit, 16-Bit, 32-Bit data frames
    - Hardware endian swap in 16-Bit and 32-Bit mode
  - 3-Wire (clock synchronous) or 4-Wire (SPI) Mode
- Configurable bitrate
- Supports Full Duplex or Transmit Only Mode
- DTC Support
- Callback Events
  - Transfer Complete
  - RX Overflow Error (The SPI shift register is copied to the data register before previous data was read)

- TX Underrun Error (No data to load into shift register for transmitting)
- Parity Error (When parity is enabled and a parity error is detected)

# Configuration

### Build Time Configurations for r_spi

The following build time configurations are defined in fsp_cfg/r_spi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Enable Support for using DTC | • Enabled<br>• Disabled | Enabled | If enabled, DTC instances will be included in the build for both transmission and reception. |
| Enable Transmitting from RXI Interrupt | • Enabled<br>• Disabled | Disabled | If enabled, all operations will be handled from the RX (receive) interrupt. This setting only provides a performance boost when DTC is not used. In addition, Transmit Only mode is not supported when this configuration is enabled. |

### Configurations for Driver > Connectivity > SPI Driver on r_spi

This module can be added to the Stacks tab via New Stack > Driver > Connectivity > SPI Driver on r_spi:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_spi0 | Module name. |
| Channel | Select channel 0 or channel 1 | 0 | Select the SPI channel. |
| Receive Interrupt Priority | MCU Specific Options | | Select the interrupt priority for all SPI interrupts. |
| Transmit Buffer Empty Interrupt Priority | MCU Specific Options | | Select the interrupt priority for all SPI |

| | | | |
|---|---|---|---|
| | | | interrupts. |
| Transfer Complete Interrupt Priority | MCU Specific Options | | Select the interrupt priority for all SPI interrupts. |
| Error Interrupt Priority | MCU Specific Options | | Select the interrupt priority for all SPI interrupts. |
| Operating Mode | • Master<br>• Slave | Master | Select the SPI operating mode. |
| Clock Phase | • Data sampling on odd edge, data variation on even edge<br>• Data sampling on even edge, data variation on odd edge | Data sampling on odd edge, data variation on even edge | Select the clock edge to sample data. |
| Clock Polarity | • Low when idle<br>• High when idle | Low when idle | Select clock level when idle. |
| Mode Fault Error | • Enable<br>• Disable | Disable | Detect master/slave mode conflicts. |
| Bit Order | • MSB First<br>• LSB First | MSB First | Select the data bit order. |
| Callback | Name must be a valid C symbol | spi_callback | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |
| SPI Mode | • SPI Operation<br>• Clock Synchronous Operation | Clock Synchronous Operation | Select the clock sync mode. |
| Full or Transmit Only Mode | • Full Duplex<br>• Transmit Only | Full Duplex | Select Full Duplex or Transmit Only Mode. |
| Slave Select Polarity | • Active Low<br>• Active High | Active Low | Select the slave select active level. |
| Select SSL(Slave Select) | • SSL0<br>• SSL1<br>• SSL2<br>• SSL3 | SSL0 | Select which slave to use. |
| MOSI Idle State | • MOSI Idle Value Fixing Disable<br>• MOSI Idle Value Fixing Low | MOSI Idle Value Fixing Disable | Select the MOSI idle level if MOSI idle is enabled. |

| | | | |
|---|---|---|---|
| | • MOSI Idle Value Fixing High | | |
| Parity Mode | • Parity Disabled<br>• Parity Odd<br>• Parity Even | Parity Disabled | Select the parity mode if parity is enabled. |
| Byte Swapping | • Disable<br>• Enable | Disable | Select the byte swap mode for 16/32-Bit Data Frames. |
| Bitrate | Value must be an integer greater than 0 | 30000000 | Enter the desired bitrate. |
| Clock Delay | • SPI_DELAY_COUNT_1<br>• SPI_DELAY_COUNT_2<br>• SPI_DELAY_COUNT_3<br>• SPI_DELAY_COUNT_4<br>• SPI_DELAY_COUNT_5<br>• SPI_DELAY_COUNT_6<br>• SPI_DELAY_COUNT_7<br>• SPI_DELAY_COUNT_8 | SPI_DELAY_COUNT_1 | Configure the number of SPI clock cycles before each data frame. |
| SSL Negation Delay | • SPI_DELAY_COUNT_1<br>• SPI_DELAY_COUNT_2<br>• SPI_DELAY_COUNT_3<br>• SPI_DELAY_COUNT_4<br>• SPI_DELAY_COUNT_5<br>• SPI_DELAY_COUNT_6<br>• SPI_DELAY_COUNT_7<br>• SPI_DELAY_COUNT_8 | SPI_DELAY_COUNT_1 | Configure the number of SPI clock cycles after each data frame. |
| Next Access Delay | • SPI_DELAY_COUNT_1<br>• SPI_DELAY_COUNT_2<br>• SPI_DELAY_COUNT_3<br>• SPI_DELAY_COUNT_4<br>• SPI_DELAY_COU | SPI_DELAY_COUNT_1 | Configure the number of SPI clock cycles between each data frame. |

NT_5
- SPI_DELAY_COUNT_6
- SPI_DELAY_COUNT_7
- SPI_DELAY_COUNT_8

### Clock Configuration

The SPI clock is derived from the following peripheral clock on each device.

| MCU | Peripheral Clock |
|---|---|
| RA2A1 | PCLKB |
| RA4M1 | PCLKA |
| RA6M1 | PCLKA |
| RA6M2 | PCLKA |
| RA6M3 | PCLKA |

### Pin Configuration

This module uses MOSI, MISO, RSPCK, and SSL pins to communicate with on board devices.

*Note*

*At high bitrates, it might be nessecary to configure the pins with IOPORT_CFG_DRIVE_HIGH.*

## Usage Notes

### Performance

At high bitrates, interrupts may not be able to service transfers fast enough. In master mode this means there will be a delay between each data frame. In slave mode this could result in TX Underrun and RX Overflow errors.

In order to improve performance at high bitrates, it is recommended that the instance be configured to service transfers using the DTC.

Another way to improve performance is to transfer the data in 16/32 bit wide data frames when possible. A typical use-case where this is possible is when reading/writing to a block device.

### Transmit From RXI Interrupt

After every data frame the SPI peripheral generates a transmit buffer empty interrupt and a receive buffer full interrupt. It is possible to configure the driver to handle transmit buffer empty interrupts in the receive buffer full isr. This only improves performance when the DTC is not being used.

*Note*

*Configuring the module to use RX DTC instance without also providing a TX DTC instance results in an invalid configuration when RXI transmit is enabled.*
*Transmit Only mode is not supported when Transmit from RXI is enabled.*

### Clock Auto-Stopping

In master mode, if the Receive Buffer Full Interrupts are not handled fast enough, instead of generating a RX Overflow error, the last clock cycle will be stretched until the receive buffer is read.

### Parity Mode

When parity mode is configured, the LSB of each data frame is used as a parity bit. When odd parity is selected, the LSB is set such that there are an odd number of ones in the data frame. When even parity is selected, the LSB is set such that there are an even number of ones in the data frame.

### Limitations

Developers should be aware of the following limitations when using the SPI:

- In master mode, the driver will only configure 4-Wire mode if the device supports SSL Level Keeping (SSLKP bit in SPCMD0) and will return FSP_ERR_UNSUPPORTED if configured for 4-Wire mode on devices without SSL Level Keeping. Without SSL Level Keeping, the SSL pin is toggled after every data frame. In most cases this is not desirable behavior so it is recommended that the SSL pin be driven in software if SSL Level Keeping is not present on the device.
- In order to use CPHA=0 setting in slave mode, the master must toggle the SSL pin after every data frame (Even if the device supports SSL Level Keeping). Because of this hardware limitation, the module will return FSP_ERR_UNSUPPORTED when it is configured to use CPHA=0 setting in slave mode.
- The module does not support communicating with multiple slaves using different SSL pins. In order to achieve this, the module must either be closed and re-opened to change the SSL pin or drive SSL in software. It is recommended that SSL be driven in software when controlling multiple slave devices.
- The SPI peripheral has a minimum 3 SPI CLK delay between each data frame.

## Examples

### Basic Example

This is a basic example of minimal use of the SPI in an application.

```
static volatile bool g_transfer_complete = false;
void spi_basic_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the SPI module. */
    err = R_SPI_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Start a write/read transfer */
```

```
    err = R_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    handle_error(err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
}
static void r_spi_callback (spi_callback_args_t * p_args)
{
 if (SPI_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_transfer_complete = true;
    }
}
```

### Driving Software Slave Select Line

This is an example of communicating with multiple slave devices by asserting SSL in software.

```
void spi_software_ssl_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
 /* Configure Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Configure Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the SPI module. */
    err = R_SPI_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Assert Slave Select Line 1 */
```

```
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err                 = R_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer,
TRANSFER_SIZE, SPI_BIT_WIDTH_8_BITS);
    handle_error(err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Wait for minimum time required between transfers. */
 R_BSP_SoftwareDelay(SSL_NEXT_ACCESS_DELAY, BSP_DELAY_UNITS_MICROSECONDS);
 /* Assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err                 = R_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer,
TRANSFER_SIZE, SPI_BIT_WIDTH_8_BITS);
    handle_error(err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
}
```

### Configuring the SPI Clock Divider Registers

This example demonstrates how to set the SPI clock divisors at runtime.

```
void spi_bitrate_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    g_spi_cfg.p_extend = &g_spi_extended_cfg;
 /* Configure SPI Clock divider to achieve largest bitrate less than or equal to the
desired bitrate. */
    err = R_SPI_CalculateBitrate(BITRATE, &(g_spi_extended_cfg.spck_div));
    handle_error(err);
 /* Initialize the SPI module. */
    err = R_SPI_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
}
```

## Data Structures

| | |
|---|---|
| struct | rspck_div_setting_t |
| struct | spi_extended_cfg_t |
| struct | spi_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | spi_ssl_mode_t |
| enum | spi_communication_t |
| enum | spi_ssl_polarity_t |
| enum | spi_ssl_select_t |
| enum | spi_mosi_idle_value_fixing_t |
| enum | spi_parity_t |
| enum | spi_byte_swap_t |
| enum | spi_delay_count_t |

## Data Structure Documentation

### ◆ rspck_div_setting_t

| struct rspck_div_setting_t | | |
|---|---|---|
| SPI Clock Divider settings. | | |
| Data Fields | | |
| uint8_t | spbr | SPBR register setting. |
| uint8_t | brdv: 2 | BRDV setting in SPCMD0. |

### ◆ spi_extended_cfg_t

| struct spi_extended_cfg_t | | |
|---|---|---|
| Extended SPI interface configuration | | |
| Data Fields | | |
| spi_ssl_mode_t | spi_clksyn | Select spi or clock syn mode operation. |
| spi_communication_t | spi_comm | Select full-duplex or transmit-only communication. |
| spi_ssl_polarity_t | ssl_polarity | Select SSLn signal polarity. |
| spi_ssl_select_t | ssl_select | Select which slave to use: 0-SSL0, 1-SSL1, 2-SSL2, 3-SSL3. |
| spi_mosi_idle_value_fixing_t | mosi_idle | Select MOSI idle fixed value and selection. |
| spi_parity_t | parity | Select parity and enable/disable parity. |
| spi_byte_swap_t | byte_swap | Select byte swap mode. |
| rspck_div_setting_t | spck_div | Register values for configuring the SPI Clock Divider. |
| spi_delay_count_t | spck_delay | SPI Clock Delay Register Setting. |
| spi_delay_count_t | ssl_negation_delay | SPI Slave Select Negation Delay Register Setting. |
| spi_delay_count_t | next_access_delay | SPI Next-Access Delay Register Setting. |

### ◆ spi_instance_ctrl_t

| struct spi_instance_ctrl_t | | |
|---|---|---|
| Channel control block. DO NOT INITIALIZE. Initialization occurs when spi_api_t::open is called. | | |
| Data Fields | | |
| uint32_t | open | Indicates whether the open() API has been successfully called. |
| spi_cfg_t const * | p_cfg | Pointer to instance configuration. |

| | | |
|---|---|---|
| R_SPI0_Type * | p_regs | Base register for this channel. |
| void const * | p_tx_data | Buffer to transmit. |
| void * | p_rx_data | Buffer to receive. |
| uint32_t | tx_count | Number of Data Frames to transfer (8-bit, 16-bit, 32-bit) |
| uint32_t | rx_count | Number of Data Frames to transfer (8-bit, 16-bit, 32-bit) |
| uint32_t | count | Number of Data Frames to transfer (8-bit, 16-bit, 32-bit) |
| spi_bit_width_t | bit_width | Bits per Data frame (8-bit, 16-bit, 32-bit) |

## Enumeration Type Documentation

### ◆ spi_ssl_mode_t

| enum spi_ssl_mode_t | |
|---|---|
| 3-Wire or 4-Wire mode. | |
| Enumerator | |
| SPI_SSL_MODE_SPI | SPI operation (4-wire method) |
| SPI_SSL_MODE_CLK_SYN | Clock Synchronous operation (3-wire method) |

### ◆ spi_communication_t

| enum spi_communication_t | |
|---|---|
| Transmit Only (Half Duplex), or Full Duplex. | |
| Enumerator | |
| SPI_COMMUNICATION_FULL_DUPLEX | Full-Duplex synchronous serial communication. |
| SPI_COMMUNICATION_TRANSMIT_ONLY | Transit only serial communication. |

### ◆ spi_ssl_polarity_t

| enum spi_ssl_polarity_t | |
|---|---|
| Slave Select Polarity. | |
| Enumerator | |
| SPI_SSLP_LOW | SSLP signal polarity active low. |
| SPI_SSLP_HIGH | SSLP signal polarity active high. |

### ◆ spi_ssl_select_t

| enum spi_ssl_select_t | |
|---|---|
| The Slave Select Line | |
| Enumerator | |
| SPI_SSL_SELECT_SSL0 | Select SSL0. |
| SPI_SSL_SELECT_SSL1 | Select SSL1. |
| SPI_SSL_SELECT_SSL2 | Select SSL2. |
| SPI_SSL_SELECT_SSL3 | Select SSL3. |

### ◆ spi_mosi_idle_value_fixing_t

| enum spi_mosi_idle_value_fixing_t | |
|---|---|
| MOSI Idle Behavior. | |
| Enumerator | |
| SPI_MOSI_IDLE_VALUE_FIXING_DISABLE | MOSI output value=value set in MOIFV bit. |
| SPI_MOSI_IDLE_VALUE_FIXING_LOW | MOSIn level low during MOSI idling. |
| SPI_MOSI_IDLE_VALUE_FIXING_HIGH | MOSIn level high during MOSI idling. |

### ◆ spi_parity_t

| enum spi_parity_t | |
|---|---|
| Parity Mode | |
| Enumerator | |
| SPI_PARITY_MODE_DISABLE | Disable parity. |
| SPI_PARITY_MODE_ODD | Select even parity. |
| SPI_PARITY_MODE_EVEN | Select odd parity. |

### ◆ spi_byte_swap_t

| enum spi_byte_swap_t | |
|---|---|
| Byte Swapping Enable/Disable. | |
| Enumerator | |
| SPI_BYTE_SWAP_DISABLE | Disable Byte swapping for 16/32-Bit transfers. |
| SPI_BYTE_SWAP_ENABLE | Enable Byte swapping for 16/32-Bit transfers. |

### ◆ spi_delay_count_t

| enum spi_delay_count_t | |
|---|---|
| Delay count for SPI delay settings. | |
| Enumerator | |
| SPI_DELAY_COUNT_1 | Set RSPCK delay count to 1 RSPCK. |
| SPI_DELAY_COUNT_2 | Set RSPCK delay count to 2 RSPCK. |
| SPI_DELAY_COUNT_3 | Set RSPCK delay count to 3 RSPCK. |
| SPI_DELAY_COUNT_4 | Set RSPCK delay count to 4 RSPCK. |
| SPI_DELAY_COUNT_5 | Set RSPCK delay count to 5 RSPCK. |
| SPI_DELAY_COUNT_6 | Set RSPCK delay count to 6 RSPCK. |
| SPI_DELAY_COUNT_7 | Set RSPCK delay count to 7 RSPCK. |
| SPI_DELAY_COUNT_8 | Set RSPCK delay count to 8 RSPCK. |

## Function Documentation

### ◆ R_SPI_Open()

fsp_err_t R_SPI_Open ( spi_ctrl_t * *p_api_ctrl*, spi_cfg_t const *const *p_cfg* )

This functions initializes a channel for SPI communication mode. Implements spi_api_t::open.

This function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Configures the pperipheral registers acording to the configuration.
- Initialize the control structure for use in other SPI Interface functions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel initialized successfully. |
| FSP_ERR_ALREADY_OPEN | Instance was already initialized. |
| FSP_ERR_ASSERTION | An invalid argument was given in the configuration structure. |
| FSP_ERR_UNSUPPORTED | A requested setting is not possible on this device with the current build configuration. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel number is invalid. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls: transfer_api_t::open

***Note***

*This function is reentrant.*

### ◆ R_SPI_Read()

fsp_err_t R_SPI_Read ( spi_ctrl_t *const *p_api_ctrl*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

This function receives data from a SPI device. Implements spi_api_t::read.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI read operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Read operation successfully completed. |
| FSP_ERR_ASSERTION | NULL pointer to control or destination parameters or transfer length is zero. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open channel first. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

### ◆ R_SPI_Write()

fsp_err_t R_SPI_Write ( spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

This function transmits data to a SPI device using the TX Only Communications Operation Mode. Implements spi_api_t::write.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI write operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write operation successfully completed. |
| FSP_ERR_ASSERTION | NULL pointer to control or source parameters or transfer length is zero. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

### ◆ R_SPI_WriteRead()

fsp_err_t R_SPI_WriteRead ( spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

This function simultaneously transmits and receive data. Implements spi_api_t::writeRead.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI writeRead operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write operation successfully completed. |
| FSP_ERR_ASSERTION | NULL pointer to control, source or destination parameters or transfer length is zero. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

### ◆ R_SPI_Close()

fsp_err_t R_SPI_Close ( spi_ctrl_t *const *p_api_ctrl*)

This function manages the closing of a channel by the following task. Implements spi_api_t::close.

Disables SPI operations by disabling the SPI bus.

- Disables the SPI peripheral.
- Disables all the associated interrupts.
- Update control structure so it will not work with SPI Interface functions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | A required pointer argument is NULL. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |

### ◆ R_SPI_VersionGet()

fsp_err_t R_SPI_VersionGet ( fsp_version_t * *p_version*)

This function gets the version information of the underlying driver. Implements spi_api_t::versionGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful version get. |
| FSP_ERR_ASSERTION | The parameter p_version is NULL. |

### ◆ R_SPI_CalculateBitrate()

fsp_err_t R_SPI_CalculateBitrate ( uint32_t *bitrate*, rspck_div_setting_t * *spck_div* )

Calculates the SPBR register value and the BRDV bits for a desired bitrate. If the desired bitrate is faster than the maximum bitrate, than the bitrate is set to the maximum bitrate. If the desired bitrate is slower than the minimum bitrate, than an error is returned.

**Parameters**

| | | |
|---|---|---|
| [in] | bitrate | Desired bitrate |
| [out] | spck_div | Memory location to store bitrate register settings. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Valid spbr and brdv values were calculated |
| FSP_ERR_UNSUPPORTED | Bitrate is not achievable |

## 4.2.45 Serial Sound Interface (r_ssi)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_SSI_Open (i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg) |
| fsp_err_t | R_SSI_Stop (i2s_ctrl_t *const p_ctrl) |
| fsp_err_t | R_SSI_StatusGet (i2s_ctrl_t *const p_ctrl, i2s_status_t *const p_status) |
| fsp_err_t | R_SSI_Write (i2s_ctrl_t *const p_ctrl, void const *const p_src, uint32_t const bytes) |
| fsp_err_t | R_SSI_Read (i2s_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_SSI_WriteRead (i2s_ctrl_t *const p_ctrl, void const *const p_src, void *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_SSI_Mute (i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable) |
| fsp_err_t | R_SSI_Close (i2s_ctrl_t *const p_ctrl) |

fsp_err_t R_SSI_VersionGet (fsp_version_t *const p_version)

### Detailed Description

Driver for the SSIE peripheral on RA MCUs. This module implements the I2S Interface.

# Overview

### Features

The SSI module supports the following features:

- Transmission and reception of uncompressed audio data using the standard I2S protocol in master mode
- Full-duplex I2S communication (channel 0 only)
- Integration with the DTC transfer module
- Internal connection to GPT GTIOC1A timer output to generate the audio clock
- Callback function notification when all data is loaded into the SSI FIFO

# Configuration

### Build Time Configurations for r_ssi

The following build time configurations are defined in fsp_cfg/r_ssi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| DTC Support | • Enabled<br>• Disabled | Enabled | If code for DTC transfer support is included in the build. |

### Configurations for Driver > Connectivity > I2S Driver on r_ssi

This module can be added to the Stacks tab via New Stack > Driver > Connectivity > I2S Driver on r_ssi:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_i2s0 | Module name. |
| Channel | Value must be an integer between 0 and 1 | 0 | Specify the I2S channel. |
| Bit Depth | • 8 Bits | 16 Bits | Select the bit depth of |

| | | | |
|---|---|---|---|
| | • 16 Bits<br>• 18 Bits<br>• 20 Bits<br>• 22 Bits<br>• 24 Bits<br>• 32 Bits | | one sample of audio data. |
| Word Length | • 8 Bits<br>• 16 Bits<br>• 24 Bits<br>• 32 Bits<br>• 48 Bits<br>• 64 Bits<br>• 128 Bits<br>• 256 Bits | 16 Bits | Select the word length of audio data. Must be at least as large as Data bits. |
| WS Continue Mode | • Enabled<br>• Disabled | Disabled | Enable WS continue mode to output the word select (WS) pin even when transmission is idle. |
| Bit Clock Source | • AUDIO_CLK<br>• GTIOC1A | AUDIO_CLK | Select AUDIO_CLK for external signal to AUDIO_CLK input pin or GTIOC1A for internal connection to GPT channel 1 GTIOC1A. |
| Bit Clock Divider | Refer to the RA Configuration tool for available options. | Audio Clock / 1 | Select divider used to generate bit clock from audio clock. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called from all three interrupt service routines (ISR). |
| Transmit Interrupt Priority | MCU Specific Options | | Select the transmit interrupt priority. |
| Receive Interrupt Priority | MCU Specific Options | | Select the receive interrupt priority. |
| Idle/Error Interrupt Priority | MCU Specific Options | | Select the Idle/Error interrupt priority. |

### Clock Configuration

The SSI peripheral runs on PCLKB. The PCLKB frequency can be configured on the **Clocks** tab of the RA Configuration editor. The SSI audio clock can optionally be supplied from an external source through the AUDIO_CLK pin in master mode.

### Pin Configuration

The SSI uses the following pins:

- AUDIO_CLK (optional, master mode only): The AUDIO_CLK pin is used to supply the audio clock from an external source.
- SSIBCKn: Bit clock pin for channel n
- SSILRCKn/SSIFSn: Channel selection pin for channel n
- SSIRXD0: Reception pin for channel 0
- SSITXD0: Transmission pin for channel 0
- SSIDATA1: Transmission or reception pin for channel 1

# Usage Notes

### SSI Frames

An SSI frame is 2 samples worth of data. The frame boundary (end of previous frame, start of next frame) is on the falling edge of the SSILRCKn signal.

### Audio Data

Only uncompressed PCM data is supported.

Data arrays have the following size, alignment, and length based on the "Bit Depth" setting:

| Bit Depth | Array Data Type | Required Alignment | Required Length (bytes) |
|---|---|---|---|
| 8 Bits | 8-bit integer | 1 byte alignment | Multiple of 2 |
| 16 Bits | 16-bit integer | 2 byte alignment | Multiple of 4 |
| 18 Bits | 32-bit integer, right justified | 4 byte alignment | Multiple of 8 |
| 20 Bits | 32-bit integer, right justified | 4 byte alignment | Multiple of 8 |
| 22 Bits | 32-bit integer, right justified | 4 byte alignment | Multiple of 8 |
| 24 Bits | 32-bit integer, right justified | 4 byte alignment | Multiple of 8 |
| 32 Bits | 32-bit integer | 4 byte alignment | Multiple of 8 |

*Note*

*The length of the array must be a multiple of 2 when the data type is the recommended data type. The 2 represents the frame size (left and right channel) of I2S communication. The SSIE peripheral does not support odd read/write lengths in I2S mode.*

### Audio Clock

The audio clock is only required for master mode.

### Audio Clock Frequency

The bit clock frequency is the product of the sampling frequency and channels and bits per system

word:

bit_clock (Hz) = sampling_frequency (Hz) * channels * system_word_bits

I2S data always has 2 channels.

For example, the bit clock for transmitting 2 channels of 16-bit data (using a 16-bit system word) at 44100 Hz would be:

44100 * 2 * 16 = 1,411,200 Hz

The audio clock frequency is used to generate the bit clock frequency. It must be a multiple of the bit clock frequency. Refer to the Bit Clock Divider configuration for divider options. The input audio clock frequency must be:

audio_clock (Hz) = desired_bit_clock (Hz) * bit_clock_divider

To get a bit clock of 1.4 MHz from an audio clock of 2.8 MHz, select the divider Audio Clock / 2.

### Audio Clock Source

The audio clock source can come from:

- An external source input to the AUDIO_CLK pin
- An internal connection to the GPT channel 1 A output (GTIOC1A)

*Note*

*When using the GTIOC1A output, the GPT channel must be set to channel 1, Pin Output Support must be Enabled, and GTIOCA Output Enabled must be True.*

### Limitations

Developers should be aware of the following limitations when using the SSI:

- When using channel 1, full duplex communication is not possible. Only tranmission or reception is possible.
- SSI must go idle before changing the communication mode (between read only, write only, and full duplex)

## Examples

### Basic Example

This is a basic example of minimal use of the SSI in an application.

```
#define SSI_EXAMPLE_SAMPLES_TO_TRANSFER (1024)

#define SSI_EXAMPLE_TONE_FREQUENCY_HZ (800)

int16_t g_src[SSI_EXAMPLE_SAMPLES_TO_TRANSFER];

int16_t g_dest[SSI_EXAMPLE_SAMPLES_TO_TRANSFER];

void ssi_basic_example (void)

{
```

```
 fsp_err_t err = FSP_SUCCESS;
 /* Create a stereo sine wave. Using formula sample = sin(2 * pi * tone_frequency * t
/ sampling_frequency) */
    uint32_t freq = SSI_EXAMPLE_TONE_FREQUENCY_HZ;
 for (uint32_t t = 0; t < SSI_EXAMPLE_SAMPLES_TO_TRANSFER / 2; t += 1)
    {
 float input = (((float) (freq * t)) * (M_TWOPI)) /
SSI_EXAMPLE_AUDIO_SAMPLING_FREQUENCY_HZ;
       g_src[2 * t]    = (int16_t) ((INT16_MAX * sinf(input)));
       g_src[2 * t + 1] = (int16_t) ((INT16_MAX * sinf(input)));
    }
 /* Initialize the module. */
    err = R_SSI_Open(&g_i2s_ctrl, &g_i2s_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Transfer data. */
    (void) R_SSI_WriteRead(&g_i2s_ctrl,
                           (uint8_t *) &g_src[0],
                           (uint8_t *) &g_dest[0],
                           SSI_EXAMPLE_SAMPLES_TO_TRANSFER * sizeof(int16_t));
}
```

### Streaming Example

This is an example of using SSI to stream audio data. This application uses a double buffer to store PCM sine wave data. It starts transmitting in the main loop, then loads the next buffer if it is ready in the callback. If the next buffer is not ready, a flag is set in the callback so the application knows to restart transmission in the main loop.

This example also checks the return code of R_SSI_Write() because R_SSI_Write() can return an error if a transmit overflow occurs before the FIFO is reloaded. If a transmit overflow occurs before the FIFO is reloaded, the SSI will be stopped in the error interrupt, and it cannot be restarted until the I2S_EVENT_IDLE callback is received.

```
#define SSI_STREAMING_EXAMPLE_AUDIO_SAMPLING_FREQUENCY_HZ (22050)
#define SSI_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK (1024)
#define SSI_STREAMING_EXAMPLE_TONE_FREQUENCY_HZ (800)
int16_t       g_stream_src[2][SSI_EXAMPLE_SAMPLES_TO_TRANSFER];
```

```
uint32_t      g_buffer_index          = 0;
volatile bool g_send_data_in_main_loop = true;
volatile bool g_data_ready = false;
/* Example callback called when SSI is ready for more data. */
void ssi_example_callback (i2s_callback_args_t * p_args)
{
 /* Reload the FIFO if we hit the transmit watermark or restart transmission if the
SSI is idle because it was
  * stopped after a transmit FIFO overflow. */
 if ((I2S_EVENT_TX_EMPTY == p_args->event) || (I2S_EVENT_IDLE == p_args->event))
    {
 if (g_data_ready)
      {
 /* Reload FIFO and handle errors. */
            ssi_example_write();
      }
 else
      {
 /* Data was not ready yet, send it in the main loop. */
            g_send_data_in_main_loop = true;
      }
    }
}
/* Load the transmit FIFO and check for error conditions. */
void ssi_example_write (void)
{
 /* Transfer data. This call is non-blocking. */
 fsp_err_t err = R_SSI_Write(&g_i2s_ctrl,
                              (uint8_t *) &g_stream_src[g_buffer_index][0],
                              SSI_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK * sizeof
(int16_t));
 if (FSP_SUCCESS == err)
    {
 /* Switch the buffer after data is sent. */
```

```
        g_buffer_index = !g_buffer_index;
 /* Allow loop to calculate next buffer only if transmission was successful. */
        g_data_ready = false;
    }
 else
    {
 /* Getting here most likely means a transmit overflow occurred before the FIFO could
be reloaded. The
  * application must wait until the SSI is idle, then restart transmission. In this
example, the idle
  * callback transmits data or resets the flag g_send_data_in_main_loop. */
    }
}
/* Calculate samples. This example is just a sine wave. For this type of data, it
would be better to calculate
 * one period and loop it. This example should be updated for the audio data used by
the application. */
void ssi_example_calculate_samples (uint32_t buffer_index)
{
 static uint32_t t = 0U;
 /* Create a stereo sine wave. Using formula sample = sin(2 * pi * tone_frequency * t
/ sampling_frequency) */
    uint32_t freq = SSI_STREAMING_EXAMPLE_TONE_FREQUENCY_HZ;
 for (uint32_t i = 0; i < SSI_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK / 2; i += 1)
    {
 float input = (((float) (freq * t)) * M_TWOPI) /
SSI_STREAMING_EXAMPLE_AUDIO_SAMPLING_FREQUENCY_HZ;
        t++;
 /* Store sample twice, once for left channel and once for right channel. */
        int16_t sample = (int16_t) ((INT16_MAX * sinf(input)));
        g_stream_src[buffer_index][2 * i]    = sample;
        g_stream_src[buffer_index][2 * i + 1] = sample;
    }
 /* Data is ready to be sent in the interrupt. */
```

```
    g_data_ready = true;
}
void ssi_streaming_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the module. */
    err = R_SSI_Open(&g_i2s_ctrl, &g_i2s_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (true)
    {
 /* Prepare data in a buffer that is not currently used for transmission. */
       ssi_example_calculate_samples(g_buffer_index);
 /* Send data in main loop the first time, and if it was not ready in the interrupt.
*/
 if (g_send_data_in_main_loop)
      {
 /* Clear flag. */
            g_send_data_in_main_loop = false;
 /* Reload FIFO and handle errors. */
            ssi_example_write();
      }
 /* If the next buffer is ready, wait for the data to be sent in the interrupt. */
 while (g_data_ready)
      {
 /* Do nothing. */
      }
    }
}
```

## Data Structures

| | |
|---|---|
| struct | ssi_instance_ctrl_t |
| struct | ssi_extended_cfg_t |

## Enumerations

| | |
|---|---|
| enum | ssi_audio_clock_t |
| enum | ssi_clock_div_t |

## Data Structure Documentation

### ssi_instance_ctrl_t

| struct ssi_instance_ctrl_t |
|---|
| Channel instance control block. DO NOT INITIALIZE. Initialization occurs when i2s_api_t::open is called. |

### ssi_extended_cfg_t

| struct ssi_extended_cfg_t | | |
|---|---|---|
| SSI configuration extension. This extension is optional. | | |
| Data Fields | | |
| ssi_audio_clock_t | audio_clock | Audio clock source, default is SSI_AUDIO_CLOCK_EXTERNAL. |
| ssi_clock_div_t | bit_clock_div | Select bit clock division ratio. |

## Enumeration Type Documentation

### ssi_audio_clock_t

| enum ssi_audio_clock_t | |
|---|---|
| Audio clock source. | |
| Enumerator | |
| 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. |

### ◆ ssi_clock_div_t

| enum ssi_clock_div_t | |
|---|---|
| Bit clock division ratio. Bit clock frequency = audio clock frequency / bit clock division ratio. | |
| Enumerator | |
| SSI_CLOCK_DIV_1 | Clock divisor 1. |
| SSI_CLOCK_DIV_2 | Clock divisor 2. |
| SSI_CLOCK_DIV_4 | Clock divisor 4. |
| SSI_CLOCK_DIV_6 | Clock divisor 6. |
| SSI_CLOCK_DIV_8 | Clock divisor 8. |
| SSI_CLOCK_DIV_12 | Clock divisor 12. |
| SSI_CLOCK_DIV_16 | Clock divisor 16. |
| SSI_CLOCK_DIV_24 | Clock divisor 24. |
| SSI_CLOCK_DIV_32 | Clock divisor 32. |
| SSI_CLOCK_DIV_48 | Clock divisor 48. |
| SSI_CLOCK_DIV_64 | Clock divisor 64. |
| SSI_CLOCK_DIV_96 | Clock divisor 96. |
| SSI_CLOCK_DIV_128 | Clock divisor 128. |

## Function Documentation

### ◆ R_SSI_Open()

fsp_err_t R_SSI_Open ( i2s_ctrl_t *const *p_ctrl*, i2s_cfg_t const *const *p_cfg* )

Opens the SSI. Implements i2s_api_t::open.

This function sets this clock divisor and the configurations specified in i2s_cfg_t. It also opens the timer and transfer instances if they are provided.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Ready for I2S communication. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl or p_cfg is null. |
| FSP_ERR_ALREADY_OPEN | The control block has already been opened. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Channel number is not available on this MCU. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::open

### ◆ R_SSI_Stop()

fsp_err_t R_SSI_Stop ( i2s_ctrl_t *const *p_ctrl*)

Stops SSI. Implements i2s_api_t::stop.

This function disables both transmission and reception, and disables any transfer instances used.

The SSI will stop on the next frame boundary. Do not restart SSI until it is idle.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | I2S communication stop request issued. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |

**Returns**

See Common Error Codes or lower level drivers for other possible return codes.

### ◆ R_SSI_StatusGet()

fsp_err_t R_SSI_StatusGet ( i2s_ctrl_t *const *p_ctrl*, i2s_status_t *const *p_status* )

Gets SSI status and stores it in provided pointer p_status. Implements i2s_api_t::statusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored successfully. |
| FSP_ERR_ASSERTION | The p_instance_ctrl or p_status parameter was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |

### ◆ R_SSI_Write()

fsp_err_t R_SSI_Write ( i2s_ctrl_t *const *p_ctrl*, void const *const *p_src*, uint32_t const *bytes* )

Writes data buffer to SSI. Implements i2s_api_t::write.

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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write initiated successfully. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl or p_src was null, or bytes requested was 0. |
| FSP_ERR_IN_USE | Another transfer is in progress, data was not written. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |
| FSP_ERR_UNDERFLOW | A transmit underflow error is pending. Wait for the SSI to go idle before resuming communication. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

### ◆ R_SSI_Read()

fsp_err_t R_SSI_Read ( i2s_ctrl_t *const *p_ctrl*, void *const *p_dest*, uint32_t const *bytes* )

Reads data into provided buffer. Implements i2s_api_t::read.

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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Read initiated successfully. |
| FSP_ERR_IN_USE | Peripheral is in the wrong mode or not idle. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl or p_dest was null, or bytes requested was 0. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |
| FSP_ERR_OVERFLOW | A receive overflow error is pending. Wait for the SSI to go idle before resuming communication. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

### ◆ R_SSI_WriteRead()

fsp_err_t R_SSI_WriteRead ( i2s_ctrl_t *const *p_ctrl*, void const *const *p_src*, void *const *p_dest*, uint32_t const *bytes* )

Writes from source buffer and reads data into destination buffer. Implements i2s_api_t::writeRead.

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.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write and read initiated successfully. |
| FSP_ERR_IN_USE | Peripheral is in the wrong mode or not idle. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |
| FSP_ERR_UNDERFLOW | A transmit underflow error is pending. Wait for the SSI to go idle before resuming communication. |
| FSP_ERR_OVERFLOW | A receive overflow error is pending. Wait for the SSI to go idle before resuming communication. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

### ◆ R_SSI_Mute()

fsp_err_t R_SSI_Mute ( i2s_ctrl_t *const *p_ctrl*, i2s_mute_t const *mute_enable* )

Mutes SSI on the next frame boundary. Implements i2s_api_t::mute.

Data is still written while mute is enabled, but the transmit line outputs zeros.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transmission is muted. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |

### ◆ R_SSI_Close()

fsp_err_t R_SSI_Close ( i2s_ctrl_t *const *p_ctrl*)

Closes SSI. Implements i2s_api_t::close.

This function powers down the SSI and closes the lower level timer and transfer drivers if they are used.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device closed successfully. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |

### ◆ R_SSI_VersionGet()

fsp_err_t R_SSI_VersionGet ( fsp_version_t *const *p_version*)

Sets driver version based on compile time macros.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | The parameter p_version is NULL. |

## 4.2.46 USB (r_usb_basic)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | R_USB_Open (usb_ctrl_t *const p_api_ctrl, usb_cfg_t const *const p_cfg)<br>Applies power to the USB module specified in the argument (p_ctrl). More... |
| fsp_err_t | R_USB_Close (usb_ctrl_t *const p_api_ctrl)<br>Terminates power to the USB module specified in argument (p_ctrl). USB0 module stops when USB_IP0 is specified to the member (module), USB1 module stops when USB_IP1 is specified to the member (module). More... |

| | |
|---|---|
| fsp_err_t | R_USB_Read (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t destination)<br>Bulk/interrupt data transfer and control data transfer. More... |
| fsp_err_t | R_USB_Write (usb_ctrl_t *const p_api_ctrl, uint8_t const *const p_buf, uint32_t size, uint8_t destination)<br>Bulk/Interrupt data transfer and control data transfer. More... |
| fsp_err_t | R_USB_Stop (usb_ctrl_t *const p_api_ctrl, usb_transfer_t direction, uint8_t destination)<br>Requests a data read/write transfer be terminated when a data read/write transfer is being performed. More... |
| fsp_err_t | R_USB_Suspend (usb_ctrl_t *const p_api_ctrl)<br>Sends a SUSPEND signal from the USB module assigned to the member (module) of the usb_crtl_t structure. More... |
| fsp_err_t | R_USB_Resume (usb_ctrl_t *const p_api_ctrl)<br>Sends a RESUME signal from the USB module assigned to the member (module) of the usb_ctrl_tstructure. More... |
| fsp_err_t | R_USB_VbusSet (usb_ctrl_t *const p_api_ctrl, uint16_t state)<br>Specifies starting or stopping the VBUS supply. More... |
| fsp_err_t | R_USB_InfoGet (usb_ctrl_t *const p_api_ctrl, usb_info_t *p_info, uint8_t destination)<br>Obtains completed USB-related events. More... |
| fsp_err_t | R_USB_PipeRead (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number)<br>Requests a data read (bulk/interrupt transfer) via the pipe specified in the argument. More... |
| fsp_err_t | R_USB_PipeWrite (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number)<br>Requests a data write (bulk/interrupt transfer). More... |

| | |
|---|---|
| fsp_err_t | R_USB_PipeStop (usb_ctrl_t *const p_api_ctrl, uint8_t pipe_number)<br>Terminates a data read/write operation. More... |
| fsp_err_t | R_USB_UsedPipesGet (usb_ctrl_t *const p_api_ctrl, uint16_t *p_pipe, uint8_t destination)<br>Gets the selected pipe number (number of the pipe that has completed initalization) via bit map information. More... |
| fsp_err_t | R_USB_PipeInfoGet (usb_ctrl_t *const p_api_ctrl, usb_pipe_t *p_info, uint8_t pipe_number)<br>Gets the following pipe information regarding the pipe specified in the argument (p_ctrl) member (pipe): endpoint number, transfer type, transfer direction and maximum packet size. More... |
| fsp_err_t | R_USB_PullUp (usb_ctrl_t *const p_api_ctrl, uint8_t state)<br>This API enables or disables pull-up of D+/D- line. More... |
| fsp_err_t | R_USB_EventGet (usb_ctrl_t *const p_api_ctrl, usb_status_t *event)<br>Obtains completed USB related events. (OS-less Only) More... |
| fsp_err_t | R_USB_VersionGet (fsp_version_t *const p_version)<br>Returns the version of this module. More... |
| fsp_err_t | R_USB_Callback (usb_callback_t *p_callback)<br>Register a callback function to be called upon completion of a USB related event. (RTOS only) More... |
| fsp_err_t | R_USB_HostControlTransfer (usb_ctrl_t *const p_api_ctrl, usb_setup_t *p_setup, uint8_t *p_buf, uint8_t device_address)<br>Performs settings and transmission processing when transmitting a setup packet. More... |
| fsp_err_t | R_USB_PeriControlDataGet (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size)<br>Receives data sent by control transfer. More... |
| fsp_err_t | R_USB_PeriControlDataSet (usb_ctrl_t *const p_api_ctrl, uint8_t |

| | |
|---|---|
| | *p_buf, uint32_t size)<br>Performs transfer processing for control transfer. More... |
| fsp_err_t | R_USB_PeriControlStatusSet (usb_ctrl_t *const p_api_ctrl, usb_setup_status_t status)<br>Set the response to the setup packet. More... |
| fsp_err_t | R_USB_RemoteWakeup (usb_ctrl_t *const p_api_ctrl)<br>Sends a remote wake-up signal to the connected Host. More... |
| fsp_err_t | R_USB_ModuleNumberGet (usb_ctrl_t *const p_api_ctrl, uint8_t *module_number)<br>This API gets the module number. More... |
| fsp_err_t | R_USB_ClassTypeGet (usb_ctrl_t *const p_api_ctrl, usb_class_t *class_type)<br>This API gets the class type. More... |
| fsp_err_t | R_USB_DeviceAddressGet (usb_ctrl_t *const p_api_ctrl, uint8_t *device_address)<br>This API gets the device address. More... |
| fsp_err_t | R_USB_PipeNumberGet (usb_ctrl_t *const p_api_ctrl, uint8_t *pipe_number)<br>This API gets the pipe number. More... |
| fsp_err_t | R_USB_DeviceStateGet (usb_ctrl_t *const p_api_ctrl, uint16_t *state)<br>This API gets the state of the device. More... |
| fsp_err_t | R_USB_DataSizeGet (usb_ctrl_t *const p_api_ctrl, uint32_t *data_size)<br>This API gets the data size. More... |
| fsp_err_t | R_USB_SetupGet (usb_ctrl_t *const p_api_ctrl, usb_setup_t *setup)<br>This API gets the setup type. More... |

### Detailed Description

The USB module (r_usb_basic) provides an API to perform hardware control of USB communication. It implements the USB Interface.

# Overview

The USB module performs USB hardware control. The USB module operates in combination with the device class drivers provided by Renesas.

### Features

The USB module has the following key features:

- Overall
  - Supporting USB Host or USB Peripheral.
  - Device connect/disconnect, suspend/resume, and USB bus reset processing.
  - Control transfer on pipe 0.
  - Data transfer on pipes 1 to 9. (Bulk or Interrupt transfer)
  - This driver supports RTOS version (hereinafter called "RTOS") and Non-OS version (hereinafter called "Non-OS"). RTOS uses the realtime OS (FreeRTOS). Non-OS does not use the real time OS.
- Host mode
  - In host mode, enumeration as Low-speed/Full-speed/High-speed device (However, operating speed is different by devices ability.)
  - Transfer error determination and transfer retry.
- Peripheral mode
  - In peripheral mode, enumeration as USB Host of USB1.1/2.0/3.0.

# Configuration

### Build Time Configurations for r_usb_basic

The following build time configurations are defined in fsp_cfg/r_usb_basic_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| PLL clock frequency setting | • 24MHz<br>• 20MHz<br>• 12MHz<br>• Other than 24/20/12MHz | 24MHz | In the case of a USB module other than USB1 module, this definition is ignored. |
| CPU bus access wait setting | Refer to the RA Configuration tool for available options. | 9 access cycles | CPU Bus Access Wait Select(CPU Bus Wait Register (BUSWAIT)BWAIT[3:0]) 2-17 access cycle wait |
| Setting the battery | • Enable | Enable | Not using the battery |

| | | | |
|---|---|---|---|
| charging function | • Disable | | charging function Using the battery charging function |
| Setting the power source IC | • High assert<br>• Low assert | High assert | Select High assert or Low assert. |
| Setting USB port operation when using the battery charging function | • DCP enabled<br>• DCP disabled | DCP disabled | Please select whether to deactivate or activate the DCP. |
| Setting whether to notify the application when receiving the req uest(SET_INTERFACE/S ET_FEATURE/CLEAR_FE ATURE) | • Not notifying.<br>• Notifying | Notifying | Please choose whether it corresponds to the class request. |
| Select whether to use the double buffer function. | • Disable<br>• Enable | Enable | Please choose whether it corresponds to the double buffer. |
| Select whether to use the continuous transfer mode. | • Disable<br>• Enable | Disable | Please choose whether it corresponds to the continuous transfer mode. |
| DMA Support. | • Disable<br>• Enable | Disable | Enable DMA support for the USB module. |
| Transfer source address when DMA is supported. | • DMA not support.<br>• Setting for FS module.<br>• Setting for HS module. | DMA not support. | It changes with the IP number used. |
| Transfer destination address when DMA is supported. | • DMA not support.<br>• Setting for FS module.<br>• Setting for HS module. | DMA not support. | It changes with the IP number used. |

**Configurations for Middleware > USB > USB Driver on r_usb_basic**

This module can be added to the Stacks tab via New Stack > Middleware > USB > USB Driver on r_usb_basic:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_basic0 | Module name. |
| USB Mode | • Host mode<br>• Peri mode | Host mode | Select the usb mode. |

| | | | |
|---|---|---|---|
| USB Speed | • Full Speed<br>• Hi Speed<br>• Low Speed | Full Speed | Select the usb speed. |
| USB Module Number | • USB_IP0 Port<br>• USB_IP1 Port | USB_IP0 Port | Specify the USB module number to be used. |
| USB Class Type | • Peripheral communication device class<br>• Peripheral human interface device class<br>• Peripheral mass strage class<br>• Peripheral vender class<br>• Host communication device class<br>• Host human interface device class<br>• Host mass strage class<br>• Host vender class | Peripheral communication device class | Select the usb device class. |
| USB Descriptor | Define with usb_descriptor_t. | g_usb_descriptor | Enter the name of the descriptor to be used. For how to create a descriptor structure, refer to the Descriptor definition chapter in the usb_basic manual. Specify NULL when using the Host class. |
| USB Complience CallBack | Define with usb_compliance_cb_t. | NULL | Member variable for setting callback used in compliance test. |
| USBFS USBI Interrupt Priority | MCU Specific Options | | Select the interrupt priority used by USBI. |
| USBFS USBR Interrupt Priority | MCU Specific Options | | Select the interrupt priority used by USBR. |
| USBFS D0FIFO Interrupt Priority | MCU Specific Options | | Select the interrupt priority used by FS D0FIFO. |
| USBFS D1FIFO Interrupt Priority | MCU Specific Options | | Select the interrupt priority used by FS D1FIFO. |

| | | | |
|---|---|---|---|
| USBHS USBIR Interrupt Priority | MCU Specific Options | | Select the interrupt priority used by USBI. |
| USBHS D0FIFO Interrupt Priority | MCU Specific Options | | Select the interrupt priority used by HS D0FIFO. |
| USBHS D1FIFO Interrupt Priority | MCU Specific Options | | Select the interrupt priority used by HS D1FIFO. |
| USB RTOS CallBack | Enter the address of the function. | NULL | Member variable for storing callbacks used by RTOS. |
| USB Other Context | Enter the address of the context. | NULL | Enter the information you want to specify. |

### Clock Configuration

The USB module uses PLL as the clock source. Set the PLL clock source frequency in the configuration file.

### Pin Configuration

USB input/output pin settings are necessary to use the USB controller. The following is a list of USB pins that need to be set. Set the following pins as necessary.

**USB I/O Pin Settings for USB Peripheral Operation.**

| Pin Name | I/O | Function |
|---|---|---|
| USB_VBUS | input | VBUS pin for USB communication |
| USBHS_VBUS | input | VBUS pin for USB communication |

**USB I/O Pin Settings for USB Host Operation.**

| Pin Name | I/O | Function |
|---|---|---|
| USBHS_VBUSEN | output | VBUS output enabled pin for USB communication |
| USBHS_OVRCURA | input | Overcurrent detection pin for USB communication |
| USBHS_OVRCURB | input | Overcurrent detection pin for USB communication |

### DMA Configuration

To use DMA, select an empty box under Basic in RA Configuration. The box has separate sections for sending and receiving. When you make a selection, a menu will appear, allowing you to select a DMA module. In addition, since the box is separated for transmission and reception, set the DMA in the direction you want to use.

When using DMA with USB, it is necessary to set the DMA items in RA Configuration.

| Config Name | Select Name | Description |
|---|---|---|
| Transfer Size | 2 Bytes<br>4 Bytes | When operating with FS, select "2 Bytes"<br>When operating with HS, select "4 Bytes" |
| Activation source | USBFS FIFO 0<br>USBFS FIFO 1<br>USBHS FIFO 0<br>USBHS FIFO 1 | Select USBFS FIFO 0 when receiving data with FS<br>Select USBFS FIFO 1 when sending data with FS<br>Select USBHS FIFO 0 when receiving data with HS<br>Select USBHS FIFO 1 when sending data with HS |

# Descriptor definition

The usb_descriptor_t structure stores descriptor information such as device descriptor and configuration descriptor.

The descriptor information set in this structure is sent to the USB host as response data to a standard request during enumeration of the USB host.

This structure is specified in the R_USB_Open function argument.

```
typedef struct usb_descriptor
{
     uint8_t *p_device;    /* Note 1 */
     uint8_t *p_config_f;  /* Note 2 */
     uint8_t *p_config_h;  /* Note 3 */
     uint8_t *p_qualifier; /* Note 4 */
     uint8_t **pp_string;  /* Note 5 */
     uint8_t num_string;   /* Note 6 */
} usb_descriptor_t;
```

**Note:**

1. Specify the top address of the area that stores the device descriptor in the member (p_device).
2. Specify the top address of the area that stores the Full-speed configuration descriptor in the member (p_config_f).
   Even when using High speed, make sure you specify the top address of the area that stores the Full-speed configuration descriptor in this member.
3. Specify the top address of the area that stores the High speed configuration descriptor in the member (p_config_h).
   For Full speed, specify USB_NULL to this member.
4. Specify the top address of the area that stores the qualifier descriptor in the member (p_qualifier). For Full speed, specify USB_NULL to this member.

5. Specify the top address of the string descriptor table in the member (pp_string).
   In the string descriptor table,specify the top address of the areas that store each string descriptor.

```
Ex. 1) Full speed                                Ex. 2) High speed

     usb_descriptor_t g_usb_descriptor =              usb_descriptor_t g_usb_
descriptor =
     {                                                {
         smp_device,                                      smp_device,
         smp_config_f,                                    smp_config_f,
         USB_NULL,                                        smp_config_h,
         USB_NULL,                                        smp_qualifier,
         smp_str_table,                                   smp_str_table,
         3,                                               3,
     };                                               };
```

6. Specify the number of the string descriptor which set in the string descriptor table to the member (num_string).
7. After setting the descriptors, enter the name of the structure that stores the start address of each descriptor in the USB Descriptor item of RA Configuration.

### String Descriptor

This USB driver requires each string descriptor that is constructed to be registered in the string descriptor table. The following describes how to register a string descriptor.

1. First construct each string descriptor. Then, define the variable of each string descriptor in uint8_t* type.

```
 Example descriptor construction
    uint8_t smp_str_descriptor0[]
    {
        0x04,      /* Length */
        0x03,      /* Descriptor type */
        0x09, 0x04 /* Language ID */
    };

    uint8_t smp_str_descriptor1[] =
    {
        0x10,      /* Length */
        0x03,      /* Descriptor type */
        'R', 0x00,
        'E', 0x00,
        'N', 0x00,
        'E', 0x00,
        'S', 0x00,
        'A', 0x00,
        'S', 0x00
    };
    uint8_t smp_str_descriptor2[] =
    {
        0x12,      /* Length */
        0x03,      /* Descriptor type */
        'C', 0x00,
        'D', 0x00,
        'C', 0x00,
        '_', 0x00,
```

```
        'D', 0x00,
        'E', 0x00,
        'M', 0x00,
        'O', 0x00
    };
```

2. Set the top address of each string descriptor constructed above in the string descriptor table. Define the variables of the string descriptor table as uint8_t* type.
3. Set the top address of the string descriptor table in the usb_descriptor_t structure member (pp_string).
4. Set the number of the string descriptor which is set in the string descriptor table to usb_descriptor_t structure member (num_string). In the above example, the value 3 is set to the member (num_string).

*Note*

*The position set for each string descriptor in the string descriptor table is determined by the index values set in the descriptor itself (iManufacturer, iConfiguration, etc.).*
*For example, in the table below, the manufacturer is described in smp_str_descriptor1 and the value of iManufacturer in the device descriptor is "1". Therefore, the top address "smp_str_descriptor1" is set at Index 1 in the string descriptor table.*

```
/* String Descriptor table */
    uint8_t *smp_str_table[] =
    {
        smp_str_descriptor0, /* Index: 0 */
        smp_str_descriptor1, /* Index: 1 */
        smp_str_descriptor2, /* Index: 2 */
    };
```

**Other Descriptors**

1. Please construct the device descriptor, configuration descriptor, and qualifier descriptor based on instructions provided in the Universal Serial Bus Revision 2.0 specification (http://www.usb.org/developers/docs/) Each descriptor variable should be defined as uint8_t* type.
2. The top address of each descriptor should be registered in the corresponding usb_descriptor_t function member.

# Usage Notes

### Creating an Application Program

This chapter provides information for creating application programs.

### Descriptor Creation

For USB peripheral operations, your will need to create descriptors to meet your system specifications. Register the created descriptors in the usb_descriptor_t function members. USB host operations do not require creation of special descriptors. Set usb_descriptor_t structure in USB Descriptor of RA Configuation.

### Creation and Registration of Callback Functions (RTOS only)

Create and register a callback function to be registered in RA Configuration. In addition to the USB

completion event, a variety of information about the event is also set by the USB driver. Be sure to notify the application task of the relevant argument information using, for example, the RTOS API. The item to register a callback in RA Configuration is USB RTOS CallBack.

**Creation of Main Routine and Application Program Tasks**

1. Non-OS

   Describe the main routine in the main loop format.
   Make sure you call the R_USB_EventGet function in the main loop.
   The USB-related completed events are obtained from the return value of the R_USB_EventGet function. Also make sure your application program has a routine for each return value.
   The routine is triggered by the corresponding return value
   Note: Carry out USB data communication using the R_USB_Read, R_USB_Write functions after checking the return value USB_STATUS_CONFIGURED of R_USB_EventGet function.

2. RTOS

   Write application program tasks in loop format.
   In the main loop, be sure to call the RTOS API to retrieve the information (USB completion events and the like) that is received as notifications from the callback function.
   Write programs that correspond to the respective USB completion events, with the USB completion events retrieved by the application task as a trigger.

**Registration to the real time OS (RTOS only)**

Register the following in RTOS.

1. Application Program Tasks
2. RTOS features used by application tasks and callback functions

Note: The priority of the application program task is 1 by default. To increase the priority, increase the value of Max Priorities in the RTOS config.

**Limitations**

Developers should be aware of the following limitations when using the USB:

- The current USB driver does not support hub mode.
- In USB host mode, the module does not support suspend during data transfer. Execute suspend only after confirming that data transfer is complete.
- Multiconfigurations are not supported.
- The USB host and USB peripheral modes cannot operate at the same time.
- This USB driver does not support the error processing when the out of specification values are specified to the arguments of each function in the driver.
- This driver does not support the CPU transfer using D0FIFO/D1FIFO register.
- This driver does not support multiple device class drivers at the same time.
- The USB High speed module only supports High speed operation.

# Examples

**USB Basic Example**

This is a basic example of minimal use of the USB in an application.

```
void usb_basic_example (void)
{
    usb_event_info_t event_info;
 usb_status_t    event;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 /* Loop back between PC(TerminalSoft) and USB MCU */
 while (1)
    {
       g_usb_on_usb.eventGet(&event_info, &event);
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
 case USB_STATUS_WRITE_COMPLETE:
                g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PCDC);
 break;
 case USB_STATUS_READ_COMPLETE:
                g_usb_on_usb.write(&g_basic0_ctrl, g_buf, event_info.data_size,
USB_CLASS_PCDC);
 break;
 case USB_STATUS_REQUEST:   /* Receive Class Request */
 if (USB_PCDC_SET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlDataGet(&g_basic0_ctrl, (uint8_t *)
&g_line_coding, LINE_CODING_LENGTH);
                }
 else if (USB_PCDC_GET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, (uint8_t *)
&g_line_coding, LINE_CODING_LENGTH);
                }
 else
      {
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
```

```
USB_SETUP_STATUS_ACK);
                }
 break;
 case USB_STATUS_SUSPEND:
 case USB_STATUS_DETACH:
 break;
 default:
 break;
       }
    }
} /* End of function usb_main() */
```

### Typedefs

| | |
|---|---|
| typedef usb_event_info_t | usb_instance_ctrl_t |

## Typedef Documentation

### ◆ usb_instance_ctrl_t

| |
|---|
| typedef usb_event_info_t usb_instance_ctrl_t |
| ICU private control block. DO NOT MODIFY. Initialization occurs when R_ICU_ExternalIrqOpen is called. |

## Function Documentation

### ◆ R_USB_Open()

| |
|---|
| fsp_err_t R_USB_Open ( usb_ctrl_t *const *p_api_ctrl*, usb_cfg_t const *const *p_cfg* ) |
| Applies power to the USB module specified in the argument (p_ctrl). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success in open. |
| FSP_ERR_USB_BUSY | Specified USB module now in use. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_Close()

fsp_err_t R_USB_Close ( usb_ctrl_t *const *p_api_ctrl*)

Terminates power to the USB module specified in argument (p_ctrl). USB0 module stops when USB_IP0 is specified to the member (module), USB1 module stops when USB_IP1 is specified to the member (module).

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_USB_NOT_OPEN | USB module is not open. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_Read()

fsp_err_t R_USB_Read ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_buf*, uint32_t *size*, uint8_t *destination* )

Bulk/interrupt data transfer and control data transfer.

1. Bulk/interrupt data transfer

   Requests USB data read (bulk/interrupt transfer). The read data is stored in the area specified by argument (p_buf). After data read is completed, confirm the operation by checking the return value (USB_STATUS_READ_COMPLETE) of the R_USB_GetEvent function. The received data size is set in member (size) of the usb_ctrl_t structure. To figure out the size of the data when a read is complete, check the return value (USB_STATUS_READ_COMPLETE) of the R_USB_GetEvent function, and then refer to the member (size) of the usb_crtl_t structure.

2. Control data transfer

   The R_USB_Read function is used to receive data in the data stage and the R_USB_Write function is used to send data to the USB host.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed (Data read request completed). |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_USB_BUSY | Data receive request already in process for USB device with same device address. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*The address specified in the argument p_buf must be 4-byte aligned.*

### ◆ R_USB_Write()

fsp_err_t R_USB_Write ( usb_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_buf*, uint32_t *size*, uint8_t *destination* )

Bulk/Interrupt data transfer and control data transfer.

1. Bulk/Interrupt data transfer

   Requests USB data write (bulk/interrupt transfer). Stores write data in area specified by argument (p_buf). Set the device class type in usb_ctrl_t structure member (type). Confirm after data write is completed by checking the return value (USB_STATUS_WRITE_COMPLETE) of the R_USB_GetEvent function. To request the transmission of a NULL packet, assign USB_NULL(0) to the third argument (size).

2. Control data transfer

   The R_USB_Read function is used to receive data in the data stage and the R_USB_Write function is used to send data to the USB host.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. (Data write request completed) |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_USB_BUSY | Data write request already in process for USB device with same device address. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*The address specified in the argument p_buf must be 4-byte aligned.*

### ◆ R_USB_Stop()

fsp_err_t R_USB_Stop ( usb_ctrl_t *const *p_api_ctrl*, usb_transfer_t *direction*, uint8_t *destination* )

Requests a data read/write transfer be terminated when a data read/write transfer is being performed.

To stop a data read, set USB_TRANSFER_READ as the argument (type); to stop a data write, specify USB_WRITE as the argument (type).

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. (stop completed) |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_BUSY | Stop processing is called multiple times. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_Suspend()

fsp_err_t R_USB_Suspend ( usb_ctrl_t *const *p_api_ctrl*)

Sends a SUSPEND signal from the USB module assigned to the member (module) of the usb_crtl_t structure.

After the suspend request is completed, confirm the operation with the return value (USB_STATUS_SUSPEND) of the R_USB_EventGet function.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_USB_BUSY | During a suspend request to the specified USB module, or when the USB module is already in the suspended state. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_Resume()

fsp_err_t R_USB_Resume ( usb_ctrl_t *const *p_api_ctrl*)

Sends a RESUME signal from the USB module assigned to the member (module) of the usb_ctrl_tstructure.

After the resume request is completed, confirm the operation with the return value (USB_STATUS_RESUME) of the R_USB_EventGet function

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_USB_BUSY | Resume already requested for same device address. (USB host mode only) |
| FSP_ERR_USB_NOT_SUSPEND | USB device is not in the SUSPEND state. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_VbusSet()

fsp_err_t R_USB_VbusSet ( usb_ctrl_t *const *p_api_ctrl*, uint16_t *state* )

Specifies starting or stopping the VBUS supply.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. (VBUS supply start/stop completed) |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_InfoGet()

fsp_err_t R_USB_InfoGet ( usb_ctrl_t *const *p_api_ctrl*, usb_info_t * *p_info*, uint8_t *destination* )

Obtains completed USB-related events.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. (VBUS supply start/stop completed) |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_PipeRead()

fsp_err_t R_USB_PipeRead ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_buf*, uint32_t *size*, uint8_t *pipe_number* )

Requests a data read (bulk/interrupt transfer) via the pipe specified in the argument.

The read data is stored in the area specified in the argument (p_buf). After the data read is completed, confirm the operation with the R_USB_GetEvent function return value(USB_STATUS_READ_COMPLETE). To figure out the size of the data when a read is complete, check the return value (USB_STATUS_READ_COMPLETE) of the R_USB_GetEvent function, and then refer to the member (size) of the usb_crtl_t structure.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_BUSY | Specified pipe now handling data receive/send request. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*The address specified in the argument p_buf must be 4-byte aligned.*

### ◆ R_USB_PipeWrite()

fsp_err_t R_USB_PipeWrite ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_buf*, uint32_t *size*, uint8_t *pipe_number* )

Requests a data write (bulk/interrupt transfer).

The write data is stored in the area specified in the argument (p_buf). After data write is completed, confirm the operation with the return value (USB_STATUS_WRITE_COMPLETE) of the EventGet function. To request the transmission of a NULL packet, assign USB_NULL (0) to the third argument (size).

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_BUSY | Specified pipe now handling data receive/send request. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*The address specified in the argument p_buf must be 4-byte aligned.*

### ◆ R_USB_PipeStop()

fsp_err_t R_USB_PipeStop ( usb_ctrl_t *const *p_api_ctrl*, uint8_t *pipe_number* )

Terminates a data read/write operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. (Stop request completed) |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_UsedPipesGet()

fsp_err_t R_USB_UsedPipesGet ( usb_ctrl_t *const *p_api_ctrl*, uint16_t * *p_pipe*, uint8_t *destination* )

Gets the selected pipe number (number of the pipe that has completed initalization) via bit map information.

The bit map information is stored in the area specified in argument (p_pipe). Based on the information (module member and address member) assigned to the usb_ctrl_t structure, obtains the PIPE information of that USB device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_PipeInfoGet()

fsp_err_t R_USB_PipeInfoGet ( usb_ctrl_t *const *p_api_ctrl*, usb_pipe_t * *p_info*, uint8_t *pipe_number* )

Gets the following pipe information regarding the pipe specified in the argument (p_ctrl) member (pipe): endpoint number, transfer type, transfer direction and maximum packet size.

The obtained pipe information is stored in the area specified in the argument (p_info).

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_PullUp()

fsp_err_t R_USB_PullUp ( usb_ctrl_t *const *p_api_ctrl*, uint8_t *state* )

This API enables or disables pull-up of D+/D- line.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. (Pull-up enable/disable setting completed) |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_EventGet()

fsp_err_t R_USB_EventGet ( usb_ctrl_t *const *p_api_ctrl*, usb_status_t * *event* )

Obtains completed USB related events. (OS-less Only)

In USB host mode, the device address value of the USB device that completed an event is specified in the usb_ctrl_t structure member (address) specified by the event's argument. In USB peripheral mode, USB_NULL is specified in member (address). If this function is called in the RTOS execution environment, a failure is returned.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Event Get Success. |
| FSP_ERR_USB_FAILED | If called in the RTOS environment, an error is returned. |

***Note***

*Do not use the same variable as the first argument of R_USB_Open for the first argument.*

### ◆ R_USB_VersionGet()

fsp_err_t R_USB_VersionGet ( fsp_version_t *const *p_version*)

Returns the version of this module.

The version number is encoded such that the top two bytes are the major version number and the bottom two bytes are the minor version number.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_ASSERTION | Failed in acquiring version information. |

### ◆ R_USB_Callback()

fsp_err_t R_USB_Callback ( usb_callback_t * *p_callback*)

Register a callback function to be called upon completion of a USB related event. (RTOS only)

This function registers a callback function to be called when a USB-related event has completed. If this function is called in the OS-less execution environment, a failure is returned.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_FAILED | If this function is called in the OS-less execution environment, a failure is returned. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |

### ◆ R_USB_HostControlTransfer()

fsp_err_t R_USB_HostControlTransfer ( usb_ctrl_t *const *p_api_ctrl*, usb_setup_t * *p_setup*, uint8_t * *p_buf*, uint8_t *device_address* )

Performs settings and transmission processing when transmitting a setup packet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |
| FSP_ERR_USB_BUSY | Specified pipe now handling data receive/send request. |

***Note***

*The address specified in the argument p_buf must be 4-byte aligned.*

### ◆ R_USB_PeriControlDataGet()

fsp_err_t R_USB_PeriControlDataGet ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_buf*, uint32_t *size* )

Receives data sent by control transfer.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_BUSY | Specified pipe now handling data receive/send request. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*The address specified in the argument p_buf must be 4-byte aligned.*

### ◆ R_USB_PeriControlDataSet()

fsp_err_t R_USB_PeriControlDataSet ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_buf*, uint32_t *size* )

Performs transfer processing for control transfer.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_BUSY | Specified pipe now handling data receive/send request. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*The address specified in the argument p_buf must be 4-byte aligned.*

### ◆ R_USB_PeriControlStatusSet()

fsp_err_t R_USB_PeriControlStatusSet ( usb_ctrl_t *const *p_api_ctrl*, usb_setup_status_t *status* )

Set the response to the setup packet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |

### ◆ R_USB_RemoteWakeup()

fsp_err_t R_USB_RemoteWakeup ( usb_ctrl_t *const *p_api_ctrl*)

Sends a remote wake-up signal to the connected Host.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_NOT_SUSPEND | Device is not suspended. |
| FSP_ERR_USB_BUSY | The device is in resume operation. |

### ◆ R_USB_ModuleNumberGet()

fsp_err_t R_USB_ModuleNumberGet ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *module_number* )

This API gets the module number.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |

### ◆ R_USB_ClassTypeGet()

fsp_err_t R_USB_ClassTypeGet ( usb_ctrl_t *const *p_api_ctrl*, usb_class_t * *class_type* )

This API gets the class type.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |

### ◆ R_USB_DeviceAddressGet()

fsp_err_t R_USB_DeviceAddressGet ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *device_address* )

This API gets the device address.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |

### ◆ R_USB_PipeNumberGet()

fsp_err_t R_USB_PipeNumberGet ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *pipe_number* )

This API gets the pipe number.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |

### ◆ R_USB_DeviceStateGet()

fsp_err_t R_USB_DeviceStateGet ( usb_ctrl_t *const *p_api_ctrl*, uint16_t * *state* )

This API gets the state of the device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |

### ◆ R_USB_DataSizeGet()

| fsp_err_t R_USB_DataSizeGet ( usb_ctrl_t *const *p_api_ctrl*, uint32_t * *data_size* ) | |
|---|---|
| This API gets the data size.<br><br>**Return values** | |
| FSP_SUCCESS | Successful completion. |

### ◆ R_USB_SetupGet()

| fsp_err_t R_USB_SetupGet ( usb_ctrl_t *const *p_api_ctrl*, usb_setup_t * *setup* ) | |
|---|---|
| This API gets the setup type.<br><br>**Return values** | |
| FSP_SUCCESS | Successful completion. |

## 4.2.47 USB Host Communications Device Class Driver (r_usb_hcdc)

Modules

This module is USB Host Communication Device Class Driver (HCDC). It implements the USB HCDC Interface. This module works in combination with (r_usb_basic module).

### Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

### Detailed Description

# Overview

The r_usb_hcdc module, when used in combination with the r_usb_basic module, operates as a USB host communications device class driver (HCDC). The HCDC conforms to the PSTN device subclass abstract control model of the USB communication device class specification (CDC) and enables communication with a CDC peripheral device.

### Features

The r_usb_hcdc module has the following key features:

- Checking of connected devices.
- Implementation of communication line settings.
- Acquisition of the communication line state.
- Data transfer to and from a CDC peripheral device.
- HCDC can connect maximum 2 CDC devices to 1 USB module by using USB Hub.

### Communication Device Class (CDC), PSTN and ACM

This software conforms to the Abstract Control Model (ACM) subclass of the Communication Device Class specification, as specified in detail in the PSTN Subclass document listed in 'Related Documents'.

The Abstract Control Model subclass is a technology that bridges the gap between USB devices and earlier modems (employing RS-232C connections), enabling use of application programs designed for older modems.

### Basic Functions

The main functions of HCDC are as follows.

- Verify connected devices
- Make communication line settings
- Acquire the communication line state
- Transfer data to and from the CDC peripheral device

### Abstract Control Model Class Requests - Host to Device

This driver supports the following class requests.

CDC Class Requests

| Request | Code | Description |
|---|---|---|
| SendEncapsulatedCommand | 0x00 | Transmits an AT command as defined by the protocol used by the device (normally 0 for USB). |
| GetEncapsulatedResponse | 0x01 | Requests a response to a command transmitted by SendEncapsulatedCommand. |
| SetCommFeature | 0x02 | Enables or disables features such as device-specific 2-byte code and country setting. |
| GetCommFeature | 0x03 | Acquires the enabled/disabled state of features such as device-specific 2-byte code and country setting. |
| ClearCommFeature | 0x04 | Restores the default enabled/disabled settings of features such as device-specific 2-byte code and country setting. |
| SetLineCoding | 0x20 | Makes communication line |

| | | |
|---|---|---|
| | | settings (communication speed, data length, parity bit, and stop bit length). |
| GetLineCoding | 0x21 | Acquires the communication line setting state. |
| SetControlLineState | 0x22 | Makes communication line control signal (RTS, DTR) settings. |
| SendBreak | 0x23 | Transmits a break signal. |

For details concerning the Abstract Control Model requests, refer to Table 11, 'Requests - Abstract Control Model' in 'USB Communications Class Subclass Specification for PSTN Devices', Revision 1.2.

**SendEncapsulatedCommand**

The SendEncapsulatedCommand data format is shown below.

SendEncapsulatedCommand Data Format

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0x21 | SEND_ENCAPSULATED_COMMAND(0x00) | 0x0000 | 0x0000 | Data length | Control protoco command |

*Note*

*Items such as AT commands for modem control are set as Data, and wLength is set to match the length of the data.*

**GetEncapsulatedResponse**

The GetEncapsulatedResponse data format is shown below.

GetEncapsulatedResponse Data Format

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0x21 | GET_ENCAPSULATED_RESPONSE (0x01) | 0x0000 | 0x0000 | Data length | The data depends on the protocol. |

*Note*

*The response data to SendEncapsulatedCommand is set as Data, and wLength is set to match the length of the data.*

**SetCommFeature**

The SetCommFeature data format is shown below.

SetCommFeature Data Format

| | | | | | |
|---|---|---|---|---|---|

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0x21 | SET_COMM_FEATURE(0x02) | Feature Selector | 0x0000 | Data length | Status Either the country code or the Abstract Control Model idle setting/multiplexing setting for Feature Selector. |

*Note*

*See Feature Selector setting list.*

### GetCommFeature Data Format

The GetCommFeature data format is shown below.

GetCommFeature Data Format

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0x21 | GET_COMM_FEATURE(0x03) | Feature Selector | 0x0000 | Data length | Status Either the country code or the Abstract Control Model idle setting/multiplexing setting for Feature Selector. |

*Note*

*See Feature Selector setting list.*

Feature Selector Settings

| Feature Selector | Code | Targets | Length of Data | Description |
|---|---|---|---|---|
| RESERVED | 0x00 | None | None | Reserved |
| ABSTRACT_STATE | 0x01 | Interface | 2 | Selects the setting for Abstract Control Model idle state and signal multiplexing. |
| COUNTRY_SETTING | 0x02 | Interface | 2 | Selects the country code in hexadecimal format, as defined |

by ISO 3166.

Status Format when ABSTRACT_STATE Selected

| Bit Position | Description |
|---|---|
| D15 to D2 | Reserved |
| D1 | Data multiplexing setting<br>1: Multiplexing of call management commands is enabled for the Data class.<br>0: Multiplexing is disabled. |
| D0 | Idle setting<br>1: No endpoints of the target interface accept data from the host, and data is not supplied to the host.<br>0: Endpoints continue to accept data and it is supplied to the host. |

**ClearCommFeature**

The ClearCommFeature data format is shown below.

ClearCommFeature Data Format

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0x21 | CLEAR_COMM_FEATURE (0x04) | Feature Selector | 0x0000 | Data length | None |

*Note*

*See Feature Selector setting list.*

**SetLineCoding**

The SetLineCoding data format is shown below.

SetLineCoding Data Format

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0x21 | SET_LINE_CODING(0x20) | 0x0000 | 0x0000 | 0x0000 | Line Coding Structure See Line Coding Structure format. |

Line Coding Structure Format

| | | | | |
|---|---|---|---|---|

| Offset | Field | size | Value | Description |
|---|---|---|---|---|
| 0 | dwDTERate | 4 | Number | Data terminal speed (bps) |
| 4 | bCharFormat | 1 | Number | Stop bits<br>0 - 1 stop bit<br>1 - 1.5 stop bits<br>2 - 2 stop bits |
| 5 | bParityType | 1 | Number | Parity<br>0 - None<br>1 - Odd<br>2 - Even<br>3 - Mask<br>4 - Space |
| 6 | bDataBits | 1 | Number | Data bits (5, 6, 7, 8) |

**GetLineCoding**

The GetLineCoding data format is shown below.

GetLineCoding Data Format

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0xA1 | GET_LINE_CODING(0x21) | 0x0000 | 0x0000 | 0x0007 | Line Coding Structure See Line Coding Structure format. |

**SetControlLineState**

The SetControlLineState data format is shown below.

SetControlLineState Data Format

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0x21 | SET_CONTROL_LINE_STATE (0x22) | Control Signal Bitmap Control Signal Refer to bit map format. | 0x0000 | 0x0000 | None |

Control Signal Bitmap

| Bit Position | Description |
|---|---|

| | |
|---|---|
| D15 to D2 | Reserved |
| D1 | DCE transmit function control<br>0 - RTS OFF<br>1 - RTS ON |
| D0 | Notification of DTE ready state<br>0 - DTR OFF<br>1 - DTR ON |

**SendBreak**

The SendBreak data format is shown below.

SendBreak Data Format

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0x21 | SEND_BREAK (0x23) | Break signal output duration | 0x0000 | 0x0000 | None |

**ACM Notifications from Device to Host**

The following are the class notifications supported and not supported by the software.

CDC Class Notifications

| Notification | Code | Description | Supported |
|---|---|---|---|
| NETWORK_CONNECTION | 0x00 | Notification of network connection state | No |
| RESPONSE_AVAILABLE | 0x01 | Response to GET_ENCAPSLATED_RESPONSE | Yes |
| SERIAL_STATE | 0x20 | Notification of serial line state | Yes |

**SerialState**

The SerialState data format is shown below.

SerialState Data Format

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0xA1 | SERIAL_STATE(0x20) | 0x0000 | 0x0000 | 0x0000 | UART State bitmap<br>See UART State bitmap format. |

UART State bitmap format is shown below.

| Bit Position | Field | Description |
|---|---|---|
| D15 to D7 | | Reserved |
| D6 | bOverRun | Overrun error detected |
| D5 | bParity | Parity error detected |
| D4 | bFraming | Framing error detected |
| D3 | bRingSignal | INCOMING signal (ring signal) detected |
| D2 | bBreak | Break signal detected |
| D1 | bTxCarrier | Data Set Ready: Line connected and ready for communication |
| D0 | bRxCarrier | Data Carrier Detect: Carrier detected on line |

**ResponseAvailable**

The ResponseAvailable data format is shown below.

ResponseAvailable Data Format

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0xA1 | RESPONSE_AVAILABLE(0x01) | 0x0000 | 0x0000 | 0x0000 | None |

**USB Host Communication Device Class Driver (HCDC)**

**Basic Functions**

This software conforms to the Abstract Control Model subclass of the communication device class specification.

The main functions of HCDC are to:

- Send class requests to the CDC peripheral
- Transfer data to and from the CDC peripheral
- Receive communication error information from the CDC peripheral

**Structure / Union**

The following structure or union is defined in r_usb_hcdc_api.h.

**HCDC Request Structure**

Below describes the 'UART settings' parameter structure used for the CDC requests SetLineCoding and GetLineCoding.

usb_hcdc_linecoding_t Structure

| Type | Member | Description | Remarks |
|---|---|---|---|
| uint32_t | dwdte_rate | Line speed | Unit: bps |
| uint8_t | bchar_format | Stop bits setting | |
| uint8_t | bparity_type | Parity setting | |
| uint8_t | bdata_bits | Data bit lengt | |

usb_hcdc_controllinestate_t Structure

| Type | Member | Description | Remarks |
|---|---|---|---|
| uint16_t (D1) | brts:1 | Carrier control for half duplex modems<br>0 - Deactivate carrier, 1 - Activate carrier | |
| uint16_t (D0) | brts:1 | Indicates to DCE if DTE is present or not<br>0 - Not Present, 1 - Present | |

Below describes the 'AT command' parameter structure used for the CDC requests SendEncapsulatedCommand and GetEncapsulatedResponse.

| Type | Member | Description | Remarks |
|---|---|---|---|
| uint8_t | *p_data | Area where AT command data is stored | |
| uint16_t | wlength | Size of AT command data | Unit: byte |

Below describes the 'Break signal' parameter structure used for the CDC requests SendBreak.

| Type | Member | Description | Remarks |
|---|---|---|---|
| uint16_t | wtime_ms | Duration of Break | Unit: ms |

**CommFeature Function Selection Union**

usb_hcdc_abstractstate_t Structure and and usb_hcdc_countrysetting_t Structure describe the 'Feature Selector' parameter structure used for the CDC requests SetCommFeature and GetCommFeature, and usb_hcdc_commfeature_t Union describes the parameter union.

usb_hcdc_abstractstate_t Structure

| Type | Member | Description | Remarks |
|---|---|---|---|
| uint16_t | rsv1:14 | Reserved | |

| | | |
|---|---|---|
| uint16_t | bdms:1 | Data Multiplexed State |
| iomt16_t | bis:1 | Idle Setting |

usb_hcdc_countrysetting_t Structure

| Type | Member | Description | Remarks |
|---|---|---|---|
| uint16_t | country_code | Country code in hexadecimal format as defined in [ISO3166], | |

usb_hcdc_commfeature_t Union

| Type | Member | Description | Remarks |
|---|---|---|---|
| usb_hcdc_abstractstate_t | abstract_state | Parameter when selecting Abstract Control Model | |
| usb_hcdc_countrysetting_t | country_setting | Parameter when selecting Country Setting | |

**CDC Notification Format**

'Response_Available notification format' and 'Serial_State notification format' describe the data format of the CDC notification.

Response_Available notification format

| Type | Member | Description | Remarks |
|---|---|---|---|
| uint8_t | bmRequestType | 0xA1 | |
| uint8_t | bRequest | RESPONSE_AVAILABLE(0x01) | |
| uint16_t | wValue | 0x0000 | |
| uint16_t | wIndex | Interface | |
| uint16_t | wLength | 0x0000 | |
| uint8_t | Data | none | |

Serial_State notification format

| Type | Member | Description | Remarks |
|---|---|---|---|
| uint8_t | bmRequestType | 0xA1 | |
| uint8_t | bRequest | SERIAL_STATE(0x20) | |
| uint16_t | wValue | 0x0000 | |
| uint16_t | wIndex | Interface | |

| | | | |
|---|---|---|---|
| uint16_t | wLength | 0x0002 | |
| uint8_t | Data | UART State bitmap | Refer to 'usb_hcdc_serialstate_t Structure' |

The host is notified of the 'SerialState' when a change in the UART port state is detected. 'usb_hcdc_serialstate_t Structure' describes the structure of the UART State bitmap.

usb_hcdc_serialstate_t Structure

| Type | Member | Description | Remarks |
|---|---|---|---|
| uint16_t (D15-D7) | rsv1:9 | Reserved | |
| uint16_t (D6) | bover_run:1 | Overrun error detected | |
| uint16_t (D5) | bparity:1 | Parity error detected | |
| uint16_t (D4) | bframing:1 | Framing error detected | |
| uint16_t (D3) | bring_signal:1 | Incoming signal (Ring signal) detected | |
| uint16_t (D2) | bbreak:1 | Break signal detected | |
| uint16_t (D1) | btx_carrier:1 | Line connected and ready for communication | Data Set Ready |
| uint16_t (D0) | brx_carrier:1 | Carrier detected on line | Data Carrier Detect |

# Configuration

### Build Time Configurations for r_usb_hcdc

The following build time configurations are defined in fsp_cfg/r_usb_hcdc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Multiple connection Setting | • Multiple connection not supported<br>• Multiple connection supported | Multiple connection not supported | Currently, multiple connections are not available. |
| Specify the device class ID of the CDC device to be connected. | • CDC class supported device<br>• Vendor class device | CDC class supported device | Specify the device class ID of the CDC device to be connected. |
| Pipe to use for Bulk IN transfer. | • USB PIPE1<br>• USB PIPE2<br>• USB PIPE3<br>• USB PIPE4 | USB PIPE1 | Please choose between 1 and 5. |

| | | | |
|---|---|---|---|
| | • USB PIPE5 | | |
| Pipe to use for Bulk OUT transfer. | • USB PIPE1<br>• USB PIPE2<br>• USB PIPE3<br>• USB PIPE4<br>• USB PIPE5 | USB PIPE2 | Please choose between 1 and 5. |
| Pipe to use for Interrupt IN transfer. | • USB PIPE6<br>• USB PIPE7<br>• USB PIPE8<br>• USB PIPE9 | USB PIPE6 | Please choose between 6 and 9. |

**Configurations for Middleware > USB > USB HCDC driver on r_usb_hcdc**

This module can be added to the Stacks tab via New Stack > Middleware > USB > USB HCDC driver on r_usb_hcdc:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_hcdc0 | Module name. |

**Clock Configuration**

Refer to r_usb_basic module.

**Pin Configuration**

Refer to r_usb_basic module.

# Usage Notes

**Limitations**

This driver is subject to the following limitations.

1. Only one stage of the USB hub can be used.
2. Suspend and resume are not supported for CDC devices connected to the USB hub and USB hub downstream ports.
3. Suspend is not supported when data transfer is in progress. Confirm that data transfer has completed before executing suspend.
4. Use of compound USB devices with CDC class support is not supported.
5. This module needs to be incorporated into a project using r_usb_basic. Once incorporated into a project, use the API to perform USB hardware control.

# Examples

**USB HCDC Example**

**Application Specifications**

The main functions of the APL are as follows:

1. Sends receive (Bulk In transfer) requests to the CDC device and receives data.
2. Transfers received data to the CDC device by means of Bulk Out transfers (loopback).
3. The communication speed and other settings are made by transmitting the class request SET_LINE_CODING to the CDC device. This class request can be used to set the communication speed, number of data bits, number of stop bits, and the parity bit.

### Data Transfer Image

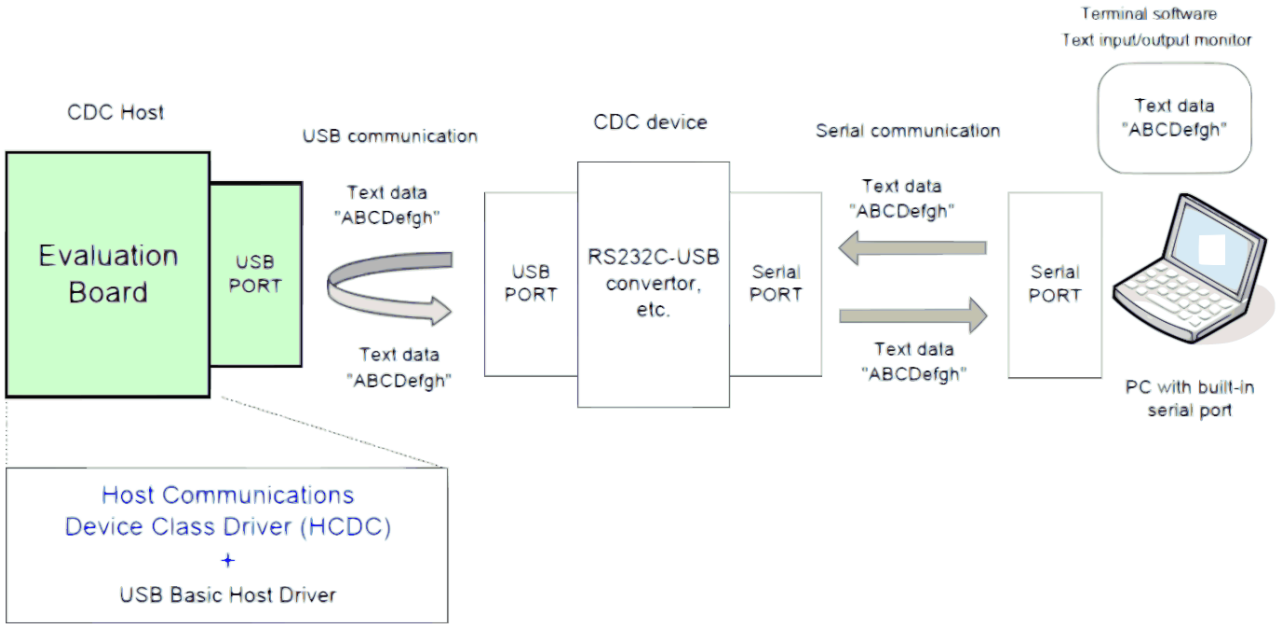



Figure 128: Data Transfer (Loopback)

### Application Processing

The application comprises two parts: initial settings and main loop. An overview of the processing in these two parts is provided below.

### Initial setting

Initial settings consist of MCU pin settings, USB driver settings, and initial settings to the USB controller.

### Main Loop (for RTOS)

The loop performs loop-back processing in which data received from the CDC device is transmitted unaltered back to the CDC device as part of the main routine. An overview of the processing performed by the loop is shown below.

1. When a USB-related event has completed, the USB driver calls the callback function (usb_apl_callback). In the callback function (usb_apl_callback), the application task (APL) is notified of the USB completion event using the real-time OS functionality.
2. In APL, information regarding the USB completion event was notified from the callback function is retrieved using the real-time OS functionality.
3. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STS_CONFIGURED, APL sends the class request (SET_LINECODING) to the CDC device.

4. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STS_REQUEST_COMPLETE, APL performs a data reception request to receive data transmitted from the CDC device by calling the R_USB_Read function and also performs a class notification reception request from CDC device.
5. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STS_READ_COMPLETE, APL performs a data transmission request to send the reception data by calling the R_USB_Write function. The reception data is stored in the gloval variable (g_data). The reception data size is set in the member (size) of the usb_ctrl_t structure. If this member (size) is zero, the USB driver judges that the NULL packet is received and performs a data reception request to the CDC device again.
6. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STS_WRITE_COMPLETE, APL performs a data reception request to receive the data sent from CDC device.
7. The avove processing is repeated.

An overview of the processing performed by the APL is shown below:

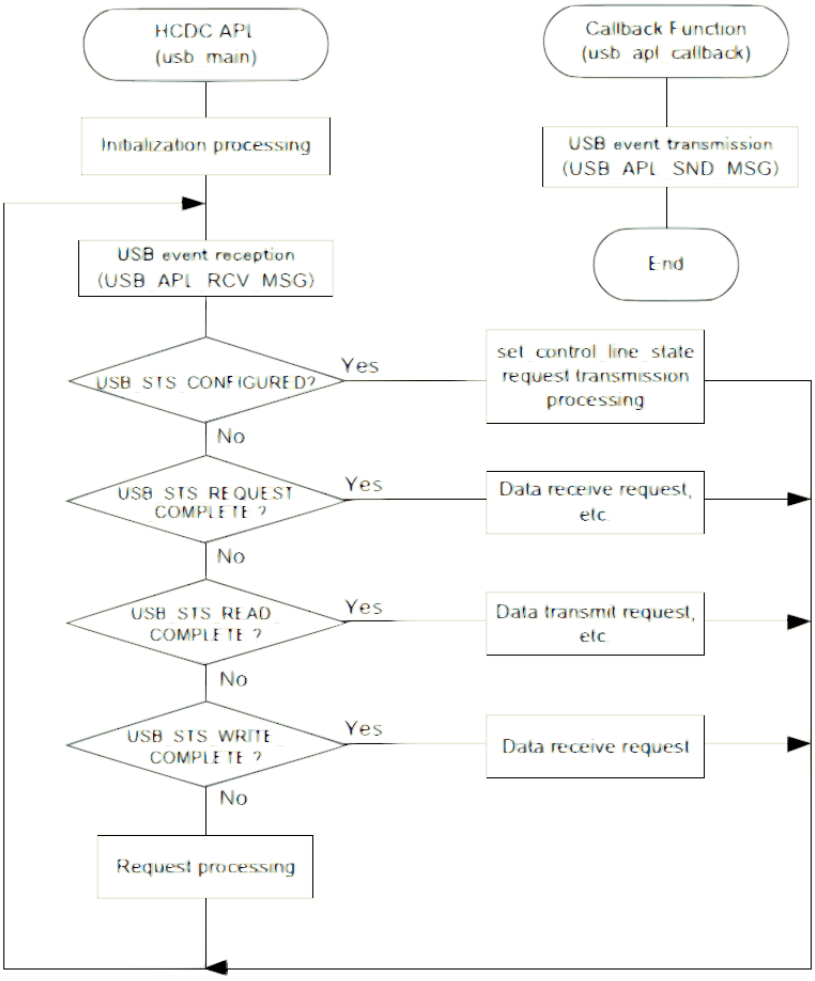



Figure 129: Main Loop processing (for RTOS)

### Connecting Multiple CDC Devices

This is a hcdc example of minimal use of the USB in an application.

```
#define SET_LINE_CODING (USB_CDC_SET_LINE_CODING | USB_HOST_TO_DEV | USB_CLASS |
USB_INTERFACE)
#define GET_LINE_CODING (USB_CDC_GET_LINE_CODING | USB_DEV_TO_HOST | USB_CLASS |
USB_INTERFACE)
#define SET_CONTROL_LINE_STATE (USB_CDC_SET_CONTROL_LINE_STATE | USB_HOST_TO_DEV |
USB_CLASS | USB_INTERFACE)
void usb_basic_example (void)
{
 usb_status_t    event;
    usb_event_info_t event_info;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
 /* Message reception processing is performed here. */
 /* Analyzing the received message */
       g_usb_on_usb.eventGet(&event_info, &event);
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
                set_line_coding(&g_basic0_ctrl, event_info.device_address); /* CDC
Class request "SetLineCoding" */
 break;
 case USB_STATUS_READ_COMPLETE:
 if (USB_CLASS_HCDC == event_info.type)
      {
 if (event_info.data_size > 0)
      {
 /* Send the received data to USB Host */
                       g_usb_on_usb.write(&g_basic0_ctrl, g_snd_buf,
event_info.data_size, USB_DEVICE_ADDRESS_1);
                   }
 else
      {
 /* Send the data reception request when the zero-length packet is received. */
```

```
                        g_usb_on_usb.read(&g_basic0_ctrl, g_rcv_buf, CDC_DATA_LEN,
USB_DEVICE_ADDRESS_1);
                    }
                }
 else
 /* USB_HCDCC */
      {
 /* Class notification "SerialState" receive start */
                    g_usb_on_usb.read(&g_basic0_ctrl,
                                      (uint8_t *) &g_serial_state,
                                      USB_HCDC_SERIAL_STATE_MSG_LEN,
                                      USB_DEVICE_ADDRESS_1);
                }
 break;
 case USB_STATUS_WRITE_COMPLETE:
 /* Report receive start */
                g_usb_on_usb.read(&g_basic0_ctrl, g_rcv_buf, CDC_DATA_LEN,
USB_DEVICE_ADDRESS_1);
 break;
 case USB_STATUS_REQUEST_COMPLETE:
 /* Check Complete request "SetLineCoding" */
 if (USB_CDC_SET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
 /* Class notification "SerialState" receive start */
                    set_control_line_state(&g_basic0_ctrl, event_info.device_address);
/* CDC Class request "SetControlLineState" */
                }
 /* Check Complete request "SetControlLineState" */
 else if (USB_CDC_SET_CONTROL_LINE_STATE == (event_info.setup.request_type &
USB_BREQUEST))
      {
                    get_line_coding(&g_basic0_ctrl, event_info.device_address); /* CDC
Class request "SetLineCoding" */
                }
```

```
 else if (USB_CDC_GET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.write(&g_basic0_ctrl, g_snd_buf, CDC_DATA_LEN,
USB_DEVICE_ADDRESS_1);
                }
 else
      {
 /* Not support request */
                }
 break;
 default:                    /* Other event */
 break;
       }
    }
} /* End of function usb_main */
void set_control_line_state (usb_instance_ctrl_t * p_ctrl, uint8_t device_address)
{
    usb_setup_t setup;
    setup.request_type   = SET_CONTROL_LINE_STATE; /*
bRequestCode:SET_CONTROL_LINE_STATE, bmRequestType */
    setup.request_value = 0x0000;                /* wValue:Zero */
    setup.request_index = 0x0000;                /* wIndex:Interface */
    setup.request_length = 0x0000;                /* wLength:Zero */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_usb_dummy,
device_address);
} /* End of function cdc_set_control_line_state */
void set_line_coding (usb_instance_ctrl_t * p_ctrl, uint8_t device_address)
{
    usb_setup_t setup;
    g_com_parm.dwdte_rate   = (uint32_t) COM_SPEED;
    g_com_parm.bdata_bits   = COM_DATA_BIT;
    g_com_parm.bchar_format = COM_STOP_BIT;
    g_com_parm.bparity_type = COM_PARITY_BIT;
    setup.request_type   = SET_LINE_CODING;    /* bRequestCode:SET_LINE_CODING,
```

```
bmRequestType */
    setup.request_value = 0x0000;             /* wValue:Zero */
    setup.request_index = 0x0000;             /* wIndex:Interface */
    setup.request_length = LINE_CODING_LENGTH; /* Data:Line Coding Structure */
 /* Request Control transfer */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_com_parm,
device_address);
} /* End of function cdc_set_line_coding */
void get_line_coding (usb_instance_ctrl_t * p_ctrl, uint8_t device_address)
{
    usb_setup_t setup;
    setup.request_type   = GET_LINE_CODING;    /* bRequestCode:GET_LINE_CODING,
bmRequestType */
    setup.request_value = 0x0000;             /* wValue:Zero */
    setup.request_index = 0x0000;             /* wIndex:Interface */
    setup.request_length = LINE_CODING_LENGTH; /* Data:Line Coding Structure */
 /* Request Control transfer */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_com_parm,
device_address);
} /* End of function cdc_get_line_coding */
```

## 4.2.48 USB Host Human Interface Device Class Driver (r_usb_hhid)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | R_USB_HHID_TypeGet (usb_ctrl_t *const p_api_ctrl, uint8_t *p_type, uint8_t device_address)<br>Get HID protocol.(USB Mouse/USB Keyboard/Other Type.) More... |
| fsp_err_t | R_USB_HHID_MaxPacketSizeGet (usb_ctrl_t *const p_api_ctrl, uint16_t *p_size, uint8_t direction, uint8_t device_address)<br>Obtains max packet size for the connected HID device. The max packet size is set to the area. Set the direction (USB_HID_IN/USB_HID_OUT). More... |

### Detailed Description

The USB module (r_usb_hhid) provides an API to perform hardware control of USB communications. It implements the USB HHID Interface.

This module is USB Basic Host and Peripheral. It works in combination with Driver (r_usb_basic module).

# Overview

The r_usb_hhid module, when used in combination with the r_usb_basic module, operates as a USB host human interface device class driver (HHID).

### Features

The r_usb_hhid module has the following key features:

- Data communication with a connected HID device (USB mouse, USB keyboard)
- Issuing of HID class requests to a connected HID device
- Supporting Interrupt OUT transfer

### Class Driver Overview

#### 1. Class Requests

The class requests supported by this driver are shown below.

| Request | Code | Description |
|---|---|---|
| USB_GET_REPORT | 0x01 | Receives a report from the HID device. |
| USB_SET_REPORT | 0x09 | Sends a report to the HID device. |
| USB_GET_IDLE | 0x02 | Receives a duration (time) from the HID device. |
| USB_SET_IDLE | 0x0A | Sends a duration (time) to the HID device. |
| USB_GET_PROTOCOL | 0x03 | Reads a protocol from the HID device. |
| USB_SET_PROTOCOL | 0x0B | Sends a protocol to the HID device. |
| USB_GET_REPORT_DESCRIPTOR | 0x06 | Transmits report descriptor. |
| USB_GET_HID_DESCRIPTOR | 0x06 | Transmits an HID descriptor. |

#### 2. Data Format

The boot protocol data format of data received from the keyboard or mouse through interrupt-IN transfers is shown below.

| offset | Keyboard (8 Bytes) | Mouse (3 Bytes) |
|---|---|---|
| 0 (Top Byte) | Modifier keys | b0 : Button 1<br>b1 : Button 2<br>b2 : Button 3<br>b3-b7 : Reserved |
| +1 | Reserved | X displacement |
| +2 | Keycode 1 | Y displacement |
| +3 | Keycode 2 | - |
| +4 | Keycode 3 | - |
| +5 | Keycode 4 | - |
| +6 | Keycode 5 | - |
| +7 | Keycode 6 | - |

# Configuration

### Build Time Configurations for r_usb_hhid

The following build time configurations are defined in fsp_cfg/r_usb_hhid_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Pipe to use for Interrupt IN transfer. | • USB PIPE6<br>• USB PIPE7<br>• USB PIPE8<br>• USB PIPE9 | USB PIPE6 | Please choose between 6 and 9. |
| Pipe to use for Interrupt OUT transfer. | • USB PIPE6<br>• USB PIPE7<br>• USB PIPE8<br>• USB PIPE9 | USB PIPE9 | Please choose between 6 and 9. |

### Configurations for Middleware > USB > USB HHID driver on r_usb_hhid

This module can be added to the Stacks tab via New Stack > Middleware > USB > USB HHID driver on r_usb_hhid:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_hhid0 | Module name. |

### Clock Configuration

Refer to USB (r_usb_basic) basic module.

### Pin Configuration

Refer to USB (r_usb_basic) basic module.

# Usage Notes

- This driver is not guaranteed to provide USB communication operation. The customer should verify operation when utilizing it in a system and confirm the ability to connect to a variety of different types of devices.

### Limitations

1. The HID driver does not analyze the report descriptor. This driver determines the report format from the interface protocol.
2. This driver does not support DMA transfer.

# Examples

### USB HHID Example

### Example Operating Environment

The following shows an example operating environment for the HHID.

Refer to the associated instruction manuals for details on setting up the evaluation board and using the emulator, etc.

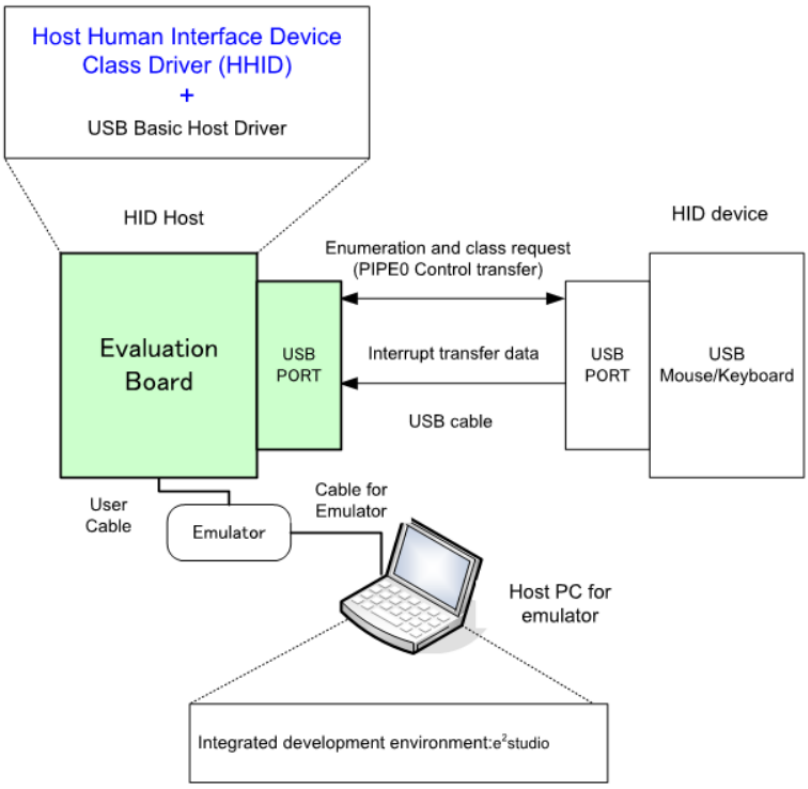



Figure 130: Example Operating Environment

### Application Specifications

The main functions of the application are as follows:

1. Performs enumeration and drive recognition processing on HID devices.
2. Transfers data to and from an HID device (mouse or keyboard) connected to the evaluation board. Data received from the HID device is read and discarded.

### Application Processing (for RTOS)

The application comprises two parts: initial settings and main loop. An overview of the processing in these two parts is provided below.

### Initial setting

Initial settings consist of MCU pin settings, USB driver settings, and initial settings to the USB controller.

### usb_apl_task

The main loop performs processing to receive data from the HID device as part of the main routine. An overview of the processing performed by the loop is shown below.

1. When a USB-related event has completed, the USB driver calls the callback function (usb_apl_callback). In the callback function (usb_apl_callback), the application task (APL) is notified of the USB completion event using the real-time OS functionality.
2. In APL, information regarding the USB completion event was notified from the callback function is retrieved using the real-time OS functionality.
3. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STATUS_CONFIGURED, APL sends the class request (SET_PROTOCOL) to the HID device.
4. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STATUS_REQUEST_COMPLETE, APL performs a data reception request to receive data transmitted from the HID device by calling the R_USB_Read function.
5. The avove processing is repeated.

An overview of the processing performed by the APL is shown below:

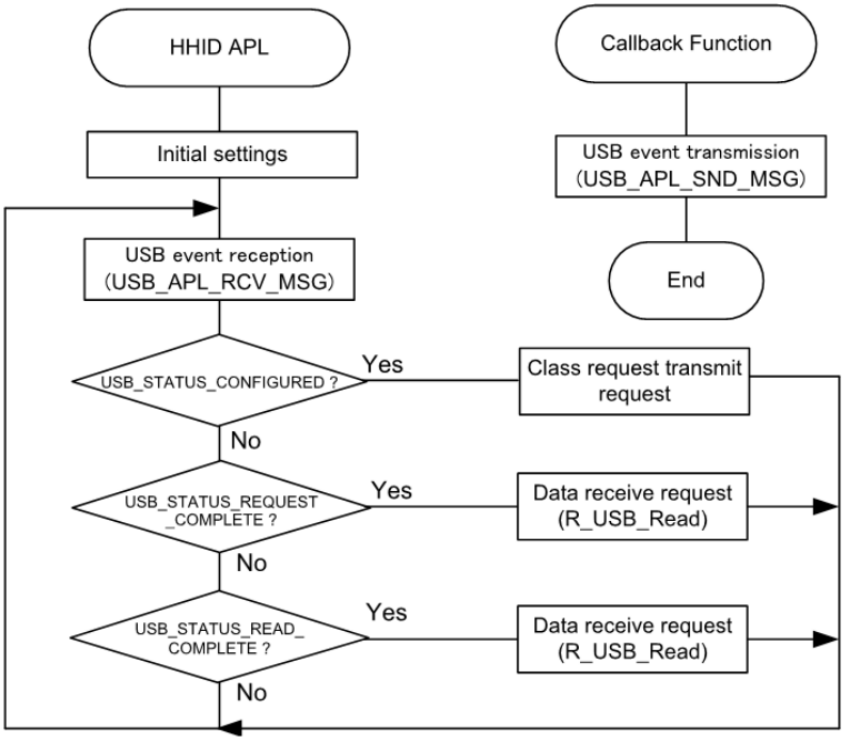



Figure 131: Main Loop (Normal mode)

### Application Processing (for Non-OS)

The application comprises two parts: initial settings and main loop. An overview of the processing in these two parts is provided below.

### Initial setting

Initial settings consist of MCU pin settings, USB driver settings, and initial settings to the USB controller.

### usb_apl_task

The main loop performs processing to receive data from the HID device as part of the main routine. An overview of the processing of the main loop is presented below.

1. When the R_USB_GetEvent function is called after an HID device attaches to the USB host and enumeration completes, USB_STATUS_CONFIGURED is set as the return value. When the APL confirms USB_STATUS_CONFIGURED, it calls the R_USB_Write function to request transmission of data to the HID device.
2. When the R_USB_GetEvent function is called after sending of class request SET_PROTOCOL to the HID device has completed, USB_STATUS_REQUEST_COMPLETE is set as the return value. When the APL confirms USB_STATUS_REQUEST_COMPLETE, it calls the R_USB_Read function to make a data receive request for data sent by the HID device.
3. When the R_USB_GetEvent function is called after reception of data from the HID device has completed, USB_STATUS_READ_COMPLETE is set as the return value. When the APL confirms USB_STATUS_READ_COMPLETE, it calls the R_USB_Read function to make a data receive request for data sent by the HID device.
4. The processing in step 3, above, is repeated.

An overview of the processing performed by the APL is shown below:

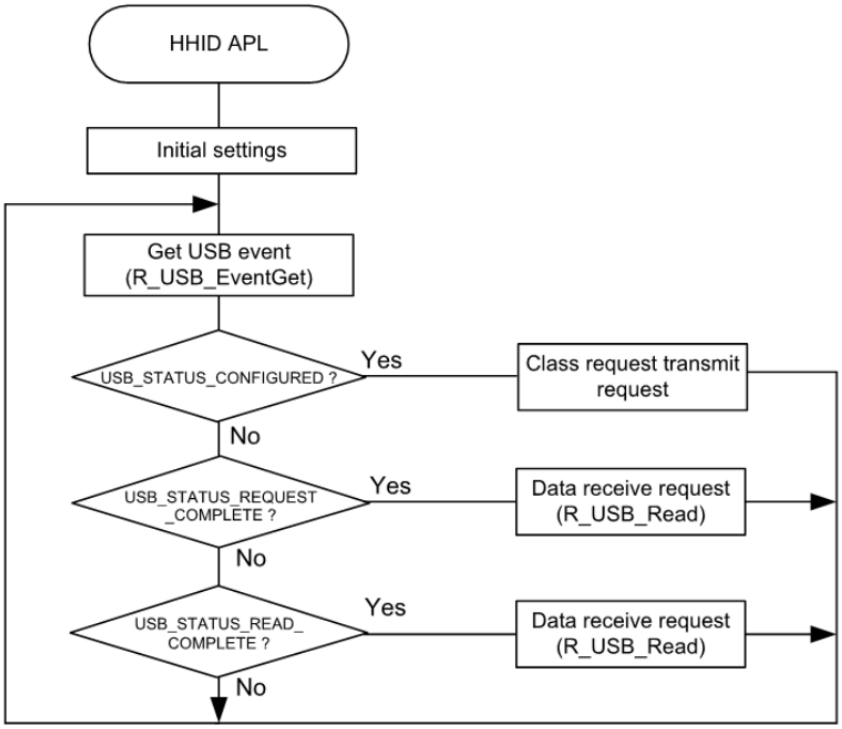



Figure 132: Main Loop (Normal mode)

```
/******************************************************************************
 * Macro definitions
 ******************************************************************************/
#define SET_PROTOCOL (USB_HID_SET_PROTOCOL | USB_HOST_TO_DEV | USB_CLASS |
USB_INTERFACE)
#define BOOT_PROTOCOL (0)
#define USB_FS_DEVICE_ADDRESS_1 (1)
/******************************************************************************
 * Function Name : r_usb_hhid_example
 * Description : Host HID application main process
 * Arguments : none
 * Return value : none
 ******************************************************************************/
static void r_usb_hhid_example (void)
{
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
```

```
#endif  /* (BSP_CFG_RTOS == 2) */
 usb_status_t    event;
    usb_event_info_t event_info;
    uint16_t         offset = 0;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
#if (BSP_CFG_RTOS == 2)
       USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **) &p_mess);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
       g_usb_on_usb.eventGet(&event_info, &event); /* Get event code */
#endif  /* (BSP_CFG_RTOS == 2) */
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
      {
                g_hhid_on_usb.typeGet(&g_basic0_ctrl, &g_hid_type,
USB_FS_DEVICE_ADDRESS_1);
                g_hhid_on_usb.maxPacketSizeGet(&g_basic0_ctrl, &g_mxps, USB_HID_IN,
USB_FS_DEVICE_ADDRESS_1);
 /* Send the HID request(SetProtocol) to HID device */
                set_protocol(&g_basic0_ctrl, BOOT_PROTOCOL, USB_FS_DEVICE_ADDRESS_1);
 break;
            }
 case USB_STATUS_READ_COMPLETE:
      {
                offset = hid_memcpy(g_store_buf, g_buf, offset, g_mxps);
                g_usb_on_usb.read(&g_basic0_ctrl, g_buf, (uint32_t) g_mxps,
USB_FS_DEVICE_ADDRESS_1);
 break;
            }
 case USB_STATUS_REQUEST_COMPLETE:
```

```
      {
 if (USB_HID_SET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
      {
                  g_usb_on_usb.read(&g_basic0_ctrl, g_buf, (uint32_t) g_mxps,
USB_FS_DEVICE_ADDRESS_1);
              }
 break;
            }
 default:
      {
 break;
            }
       }
    }
} /* End of function usb_main */
/******************************************************************************
 * Function Name : set_protocol
 * Description : Sending SetProtocol request to HID device
 * Arguments : usb_ctrl_t *p_ctrl : Pointer to usb_instance_ctrl_t structure.
 * : uint8_t ptorocol: Protocol Type
 * : uint8_t device_address: Device address that sends this request
 * Return value : none
 ******************************************************************************/
static void set_protocol (usb_instance_ctrl_t * p_ctrl, uint8_t protocol, uint8_t
device_address)
{
    usb_setup_t setup;
    setup.request_type   =
SET_PROTOCOL;                                                          /*
bRequestCode:SET_PROTOCOL, bmRequestType */
    setup.request_value =
protocol;                                                              /* wValue:
Protocol Type */
    setup.request_index =
```

```
0x0000;                                                                 /*
wIndex:Interface */
    setup.request_length =
0x0000;                                                                 /* wLength:Zero
*/
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_setup_data,
device_address); /* Request Control transfer */
} /* End of function set_protocol */
/******************************************************************************
 * Function Name : hid_memcpy
 * Description : Copy received hhid data to the application buffer
 * Arguments : uint8_t *p_dest : Pointer to application buffer
 * : uint8_t *p_src : Pointer to received buffer
 * : uint16_t offset : Application buffer offset
 * : uint16_t size : Size of receiced hhid data
 * Return value : uint16_t offset + i: Offset
 ******************************************************************************/
static uint16_t hid_memcpy (uint8_t * p_dest, uint8_t * p_src, uint16_t offset,
uint16_t size)
{
    uint16_t i;
 for (i = 0; i < size; i++)
    {
 if ((offset + i) == BUFSIZE)
      {
            offset = 0;
       }
       *(p_dest + offset + i) = *(p_src + i);
    }
 return (uint16_t) (offset + i);
} /* End of function hid_memcpy */
```

## Function Documentation

### ◆ R_USB_HHID_TypeGet()

fsp_err_t R_USB_HHID_TypeGet ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_type*, uint8_t *device_address* )

Get HID protocol.(USB Mouse/USB Keyboard/Other Type.)

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter Null pointer error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_HHID_MaxPacketSizeGet()

fsp_err_t R_USB_HHID_MaxPacketSizeGet ( usb_ctrl_t *const *p_api_ctrl*, uint16_t * *p_size*, uint8_t *direction*, uint8_t *device_address* )

Obtains max packet size for the connected HID device. The max packet size is set to the area. Set the direction (USB_HID_IN/USB_HID_OUT).

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter Null pointer error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

## 4.2.49 USB Host Mass Storage Class Driver (r_usb_hmsc)

Modules

**Functions**

| | |
|---|---|
| FSP_HEADER fsp_err_t | R_USB_HMSC_StorageCommand (usb_ctrl_t *const p_api_ctrl, uint8_t *buf, uint8_t command, uint8_t destination)<br>Processing for MassStorage(ATAPI) command. More... |
| fsp_err_t | R_USB_HMSC_DriveNumberGet (usb_ctrl_t *const p_api_ctrl, uint8_t |

| | |
|---|---|
| | *p_drive, uint8_t destination)<br>Get number of Storage drive. More... |
| fsp_err_t | R_USB_HMSC_SemaphoreGet (void)<br>Get a semaphore. (RTOS only) More... |
| fsp_err_t | R_USB_HMSC_SemaphoreRelease (void)<br>Release a semaphore. (RTOS only) More... |
| fsp_err_t | R_USB_HMSC_StorageReadSector (uint16_t drive_number, uint8_t *const buff, uint32_t sector_number, uint16_t sector_count)<br>Read sector information. More... |
| fsp_err_t | R_USB_HMSC_StorageWriteSector (uint16_t drive_number, uint8_t const *const buff, uint32_t sector_number, uint16_t sector_count)<br>Write sector information. More... |

### Detailed Description

The USB module (r_usb_hmsc) provides an API to perform hardware control of USB communications. It implements the USB HMSC Interface.

This module is USB Basic Host and Peripheral. It works in combination with Driver (r_usb_basic module).

# Overview

The r_usb_hmsc module, when used in combination with the r_usb_basic module, operates as a USB host mass storage class driver (HMSC). HMSC is built on the USB mass storage class Bulk-Only Transport (BOT) protocol. It is possible to communicate with BOT-compatible USB storage devices by combining it with the file system and storage device driver. This module should be used in combination with the FreeRTOS+FAT File System.

### Features

The r_usb_hmsc module has the following key features:

- Checking of connected USB storage devices to determine whether or not operation is supported
- Storage command communication using the BOT protocol
- Support for SFF-8070i (ATAPI) USB mass storage subclass
- Sharing of a single pipe for IN/OUT directions or multiple devices
- Maximum 4 USB storage devices can be connected

### Class Driver Overview

### Class Requests

The class requests supported by this driver are shown below.

| Request | Description |
|---|---|
| GetMaxLun | Gets the maximum number of units that are supported. |
| MassStorageReset | Cancels a protocol error. |

### Storage Commands

This driver supports the following storage commands:

- TEST_UNIT_READY
- This module is USB Peripheral Communication Device Class Driver
- MODE_SELECT10
- MODE_SENSE10
- PREVENT_ALLOW
- READ_FORMAT_CAPACITY
- READ10
- WRITE10

## Configuration

Refer to USB (r_usb_basic) basic module.

### Clock Configuration

Refer to USB (r_usb_basic) basic module.

### Pin Configuration

Refer to USB (r_usb_basic) basic module.

## Usage Notes

- Due to the wide variety of USB mass storage device implementations, this driver is not guaranteed to work with all devices. When implementing the driver it is important to verify correct operation with the mass storage devices that the end user is expected to use.
- This module must be incorporated into a project using r_usb_basic. Once incorporated into a project, use the API to perform USB hardware control.

### Limitations

1. Some MSC devices may be unable to connect because they are not recognized as storage devices.
2. MSC devices that return values of 1 or higher in response to the GetMaxLun command (mass storage class command) are not supported.
3. Maximum 4 USB storage devices can be connected.
4. Only USB storage devices with a sector size of 512 bytes can be connected.

5. A device that does not respond to the READ_CAPACITY command operates as a device with a sector size of 512 bytes.
6. The continuous transfer mode cannot be used when using DMA.

# Examples

### USB HMSC Example

### Example Operating Environment

The following shows an example operating environment for the HMSC.

Refer to the associated instruction manuals for details on setting up the evaluation board and using the emulator, etc.

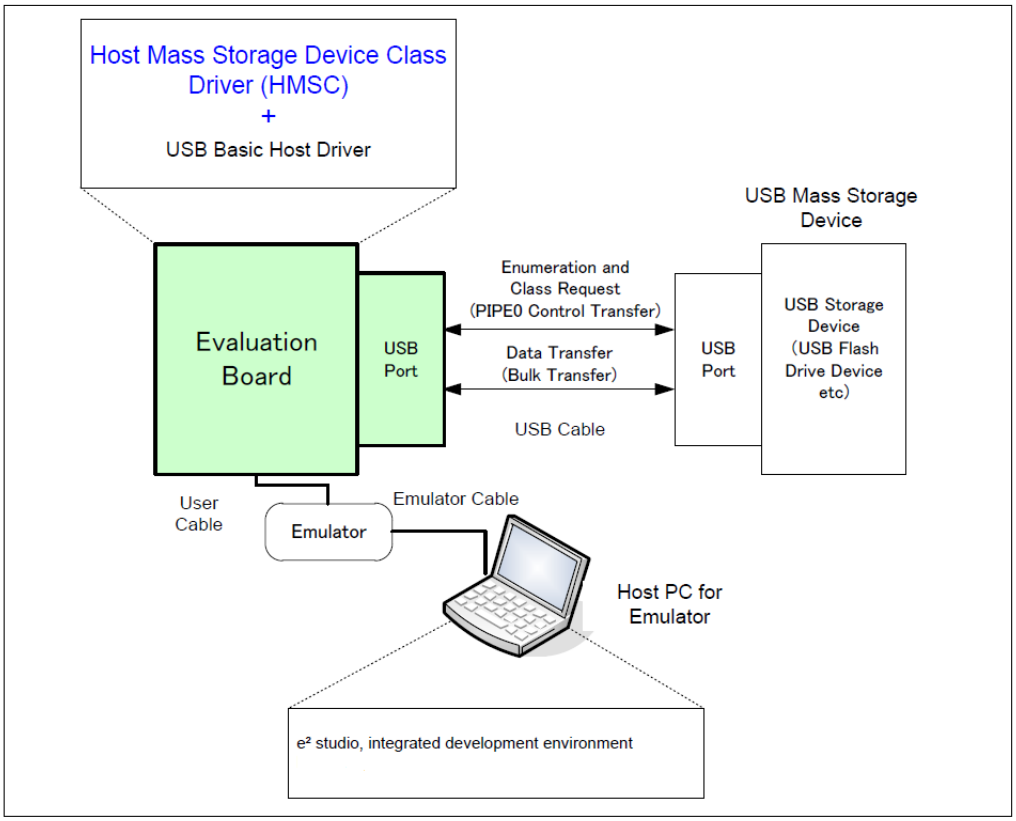



Figure 133: Example Operating Environment

### Application Specifications

The main functions of the application are as follows:

1. Performs enumeration and drive recognition processing on MSC devices.
2. After the above processsing finisihes, the application writes the file hmscdemo.txt to the MSC device once.
3. After writing the above file, the APL repeatedly reads the file hmscdemo.txt. It continues to read the file repeatedly until the switch is pressed again.

### Application Processing (for RTOS)

This application has two tasks. An overview of the processing in these two tasks is provided below.

#### usb_apl_task

1. After start up, MCU pin setting, USB controller initialization, and application program initialization are performed.
2. The MSC device is attached to the kit. When enumeration and drive recognition processing have completed, the USB driver calls the callback function (usb_apl_callback). In the callback function (usb_apl_callback), the application task is notified of the USB completion event using the FreeRTOS functionality.
3. In the application task, information regarding the USB completion event about which notification was received from the callback function is retrieved using the real-time OS functionality.
4. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STS_CONFIGURED then, based on the USB completion event, the MSC device is mounted and the file is written to the MSC device.
5. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STS_DETACH, the application initializes the variables for state management.

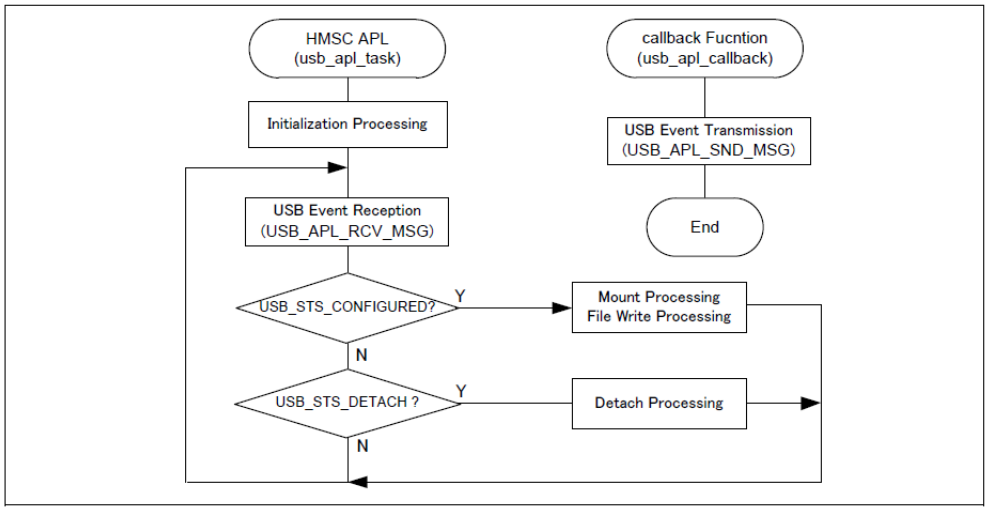



Figure 134: usb_apl_task

#### file_read_task

Of the application tasks usb_apl_task and file_read_task, file_read_task is processed while usb_apl_task is in the wait state. This task performs file read processing on the file that was written to the MSC device (hmscdemo.txt).

### Example Code

*Note*

*For example code refer to the USB HMSC Block Media example.*

### Function Documentation

### ◆ R_USB_HMSC_StorageCommand()

fsp_err_t R_USB_HMSC_StorageCommand ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *buf*, uint8_t *command*, uint8_t *destination* )

Processing for MassStorage(ATAPI) command.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter Null pointer error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_HMSC_DriveNumberGet()

fsp_err_t R_USB_HMSC_DriveNumberGet ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_drive*, uint8_t *destination* )

Get number of Storage drive.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter Null pointer error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_HMSC_SemaphoreGet()

fsp_err_t R_USB_HMSC_SemaphoreGet ( void )

Get a semaphore. (RTOS only)

If this function is called in the OS less execution environment, a failure is returned.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |

### ◆ R_USB_HMSC_SemaphoreRelease()

fsp_err_t R_USB_HMSC_SemaphoreRelease ( void )

Release a semaphore. (RTOS only)

If this function is called in the OS less execution environment, a failure is returned.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |

### ◆ R_USB_HMSC_StorageReadSector()

fsp_err_t R_USB_HMSC_StorageReadSector ( uint16_t *drive_number*, uint8_t *const *buff*, uint32_t *sector_number*, uint16_t *sector_count* )

Read sector information.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter Null pointer error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*The address specified in the argument buff must be 4-byte aligned.*

### ◆ R_USB_HMSC_StorageWriteSector()

fsp_err_t R_USB_HMSC_StorageWriteSector ( uint16_t *drive_number*, uint8_t const *const *buff*, uint32_t *sector_number*, uint16_t *sector_count* )

Write sector information.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter Null pointer error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*The address specified in the argument buff must be 4-byte aligned.*

## 4.2.50 USB Peripheral Communication Device Class (r_usb_pcdc)

Modules

This module is USB Peripheral Communication Device Class Driver (PCDC). It implements the USB PCDC Interface. This module works in combination with (r_usb_basic module).

### Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

### Detailed Description

# Overview

The r_usb_pcdc module combines with the r_usb_basic module to provide USB Peripheral It operates as a communication device class driver (hereinafter referred to as PCDC). PCDC conforms to Abstract Control Model of USB communication device class specification (hereinafter referred to as CDC) and can communicate with USB host.

### Features

The r_usb_pcdc module has the following key features:

- Data transfer to and from a USB host.
- Response to CDC class requests.
- Provision of communication device class notification transmit service.

### Basic Functions

CDC conforms to the communication device class specification Abstract Control Model subclass.

### Abstract Control Model Overview

The Abstract Control Model subclass of CDC is a technology that bridges the gap between USB devices and earlier modems (employing RS-232C connections), enabling use of application programs designed for older modems. The class requests and class notifications supported are listed below.

### Class Requests (Host to Peripheral)

This driver notifies to the application program when receiving the following class request.

| Request | Code | Description |
|---|---|---|
| SetLineCoding | 0x20 | Makes communication line settings (communication speed, data length,parity bit, and stop bit length). |
| GetLineCoding | 0x21 | Acquires the communication line setting state. |
| SetControlLineState | 0x22 | Makes communication line control signal (RTS,DTR) settings. |

For details concerning the Abstract Control Model requests, refer to Table 11, [Requests - Abstract Control Model] in [USB Communications Class Subclass Specification for PSTN Devices], Revision 1.2.

### Data Format of Class Requests

The data format of the class requests supported by the class driver software is described below.

### SetLineCoding

This is the class request the host transmits to perform the UART line setting. The SetLineCoding data format is shown below.

SetLineCoding Format

| bmRequestTyp e t | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0x21 | SET_LINE_CODI NG(0x20) | 0x00 | 0x0 | 0x07 | Line Coding Structure |

Line Coding Structure

| Offset | Field | Size | Value | Description |
|---|---|---|---|---|
| 0 | DwDTERate | 4 | Number | Data terminal speed (bps) |
| 4 | BcharFormat | 1 | Number | Stop bits: |

| | | | | |
|---|---|---|---|---|
| | | | | 0 - 1 stop bit<br>1 - 1.5 stop bits<br>1 - 1.5 stop bits<br>2 - 2 stop bits |
| 5 | BparityType | 1 | Number | Parity:<br>0 - None<br>1 - Odd<br>2 - Even |
| 6 | BdataBits | 1 | | Data bits (5, 6, 7, 8) |

**GetLineCoding**

This is the class request the host transmits to request the UART line state. The GetLineCoding data format is shown below.

GetLineCoding Format

| bmRequestTyp e t | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0xA1 | GET_LINE_COD ING(0x21) | 0x00 | 0x0 | 0x07 | Line Coding Structure |

**SetControlLineState**

This is a class request that the host sends to set up the signal for flow controls of UART. This software does not support RTS/DTR control. The SET_CONTROL_LINE_STATE data format is shown below.

SET_CONTROL_LINE_STATE Format

| bmRequestTyp e t | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0x21 | SET_CONTROL _LINE_STATE(0 x22) | Control Signal Bitmap | 0x0 | 0x00 | None |

Control Signal Bitmap

| Bit Position | Description |
|---|---|
| D15 to D2 | Reserved (reset to 0) |
| D1 | DCE transmit function control:<br>0 - RTS Off<br>1 - RTS On |
| D0 | Notification of DTE ready state:<br>0 - DTR Off<br>1 - DTR On |

**Class Notifications (Peripheral to Host)**

The table below shows the class notification support/non-support of this software.

| Notification | Code | Description | Supported |
|---|---|---|---|
| NETWORK_CONNECTION | 0x00 | Notification of network connection state | No |
| RESPONSE_AVAILABLE | 0x01 | Response to GET_ENCAPSLATED_RESPONSE | No |
| SERIAL_STATE | 0x20 | Notification of serial line state | Yes |

**1.Serial State**

The host is notified of the serial state when a change in the UART port state is detected.

This software supports the detection of overrun, parity and framing errors. A state notification is performed when a change from normal state to error is detected. However, notification is not continually transmitted when an error is continually detected.

SerialState Format

| bmRequestTypet | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0xA1 | SERIAL_STATE(0x20) | 0x00 | 0x0 | 0x02 | UART State bitmap |

UART state bitmap format

| Bits | Fieeld | Description | Supported |
|---|---|---|---|
| D15 to D7 | | Reserved | - |
| D6 | b_over_run | Overrun error detected | Yes |
| D5 | b_parity | Parity error detected | Yes |
| D4 | b_framing | Framing error detected | Yes |
| D3 | b_ring_signal | INCOMING signal (ring signal) detected | No |
| D2 | b_break | Break signal detected | No |
| D1 | btx_arrier | Data Set Ready: Line connected and ready for communication | No |
| D0 | brx_carrier | Data Carrier Detect: Carrier detected on line | No |

**PC Virtual COM-port Usage**

The CDC device can be used as a virtual COM port when operating in Windows OS.

Use a PC running Windows OS, and connect an board. After USB enumeration, the CDC class requests GetLineCoding and SetControlLineState are executed by the target, and the CDC device is registered in Windows Device Manager as a virtual COM device.

Registering the CDC device as a virtual COM-port in Windows Device Manager enables data communication with the CDC device via a terminal app such as HyperTerminal which comes standard with Windows OS. When changing settings of the serial port in the Windows terminal application, the UART setting is propagated to the firmware via the class request SetLineCoding.

Data input (or file transmission) from the terminal app window is transmitted to the board using endpoint 2 (EP2); data from the board side is transmitted to the PC using EP1.

When the last packet of data received is the maximum packet size, and the terminal determines that there is continuous data, the received data may not be displayed in the terminal. If the received data is smaller than the maximum packet size, the data received up to that point is displayed in the terminal.

The received data is outputted on the terminal when the data less than Maximum packet size is received.

# Configuration

### Build Time Configurations for r_usb_pcdc

The following build time configurations are defined in fsp_cfg/r_usb_pcdc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Pipe to use for Bulk IN transfer. | • USB PIPE1<br>• USB PIPE2<br>• USB PIPE3<br>• USB PIPE4<br>• USB PIPE5 | USB PIPE1 | Please choose between 1 and 5. |
| Pipe to use for Bulk OUT transfer. | • USB PIPE1<br>• USB PIPE2<br>• USB PIPE3<br>• USB PIPE4<br>• USB PIPE5 | USB PIPE2 | Please choose between 1 and 5. |
| Pipe to use for Interrupt IN transfer. | • USB PIPE6<br>• USB PIPE7<br>• USB PIPE8<br>• USB PIPE9 | USB PIPE6 | Please choose between 6 and 9. |

### Configurations for Middleware > USB > USB PCDC driver on r_usb_pcdc

This module can be added to the Stacks tab via New Stack > Middleware > USB > USB PCDC driver on r_usb_pcdc:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid | g_pcdc0 | Module name. |

C symbol

Setting as r_usb_pcdc module is not necessary. Refer to r_usb_basic module.

### Clock Configuration

Refer to r_usb_basic module.

### Pin Configuration

Refer to r_usb_basic module.

## Usage Notes

### Limitations

This module needs to be incorporated into a project using r_usb_basic. Once incorporated into a project, use the API to perform USB hardware control.

## Examples

### USB PCDC Example

### Example Operating Environment

The following shows an example of the operating environment for the PCDC echo mode.

Refer to the associated instruction manuals for details on setting up the evaluation board and using the emulator, etc.

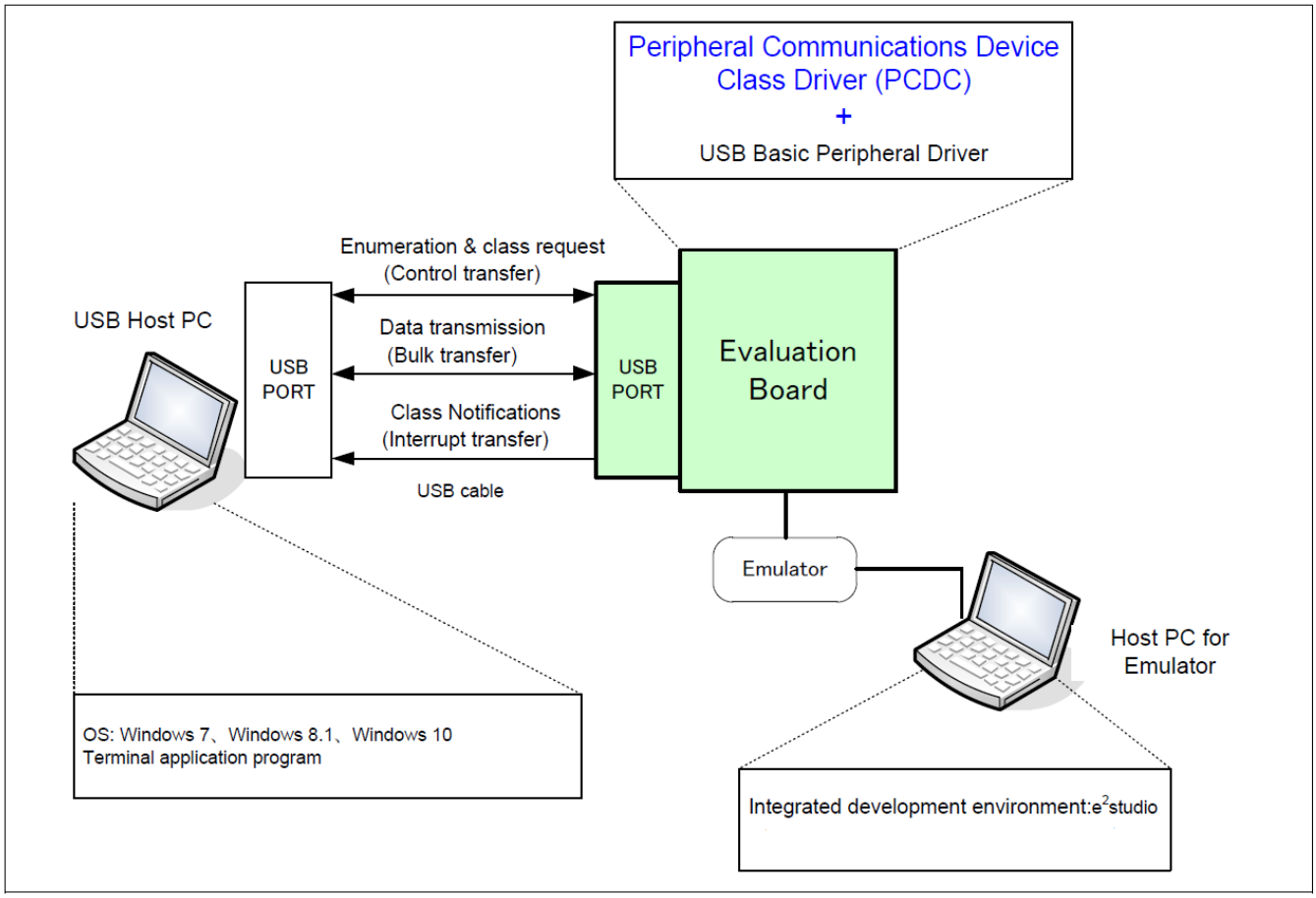



Figure 135: Example Operating Environment

### Application Specifications

The main functions of the APL are as follows:

1. Echo mode (Loopback mode)
   Transmits data received from the USB host back to the USB host.

2. Low-power functionality
   This functionality transitions the MCU to low-power mode according to the status of the USB.
   a) The APL transitions the MCU to sleep mode when the USB is suspended.
   b) When the USB is detached (disconnected), the APL transitions the MCU to software standby mode.

### Application Processing (for RTOS)

The APL comprises two parts: initial settings and main loop. An overview of the processing in these two parts is provided below.

### Initial Setting

Initial settings consist of MCU pin settings, USB driver settings, and initial settings to the USB controller.

### Main Loop (Echo mode)

In Echo mode, loop-back processing in which data sent by the USB host is received and then transmitted unmodified back to the USB host takes place as part of the main routine. An overview of the processing performed by the loop is shown below.

1. When a USB-related event has completed, the USB driver calls the callback function (usb_apl_callback). In the callback function (usb_apl_callback), the application task (APL) is notified of the USB completion event using the real-time OS functionality.
2. In APL, information regarding the USB completion event was notified from the callback function is retrieved using the real-time OS functionality.
3. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STS_CONFIGURED, APL performs a data reception request to receive data transmitted from the USB Host by calling the R_USB_Read function.
4. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STS_REQUEST, APL performs processing in response to the received request.
5. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STS_READ_COMPLETE, APL performs a data transmission request to send USB Host the reception data by calling the R_USB_Write function.
6. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STS_WRITE_COMPLETE, APL performs a data reception request to receive the data sent from USB Host by calling the R_USB_Read function.
7. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STS_SUSPEND or USB_STS_DETACH, APL performs processing to transition the CDC device to low-power mode.

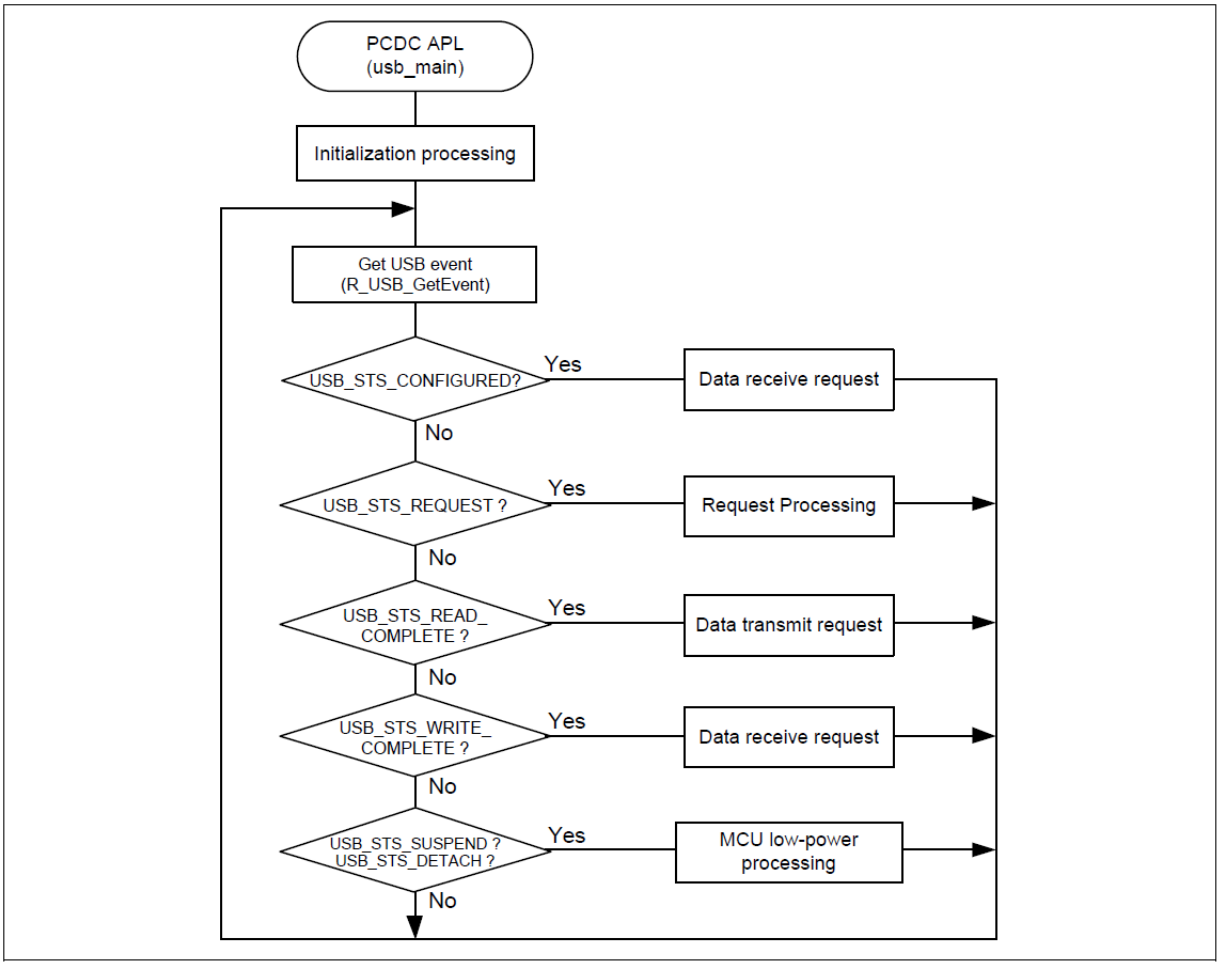



Figure 136: Main Loop processing (Echo mode)

Below is an example of minimal use of the USB PCDC module in an application.

```
void usb_basic_example (void)
{
    usb_event_info_t event_info;
 usb_status_t    event;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 /* Loop back between PC(TerminalSoft) and USB MCU */
 while (1)
    {
       g_usb_on_usb.eventGet(&event_info, &event);
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
```

```
 case USB_STATUS_WRITE_COMPLETE:
                g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PCDC);
 break;
 case USB_STATUS_READ_COMPLETE:
                g_usb_on_usb.write(&g_basic0_ctrl, g_buf, event_info.data_size,
USB_CLASS_PCDC);
 break;
 case USB_STATUS_REQUEST:   /* Receive Class Request */
 if (USB_PCDC_SET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlDataGet(&g_basic0_ctrl, (uint8_t *)
&g_line_coding, LINE_CODING_LENGTH);
                }
 else if (USB_PCDC_GET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, (uint8_t *)
&g_line_coding, LINE_CODING_LENGTH);
                }
 else
      {
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
                }
 break;
 case USB_STATUS_SUSPEND:
 case USB_STATUS_DETACH:
 break;
 default:
 break;
       }
    }
} /* End of function usb_main() */
```

# Descriptor

A template for PCDC descriptors can be found in ra/fsp/src/r_usb_pcdc/r_usb_pcdc_descriptor.c.template. Also, please be sure to use your vendor ID.

## 4.2.51 USB Peripheral Human Interface Device Class (r_usb_phid)

Modules

This module is USB Peripheral Human Interface Device Class Driver (PHID). It implements the USB PHID Interface.
This module works in combination with (r_usb_basic module).

### Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

### Detailed Description

# Overview

The r_usb_phid module combines with the r_usb_basic module to provide USB Peripheral It operates as a human interface device class driver (hereinafter referred to as PHID).
The PHID conforms to the USB Human Interface Device class specifications (referred to here as HID) and implements communication with a HID host.

### Features

This module supports the following functions.

- Data transfer to and from a USB host.
- Response to HID class requests.
- Response to function references from the HID host.

Note: This driver is not guaranteed to provide USB communication operation.
The customer should verify operation when utilizing it in a system and confirm the ability to connect to a variety of different types of devices.

### Basic Functions

### Class Requests (Host to Peripheral)

This driver notifies to the application program when receiving the following class request.

| Request | Code | Description |
|---|---|---|
| Get_Report | 0x01 | Receives a report from the HID host |

| | | |
|---|---|---|
| Set_Report | 0x09 | Sends a report to the HID host |
| Get_Idle | 0x02 | Receives a duration (time) from the HID host |
| Set_Idle | 0x0A | Sends a duration (time) to the HID host |
| Get_Protocol | 0x03 | Reads a protocol from the HID host |
| Set_Protocol | 0x0B | Sends a protocol to the HID host |
| Get_Descriptor<br>Descriptor Type : Class<br>Class Descriptor Type : Report | 0x06 | Transmits a report descriptor |
| Get_Descriptor<br>Descriptor Type : Class<br>Class Descriptor Type : HID | 0x06 | Transmits an HID descriptor |

**Data Format of Class Requests**

The data format of the class requests supported by the class driver software is described below.

**1.GetReport**

GetReport Format

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0xA1 | GET_REPORT(0x01) | ReportType & ReportID | Interface | ReportLength | Report |

**2.SetReport**

SetRepor Format

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0x21 | SET_REPORT(0x09) | ReportType & ReportID | Interface | ReportLength | Report |

**3.GetIdle**

GetIdle Format

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0xA1 | GET_IDLE(0x02) | 0(Zero) & ReportID | Interface | 1(one) | Idle rate |

**4.SetIdle**

SetIdle Format

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0x21 | SET_IDLE(0x0A) | Duration & ReportID | Interface | 0(zero) | Idle rate |

**5.GetProtocol**

GetProtocol Format

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0xA1 | GET_PROTOCOL(0x03) | 0(zero) | Interface | 0(zero) | 0 (Boot Protocol) / 1 (Report Protocol) |

**6.SetProtocol**

SetProtocol Format

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0x21 | SET_PROTOCOL(0x0B) | 0 (Boot Protocol) / 1 (Report Protocol) | Interface | 0(zero) | Not applicable |

# API

Refer to UsageNote of r_usb_basic for the API used in the application program.

# Configuration

**Build Time Configurations for r_usb_phid**

The following build time configurations are defined in fsp_cfg/r_usb_phid_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Pipe to use for Interrupt IN transfer. | • USB PIPE6<br>• USB PIPE7<br>• USB PIPE8<br>• USB PIPE9 | USB PIPE6 | Please select from PIPE6 to PIPE9. |
| Pipe to use for | • USB PIPE6 | USB PIPE7 | Please select from |

| | | |
|---|---|---|
| Interrupt OUT transfer. | • USB PIPE7<br>• USB PIPE8<br>• USB PIPE9 | PIPE6 to PIPE9. |

**Configurations for Middleware > USB > USB PHID driver on r_usb_phid**

This module can be added to the Stacks tab via New Stack > Middleware > USB > USB PHID driver on r_usb_phid:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_phid0 | Module name. |

Setting as r_usb_phid module is not necessary.
Refer to r_usb_basic module.

**Clock Configuration**

Refer to r_usb_basic module.

**Pin Configuration**

Refer to r_usb_basic module.

# Usage Notes

**Limitations**

- This module needs to be incorporated into a project using r_usb_basic. Once incorporated into a project, use the API to perform USB H / W control.
- This driver does not support USB Hi-speed.

# Examples

**USB PHID Example**

**Example Operating Environment**
The following is an example of the PHID operating environment.
Refer to the associated instruction manuals for details on setting up the evaluation board and using the emulator, etc.

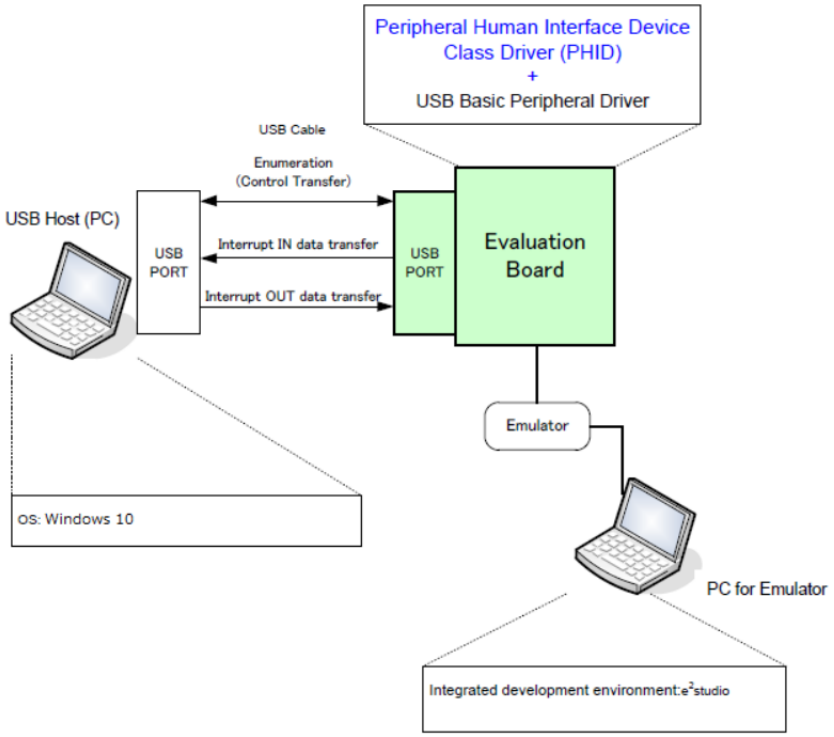



Figure 137: Example Operating Environment

### Application Specifications

The PHID Example code (OS less) does the following:

1. Complete USB module initialization and enumeration.
2. Perform USB_STATUS_CONFIGURED processing with the USB event scheduler and send data to the Host.
3. After data transmission is completed, execute USB_STATUS_WRITE_COMPLETE processing with the USB event scheduler,
   wait for a certain period of time, and then transmit data.
4. Repeat Step 3.

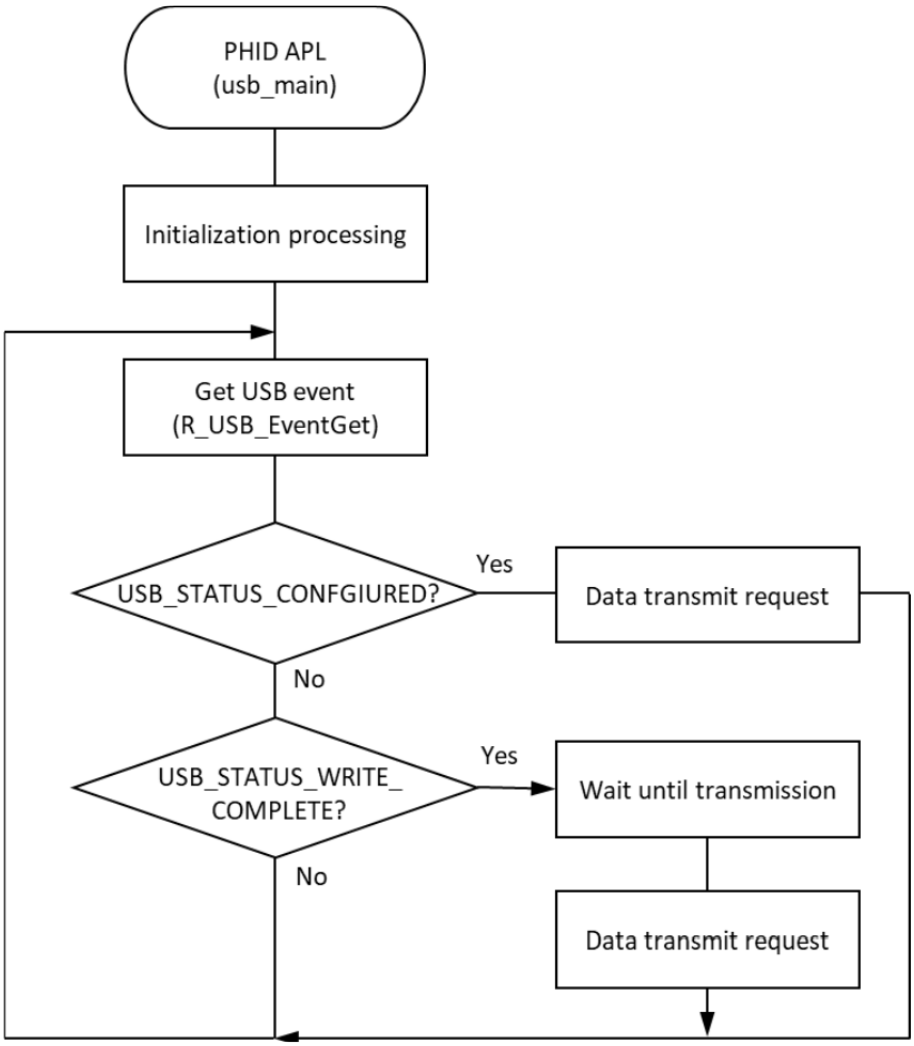



Figure 138: Main Loop processing for OS less

The PHID Example code (FreeRTOS) does the following:

1. When the USB-related event is completed, the USB driver calls the callback function (usb_apl_callback) registered in the usb_cfg_t structure. In the callback function (usb_apl_callback), a USB completion event is notified to the application task (APL) using the real-time OS function.
2. In APL, information regarding the USB completion event was notified from the callback function is retrieved using the real-time OS functionality.
3. If the USB completion event (event member of the usb_event_info_t structure) obtained in step 2 above is USB_STATUS_CONFIGURED,
   APL calls the R_USB_Write function and executes a data transmission request to transmit data to the USB host.
4. If the USB completion event (event member of the usb_event_info_t structure) obtained in step 2 above is USB_STATUS_WRITE_COMPLETE,
   APL waits for a certain period of time and then calls the R_USB_Write function to execute a data transmission request to the USB host.
5. Repeat Step 4.

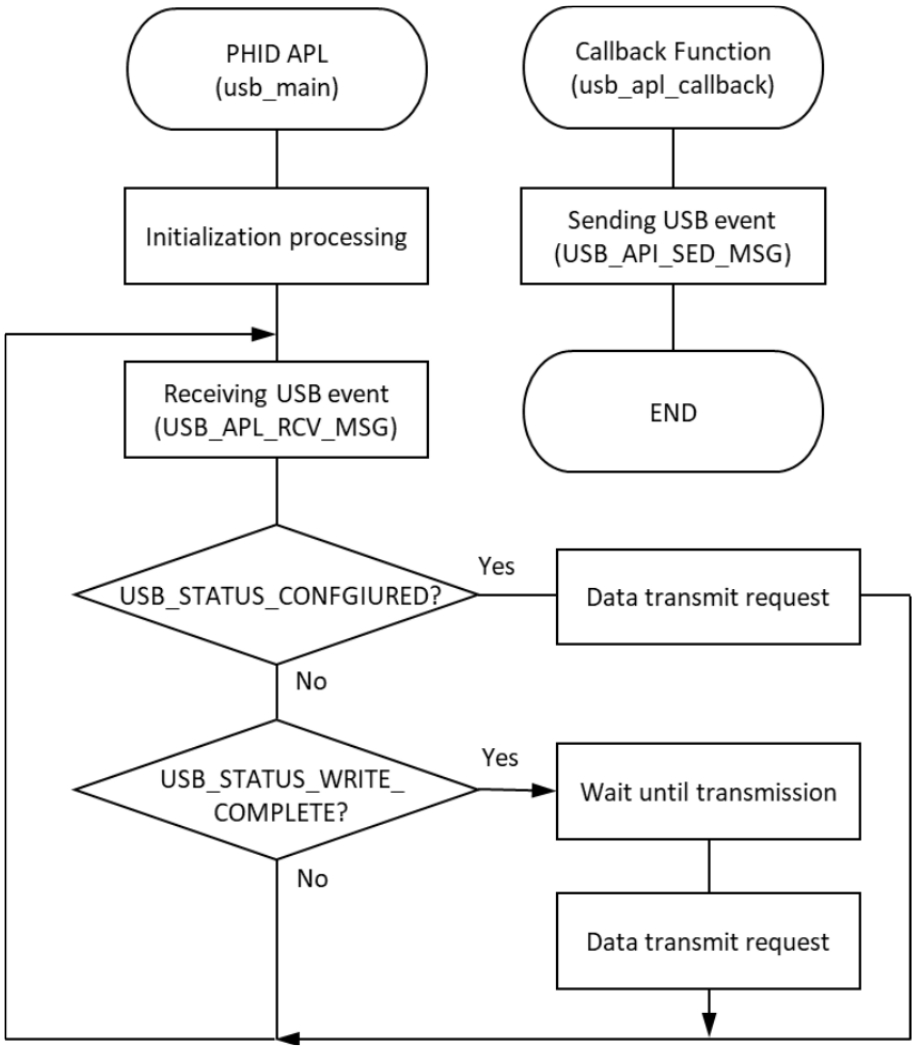



Figure 139: Main Loop processing for RTOS

This is a minimal example for implementing PHID in an application. (OS less)

```
#define USB_RECEIVE_REPORT_DESCRIPTOR (76)
#define USB_RECEIVE_HID_DESCRIPTOR (9)
#define USB_WAIT_1000MS (1000)
#define SW_ACTIVE 0
#define SW R_PFS->PORT[0].PIN[8].PmnPFS_b.PIDR
#define SW_PDR R_PFS->PORT[0].PIN[8].PmnPFS_b.PDR
#define SW_PMR R_PFS->PORT[0].PIN[8].PmnPFS_b.PMR
static uint8_t g_buf[] = {0, 0, 0, 0, 0, 0, 0, 0}; /* HID data */
static const uint8_t g_zero_data[] = {0, 0, 0, 0, 0, 0, 0, 0}; /* zero data */
static uint16_t g_numlock = 0;
static uint8_t g_idle = 0;
```

```
uint8_t             g_remote_wakeup_enable = USB_OFF;
uint8_t             g_status               = NO_WRITING;
void usb_basic_example (void)
{
    usb_event_info_t event_info;
 usb_status_t    event;
    uint8_t       * p_idle_value;
    uint8_t          sw_data;
    usb_info_t       info;
 fsp_err_t       ret_code = FSP_SUCCESS;
    uint8_t          send_data[16] BSP_ALIGN_VARIABLE(4);
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
    set_key_data(g_buf);
 /* Loop back between PC(TerminalSoft) and USB MCU */
 while (1)
    {
       g_usb_on_usb.eventGet(&event_info, &event);
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
 break;
 case USB_STATUS_WRITE_COMPLETE:
 if (DATA_WRITING == g_status)
      {
                  g_status = ZERO_WRITING;
                  g_usb_on_usb.write(&g_basic0_ctrl, (uint8_t *) g_zero_data,
DATA_LEN, USB_CLASS_PHID); /* Sending the zero data (8 bytes) */
               }
 else
      {
                  g_status = DATA_WRITING;
                  usb_cpu_delay_xms(USB_WAIT_1000MS);
                  g_usb_on_usb.write(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PHID
);
```

```
                }
 break;
 case USB_STATUS_REQUEST:
/* Receive Class Request */
 if (USB_SET_REPORT == (event_info.setup.request_type & USB_BREQUEST))
      {
                    g_usb_on_usb.read(&g_basic0_ctrl, (uint8_t *) &g_numlock, 2,
USB_CLASS_PHID); /* Get the NumLock data (NumLock data is not used) */
                }
 else if (USB_GET_DESCRIPTOR == (event_info.setup.request_type & USB_BREQUEST))
      {
 if (USB_GET_REPORT_DESCRIPTOR == event_info.setup.request_value)
      {
                        g_usb_on_usb.periControlDataSet(&g_basic0_ctrl,
                                                        (uint8_t *) g_apl_report,

USB_RECEIVE_REPORT_DESCRIPTOR);
                  }
 else if (USB_GET_HID_DESCRIPTOR == event_info.setup.request_value)
      {
 for (uint8_t i = 0; i < USB_RECEIVE_HID_DESCRIPTOR; i++)
      {
                            send_data[i] = g_apl_configuration[18 + i];
                  }
 /* Configuration Descriptor address set. */
                        g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, send_data,
USB_RECEIVE_HID_DESCRIPTOR);
                  }
 else
      {
                        g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
                  }
                }
```

```
 else if (USB_SET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
      {
 /* Get SetIdle value */
                    p_idle_value = (uint8_t *) &event_info.setup.request_value;
                    g_idle = p_idle_value[1];
                    g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
                }
 else if (USB_GET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
      {
                    g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, &g_idle, 1);
                }
 else if (USB_SET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
      {
                    g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
                }
 else if (USB_GET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
      {
                    g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
                }
 else
      {
                    g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
                }
 break;
 case USB_STATUS_REQUEST_COMPLETE: /* Complete Class Request */
 if (USB_SET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
      {
                    p_idle_value = (uint8_t *) &event_info.setup.request_value;
                    g_idle = p_idle_value[1];
                }
```

```
 else if (USB_SET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
      {
 /* None */
 /* g_protocol = event_info.setup.value; */
                }
 else
      {
                   g_status = DATA_WRITING;
                   g_usb_on_usb.write(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PHID
);
 /* none */
                }
 break;
 case USB_STATUS_SUSPEND:
 break;
 case USB_STATUS_DETACH:
                g_remote_wakeup_enable = USB_OFF;
 break;
 default:
 break;
       }
       ret_code = g_usb_on_usb.infoGet(&g_basic0_ctrl, &info, 0);
 if (FSP_SUCCESS == ret_code)
      {
            sw_data = USB_RSK_GET_KEY_NO();
 if (USB_STATUS_SUSPEND == info.device_status)
      {
 if (0 != (sw_data & SW_PUSH))
      {
                   g_usb_on_usb.remoteWakeup(&g_basic0_ctrl);
                }
         }
       }
    }
```

```
} /* End of function usb_basic_example() */
```

This is a minimal example for implementing PHID in an application. (FreeRTOS)

```
 #define USB_APL_MBX (0)
void usb_basic_example_rtos (void)
{
    usb_event_info_t * p_mess;
    usb_event_info_t   event_info;
    uint8_t          * p_idle_value;
    uint8_t            sw_data;
    usb_info_t         info;
 fsp_err_t          ret_code = FSP_SUCCESS;
    uint8_t            send_data[16] BSP_ALIGN_VARIABLE(4);
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
    set_key_data(g_buf);
 /* Loop back between PC(TerminalSoft) and USB MCU */
 while (1)
    {
       USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **) &p_mess);
       event_info = *p_mess;
 switch (event_info.event)
      {
 case USB_STATUS_CONFIGURED:
 break;
 case USB_STATUS_WRITE_COMPLETE:
 if (DATA_WRITING == g_status)
      {
                    g_status = ZERO_WRITING;
                    g_usb_on_usb.write(&g_basic0_ctrl, (uint8_t *) g_zero_data,
DATA_LEN, USB_CLASS_PHID); /* Sending the zero data (8 bytes) */
                }
 else
      {
```

```
                    g_status = DATA_WRITING;
                    usb_cpu_delay_xms(USB_WAIT_1000MS);
                    g_usb_on_usb.write(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PHID
);
                }
 break;
 case USB_STATUS_REQUEST:
/* Receive Class Request */
 if (USB_SET_REPORT == (event_info.setup.request_type & USB_BREQUEST))
      {
                    g_usb_on_usb.read(&g_basic0_ctrl, (uint8_t *) &g_numlock, 2,
USB_CLASS_PHID); /* Get the NumLock data (NumLock data is not used) */
                }
 else if (USB_GET_DESCRIPTOR == (event_info.setup.request_type & USB_BREQUEST))
      {
 if (USB_GET_REPORT_DESCRIPTOR == event_info.setup.request_value)
      {
                         g_usb_on_usb.periControlDataSet(&g_basic0_ctrl,
                                                         (uint8_t *) g_apl_report,

USB_RECEIVE_REPORT_DESCRIPTOR);
                   }
 else if (USB_GET_HID_DESCRIPTOR == event_info.setup.request_value)
      {
 for (uint8_t i = 0; i < USB_RECEIVE_HID_DESCRIPTOR; i++)
      {
                            send_data[i] = g_apl_configuration[18 + i];
                   }
 /* Configuration Descriptor address set. */
                         g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, send_data,
USB_RECEIVE_HID_DESCRIPTOR);
                   }
 else
      {
```

```
                        g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
                  }
                }
 else if (USB_SET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
      {
 /* Get SetIdle value */
                   p_idle_value = (uint8_t *) &event_info.setup.request_value;
                   g_idle = p_idle_value[1];
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
                }
 else if (USB_GET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, &g_idle, 1);
                }
 else if (USB_SET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
                }
 else if (USB_GET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
                }
 else
      {
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
                }
 break;
 case USB_STATUS_REQUEST_COMPLETE: /* Complete Class Request */
 if (USB_SET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
```

```
     {
                    p_idle_value = (uint8_t *) &event_info.setup.request_value;
                    g_idle = p_idle_value[1];
                }
 else if (USB_SET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
     {
 /* None */
 /* g_protocol = event_info.setup.value; */
                }
 else
     {
                    g_status = DATA_WRITING;
                    g_usb_on_usb.write(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PHID
);
 /* none */
                }
 break;
 case USB_STATUS_SUSPEND:
 break;
 case USB_STATUS_DETACH:
                g_remote_wakeup_enable = USB_OFF;
 break;
 default:
 break;
      }
      ret_code = g_usb_on_usb.infoGet(&g_basic0_ctrl, &info, NULL);
 if (FSP_SUCCESS == ret_code)
     {
           sw_data = USB_RSK_GET_KEY_NO();
 if (USB_STATUS_SUSPEND == info.device_status)
     {
 if (0 != (sw_data & SW_PUSH))
     {
                    g_usb_on_usb.remoteWakeup(&g_basic0_ctrl);
```

```
                }
            }
        }
    }
} /* End of function usb_basic_example_rtos() */
```

# Descriptor

A template for PHID descriptors can be found in ra/fsp/src/r_usb_phid/r_usb_phid_descriptor.c.template. Also, please be sure to use your vendor ID.

## 4.2.52 USB Peripheral Mass Storage Class (r_usb_pmsc)

Modules

This module is USB Peripheral Mass Storage Class Driver (PMSC). It implements the USB PMSC Interface. This module works in combination with (r_usb_basic module).

### Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

### Detailed Description

# Overview

The r_usb_pmsc module combines with the r_usb_basic module to provide USB Peripheral It operates as a Mass Storage class driver (hereinafter referred to as PMSC). The USB peripheral mass storage class driver (PMSC) comprises a USB mass storage class bulk-only transport (BOT) protocol. When combined with a USB peripheral control driver and media driver, it enables communication with a USB host as a BOT-compatible storage device.

### Features

The r_usb_pmsc module has the following key features:

- Storage command control using the BOT protocol.
- Response to mass storage device class requests from a USB host.

### Class Driver Overview

### Class Requests

The class requests supported by this driver are shown below.

| Request | Code | Description |
|---|---|---|
| Bulk-Only Mass Storage Reset | 0xFF | Resets the connection interface to the mass storage device. |
| Get Max Lun | 0xFE | Reports the logical numbers supported by the device. |

**Storage Commands**

This driver supports the following storage command.

This driver send the STALL or FAIL error (CSW) to USB HOST when receiving other than the following command.

| Command | Code | Description |
|---|---|---|
| TEST_UNIT_READY | 0x00 | Checks the state of the peripheral device. |
| REQUEST_SENSE | 0x03 | Gets the error information of the previous storage command execution result. |
| INQUIRY | 0x12 | Gets the parameter information of the logical unit. |
| READ_FORMAT_CAPACITY | 0x23 | Gets the formattable capacity. |
| READ_CAPACITY | 0x25 | Gets the capacity information of the logical unit. |
| READ10 | 0x28 | Reads data. |
| WRITE10 | 0x1A | Writes data. |
| MODE_SENSE10 | 0x5A | Gets the parameters of the logical unit. |

**Basic Functions**

The functions of PDCD are to:

1. Supporting SFF-8070i (ATAPI)
2. Respond to mass storage class requests from USB host.

**BOT Protocol Overview**

BOT (USB MSC Bulk-Only Transport) is a transfer protocol that, encapsulates command, data, and status (results of commands) using only two endpoints (one bulk in and one bulk out). The ATAPI storage commands and the response status are embedded in a "Command Block Wrapper"(CBW) and a "Command Status Wrapper"(CSW). fllow shows an overview of how the BOT protocol progresses with command and status data flowing between USB host and peripheral.

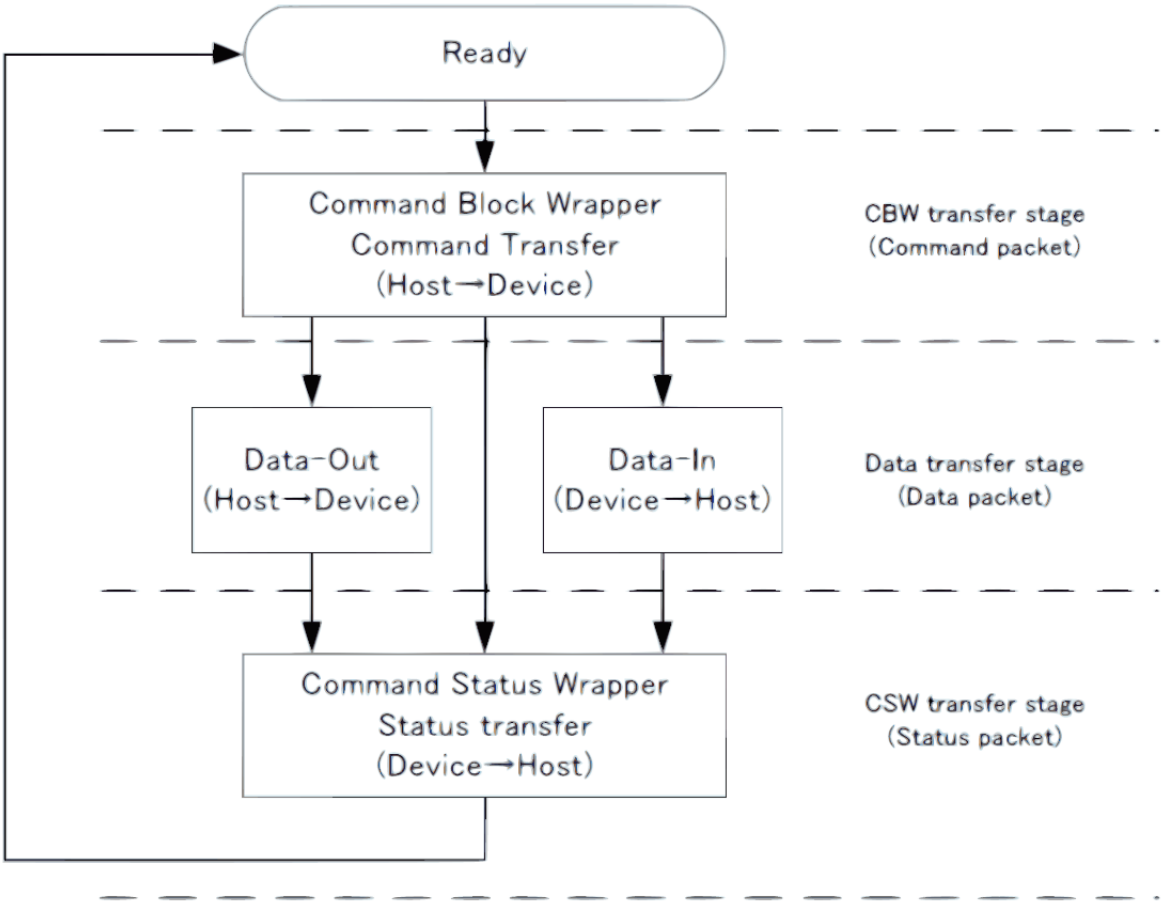



Figure 140: BOT protocol Overview

### Block Media Interface

PMSC implements a block media interface to enable access to media with different specifications. If the block media interface supports multiple media, users can select any media to access.

## Configuration

### Build Time Configurations for r_usb_pmsc

The following build time configurations are defined in fsp_cfg/r_usb_pmsc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Pipe to use for Bulk IN transfer. | • USB PIPE1<br>• USB PIPE2<br>• USB PIPE3<br>• USB PIPE4<br>• USB PIPE5 | USB PIPE1 | Please choose between 1 and 5.<br><br>NOTE: The same pipe number as that used for bulk-out transfer cannot be used. |
| Pipe to use for Bulk | • USB PIPE1 | USB PIPE2 | Please choose between |

| | | | |
|---|---|---|---|
| OUT transfer. | • USB PIPE2<br>• USB PIPE3<br>• USB PIPE4<br>• USB PIPE5 | | 1 and 5.<br><br>NOTE: The same pipe number as that used for bulk-in transfer cannot be used. |
| Vendor Information | Must be entered with 8 bytes of data. | Vendor | Specify the vendor information which is response data of Inquiry command. |
| Product Information | Must be entered as 16 bytes data. | Mass Storage | Specify the product information which is response data of Inquiry command. |
| Product Revision Level. | Must be entered as 4 bytes data. | 1.00 | Specify the product revision level which is response data of Inquiry command. |
| The number of transfer sector. | Please enter a number between 1 and 255. | 8 | Specify the maximum sector size to request to PCD (Peripheral Control Driver) at one data transfer. |

**Configurations for Middleware > USB > USB PMSC driver on r_usb_pmsc**

This module can be added to the Stacks tab via New Stack > Middleware > USB > USB PMSC driver on r_usb_pmsc:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_pmsc0 | Module name. |

Setting as r_usb_pmsc module is not necessary.

Refer to r_usb_basic module.

**Clock Configuration**

Refer to r_usb_basic module.

**Pin Configuration**

Refer to r_usb_basic module.

# Usage Notes

**Limitations**

1. This driver returns the value 0 (zero) to the mass storage command (GetMaxLun) sent from USB Host.
2. The sector size which this driver supports is 512 only.
3. To use a removable storage device, the media must be inserted beforehand.
4. For removing the removable media (PMSC), remove the USB device on the Windows PC and then disconnect the USB cable.
5. Currently, the only media supported by the block media interface is the SD card, so the user-selectable block media interface is the SD card.
6. When using DMA transfer at Hi Speed, it is necessary to set the r_usb_basic configuration so that the continuous transfer mode is not used.
7. The storage area must be formatted.
8. When using SD/MMC Block Media Implementation (rm_block_media_sdmmc), "Card Detection" must be set to "Not Used" in the SD/MMC Host Interface (r_sdhi) settings.

# Examples

### USB PMSC Example

### Example Operating Environment

The following is an image of connecting a PC and PMSC. When the evaluation board is connected to the host PC, it is recognized as a removable disk, and data transfer such as reading/writing files is possible. The media area of the removable disk is the media specified by the user in the block media interface.

The FAT type depends on the size of the media used, and is FAT12, FAT16, or FAT32.

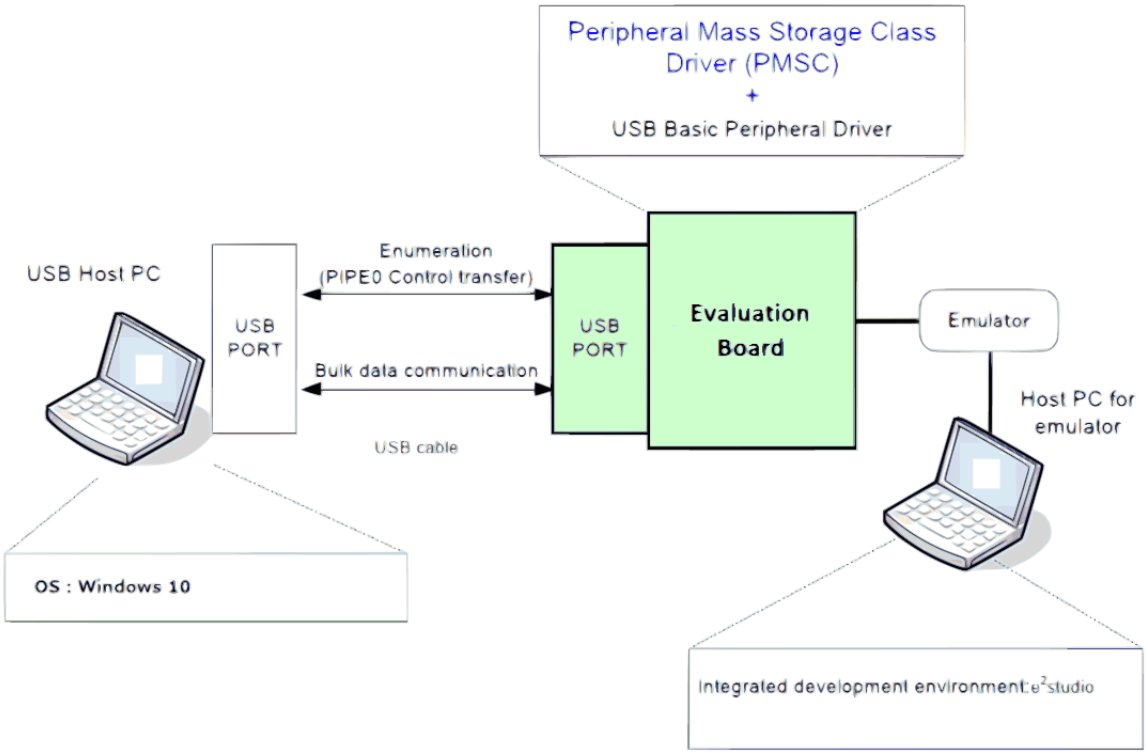



Figure 141: Example Operating Environment

This is a pmsc example of minimal use of the USB in an application.

```
void usb_pmsc_example (void)
{
    usb_event_info_t usb_event;
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#else  /* (BSP_CFG_RTOS == 2) */
 usb_status_t event;
#endif /* (BSP_CFG_RTOS == 2) */
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 /* Loop back between PC(TerminalSoft) and USB MCU */
 while (1)
    {
#if (BSP_CFG_RTOS == 2)
       USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **) &p_mess);
       usb_event = *p_mess;
 /* Analyzing the received message */
 switch (usb_event.event)
#else  /* (BSP_CFG_RTOS == 2) */
       g_usb_on_usb.eventGet(&usb_event, &event);
 switch (event)
#endif  /* (BSP_CFG_RTOS == 2) */
      {
 case USB_STATUS_CONFIGURED:
      {
 break;
            }
 case USB_STATUS_SUSPEND:
 case USB_STATUS_DETACH:
      {
#if USB_SUPPORT_LPW == USB_APL_ENABLE
// @@ low_power_mcu();
#endif  /* USB_SUPPORT_LPW == USB_APL_ENABLE */
 break;
            }
```

```
 default:
      {
 break;
            }
       }
    }
} /* End of function usb_main() */
```

# Descriptor

A template for PMSC descriptors can be found in ra/fsp/src/r_usb_pmsc/r_usb_pmsc_descriptor.c.template. Also, please be sure to use your vendor ID.

## 4.2.53 Watchdog Timer (r_wdt)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | R_WDT_Refresh (wdt_ctrl_t *const p_ctrl) |
| fsp_err_t | R_WDT_Open (wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg) |
| fsp_err_t | R_WDT_StatusClear (wdt_ctrl_t *const p_ctrl, const wdt_status_t status) |
| fsp_err_t | R_WDT_StatusGet (wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status) |
| fsp_err_t | R_WDT_CounterGet (wdt_ctrl_t *const p_ctrl, uint32_t *const p_count) |
| fsp_err_t | R_WDT_TimeoutGet (wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout) |
| fsp_err_t | R_WDT_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Driver for the WDT peripheral on RA MCUs. This module implements the WDT Interface.

# Overview

The watchdog timer is used to recover from unexpected errors in an application. The watchdog timer must be refreshed periodically in the permitted count window by the application. 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.

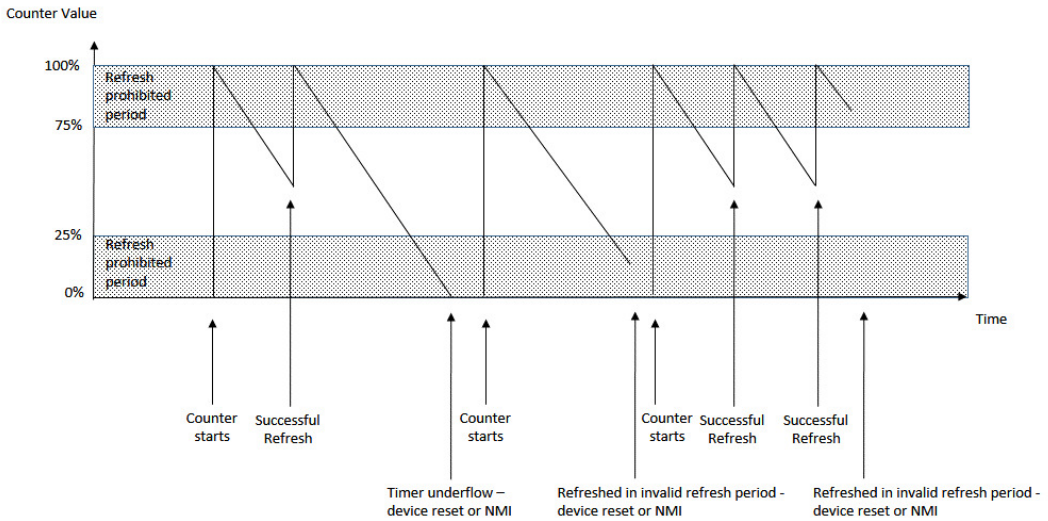



Figure 142: Watchdog Timer Operation Example

### 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
- The WDT has two supported modes:
  - In auto start mode, the WDT begins counting at reset.
  - In register start mode, the WDT can be started from the application.

### Selecting a Watchdog

RA MCUs have two watchdog peripherals: the watchdog timer (WDT) and the independent watchdog timer (IWDT). When selecting between them, consider these factors:

| | WDT | IWDT |
|---|---|---|
| Start Mode | The WDT can be started from the application (register start | The IWDT can only be configured by hardware to start |

| | | |
|---|---|---|
| | mode) or configured by hardware to start automatically (auto start mode). | automatically. |
| Clock Source | The WDT runs off a peripheral clock. | The IWDT has its own clock source which improves safety. |

# Configuration

When using register start mode, configure the watchdog timer on the Stacks tab.

*Note*

*When using auto start mode, configurations on the **Stacks** tab are ignored. Configure the watchdog using the OFS settings on the BSP tab.*

### Build Time Configurations for r_wdt

The following build time configurations are defined in fsp_cfg/r_wdt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Register Start NMI Support | • Enabled<br>• Disabled | Disabled | If enabled, code for NMI support in register start mode is included in the build. |

### Configurations for Driver > Monitoring > Watchdog Driver on r_wdt

This module can be added to the Stacks tab via New Stack > Driver > Monitoring > Watchdog Driver on r_wdt:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_wdt0 | Module name. |
| Timeout | • 1,024 Cycles<br>• 4,096 Cycles<br>• 8,192 Cycles<br>• 16,384 Cycles | 16,384 Cycles | Select the watchdog timeout in cycles. |
| Clock Division Ratio | • PCLK/4<br>• PCLK/64<br>• PCLK/128<br>• PCLK/512<br>• PCLK/2048<br>• PCLK/8192 | PCLK/8192 | Select the watchdog clock divisor. |
| Window Start Position | • 100% (Window | 100% (Window Position | Select the allowed |

| | | | |
|---|---|---|---|
| | Position Not Specified)<br>• 75%<br>• 50%<br>• 25 | Not Specified) | watchdog refresh start point. |
| Window End Position | • 0% (Window Position Not Specified)<br>• 25%<br>• 50%<br>• 75% | 0% (Window Position Not Specified) | Select the allowed watchdog refresh end point. |
| Reset Control | • Reset Output<br>• NMI Generated | Reset Output | Select what happens when the watchdog timer expires. |
| Stop Control | • WDT Count Enabled in Low Power Mode<br>• WDT Count Disabled in Low Power Mode | WDT Count Disabled in Low Power Mode | Select the watchdog state in low power mode. |
| NMI Callback | Name must be a valid C symbol | NULL | A user callback function must be provided if the WDT is configured to generate an NMI when the timer underflows or a refresh error occurs. If this callback function is provided, it will be called from the NMI handler each time the watchdog triggers. |

### Clock Configuration

The WDT clock is based on the PCLKB frequency. You can set the PCLKB frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time. The maximum timeout period with PCLKB running at 60 MHz is approximately 2.2 seconds.

### Pin Configuration

This module does not use I/O pins.

## Usage Notes

### NMI Interrupt

The watchdog timer uses the NMI, which is enabled by default. No special configuration is required. When the NMI is triggered, the callback function registered during open is called.

*Note*

*When using the WDT in software start mode with NMI and the timer underflows, the WDT status must be reset by*

*calling R_WDT_StatusClear before restarting the timer via R_WDT_Refresh.*

### Period Calculation

The WDT operates from PCLKB. With a PCLKB of 60 MHz, the maximum 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

### Limitations

Developers should be aware of the following limitations when using the WDT:

- When using a J-Link debugger the WDT counter does not count and therefore will not reset the device or generate an NMI. To enable the watchdog to count and generate a reset or NMI while debugging, add this line of code in the application:

```
/* (Optional) Enable the WDT to count and generate NMI or reset when the
 * debugger is connected. */
   R_DEBUG->DBGSTOPCR_b.DBGSTOP_WDT = 0;
```

- If the WDT is configured to stop the counter in low power mode, then your application must restart the watchdog by calling R_WDT_Refresh() after the MCU wakes from low power mode.

## Examples

### WDT Basic Example

This is a basic example of minimal use of the WDT in an application.

```
void wdt_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* In auto start mode, the WDT starts counting immediately when the MCU is powered
on. */
 /* Initializes the module. */
    err = R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* In register start mode, start the watchdog by calling R_WDT_Refresh. */
```

```
    err = R_WDT_Refresh(&g_wdt0_ctrl);
    handle_error(err);
 while (true)
    {
 /* Application work here. */
 /* Refresh before the counter underflows to prevent reset or NMI. */
      err = R_WDT_Refresh(&g_wdt0_ctrl);
      handle_error(err);
    }
}
```

### WDT Advanced Example

This example demonstrates using a start window and gives an example callback to handle an NMI generated by an underflow or refresh error.

```
#define WDT_TIMEOUT_COUNTS (16384U)
#define WDT_MAX_COUNTER (WDT_TIMEOUT_COUNTS - 1U)
#define WDT_START_WINDOW_75 ((WDT_MAX_COUNTER * 3) / 4)
/* Example callback called when a watchdog NMI occurs. */
void wdt_callback (wdt_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) Determine the source of the NMI. */
 wdt_status_t status = WDT_STATUS_NO_ERROR;
    err = R_WDT_StatusGet(&g_wdt0_ctrl, &status);
    handle_error(err);
 /* (Optional) Log source of NMI and any other debug information. */
 /* (Optional) Clear the error flags. */
    err = R_WDT_StatusClear(&g_wdt0_ctrl, status);
    handle_error(err);
 /* (Register start mode) In register start mode, call R_WDT_Refresh() to
  * continue using the watchdog after an error. */
    err = R_WDT_Refresh(&g_wdt0_ctrl);
```

```
    handle_error(err);
 /* (Optional) Issue a software reset to reset the MCU. */
    __NVIC_SystemReset();
}
void wdt_advanced_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) Enable the WDT to count and generate NMI or reset when the
  * debugger is connected. */
    R_DEBUG->DBGSTOPCR_b.DBGSTOP_WDT = 0;
 /* (Optional) Check if the WDTRF flag is set to know if the system is
  * recovering from a WDT reset. */
 if (R_SYSTEM->RSTSR1_b.WDTRF)
    {
 /* Clear the flag. */
       R_SYSTEM->RSTSR1 = 0U;
    }
 /* Open the module. */
    err = R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Initialize other application code. */
 /* (Register start mode) Call R_WDT_Refresh() to start the WDT in register
  * start mode. Do not call R_WDT_Refresh() in auto start mode unless the
  * counter is in the acceptable refresh window. */
    err = R_WDT_Refresh(&g_wdt0_ctrl);
    handle_error(err);
 while (true)
    {
 /* Application work here. */
 /* (Optional) If there is a chance the application takes less time than
  * the start window, verify the WDT counter is past the start window
  * before refreshing the WDT. */
       uint32_t wdt_counter = 0U;
```

```
do
     {
/* Read the current WDT counter value. */
           err = R_WDT_CounterGet(&g_wdt0_ctrl, &wdt_counter);
     handle_error(err);
     } while (wdt_counter >= WDT_START_WINDOW_75);
/* Refresh before the counter underflows to prevent reset or NMI. */
     err = R_WDT_Refresh(&g_wdt0_ctrl);
     handle_error(err);
   }
}
```

## Data Structures

| | |
|---|---|
| struct | wdt_instance_ctrl_t |

## Data Structure Documentation

### ◆ wdt_instance_ctrl_t

| struct wdt_instance_ctrl_t |
|---|
| WDT private control block. DO NOT MODIFY. Initialization occurs when R_WDT_Open() is called. |

## Function Documentation

### ◆ R_WDT_Refresh()

fsp_err_t R_WDT_Refresh ( wdt_ctrl_t *const *p_ctrl*)

Refresh the watchdog timer. Implements wdt_api_t::refresh.

In addition to refreshing the watchdog counter this function can be used to start the counter in register start mode.

Example:

```
/* Refresh before the counter underflows to prevent reset or NMI. */

    err = R_WDT_Refresh(&g_wdt0_ctrl);

    handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WDT successfully refreshed. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not initialized. |

***Note***

*This function only returns FSP_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.*

### ◆ R_WDT_Open()

fsp_err_t R_WDT_Open ( wdt_ctrl_t *const *p_ctrl*, wdt_cfg_t const *const *p_cfg* )

Configure the WDT in register start mode. In auto-start_mode the NMI callback can be registered. Implements wdt_api_t::open.

This function should only be called once as WDT configuration registers can only be written to once so subsequent calls will have no effect.

Example:

```
/* Initializes the module. */

   err = R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WDT successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |

***Note***

*In auto start mode the only valid configuration option is for registering the callback for the NMI ISR if NMI output has been selected.*

### ◆ R_WDT_StatusClear()

fsp_err_t R_WDT_StatusClear ( wdt_ctrl_t *const *p_ctrl*, const wdt_status_t *status* )

Clear the WDT status and error flags. Implements wdt_api_t::statusClear.

Example:

```
/* (Optional) Clear the error flags. */
   err = R_WDT_StatusClear(&g_wdt0_ctrl, status);
   handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WDT flag(s) successfully cleared. |
| FSP_ERR_ASSERTION | Null pointer as a parameter. |
| FSP_ERR_NOT_OPEN | Instance control block is not initialized. |
| FSP_ERR_UNSUPPORTED | This function is only valid if the watchdog generates an NMI when an error occurs. |

***Note***

*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.*

### ◆ R_WDT_StatusGet()

fsp_err_t R_WDT_StatusGet ( wdt_ctrl_t *const *p_ctrl*, wdt_status_t *const *p_status* )

Read the WDT status flags. Implements wdt_api_t::statusGet.

Indicates both status and error conditions.

Example:

```
/* (Optional) Determine the source of the NMI. */
wdt_status_t status = WDT_STATUS_NO_ERROR;
   err = R_WDT_StatusGet(&g_wdt0_ctrl, &status);
   handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WDT status successfully read. |
| FSP_ERR_ASSERTION | Null pointer as a parameter. |
| FSP_ERR_NOT_OPEN | Instance control block is not initialized. |
| FSP_ERR_UNSUPPORTED | This function is only valid if the watchdog generates an NMI when an error occurs. |

***Note***

*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.*

### ◆ R_WDT_CounterGet()

fsp_err_t R_WDT_CounterGet ( wdt_ctrl_t *const *p_ctrl*, uint32_t *const *p_count* )

Read the current count value of the WDT. Implements wdt_api_t::counterGet.

Example:

```
/* Read the current WDT counter value. */
           err = R_WDT_CounterGet(&g_wdt0_ctrl, &wdt_counter);
     handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WDT current count successfully read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Instance control block is not initialized. |

### ◆ R_WDT_TimeoutGet()

fsp_err_t R_WDT_TimeoutGet ( wdt_ctrl_t *const *p_ctrl*, wdt_timeout_values_t *const *p_timeout* )

Read timeout information for the watchdog timer. Implements wdt_api_t::timeoutGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WDT timeout information retrieved successfully. |
| FSP_ERR_ASSERTION | Null Pointer. |
| FSP_ERR_NOT_OPEN | Instance control block is not initialized. |

### ◆ R_WDT_VersionGet()

fsp_err_t R_WDT_VersionGet ( fsp_version_t *const *p_version*)

Return WDT HAL driver version. Implements wdt_api_t::versionGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version information successfully read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter |

## 4.2.54 AWS PKCS11 PAL (rm_aws_pkcs11_pal)

Modules

PKCS#11 PAL layer implementation for use by FreeRTOS TLS.

# Overview

*Note*

***The PKCS#11 PAL Interface does not provide any interfaces to the user. Consult the AWS documentation for more info: https://docs.aws.amazon.com/freertos/latest/portingguide/afr-porting-pkcs.html.***

# Configuration

There is no user configuration for this module

### Data Flash Usage

The current implementation utilizes 16K of Data flash of which 8K is used for storage and the other 8K is used for backup.

## Usage Notes

### Limitations

- Interrupts are disabled while write or erase operations are being performed.
- Credentials are stored on data flash with no tamper protection other than SHA256 for integrity.
- Credential access is not limited in any way. The credential access and tamper issues can be resolved by updating the implementation to use code flash instead of data flash and using the Secure MPU to control access to it.

## 4.2.55 AWS PKCS11 PAL LITTLEFS (rm_aws_pkcs11_pal_littlefs)

Modules

PKCS#11 PAL LittleFS layer implementation for use by FreeRTOS TLS.

## Overview

*Note*

***The PKCS#11 PAL LittleFS Interface does not provide any interfaces to the user. Consult the AWS documentation for more info: https://docs.aws.amazon.com/freertos/latest/portingguide/afr-porting-pkcs.html.***

## Configuration

There is no user configuration for this module

## Usage Notes

The current implementation utilizes LittleFS Flash Port (rm_littlefs_flash) for storage.

### Limitations

- Credential access is not limited in any way.

## 4.2.56 Bluetooth Low Energy Abstraction (rm_ble_abs)

Modules

## Functions

| | |
|---|---|
| fsp_err_t | RM_BLE_ABS_Open (ble_abs_ctrl_t *const p_ctrl, ble_abs_cfg_t const *const p_cfg) |
| fsp_err_t | RM_BLE_ABS_Close (ble_abs_ctrl_t *const p_ctrl)<br>Close the BLE channel. Implements ble_abs_api_t::close. More... |
| fsp_err_t | RM_BLE_ABS_Reset (ble_abs_ctrl_t *const p_ctrl, ble_event_cb_t init_callback) |
| fsp_err_t | RM_BLE_ABS_VersionGet (fsp_version_t *const p_version) |
| fsp_err_t | RM_BLE_ABS_StartLegacyAdvertising (ble_abs_ctrl_t *const p_ctrl, ble_abs_legacy_advertising_parameter_t const *const p_advertising_parameter) |
| fsp_err_t | RM_BLE_ABS_StartExtendedAdvertising (ble_abs_ctrl_t *const p_ctrl, ble_abs_extend_advertising_parameter_t const *const p_advertising_parameter) |
| fsp_err_t | RM_BLE_ABS_StartNonConnectableAdvertising (ble_abs_ctrl_t *const p_ctrl, ble_abs_non_connectable_advertising_parameter_t const *const p_advertising_parameter) |
| fsp_err_t | RM_BLE_ABS_StartPeriodicAdvertising (ble_abs_ctrl_t *const p_ctrl, ble_abs_periodic_advertising_parameter_t const *const p_advertising_parameter) |
| fsp_err_t | RM_BLE_ABS_StartScanning (ble_abs_ctrl_t *const p_ctrl, ble_abs_scan_parameter_t const *const p_scan_parameter) |
| fsp_err_t | RM_BLE_ABS_CreateConnection (ble_abs_ctrl_t *const p_ctrl, ble_abs_connection_parameter_t const *const p_connection_parameter) |
| fsp_err_t | RM_BLE_ABS_SetLocalPrivacy (ble_abs_ctrl_t *const p_ctrl, uint8_t const *const p_lc_irk, uint8_t privacy_mode) |
| fsp_err_t | RM_BLE_ABS_StartAuthentication (ble_abs_ctrl_t *const p_ctrl, uint16_t connection_handle) |

## Detailed Description

Middleware for the Bluetooth peripheral on RA MCUs. This module implements the BLE ABS Interface .

# Overview

This module provides BLE GAP functionality that complies with the Bluetooth Core Specification version 5.0 specified by the Bluetooth SIG. This module is configured via the QE for BLE.

### Features

The Bluetooth Low Energy Abstraction module supports the following features:

- following GAP Role support
  - Central: The device that sends a connection request to the Peripheral device.
  - Peripheral: The device that accepts a connection request from Central and establishes a connection.
  - Observer : The device that scans for advertising.
  - Broadcaster : The device that sends advertising.
- LE 2M PHY
  - BLE communication is supported on the 2 Msym/s PHY.
- LE Coded PHY -Supports BLE communication on the Coded PHY. This enables communication over longer distances than 1M PHY and 2M PHY.
- LE Advertising Extensions
  - Up to four independent adverts can be executed simultaneously.
  - The size of Advertising Data/Scan Response Data has been expanded to a maximum of 1650 bytes.
  - Periodic Advertising is available.
- LE Channel Selection Algorithm #2
  - With the hopping channel selection algorithm added in Version 5.0, the machine that selects the channel It is possible.
- High Duty Cycle Non-Connectable Advertising
  - The ability to support non-connectable advertising with a minimum interval of up to 20 msec.
- LE Secure Connections
  - Elliptic curve Diffie-Hellman key sharing (ECDH) for pairing with passive eavesdropping support.
- Link Layer privacy
  - This feature avoids being tracked by other BLE devices by periodically changing the Bluetooth device address.
- Link Layer Extended Scanner Filter policies
  - Scan Filter support for Resolvable private addresses.
- LE Data Packet Length Extension
  - This function expands the packet size of BLE data communications. It is possible to scale up to 251 bytes.
- LE L2CAP Connection Oriented Channel Support
  - The ability to support communication using the L2CAP credit based flow control channel.
- Low Duty Cycle Directed Advertising
  - The ability to support the advertising of the Low Duty Cycle for reconnecting to a known device.
- LE Link Layer Topology
  - It supports both Master and Slave roles and can operate as Master when connected to one remote device and as Slave when connected to another remote device.
- LE Ping
  - This function checks whether the link is maintained or not by requesting the transmission of packets containing MIC after link encryption.

### BLE Library Configuration

There are three types of BLE Protocol Stacks, and the functions provided are different depending on the type of BLE Protocol Stack you select.

| BLE library feature | All | Balance | Compact |
|---|---|---|---|
| GAP Role | Central Peripheral Observer Broadcaster | Central Peripheral Observer Broadcaster | Peripheral Broadcaster |
| LE 2M PHY | Yes | Yes | No |
| LE Coded PHY | Yes | Yes | No |
| LE Advertising Extensions | Yes | No | No |
| LE Channel Selection Algorithm #2 | Yes | Yes | No |
| High Duty Cycle Non-Connectable Advertising | Yes | Yes | Yes |
| LE Secure Connections | Yes | Yes | Yes |
| Link Layer privacy | Yes | Yes | Yes |
| Link Layer Extended Scanner Filter policies | Yes | Yes | No |
| LE Data Packet Length Extension | Yes | Yes | Yes |
| LE L2CAP Connection Oriented Channel Support | Yes | No | No |
| Low Duty Cycle Directed Advertising | Yes | Yes | Yes |
| LE Link Layer Topology | Yes | Yes | No |
| LE Ping | Yes | Yes | Yes |
| 32-bit UUID Support in LE | Yes | Yes | Yes |

### Target Devices

The Bluetooth Low Energy Abstraction module supports the following devices.

- RA4W1

## Configuration

### Build Time Configurations for rm_ble_abs

The following build time configurations are defined in fsp_cfg/rm_ble_abs_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Debug Public Address | Manual Entry | {0xFF,0xFF,0xFF,0x50,0x90,0x74} | Public Address of firmware initial value. |
| Debug Random Address | Manual Entry | {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF} | Random Address of firmware initial value. |
| Maximum number of connections | Value must be greater than 1 | 7 | Maximum number of connections. |
| Maximum connection data length | Value must be greater than 27 | 251 | Maximum connection data length. |
| Maximum advertising data length | Value must be greater than 31 | 1650 | Maximum advertising data length. |
| Maximum advertising set number | Value must be greater than 1 | 4 | Maximum advertising set number. |
| Maximum periodic sync set number. | Value must be greater than 1 | 2 | Maximum periodic sync set number. |
| Store Security Data | • Disable<br>• Enable | Disable | Store Security Data in DataFlash. |
| Data Flash Block for Security Data | Value must be greater than 0 | 0 | Data Flash Block for Security Data Management. |
| Remote Device Bonding Number | Value must be greater than 1 | 7 | Number of remote device bonding information. |
| Connection Event Start Notify | • Disable notify<br>• Enable notify | Disable notify | Set Connection event start notify enable/disable. |
| Connection Event Close Notify | • Disable notify<br>• Enable notify | Disable notify | Set Connection event close notify enable/disable. |
| Advertising Event Start Notify | • Disable notify<br>• Enable notify | Disable notify | Set Advertising event start notify enable/disable. |
| Advertising Event Close Notify | • Disable notify<br>• Enable notify | Disable notify | Set Advertising event close notify enable/disable. |
| Scanning Event Start Notify | • Disable notify<br>• Enable notify | Disable notify | Set Scanning event start notify enable/disable. |
| Scanning Event Close Notify | • Disable notify<br>• Enable notify | Disable notify | Set Scanning event close notify enable/disable. |
| Initiating Event Start Notify | • Disable notify<br>• Enable notify | Disable notify | Set Initiating event start notify enable/disable. |

| | | | |
|---|---|---|---|
| Initiating Event Close Notify | • Disable notify<br>• Enable notify | Disable notify | Set Initiating event close notify enable/disable. |
| RF Deep Sleep Start Notify | • Disable notify<br>• Enable notify | Disable notify | Set RF_DEEP_SLEEP start notify enable/disable. |
| RF Deep Sleep Wakeup Notify | • Disable notify<br>• Enable notify | Disable notify | Set RF_DEEP_SLEEP wakeup notify enable/disable. |
| Bluetooth dedicated clock | Value must be greater than 0 | 6 | Load capacitance adjustment. |
| DC-DC Converter | • Disable DC-DC Converter<br>• Enable DC-DC Converter | Disable DC-DC Converter | Set DC-DC converter for RF part. |
| Slow Clock Source | • Use RF_LOCO<br>• Use External 32.768kHz | Use RF_LOCO | Set slow clock source for RF part. |
| MCU CLKOUT Port | • P109<br>• P205 | P109 | When BLE_ABS_CFG_RF_EXTERNAL_32K_ENABLE = 1, Set port of MCU CLKOUT. |
| MCU CLKOUT Frequency Output | • MCU CLKOUT frequency 32.768kHz<br>• MCU CLKOUT frequency 16.384kHz | MCU CLKOUT frequency 32.768kHz | When BLE_ABS_CFG_RF_EXTERNAL_32K_ENABLE = 1, set frequency output from CLKOUT of MCU part. |
| Sleep Clock Accuracy(SCA) | Value must be greater than 0 | 250 | When BLE_ABS_CFG_RF_EXTERNAL_32K_ENABLE = 1, set Sleep Clock Accuracy(SCA) for RF slow clock. |
| Transmission Power Maximum Value | • max +0dBm<br>• max +4dBm | max +4dBm | Set transmission power maximum value. |
| Transmission Power Default Value | • High 0dBm(Transmission Power Maximum Value = +0dBm) / +4dBm(Transmission Power Maximum Value = +4dBm)<br>• Mid 0dBm(Transmission Power Maximum Value = +0dBm) / 0dBm(Transmissio | High 0dBm(Transmission Power Maximum Value = +0dBm) / +4dBm(Transmission Power Maximum Value = +4dBm) | Set default transmit power. Default transmit power is dependent on the configuration of Maximum transmission power(BLE_ABS_CFG_RF_MAX_TX_POW). |

| | | | |
|---|---|---|---|
| | n Power Maximum Value = +4dBm)<br>• Low -18dBm(Transmission Power Maximum Value = +0dBm) / -20 dBm(Transmission Power Maximum Value = +4dBm) | | |
| CLKOUT_RF Output | • No output<br>• 4MHz output<br>• 2MHz output<br>• 1MHz output | No output | Set CLKOUT_RF output setting. |
| RF_DEEP_SLEEP Transition | • Disable<br>• Enable | Enable | Set RF_DEEP_SLEEP transition. |
| MCU Main Clock Frequency | Value must be greater than 1000 | 8000 | Set MCU Main Clock Frequency (kHz). Set clock source according to your board environment. HOCO: don't care. / Main Clock: 1000 to 20000 kHz / PLL Circuit: 4000 to 12500 kHz |
| Code Flash(ROM) Device Data Block | Value must be greater than -1 | 255 | Device specific data block on Code Flash (ROM). |
| Device Specific Data Flash Block | Value must be greater than -1 | -1 | Device specific data block on E2 Data Flash. |
| MTU Size Configured | Value must be greater than 23 | 247 | MTU Size configured by GATT MTU exchange procedure. |
| Timer Slot Maximum Number | Manual Entry | 10 | The maximum number of timer slot. |
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | Specify whether to include code for API parameter checking. Valid settings include. |

**Configurations for Driver > Network > BLE Abstraction Driver on rm_ble_abs**

This module can be added to the Stacks tab via New Stack > Driver > Network > BLE Abstraction Driver on rm_ble_abs:

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Interrupts > Callback provided when an ISR occurs | Name must be a valid C symbol | NULL | Callback provided when BLE ABS ISR occurs |
| Name | Name must be a valid C symbol | g_ble_abs0 | Module name. |
| Gap callback | Name must be a valid C symbol | gap_cb | A user callback function must be provided if the BLE_ABS is configured to generate a GAP. If QE is used, set to NULL. |
| Vendor specific callback | Name must be a valid C symbol | vs_cb | A user callback function must be provided if the BLE_ABS is configured to generate a Vendor Specific. If QE is used, set to NULL. |
| Pairing parameters | Name must be a valid C symbol | gs_abs_pairing_param | Set pairing parameters. |
| GATT server callback parameter | Name must be a valid C symbol | gs_abs_gatts_cb_param | Set GATT server callback parameter. If QE is used, set to NULL. |
| GATT server callback number | Must be a valid number | 2 | The number of GATT Server callback functions. |
| GATT client callback parameter | Name must be a valid C symbol | gs_abs_gattc_cb_param | Set GATT client callback parameter. If QE is used, set to NULL. |
| GATT client callback number | Must be a valid number | 2 | The number of GATT Server callback functions. |
| IO capabilities of local device. | • BLE_GAP_IOCAP_DISPLAY_ONLY<br>• BLE_GAP_IOCAP_DISPLAY_YESNO<br>• BLE_GAP_IOCAP_KEYBOARD_ONLY<br>• BLE_GAP_IOCAP_NOINPUT_NOOUTPUT<br>• BLE_GAP_IOCAP_KEYBOARD_DISPLAY | BLE_GAP_IOCAP_NOINPUT_NOOUTPUT | Select IO capabilities of local device. |

| | | | |
|---|---|---|---|
| MITM protection policy. | • BLE_GAP_SEC_MITM_BEST_EFFORT<br>• BLE_GAP_SEC_MITM_STRICT | BLE_GAP_SEC_MITM_BEST_EFFORT | Select MITM protection policy. |
| Determine whether to accept only Secure Connections or not. | • BLE_GAP_SC_BEST_EFFORT<br>• BLE_GAP_SC_STRICT | BLE_GAP_SC_BEST_EFFORT | Select determine whether to accept only Secure Connections or not. |
| Type of keys to be distributed from local device. | • BLE_GAP_KEY_DIST_ENCKEY<br>• BLE_GAP_KEY_DIST_IDKEY<br>• BLE_GAP_KEY_DIST_SIGNKEY | BLE_GAP_KEY_DIST_ENCKEY | Select type of keys to be distributed from local device. |
| Type of keys which local device requests a remote device to distribute. | Must be a valid number | 0 | Set type of keys which local device requests a remote device to distribute. |
| Maximum LTK size. | Must be a valid number | 16 | Set Maximum LTK size. |

### Clock Configuration

*Note*

*System clock (ICLK): 8 MHz or more*
*Peripheral module clock A (PCLKA): 8MHz or more*
*The BLE Protocol Stack is optimized for ICLK and PCLKA frequencies of 32 MHz.*
*It is recommended that the clock be set so that the ICLK and PCLKA frequencies are 32MHz in order to get the best performance from the BLE.*

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### Limitations

Developers should be aware of the following limitations when using the BLE_ABS:

# Examples

### BLE_ABS Basic Example

This is a basic example of minimal use of the BLE_ABS in an application.

```
#define BLE_ABS_EVENT_FLAG_ADV_ON (0x01 << 0)

#define BLE_ABS_EVENT_FLAG_ADV_OFF (0x01 << 1)

#define BLE_ABS_EVENT_FLAG_CONN_IND (0x01 << 2)
```

```
#define BLE_ABS_EVENT_FLAG_DISCONN_IND (0x01 << 3)
#define BLE_ABS_EVENT_FLAG_ADV_DATA_UPD_COMP (0x01 << 4)
#define BLE_ABS_EVENT_FLAG_RSLV_LIST_CONF_COMP (0x01 << 5)
#define BLE_ABS_EXAMPLE_SHORTENED_LOCAL_NAME 0x45, 0x78, 0x61, 0x6d, 0x70, 0x6c, 0x65
#define BLE_ABS_EXAMPLE_COMPLETE_LOCAL_NAME 0x54, 0x45, 0x53, 0x54, 0x5f, 0x45, 0x78,
0x61, 0x6d, 0x70, 0x6c, \
  0x65
#define BLE_ABS_EXAMPLE_SLOW_ADVERTISING_INTERVAL (0x00000640)
void ble_abs_peripheral_example (void)
{
 fsp_err_t         err    = FSP_SUCCESS;
 volatile uint32_t timeout = UINT16_MAX * UINT8_MAX * 8;
    uint8_t * p_lc_irk    = NULL;
    uint8_t   privacy_mode = BLE_GAP_NET_PRIV_MODE;
    uint8_t advertising_data[] =
    {
 /* Flags */
       0x02,
       0x01,
       (0x1a),
 /* Shortened Local Name */
       0x08,
       0x08,
       BLE_ABS_EXAMPLE_SHORTENED_LOCAL_NAME,
    };
 /* Scan Response Data */
    uint8_t scan_response_data[] =
    {
 /* Complete Local Name */
       0x0D,
       0x09,
       BLE_ABS_EXAMPLE_COMPLETE_LOCAL_NAME,
    };
 ble_abs_legacy_advertising_parameter_t legacy_advertising_parameter =
```

```
    {
        .p_peer_address             =
NULL,
        .slow_advertising_interval =
BLE_ABS_EXAMPLE_SLOW_ADVERTISING_INTERVAL,
        .slow_advertising_period    =
0x0000,
        .p_advertising_data         =
advertising_data,
        .advertising_data_length    = sizeof
(advertising_data),
        .p_scan_response_data       =
scan_response_data,
        .scan_response_data_length = sizeof
(scan_response_data),
        .advertising_filter_policy = BLE_ABS_ADVERTISING_FILTER_ALLOW_ANY
,
        .advertising_channel_map    = (BLE_GAP_ADV_CH_37 | BLE_GAP_ADV_CH_38 |
BLE_GAP_ADV_CH_39),
        .own_bluetooth_address_type = BLE_GAP_ADDR_PUBLIC
,
        .own_bluetooth_address      = {0},
    };
    g_ble_event_flag = 0;
 /* Open the module. */
    err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 do
    {
 if (1 == R_BLE_IsTaskFree())
      {
 /* Set local privacy. */
            err = RM_BLE_ABS_SetLocalPrivacy(&g_ble_abs0_ctrl, p_lc_irk,
```

```
privacy_mode);
 /* Handle any errors. This function should be defined by the user. */
      handle_error(err);
       }
 R_BLE_Execute();
    } while ((BLE_ABS_EVENT_FLAG_RSLV_LIST_CONF_COMP != (g_ble_event_flag &
BLE_ABS_EVENT_FLAG_RSLV_LIST_CONF_COMP)) &&
             (timeout-- > 0U));
    time_out_handle_error(timeout);
    g_ble_event_flag = 0;
    timeout = UINT16_MAX * UINT8_MAX * 8;
 do
    {
 if (1 == R_BLE_IsTaskFree())
      {
 /* Start advertising. */
            err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
 /* Handle any errors. This function should be defined by the user. */
      handle_error(err);
       }
 R_BLE_Execute();
    } while ((BLE_ABS_EVENT_FLAG_ADV_ON != (g_ble_event_flag &
BLE_ABS_EVENT_FLAG_ADV_ON)) && (timeout-- > 0U));
    time_out_handle_error(timeout);
 /* Wait scan or connection from remote device. */
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_SECONDS);
 /* Clean up & Close BLE driver */
    g_ble_event_flag = 0;
 /* Close BLE driver */
    err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
}
```

```
#define BLE_ABS_EVENT_FLAG_CONN_IND (0x01 << 0)
#define BLE_ABS_EVENT_FLAG_ADV_REPT_IND (0x01 << 1)
#define BLE_ABS_EVENT_FLAG_PAIRING_COMP (0x01 << 2)
#define BLE_ABS_EXAMPLE_FAST_SCAN_INTERVAL (0x0060)
#define BLE_ABS_EXAMPLE_FAST_SCAN_WINDOW (0x0030)
#define BLE_ABS_EXAMPLE_SLOW_SCAN_INTERVAL (0x0800)
#define BLE_ABS_EXAMPLE_SLOW_SCAN_WINDOW (0x0012)
#define BLE_ABS_EXAMPLE_FAST_SCAN_PERIOD (0x0BB8)
#define BLE_ABS_EXAMPLE_SLOW_SCAN_PERIOD (0x0000)
#define BLE_ABS_EXAMPLE_CONNECTION_INTERVAL (0x0028)
#define BLE_ABS_EXAMPLE_SUPERVISION_TIMEOUT (0x0200)
void ble_abs_central_example (void)
{
 fsp_err_t         err    = FSP_SUCCESS;
 volatile uint32_t timeout = UINT16_MAX * UINT8_MAX * 8;
    connection_handle = BLE_GAP_INVALID_CONN_HDL;
 static ble_abs_scan_phy_parameter_t scan_phy_parameter =
    {
       .fast_scan_interval = BLE_ABS_EXAMPLE_FAST_SCAN_INTERVAL, /* 60.0(ms) */
       .fast_scan_window   = BLE_ABS_EXAMPLE_FAST_SCAN_WINDOW,   /* 30.0(ms) */
       .slow_scan_interval = BLE_ABS_EXAMPLE_SLOW_SCAN_INTERVAL, /* 1,280.0(ms) */
       .slow_scan_window   = BLE_ABS_EXAMPLE_SLOW_SCAN_WINDOW,   /* 11.25(ms) */
       .scan_type          = BLE_GAP_SCAN_ACTIVE
    };
 /* Scan parameters */
 ble_abs_scan_parameter_t scan_parameter =
    {
       .p_phy_parameter_1M       = &scan_phy_parameter,
       .fast_scan_period          = BLE_ABS_EXAMPLE_FAST_SCAN_PERIOD, /* 30,000(ms)
*/
       .slow_scan_period          = BLE_ABS_EXAMPLE_SLOW_SCAN_PERIOD,
       .p_filter_data             = NULL,
       .filter_data_length       = 0,
```

```
        .filter_ad_type            = 0x09,                            /* Data type:
Complete Local Name */
        .device_scan_filter_policy = BLE_GAP_SCAN_ALLOW_ADV_ALL,
        .filter_duplicate          = BLE_GAP_SCAN_FILT_DUPLIC_ENABLE,
    };
 /* Connection phy parameters */
 ble_abs_connection_phy_parameter_t connection_phy_parameter =
    {
        .connection_interval      = BLE_ABS_EXAMPLE_CONNECTION_INTERVAL, /* 50.0(ms)
*/
        .supervision_timeout      = BLE_ABS_EXAMPLE_SUPERVISION_TIMEOUT, /* 5,120(ms)
*/
        .connection_slave_latency = 0x0000,
    };
 /* Connection device address */
 ble_device_address_t connection_device_address;
 /* Connection parameters */
 ble_abs_connection_parameter_t connection_parameter =
    {
        .p_connection_phy_parameter_1M = &connection_phy_parameter,
        .p_device_address              = &connection_device_address,
        .filter_parameter              = BLE_GAP_INIT_FILT_USE_ADDR,
        .connection_timeout             = 0x05, /* 5(s) */
    };
    g_ble_event_flag = 0;
 /* Open the module. */
    err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while ((BLE_ABS_EVENT_FLAG_ADV_REPT_IND != (g_ble_event_flag &
BLE_ABS_EVENT_FLAG_ADV_REPT_IND)) &&
          (timeout-- > 0U))
    {
 if (1 == R_BLE_IsTaskFree())
```

```
      {
 /* Start scanning. */
            err = RM_BLE_ABS_StartScanning(&g_ble_abs0_ctrl, &scan_parameter);
 /* Handle any errors. This function should be defined by the user. */
      handle_error(err);
       }
 R_BLE_Execute();
    }
    g_ble_event_flag = 0;
    time_out_handle_error(timeout);
    timeout = UINT16_MAX * UINT8_MAX * 8;
 while ((BLE_ABS_EVENT_FLAG_CONN_IND != (g_ble_event_flag &
BLE_ABS_EVENT_FLAG_CONN_IND)) && (timeout-- > 0U))
    {
 if (1 == R_BLE_IsTaskFree())
      {
 /* Create connection with remote device. */
            err = RM_BLE_ABS_CreateConnection(&g_ble_abs0_ctrl,
&connection_parameter);
 /* Handle any errors. This function should be defined by the user. */
      handle_error(err);
       }
 R_BLE_Execute();
    }
    time_out_handle_error(timeout);
    g_ble_event_flag = 0;
    timeout = UINT16_MAX * UINT8_MAX * 8;
 while ((BLE_ABS_EVENT_FLAG_PAIRING_COMP != (g_ble_event_flag &
BLE_ABS_EVENT_FLAG_PAIRING_COMP)) &&
          (timeout-- > 0U))
    {
 if (1 == R_BLE_IsTaskFree())
      {
 /* Start authentication with remote device. */
```

```
RM_BLE_ABS_StartAuthentication(&g_ble_abs0_ctrl, connection_handle);
/* Handle any errors. This function should be defined by the user. */
     handle_error(err);
      }
R_BLE_Execute();
   }
   time_out_handle_error(timeout);
/* Clean up & Close BLE driver */
   g_ble_event_flag = 0;
   err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
/* Handle any errors. This function should be defined by the user. */
   handle_error(err);
}
```

## Data Structures

| | |
|---|---|
| struct | abs_advertising_parameter_t |
| struct | abs_scan_parameter_t |
| struct | ble_abs_instance_ctrl_t |
| struct | st_ble_rf_notify_t<br>This structure is RF event notify management. More... |

## Typedefs

| | |
|---|---|
| typedef void(* | ble_abs_timer_cb_t) (uint32_t timer_hdl) |
| typedef void(* | ble_mcu_clock_change_cb_t) (void)<br>ble_mcu_clock_change_cb_t is the callback function type to use CLKOUT_RF as the MCU main clock source. More... |
| typedef void(* | ble_rf_notify_cb_t) (uint32_t)<br>ble_rf_notify_cb_t is the RF event notify callback function type. More... |

## Enumerations

| | |
|---|---|
| enum | e_ble_timer_type_t |

## Data Structure Documentation

### ◆ abs_advertising_parameter_t

| struct abs_advertising_parameter_t | | |
|---|---|---|
| advertising set parameters structure | | |
| Data Fields | | |
| union abs_advertising_parameter_t | advertising_parameter | Advertising parameters. |
| uint32_t | advertising_status | Advertising status. |
| ble_device_address_t | remote_device_address | Remote device address for direct advertising. |

### ◆ abs_scan_parameter_t

| struct abs_scan_parameter_t | | |
|---|---|---|
| scan parameters structure | | |
| Data Fields | | |
| ble_abs_scan_parameter_t | scan_parameter | Scan parameters. |
| ble_abs_scan_phy_parameter_t | scan_phy_parameter_1M | 1M phy parameters for scan. |
| ble_abs_scan_phy_parameter_t | scan_phy_parameter_coded | Coded phy parameters for scan. */. |
| uint32_t | scan_status | |

### ◆ ble_abs_instance_ctrl_t

| struct ble_abs_instance_ctrl_t | | |
|---|---|---|
| BLE ABS private control block. DO NOT MODIFY. Initialization occurs when RM_BLE_ABS_Open() is called. | | |
| Data Fields | | |
| uint32_t | open | Indicates whether the open() API has been successfully called. |
| void const * | p_context | Placeholder for user data. Passed to the user callback in ble_abs_callback_args_t. |
| ble_gap_application_callback_t | abs_gap_callback | GAP callback function. |
| ble_vendor_specific_application_callback_t | abs_vendor_specific_callback | Vendor specific callback function. |
| uint32_t | connection_timer_handle | Cancel a request for connection timer. |
| uint32_t | advertising_timer_handle | Advertising timer for legacy advertising. |

| | | |
|---|---|---|
| abs_advertising_parameter_t | advertising_sets[BLE_MAX_NO_OF_ADV_SETS_SUPPORTED] | Advertising set information. |
| abs_scan_parameter_t | abs_scan | Scan information. |
| st_ble_dev_addr_t | loc_bd_addr | Local device address. |
| uint8_t | privacy_mode | Privacy mode. |
| uint32_t | set_privacy_status | Local privacy status. |
| ble_abs_timer_t | timer[BLE_ABS_CFG_TIMER_NUMBER_OF_SLOT] | |
| uint32_t | current_timeout_ms | Current timeout. |
| uint32_t | elapsed_timeout_ms | Elapsed timeout. |
| ble_abs_cfg_t const * | p_cfg | Pointer to the BLE ABS configuration block. |

### ◆ st_ble_rf_notify_t

| struct st_ble_rf_notify_t | | |
|---|---|---|
| This structure is RF event notify management. | | |
| Data Fields | | |
| uint32_t | enable | Set enable/disable of each RF event notification.<br><br>Bit0 Notify Connection event start(0:Disable/1:Enable)<br>Bit1 Notify Advertising event start(0:Disable/1:Enable)<br>Bit2 Notify Scanning event start(0:Disable/1:Enable)<br>Bit3 Notify Initiating event start(0:Disable/1:Enable)<br>Bit4 Notify Connection event close(0:Disable/1:Enable)<br>Bit5 Notify Advertising event close(0:Disable/1:Enable)<br>Bit6 Notify Scanning event close(0:Disable/1:Enable)<br>Bit7 Notify Initiating event close(0:Disable/1:Enable)<br>Bit8 Notify RF_DEEP_SLEEP event start(0:Disable/1:Enable)<br>Bit9 Notify RF_DEEP_SLEEP event close(0:Disable/1:Enable)<br>Other Bit: Reserved for future use. |
| ble_rf_notify_cb_t | start_cb | Set callback function pointer for |

| | | |
|---|---|---|
| | | RF event start. |
| ble_rf_notify_cb_t | close_cb | Set callback function pointer for RF event close. |
| ble_rf_notify_cb_t | dsleep_cb | Set callback function pointer for RF_DEEP_SLEEP. |

## Typedef Documentation

### ◆ ble_abs_timer_cb_t

typedef void(* ble_abs_timer_cb_t) (uint32_t timer_hdl)

The timer callback invoked when the timer expired.

### ◆ ble_mcu_clock_change_cb_t

ble_mcu_clock_change_cb_t

ble_mcu_clock_change_cb_t is the callback function type to use CLKOUT_RF as the MCU main clock source.

**Parameters**

| | |
|---|---|
| none | |

**Returns**

none

### ◆ ble_rf_notify_cb_t

ble_rf_notify_cb_t

ble_rf_notify_cb_t is the RF event notify callback function type.

**Parameters**

| | | |
|---|---|---|
| [in] | uint32_t | The infomation of RF event notification. |

**Returns**

none

## Enumeration Type Documentation

### ◆ e_ble_timer_type_t

| enum e_ble_timer_type_t | |
|---|---|
| The timer type. | |
| Enumerator | |
| BLE_TIMER_ONE_SHOT | One shot timer type |
| BLE_TIMER_PERIODIC | Periodic timer type |

## Function Documentation

### ◆ RM_BLE_ABS_Open()

fsp_err_t RM_BLE_ABS_Open ( ble_abs_ctrl_t *const *p_ctrl*, ble_abs_cfg_t const *const *p_cfg* )

Host stack is initialized with this function. Before using All the R_BLE APIs, it's necessary to call this function. A callback functions are registered with this function. In order to receive the GAP, GATT, Vendor specific event, it's necessary to register a callback function. The result of this API call is notified in BLE_GAP_EVENT_STACK_ON event. Implements ble_abs_api_t::open.

Example:

```
/* Open the module. */

   err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Null pointer presented. |
| FSP_ERR_INVALID_CHANNEL | The channel number is invalid. |
| FSP_ERR_ALREADY_OPEN | Requested channel is already open in a different configuration. |
| FSP_ERR_INVALID_ARGUMENT | Invalid input parameter. |

### ◆ RM_BLE_ABS_Close()

fsp_err_t RM_BLE_ABS_Close ( ble_abs_ctrl_t *const *p_ctrl*)

Close the BLE channel. Implements ble_abs_api_t::close.

Example:

```
/* Close BLE driver */
   err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel closed successfully. |
| FSP_ERR_ASSERTION | Null pointer presented. |
| FSP_ERR_NOT_OPEN | Control block not open. |

### ◆ RM_BLE_ABS_Reset()

fsp_err_t RM_BLE_ABS_Reset ( ble_abs_ctrl_t *const *p_ctrl*, ble_event_cb_t *init_callback* )

BLE is reset with this function. The process is carried out in the following order. R_BLE_Close() -> R_BLE_GAP_Terminate() -> R_BLE_Open() -> R_BLE_SetEvent(). The init_cb callback initializes the others (Host Stack, timer, etc...). Implements ble_abs_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel closed successfully. |
| FSP_ERR_ASSERTION | Null pointer presented. |
| FSP_ERR_NOT_OPEN | Control block not open. |

### ◆ RM_BLE_ABS_VersionGet()

fsp_err_t RM_BLE_ABS_VersionGet ( fsp_version_t *const *p_version*)

Get BLE module code and API versions. Implements ble_abs_api_t::versionGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_version is NULL. |

### ◆ RM_BLE_ABS_StartLegacyAdvertising()

fsp_err_t RM_BLE_ABS_StartLegacyAdvertising ( ble_abs_ctrl_t *const *p_ctrl*, ble_abs_legacy_advertising_parameter_t const *const *p_advertising_parameter* )

Start Legacy Advertising after setting advertising parameters, advertising data and scan response data. The legacy advertising uses the advertising set whose advertising handle is 0. The advertising type is connectable and scannable(ADV_IND). The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Implements ble_abs_api_t::startLegacyAdvertising

Example:

```
/* Start advertising. */
            err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded |
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_STATE | Host stack hasn't been initialized. |
| FSP_ERR_INVALID_POINTER | p_advertising_parameter is specified as NULL. |
| FSP_ERR_INVALID_ARGUMENT | The advertising parameter is out of range. |

### ◆ RM_BLE_ABS_StartExtendedAdvertising()

fsp_err_t RM_BLE_ABS_StartExtendedAdvertising ( ble_abs_ctrl_t *const *p_ctrl*, ble_abs_extend_advertising_parameter_t const *const *p_advertising_parameter* )

Start Extended Advertising after setting advertising parameters, advertising data. The extended advertising uses the advertising set whose advertising handle is 1. The advertising type is connectable and non-scannable. The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Implements ble_abs_api_t::startExtendedAdvertising

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_POINTER | p_advertising_parameter is specified as NULL. |
| FSP_ERR_INVALID_STATE | Host stack hasn't been initialized. |
| FSP_ERR_INVALID_ARGUMENT | The advertising parameter is out of range. |
| FSP_ERR_UNSUPPORTED | Subordinate modules do not support this feature. |

### ◆ RM_BLE_ABS_StartNonConnectableAdvertising()

fsp_err_t RM_BLE_ABS_StartNonConnectableAdvertising ( ble_abs_ctrl_t *const *p_ctrl*, ble_abs_non_connectable_advertising_parameter_t const *const *p_advertising_parameter* )

Start Non-Connectable Advertising after setting advertising parameters, advertising data. The non-connectable advertising uses the advertising set whose advertising handle is 2. The advertising type is non-connectable and non-scannable. The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Secondary Advertising Max Skip is 0. Implements ble_abs_api_t::startNonConnectableAdvertising.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_POINTER | p_advertising_parameter is specified as NULL. |
| FSP_ERR_INVALID_STATE | Host stack hasn't been initialized. |
| FSP_ERR_INVALID_ARGUMENT | The advertising parameter is out of range. |

### ◆ RM_BLE_ABS_StartPeriodicAdvertising()

fsp_err_t RM_BLE_ABS_StartPeriodicAdvertising ( ble_abs_ctrl_t *const *p_ctrl*, ble_abs_periodic_advertising_parameter_t const *const *p_advertising_parameter* )

Start Periodic Advertising after setting advertising parameters, periodic advertising parameters, advertising data and periodic advertising data. The periodic advertising uses the advertising set whose advertising handle is 3. The advertising type is non-connectable and non-scannable. The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Secondary Advertising Max Skip is 0. Implements ble_abs_api_t::startPeriodicAdvertising

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_POINTER | p_advertising_parameter is specified as NULL. |
| FSP_ERR_INVALID_ARGUMENT | The advertising parameter is out of range. |
| FSP_ERR_UNSUPPORTED | Subordinate modules do not support this feature. |

### ◆ RM_BLE_ABS_StartScanning()

fsp_err_t RM_BLE_ABS_StartScanning ( ble_abs_ctrl_t *const *p_ctrl*, ble_abs_scan_parameter_t const *const *p_scan_parameter* )

Start scanning after setting scan parameters. The scanner address type is Public Identity Address. Fast scan is followed by slow scan. The end of fast scan or slow scan is notified with BLE_GAP_EVENT_SCAN_TO event. If fast_period is 0, only slow scan is carried out. If scan_period is 0, slow scan continues. Implements ble_abs_api_t::startScanning.

Example:

```
/* Start scanning. */
            err = RM_BLE_ABS_StartScanning(&g_ble_abs0_ctrl, &scan_parameter);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_POINTER | p_scan_parameter is specified as NULL. |
| FSP_ERR_INVALID_ARGUMENT | The scan parameter is out of range. |

### ◆ RM_BLE_ABS_CreateConnection()

fsp_err_t RM_BLE_ABS_CreateConnection ( ble_abs_ctrl_t *const *p_ctrl*, ble_abs_connection_parameter_t const *const *p_connection_parameter* )

Request create connection. The initiator address type is Public Identity Address. The scan interval is 60ms and the scan window is 30ms in case of 1M PHY or 2M PHY. The scan interval is 180ms and the scan window is 90ms in case of coded PHY. The Minimum CE Length and the Maximum CE Length are 0xFFFF. When the request for a connection has been received by the Controller, BLE_GAP_EVENT_CREATE_CONN_COMP event is notified. When a link has beens established, BLE_GAP_EVENT_CONN_IND event is notified. Implements ble_abs_api_t::createConnection.

Example:

```
/* Create connection with remote device. */

            err = RM_BLE_ABS_CreateConnection(&g_ble_abs0_ctrl,

&connection_parameter);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_POINTER | p_connection_parameter is specified as NULL. |
| FSP_ERR_INVALID_ARGUMENT | The create connection parameter is out of range. |
| FSP_ERR_BLE_ABS_NOT_FOUND | Couldn't find a valid timer. |
| FSP_ERR_BLE_ABS_INVALID_OPERATION | Invalid operation for the selected timer. |

### ◆ RM_BLE_ABS_SetLocalPrivacy()

fsp_err_t RM_BLE_ABS_SetLocalPrivacy ( ble_abs_ctrl_t *const *p_ctrl*, uint8_t const *const *p_lc_irk*, uint8_t *privacy_mode* )

Generate a IRK, add it to the resolving list, set privacy mode and enable RPA function. Register vendor specific callback function, if IRK is generated by this function. After configuring local device privacy, BLE_GAP_ADDR_RPA_ID_PUBLIC is specified as own device address in theadvertising/scan/create connection API. Implements ble_abs_api_t::setLocalPrivacy

Example:

```
/* Set local privacy. */
            err = RM_BLE_ABS_SetLocalPrivacy(&g_ble_abs0_ctrl, p_lc_irk,
privacy_mode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_ARGUMENT | The privacy_mode parameter is out of range. |
| FSP_ERR_BLE_ABS_INVALID_OPERATION | Host stack hasn't been initialized. configuring the resolving list or privacy mode. |

### ◆ RM_BLE_ABS_StartAuthentication()

fsp_err_t RM_BLE_ABS_StartAuthentication ( ble_abs_ctrl_t *const *p_ctrl*, uint16_t *connection_handle* )

Start pairing or encryption. If pairing has been done, start encryption. The pairing parameters are configured by RM_BLE_ABS_Open() or R_BLE_GAP_SetPairingParams(). If the pairing parameters are configure by RM_BLE_ABS_Open(),

- bonding policy is that bonding information is stored.
- Key press notification is not supported. Implements ble_abs_api_t::startAuthentication.

Example:

```
/* Start authentication with remote device. */

RM_BLE_ABS_StartAuthentication(&g_ble_abs0_ctrl, connection_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | p_instance_ctrl or connection_handle are specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_ARGUMENT | The connection handle parameter is out of range. |

## 4.2.57 SD/MMC Block Media Implementation (rm_block_media_sdmmc)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_Open (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg) |
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_MediaInit (rm_block_media_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_Read (rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks) |
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_Write (rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks) |

| | |
|---|---|
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_Erase (rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks) |
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_StatusGet (rm_block_media_ctrl_t *const p_api_ctrl, rm_block_media_status_t *const p_status) |
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_InfoGet (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info) |
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_Close (rm_block_media_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Middleware to implement the block media interface on SD cards. This module implements the Block Media Interface.

# Overview

### Features

The SD/MMC implementation of the block media interface has the following key features:

- Reading, writing, and erasing data from an SD card
- Callback called when card insertion or removal is detected
- Provides media information such as sector size and total number of sectors.

# Configuration

### Build Time Configurations for rm_block_media_sdmmc

The following build time configurations are defined in driver/rm_block_media_sdmmc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Middleware > Storage > Block Media Implementation on rm_block_media_sdmmc

This module can be added to the Stacks tab via New Stack > Middleware > Storage > Block Media Implementation on rm_block_media_sdmmc:

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rm_block_media0 | Module name. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called when a card is inserted or removed. |

### Clock Configuration

This module has no required clock configurations.

### Pin Configuration

This module does not use I/O pins.

## Examples

### Basic Example

This is a basic example of minimal use of the SD/MMC block media implementation in an application.

```
#define RM_BLOCK_MEDIA_SDMMC_BLOCK_SIZE (512)
uint8_t g_dest[RM_BLOCK_MEDIA_SDMMC_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint8_t g_src[RM_BLOCK_MEDIA_SDMMC_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint32_t g_transfer_complete = 0;
void rm_block_media_sdmmc_basic_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < RM_BLOCK_MEDIA_SDMMC_BLOCK_SIZE; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the RM_BLOCK_MEDIA_SDMMC driver. */
 fsp_err_t err = RM_BLOCK_MEDIA_SDMMC_Open(&g_rm_block_media0_ctrl,
&g_rm_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* A device shall be ready to accept the first command within 1ms from detecting VDD
min. Reference section 6.4.1.1
```

```
  * "Power Up Time of Card" in the SD Physical Layer Simplified Specification Version
6.00. */
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
 /* Initialize the SD card. This should not be done until the card is plugged in for
SD devices. */
    err = RM_BLOCK_MEDIA_SDMMC_MediaInit(&g_rm_block_media0_ctrl);
    handle_error(err);
 /* Write a block of data to sector 3 of an SD card. */
    err = RM_BLOCK_MEDIA_SDMMC_Write(&g_rm_block_media0_ctrl, g_src, 3, 1);
    handle_error(err);
 /* Read a block of data from sector 3 of an SD card. */
    err = RM_BLOCK_MEDIA_SDMMC_Read(&g_rm_block_media0_ctrl, g_dest, 3, 1);
    handle_error(err);
}
```

## Function Documentation

### ◆ RM_BLOCK_MEDIA_SDMMC_Open()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Open ( rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_cfg_t const *const *p_cfg* )

Opens the module.

Implements rm_block_media_api_t::open().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is available and is now open. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_ALREADY_OPEN | Module has already been opened with this instance of the control structure. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::open

### ◆ RM_BLOCK_MEDIA_SDMMC_MediaInit()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_MediaInit ( rm_block_media_ctrl_t *const *p_ctrl*)

Initializes the SD or eMMC device. This procedure requires several sequential commands. This function blocks until all identification and configuration commands are complete.

Implements rm_block_media_api_t::mediaInit().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is initialized and ready to access the memory device. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::mediaInit

### ◆ RM_BLOCK_MEDIA_SDMMC_Read()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Read ( rm_block_media_ctrl_t *const *p_ctrl*, uint8_t *const *p_dest_address*, uint32_t const *block_address*, uint32_t const *num_blocks* )

Reads data from an SD or eMMC device. Up to 0x10000 sectors can be read at a time. Implements rm_block_media_api_t::read().

This function blocks until the data is read into the destination buffer.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::read

### ◆ RM_BLOCK_MEDIA_SDMMC_Write()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Write ( rm_block_media_ctrl_t *const *p_ctrl*, uint8_t const *const *p_src_address*, uint32_t const *block_address*, uint32_t const *num_blocks* )

Writes data to an SD or eMMC device. Up to 0x10000 sectors can be written at a time. Implements rm_block_media_api_t::write().

This function blocks until the write operation completes.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write finished successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::write

### ◆ RM_BLOCK_MEDIA_SDMMC_Erase()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Erase ( rm_block_media_ctrl_t *const *p_ctrl*, uint32_t const *block_address*, uint32_t const *num_blocks* )

Erases sectors of an SD card or eMMC device. Implements rm_block_media_api_t::erase().

This function blocks until erase is complete.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Erase operation requested. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::erase
- sdmmc_api_t::statusGet

### ◆ RM_BLOCK_MEDIA_SDMMC_StatusGet()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_StatusGet ( rm_block_media_ctrl_t *const *p_api_ctrl*, rm_block_media_status_t *const *p_status* )

Provides driver status. Implements rm_block_media_api_t::statusGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Status stored in p_status. |
| FSP_ERR_ASSERTION | NULL pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLOCK_MEDIA_SDMMC_InfoGet()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_InfoGet ( rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_info_t *const *p_info* )

Retrieves module information. Implements rm_block_media_api_t::infoGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Erase operation requested. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

### ◆ RM_BLOCK_MEDIA_SDMMC_Close()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Close ( rm_block_media_ctrl_t *const *p_ctrl*)

Closes an open SD/MMC device. Implements rm_block_media_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLOCK_MEDIA_SDMMC_VersionGet()

| fsp_err_t RM_BLOCK_MEDIA_SDMMC_VersionGet ( fsp_version_t *const *p_version*) |
|---|
| Returns the version of the firmware and API. Implements rm_block_media_api_t::versionGet(). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed successfully. |
| FSP_ERR_ASSERTION | Null Pointer. |

## 4.2.58 USB HMSC Block Media Implementation (rm_block_media_usb)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | RM_BLOCK_MEDIA_USB_Open (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg) |
| fsp_err_t | RM_BLOCK_MEDIA_USB_MediaInit (rm_block_media_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLOCK_MEDIA_USB_Read (rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks) |
| fsp_err_t | RM_BLOCK_MEDIA_USB_Write (rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks) |
| fsp_err_t | RM_BLOCK_MEDIA_USB_Erase (rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks) |
| fsp_err_t | RM_BLOCK_MEDIA_USB_StatusGet (rm_block_media_ctrl_t *const p_api_ctrl, rm_block_media_status_t *const p_status) |
| fsp_err_t | RM_BLOCK_MEDIA_USB_InfoGet (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info) |
| fsp_err_t | RM_BLOCK_MEDIA_USB_Close (rm_block_media_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLOCK_MEDIA_USB_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Middleware to implement the block media interface on USB mass storage devices. This module implements the Block Media Interface.

# Overview

### Features

The USB implementation of the block media interface has the following key features:

- Reading, writing, and erasing data from a USB mass storage device
- Callback called when device insertion or removal is detected
- Provides media information such as sector size and total number of sectors.

# Configuration

### Build Time Configurations for rm_block_media_usb

The following build time configurations are defined in driver/rm_block_media_usb_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking Enable | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Middleware > Storage > Block Media Implementation on rm_block_media_usb

This module can be added to the Stacks tab via New Stack > Middleware > Storage > Block Media Implementation on rm_block_media_usb:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rm_block_media0 | Module name. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called when a device is attached or removed. |
| Pointer to user context | Name must be a valid C symbol | NULL | A user context can be provided. If this context is provided, it will be passed to callback function when a device is attached or removed. |

*Note*

*RM_BLOCK_MEDIA_USB_MediaInit function must be called after receiving the insert event notification.*

### Clock Configuration

This module has no required clock configurations.

### Pin Configuration

This module does not use I/O pins.

## Examples

### Basic Example

This is a basic example of minimal use of the USB mass storage block media implementation in an application.

```
#define RM_BLOCK_MEDIA_USB_BLOCK_SIZE (512)
volatile bool g_usb_inserted = false;
uint8_t       g_dest[RM_BLOCK_MEDIA_USB_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint8_t       g_src[RM_BLOCK_MEDIA_USB_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
void rm_block_media_usb_basic_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < RM_BLOCK_MEDIA_USB_BLOCK_SIZE; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the RM_BLOCK_MEDIA_USB driver. */
 fsp_err_t err = RM_BLOCK_MEDIA_USB_Open(&g_rm_block_media0_ctrl,
&g_rm_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (!g_usb_inserted)
    {
 /* Wait for a card insertion interrupt. */
    }
 /* Initialize the mass storage device. This should not be done until the device is
plugged in and initialized. */
```

```
    err = RM_BLOCK_MEDIA_USB_MediaInit(&g_rm_block_media0_ctrl);
    handle_error(err);
 /* Write a block of data to sector 3 of an USB mass storage device. */
    err = RM_BLOCK_MEDIA_USB_Write(&g_rm_block_media0_ctrl, g_src, 3, 1);
    handle_error(err);
 /* Read a block of data from sector 3 of an USB mass storage device. */
    err = RM_BLOCK_MEDIA_USB_Read(&g_rm_block_media0_ctrl, g_dest, 3, 1);
    handle_error(err);
}
```

### Device Insertion

This is an example of using a callback to determine when a mass storage device is plugged in and enumerated.

```
/* The callback is called when a media insertion event occurs. */
void rm_block_media_usb_media_insertion_example_callback
(rm_block_media_callback_args_t * p_args)
{
 if (RM_BLOCK_MEDIA_EVENT_MEDIA_INSERTED == p_args->event)
    {
       g_usb_inserted = true;
    }
 if (RM_BLOCK_MEDIA_EVENT_MEDIA_REMOVED == p_args->event)
    {
       g_usb_inserted = false;
    }
}
```

## Function Documentation

### ◆ RM_BLOCK_MEDIA_USB_Open()

fsp_err_t RM_BLOCK_MEDIA_USB_Open ( rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_cfg_t const *const *p_cfg* )

Opens the module.

Implements rm_block_media_api_t::open().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is available and is now open. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_ALREADY_OPEN | Module has already been opened with this instance of the control structure. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_BLOCK_MEDIA_USB_MediaInit()

fsp_err_t RM_BLOCK_MEDIA_USB_MediaInit ( rm_block_media_ctrl_t *const *p_ctrl*)

Initializes the USB device.

Implements rm_block_media_api_t::mediaInit().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is initialized and ready to access the memory device. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLOCK_MEDIA_USB_Read()

fsp_err_t RM_BLOCK_MEDIA_USB_Read ( rm_block_media_ctrl_t *const *p_ctrl*, uint8_t *const *p_dest_address*, uint32_t const *block_address*, uint32_t const *num_blocks* )

Reads data from an USB device. Implements rm_block_media_api_t::read().

This function blocks until the data is read into the destination buffer.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_BLOCK_MEDIA_USB_Write()

fsp_err_t RM_BLOCK_MEDIA_USB_Write ( rm_block_media_ctrl_t *const *p_ctrl*, uint8_t const *const *p_src_address*, uint32_t const *block_address*, uint32_t const *num_blocks* )

Writes data to an USB device. Implements rm_block_media_api_t::write().

This function blocks until the write operation completes.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write finished successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_BLOCK_MEDIA_USB_Erase()

fsp_err_t RM_BLOCK_MEDIA_USB_Erase ( rm_block_media_ctrl_t *const *p_ctrl*, uint32_t const *block_address*, uint32_t const *num_blocks* )

Erases sectors of an USB device. Implements rm_block_media_api_t::erase().

This function blocks until erase is complete.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Erase operation requested. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_BLOCK_MEDIA_USB_StatusGet()

fsp_err_t RM_BLOCK_MEDIA_USB_StatusGet ( rm_block_media_ctrl_t *const *p_api_ctrl*, rm_block_media_status_t *const *p_status* )

Provides driver status. Implements rm_block_media_api_t::statusGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Status stored in p_status. |
| FSP_ERR_ASSERTION | NULL pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_BLOCK_MEDIA_USB_InfoGet()

fsp_err_t RM_BLOCK_MEDIA_USB_InfoGet ( rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_info_t *const *p_info* )

Retrieves module information. Implements rm_block_media_api_t::infoGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Erase operation requested. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

### ◆ RM_BLOCK_MEDIA_USB_Close()

fsp_err_t RM_BLOCK_MEDIA_USB_Close ( rm_block_media_ctrl_t *const *p_ctrl*)

Closes an open USB device. Implements rm_block_media_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLOCK_MEDIA_USB_VersionGet()

fsp_err_t RM_BLOCK_MEDIA_USB_VersionGet ( fsp_version_t *const *p_version*)

Returns the version of the firmware and API. Implements rm_block_media_api_t::versionGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed successfully. |
| FSP_ERR_ASSERTION | Null Pointer. |

## 4.2.59 SEGGER emWin Port (rm_emwin_port)

Modules

SEGGER emWin port for RA MCUs.

# Overview

The SEGGER emWin RA Port module provides the configuration and hardware acceleration support necessary for use of emWin on RA products. The port provides full integration with the graphics peripherals (GLCDC, DRW and JPEG) as well as FreeRTOS.

*Note*

*This port layer primarily enables hardware acceleration and background handling of many display operations and does not contain code intended to be directly called by the user. Please consult the SEGGER emWin User Guide (UM03001) for details on how to use emWin in your project.*

### Hardware Acceleration

The following functions are currently performed with hardware acceleration:

- DRW Engine (r_drw)
  - Drawing bitmaps (ARGB8888 and RGB565)
  - Rectangle fill
  - Line and shape drawing
  - Anti-aliased operations
    - Circle stroke and fill
    - Polygon stroke and fill
    - Lines and arcs
- JPEG Codec (r_jpeg)
  - JPEG decoding
- Graphics LCD Controller (r_glcdc)
  - Brightness, contrast and gamma correction
  - Pixel format conversion (framebuffer to LCD)

# Configuration

### Build Time Configurations for rm_emwin_port

The following build time configurations are defined in fsp_cfg/rm_emwin_port_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Memory Allocation > GUI Heap Size | Value must be a non-negative integer | 32768 | Set the size of the heap to be allocated for use exclusively by emWin. |
| Memory Allocation > Section for GUI Heap | Manual Entry | .noinit | Specify the section in which to allocate the GUI heap. |
| Memory Allocation > Maximum Layers | Value must be a non-negative integer | 16 | Set the maximum number of available display layers in emWin. |

| | | | |
|---|---|---|---|
| | | | This setting is not related to GLCDC Layer 1 or 2. |
| Memory Allocation > AA Font Conversion Buffer Size | Value must be a non-negative integer | 400 | Set the size of the conversion buffer for anti-aliased font glyphs. This should be set to the size (in bytes) of the largest AA character to be rendered. |
| Configuration > Multi-thread Support | • Enabled<br>• Disabled | Enabled | Enable or disable multithreading support. |
| Configuration > Number of emWin-supported threads | Manual Entry | 5 | If multithreading support is enabled this configuration specifies the number of different tasks that can call emWin functions. |
| Configuration > Touch Panel Support | • Enabled<br>• Disabled | Enabled | Enable or disable touch panel support. |
| Configuration > Mouse Support | • Enabled<br>• Disabled | Disabled | Enable or disable support for mouse input. |
| Configuration > Memory Devices | • Enabled<br>• Disabled | Enabled | Enable or disable support for memory devices, which allow the user to allocate their own memory in the GUI heap. |
| Configuration > Text Rotation | • Enabled<br>• Disabled | Disabled | Enable or disable support for displaying rotated text. |
| Configuration > Window Manager | • Enabled<br>• Disabled | Enabled | Enable or disable the emWin Window Manager (WM). |
| Configuration > Bidirectional Text | • Enabled<br>• Disabled | Disabled | Enable or disable support for bidirectional text (such as Arabic or Hebrew). |
| Configuration > Debug Logging Level | • None (0)<br>• Parameter checking only (1)<br>• All checks enabled (2)<br>• Log errors (3)<br>• Log warnings | All checks enabled (2) | Set the debug logging level. |

| | | | |
|---|---|---|---|
| | (4)<br>• Log all messages (5) | | |
| LCD Settings > Wait for Vertical Sync | • Enabled<br>• Disabled | Enabled | When enabled emWin will wait for a vertical sync event each time the display is updated. If an RTOS is used the thread will yield; otherwise each frame will block until Vsync.<br><br>WARNING: Disabling vertical sync will result in tearing. It is recommended to always leave this setting Enabled if an RTOS is used. |
| JPEG Decoding > General > Input Alignment | • 8-byte aligned (faster)<br>• Unaligned (slower) | Unaligned (slower) | Setting this option to 8-bit alignment can allow the hardware JPEG Codec to directly read JPEG data. This speeds up JPEG decoding operations and reduces RAM overhead, but all JPEG images must reside on an 8-byte boundary.<br><br>When this option is enabled the input buffer is not allocated. |
| JPEG Decoding > General > Double-Buffer Output | • Enabled<br>• Disabled | Disabled | Enable this option to configure JPEG decoding operations to use a double-buffered output pipeline. This allows the JPEG to be rendered to the display at the same time as decoding at the cost of additional RAM usage.<br><br>Enabling this option automatically allocates double the output buffer size. |
| JPEG Decoding > General > Error Timeout | Value must be a non-negative integer | 50 | Set the timeout for JPEG decoding operations (in RTOS |

| | | | |
|---|---|---|---|
| | | | ticks) in the event of a decode error. |
| JPEG Decoding > Buffers > Input Buffer Size | Value must be a non-negative integer | 0x1000 | Set the size of the JPEG decode input buffer (in bytes). This buffer is used to ensure 8-byte alignment of input data. Specifying a size smaller than the size of the JPEG to decode will use additional interrupts to stream data in during the decoding process. |
| JPEG Decoding > Buffers > Output Buffer Size | Value must be a non-negative integer | 0x3C00 | Set the size of the JPEG decode output buffer (in bytes). An output buffer smaller than the size of a decoded image will use additional interrupts to stream the data into a framebuffer.<br><br>Unless you are sure of the subsampling used in and the size of the input JPEG images it is recommended to allocate at least 16 framebuffer lines of memory. |
| JPEG Decoding > Buffers > Section for Buffers | Manual Entry | .noinit | Specify the section in which to allocate the JPEG work buffers. |

### Hardware Configuration

No clocks or pins are directly required by this module. Please consult the submodules' documentation for their requirements.

## Usage Notes

### Getting Started

To get started with emWin in an RA project the following must be performed:

1. Open the RA Configuration editor for your project
2. Add emWin to your project in the Stacks view by clicking **New Stack -> SEGGER -> SEGGER emWin**
3. Ensure the configuration options for emWin are set as necessary for your application
4. Set the proporties for the GLCDC module to match the timing and memory requirements of your display panel

5. Set the JPEG decode color depth to the desired value (if applicable)
6. Ensure interrupts on all modules are enabled:
    - GLCDC Vertical Position (Vpos) Interrupt
    - DRW Interrupt (if applicable)
    - JPEG Encode/Decode and Data Transfer Interrupts (if applicable)
7. Click Generate Project Content to save and commit configuration changes
8. (Non-RTOS projects only) Before calling GUI_Init, call g_hal_init to initialize the framebuffer address.

At this point the project is now ready to build with emWin. Please refer to the SEGGER emWin User Guide (UM03001) as well as demo and sample code for details on how to create a GUI application.

### Using Hardware Acceleration

In most cases there is no need to perform additional configuration to ensure the DRW Engine is used. However, there are some guidelines that should be followed depending on the item in question:

- Bitmaps:
    - ARGB8888, RGB888 and RGB565 bitmaps require no additional settings.
- Anti-aliased shapes:
    - Anti-aliased lines, circles, polygons, polygon outlines and arcs are rendered with the DRW Engine.
- Anti-aliased (4bpp) fonts:
    - Set the text mode to GUI_TM_TRANS or create the relevant widget with WM_CF_HASTRANS set.
    - Ensure the "AA Font Conversion Buffer Size" configuration option is set to a size equal to or greater than the size (in bytes) of the largest glyph.
- 8bpp palletized images:
    - When creating these images ensure transparency is not enabled as the SEGGER method for this is not compatible with the DRW Engine.
- RLE-encoded images:
    - Hardware acceleration is not available for SEGGER's RLE format.

### Multi-thread Support

When the "Multi-thread Support" configuration is enabled, emWin can be called from multiple threads. This comes with advantages and disadvantages:

Advantages:

- High flexibility in development of applications
- Threads can pend and post on emWin events

Disadvantages:

- Slightly higher RAM/ROM use
- Large GUI projects can become difficult to debug

*Note*

*Multi-thread support is independent of RTOS support. RTOS support is managed internally and cannot be manually configured.*

### Limitations

Developers should be aware of the following limitations when using SEGGER emWin with FSP:

- Hardware acceleration is not available when using color modes lower than 16 bits.
- Support for rotated screen modes is in development.
- Hardware acceleration is not available for SEGGER's RLE image format.

# Examples

### Basic Example

This is a basic example demonstrating a very simple emWin application. The screen is cleared to white and "Hello World!" is printed in the center.

*Note*

*emWin manages the GLCDC, DRW and JPEG Codec submodules internally; they do not need to be opened directly.*

```
#include "DIALOG.h"
#define COLOR_WHITE 0x00FFFFFFU
#define COLOR_BLACK 0x00000000U
#define GUI_DRAW_DELAY 100
static void _cbMain (WM_MESSAGE * pMsg)
{
    GUI_RECT Rect;
 switch (pMsg->MsgId)
    {
 case WM_CREATE:
      {
 break;
       }
 case WM_PAINT:
      {
 /* Clear background to white */
            GUI_SetBkColor(COLOR_WHITE);
            GUI_Clear();
 /* Draw "Hello World!" in black in the center */
            WM_GetClientRect(&Rect);
            GUI_SetColor(COLOR_BLACK);
            GUI_DispStringInRect("Hello World!", &Rect, GUI_TA_VCENTER |
GUI_TA_HCENTER);
 break;
```

```
      }
 default:
     {
           WM_DefaultProc(pMsg);
 break;
      }
   }
}
void emWinTask (void)
{
   int32_t xSize;
   int32_t ySize;
 /* Initialize emWin */
   GUI_Init();
 /* Get screen dimensions */
   xSize = LCD_GetXSize();
   ySize = LCD_GetYSize();
 /* Create main window */
   WM_CreateWindowAsChild(0, 0, xSize, ySize, WM_HBKWIN, WM_CF_SHOW, _cbMain, 0);
 /* Enter main drawing loop */
 while (1)
   {
      GUI_Delay(GUI_DRAW_DELAY);
   }
}
```

*Note*

*For further example code please consult SEGGER emWin documentation, which can be downloaded here, as well as the Quick Start Guide and example project(s) provided with your Evaluation Kit (if applicable).*

## 4.2.60 FreeRTOS+FAT Port (rm_freertos_plus_fat)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_FREERTOS_PLUS_FAT_Open (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_cfg_t const *const p_cfg) |
| fsp_err_t | RM_FREERTOS_PLUS_FAT_MediaInit (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_device_t *const p_device) |
| fsp_err_t | RM_FREERTOS_PLUS_FAT_DiskInit (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_disk_cfg_t const *const p_disk_cfg, FF_Disk_t *const p_disk) |
| fsp_err_t | RM_FREERTOS_PLUS_FAT_DiskDeinit (rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk) |
| fsp_err_t | RM_FREERTOS_PLUS_FAT_InfoGet (rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk, rm_freertos_plus_fat_info_t *const p_info) |
| fsp_err_t | RM_FREERTOS_PLUS_FAT_Close (rm_freertos_plus_fat_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_FREERTOS_PLUS_FAT_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

Middleware for the FAT File System control on RA MCUs.

# Overview

This module provides the hardware port layer for FreeRTOS+FAT file system. After initializing this module, refer to the FreeRTOS+FAT API reference to use the file system: https://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_FAT/index.html

### Features

The FreeRTOS+FAT port module supports the following features:

- Callbacks for insertion and removal for removable devices.
- Helper function to initialize FF_Disk_t
- Blocking read and write port functions that use FreeRTOS task notification to pend if FreeRTOS is used
- FreeRTOS is optional

# Configuration

### Build Time Configurations for rm_freertos_plus_fat

The following build time configurations are defined in fsp_cfg/middleware/rm_freertos_plus_fat_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking Enable | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Configurations for FreeRTOS+ > FreeRTOS+FAT Port for RA**

This module can be added to the Stacks tab via New Stack > FreeRTOS+ > FreeRTOS+FAT Port for RA:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rm_freertos_plus_fat0 | Module name. |
| Total Number of Sectors | Must be a non-negative integer | 31293440 | Enter the total number of sectors on the device. If this is not known, update rm_freertos_plus_fat_disk_cfg_t::num_blocks after calling RM_FREERTOS_PLUS_FAT_MediaInit(). |
| Sector Size (bytes) | Must be a power of 2 multiple of 512 | 512 | Select the sector size. Must match the underlying media sector size and at least 512. If this is not known, update rm_freertos_plus_fat_disk_cfg_t::num_blocks after calling RM_FREERTOS_PLUS_FAT_MediaInit(). |
| Cache Size (bytes) | Must be a power of 2 multiple of 512 | 1024 | Select the cache size. Must be a multiple of the sector size and at least 2 times the sector size. |
| Partition Number | Must be a non-negative integer | 0 | Select the partition number for this disk. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called when a card is inserted or removed. |

# Usage Notes

### Pending during Read/Write

If the underlying driver supports non-blocking operations, the FreeRTOS+FAT port pends the active FreeRTOS task during read and write operations so other tasks can run in the background.

If FreeRTOS is not used, the FreeRTOS+FAT port spins in a while loop waiting for read and write operations to complete.

### FreeRTOS+FAT without FreeRTOS

To use FreeRTOS+FAT without FreeRTOS, copy FreeRTOSConfigMinimal.h to one of your project's include paths and rename it FreeRTOSConfig.h.

Also, update the Malloc function to malloc and the Free function to free in the Common configurations.

## Examples

### Basic Example

This is a basic example of FreeRTOS+FAT in an application.

```
#define RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME "TEST_FILE.txt"
#define RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES (10240)
#define RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER (0)
extern rm_freertos_plus_fat_instance_ctrl_t g_freertos_plus_fat0_ctrl;
extern const rm_freertos_plus_fat_cfg_t g_freertos_plus_fat0_cfg;
extern const rm_freertos_plus_fat_disk_cfg_t g_rm_freertos_plus_fat_disk_cfg;
extern uint8_t g_file_data[RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES];
extern uint8_t g_read_buffer[RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES];
void rm_freertos_plus_fat_example (void)
{
 /* Open media driver.*/
 fsp_err_t err = RM_FREERTOS_PLUS_FAT_Open(&g_freertos_plus_fat0_ctrl,
&g_freertos_plus_fat0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
 rm_freertos_plus_fat_device_t device;
    err = RM_FREERTOS_PLUS_FAT_MediaInit(&g_freertos_plus_fat0_ctrl, &device);
```

```
    handle_error(err);
 /* Initialize one disk for each partition used in the application. */
    FF_Disk_t disk;
    err = RM_FREERTOS_PLUS_FAT_DiskInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg, &disk);
    handle_error(err);
 /* Mount each disk. This assumes the disk is already partitioned and formatted. */
    FF_Error_t ff_err = FF_Mount(&disk,
RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
    handle_ff_error(ff_err);
 /* Add the disk to the file system. */
    FF_FS_Add("/", &disk);
 /* Open a source file for writing. */
    FF_FILE * pxSourceFile = ff_fopen((const char *)
RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME, "w");
    assert(NULL != pxSourceFile);
 /* Write file data. */
 size_t size_return = ff_fwrite(g_file_data, sizeof(g_file_data), 1, pxSourceFile);
    assert(1 == size_return);
 /* Close the file. */
 int close_err = ff_fclose(pxSourceFile);
    assert(0 == close_err);
 /* Open the source file in read mode. */
    pxSourceFile = ff_fopen((const char *) RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME,
"r");
    assert(NULL != pxSourceFile);
 /* Read file data. */
    size_return = ff_fread(g_read_buffer, sizeof(g_file_data), 1, pxSourceFile);
    assert(1 == size_return);
 /* Close the file. */
    close_err = ff_fclose(pxSourceFile);
    assert(0 == close_err);
 /* Verify the file data read matches the file written. */
    assert(0U == memcmp(g_file_data, g_read_buffer, sizeof(g_file_data)));
```

```
}
```

### Format Example

This shows how to partition and format a disk if it is not already partitioned and formatted.

```
void rm_freertos_plus_fat_format_example (void)
{
 /* Open media driver.*/
 fsp_err_t err = RM_FREERTOS_PLUS_FAT_Open(&g_freertos_plus_fat0_ctrl,
&g_freertos_plus_fat0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
 rm_freertos_plus_fat_device_t device;
    err = RM_FREERTOS_PLUS_FAT_MediaInit(&g_freertos_plus_fat0_ctrl, &device);
    handle_error(err);
 /* Initialize one disk for each partition used in the application. */
    FF_Disk_t disk;
    err = RM_FREERTOS_PLUS_FAT_DiskInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg, &disk);
    handle_error(err);
 /* Try to mount the disk. If the disk is not formatted, mount will fail. */
    FF_Error_t ff_err = FF_Mount(&disk,
RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
 if (FF_isERR((uint32_t) ff_err))
    {
 /* The disk is likely not formatted. Partition and format the disk, then mount
again. */
       FF_PartitionParameters_t partition_params;
       partition_params.ulSectorCount   = device.sector_count;
       partition_params.ulHiddenSectors = 1;
       partition_params.ulInterSpace    = 0;
```

```
       memset(partition_params.xSizes, 0, sizeof(partition_params.xSizes));
       partition_params.xSizes[RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER] =
            (BaseType_t) partition_params.ulSectorCount - 1;
       partition_params.xPrimaryCount = 1;
       partition_params.eSizeType    = eSizeIsSectors;
       ff_err = FF_Partition(&disk, &partition_params);
      handle_ff_error(ff_err);
       ff_err = FF_Format(&disk, RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER,
pdFALSE, pdFALSE);
      handle_ff_error(ff_err);
       ff_err = FF_Mount(&disk, RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
      handle_ff_error(ff_err);
    }
}
```

### Media Insertion Example

This shows how to use the callback to wait for media insertion.

```
#if 2 == BSP_CFG_RTOS
static EventGroupHandle_t xUSBEventGroupHandle = NULL;
#else
volatile uint32_t g_rm_freertos_plus_fat_insertion_events = 0;
volatile uint32_t g_rm_freertos_plus_fat_removal_events = 0;
#endif
/* Callback called by media driver when a removable device is inserted or removed. */
void rm_freertos_plus_fat_test_callback (rm_freertos_plus_fat_callback_args_t *
p_args)
{
#if 2 == BSP_CFG_RTOS
 /* Post an event if FreeRTOS is available. */
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    xEventGroupSetBitsFromISR(xUSBEventGroupHandle, p_args->event,
&xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
```

```
#else
 /* If FreeRTOS is not used, set a global flag. */
 if (p_args->event & RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED)
    {
       g_rm_freertos_plus_fat_insertion_events++;
    }
 if (p_args->event & RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_REMOVED)
    {
       g_rm_freertos_plus_fat_removal_events++;
    }
#endif
}
void rm_freertos_plus_fat_media_insertion_example (void)
{
#if 2 == BSP_CFG_RTOS
 /* Create event flags if FreeRTOS is used. */
    xUSBEventGroupHandle = xEventGroupCreate();
    TEST_ASSERT_NOT_EQUAL(NULL, xUSBEventGroupHandle);
#endif
 /* Open media driver.*/
 fsp_err_t err = RM_FREERTOS_PLUS_FAT_Open(&g_freertos_plus_fat0_ctrl,
&g_freertos_plus_fat0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Wait for media insertion. */
#if 2 == BSP_CFG_RTOS
    EventBits_t xEventGroupValue = xEventGroupWaitBits(xUSBEventGroupHandle,
 RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED,
                                                       pdTRUE,
                                                       pdFALSE,
                                                       portMAX_DELAY);
    assert(RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED ==
(RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED & xEventGroupValue));
#else
```

```
 while (0U == g_rm_freertos_plus_fat_insertion_events)
    {
 /* Wait for media insertion. */
    }
#endif
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
 rm_freertos_plus_fat_device_t device;
    err = RM_FREERTOS_PLUS_FAT_MediaInit(&g_freertos_plus_fat0_ctrl, &device);
    handle_error(err);
 /* Initialize one disk for each partition used in the application. */
    FF_Disk_t disk;
    err = RM_FREERTOS_PLUS_FAT_DiskInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg, &disk);
    handle_error(err);
}
```

## Data Structures

| | |
|---|---|
| struct | rm_freertos_plus_fat_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_freertos_plus_fat_instance_ctrl_t

| struct rm_freertos_plus_fat_instance_ctrl_t |
|---|
| FreeRTOS plus FAT private control block. DO NOT MODIFY. Initialization occurs when RM_FREERTOS_PLUS_FAT_Open is called. |

## Function Documentation

### ◆ RM_FREERTOS_PLUS_FAT_Open()

fsp_err_t RM_FREERTOS_PLUS_FAT_Open ( rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, rm_freertos_plus_fat_cfg_t const *const *p_cfg* )

Initializes lower layer media device.

Implements rm_freertos_plus_fat_api_t::open().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_OUT_OF_MEMORY | Not enough memory to create semaphore. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- rm_block_media_api_t::open

### ◆ RM_FREERTOS_PLUS_FAT_MediaInit()

fsp_err_t RM_FREERTOS_PLUS_FAT_MediaInit ( rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, rm_freertos_plus_fat_device_t *const *p_device* )

Initializes the media device. This function blocks until all identification and configuration commands are complete.

Implements rm_freertos_plus_fat_api_t::mediaInit().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is initialized and ready to access the memory device. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module has not been initialized. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- rm_block_media_api_t::mediaInit
- rm_block_media_api_t::infoGet

### ◆ RM_FREERTOS_PLUS_FAT_DiskInit()

fsp_err_t RM_FREERTOS_PLUS_FAT_DiskInit ( rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, rm_freertos_plus_fat_disk_cfg_t const *const *p_disk_cfg*, FF_Disk_t *const *p_disk* )

Initializes a FreeRTOS+FAT disk structure. This function calls FF_CreateIOManger.

Implements rm_freertos_plus_fat_api_t::diskInit().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is initialized and ready to access the memory device. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module has not been initialized. |
| FSP_ERR_INTERNAL | Call to FF_CreateIOManger failed. |

### ◆ RM_FREERTOS_PLUS_FAT_DiskDeinit()

fsp_err_t RM_FREERTOS_PLUS_FAT_DiskDeinit ( rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, FF_Disk_t *const *p_disk* )

Deinitializes a FreeRTOS+FAT disk structure. This function calls FF_DeleteIOManger.

Implements rm_freertos_plus_fat_api_t::diskDeinit().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is initialized and ready to access the memory device. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module has not been initialized. |

### ◆ RM_FREERTOS_PLUS_FAT_InfoGet()

fsp_err_t RM_FREERTOS_PLUS_FAT_InfoGet ( rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, FF_Disk_t *const *p_disk*, rm_freertos_plus_fat_info_t *const *p_info* )

Get partition information. This function can only be called after rm_freertos_plus_fat_api_t::diskInit() .

Implements rm_freertos_plus_fat_api_t::infoGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in p_info. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |

### ◆ RM_FREERTOS_PLUS_FAT_Close()

fsp_err_t RM_FREERTOS_PLUS_FAT_Close ( rm_freertos_plus_fat_ctrl_t *const *p_ctrl*)

Closes media device.

Implements rm_freertos_plus_fat_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Media device closed. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- rm_block_media_api_t::close

### ◆ RM_FREERTOS_PLUS_FAT_VersionGet()

fsp_err_t RM_FREERTOS_PLUS_FAT_VersionGet ( fsp_version_t *const *p_version*)

Returns the version of this module.

Implements rm_freertos_plus_fat_api_t::versionGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_ASSERTION | Failed in acquiring version information. |

### 4.2.61 FreeRTOS Plus TCP (rm_freertos_plus_tcp)

Modules

Middleware for using TCP on RA MCUs.

# Overview

FreeRTOS Plus TCP is a TCP stack created for use with FreeRTOS.

This module provides the NetworkInterface required to use FreeRTOS Plus TCP with the Ethernet (r_ether) driver.

Please refer to the FreeRTOS Plus TCP documentation for further details.

# Configuration

#### Build Time Configurations for FreeRTOS_Plus_TCP

The following build time configurations are defined in aws/FreeRTOSIPConfig.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Print debug messages | • Disable<br>• Enable | Disable | If ipconfigHAS_DEBUG_PRINTF is set to 1 then FreeRTOS_debug_printf should be defined to the function used to print out the debugging messages. |
| Print info messages | • Disable<br>• Enable | Disable | Set to 1 to print out non debugging messages, for example the output of the FreeRTOS_netstat() command, and ping replies. If ipconfigHAS_PRINTF is set to 1 then FreeRTOS_printf should be set to the function used to print out the messages. |
| Byte order of the target MCU | pdFREERTOS_LITTLE_ENDIAN | pdFREERTOS_LITTLE_ENDIAN | Define the byte order of the target MCU |

| | | | |
|---|---|---|---|
| IP/TCP/UDP checksums | • Disable<br>• Enable | Enable | If the network card/driver includes checksum offloading (IP/TCP/UDP checksums) then set ipconfigDRIVER_INCLUDED_RX_IP_CHECKSUM to 1 to prevent the software stack repeating the checksum calculations. |
| Receive Block Time | Value must be an integer | 10000 | Amount of time FreeRTOS_recv() will block for. The timeouts can be set per socket, using setsockopt(). |
| Send Block Time | Value must be an integer | 10000 | Amount of time FreeRTOS_send() will block for. The timeouts can be set per socket, using setsockopt(). |
| DNS caching | • Disable<br>• Enable | Enable | DNS caching |
| DNS Request Attempts | Value must be an integer | 2 | When a cache is present, ipconfigDNS_REQUEST_ATTEMPTS can be kept low and also DNS may use small timeouts. |
| IP stack task priority | Manual Entry | configMAX_PRIORITIES - 2 | Set the priority of the task that executes the IP stack. |
| Stack size in words (not bytes) | Manual Entry | configMINIMAL_STACK_SIZE * 5 | The size, in words (not bytes), of the stack allocated to the FreeRTOS+TCP stack. |
| Network Events call vApplicationIPNetworkEventHook | • Disable<br>• Enable | Enable | vApplicationIPNetworkEventHook is called when the network connects or disconnects. |
| Max UDP send block time | Manual Entry | 15000 / portTICK_PERIOD_MS | Max UDP send block time |
| Use DHCP | • Disable<br>• Enable | Enable | If ipconfigUSE_DHCP is 1 then FreeRTOS+TCP will attempt to retrieve an IP address, netmask, DNS server address and gateway |

| | | | |
|---|---|---|---|
| | | | address from a DHCP server. |
| DHCP Register Hostname | • Disable<br>• Enable | Enable | Register hostname when using DHCP |
| DHCP Uses Unicast | • Disable<br>• Enable | Enable | DHCP uses unicast. |
| DHCP Send Discover After Auto IP | • Disable<br>• Enable | Disable | DHCP Send Discover After Auto IP |
| DHCP callback function | • Disable<br>• Enable | Disable | Provide an implementation of the DHCP callback function (xApplicationDHCPHook ) |
| Interval between transmissions | Manual Entry | 120000 / portTICK_PERIOD_MS | When ipconfigUSE_DHCP is set to 1, DHCP requests will be sent out at increasing time intervals until either a reply is received from a DHCP server and accepted, or the interval between transmissions reaches i pconfigMAXIMUM_DISC OVER_TX_PERIOD. |
| ARP Cache Entries | Value must be an integer | 6 | The maximum number of entries that can exist in the ARP table at any one time |
| ARP Request Retransmissions | Value must be an integer | 5 | ARP requests that do not result in an ARP response will be re-transmitted a maximum of ipconfigM AX_ARP_RETRANSMISSI ONS times before the ARP request is aborted. |
| Maximum time before ARP table entry becomes stale | Value must be an integer | 150 | The maximum time between an entry in the ARP table being created or refreshed and the entry being removed because it is stale |
| Use string for IP Address | • Disable<br>• Enable | Enable | Take an IP in decimal dot format (for example, "192.168.0.1") as its |

| | | | |
|---|---|---|---|
| | | | parameter FreeRTOS_inet_addr_quick() takes an IP address as four separate numerical octets (for example, 192, 168, 0, 1) as its parameters |
| Total number of avaiable network buffers | Value must be an integer | 10 | Define the total number of network buffer that are available to the IP stack |
| Set the maximum number of events | Please enter a valid function name without spaces or funny characters | ipconfigNUM_NETWORK_BUFFER_DESCRIPTORS + 5 | Set the maximum number of events that can be queued for processing at any one time. The event queue must be a minimum of 5 greater than the total number of network buffers |
| Enable FreeRTOS_sendto() without calling Bind | • Enable<br>• Disable | Disable | Set to 1 then calling FreeRTOS_sendto() on a socket that has not yet been bound will result in the IP stack automatically binding the socket to a port number from the range socketAUTO_PORT_ALLOCATION_START_NUMBER to 0xffff. If ipconfigALLOW_SOCKET_SEND_WITHOUT_BIND is set to 0 then calling FreeRTOS_sendto() on a socket that has not yet been bound will result in the send operation being aborted. |
| TTL values for UDP packets | Value must be an integer | 128 | Define the Time To Live (TTL) values used in outgoing UDP packets |
| TTL values for TCP packets | Value must be an integer | 128 | Defines the Time To Live (TTL) values used in outgoing TCP packets |
| Use TCP and all its features | • Disable<br>• Enable | Enable | Use TCP and all its features |
| Let TCP use windowing | • Disable | Disable | Let TCP use windowing |

| | | | |
|---|---|---|---|
| mechanism | • Enable | | mechanism |
| Maximum number of bytes the payload of a network frame can contain | Value must be an integer | 1500 | Maximum number of bytes the payload of a network frame can contain |
| Basic DNS client or resolver | • Disable<br>• Enable | Enable | Set ipconfigUSE_DNS to 1 to include a basic DNS client/resolver. DNS is used through the FreeRTOS_gethostbyname() API function. |
| Reply to incoming ICMP echo (ping) requests | • Disable<br>• Enable | Enable | If ipconfigREPLY_TO_INCOMING_PINGS is set to 1 then the IP stack will generate replies to incoming ICMP echo (ping) requests. |
| FreeRTOS_SendPingRequest() is available | • Disable<br>• Enable | Disable | If ipconfigSUPPORT_OUTGOING_PINGS is set to 1 then the FreeRTOS_SendPingRequest() API function is available. |
| FreeRTOS_select() (and associated) API function is available | • Disable<br>• Enable | Disable | If ipconfigSUPPORT_SELECT_FUNCTION is set to 1 then the FreeRTOS_select() (and associated) API function is available |
| Filter out non Ethernet II frames. | • Disable<br>• Enable | Enable | If ipconfigFILTER_OUT_NON_ETHERNET_II_FRAMES is set to 1 then Ethernet frames that are not in Ethernet II format will be dropped. This option is included for potential future IP stack developments |
| Responsibility of the Ethernet interface to filter out packets | • Disable<br>• Enable | Disable | If ipconfigETHERNET_DRIVER_FILTERS_FRAME_TYPES is set to 1 then it is the responsibility of the Ethernet interface to filter out packets that are of no interest. |
| Block time to simulate MAC interrupts | Please enter a valid function name without spaces or funny characters | 20 / portTICK_PERIOD_MS | The windows simulator cannot really simulate MAC interrupts, and needs to block occasionally to allow other tasks to run |

| | | | |
|---|---|---|---|
| Access 32-bit fields in the IP packets | Value must be an integer | 2 | To access 32-bit fields in the IP packets with 32-bit memory instructions, all packets will be stored 32-bit-aligned, plus 16-bits. This has to do with the contents of the IP-packets: all 32-bit fields are 32-bit-aligned, plus 16-bit |
| Size of the pool of TCP window descriptors | Value must be an integer | 240 | Define the size of the pool of TCP window descriptors |
| Size of Rx buffer for TCP sockets | Value must be an integer | 3000 | Define the size of Rx buffer for TCP sockets |
| Size of Tx buffer for TCP sockets | Value must be an integer | 3000 | Define the size of Tx buffer for TCP sockets |
| TCP keep-alive | • Disable<br>• Enable | Enable | TCP keep-alive is avaiable or not |
| TCP keep-alive interval | Value must be an integer | 120 | TCP keep-alive interval in second |
| The socket semaphore to unblock the MQTT task (USER_SEMAPHORE) | • Disable<br>• Enable | Disable | The socket semaphore is used to unblock the MQTT task |
| The socket semaphore to unblock the MQTT task (WAKE_CALLBACK) | • Disable<br>• Enable | Enable | The socket semaphore is used to unblock the MQTT task |
| The socket semaphore to unblock the MQTT task (USE_CALLBACKS) | • Disable<br>• Enable | Disable | The socket semaphore is used to unblock the MQTT task |
| The socket semaphore to unblock the MQTT task (TX_DRIVER) | • Disable<br>• Enable | Disable | The socket semaphore is used to unblock the MQTT task |
| The socket semaphore to unblock the MQTT task (RX_DRIVER) | • Disable<br>• Enable | Disable | The socket semaphore is used to unblock the MQTT task |
| Possible optimisation for expert users | • Disable<br>• Enable | Disable | Possible optimisation for expert users - requires network driver support. It is is useful when there is high network traffic. If non-zero value then instead of passing received packets into the IP task one at a time the |

network interface can chain received packets together and pass them into the IP task in one go. If set to 0 then only one buffer will be sent at a time.

# Usage Notes

In order to use the NetworkInterface implementation provided by Renesas for RA devices:

- Configure an r_ether instance and provide a pointer to the instance of the NetworkInterface as follows:

```
/* Reference used by the NetworkInterface to access the ethernet instance. */
extern ether_instance_t const * gp_freertos_ether;
...
/* Make it reference the configured ether instance. */
ether_instance_t const * gp_freertos_ether = &g_ether_instance;
```

- Follow the TCP stack initialization procedure as described here: FreeRTOS+TCP Networking Tutorial: Initializing the TCP/IP Stack

*Note*

*The MAC address passed to FreeRTOS_IPInit must match the MAC address configured in the r_ether instance.*
*g_ether_instance must have vEtherISRCallback configured as the callback.*
*The xApplicationGetRandomNumber and ulApplicationGetNextSequenceNumber functions should be implemented in systems using FreeRTOS Plus TCP without Secure Sockets.*
*To connect to a server using an IP address the macro ipconfigINCLUDE_FULL_INET_ADDR must be set to 1.*

### Limitations

- Zero-copy is not currently supported by the NetworkInterface.

## 4.2.62 FreeRTOS Port (rm_freertos_port)

Modules

FreeRTOS port for RA MCUs.

# Overview

*Note*

***The FreeRTOS Port does not provide any interfaces to the user. Consult the FreeRTOS documentation at https://www.freertos.org/Documentation for further information.***

### Features

The RA FreeRTOS port supports the following features:

- Standard FreeRTOS configurations
- Hardware stack monitor

# Configuration

### Build Time Configurations for all

The following build time configurations are defined in aws/FreeRTOSConfig.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General > Custom FreeRTOSConfig.h | Manual Entry | | Add a path to your custom FreeRTOSConfig.h file. It can be used to override some or all of the configurations defined here, and to define additional configurations. |
| General > Use Preemption | • Enabled<br>• Disabled | Enabled | Set to Enabled to use the preemptive RTOS scheduler, or Disabled to use the cooperative RTOS scheduler. |
| General > Use Port Optimised Task Selection | • Enabled<br>• Disabled | Disabled | Some FreeRTOS ports have two methods of selecting the next task to execute - a generic method, and a method that is specific to that port.<br><br>The Generic method:<br>Is used when Use Port Optimized Task Selection is set to 0, or when a port specific method is not implemented.<br>Can be used with all FreeRTOS ports.<br>Is completely written in C, making it less |

| | | | |
|---|---|---|---|
| | | | efficient than a port specific method.<br>Does not impose a limit on the maximum number of available priorities.<br><br>A port specific method:<br>Is not available for all ports.<br>Is used when Use Port Optimized Task Selection is Enabled.<br>Relies on one or more architecture specific assembly instructions (typically a Count Leading Zeros [CLZ] or equivalent instruction) so can only be used with the architecture for which it was specifically written.<br>Is more efficient than the generic method.<br>Typically imposes a limit of 32 on the maximum number of available priorities. |
| General > Use Tickless Idle | • Enabled<br>• Disabled | Disabled | Set Use Tickless Idle to Enabled to use the low power tickless mode, or Disabled to keep the tick interrupt running at all times. Low power tickless implementations are not provided for all FreeRTOS ports. |
| General > Cpu Clock Hz | Manual Entry | SystemCoreClock | Enter the frequency in Hz at which the internal clock that drives the peripheral used to generate the tick interrupt will be executing - this is normally the same clock that drives the internal CPU clock. This value is required in order to correctly configure timer peripherals. |

| | | | |
|---|---|---|---|
| General > Tick Rate Hz | Must be an integer and greater than 0 | 1000 | The frequency of the RTOS tick interrupt. The tick interrupt is used to measure time. Therefore a higher tick frequency means time can be measured to a higher resolution. However, a high tick frequency also means that the RTOS kernel will use more CPU time so be less efficient. The RTOS demo applications all use a tick rate of 1000Hz. This is used to test the RTOS kernel and is higher than would normally be required.<br><br>More than one task can share the same priority. The RTOS scheduler will share processor time between tasks of the same priority by switching between the tasks during each RTOS tick. A high tick rate frequency will therefore also have the effect of reducing the 'time slice' given to each task. |
| General > Max Priorities | Must be an integer and greater than 0 | 5 | The number of priorities available to the application tasks. Any number of tasks can share the same priority.<br>Each available priority consumes RAM within the RTOS kernel so this value should not be set any higher than actually required by your application. |
| General > Minimal Stack Size | Must be an integer and greater than 0 | 128 | The size of the stack used by the idle task. Generally this should not be reduced from the value set in the |

| | | | |
|---|---|---|---|
| | | | FreeRTOSConfig.h file provided with the demo application for the port you are using. Like the stack size parameter to the xTaskCreate() and xTaskCreateStatic() functions, the stack size is specified in words, not bytes. If each item placed on the stack is 32-bits, then a stack size of 100 means 400 bytes (each 32-bit stack item consuming 4 bytes). |
| General > Max Task Name Len | Must be an integer and greater than 0 | 16 | The maximum permissible length of the descriptive name given to a task when the task is created. The length is specified in the number of characters including the NULL termination byte. |
| General > Use 16-bit Ticks | Disabled | Disabled | Time is measured in 'ticks' - which is the number of times the tick interrupt has executed since the RTOS kernel was started. The tick count is held in a variable of type TickType_t. Defining configUSE_16_BIT_TICKS as 1 causes TickType_t to be defined (typedef'ed) as an unsigned 16bit type. Defining configUSE_16_BIT_TICKS as 0 causes TickType_t to be defined (typedef'ed) as an unsigned 32bit type.<br><br>Using a 16-bit type will greatly improve performance on 8- and 16-bit architectures, but limits the |

| | | | |
|---|---|---|---|
| | | | maximum specifiable time period to 65535 'ticks'. Therefore, assuming a tick frequency of 250Hz, the maximum time a task can delay or block when a 16bit counter is used is 262 seconds, compared to 17179869 seconds when using a 32-bit counter. |
| General > Idle Should Yield | • Enabled<br>• Disabled | Enabled | This parameter controls the behaviour of tasks at the idle priority. It only has an effect if:<br>The preemptive scheduler is being used.<br>The application creates tasks that run at the idle priority.<br>If Use Time Slicing is Enabled then tasks that share the same priority will time slice. If none of the tasks get preempted then it might be assumed that each task at a given priority will be allocated an equal amount of processing time - and if the priority is above the idle priority then this is indeed the case.<br>When tasks share the idle priority the behaviour can be slightly different. If Idle Should Yield is Enabled then the idle task will yield immediately if any other task at the idle priority is ready to run. This ensures the minimum amount of time is spent in the idle task when application tasks are available for scheduling. This behaviour can however |

have undesirable effects (depending on the needs of your application) as depicted below:

The diagram above shows the execution pattern of four tasks that are all running at the idle priority. Tasks A, B and C are application tasks. Task I is the idle task. A context switch occurs with regular period at times T0, T1, ..., T6. When the idle task yields task A starts to execute - but the idle task has already consumed some of the current time slice. This results in task I and task A effectively sharing the same time slice. The application tasks B and C therefore get more processing time than the application task A.

This situation can be avoided by:

If appropriate, using an idle hook in place of separate tasks at the idle priority.
Creating all application tasks at a priority greater than the idle priority.
Setting Idle Should Yield to Disabled.
Setting Idle Should Yield to Disabled prevents the idle task from yielding processing time until the end of its time slice. This ensure all tasks at the idle priority are allocated

| | | | |
|---|---|---|---|
| | | | an equal amount of processing time (if none of the tasks get pre-empted) - but at the cost of a greater proportion of the total processing time being allocated to the idle task. |
| General > Use Task Notifications | • Enabled<br>• Disabled | Enabled | Setting Use Task Notifications to Enabled will include direct to task notification functionality and its associated API in the build.<br>Setting Use Task Notifications to Disabled will exclude direct to task notification functionality and its associated API from the build.<br><br>Each task consumes 8 additional bytes of RAM when direct to task notifications are included in the build. |
| General > Use Mutexes | • Enabled<br>• Disabled | Disabled | Set to Enabled to include mutex functionality in the build, or Disabled to omit mutex functionality from the build. Readers should familiarise themselves with the differences between mutexes and binary semaphores in relation to the FreeRTOS functionality. |
| General > Use Recursive Mutexes | • Enabled<br>• Disabled | Disabled | Set to Enabled to include recursive mutex functionality in the build, or Disabled to omit recursive mutex functionality from the build. |
| General > Use Counting Semaphores | • Enabled<br>• Disabled | Enabled | Set to Enabled to include counting |

| | | | |
|---|---|---|---|
| | | | semaphore functionality in the build, or Disabled to omit counting semaphore functionality from the build. |
| General > Queue Registry Size | Must be an integer and greater than 0 | 10 | The queue registry has two purposes, both of which are associated with RTOS kernel aware debugging:<br>It allows a textual name to be associated with a queue for easy queue identification within a debugging GUI.<br>It contains the information required by a debugger to locate each registered queue and semaphore.<br>The queue registry has no purpose unless you are using a RTOS kernel aware debugger. Registry Size defines the maximum number of queues and semaphores that can be registered. Only those queues and semaphores that you want to view using a RTOS kernel aware debugger need be registered. See the API reference documentation for vQueueAddToRegistry() and vQueueUnregisterQueue() for more information. |
| General > Use Queue Sets | • Enabled<br>• Disabled | Disabled | Set to Enabled to include queue set functionality (the ability to block, or pend, on multiple queues and semaphores), or Disabled to omit queue set functionality. |

| | | | |
|---|---|---|---|
| General > Use Time Slicing | • Enabled<br>• Disabled | Disabled | If Use Time Slicing is Enabled, FreeRTOS uses prioritised preemptive scheduling with time slicing. That means the RTOS scheduler will always run the highest priority task that is in the Ready state, and will switch between tasks of equal priority on every RTOS tick interrupt. If Use Time Slicing is Disabled then the RTOS scheduler will still run the highest priority task that is in the Ready state, but will not switch between tasks of equal priority just because a tick interrupt has occurred. |
| General > Use Newlib Reentrant | • Enabled<br>• Disabled | Disabled | If Use Newlib Reentrant is Enabled then a newlib reent structure will be allocated for each created task. Note Newlib support has been included by popular demand, but is not used by the FreeRTOS maintainers themselves. FreeRTOS is not responsible for resulting newlib operation. User must be familiar with newlib and must provide system-wide implementations of the necessary stubs. Be warned that (at the time of writing) the current newlib design implements a system-wide malloc() that must be provided with locks. |
| General > Enable Backward Compatibility | • Enabled<br>• Disabled | Disabled | The FreeRTOS.h header file includes a set of #define macros that map the names of data types used in versions |

| | | | |
|---|---|---|---|
| | | | of FreeRTOS prior to version 8.0.0 to the names used in FreeRTOS version 8.0.0. The macros allow application code to update the version of FreeRTOS they are built against from a pre 8.0.0 version to a post 8.0.0 version without modification. Setting Enable Backward Compatibility to Disabled in FreeRTOSConfig.h excludes the macros from the build, and in so doing allowing validation that no pre version 8.0.0 names are being used. |
| General > Num Thread Local Storage Pointers | Must be an integer and greater than 0 | 5 | Sets the number of indexes in each task's thread local storage array. |
| General > Stack Depth Type | Manual Entry | uint32_t | Sets the type used to specify the stack depth in calls to xTaskCreate(), and various other places stack sizes are used (for example, when returning the stack high water mark). Older versions of FreeRTOS specified stack sizes using variables of type UBaseType_t, but that was found to be too restrictive on 8-bit microcontrollers. Stack Depth Type removes that restriction by enabling application developers to specify the type to use. |
| General > Message Buffer Length Type | Manual Entry | size_t | FreeRTOS Message buffers use variables of type Message Buffer Length Type to store the length of each |

| | | |
|---|---|---|
| | | message. If Message Buffer Length Type is not defined then it will default to size_t. If the messages stored in a message buffer will never be larger than 255 bytes then defining Message Buffer Length Type to uint8_t will save 3 bytes per message on a 32-bit microcontroller. Likewise if the messages stored in a message buffer will never be larger than 65535 bytes then defining Message Buffer Length Type to uint16_t will save 2 bytes per message on a 32-bit microcontroller. |
| General > Library Max Syscall Interrupt Priority | MCU Specific Options | The highest interrupt priority that can be used by any interrupt service routine that makes calls to interrupt safe FreeRTOS API functions. DO NOT CALL INTERRUPT SAFE FREERTOS API FUNCTIONS FROM ANY INTERRUPT THAT HAS A HIGHER PRIORITY THAN THIS! (higher priorities are lower numeric values)<br><br>Below is explanation for macros that are set based on this value from FreeRTOS website.<br><br>In the RA port, configKERNEL_INTERRUPT_PRIORITY is not used and the kernel runs at the lowest priority.<br><br>Note in the following discussion that only API |

functions that end in "FromISR" can be called from within an interrupt service routine.

configMAX_SYSCALL_INTERRUPT_PRIORITY sets the highest interrupt priority from which interrupt safe FreeRTOS API functions can be called.

A full interrupt nesting model is achieved by setting configMAX_SYSCALL_INTERRUPT_PRIORITY above (that is, at a higher priority level) than configKERNEL_INTERRUPT_PRIORITY. This means the FreeRTOS kernel does not completely disable interrupts, even inside critical sections. Further, this is achieved without the disadvantages of a segmented kernel architecture.

Interrupts that do not call API functions can execute at priorities above configMAX_SYSCALL_INTERRUPT_PRIORITY and therefore never be delayed by the RTOS kernel execution.

A special note for ARM Cortex-M users: Please read the page dedicated to interrupt priority settings on ARM Cortex-M devices. As a minimum, remember that ARM Cortex-M cores use numerically low priority numbers to represent HIGH priority interrupts, which can

| | | | |
|---|---|---|---|
| | | | seem counter-intuitive and is easy to forget! If you wish to assign an interrupt a low priority do NOT assign it a priority of 0 (or other low numeric value) as this can result in the interrupt actually having the highest priority in the system - and therefore potentially make your system crash if this priority is above config MAX_SYSCALL_INTERRUPT_PRIORITY.<br><br>The lowest priority on a ARM Cortex-M core is in fact 255 - however different ARM Cortex-M vendors implement a different number of priority bits and supply library functions that expect priorities to be specified in different ways. For example, on the RA6M3 the lowest priority you can specify is 15 - and the highest priority you can specify is 0. |
| General > Assert | Manual Entry | assert ( x ) | The semantics of the configASSERT() macro are the same as the standard C assert() macro. An assertion is triggered if the parameter passed into configASSERT() is zero. configASSERT() is called throughout the FreeRTOS source files to check how the application is using FreeRTOS. It is highly recommended to develop FreeRTOS applications with configASSERT() defined. |

The example definition (shown at the top of the file and replicated below) calls vAssertCalled(), passing in the file name and line number of the triggering configASSERT() call (__FILE__ and __LINE__ are standard macros provided by most compilers). This is just for demonstration as vAssertCalled() is not a FreeRTOS function, configASSERT() can be defined to take whatever action the application writer deems appropriate.

It is normal to define configASSERT() in such a way that it will prevent the application from executing any further. This if for two reasons; stopping the application at the point of the assertion allows the cause of the assertion to be debugged, and executing past a triggered assertion will probably result in a crash anyway.

Note defining configASSERT() will increase both the application code size and execution time. When the application is stable the additional overhead can be removed by simply commenting out the configASSERT() definition in FreeRTOSConfig.h.

/* Define configASSERT() to call

| | | | |
|---|---|---|---|
| | | | vAssertCalled() if the assertion fails. The assertion has failed if the value of the parameter passed into configASSERT() equals zero. */ #define configASSERT( ( x ) ) if( ( x ) == 0 ) vAssertCalled( __FILE__, __LINE__ ) If running FreeRTOS under the control of a debugger, then configASSERT() can be defined to just disable interrupts and sit in a loop, as demonstrated below. That will have the effect of stopping the code on the line that failed the assert test - pausing the debugger will then immediately take you to the offending line so you can see why it failed.<br><br>/* Define configASSERT() to disable interrupts and sit in a loop. */ #define configASSERT( ( x ) ) if( ( x ) == 0 ) { taskDISABLE_INTERRUPTS(); for( ;; ); } |
| General > Include Application Defined Privileged Functions | • Enabled<br>• Disabled | Disabled | Include Application Defined Privileged Functions is only used by FreeRTOS MPU. If Include Application Defined Privileged Functions is Enabled then the application writer must provide a header file called "application_defined_privileged_functions.h", in which functions the application writer needs to execute in privileged mode can be |

implemented. Note that, despite having a .h extension, the header file should contain the implementation of the C functions, not just the functions' prototypes.

Functions implemented in "application_defined_privileged_functions.h" must save and restore the processor's privilege state using the prvRaisePrivilege() function and portRESET_PRIVILEGE() macro respectively. For example, if a library provided print function accesses RAM that is outside of the control of the application writer, and therefore cannot be allocated to a memory protected user mode task, then the print function can be encapsulated in a privileged function using the following code:

```
void MPU_debug_printf(
const char *pcMessage
)
{
/* State the privilege
level of the processor
when the function was
called. */
BaseType_t
xRunningPrivileged =
prvRaisePrivilege();

/* Call the library
function, which now
has access to all RAM.
*/
debug_printf(
pcMessage );

/* Reset the processor
```

| | | | |
|---|---|---|---|
| | | | privilege level to its original value. */ portRESET_PRIVILEGE( xRunningPrivileged ); } This technique should only be use during development, and not deployment, as it circumvents the memory protection. |
| Hooks > Use Idle Hook | • Enabled<br>• Disabled | Enabled | Set to Enabled if you wish to use an idle hook, or Disabled to omit an idle hook. |
| Hooks > Use Malloc Failed Hook | • Enabled<br>• Disabled | Disabled | The kernel uses a call to pvPortMalloc() to allocate memory from the heap each time a task, queue or semaphore is created. The official FreeRTOS download includes four sample memory allocation schemes for this purpose. The schemes are implemented in the heap_1.c, heap_2.c, heap_3.c, heap_4.c and heap_5.c source files respectively. Use Malloc Failed Hook is only relevant when one of these three sample schemes is being used. The malloc() failed hook function is a hook (or callback) function that, if defined and configured, will be called if pvPortMalloc() ever returns NULL. NULL will be returned only if there is insufficient FreeRTOS heap memory remaining for the requested allocation to succeed.<br><br>If Use Malloc Failed Hook is Enabled then the application must |

| | | | |
|---|---|---|---|
| | | | define a malloc() failed hook function. If Use Malloc Failed Hook is set to Dosab;ed then the malloc() failed hook function will not be called, even if one is defined. Malloc() failed hook functions must have the name and prototype shown below.<br><br>void vApplicationMalloc FailedHook( void ); |
| Hooks > Use Daemon Task Startup Hook | • Enabled<br>• Disabled | Disabled | If Use Timers and Use Daemon Task Startup Hook are both Enabled then the application must define a hook function that has the exact name and prototype as shown below. The hook function will be called exactly once when the RTOS daemon task (also known as the timer service task) executes for the first time. Any application initialisation code that needs the RTOS to be running can be placed in the hook function. void void vApplicationD aemonTaskStartupHoo k( void ); |
| Hooks > Use Tick Hook | • Enabled<br>• Disabled | Disabled | Set to Enabled if you wish to use an tick hook, or Disabled to omit an tick hook. |
| Hooks > Check For Stack Overflow | • Enabled<br>• Disabled | Disabled | The stack overflow detection page describes the use of this parameter. This is not recommended for RA MCUs with hardware stack monitor support. RA MCU designs should enable the RA hardware stack monitor instead. |

| | | | |
|---|---|---|---|
| Stats > Use Trace Facility | • Enabled<br>• Disabled | Disabled | Set to Enabled if you wish to include additional structure members and functions to assist with execution visualisation and tracing. |
| Stats > Use Stats Formatting Functions | • Enabled<br>• Disabled | Disabled | Set Use Trace Facility and Use Stats Formatting Functions to Enabled to include the vTaskList() and vTaskGetRunTimeStats() functions in the build. Setting either to Disabled will omit vTaskList() and vTaskGetRunTimeStates() from the build. |
| Stats > Generate Run Time Stats | • Enabled<br>• Disabled | Disabled | The Run Time Stats page describes the use of this parameter. |
| Memory Allocation > Support Static Allocation | • Enabled<br>• Disabled | Enabled | If Support Static Allocation is Enabled then RTOS objects can be created using RAM provided by the application writer.<br>If Support Static Allocation is Disabled then RTOS objects can only be created using RAM allocated from the FreeRTOS heap.<br><br>If Support Static Allocation is left undefined it will default to 0.<br><br>If Support Static Allocation is Enabled then the application writer must also provide two callback functions: vApplicationGetIdleTaskMemory() to provide the memory for use by the RTOS Idle task, and (if Use Timers is Enabled) vApplicationGetTimerTaskMemory() to provide |

memory for use by the RTOS Daemon/Timer Service task. Examples are provided below.

/* Support Static Allocation is Enabled, so the application must provide an implementation of vApplicationGetIdleTaskMemory() to provide the memory that is used by the Idle task. */
void vApplicationGetIdleTaskMemory( StaticTask_t **ppxIdleTaskTCBBuffer,<br> StackType_t **ppxIdleTaskStackBuffer,<br> uint32_t *pulIdleTaskStackSize )
{
/* If the buffers to be provided to the Idle task are declared inside this function then they must be declared static - otherwise they will be allocated on the stack and so not exists after this function exits. */
static StaticTask_t xIdleTaskTCB;
static StackType_t uxIdleTaskStack[ configMINIMAL_STACK_SIZE ];

/* Pass out a pointer to the StaticTask_t structure in which the Idle task's state will be stored. */
*ppxIdleTaskTCBBuffer =

/* Pass out the array that will be used as the Idle task's stack. */
*ppxIdleTaskStackBuffer = uxIdleTaskStack;

/* Pass out the size of the array pointed to by *ppxIdleTaskStackBuffer.
Note that, as the array is necessarily of type StackType_t, configMINIMAL_STACK_SIZE is specified in words, not bytes. */
*pulIdleTaskStackSize = configMINIMAL_STACK_SIZE;
}
/*----------*/

/* Support Static Allocation and Use Timers are both Enabled, so the application must provide an implementation of vApplicationGetTimerTaskMemory()
to provide the memory that is used by the Timer service task. */
void vApplicationGetTimerTaskMemory(
StaticTask_t **ppxTimerTaskTCBBuffer,<br>
StackType_t **ppxTimerTaskStackBuffer,<br>
uint32_t *pulTimerTaskStackSize )
{
/* If the buffers to be provided to the Timer task are declared inside this
function then they must be declared static - otherwise they will be allocated on
the stack and so not exists after this function exits. */
static StaticTask_t xTimerTaskTCB;
static StackType_t uxTimerTaskStack[ configTIMER_TASK_STACK_

| | | | |
|---|---|---|---|
| | | | DEPTH ];<br><br>/* Pass out a pointer to the StaticTask_t structure in which the Timer task's state will be stored. */ *ppxTimerTaskTCBBuffer =<br><br>/* Pass out the array that will be used as the Timer task's stack. */ *ppxTimerTaskStackBuffer = uxTimerTaskStack;<br><br>/* Pass out the size of the array pointed to by *ppxTimerTaskStackBuffer.<br>Note that, as the array is necessarily of type StackType_t, configTIMER_TASK_STACK_DEPTH is specified in words, not bytes. */ *pulTimerTaskStackSize = configTIMER_TASK_STACK_DEPTH;<br>}<br><br>Examples of the callback functions that must be provided by the application to supply the RAM used by the Idle and Timer Service tasks if Support Static Allocation is Enabled.<br><br>See the Static Vs Dynamic Memory Allocation page for more information. |
| Memory Allocation > Support Dynamic Allocation | • Enabled<br>• Disabled | Disabled | If Support Dynamic Allocation is Enabled then RTOS objects can be created using RAM that is automatically allocated from the FreeRTOS heap. |

| | | | |
|---|---|---|---|
| | | | If Support Dynamic Allocation is set to 0 then RTOS objects can only be created using RAM provided by the application writer.<br><br>See the Static Vs Dynamic Memory Allocation page for more information. |
| Memory Allocation > Total Heap Size | Must be an integer and greater than 0 | 1024 | The total amount of RAM available in the FreeRTOS heap.<br>This value will only be used if Support Dynamic Allocation is Enabled and the application makes use of one of the sample memory allocation schemes provided in the FreeRTOS source code download. See the memory configuration section for further details. |
| Memory Allocation > Application Allocated Heap | • Enabled<br>• Disabled | Disabled | By default the FreeRTOS heap is declared by FreeRTOS and placed in memory by the linker. Setting Application Allocated Heap to Enabled allows the heap to instead be declared by the application writer, which allows the application writer to place the heap wherever they like in memory.<br>If heap_1.c, heap_2.c or heap_4.c is used, and Application Allocated Heap is Enabled, then the application writer must provide a uint8_t array with the exact name and dimension as shown below. The array will be used as the FreeRTOS heap. How the array is placed |

| | | | |
|---|---|---|---|
| | | | at a specific memory location is dependent on the compiler being used - refer to your compiler's documentation.<br><br>uint8_t ucHeap[ configTOTAL_HEAP_SIZE ]; |
| Timers > Use Timers | • Enabled<br>• Disabled | Enabled | Set to Enabled to include software timer functionality, or Disabled to omit software timer functionality. See the FreeRTOS software timers page for a full description. |
| Timers > Timer Task Priority | Must be an integer and greater than 0 | 3 | Sets the priority of the software timer service/daemon task. See the FreeRTOS software timers page for a full description. |
| Timers > Timer Queue Length | Must be an integer and greater than 0 | 10 | Sets the length of the software timer command queue. See the FreeRTOS software timers page for a full description. |
| Timers > Timer Task Stack Depth | Must be an integer and greater than 0 | 128 | Sets the stack depth allocated to the software timer service/daemon task. See the FreeRTOS software timers page for a full description. |
| Optional Functions > vTaskPrioritySet() Function | • Enabled<br>• Disabled | Enabled | Include vTaskPrioritySet() function in build |
| Optional Functions > uxTaskPriorityGet() Function | • Enabled<br>• Disabled | Enabled | Include uxTaskPriorityGet() function in build |
| Optional Functions > vTaskDelete() Function | • Enabled<br>• Disabled | Enabled | Include vTaskDelete() function in build |
| Optional Functions > vTaskSuspend() Function | • Enabled<br>• Disabled | Enabled | Include vTaskSuspend() function in build |

| | | | |
|---|---|---|---|
| Optional Functions > xResumeFromISR() Function | • Enabled<br>• Disabled | Enabled | Include xResumeFromISR() function in build |
| Optional Functions > vTaskDelayUntil() Function | • Enabled<br>• Disabled | Enabled | Include vTaskDelayUntil() function in build |
| Optional Functions > vTaskDelay() Function | • Enabled<br>• Disabled | Enabled | Include vTaskDelay() function in build |
| Optional Functions > xTaskGetSchedulerState() Function | • Enabled<br>• Disabled | Enabled | Include xTaskGetSchedulerState() function in build |
| Optional Functions > xTaskGetCurrentTaskHandle() Function | • Enabled<br>• Disabled | Enabled | Include xTaskGetCurrentTaskHandle() function in build |
| Optional Functions > uxTaskGetStackHighWaterMark() Function | • Enabled<br>• Disabled | Disabled | Include uxTaskGetStackHighWaterMark() function in build |
| Optional Functions > xTaskGetIdleTaskHandle() Function | • Enabled<br>• Disabled | Disabled | Include xTaskGetIdleTaskHandle() function in build |
| Optional Functions > eTaskGetState() Function | • Enabled<br>• Disabled | Disabled | Include eTaskGetState() function in build |
| Optional Functions > xEventGroupSetBitFromISR() Function | • Enabled<br>• Disabled | Enabled | Include xEventGroupSetBitFromISR() function in build |
| Optional Functions > xTimerPendFunctionCall() Function | • Enabled<br>• Disabled | Disabled | Include xTimerPendFunctionCall() function in build |
| Optional Functions > xTaskAbortDelay() Function | • Enabled<br>• Disabled | Disabled | Include xTaskAbortDelay() function in build |
| Optional Functions > xTaskGetHandle() Function | • Enabled<br>• Disabled | Disabled | Include xTaskGetHandle() function in build |
| Optional Functions > xTaskResumeFromISR() Function | • Enabled<br>• Disabled | Enabled | Include xTaskResumeFromISR() function in build |
| RA > Hardware Stack Monitor | • Enabled<br>• Disabled | Disabled | Include RA stack monitor |
| Logging > Print String Function | Manual Entry | printf(x) | |
| Logging > Logging Max Message Length | Manual Entry | 192 | |
| Logging > Logging | • Disabled | Disabled | |

| | |
|---|---|
| Include Time and Task Name | • Enabled |

### Clock Configuration

The FreeRTOS port uses the SysTick timer as the system clock. The timer rate is configured in the FreeRTOS component under General > Tick Rate Hz.

### Pin Configuration

This module does not use I/O pins.

## Usage Notes

### Hardware Stack Monitor

The hardware stack monitor generates an NMI if the PSP goes out of the memory area for the stack allocated for the current task. A callback can be registered using R_BSP_GroupIrqWrite() to be called whenever a stack overflow or underflow of the PSP for a particular thread is detected.

### Stack Monitor Underflow Detection

By default the hardware stack monitor only checks for overflow of the process stack. To check for underflow define configRECORD_STACK_HIGH_ADDRESS as 1 on the command line.

### Low Power Modes

When FreeRTOS is configured to use tickless idle, the idle task executes WFI() when no task is ready to run. If the MCU is configured to enter software standby mode or deep software standby mode when the idle task executes WFI(), the RA FreeRTOS port changes the low power mode to sleep mode so the idle task can wake from SysTick. The low power mode settings are restored when the MCU wakes from sleep mode.

## Examples

### Stack Monitor Example

This is an example of using the stack monitor in an application.

```
void stack_monitor_callback(bsp_grp_irq_t irq);
void rm_freertos_port_stack_monitor_example(void);
void stack_monitor_callback (bsp_grp_irq_t irq)
{
 FSP_PARAMETER_NOT_USED(irq);
 if (1U == R_MPU_SPMON->SP[0].CTL_b.ERROR)
    {
 /* Handle main stack monitor error here. */
    }
```

```
 if (1U == R_MPU_SPMON->SP[1].CTL_b.ERROR)
    {
 /* Handle process stack monitor error here. */
    }
}
void rm_freertos_port_stack_monitor_example (void)
{
 /* Register a callback to be called when the stack goes outside the allocated stack
area. */
 R_BSP_GroupIrqWrite(BSP_GRP_IRQ_MPU_STACK, stack_monitor_callback);
}
```

### 4.2.63 LittleFS Flash Port (rm_littlefs_flash)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | RM_LITTLEFS_FLASH_Open (rm_littlefs_ctrl_t *const p_ctrl, rm_littlefs_cfg_t const *const p_cfg) |
| fsp_err_t | RM_LITTLEFS_FLASH_Close (rm_littlefs_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_LITTLEFS_FLASH_VersionGet (fsp_version_t *const p_version) |

**Detailed Description**

Middleware for the LittleFS File System control on RA MCUs.

# Overview

This module provides the hardware port layer for the LittleFS file system. After initializing this module, refer to the LittleFS documentation to use the file system:
https://github.com/ARMmbed/littlefs

# Configuration

**Build Time Configurations for rm_littlefs_flash**

The following build time configurations are defined in fsp_cfg/rm_littlefs_flash_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking Enable | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Configurations for Middleware > LittleFS on Flash**

This module can be added to the Stacks tab via New Stack > Middleware > LittleFS on Flash:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rm_littlefs0 | Module name. |
| Read Size | Must be a non-negative integer | 1 | Minimum size of a block read. All read operations will be a multiple of this value. |
| Program Size | Must be a non-negative integer | 4 | Minimum size of a block program. All program operations will be a multiple of this value. |
| Block Size (bytes) | Must be a multiple of 64 | 128 | Size of an erasable block. This does not impact RAM consumption and may be larger than the physical erase size. However, non-inlined files take up at minimum one block. Must be a multiple of the read and program sizes. |
| Block Count | Manual Entry | (BSP_DATA_FLASH_SIZE_BYTES/128) | Number of erasable blocks on the device. |
| Block Cycles | Must be an integer | 1024 | Number of erase cycles before LittleFS evicts metadata logs and moves the metadata to another block. Suggested values are in the range 100-1000, with large values having better performance at the cost of less consistent wear distribution. Set to -1 to disable block-level wear-leveling. |

| | | | |
|---|---|---|---|
| Cache Size | Must be a non-negative integer | 64 | Size of block caches. Each cache buffers a portion of a block in RAM. The LittleFS needs a read cache, a program cache, and one additional cache per file. Larger caches can improve performance by storing more data and reducing the number of disk accesses. Must be a multiple of the read and program sizes, and a factor of the block size. |
| Lookahead Size | Must be a non-negative multiple of 8 | 16 | Size of the lookahead buffer in bytes. A larger lookahead buffer increases the number of blocks found during an allocation pass. The lookahead buffer is stored as a compact bitmap, so each byte of RAM can track 8 blocks. Must be a multiple of 8. |

**Common LittleFS Configuration**

**Build Time Configurations for LittleFS**

The following build time configurations are defined in arm/littlefs/lfs_util.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Custom lfs_util.h | Manual Entry | | Add a path to your custom lfs_util.h file. It can be used to override some or all of the configurations defined here, and to define additional configurations. |
| Use Malloc | • Enabled<br>• Disabled | Enabled | Configures the use of malloc by LittleFS. |
| Use Assert | • Enabled<br>• Disabled | Enabled | Configures the use of assert by LittleFS. |
| Debug Messages | • Enabled<br>• Disabled | Enabled | Configures debug messages. |

| | | | |
|---|---|---|---|
| Warning Messages | • Enabled<br>• Disabled | Enabled | Configures warning messages. |
| Error Messages | • Enabled<br>• Disabled | Enabled | Configures error messages. |
| Trace Messages | • Enabled<br>• Disabled | Disabled | Configures trace messages. |
| Intrinsics | • Enabled<br>• Disabled | Enabled | Configures intrinsic functions such as __builtin_clz. |
| Instance Name for STDIO wrapper | Name must be a valid C symbol | g_rm_littlefs0 | The rm_littlefs instance name to use with the STDIO wrapper. |

# Usage Notes

### Blocking Read/Write/Erase

The LittleFS port blocks on Read/Write/Erase calls until the operation has completed.

### Memory Constraints

The block size defined in the LittleFS configuration must be a multiple of the data flash erase size of the MCU. It must be greater than 104bytes which is the minimum block size of a LittleFS block. For information about data flash erase sizes refer to the "Specifications of the code flash memory and data flash memory" table of the "Flash Memory" chapter's "Overview" section.

### Limitations

This module is not thread safe.

# Examples

### Basic Example

This is a basic example of LittleFS on Flash in an application.

```
extern const rm_littlefs_cfg_t g_rm_littlefs_flash0_cfg;
#ifdef LFS_NO_MALLOC
static uint8_t g_file_buffer[LFS_CACHE_SIZE];
static struct lfs_file_config g_file_cfg =
{
    .buffer = g_file_buffer
};
#endif
void rm_littlefs_example (void)
```

```
{
    uint8_t    buffer[30];
    lfs_file_t file;
 /* Open LittleFS Flash port.*/
 fsp_err_t err = RM_LITTLEFS_FLASH_Open(&g_rm_littlefs_flash0_ctrl,
&g_rm_littlefs_flash0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Format the filesystem. */
 int lfs_err = lfs_format(&g_rm_littlefs_flash0_lfs, &g_rm_littlefs_flash0_lfs_cfg);
    handle_lfs_error(lfs_err);
 /* Mount the filesystem. */
    lfs_err = lfs_mount(&g_rm_littlefs_flash0_lfs, &g_rm_littlefs_flash0_lfs_cfg);
    handle_lfs_error(lfs_err);
 /* Create a breakfast directory. */
    lfs_err = lfs_mkdir(&g_rm_littlefs_flash0_lfs, "breakfast");
    handle_lfs_error(lfs_err);
 /* Create a file toast in the breakfast directory. */
 const char * path = "breakfast/toast";
#ifdef LFS_NO_MALLOC
 /*********************************************************************************
************************************
  * By default LittleFS uses malloc to allocate buffers. This can be disabled in the
RA Configuration editor.
  * Buffers will be generated from the configuration for the read, program and
lookahead buffers.
  * When opening a file a unique buffer must be passed in for use as a file buffer.
  * The buffer size must be equal to the cache size.
  *********************************************************************************
************************************/
    lfs_err = lfs_file_opencfg(&g_rm_littlefs_flash0_lfs,
                               &file,
                               path,
                               LFS_O_WRONLY | LFS_O_CREAT | LFS_O_APPEND,
```

```
                                &g_file_cfg);
    handle_lfs_error(lfs_err);
#else
    lfs_err = lfs_file_open(&g_rm_littlefs_flash0_lfs, &file, path, LFS_O_WRONLY |
LFS_O_CREAT | LFS_O_APPEND);
    handle_lfs_error(lfs_err);
#endif
 const char * contents = "butter";
    lfs_size_t   len      = strlen(contents);
 /* Apply butter to toast 10 times. */
 for (uint32_t i = 0; i < 10; i++)
    {
       lfs_err = lfs_file_write(&g_rm_littlefs_flash0_lfs, &file, contents, len);
 if (lfs_err < 0)
      {
      handle_lfs_error(lfs_err);
       }
    }
 /* Close the file. */
    lfs_err = lfs_file_close(&g_rm_littlefs_flash0_lfs, &file);
    handle_lfs_error(lfs_err);
 /* Unmount the filesystem. */
    lfs_err = lfs_unmount(&g_rm_littlefs_flash0_lfs);
    handle_lfs_error(lfs_err);
 /* Remount the filesystem. */
    lfs_err = lfs_mount(&g_rm_littlefs_flash0_lfs, &g_rm_littlefs_flash0_lfs_cfg);
    handle_lfs_error(lfs_err);
 /* Open breakfast/toast. */
#ifdef LFS_NO_MALLOC
    lfs_err = lfs_file_opencfg(&g_rm_littlefs_flash0_lfs, &file, path, LFS_O_RDONLY,
&g_file_cfg);
    handle_lfs_error(lfs_err);
#else
    lfs_err = lfs_file_open(&g_rm_littlefs_flash0_lfs, &file, path, LFS_O_RDONLY);
```

```
    handle_lfs_error(lfs_err);
#endif
    handle_lfs_error(lfs_err);
 /* Verify the toast is buttered the correct amount. */
 for (uint32_t i = 0; i < 10; i++)
    {
       lfs_err = lfs_file_read(&g_rm_littlefs_flash0_lfs, &file, buffer, len);
 if (lfs_err < 0)
      {
      handle_lfs_error(lfs_err);
       }
 if (0 != memcmp(buffer, contents, len))
      {
            handle_error(FSP_ERR_ASSERTION);
       }
    }
 /* Close the file. */
    lfs_err = lfs_file_close(&g_rm_littlefs_flash0_lfs, &file);
    handle_lfs_error(lfs_err);
}
```

## Function Documentation

### ◆ RM_LITTLEFS_FLASH_Open()

fsp_err_t RM_LITTLEFS_FLASH_Open ( rm_littlefs_ctrl_t *const *p_ctrl*, rm_littlefs_cfg_t const *const *p_cfg* )

Opens the driver and initializes lower layer driver.

Implements rm_littlefs_api_t::open().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_INVALID_SIZE | The provided block size is invalid. |
| FSP_ERR_INVALID_ARGUMENT | Flash BGO mode must be disabled. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- flash_api_t::open

### ◆ RM_LITTLEFS_FLASH_Close()

fsp_err_t RM_LITTLEFS_FLASH_Close ( rm_littlefs_ctrl_t *const *p_ctrl*)

Closes the lower level driver.

Implements rm_littlefs_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Media device closed. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- flash_api_t::close

### ◆ RM_LITTLEFS_FLASH_VersionGet()

fsp_err_t RM_LITTLEFS_FLASH_VersionGet ( fsp_version_t *const *p_version*)

Returns the version of this module.

Implements rm_littlefs_api_t::versionGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_ASSERTION | Failed in acquiring version information. |

## 4.2.64 Crypto Middleware (rm_psa_crypto)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_PSA_CRYPTO_TRNG_Read (uint8_t *const p_rngbuf, uint32_t num_req_bytes, uint32_t *p_num_gen_bytes)<br>Reads requested length of random data from the TRNG. Generate nbytes of random bytes and store them in p_rngbuf buffer. More... |
| int | mbedtls_platform_setup (mbedtls_platform_context *ctx) |
| void | mbedtls_platform_teardown (mbedtls_platform_context *ctx) |

### Detailed Description

Hardware acceleration for the mbedCrypto implementation of the ARM PSA Crypto API.

# Overview

*Note*

***The PSA Crypto module does not provide any interfaces to the user. This release uses the mbed-Crypto version 3.1.0 which conforms to the PSA Crypto API 1.0 beta3 specification. Consult the ARM mbedCrypto documentation at https://github.com/ARMmbed/mbed-crypto/blob/mbedcrypto-3.1.0/docs/getting_started.md for further information.***

### Features

The PSA_Crypto module provides hardware support for the following PSA Crypto operations

- SHA256 calculation
- SHA224 calculation
- AES
    - Keybits - 128, 256
    - Plain-Text Key Generation
    - Wrapped Key Generation
    - Encryption and Decryption with no padding and with PKCS7 padding.
    - CBC, CTR and GCM modes
    - Export and Import for Plaintext and Wrapped keys
- ECC
    - Curves:
        - SECP256R1
        - SECP256K1
        - Brainpool256R1
        - SECP384R1
        - Brainpool384R1
    - Plain-Text Key Generation
    - Wrapped Key Generation
    - Signing and Verification
    - Export and Import for Plaintext and Wrapped keys
- RSA
    - Keybits - 2048
    - Plain-Text Key Generation
    - Wrapped Key Generation
    - Signing and Verification
    - Encryption and Decryption with PKCS1V15 and OAEP padding
    - Export and Import for Plaintext and Wrapped keys
- Random number generation
- Persistent Key Storage

# Configuration

### Build Time Configurations for mbedCrypto

The following build time configurations are defined in arm/mbedtls/config.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Hardware Acceleration > Key Format > AES | MCU Specific Options | | Select AES key formats used |
| Hardware Acceleration > Key Format > ECC | MCU Specific Options | | Select ECC key formats used |
| Hardware Acceleration > Key Format > RSA | MCU Specific Options | | Select RSA key formats used |
| Hardware Acceleration > Hash > SHA256/224 | MCU Specific Options | | Defines MBEDTLS_SHA256_ALT and MBEDTLS_SHA256_PROCESS_ALT. |
| Hardware Acceleration > Cipher > AES | MCU Specific Options | | Defines MBEDTLS_AES_ALT, MBEDTLS_AES_SETKEY_EN |

| | | | |
|---|---|---|---|
| | | | C_ALT, MBEDTLS_AES_SETKEY_DEC_ALT, MBEDTLS_AES_ENCRYPT_ALT and MBEDTLS_AES_DECRYPT_ALT |
| Hardware Acceleration > Public Key Cryptography (PKC) > ECC | MCU Specific Options | | Defines MBEDTLS_ECP_ALT |
| Hardware Acceleration > Public Key Cryptography (PKC) > ECDSA | MCU Specific Options | | Defines MBEDTLS_ECDSA_SIGN_ALT and MBEDTLS_ECDSA_VERIFY_ALT |
| Hardware Acceleration > Public Key Cryptography (PKC) > RSA | MCU Specific Options | | Defines MBEDTLS_RSA_ALT. |
| Hardware Acceleration > TRNG | Enabled | Enabled | Defines MBEDTLS_ENTROPY_HARDWARE_ALT. |
| Hardware Acceleration > Secure Crypto Engine Initialization | Enabled | Enabled | MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT |
| Platform > Alternate > MBEDTLS_PLATFORM_EXIT_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_EXIT_ALT |
| Platform > Alternate > MBEDTLS_PLATFORM_TIME_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_TIME_ALT |
| Platform > Alternate > MBEDTLS_PLATFORM_FPRINTF_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_FPRINTF_ALT |
| Platform > Alternate > MBEDTLS_PLATFORM_PRINTF_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_PRINTF_ALT |
| Platform > Alternate > MBEDTLS_PLATFORM_SNPRINTF_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_SNPRINTF_ALT |
| Platform > Alternate > MBEDTLS_PLATFORM_VSNPRINTF_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_VSNPRINTF_ALT |
| Platform > Alternate > MBEDTLS_PLATFORM_NV_SEED_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_NV_SEED_ALT |
| Platform > Alternate > MBEDTLS_PLATFORM_ZEROIZE_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_ZEROIZE_ALT |

| | | | |
|---|---|---|---|
| Platform > Alternate > MBEDTLS_PLATFORM_GMTIME_R_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_GMTIME_R_ALT |
| Platform > MBEDTLS_HAVE_ASM | • Define<br>• Undefine | Undefine | MBEDTLS_HAVE_ASM |
| Platform > MBEDTLS_NO_UDBL_DIVISION | • Define<br>• Undefine | Undefine | MBEDTLS_NO_UDBL_DIVISION |
| Platform > MBEDTLS_NO_64BIT_MULTIPLICATION | • Define<br>• Undefine | Undefine | MBEDTLS_NO_64BIT_MULTIPLICATION |
| Platform > MBEDTLS_HAVE_SSE2 | • Define<br>• Undefine | Undefine | MBEDTLS_HAVE_SSE2 |
| Platform > MBEDTLS_HAVE_TIME | • Define<br>• Undefine | Undefine | MBEDTLS_HAVE_TIME |
| Platform > MBEDTLS_HAVE_TIME_DATE | • Define<br>• Undefine | Undefine | MBEDTLS_HAVE_TIME_DATE |
| Platform > MBEDTLS_PLATFORM_MEMORY | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_MEMORY |
| Platform > MBEDTLS_PLATFORM_NO_STD_FUNCTIONS | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_NO_STD_FUNCTIONS |
| Platform > MBEDTLS_TIMING_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_TIMING_ALT |
| Platform > MBEDTLS_NO_PLATFORM_ENTROPY | • Define<br>• Undefine | Define | MBEDTLS_NO_PLATFORM_ENTROPY |
| Platform > MBEDTLS_ENTROPY_C | • Define<br>• Undefine | Define | MBEDTLS_ENTROPY_C |
| Platform > MBEDTLS_PLATFORM_C | • Define<br>• Undefine | Define | MBEDTLS_PLATFORM_C |
| Platform > MBEDTLS_PLATFORM_STD_CALLOC | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_CALLOC |
| Platform > MBEDTLS_PLATFORM_STD_CALLOC value | Manual Entry | calloc | MBEDTLS_PLATFORM_STD_CALLOC value |
| Platform > MBEDTLS_PLATFORM_STD_FREE | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_FREE |
| Platform > MBEDTLS_PLATFORM_STD_FREE value | Manual Entry | free | MBEDTLS_PLATFORM_STD_FREE value |
| Platform > MBEDTLS_PLATFORM_STD_EXIT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_EXIT |
| Platform > MBEDTLS_PLATFORM_STD_EXIT value | Manual Entry | exit | MBEDTLS_PLATFORM_STD_EXIT value |

| | | | |
|---|---|---|---|
| Platform > MBEDTLS_PLATFORM_STD_TIME | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_TIME |
| Platform > MBEDTLS_PLATFORM_STD_TIME value | Manual Entry | time | MBEDTLS_PLATFORM_STD_TIME value |
| Platform > MBEDTLS_PLATFORM_STD_FPRINTF | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_FPRINTF |
| Platform > MBEDTLS_PLATFORM_STD_FPRINTF value | Manual Entry | fprintf | MBEDTLS_PLATFORM_STD_FPRINTF value |
| Platform > MBEDTLS_PLATFORM_STD_PRINTF | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_PRINTF |
| Platform > MBEDTLS_PLATFORM_STD_PRINTF value | Manual Entry | printf | MBEDTLS_PLATFORM_STD_PRINTF value |
| Platform > MBEDTLS_PLATFORM_STD_SNPRINTF | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_SNPRINTF |
| Platform > MBEDTLS_PLATFORM_STD_SNPRINTF value | Manual Entry | snprintf | MBEDTLS_PLATFORM_STD_SNPRINTF value |
| Platform > MBEDTLS_PLATFORM_STD_EXIT_SUCCESS | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_EXIT_SUCCESS |
| Platform > MBEDTLS_PLATFORM_STD_EXIT_SUCCESS value | Manual Entry | 0 | MBEDTLS_PLATFORM_STD_EXIT_SUCCESS value |
| Platform > MBEDTLS_PLATFORM_STD_EXIT_FAILURE | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_EXIT_FAILURE |
| Platform > MBEDTLS_PLATFORM_STD_EXIT_FAILURE value | Manual Entry | 1 | MBEDTLS_PLATFORM_STD_EXIT_FAILURE value |
| Platform > MBEDTLS_PLATFORM_STD_NV_SEED_READ | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_NV_SEED_READ |
| Platform > MBEDTLS_PLATFORM_STD_NV_SEED_READ value | Manual Entry | mbedtls_platform_std_nv_seed_read | MBEDTLS_PLATFORM_STD_NV_SEED_READ value |
| Platform > MBEDTLS_PLATFORM_STD_NV_SEED_WRITE | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_NV_SEED_WRITE |
| Platform > MBEDTLS_PLATFORM_STD_NV_SEED_WRITE value | Manual Entry | mbedtls_platform_std_nv_seed_write | MBEDTLS_PLATFORM_STD_NV_SEED_WRITE value |

| | | | |
|---|---|---|---|
| Platform > MBEDTLS_PLATFORM_STD_NV_SEED_FILE | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_NV_SEED_FILE |
| Platform > MBEDTLS_PLATFORM_STD_NV_SEED_FILE value | Manual Entry | | MBEDTLS_PLATFORM_STD_NV_SEED_FILE value |
| Platform > MBEDTLS_PLATFORM_CALLOC_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_CALLOC_MACRO |
| Platform > MBEDTLS_PLATFORM_CALLOC_MACRO value | Manual Entry | calloc | MBEDTLS_PLATFORM_CALLOC_MACRO value |
| Platform > MBEDTLS_PLATFORM_FREE_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_FREE_MACRO |
| Platform > MBEDTLS_PLATFORM_FREE_MACRO value | Manual Entry | free | MBEDTLS_PLATFORM_FREE_MACRO value |
| Platform > MBEDTLS_PLATFORM_EXIT_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_EXIT_MACRO |
| Platform > MBEDTLS_PLATFORM_EXIT_MACRO value | Manual Entry | exit | MBEDTLS_PLATFORM_EXIT_MACRO value |
| Platform > MBEDTLS_PLATFORM_TIME_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_TIME_MACRO |
| Platform > MBEDTLS_PLATFORM_TIME_MACRO value | Manual Entry | time | MBEDTLS_PLATFORM_TIME_MACRO value |
| Platform > MBEDTLS_PLATFORM_TIME_TYPE_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_TIME_TYPE_MACRO |
| Platform > MBEDTLS_PLATFORM_TIME_TYPE_MACRO value | Manual Entry | time_t | MBEDTLS_PLATFORM_TIME_TYPE_MACRO value |
| Platform > MBEDTLS_PLATFORM_FPRINTF_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_FPRINTF_MACRO |
| Platform > MBEDTLS_PLATFORM_FPRINTF_MACRO value | Manual Entry | fprintf | MBEDTLS_PLATFORM_FPRINTF_MACRO value |
| Platform > MBEDTLS_PLATFORM_PRINTF_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_PRINTF_MACRO |
| Platform > MBEDTLS_PLATFORM_PRINTF_MAC | Manual Entry | printf | MBEDTLS_PLATFORM_PRINTF_MACRO value |

| | | | |
|---|---|---|---|
| RO value | | | |
| Platform > MBEDTLS_PLATFORM_SNPRINTF_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_SNPRINTF_MACRO |
| Platform > MBEDTLS_PLATFORM_SNPRINTF_MACRO value | Manual Entry | snprintf | MBEDTLS_PLATFORM_SNPRINTF_MACRO value |
| Platform > MBEDTLS_PLATFORM_VSNPRINTF_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_VSNPRINTF_MACRO |
| Platform > MBEDTLS_PLATFORM_VSNPRINTF_MACRO value | Manual Entry | vsnprintf | MBEDTLS_PLATFORM_VSNPRINTF_MACRO value |
| Platform > MBEDTLS_PLATFORM_NV_SEED_READ_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_NV_SEED_READ_MACRO |
| Platform > MBEDTLS_PLATFORM_NV_SEED_READ_MACRO value | Manual Entry | mbedtls_platform_std_nv_seed_read | MBEDTLS_PLATFORM_NV_SEED_READ_MACRO value |
| Platform > MBEDTLS_PARAM_FAILED | • Define<br>• Undefine | Undefine | MBEDTLS_PARAM_FAILED |
| Platform > MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO |
| Platform > MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO value | Manual Entry | mbedtls_platform_std_nv_seed_write | MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO value |
| General > MBEDTLS_DEPRECATED_WARNING | • Define<br>• Undefine | Undefine | MBEDTLS_DEPRECATED_WARNING |
| General > MBEDTLS_DEPRECATED_REMOVED | • Define<br>• Undefine | Define | MBEDTLS_DEPRECATED_REMOVED |
| General > MBEDTLS_CHECK_PARAMS | • Define<br>• Undefine | Define | MBEDTLS_CHECK_PARAMS |
| General > MBEDTLS_CHECK_PARAMS_ASSERT | • Define<br>• Undefine | Undefine | MBEDTLS_CHECK_PARAMS_ASSERT |
| General > MBEDTLS_ERROR_STRERROR_DUMMY | • Define<br>• Undefine | Define | MBEDTLS_ERROR_STRERROR_DUMMY |
| General > MBEDTLS_MEMORY_DEBUG | • Define<br>• Undefine | Undefine | MBEDTLS_MEMORY_DEBUG |
| General > MBEDTLS_MEMORY_BACKTRACE | • Define<br>• Undefine | Undefine | MBEDTLS_MEMORY_BACKTRACE |
| General > MBEDTLS_PSA_CRYPTO_SPM | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_CRYPTO_SPM |

| | | | |
|---|---|---|---|
| General > MBEDTLS_SELF_TEST | • Define<br>• Undefine | Undefine | MBEDTLS_SELF_TEST |
| General > MBEDTLS_THREADING_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_THREADING_ALT |
| General > MBEDTLS_THREADING_PTHREAD | • Define<br>• Undefine | Undefine | MBEDTLS_THREADING_PTHREAD |
| General > MBEDTLS_USE_PSA_CRYPTO | • Define<br>• Undefine | Undefine | MBEDTLS_USE_PSA_CRYPTO |
| General > MBEDTLS_VERSION_FEATURES | • Define<br>• Undefine | Define | MBEDTLS_VERSION_FEATURES |
| General > MBEDTLS_ERROR_C | • Define<br>• Undefine | Define | MBEDTLS_ERROR_C |
| General > MBEDTLS_MEMORY_BUFFER_ALLOC_C | • Define<br>• Undefine | Undefine | MBEDTLS_MEMORY_BUFFER_ALLOC_C |
| General > MBEDTLS_PSA_CRYPTO_C | • Define<br>• Undefine | Define | MBEDTLS_PSA_CRYPTO_C |
| General > MBEDTLS_PSA_CRYPTO_SE_C | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_CRYPTO_SE_C |
| General > MBEDTLS_THREADING_C | • Define<br>• Undefine | Undefine | MBEDTLS_THREADING_C |
| General > MBEDTLS_TIMING_C | • Define<br>• Undefine | Undefine | MBEDTLS_TIMING_C |
| General > MBEDTLS_VERSION_C | • Define<br>• Undefine | Define | MBEDTLS_VERSION_C |
| General > MBEDTLS_MEMORY_ALIGN_MULTIPLE | • Define<br>• Undefine | Undefine | MBEDTLS_MEMORY_ALIGN_MULTIPLE |
| General > MBEDTLS_MEMORY_ALIGN_MULTIPLE value | Manual Entry | 4 | MBEDTLS_MEMORY_ALIGN_MULTIPLE value |
| Cipher > Alternate > MBEDTLS_ARC4_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ARC4_ALT |
| Cipher > Alternate > MBEDTLS_ARIA_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ARIA_ALT |
| Cipher > Alternate > MBEDTLS_BLOWFISH_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_BLOWFISH_ALT |
| Cipher > Alternate > MBEDTLS_CAMELLIA_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_CAMELLIA_ALT |
| Cipher > Alternate > MBEDTLS_CCM_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_CCM_ALT |
| Cipher > Alternate > M | • Define | Undefine | MBEDTLS_CHACHA20_ |

| | | | |
|---|---|---|---|
| BEDTLS_CHACHA20_ALT | • Undefine | | ALT |
| Cipher > Alternate > MBEDTLS_CHACHAPOLY_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_CHACHAPOLY_ALT |
| Cipher > Alternate > MBEDTLS_CMAC_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_CMAC_ALT |
| Cipher > Alternate > MBEDTLS_DES_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_DES_ALT |
| Cipher > Alternate > MBEDTLS_GCM_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_GCM_ALT |
| Cipher > Alternate > MBEDTLS_NIST_KW_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_NIST_KW_ALT |
| Cipher > Alternate > MBEDTLS_XTEA_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_XTEA_ALT |
| Cipher > Alternate > MBEDTLS_DES_SETKEY_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_DES_SETKEY_ALT |
| Cipher > Alternate > MBEDTLS_DES_CRYPT_ECB_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_DES_CRYPT_ECB_ALT |
| Cipher > Alternate > MBEDTLS_DES3_CRYPT_ECB_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_DES3_CRYPT_ECB_ALT |
| Cipher > AES > MBEDTLS_AES_ROM_TABLES | • Define<br>• Undefine | Undefine | MBEDTLS_AES_ROM_TABLES |
| Cipher > AES > MBEDTLS_AES_FEWER_TABLES | • Define<br>• Undefine | Undefine | MBEDTLS_AES_FEWER_TABLES |
| Cipher > MBEDTLS_CAMELLIA_SMALL_MEMORY | • Define<br>• Undefine | Undefine | MBEDTLS_CAMELLIA_SMALL_MEMORY |
| Cipher > MBEDTLS_CIPHER_MODE_CBC | • Define<br>• Undefine | Define | MBEDTLS_CIPHER_MODE_CBC |
| Cipher > MBEDTLS_CIPHER_MODE_CFB | • Define<br>• Undefine | Define | MBEDTLS_CIPHER_MODE_CFB |
| Cipher > MBEDTLS_CIPHER_MODE_CTR | • Define<br>• Undefine | Define | MBEDTLS_CIPHER_MODE_CTR |
| Cipher > MBEDTLS_CIPHER_MODE_OFB | • Define<br>• Undefine | Undefine | MBEDTLS_CIPHER_MODE_OFB |
| Cipher > MBEDTLS_CIPHER_MODE_XTS | • Define<br>• Undefine | Undefine | MBEDTLS_CIPHER_MODE_XTS |
| Cipher > MBEDTLS_CIP | • Define | Undefine | MBEDTLS_CIPHER_NUL |

| | | | |
|---|---|---|---|
| HER_NULL_CIPHER | • Undefine | | L_CIPHER |
| Cipher > MBEDTLS_CIPHER_PADDING_PKCS7 | • Define<br>• Undefine | Define | MBEDTLS_CIPHER_PADDING_PKCS7 |
| Cipher > MBEDTLS_CIPHER_PADDING_ONE_AND_ZEROS | • Define<br>• Undefine | Define | MBEDTLS_CIPHER_PADDING_ONE_AND_ZEROS |
| Cipher > MBEDTLS_CIPHER_PADDING_ZEROS_AND_LEN | • Define<br>• Undefine | Define | MBEDTLS_CIPHER_PADDING_ZEROS_AND_LEN |
| Cipher > MBEDTLS_CIPHER_PADDING_ZEROS | • Define<br>• Undefine | Define | MBEDTLS_CIPHER_PADDING_ZEROS |
| Cipher > MBEDTLS_AES_C | Define | Define | MBEDTLS_AES_C |
| Cipher > MBEDTLS_ARC4_C | • Define<br>• Undefine | Undefine | MBEDTLS_ARC4_C |
| Cipher > MBEDTLS_BLOWFISH_C | • Define<br>• Undefine | Undefine | MBEDTLS_BLOWFISH_C |
| Cipher > MBEDTLS_CAMELLIA_C | • Define<br>• Undefine | Undefine | MBEDTLS_CAMELLIA_C |
| Cipher > MBEDTLS_ARIA_C | • Define<br>• Undefine | Undefine | MBEDTLS_ARIA_C |
| Cipher > MBEDTLS_CCM_C | • Define<br>• Undefine | Define | MBEDTLS_CCM_C |
| Cipher > MBEDTLS_CHACHA20_C | • Define<br>• Undefine | Undefine | MBEDTLS_CHACHA20_C |
| Cipher > MBEDTLS_CHACHAPOLY_C | • Define<br>• Undefine | Undefine | MBEDTLS_CHACHAPOLY_C |
| Cipher > MBEDTLS_CIPHER_C | • Define<br>• Undefine | Define | MBEDTLS_CIPHER_C |
| Cipher > MBEDTLS_DES_C | • Define<br>• Undefine | Undefine | MBEDTLS_DES_C |
| Cipher > MBEDTLS_GCM_C | • Define<br>• Undefine | Define | MBEDTLS_GCM_C |
| Cipher > MBEDTLS_NIST_KW_C | • Define<br>• Undefine | Undefine | MBEDTLS_NIST_KW_C |
| Cipher > MBEDTLS_XTEA_C | • Define<br>• Undefine | Undefine | MBEDTLS_XTEA_C |
| Public Key Cryptography (PKC) > DHM > Alternate > MBEDTLS_DHM_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_DHM_ALT |
| Public Key | • Define | Undefine | MBEDTLS_DHM_C |

| | | | |
|---|---|---|---|
| Cryptography (PKC) > DHM > MBEDTLS_DHM_C | • Undefine | | |
| Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECJPAKE_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECJPAKE_ALT |
| Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECDSA_GENKEY_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECDSA_GENKEY_ALT |
| Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_INTERNAL_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_INTERNAL_ALT |
| Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_RANDOMIZE_JAC_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_RANDOMIZE_JAC_ALT |
| Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_ADD_MIXED_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_ADD_MIXED_ALT |
| Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_DOUBLE_JAC_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DOUBLE_JAC_ALT |
| Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_NORMALIZE_JAC_MANY_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_NORMALIZE_JAC_MANY_ALT |
| Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_NORMALIZE_JAC_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_NORMALIZE_JAC_ALT |
| Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_DOUBLE_ADD_MXZ_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DOUBLE_ADD_MXZ_ALT |
| Public Key Cryptography (PKC) > | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_RANDOMIZE_MXZ_ALT |

| | | | |
|---|---|---|---|
| ECC > Alternate > MBEDTLS_ECP_RANDOMIZE_MXZ_ALT | | | |
| Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_NORMALIZE_MXZ_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_NORMALIZE_MXZ_ALT |
| Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP192R1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_SECP192R1_ENABLED |
| Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP224R1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_SECP224R1_ENABLED |
| Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP256R1_ENABLED | • Define<br>• Undefine | Define | MBEDTLS_ECP_DP_SECP256R1_ENABLED |
| Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP384R1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_SECP384R1_ENABLED |
| Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP521R1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_SECP521R1_ENABLED |
| Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP192K1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_SECP192K1_ENABLED |
| Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP224K1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_SECP224K1_ENABLED |
| Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP256K1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_SECP256K1_ENABLED |
| Public Key | • Define | Undefine | MBEDTLS_ECP_DP_BP2 |

| | | | |
|---|---|---|---|
| Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_BP256R1_ENABLED | • Undefine | | 56R1_ENABLED |
| Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_BP384R1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_BP384R1_ENABLED |
| Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_BP512R1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_BP512R1_ENABLED |
| Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_CURVE25519_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_CURVE25519_ENABLED |
| Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_CURVE448_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_CURVE448_ENABLED |
| Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDH_GEN_PUBLIC_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECDH_GEN_PUBLIC_ALT |
| Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDH_COMPUTE_SHARED_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECDH_COMPUTE_SHARED_ALT |
| Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_NIST_OPTIM | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_NIST_OPTIM |
| Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_RESTARTABLE | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_RESTARTABLE |
| Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDH_LEGACY_CONTEXT | • Define<br>• Undefine | Undefine | MBEDTLS_ECDH_LEGACY_CONTEXT |
| Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDSA_DETERMINISTIC | • Define<br>• Undefine | Undefine | MBEDTLS_ECDSA_DETERMINISTIC |

| | | | |
|---|---|---|---|
| Public Key Cryptography (PKC) > ECC > MBEDTLS_PK_PARSE_EC_EXTENDED | • Define<br>• Undefine | Undefine | MBEDTLS_PK_PARSE_EC_EXTENDED |
| Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDH_C | • Define<br>• Undefine | Undefine | MBEDTLS_ECDH_C |
| Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDSA_C | • Define<br>• Undefine | Define | MBEDTLS_ECDSA_C |
| Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_C | • Define<br>• Undefine | Define | MBEDTLS_ECP_C |
| Public Key Cryptography (PKC) > ECC > MBEDTLS_ECJPAKE_C | • Define<br>• Undefine | Undefine | MBEDTLS_ECJPAKE_C |
| Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_MAX_BITS | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_MAX_BITS |
| Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_MAX_BITS value | Manual Entry | 521 | MBEDTLS_ECP_MAX_BITS value |
| Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_WINDOW_SIZE | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_WINDOW_SIZE |
| Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_WINDOW_SIZE value | Manual Entry | 6 | MBEDTLS_ECP_WINDOW_SIZE value |
| Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_FIXED_POINT_OPTIM | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_FIXED_POINT_OPTIM |
| Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_FIXED_POINT_OPTIM value | Manual Entry | 1 | MBEDTLS_ECP_FIXED_POINT_OPTIM value |
| Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDH_VARIANT_EVEREST_EN | • Define<br>• Undefine | Undefine | MBEDTLS_ECDH_VARIANT_EVEREST_ENABLED |

| | | | |
|---|---|---|---|
| ABLED | | | |
| Public Key Cryptography (PKC) > RSA > MBEDTLS_PK_RSA_ALT_SUPPORT | • Define<br>• Undefine | Undefine | MBEDTLS_PK_RSA_ALT_SUPPORT |
| Public Key Cryptography (PKC) > RSA > MBEDTLS_RSA_NO_CRT | • Define<br>• Undefine | Define | MBEDTLS_RSA_NO_CRT |
| Public Key Cryptography (PKC) > RSA > MBEDTLS_RSA_C | • Define<br>• Undefine | Define | MBEDTLS_RSA_C |
| Public Key Cryptography (PKC) > MBEDTLS_GENPRIME | • Define<br>• Undefine | Define | MBEDTLS_GENPRIME |
| Public Key Cryptography (PKC) > MBEDTLS_PKCS1_V15 | • Define<br>• Undefine | Define | MBEDTLS_PKCS1_V15 |
| Public Key Cryptography (PKC) > MBEDTLS_PKCS1_V21 | • Define<br>• Undefine | Define | MBEDTLS_PKCS1_V21 |
| Public Key Cryptography (PKC) > MBEDTLS_ASN1_PARSE_C | • Define<br>• Undefine | Define | MBEDTLS_ASN1_PARSE_C |
| Public Key Cryptography (PKC) > MBEDTLS_ASN1_WRITE_C | • Define<br>• Undefine | Define | MBEDTLS_ASN1_WRITE_C |
| Public Key Cryptography (PKC) > MBEDTLS_BASE64_C | • Define<br>• Undefine | Define | MBEDTLS_BASE64_C |
| Public Key Cryptography (PKC) > MBEDTLS_BIGNUM_C | • Define<br>• Undefine | Define | MBEDTLS_BIGNUM_C |
| Public Key Cryptography (PKC) > MBEDTLS_OID_C | • Define<br>• Undefine | Define | MBEDTLS_OID_C |
| Public Key Cryptography (PKC) > MBEDTLS_PEM_PARSE_C | • Define<br>• Undefine | Define | MBEDTLS_PEM_PARSE_C |
| Public Key Cryptography (PKC) > MBEDTLS_PEM_WRITE_C | • Define<br>• Undefine | Define | MBEDTLS_PEM_WRITE_C |

| | | | |
|---|---|---|---|
| Public Key Cryptography (PKC) > MBEDTLS_PK_C | • Define<br>• Undefine | Define | MBEDTLS_PK_C |
| Public Key Cryptography (PKC) > MBEDTLS_PK_PARSE_C | • Define<br>• Undefine | Define | MBEDTLS_PK_PARSE_C |
| Public Key Cryptography (PKC) > MBEDTLS_PK_WRITE_C | • Define<br>• Undefine | Define | MBEDTLS_PK_WRITE_C |
| Public Key Cryptography (PKC) > MBEDTLS_PKCS5_C | • Define<br>• Undefine | Define | MBEDTLS_PKCS5_C |
| Public Key Cryptography (PKC) > MBEDTLS_PKCS12_C | • Define<br>• Undefine | Define | MBEDTLS_PKCS12_C |
| Public Key Cryptography (PKC) > MBEDTLS_MPI_WINDOW_SIZE | • Define<br>• Undefine | Undefine | MBEDTLS_MPI_WINDOW_SIZE |
| Public Key Cryptography (PKC) > MBEDTLS_MPI_WINDOW_SIZE value | Manual Entry | 6 | MBEDTLS_MPI_WINDOW_SIZE value |
| Public Key Cryptography (PKC) > MBEDTLS_MPI_MAX_SIZE | • Define<br>• Undefine | Undefine | MBEDTLS_MPI_MAX_SIZE |
| Public Key Cryptography (PKC) > MBEDTLS_MPI_MAX_SIZE value | Manual Entry | 1024 | MBEDTLS_MPI_MAX_SIZE value |
| Hash > Alternate > MBEDTLS_MD2_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_MD2_ALT |
| Hash > Alternate > MBEDTLS_MD4_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_MD4_ALT |
| Hash > Alternate > MBEDTLS_MD5_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_MD5_ALT |
| Hash > Alternate > MBEDTLS_RIPEMD160_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_RIPEMD160_ALT |
| Hash > Alternate > MBEDTLS_SHA1_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_SHA1_ALT |
| Hash > Alternate > MBEDTLS_SHA512_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_SHA512_ALT |
| Hash > Alternate > MBEDTLS_MD2_PROCESS_ | • Define<br>• Undefine | Undefine | MBEDTLS_MD2_PROCESS_ALT |

| | | | |
|---|---|---|---|
| ALT | | | |
| Hash > Alternate > MBEDTLS_MD4_PROCESS_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_MD4_PROCESS_ALT |
| Hash > Alternate > MBEDTLS_MD5_PROCESS_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_MD5_PROCESS_ALT |
| Hash > Alternate > MBEDTLS_RIPEMD160_PROCESS_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_RIPEMD160_PROCESS_ALT |
| Hash > Alternate > MBEDTLS_SHA1_PROCESS_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_SHA1_PROCESS_ALT |
| Hash > Alternate > MBEDTLS_SHA512_PROCESS_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_SHA512_PROCESS_ALT |
| Hash > MBEDTLS_SHA256_SMALLER | • Define<br>• Undefine | Undefine | MBEDTLS_SHA256_SMALLER |
| Hash > MBEDTLS_SHA512_SMALLER | • Define<br>• Undefine | Undefine | MBEDTLS_SHA512_SMALLER |
| Hash > MBEDTLS_SHA512_NO_SHA384 | • Define<br>• Undefine | Undefine | MBEDTLS_SHA512_NO_SHA384 |
| Hash > MBEDTLS_MD_C | • Define<br>• Undefine | Define | MBEDTLS_MD_C |
| Hash > MBEDTLS_MD2_C | • Define<br>• Undefine | Undefine | MBEDTLS_MD2_C |
| Hash > MBEDTLS_MD4_C | • Define<br>• Undefine | Undefine | MBEDTLS_MD4_C |
| Hash > MBEDTLS_MD5_C | • Define<br>• Undefine | Define | MBEDTLS_MD5_C |
| Hash > MBEDTLS_RIPEMD160_C | • Define<br>• Undefine | Define | MBEDTLS_RIPEMD160_C |
| Hash > MBEDTLS_SHA1_C | • Define<br>• Undefine | Define | MBEDTLS_SHA1_C |
| Hash > MBEDTLS_SHA256_C | • Define<br>• Undefine | Define | MBEDTLS_SHA256_C |
| Hash > MBEDTLS_SHA512_C | • Define<br>• Undefine | Undefine | MBEDTLS_SHA512_C |
| Message Authentication Code (MAC) > Alternate > MBEDTLS_POLY1305_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_POLY1305_ALT |
| Message | • Define | Undefine | MBEDTLS_CMAC_C |

| | | | |
|---|---|---|---|
| Authentication Code (MAC) > MBEDTLS_CMAC_C | • Undefine | | |
| Message Authentication Code (MAC) > MBEDTLS_HKDF_C | • Define<br>• Undefine | Define | MBEDTLS_HKDF_C |
| Message Authentication Code (MAC) > MBEDTLS_HMAC_DRBG_C | • Define<br>• Undefine | Undefine | MBEDTLS_HMAC_DRBG_C |
| Message Authentication Code (MAC) > MBEDTLS_POLY1305_C | • Define<br>• Undefine | Undefine | MBEDTLS_POLY1305_C |
| RNG > MBEDTLS_TEST_NULL_ENTROPY | • Define<br>• Undefine | Undefine | MBEDTLS_TEST_NULL_ENTROPY |
| RNG > MBEDTLS_NO_DEFAULT_ENTROPY_SOURCES | • Define<br>• Undefine | Undefine | MBEDTLS_NO_DEFAULT_ENTROPY_SOURCES |
| RNG > MBEDTLS_ENTROPY_FORCE_SHA256 | • Define<br>• Undefine | Undefine | MBEDTLS_ENTROPY_FORCE_SHA256 |
| RNG > MBEDTLS_ENTROPY_NV_SEED | • Define<br>• Undefine | Undefine | MBEDTLS_ENTROPY_NV_SEED |
| RNG > MBEDTLS_PSA_INJECT_ENTROPY | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_INJECT_ENTROPY |
| RNG > MBEDTLS_CTR_DRBG_C | • Define<br>• Undefine | Define | MBEDTLS_CTR_DRBG_C |
| RNG > MBEDTLS_HAVEGE_C | • Define<br>• Undefine | Undefine | MBEDTLS_HAVEGE_C |
| RNG > MBEDTLS_CTR_DRBG_ENTROPY_LEN | • Define<br>• Undefine | Undefine | RNG\|MBEDTLS_CTR_DRBG_ENTROPY_LEN |
| RNG > MBEDTLS_CTR_DRBG_ENTROPY_LEN value | Manual Entry | 48 | RNG value\|MBEDTLS_CTR_DRBG_ENTROPY_LEN |
| RNG > MBEDTLS_CTR_DRBG_RESEED_INTERVAL | • Define<br>• Undefine | Undefine | RNG\|MBEDTLS_CTR_DRBG_RESEED_INTERVAL |
| RNG > MBEDTLS_CTR_DRBG_RESEED_INTERVAL value | Manual Entry | 10000 | RNG value\|MBEDTLS_CTR_DRBG_RESEED_INTERVAL |
| RNG > MBEDTLS_CTR_DRBG_MAX_INPUT | • Define<br>• Undefine | Undefine | MBEDTLS_CTR_DRBG_MAX_INPUT |
| RNG > MBEDTLS_CTR_DRBG_MAX_INPUT | Manual Entry | 256 | MBEDTLS_CTR_DRBG_MAX_INPUT value |

| | | | |
|---|---|---|---|
| value | | | |
| RNG > MBEDTLS_CTR_DRBG_MAX_REQUEST | • Define<br>• Undefine | Undefine | MBEDTLS_CTR_DRBG_MAX_REQUEST |
| RNG > MBEDTLS_CTR_DRBG_MAX_REQUEST value | Manual Entry | 1024 | MBEDTLS_CTR_DRBG_MAX_REQUEST value |
| RNG > MBEDTLS_CTR_DRBG_MAX_SEED_INPUT | • Define<br>• Undefine | Undefine | MBEDTLS_CTR_DRBG_MAX_SEED_INPUT |
| RNG > MBEDTLS_CTR_DRBG_MAX_SEED_INPUT value | Manual Entry | 384 | MBEDTLS_CTR_DRBG_MAX_SEED_INPUT value |
| RNG > MBEDTLS_CTR_DRBG_USE_128_BIT_KEY | • Define<br>• Undefine | Undefine | MBEDTLS_CTR_DRBG_USE_128_BIT_KEY |
| RNG > MBEDTLS_HMAC_DRBG_RESEED_INTERVAL | • Define<br>• Undefine | Undefine | MBEDTLS_HMAC_DRBG_RESEED_INTERVAL |
| RNG > MBEDTLS_HMAC_DRBG_RESEED_INTERVAL value | Manual Entry | 10000 | MBEDTLS_HMAC_DRBG_RESEED_INTERVAL value |
| RNG > MBEDTLS_HMAC_DRBG_MAX_INPUT | • Define<br>• Undefine | Undefine | MBEDTLS_HMAC_DRBG_MAX_INPUT |
| RNG > MBEDTLS_HMAC_DRBG_MAX_INPUT value | Manual Entry | 256 | MBEDTLS_HMAC_DRBG_MAX_INPUT value |
| RNG > MBEDTLS_HMAC_DRBG_MAX_REQUEST | • Define<br>• Undefine | Undefine | MBEDTLS_HMAC_DRBG_MAX_REQUEST |
| RNG > MBEDTLS_HMAC_DRBG_MAX_REQUEST value | Manual Entry | 1024 | MBEDTLS_HMAC_DRBG_MAX_REQUEST value |
| RNG > MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT | • Define<br>• Undefine | Undefine | MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT |
| RNG > MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT value | Manual Entry | 384 | MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT value |
| RNG > MBEDTLS_ENTROPY_MAX_SOURCES | • Define<br>• Undefine | Undefine | MBEDTLS_ENTROPY_MAX_SOURCES |
| RNG > MBEDTLS_ENTROPY_MAX_SOURCES value | Manual Entry | 20 | MBEDTLS_ENTROPY_MAX_SOURCES value |
| RNG > MBEDTLS_ENTROPY_MAX_GATHER | • Define<br>• Undefine | Undefine | MBEDTLS_ENTROPY_MAX_GATHER |

| | | | |
|---|---|---|---|
| RNG > MBEDTLS_ENTROPY_MAX_GATHER value | Manual Entry | 128 | MBEDTLS_ENTROPY_MAX_GATHER value |
| RNG > MBEDTLS_ENTROPY_MIN_HARDWARE | • Define<br>• Undefine | Undefine | MBEDTLS_ENTROPY_MIN_HARDWARE |
| RNG > MBEDTLS_ENTROPY_MIN_HARDWARE value | Manual Entry | 32 | MBEDTLS_ENTROPY_MIN_HARDWARE value |
| Storage > MBEDTLS_FS_IO | • Define<br>• Undefine | Undefine | MBEDTLS_FS_IO |
| Storage > MBEDTLS_PSA_CRYPTO_KEY_FILE_ID_ENCODES_OWNER | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_CRYPTO_KEY_FILE_ID_ENCODES_OWNER |
| Storage > MBEDTLS_PSA_CRYPTO_STORAGE_C | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_CRYPTO_STORAGE_C |
| Storage > MBEDTLS_PSA_ITS_FILE_C | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_ITS_FILE_C |

**SHA256 Configuration**

To enable hardware acceleration for the SHA256/224 calculation, the macro MBEDTLS_SHA256_ALT and MBEDTLS_SHA256_PROCESS_ALT must be defined in the configuration file. By default SHA256 is enabled. SHA256 can be disabled, but SHA512 then needs to be enabled (software version) because the PSA implementation uses it for the entropy accumulator. This can be done using the RA Configuration editor.

**AES Configuration**

To enable hardware acceleration for the AES128/256 operation, the macro MBEDTLS_AES_SETKEY_ENC_ALT, MBEDTLS_AES_SETKEY_DEC_ALT, MBEDTLS_AES_ENCRYPT_ALT and MBEDTLS_AES_DECRYPT_ALT must be defined in the configuration file. By default AES is enabled. AES cannot be disabled because the PSA immplementation required for the CTR_DRBG randon number generator. This can be done using the RA Configuration editor.

**ECC Configuration**

To enable hardware acceleration for the ECC Key Generation operation, the macro MBEDTLS_ECP_ALT must be defined in the configuration file. For ECDSA, the macros MBEDTLS_ECDSA_SIGN_ALT and MBEDTLS_ECDSA_VERIFY_ALT must be defined. By default ECC, ECDSA and ECDHE are enabled. To disable ECC, undefine MBEDTLS_ECP_C, MBEDTLS_ECDSA_C and MBEDTLS_ECDH_C. This can be done using the RA Configuration editor.

**RSA Configuration**

To enable hardware acceleration for the RSA2048 operation, the macro MBEDTLS_RSA_ALT must be defined in the configuration file. By default RSA is enabled. To disable RSA, undefine MBEDTLS_RSA_C, MBEDTLS_PK_C, MBEDTLS_PK_PARSE_C, MBEDTLS_PK_WRITE_C. This can be done using the RA Configuration editor.

**Wrapped Key Usage**

To use the Secure Crypto Engine to generate and use wrapped keys, use PSA_KEY_LIFETIME_VOLATILE_WRAPPED when setting the key lifetime. Wrapped keys can also be generated by using PSA_KEY_LIFETIME_PERSISTENT_WRAPPED to generate persistent keys as described in the next section. Setting the key's lifetime attribute using this value will cause the SCE to use wrapped key mode for all operations related to that key. The user can use the export functionality to save the wrapped keys to user ROM and import it later for usage. This mode requires that Wrapped Key functionality for the algorithm is enabled in the project configuration.

### Persistent Key Storage

Persistent key storage can be enabled by defining MBEDTLS_FS_IO, MBEDTLS_PSA_CRYPTO_STORAGE_C, and MBEDTLS_PSA_ITS_FILE_C. The key lifetime must also be specifed as either PSA_KEY_LIFETIME_PERSISTENT or PSA_KEY_LIFETIME_PERSISTENT_WRAPPED. A lower level storage module must be added in the RA Configuration editor and initialized in the code before generating persistent keys. Persistent storage supports the use of plaintext and vendor keys. Refer to the lower level storage module documentation for information on how it should be initialized. To generate a persistent key the key must be assigned a unique id prior to calling generate using the psa_set_key_id api.

```
if (PSA_KEY_LIFETIME_IS_PERSISTENT(lifetime))
   {
/* Set the id to a positive integer. */
      psa_set_key_id(&attributes, (psa_key_id_t) 5);
   }
```

### Platform Configuration

To run the mbedCrypto implementation of the PSA Crypto API on the MCU, the macro MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT must be defined in the configuration file. This enables code that will initialize the SCE. Parameter checking (General|MBEDTLS_CHECK_PARAMS) is enabled by default. To reduce code size, disable parameter checking.

### Random Number Configuration

To run the mbedCrypto implementation of the PSA Crypto API on the MCU, the macro MBEDTLS_ENTROPY_HARDWARE_ALT must be defined in the configuration file. This enables using the TRNG as an entropy source. None of the other cryptographic operations (even in software only mode) will work without this feature.

## Usage Notes

### Hardware Initialization

mbedtls_platform_setup() must be invoked before using the PSA Crypto API to ensure that the SCE peripheral is initialized.

### Memory Usage

In general, depending on the mbedCrypto features being used a heap size of 0x1000 to 0x5000 bytes is required. The total allocated heap should be the **sum** of the heap requirements of the

individual algorithms:

| Algorithm | Required Heap (bytes) |
|---|---|
| SHA256/224 | None |
| AES | 0x200 |
| Hardware ECC | 0x400 |
| Software ECC | 0x1800 |
| RSA | 0x1500 |

A minimum stack of 0x1000 is required where the module is used. This is either the main stack in a bare metal application or the task stack of the task used for crypto operations.

**Limitations**

- Only little endian mode is supported.

# Examples

**Hash Example**

This is an example on calculating the SHA256 hash using the PSA Crypto API.

```
const uint8_t NIST_SHA256ShortMsgLen200[] =
{
    0x2e, 0x7e, 0xa8, 0x4d, 0xa4, 0xbc, 0x4d, 0x7c, 0xfb, 0x46, 0x3e, 0x3f, 0x2c,
0x86, 0x47, 0x05,
    0x7a, 0xff, 0xf3, 0xfb, 0xec, 0xec, 0xa1, 0xd2, 00
};
const uint8_t NIST_SHA256ShortMsgLen200_expected[] =
{
    0x76, 0xe3, 0xac, 0xbc, 0x71, 0x88, 0x36, 0xf2, 0xdf, 0x8a, 0xd2, 0xd0, 0xd2,
0xd7, 0x6f, 0x0c,
    0xfa, 0x5f, 0xea, 0x09, 0x86, 0xbe, 0x91, 0x8f, 0x10, 0xbc, 0xee, 0x73, 0x0d,
0xf4, 0x41, 0xb9
};
void psa_crypto_sha256_example (void)
{
    psa_algorithm_t      alg               = PSA_ALG_SHA_256;
    psa_hash_operation_t operation         = {0};
 size_t               expected_hash_len = PSA_HASH_SIZE(alg);
```

```
   uint8_t             actual_hash[PSA_HASH_MAX_SIZE];
 size_t               actual_hash_len;
 mbedtls_platform_context ctx = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
 /* Platform initialization failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_hash_setup(&operation, alg))
    {
 /* Hash setup failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_hash_update(&operation, NIST_SHA256ShortMsgLen200,
sizeof(NIST_SHA256ShortMsgLen200)))
    {
 /* Hash calculation failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_hash_finish(&operation, &actual_hash[0], sizeof
(actual_hash), &actual_hash_len))
    {
 /* Reading calculated hash failed */
       debugger_break();
    }
 else if (0 != memcmp(&actual_hash[0], &NIST_SHA256ShortMsgLen200_expected[0],
actual_hash_len))
    {
 /* Hash compare of calculated value with expected value failed */
       debugger_break();
    }
 else if (0 != memcmp(&expected_hash_len, &actual_hash_len, sizeof
(expected_hash_len)))
```

```
    {
 /* Hash size compare of calculated value with expected value failed */
       debugger_break();
    }
 else
    {
 /* SHA256 calculation succeeded */
       debugger_break();
    }
 /* De-initialize the platform. This is currently a placeholder function which does
not do anything. */
 mbedtls_platform_teardown(&ctx);
}
```

### AES Example

This is an example on using the PSA Crypto API to generate an AES256 key, encrypting and decrypting multi-block data and using PKCS7 padding.

```
static psa_status_t cipher_operation (psa_cipher_operation_t * operation,
 const uint8_t          * input,
 size_t                   input_size,
 size_t                   part_size,
                                    uint8_t                * output,
 size_t                   output_size,
 size_t                * output_len)
{
    psa_status_t status;
 size_t       bytes_to_write = 0;
 size_t       bytes_written = 0;
 size_t       len            = 0;
    *output_len = 0;
 while (bytes_written != input_size)
    {
       bytes_to_write = (input_size - bytes_written > part_size ?
```

```
                        part_size :
                        input_size - bytes_written);
      status = psa_cipher_update(operation,
                                 input + bytes_written,
                                 bytes_to_write,
                                 output + *output_len,
                                 output_size - *output_len,
                                 &len);
 if (PSA_SUCCESS != status)
     {
 return status;
     }
      bytes_written += bytes_to_write;
      *output_len   += len;
   }
   status = psa_cipher_finish(operation, output + *output_len, output_size -
*output_len, &len);
 if (PSA_SUCCESS != status)
   {
 return status;
   }
   *output_len += len;
 return status;
}
void psa_crypto_aes256cbcmultipart_example (void)
{
 enum
   {
      block_size = PSA_BLOCK_CIPHER_BLOCK_SIZE(PSA_KEY_TYPE_AES),
      key_bits   = 256,
      input_size = 100,
      part_size = 10,
   };
 mbedtls_platform_context ctx         = {0};
```

```
const psa_algorithm_t    alg         = PSA_ALG_CBC_PKCS7;
   psa_cipher_operation_t   operation_1 = PSA_CIPHER_OPERATION_INIT;
   psa_cipher_operation_t   operation_2 = PSA_CIPHER_OPERATION_INIT;
size_t iv_len = 0;
   psa_key_handle_t    key_handle       = 0;
size_t               encrypted_length = 0;
size_t               decrypted_length = 0;
   uint8_t             iv[block_size]    = {0};
   uint8_t             input[input_size] = {0};
   uint8_t             encrypted_data[input_size + block_size] = {0};
   uint8_t             decrypted_data[input_size + block_size] = {0};
   psa_key_attributes_t attributes = PSA_KEY_ATTRIBUTES_INIT;
/* Setup the platform; initialize the SCE */
if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
   {
/* Platform initialization failed */
      debugger_break();
   }
if (PSA_SUCCESS != psa_crypto_init())
   {
/* PSA Crypto Initialization failed */
      debugger_break();
   }
/* Set key attributes */
   psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_ENCRYPT |
PSA_KEY_USAGE_DECRYPT);
   psa_set_key_algorithm(&attributes, alg);
   psa_set_key_type(&attributes, PSA_KEY_TYPE_AES);
   psa_set_key_bits(&attributes, key_bits);
   psa_key_lifetime_t lifetime = PSA_KEY_LIFETIME_VOLATILE;
/* To use wrapped keys instead of plaintext:
 * - Use PSA_KEY_LIFETIME_VOLATILE_WRAPPED or PSA_KEY_LIFETIME_PERSISTENT_WRAPPED.
 * - To use persistent keys:
 * - The file system must be initialized prior to calling the generate/import key
```

```
functions.
  * - Refer to the littlefs example to see how to format and mount the filesystem. */
    psa_set_key_lifetime(&attributes, lifetime);
 if (PSA_KEY_LIFETIME_IS_PERSISTENT(lifetime))
    {
 /* Set the id to a positive integer. */
       psa_set_key_id(&attributes, (psa_key_id_t) 5);
    }
 if (PSA_SUCCESS != psa_generate_random(input, sizeof(input)))
    {
 /* Random number generation for input data failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
    {
 /* Generating AES 256 key and allocating to key slot failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_encrypt_setup(&operation_1, key_handle, alg))
    {
 /* Initializing the encryption (with PKCS7 padding) operation handle failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_generate_iv(&operation_1, iv, sizeof(iv),
&iv_len))
    {
 /* Generating the random IV failed */
       debugger_break();
    }
 else if (PSA_SUCCESS !=
             cipher_operation(&operation_1, input, input_size, part_size,
encrypted_data, sizeof(encrypted_data),
                              &encrypted_length))
    {
```

```
 /* Encryption failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_abort(&operation_1))
    {
 /* Terminating the encryption operation failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_decrypt_setup(&operation_2, key_handle, alg))
    {
 /* Initializing the decryption (with PKCS7 padding) operation handle failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_set_iv(&operation_2, iv, sizeof(iv)))
    {
 /* Setting the IV failed */
       debugger_break();
    }
 else if (PSA_SUCCESS !=
             cipher_operation(&operation_2, encrypted_data, encrypted_length,
part_size, decrypted_data,
 sizeof(decrypted_data), &decrypted_length))
    {
 /* Decryption failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_abort(&operation_2))
    {
 /* Terminating the decryption operation failed */
       debugger_break();
    }
 else if (0 != memcmp(input, decrypted_data, sizeof(input)))
    {
 /* Comparing the input data with decrypted data failed */
```

```
      debugger_break();
   }
 else if (PSA_SUCCESS != psa_destroy_key(key_handle))
   {
 /* Destroying the key handle failed */
      debugger_break();
   }
 else
   {
 /* All the operations succeeded */
   }
 /* Close the SCE */
 mbedtls_platform_teardown(&ctx);
}
```

### ECC Example

This is an example on using the PSA Crypto API to generate an ECC-P256R1 key, signing and verifying data after hashing it first using SHA256.

*Note*

*Unlike RSA, ECDSA does not have any padding schemes. Thus the hash argument for the ECC sign operation MUST have a size larger than or equal to the curve size; i.e. for PSA_ECC_CURVE_SECP256R1 the payload size must be at least 256/8 bytes. nist.fips.186-4: " A hash function that provides a lower security strength than the security strength associated with the bit length of 'n' ordinarily should not be used, since this would reduce the security strength of the digital signature process to a level no greater than that provided by the hash function."*

```
#define ECC_256_BIT_LENGTH 256
#define ECC_256_EXPORTED_SIZE 500
uint8_t exportedECC_SECP256R1Key[ECC_256_EXPORTED_SIZE];
size_t exportedECC_SECP256R1Keylength = 0;
void psa_ecc256R1_example (void)
{
/* This example generates an ECC-P256R1 keypair, performs signing and verification
operations.
 * It then exports the generated key into ASN1 DER format to a RAM array which can
then be programmed to flash.
 * It then re-imports that key, and performs signing and verification operations. */
```

```
 unsigned char            payload[] = "ASYMMETRIC_INPUT_FOR_SIGN......";
 unsigned char            signature1[PSA_SIGNATURE_MAX_SIZE] = {0};
 unsigned char            signature2[PSA_SIGNATURE_MAX_SIZE] = {0};
 size_t                   signature_length1 = 0;
 size_t                   signature_length2 = 0;
    psa_key_attributes_t    attributes       = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_attributes_t    read_attributes   = PSA_KEY_ATTRIBUTES_INIT;
 mbedtls_platform_context ctx               = {0};
    psa_key_handle_t         ecc_key_handle    = {0};
    psa_hash_operation_t hash_operation = {0};
    uint8_t             payload_hash[PSA_HASH_MAX_SIZE];
 size_t               payload_hash_len;
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_SIGN_HASH |
PSA_KEY_USAGE_VERIFY_HASH | PSA_KEY_USAGE_EXPORT);
    psa_set_key_algorithm(&attributes, PSA_ALG_ECDSA(PSA_ALG_SHA_256));
    psa_set_key_type(&attributes, PSA_KEY_TYPE_ECC_KEY_PAIR(PSA_ECC_CURVE_SECP_R1));
    psa_set_key_bits(&attributes, ECC_256_BIT_LENGTH);
 /* To use wrapped keys instead of plaintext:
  * - Use PSA_KEY_LIFETIME_VOLATILE_WRAPPED or PSA_KEY_LIFETIME_PERSISTENT_WRAPPED.
  * - To use persistent keys:
  * - The file system must be initialized prior to calling the generate/import key
functions.
  * - Refer to the littlefs example to see how to format and mount the filesystem. */
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
 /* Generate ECC P256R1 Key pair */
```

```
 if (PSA_SUCCESS != psa_generate_key(&attributes, &ecc_key_handle))
    {
       debugger_break();
    }
 /* Test the key information */
 if (PSA_SUCCESS != psa_get_key_attributes(ecc_key_handle, &read_attributes))
    {
       debugger_break();
    }
 /* Calculate the hash of the message */
 if (PSA_SUCCESS != psa_hash_setup(&hash_operation, PSA_ALG_SHA_256))
    {
       debugger_break();
    }
 if (PSA_SUCCESS != psa_hash_update(&hash_operation, payload, sizeof(payload)))
    {
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_hash_finish(&hash_operation, &payload_hash[0], sizeof(payload_hash),
&payload_hash_len))
    {
       debugger_break();
    }
 /* Sign message using the private key
  * NOTE: The hash argument (payload_hash here) MUST have a size equal to the curve
size;
  * i.e. for SECP256R1 the payload size must be 256/8 bytes.
  * Similarly for SECP384R1 the payload size must be 384/8 bytes.
  * nist.fips.186-4: " A hash function that provides a lower security strength than
  * the security strength associated with the bit length of 'n' ordinarily should not
be used, since this
  * would reduce the security strength of the digital signature process to a level no
greater than that
```

```
 * provided by the hash function." */
 if (PSA_SUCCESS !=
       psa_sign_hash(ecc_key_handle, PSA_ALG_ECDSA(PSA_ALG_SHA_256), payload_hash,
payload_hash_len, signature1,
 sizeof(signature1), &signature_length1))
    {
       debugger_break();
    }
 /* Verify the signature1 using the public key */
 if (PSA_SUCCESS !=
       psa_verify_hash(ecc_key_handle, PSA_ALG_ECDSA(PSA_ALG_SHA_256), payload_hash,
payload_hash_len, signature1,
                       signature_length1))
    {
       debugger_break();
    }
 /* Export the key. The exported key can then be save to flash for later usage. */
 if (PSA_SUCCESS !=
       psa_export_key(ecc_key_handle, exportedECC_SECP256R1Key, sizeof
(exportedECC_SECP256R1Key),
                      &exportedECC_SECP256R1Keylength))
    {
       debugger_break();
    }
 /* Destroy the key and handle */
 if (PSA_SUCCESS != psa_destroy_key(ecc_key_handle))
    {
       debugger_break();
    }
 /* Import the previously exported key pair */
 if (PSA_SUCCESS !=
       psa_import_key(&attributes, exportedECC_SECP256R1Key,
exportedECC_SECP256R1Keylength, &ecc_key_handle))
    {
```

```
        debugger_break();
    }
 /* Sign message using the private key */
 if (PSA_SUCCESS !=
        psa_sign_hash(ecc_key_handle, PSA_ALG_ECDSA(PSA_ALG_SHA_256), payload_hash,
payload_hash_len, signature2,
 sizeof(signature2), &signature_length2))
    {
        debugger_break();
    }
 /* Verify signature2 using the public key */
 if (PSA_SUCCESS !=
        psa_verify_hash(ecc_key_handle, PSA_ALG_ECDSA(PSA_ALG_SHA_256), payload_hash,
payload_hash_len, signature2,
                        signature_length2))
    {
        debugger_break();
    }
 /* Signatures cannot be compared since ECC signatures vary for the same data unless
Deterministic ECC is used which is not supported by the HW.
  * Only the verification operation can be used to validate signatures. */
}
```

### RSA Example

This is an example on using the PSA Crypto API to generate an RSA2048 key, encrypting and decrypting multi-block data and using PKCS7 padding.

```
#define RSA_2048_BIT_LENGTH 2048
#define RSA_2048_EXPORTED_SIZE 1210
/* The RSA 2048 key pair export in der format is roughly as follows
 * RSA private keys:
 * RSAPrivateKey ::= SEQUENCE { ---------------------------------------- 1 + 3
 * version Version, ---------------------------------------- 1 + 1 + 1
 * modulus INTEGER, ---------------- n ------------------- 1 + 3 + 256 + 1
```

```
 * publicExponent INTEGER, ---------------- e ------------------- 1 + 4
 * privateExponent INTEGER, ---------------- d ------------------- 1 + 3 + 256 (276
for Wrapped)
 * prime1 INTEGER, ---------------- p ------------------- 1 + 3 + (256 / 2)
 * prime2 INTEGER, ---------------- q ------------------- 1 + 3 + (256 / 2)
 * exponent1 INTEGER, ---------------- d mod (p-1) --------- 1 + 2 + (256 / 2) (4 for
Wrapped)
 * exponent2 INTEGER, ---------------- d mod (q-1) --------- 1 + 2 + (256 / 2) (4 for
Wrapped)
 * coefficient INTEGER, ---------------- (inverse of q) mod p - 1 + 2 + (256 / 2) (4
for Wrapped)
 * otherPrimeInfos OtherPrimeInfos OPTIONAL ------------------------ 0 (not
supported)
 * }
 */
uint8_t exportedRSA2048Key[RSA_2048_EXPORTED_SIZE];
size_t exportedRSA2048Keylength = 0;
void psa_rsa2048_example (void)
{
/* This example generates an RSA2048 keypair, performs signing and verification
operations.
 * It then exports the generated key into ASN1 DER format to a RAM array which can
then be programmed to flash.
 * It then re-imports that key, and performs signing and verification operations. */
 mbedtls_platform_context ctx       = {0};
    psa_key_handle_t         key_handle = {0};
 unsigned char            payload[] = "ASYMMETRIC_INPUT_FOR_SIGN";
 unsigned char            signature1[PSA_SIGNATURE_MAX_SIZE] = {0};
 unsigned char            signature2[PSA_SIGNATURE_MAX_SIZE] = {0};
 size_t               signature_length1 = 0;
 size_t               signature_length2 = 0;
    psa_key_attributes_t attributes       = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_attributes_t read_attributes   = PSA_KEY_ATTRIBUTES_INIT;
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
```

```
    {
        debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
        debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_SIGN_HASH |
PSA_KEY_USAGE_VERIFY_HASH | PSA_KEY_USAGE_EXPORT);
    psa_set_key_algorithm(&attributes, PSA_ALG_RSA_PKCS1V15_SIGN_RAW);
    psa_set_key_type(&attributes, PSA_KEY_TYPE_RSA_KEY_PAIR);
    psa_set_key_bits(&attributes, RSA_2048_BIT_LENGTH);
 /* To use wrapped keys instead of plaintext:
  * - Use PSA_KEY_LIFETIME_VOLATILE_WRAPPED or PSA_KEY_LIFETIME_PERSISTENT_WRAPPED.
  * - To use persistent keys:
  * - The file system must be initialized prior to calling the generate/import key
functions.
  * - Refer to the littlefs example to see how to format and mount the filesystem. */
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
 /* Generate RSA 2048 Key pair */
 if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
    {
        debugger_break();
    }
 /* Test the key information */
 if (PSA_SUCCESS != psa_get_key_attributes(key_handle, &read_attributes))
    {
        debugger_break();
    }
 /* Sign message using the private key */
 if (PSA_SUCCESS !=
        psa_sign_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload, sizeof
(payload), signature1,
```

```
 sizeof(signature1), &signature_length1))
    {
       debugger_break();
    }
 /* Verify the signature1 using the public key */
 if (PSA_SUCCESS !=
       psa_verify_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload,
sizeof(payload), signature1,
                        signature_length1))
    {
       debugger_break();
    }
 /* Export the key */
 if (PSA_SUCCESS !=
       psa_export_key(key_handle, exportedRSA2048Key, sizeof(exportedRSA2048Key),
&exportedRSA2048Keylength))
    {
       debugger_break();
    }
 /* Destroy the key and handle */
 if (PSA_SUCCESS != psa_destroy_key(key_handle))
    {
       debugger_break();
    }
 /* Import the previously exported key pair */
 if (PSA_SUCCESS != psa_import_key(&attributes, exportedRSA2048Key,
exportedRSA2048Keylength, &key_handle))
    {
       debugger_break();
    }
 /* Sign message using the private key */
 if (PSA_SUCCESS !=
       psa_sign_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload, sizeof
(payload), signature2,
```

```
 sizeof(signature2), &signature_length2))
    {
       debugger_break();
    }
 /* Verify signature2 using the public key */
 if (PSA_SUCCESS !=
       psa_verify_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload,
sizeof(payload), signature2,
                        signature_length2))
    {
       debugger_break();
    }
 /* Compare signatures to verify that the same signature was generated */
 if (0 != memcmp(signature2, signature1, signature_length2))
    {
       debugger_break();
    }
    mbedtls_psa_crypto_free();
 mbedtls_platform_teardown(&ctx);
}
```

## Function Documentation

### ◆ RM_PSA_CRYPTO_TRNG_Read()

fsp_err_t RM_PSA_CRYPTO_TRNG_Read ( uint8_t *const *p_rngbuf*, uint32_t *num_req_bytes*, uint32_t * *p_num_gen_bytes* )

Reads requested length of random data from the TRNG. Generate nbytes of random bytes and store them in p_rngbuf buffer.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Random number generation successful |
| FSP_ERR_ASSERTION | NULL input parameter(s). |
| FSP_ERR_CRYPTO_UNKNOWN | An unknown error occurred. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- s_generate_16byte_random_data

### ◆ mbedtls_platform_setup()

int mbedtls_platform_setup ( mbedtls_platform_context * *ctx*)

This function initializes the SCE and the TRNG. It **must** be invoked before the crypto library can be used. This implementation is used if MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT is defined.

Example:

```
mbedtls_platform_context ctx = {0};

/* Setup the platform; initialize the SCE and the TRNG */

if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
```

**Return values**

| | |
|---|---|
| 0 | Initialization was successful. |
| MBEDTLS_ERR_PLATFORM_HW_ACCEL_FAILED | SCE Initialization error. |

### ◆ mbedtls_platform_teardown()

void mbedtls_platform_teardown ( mbedtls_platform_context * *ctx*)

This implementation is used if MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT is defined. It is intended to de-initialize any items that were initialized in the mbedtls_platform_setup() function, but currently is only a placeholder function.

Example:

```
/* De-initialize the platform. This is currently a placeholder function which does
not do anything. */

mbedtls_platform_teardown(&ctx);
```

**Return values**

| | |
|---|---|
| N/A | |

## 4.2.65 Capacitive Touch Middleware (rm_touch)

Modules

**Functions**

| | |
|---|---|
| fsp_err_t | RM_TOUCH_Open (touch_ctrl_t *const p_ctrl, touch_cfg_t const *const p_cfg) |

| | |
|---|---|
| | Opens and configures the TOUCH Middle module. Implements touch_api_t::open. More... |
| fsp_err_t | RM_TOUCH_ScanStart (touch_ctrl_t *const p_ctrl)<br>This function should be called each time a periodic timer expires. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Before starting the next scan, first get the data with RM_TOUCH_DataGet(). If a different control block scan should be run, check the scan is complete before executing. Implements touch_api_t::scanStart. More... |
| fsp_err_t | RM_TOUCH_DataGet (touch_ctrl_t *const p_ctrl, uint64_t *p_button_status, uint16_t *p_slider_position, uint16_t *p_wheel_position)<br>Gets the 64-bit mask indicating which buttons are pressed. Also, this function gets the current position of where slider or wheel is being pressed. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Implements touch_api_t::dataGet. More... |
| fsp_err_t | RM_TOUCH_Close (touch_ctrl_t *const p_ctrl)<br>Disables specified TOUCH control block. Implements touch_api_t::close. More... |
| fsp_err_t | RM_TOUCH_VersionGet (fsp_version_t *const p_version) |

### Detailed Description

This module supports the Capacitive Touch Sensing Unit (CTSU). It implements the Touch Middleware Interface.

# Overview

The Touch Middleware uses the Capacitive Touch Sensing Unit (r_ctsu) API and provides application-level APIs for scanning touch buttons, sliders, and wheels. This module is configured via the QE for Capacitive Touch.

### Features

- Supports touch buttons (Self and Mutual), sliders, and wheels
- Can retrieve the status of up to 64 buttons at once
- Software and external triggering
- Callback on scan end
- Collects and calculates usable scan results:
    - Slider position from 0 to 100 (percent)

    - Wheel position from 0 to 359 (degrees)
- Optional (build time) support for real-time monitoring functionality through the QE tool over UART

# Configuration

*Note*

***This module is configured via the QE for Capacitive Touch. For information on how to use the QE tool, once the tool is installed click Help -> Help Contents in e2 studio and search for "QE".***
*Multiple configurations can be defined within a single project allowing for different scan procedures or button layouts.*

### Build Time Configurations for rm_touch

The following build time configurations are defined in fsp_cfg/rm_touch_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Support for QE monitoring using UART | • Enabled<br>• Disabled | Disabled | Enable SCI_UART support for QE monitoring. |

### Configurations for Middleware > CapTouch > TOUCH Driver on rm_touch

This module can be added to the Stacks tab via New Stack > Middleware > CapTouch > TOUCH Driver on rm_touch:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Interrupt Configuration

Refer to the Capacitive Touch Sensing Unit (r_ctsu) section for details.

### Clock Configuration

Refer to the Capacitive Touch Sensing Unit (r_ctsu) section for details.

### Pin Configuration

Refer to the Capacitive Touch Sensing Unit (r_ctsu) section for details.

# Usage Notes

### Sliders and Wheels

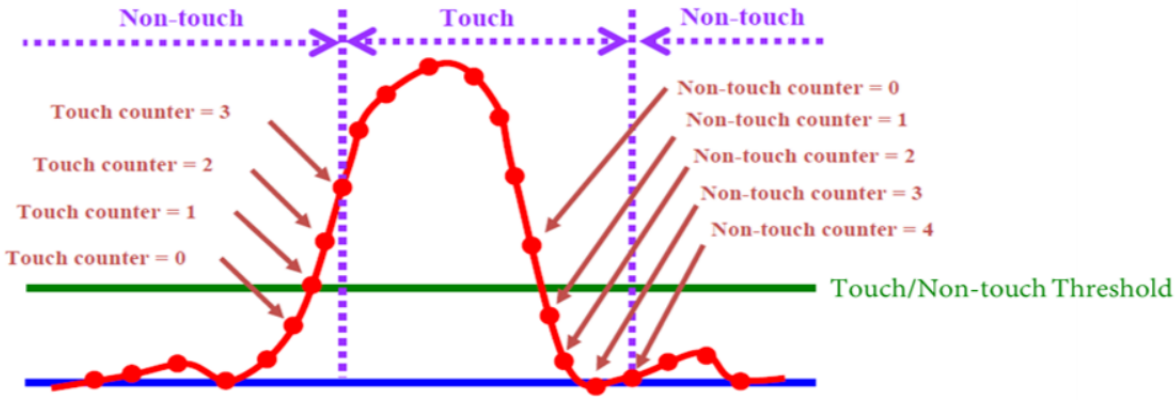
Sliders and wheels are subject so some limitations:

| | Slider | Wheel |
|---|---|---|

| | | |
|---|---|---|
| Electrode type | Self capacitance only | Self capacitance only |
| Number of electrodes | 4+ | 3-5 |
| Touch position output range | 0-100 | 0-359 |
| Default value (no touch) | 0xFFFF | 0xFFFF |

### Touch Judgement

Touch data is judged as touched or not-touched based on the threshold and hysteresis values determined during the QE tool tuning process. Refer to the QE for Capacitive Touch tool documentation in e2 studio Help for details on how these values are set.




| Measurement Count | Initialization | n | n+1 | n+2 | n+3 | n+4 | n+5 | n+6 | n+7 | n+8 | n+9 | n+10 | n+11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| User-set touch/non-touch | Non-touch | | | Touch | | | | | Non-touch | | | | |
| Touch counter buffer | 0 | 0 | 0 | 1 | 2 | 3 | 3 | 3 | 0 | 0 | 0 | 0 | 0 |
| Non-touch counter buffer | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 3 | 4 | 4 |
| Touch/Non-touch judgement buffer | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| Actual touch/Non-touch judgement | Non-touch | | | | | Touch | | | | | | Non-touch | |

Figure 143: Touch/Non-touch judgement Image

# Examples

### Basic Example

This is a basic example of minimal use of the TOUCH in an application.

```
void touch_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    err = RM_TOUCH_Open(&g_touch_ctrl, &g_touch_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
```

```
 while (true)
    {
 RM_TOUCH_ScanStart(&g_touch_ctrl);
 while (0 == g_flag)
      {
 /* Wait scan end callback */
       }
       g_flag = 0;
       err = RM_TOUCH_DataGet(&g_touch_ctrl, &button, slider, wheel);
 if (FSP_SUCCESS == err)
      {
 /* Application specific data processing. */
       }
    }
}
```

### Multi Mode Example

This is a optional example of using both Self-capacitance and Mutual-capacitance. Refer to the Multi Mode Example in CTSU usage notes.

```
void touch_optional_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    err = RM_TOUCH_Open(&g_touch_ctrl, &g_touch_cfg);
    handle_error(err);
    err = RM_TOUCH_Open(&g_touch_ctrl_mutual, &g_touch_cfg_mutual);
    handle_error(err);
 while (true)
    {
 RM_TOUCH_ScanStart(&g_touch_ctrl);
 while (0 == g_flag)
      {
 /* Wait scan end callback */
       }
```

```
      g_flag = 0;
RM_TOUCH_ScanStart(&g_touch_ctrl_mutual);
while (0 == g_flag)
     {
/* Wait scan end callback */
      }
      g_flag = 0;
      err = RM_TOUCH_DataGet(&g_touch_ctrl, &button, slider, wheel);
if (FSP_SUCCESS == err)
     {
/* Application specific data processing. */
      }
      err = RM_TOUCH_DataGet(&g_touch_ctrl_mutual, &button, slider, wheel);
if (FSP_SUCCESS == err)
     {
/* Application specific data processing. */
      }
   }
}
```

## Data Structures

| | |
|---|---|
| struct | touch_button_info_t |
| struct | touch_slider_info_t |
| struct | touch_wheel_info_t |
| struct | touch_instance_ctrl_t |

## Data Structure Documentation

### ◆ touch_button_info_t

| struct touch_button_info_t | | |
|---|---|---|
| Information of button | | |
| Data Fields | | |
| uint64_t | status | Touch result bitmap. |
| uint16_t * | p_threshold | Pointer to Threshold value array. |

| | | |
|---|---|---|
| | | g_touch_button_threshold[] is set by Open API. |
| uint16_t * | p_hysteresis | Pointer to Hysteresis value array. g_touch_button_hysteresis[] is set by Open API. |
| uint16_t * | p_reference | Pointer to Reference value array. g_touch_button_reference[] is set by Open API. |
| uint16_t * | p_on_count | Continuous touch counter. g_touch_button_on_count[] is set by Open API. |
| uint16_t * | p_off_count | Continuous non-touch counter. g_touch_button_off_count[] is set by Open API. |
| uint32_t * | p_drift_buf | Drift reference value. g_touch_button_drift_buf[] is set by Open API. |
| uint16_t * | p_drift_count | Drift counter. g_touch_button_drift_count[] is set by Open API. |
| uint8_t | on_freq | Copy from config by Open API. |
| uint8_t | off_freq | Copy from config by Open API. |
| uint16_t | drift_freq | Copy from config by Open API. |
| uint16_t | cancel_freq | Copy from config by Open API. |

### ◆ touch_slider_info_t

| struct touch_slider_info_t | | |
|---|---|---|
| Information of slider | | |
| Data Fields | | |
| uint16_t * | p_position | Calculated Position data. g_touch_slider_position[] is set by Open API. |
| uint16_t * | p_threshold | Copy from config by Open API. g_touch_slider_threshold[] is set by Open API. |

### ◆ touch_wheel_info_t

| struct touch_wheel_info_t | | |
|---|---|---|
| Information of wheel | | |
| Data Fields | | |
| uint16_t * | p_position | Calculated Position data. |

| | | |
|---|---|---|
| | | g_touch_wheel_position[] is set by Open API. |
| uint16_t * | p_threshold | Copy from config by Open API. g_touch_wheel_threshold[] is set by Open API. |

### ◆ touch_instance_ctrl_t

| struct touch_instance_ctrl_t | | |
|---|---|---|
| TOUCH private control block. DO NOT MODIFY. Initialization occurs when RM_TOUCH_Open() is called. | | |
| Data Fields | | |
| uint32_t | open | Whether or not driver is open. |
| touch_button_info_t | binfo | Information of button. |
| touch_slider_info_t | sinfo | Information of slider. |
| touch_wheel_info_t | winfo | Information of wheel. |
| touch_cfg_t const * | p_touch_cfg | Pointer to initial configurations. |
| ctsu_instance_t const * | p_ctsu_instance | Pointer to CTSU instance. |

## Function Documentation

### ◆ RM_TOUCH_Open()

fsp_err_t RM_TOUCH_Open ( touch_ctrl_t *const *p_ctrl*, touch_cfg_t const *const *p_cfg* )

Opens and configures the TOUCH Middle module. Implements touch_api_t::open.

Example:

```
err = RM_TOUCH_Open(&g_touch_ctrl, &g_touch_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | TOUCH successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Configuration parameter error. |

### ◆ RM_TOUCH_ScanStart()

fsp_err_t RM_TOUCH_ScanStart ( touch_ctrl_t *const *p_ctrl*)

This function should be called each time a periodic timer expires. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Before starting the next scan, first get the data with RM_TOUCH_DataGet(). If a different control block scan should be run, check the scan is complete before executing. Implements touch_api_t::scanStart.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CTSU_SCANNING | Scanning this instance or other. |
| FSP_ERR_CTSU_NOT_GET_DATA | The previous data has not been retrieved by DataGet. |

### ◆ RM_TOUCH_DataGet()

fsp_err_t RM_TOUCH_DataGet ( touch_ctrl_t *const *p_ctrl*, uint64_t * *p_button_status*, uint16_t * *p_slider_position*, uint16_t * *p_wheel_position* )

Gets the 64-bit mask indicating which buttons are pressed. Also, this function gets the current position of where slider or wheel is being pressed. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Implements touch_api_t::dataGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data decoded. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CTSU_SCANNING | Scanning this instance. |
| FSP_ERR_CTSU_INCOMPLETE_TUNING | Incomplete initial offset tuning. |

### ◆ RM_TOUCH_Close()

| fsp_err_t RM_TOUCH_Close ( touch_ctrl_t *const *p_ctrl*) |
|---|
| Disables specified TOUCH control block. Implements touch_api_t::close.<br><br>**Return values**<br>FSP_SUCCESS — Successfully closed.<br>FSP_ERR_ASSERTION — Null pointer passed as a parameter.<br>FSP_ERR_NOT_OPEN — Module is not open. |

### ◆ RM_TOUCH_VersionGet()

| fsp_err_t RM_TOUCH_VersionGet ( fsp_version_t *const *p_version*) |
|---|
| Return TOUCH Middle module version. Implements touch_api_t::versionGet.<br><br>**Return values**<br>FSP_SUCCESS — Version information successfully read.<br>FSP_ERR_ASSERTION — Null pointer passed as a parameter |

## 4.2.66 AWS Device Provisioning

Modules

AWS Device Provisioning example software.

# Overview

### Terminology

The terminology defined below will be used in the following sections.

| Term | Description |
|---|---|
| Service Provider | Entity that provides the cloud infrastructure and associated services, for example, AWS/Azure. |
| Device Manufacturer | Entity that provides the MCU, for example, Renesas. |

| | |
|---|---|
| OEM | Entity that uses the MCU to create a product. |
| Customer | End user of OEM product. |

**Device ID**

For systems that intend to use Public Key Certificate (PKC), the Device ID is in the form of a key pair (RSA or ECC). A PKC comprises of a **public key**, metadata, and finally a signature over all that. This signature is generated by the entity that issues the certificate and is known as a CA (Certificate Authority). The most common format for a public certificate is the X.509 format which is typically PEM (base 64) encoded such that the certificate is human-readable. It can also be DER encoded which is binary encoding and thus not human readable. The **public key** portion of the Device ID is used for the Device Certificate.

**Provisioning**

Device Provisioning refers to the process by which a service provider links a certificate to a Device ID and thus a device. Depending on the provisioning model, an existing certificate from the device may be used or a new one will be issued at this stage.  Provisioning (also referred to as Registration) occurs with respect to a particular service provider, for example, AWS or Azure. It is necessary that the certificate is issued by the service provider or a CA known to those providers. When a device is provisioned with AWS for example, the AWS IoT service associates the Device ID (and thus the device) with a specific certificate. The certificate will be programmed into the device and for all future transactions with AWS, the certificate will be used as the means of identifying the device. The public and private key are also stored on the MCU.

**Provisioning Models**

Provisioning services vary between service providers. There are essentially three general provisioning models.

1. Provisioning happens on the production line. This requires the provisioning Infrastructure to be present on the production line. This is the most secure model, but is expensive.
2. Devices are programmed with a shared credential that is linked into the code at build time and the provisioning occurs when a customer uses the device for the first time. The shared credential and a unique device serial number are used to uniquely identify the device during the provisioning process. So long as the product only has the shared credential, it will only operate with limited (as defined by certificate policy) functionality .Once the provisioning is done, then the device will be fully functional. This is the most common use case for consumer products where no sensitive information is being transmitted. AWS provides an example of this model.
3. Devices have no identity programmed in the factory; provisioning occurs through some other device like a smartphone which is already trusted by the service provider.

In all these cases, the Device Identity

1. Is unique to the device
2. Must have restricted access within the device
3. Can be used to issue more than one certificate and the certificates themselves have to be updatable in the field.

AWS uses the PKCS11 API to erase, store and retrieve certificates. These PKCS11 functions (Write, Read and Erase) are separated out into a Physical Abstraction Layer (PAL) which the OEM/Device Manufacturer is expected to implement for the type of memory that they intend to use. The internal rm_aws_pkcs11_pal module implements these requirements on RA MCU data flash.

# AWS Provisioning Example

AWS provides an **example** implementation to support device provisioning. This implementation uses the PKCS11 API to store device credentials into the PKCS11 defined memory. The implementation (aws_dev_mode_key_provisioning.c) exposes two functions:

1. vDevModeKeyProvisioning()
2. vAlternateKeyProvisioning()

Both of these functions require that the device credentials be provided in PEM format. Using either of these example functions as is in production is not recommended; but vAlternateKeyProvisioning() provides more flexibility because of the ability to provide credentials as arguments.

Credentials can be created as follows:

- Create your own CA and use that to generate the device certificate. This CA will have to be registered with the service provider with which the product will be used, for example Register your CA with AWS.
- Use AWS to generate the device certificate.

# Examples

### Basic Example

This is a basic example of provisioning a device using the AWS demo implementation.

```
#define keyCLIENT_CERTIFICATE_PEM \
  "-----BEGIN CERTIFICATE-----\n" \
  "MIIDETCCAfkCFHwd2yn8zn5qB2ChYUT9Mvbi9Xp1MA0GCSqGSIb3DQEBCwUAMEUx\n" \
  "CzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRl\n" \
  "cm5ldCBXaWRnaXRzIFB0eSBMdGQwHhcNMTkwOTExMjEyMjU0WhcNMjAwOTEwMjEy\n" \
  "MjU0WjBFMQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UE\n" \
  "CgwYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMIIBIjANBgkqhkiG9w0BAQEFAAOC\n" \
  "AQ8AMIIBCgKCAQEAo8oThJXSMDo41oL7HTpC4TX8NalBvnkFw30Av67dl/oZDjVA\n" \
  "iXPnZkhVppLnj++0/Oed0M7UwNUO2nurQt6yTYrvW7E8ZPjAlC7ueJcGYZhOaVv2\n" \
  "bhSmigjFQru2lw5odSuYy5+22CCgxft58nrRCo5Bk+GwWgZmcrxe/BzutRHQ7X4x\n" \
  "dYJhyhBOi2R1Kt8XsbuWilfgfkVhhkVklFeKqiypdQM6cnPWo/G4DyW34jOXzzEM\n" \
  "FLWvQOQLCKUZOgjJBnFdbx8oOOwMkYCChbV7gqPE6cw0Zy26CvlLQiINyonLPbNT\n" \
  "c64sS/ZBGPZFOPJmb4tG2nipYgZ1hO/r++jCbwIDAQABMA0GCSqGSIb3DQEBCwUA\n" \
  "A4IBAQCdqq59ubdRY9EiV3bleKXeqG7+8HgBHdm0X9dgq10nD37p00YLyuZLE9NM\n" \
  "066G/VcflGrx/Nzw+/UuI7/UuBbBS/3ppHRnsZqBIl8nnr/ULrFQy8z3vKtL1q3C\n" \
  "DxabjPONlPO2keJeTTA71N/RCEMwJoa8i0XKXGdu/hQo6x4n+Gq73fEiGCl99xsc\n" \
  "4tIO4yPS4lv+uXBzEUzoEy0CLIkiDesnT5lLeCyPmUNoU89HU95IusZT7kygCHHd\n" \
```

```
  "72am1ic3X8PKc268KT3ilr3VMhK67C+iIIkfrM5AiU+oOIRrIHSC/p0RigJg3rXA\n" \
  "GBIRHvt+OYF9fDeG7U4QDJNCfGW+\n" \
  "-----END CERTIFICATE-----"
#define keyCLIENT_PRIVATE_KEY_PEM \
  "-----BEGIN RSA PRIVATE KEY-----\n" \
  "MIIEowIBAAKCAQEAo8oThJXSMDo41oL7HTpC4TX8NalBvnkFw30Av67dl/oZDjVA\n" \
  "iXPnZkhVppLnj++0/Oed0M7UwNUO2nurQt6yTYrvW7E8ZPjAlC7ueJcGYZhOaVv2\n" \
  "bhSmigjFQru2lw5odSuYy5+22CCgxft58nrRCo5Bk+GwWgZmcrxe/BzutRHQ7X4x\n" \
  "dYJhyhBOi2R1Kt8XsbuWilfgfkVhhkVklFeKqiypdQM6cnPWo/G4DyW34jOXzzEM\n" \
  "FLWvQOQLCKUZOgjJBnFdbx8oOOwMkYCChbV7gqPE6cw0Zy26CvlLQiINyonLPbNT\n" \
  "c64sS/ZBGPZFOPJmb4tG2nipYgZ1hO/r++jCbwIDAQABAoIBAQCGR2hC/ZVJhqIM\n" \
  "c2uuJZKpElpIIBBPOObZwwS3IYR4UUjzVgMn7UbbmxflLXD8lzfZU4YVp0vTH5lC\n" \
  "07qvYuXpHqtnj+GEok837VYCtUY9AuHeDM/2paV3awNV15E1PFG1Jd3pqnH7tJw6\n" \
  "VBZBDiGNNt1agN/UnoSlMfvpU0r8VGPXCBNxe3JY5QyBJPI1wF4LcxRI+eYmr7Ja\n" \
  "/cjn97DZotgz4B7gUNu8XIEkUOTwPabZINY1zcLWiXTMA+8qTniPVk653h14Xqt4\n" \
  "4o4D4YCTpwJcmxSV1m21/6+uyuXr9SIKAE+Ys2cYLA46x+rwLaW5fUoQ5hHa0Ytb\n" \
  "RYJ4SrtBAoGBANWtwlE69N0hq5xDPckSbNGubIeG8P4mBhGkJxIqYoqugGLMDiGX\n" \
  "4bltrjr2TPWaxTo3pPavLJiBMIsENA5KU+c/r0jLkxgEp9MIVJrtNgkCiDQqogBG\n" \
  "j4IJL2iQwXoLCqk2tx/dh9Mww+7SETE7EPNrv4UrYaGN5AEvpf5W+NHPAoGBAMQ6\n" \
  "wVa0Mx1PlA4enY2rfE3WXP8bzjleSOwR75JXqG2WbPC0/cszwbyPWOEqRpBZfvD/\n" \
  "QFkKx06xp1C09XwiQanr2gDucYXHeEKg/9iuJV1UkMQp95ojlhtSXdRZV7/l4pmN\n" \
  "fpB2vcAptX/4gY4tDrWMO08JNnRjE7duC+rmmk1hAoGAS4L0QLCNB/h2JOq+Uuhn\n" \
  "/FGfmOVfFPFrA6D3DbxcxpWUWVWzSLvb0SOphryzxbfEKyau7V5KbDp7ZSU/IC20\n" \
  "KOygjSEkAkDi7fjrrTRW/Cgg6g6G4YIOBO4qCtHdDbwJMHNdk6096qw5EZS67qLp\n" \
  "Apz5OZ5zChySjri/+HnTxJECgYBysGSP6IJ3fytplTtAshnU5JU2BWpi3ViBoXoE\n" \
  "bndilajWhvJO8dEqBB5OfAcCF0y6TnWtlT8oH21LHnjcNKlsRw0Dvllbd1oylybx\n" \
  "3da41dRG0sCEtoflMB7nHdDLt/DZDnoKtVvyFG6gfP47utn+Ahgn+Zp6K+46J3eP\n" \
  "s3g8AQKBgE/PJiaF8pbBXaZOuwRRA9GOMSbDIF6+jBYTYp4L9wk4+LZArKtyI+4k\n" \
  "Md2DUvHwMC+ddOtKqjYnLm+V5cSbvu7aPvBZtwxghzTUDcf7EvnA3V/bQBh3R0z7\n" \
  "pVsxTyGRmBSeLdbUWACUbX9LXdpudarPAJ59daWmP3mBEVmWdzUw\n" \
  "-----END RSA PRIVATE KEY-----"
void device_provisioning_example (void)
{
 /* Initialize the crypto hardware acceleration. */
```

```
 mbedtls_platform_setup(NULL);
    ProvisioningParams_t params;
 /* Provision device with provided credentials. The provided credentials are written
to data flash.
  * In production, the credentials can be provided over a comms channel instead of
being linked into the image.
  * The same example provisioning function, vAlternateKeyProvisioning, can be used in
that case. */
    params.pucClientPrivateKey       = (uint8_t *) keyCLIENT_PRIVATE_KEY_PEM;
    params.pucClientCertificate      = (uint8_t *) keyCLIENT_CERTIFICATE_PEM;
    params.ulClientPrivateKeyLength = 1 + strlen((const char *)
params.pucClientPrivateKey);
    params.ulClientCertificateLength = 1 + strlen((const char *)
params.pucClientCertificate);
    params.pucJITPCertificate       = NULL;
    params.ulJITPCertificateLength   = 0;
    vAlternateKeyProvisioning(&params);
}
```

# Limitations

The provisioning code is an example provided by AWS. It must be modified to meet product requirements.

### 4.2.67 AWS MQTT

Modules

This module provides the AWS MQTT integration documentation.

# Overview

The AWS MQTT library can connect to either AWS or a third party MQTT broker such as Mosquitto. The documentation for the library can be found on the AWS IoT Device SDK C: MQTT website.

**Features**

- MQTT connections over TLS to an AWS IoT Endpoint or Mosquitto server
- Unsecure MQTT connections to Mosquitto servers. This is not recommended for production and should only be done to a local server for testiing.

# Configuration

### Memory Usage

The AWS MQTT library relies heavily on dynamic memory allocation for thread/task creation as well as other uses. To accomodate this it is recommended to increase the heap to 64k or tweak the thread stack configuration values. Noteable values are:

### AWS IoT Common

- IoT Thread Default Stack Size
- IoT Network Receive Task Stack Size

### FreeRTOS Thread

- General|Minimal Stack Size

### FreeRTOS Plus TCP

- Stack size in words (not bytes)

# Usage Notes

The AWS MQTT library utilizes a system taskpool to queue up messages. This system task pool must be created before calling into the MQTT library. iot_init.c has been provided for easy initialization of this taskpool via IotSdk_Init().

The AWS MQTT Demo has been provided to easily demonstrate MQTT functionality. An example of initializiing the system taskpool and running the MQTT demo has been provided below.

### Limitations

- aws_clientcredential.h and aws_clientcredential_keys.h need to be added manually.
- The IoT Thread must have a higher priorty than the Network Receive Thread.
- MbedTLS must be initialized and key provisioning must be done before starting a secure connection. Refer to AWS Secure Sockets.

### Examples

### Non-secure connection to a Mosquitto server

```
/* Default IP address configuration. Used in ipconfigUSE_DHCP is set to 0, or

 * ipconfigUSE_DHCP is set to 1 but a DNS server cannot be contacted. */

#define configIP_ADDR0 192

#define configIP_ADDR1 168

#define configIP_ADDR2 0

#define configIP_ADDR3 56
```

```
/* Default gateway IP address configuration. Used in ipconfigUSE_DHCP is set to
 * 0, or ipconfigUSE_DHCP is set to 1 but a DNS server cannot be contacted. */
#define configGATEWAY_ADDR0 192
#define configGATEWAY_ADDR1 168
#define configGATEWAY_ADDR2 0
#define configGATEWAY_ADDR3 1
/* Default DNS server configuration. OpenDNS addresses are 208.67.222.222 and
 * 208.67.220.220. Used in ipconfigUSE_DHCP is set to 0, or ipconfigUSE_DHCP is
 * set to 1 but a DNS server cannot be contacted.*/
#define configDNS_SERVER_ADDR0 208
#define configDNS_SERVER_ADDR1 67
#define configDNS_SERVER_ADDR2 222
#define configDNS_SERVER_ADDR3 222
/* Default netmask configuration. Used in ipconfigUSE_DHCP is set to 0, or
 * ipconfigUSE_DHCP is set to 1 but a DNS server cannot be contacted. */
#define configNET_MASK0 255
#define configNET_MASK1 255
#define configNET_MASK2 255
#define configNET_MASK3 0
/* Define the network addressing. These parameters will be used if either
 * ipconfigUDE_DHCP is 0 or if ipconfigUSE_DHCP is 1 but DHCP auto configuration
 * failed. */
const uint8_t ucIPAddress[4] =
{
    configIP_ADDR0,
    configIP_ADDR1,
    configIP_ADDR2,
    configIP_ADDR3
};
const uint8_t ucNetMask[4] =
{
    configNET_MASK0,
    configNET_MASK1,
    configNET_MASK2,
```

```
    configNET_MASK3
};
const uint8_t ucGatewayAddress[4] =
{
    configGATEWAY_ADDR0,
    configGATEWAY_ADDR1,
    configGATEWAY_ADDR2,
    configGATEWAY_ADDR3
};
const uint8_t ucDNSServerAddress[4] =
{
    configDNS_SERVER_ADDR0,
    configDNS_SERVER_ADDR1,
    configDNS_SERVER_ADDR2,
    configDNS_SERVER_ADDR3
};
/* Use the mac address defined in the lower layer. */
extern uint8_t g_ether0_mac_address[6];
void mqtt_non_secure_example ()
{
 bool connect_to_aws = false;
 const IotNetworkServerInfo_t serverInfo =
    {
       .pHostName = "192.168.0.100",
       .port      = 1883
    };
    IotSdk_Init();
    RunMqttDemo(connect_to_aws, "renesas-iot-demo", (void *) &serverInfo, NULL,
&IotNetworkAfr);
}
```

### Secure connection to a Mosquitto server

*Note*

*MbedTLS must be initialized and key provisioning must be done before starting a secure connection. Refer to AWS*

*Secure Sockets*.

```
static char SERVER_CERTIFICATE_PEM[] = "-----BEGIN CERTIFICATE-----\n"
 "example_certificate_formatting\n"
 "-----END CERTIFICATE-----";
static char CLIENT_CERTIFICATE_PEM[] = "-----BEGIN CERTIFICATE-----\n"
 "example_certificate_formatting\n"
 "-----END CERTIFICATE-----";
static char CLIENT_KEY_PEM[] = "-----BEGIN RSA PRIVATE KEY-----\n"
 "example_certificate_formatting\n"
 "-----END RSA PRIVATE KEY-----";
void mqtt_secure_example ()
{
 bool connect_to_aws = false;
 const IotNetworkServerInfo_t serverInfo =
    {
       .pHostName = "192.168.0.100",
       .port      = 8883
    };
 const IotNetworkCredentials_t afrCredentials =
    {
       .pAlpnProtos       = NULL,
       .maxFragmentLength = 1400,
       .disableSni       = true,
       .pRootCa          = SERVER_CERTIFICATE_PEM,
       .rootCaSize       = sizeof(SERVER_CERTIFICATE_PEM),
       .pClientCert       = CLIENT_CERTIFICATE_PEM,
       .clientCertSize    = sizeof(CLIENT_CERTIFICATE_PEM),
       .pPrivateKey       = CLIENT_KEY_PEM,
       .privateKeySize    = sizeof(CLIENT_KEY_PEM),
    };
    IotSdk_Init();
    RunMqttDemo(connect_to_aws, "renesas-iot-demo", (void *) &serverInfo, (void *)
&afrCredentials, &IotNetworkAfr);
}
```

## 4.2.68 Wifi Middleware (rm_wifi_onchip_silex)

Modules

### Functions

| | |
|---|---|
| fsp_err_t | rm_wifi_onchip_silex_open (wifi_onchip_silex_cfg_t const *const p_cfg) |
| fsp_err_t | rm_wifi_onchip_silex_version_get (fsp_version_t *const p_version) |
| fsp_err_t | rm_wifi_onchip_silex_close () |
| fsp_err_t | rm_wifi_onchip_silex_connect (const char *p_ssid, uint32_t security, const char *p_passphrase) |
| fsp_err_t | rm_wifi_onchip_silex_mac_addr_get (uint8_t *p_macaddr) |
| fsp_err_t | rm_wifi_onchip_silex_scan (WIFIScanResult_t *p_results, uint32_t maxNetworks) |
| fsp_err_t | rm_wifi_onchip_silex_ping (uint8_t *p_ip_addr, uint32_t count, uint32_t interval_ms) |
| fsp_err_t | rm_wifi_onchip_silex_ip_addr_get (uint8_t *p_ip_addr) |
| fsp_err_t | rm_wifi_onchip_silex_avail_socket_get (uint32_t *p_socket_id) |
| fsp_err_t | rm_wifi_onchip_silex_socket_status_get (uint32_t socket_no, uint32_t *p_socket_status) |
| fsp_err_t | rm_wifi_onchip_silex_socket_create (uint32_t socket_no, uint32_t type, uint32_t ipversion) |
| fsp_err_t | rm_wifi_onchip_silex_tcp_connect (uint32_t socket_no, uint32_t ipaddr, uint32_t port) |
| int32_t | rm_wifi_onchip_silex_tcp_send (uint32_t socket_no, const uint8_t *p_data, uint32_t length, uint32_t timeout_ms) |
| int32_t | rm_wifi_onchip_silex_tcp_recv (uint32_t socket_no, uint8_t *p_data, uint32_t length, uint32_t timeout_ms) |
| int32_t | rm_wifi_onchip_silex_tcp_shutdown (uint32_t socket_no, uint32_t shutdown_channels) |
| fsp_err_t | rm_wifi_onchip_silex_socket_disconnect (uint32_t socket_no) |

| | |
|---|---|
| fsp_err_t | rm_wifi_onchip_silex_disconnect () |
| fsp_err_t | rm_wifi_onchip_silex_dns_query (const char *p_textstring, uint8_t *p_ip_addr) |
| fsp_err_t | rm_wifi_onchip_silex_socket_connected (fsp_err_t *p_status) |
| void | rm_wifi_onchip_silex_uart_callback (uart_callback_args_t *p_args) |
| Socket_t | SOCKETS_Socket (int32_t lDomain, int32_t lType, int32_t lProtocol) |
| int32_t | SOCKETS_Connect (Socket_t xSocket, SocketsSockaddr_t *pxAddress, Socklen_t xAddressLength) |
| int32_t | SOCKETS_Recv (Socket_t xSocket, void *pvBuffer, size_t xBufferLength, uint32_t ulFlags) |
| int32_t | SOCKETS_Send (Socket_t xSocket, const void *pvBuffer, size_t xDataLength, uint32_t ulFlags) |
| int32_t | SOCKETS_Shutdown (Socket_t xSocket, uint32_t ulHow) |
| int32_t | SOCKETS_Close (Socket_t xSocket) |
| int32_t | SOCKETS_SetSockOpt (Socket_t xSocket, int32_t lLevel, int32_t lOptionName, const void *pvOptionValue, size_t xOptionLength) |
| uint32_t | SOCKETS_GetHostByName (const char *pcHostName) |
| BaseType_t | SOCKETS_Init (void) |
| uint32_t | ulApplicationGetNextSequenceNumber (uint32_t ulSourceAddress, uint16_t usSourcePort, uint32_t ulDestinationAddress, uint16_t usDestinationPort) |
| WIFIReturnCode_t | WIFI_On (void) |
| WIFIReturnCode_t | WIFI_Off (void) |
| WIFIReturnCode_t | WIFI_ConnectAP (const WIFINetworkParams_t *const pxNetworkParams) |
| WIFIReturnCode_t | WIFI_Disconnect (void) |
| WIFIReturnCode_t | WIFI_Reset (void) |
| WIFIReturnCode_t | WIFI_Scan (WIFIScanResult_t *pxBuffer, uint8_t ucNumNetworks) |

| | |
|---|---|
| WIFIReturnCode_t | WIFI_Ping (uint8_t *pucIPAddr, uint16_t usCount, uint32_t ulIntervalMS) |
| WIFIReturnCode_t | WIFI_GetIP (uint8_t *pucIPAddr) |
| WIFIReturnCode_t | WIFI_GetMAC (uint8_t *pucMac) |
| WIFIReturnCode_t | WIFI_GetHostIP (char *pcHost, uint8_t *pucIPAddr) |
| BaseType_t | WIFI_IsConnected (void) |

### Detailed Description

Wifi and Socket implementation using the Silex SX-ULPGN WiFi module on RA MCUs.

# Overview

This Middleware module supplies an implementation for the FreeRTOS Secure Sockets and WiFi interfaces using the Silex SX-ULPGN module.

The SX-ULPGN is a low-power, compact IEEE 802.11b/g/n 2.4GHz 1x1 Wireless LAN module equipped with the Qualcomm® QCA4010 Wireless SOC. The module comes readily equipped with radio certification for Japan, North America and Europe. More information about this module can be found at the Silex Web Site

### Features

The WiFi Onchip Silex Middleware driver supplies these features:

- Supports connect/disconnect to a b/g/n (2.4GHz) WiFi Access Point using Open, WPA, and WPA2 security. Encryption types can be either TKIP, or CCMP(AES).
- Supports retrieval of the module device MAC address.
- Once connected you can acquire the assigned module device IP.
- Supports a WiFi network scan capability to get a list of local Access Points.
- Supports a Ping function to test network connectivity.
- Supports a DNS Query call to retrieve the IPv4 address of a supplied URL.
- Supports a BSD style Secure Socket interface.
- Drive supports 1 or 2 UARTs for interfacing with the SX-ULPGN module. The second UART is considered optional.

# Configuration

### Build Time Configurations for rm_wifi_onchip_silex

The following build time configurations are defined in fsp_cfg/rm_wifi_onchip_silex_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled | Default (BSP) | If selected code for parameter checking is |

| | | | |
|---|---|---|---|
| | • Disabled | | included in the build. |
| Number of supported socket instances | Refer to the RA Configuration tool for available options. | 1 | Enable number of socket instances |
| Size of RX buffer for socket | Manual Entry | 4096 | |
| Size of TX buffer for CMD Port | Manual Entry | 1500 | |
| Size of RX buffer for CMD Port | Manual Entry | 1500 | |
| Semaphore maximum timeout | Manual Entry | 10000 | |
| Number reties for AT commands | Manual Entry | 10 | |
| Module Reset Port | Refer to the RA Configuration tool for available options. | 06 | Specify the module reset pin port for the MCU. |
| Module Reset Pin | Refer to the RA Configuration tool for available options. | 03 | Specify the module reset pin for the MCU. |

**Configurations for Middleware > WiFi > WiFi Onchip Silex Driver using r_sci_uart**

This module can be added to the Stacks tab via New Stack > Middleware > WiFi > WiFi Onchip Silex Driver using r_sci_uart:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_wifi0 | Module name. |

Note: When configuring the two UART components you will need to make sure that DTC and FIFO are both enabled in the UART configuration. Also, you must create both TX/RX DTC components per UART.

Note: If you wish to use flow control then you must enable flow control in the RA Configuration editor. This can be found in the UART setting. It is advantageous to use flow control all the time since it allows the hardware to gate the flow of data across the serial bus. Without hardware flow control for faster data rate you will most likely see an overflow condition between MCU and the module device.

Note: Higher baud rates are supported in the RA Configuration editor and should be changed in the first UART configuration. There is no need to change the second UART baud rate since it is only used as an AT command channel.

Note: It is a good idea to also enable the FIFO in the UART configuration settings if you plan to use higher baud rates.

**Interrupt Configuration**

Refer to Serial Communications Interface (SCI) UART (r_sci_uart). R_SCI_UART_Open() is called by Wifi Middleware (rm_wifi_onchip_silex).

### Clock Configuration

Refer to Serial Communications Interface (SCI) UART (r_sci_uart).

### Pin Configuration

Refer to Serial Communications Interface (SCI) UART (r_sci_uart). R_SCI_UART_Open() is called by Wifi Middleware (rm_wifi_onchip_silex)

## Usage Notes

### Limitations

- WiFi AP connections do not currently support WEP security.
- When operating with a single UART only single socket connections are possible. To support multiple sockets two UART channels must be connected to the module. When using the Renesas-provided SX-ULPGN PMOD board the second UART channel is on pins 9 and 10 of the PMOD header.
- Network connection parameters SSID and Passphrase for the Access Point can not contain any commas. This is a current limitation of the Silex module firmware. The rm_wifi_onchip_silex_connect() function will return an error if a comma is detected.

## Examples

### Basic Example

This is a basic example of minimal use of WiFi Middleware in an application.

```
void wifi_onchip_basic_example (void)
{
    WIFIReturnCode_t    wifi_err;
    WIFINetworkParams_t net_params;
    SocketsSockaddr_t   addr            = {0};
    int32_t             number_bytes_rx = 0;
    int32_t             number_bytes_tx = 0;
    memset(scan_data, 0, sizeof(WIFIScanResult_t) * MAX_WIFI_SCAN_RESULTS);
    memset(g_socket_recv_buffer, 0, sizeof(uint8_t) * SX_WIFI_SOCKET_RX_BUFFER_SIZE);
 /* Open connection to the Wifi Module */
    wifi_err = WIFI_On();
 if (wifi_err)
    {
       handle_error((fsp_err_t) wifi_err);
```

```
   }
/* Setup Access Point connection parameters */
   net_params.cChannel         = 0;
   net_params.pcPassword       = "password";
   net_params.pcSSID          = "access_point_ssid";
   net_params.ucPasswordLength = 8;
   net_params.ucSSIDLength    = 17;
   net_params.xSecurity       = eWiFiSecurityWPA2;
/* Connect to the Access Point */
   wifi_err = WIFI_ConnectAP(&net_params);
if (wifi_err)
   {
      handle_error((fsp_err_t) wifi_err);
   }
/* Get address assigned by AP */
   uint8_t ip_address_device[4] = {0};
   wifi_err = WIFI_GetIP(&ip_address_device[0]);
if (wifi_err)
   {
      handle_error((fsp_err_t) wifi_err);
   }
/* Ping an address accessible on the network */
   uint8_t       ip_address[4] = {216, 58, 194, 174}; // NOLINT
const uint16_t ping_count    = 3;
const uint32_t intervalMS    = 100;
   wifi_err = WIFI_Ping(&ip_address[0], ping_count, intervalMS);
if (wifi_err)
   {
      handle_error((fsp_err_t) wifi_err);
   }
/* Scan the local Wifi network for other APs */
   wifi_err = WIFI_Scan(&scan_data[0], MAX_WIFI_SCAN_RESULTS);
if (wifi_err)
   {
```

```
        handle_error((fsp_err_t) wifi_err);
    }
 /* Do a DNS Query for IP address of server */
    addr.ulAddress = SOCKETS_GetHostByName("www.renesas.com");
    addr.usPort    = SOCKETS_htons(80);
 /* Initialize the Socket Interface */
    BaseType_t sock_err = SOCKETS_Init();
 if (sock_err != pdPASS)
    {
        handle_error((fsp_err_t) sock_err);
    }
 /* Create a socket instance */
    Socket_t socket1 = SOCKETS_Socket(SOCKETS_AF_INET, SOCKETS_SOCK_STREAM,
SOCKETS_IPPROTO_TCP);
 if (socket1 == NULL)
    {
        handle_error((fsp_err_t) !socket1);
    }
 /* Connect to an server using address */
    sock_err = SOCKETS_Connect(socket1, &addr, sizeof(SocketsSockaddr_t));
 if (sock_err)
    {
        handle_error((fsp_err_t) sock_err);
    }
 /* Send a HTTP Get call to server */
    number_bytes_tx = SOCKETS_Send(socket1, HTTP_GET_string, strlen(HTTP_GET_string),
0);
 if (0 >= number_bytes_tx)
    {
        handle_error((fsp_err_t) ERROR_OCCURED);
    }
 /* Receive the HTTP GET call reply */
    number_bytes_rx = SOCKETS_Recv(socket1, g_socket_recv_buffer,
SX_WIFI_SOCKET_RX_BUFFER_SIZE, 0);
```

```
if (0 >= number_bytes_rx)
   {
      handle_error((fsp_err_t) ERROR_OCCURED);
   }
/* Close the socket connection */
SOCKETS_Close(socket1);
/* Shutdown the WIFI and Socket Interfaces */
WIFI_Off();
}
```

### Data Structures

| | |
|---|---|
| struct | wifi_onchip_silex_cfg_t |
| struct | ulpgn_socket_t |
| struct | uart_state_t |
| struct | wifi_onchip_silex_instance_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | sx_ulpgn_security_t |
| enum | sx_ulpgn_socket_status_t |
| enum | sx_ulpgn_socket_rw |

## Data Structure Documentation

### ◆ wifi_onchip_silex_cfg_t

| struct wifi_onchip_silex_cfg_t | | |
|---|---|---|
| User configuration structure, used in open function | | |
| Data Fields | | |
| const uint32_t | num_uarts | Number of UART interfaces to use. |
| const uint32_t | num_sockets | Number of sockets to initialize. |
| const bsp_io_port_pin_t | reset_pin | Reset pin used for module. |
| const uart_instance_t * | uart_instances[WIFI_ONCHIP_SILEX_CFG_MAX_NUMBER_UART_PORTS] | SCI UART instances. |
| void const * | p_context | User defined context passed |

| | | |
|---|---|---|
| | | into callback function. |
| void const * | p_extend | Pointer to extended configuration by instance of interface. |

### ◆ ulpgn_socket_t

| struct ulpgn_socket_t | | |
|---|---|---|
| Silex ULPGN Wifi internal socket instance structure | | |
| Data Fields | | |
| StreamBufferHandle_t | socket_byteq_hdl | Socket stream buffer handle. |
| StaticStreamBuffer_t | socket_byteq_struct | Structure to hold stream buffer info. |
| uint8_t | socket_recv_buff[WIFI_ONCHIP_SILEX_CFG_MAX_SOCKET_RX_SIZE] | Socket receive buffer used by byte queue. |
| uint32_t | socket_status | Current socket status. |
| uint32_t | socket_recv_error_count | Socket receive error count. |
| uint32_t | socket_create_flag | Flag to determine in socket has been created. |
| uint32_t | socket_read_write_flag | flag to determine if read and/or write channels are active. |

### ◆ uart_state_t

| struct uart_state_t | | |
|---|---|---|
| Silex ULPGN Wifi SCI UART state information | | |
| Data Fields | | |
| SemaphoreHandle_t | uart_tei_sem | UART transmission end binary semaphore. |

### ◆ wifi_onchip_silex_instance_ctrl_t

| struct wifi_onchip_silex_instance_ctrl_t | | |
|---|---|---|
| WIFI_ONCHIP_SILEX private control block. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | open | Flag to indicate if wifi instance has been initialized. |
| wifi_onchip_silex_cfg_t const * | p_wifi_onchip_silex_cfg | Pointer to initial configurations. |
| bsp_io_port_pin_t | reset_pin | Wifi module reset pin. |
| uint32_t | num_uarts | number of UARTS currently used for communication with module |

| | | |
|---|---|---|
| uint32_t | tx_data_size | Size of the data to send. |
| uint32_t | num_creatable_sockets | Number of simultaneous sockets supported. |
| uint32_t | curr_cmd_port | Current UART instance index for AT commands. |
| uint32_t | curr_data_port | Current UART instance index for data. |
| uint8_t | cmd_rx_queue_buf[WIFI_ONCHIP_SILEX_CFG_CMD_RX_BUF_SIZE] | Command port receive buffer used by byte queue. |
| StreamBufferHandle_t | socket_byteq_hdl | Socket stream buffer handle. |
| StaticStreamBuffer_t | socket_byteq_struct | Structure to hold stream buffer info. |
| volatile uint32_t | curr_socket_index | Currently active socket instance. |
| uint8_t | cmd_tx_buff[WIFI_ONCHIP_SILEX_CFG_CMD_TX_BUF_SIZE] | Command send buffer. |
| uint8_t | cmd_rx_buff[WIFI_ONCHIP_SILEX_CFG_CMD_RX_BUF_SIZE] | Command receive buffer. |
| uint32_t | at_cmd_mode | Current command mode. |
| uint8_t | curr_ipaddr[4] | Current IP address of module. |
| uint8_t | curr_subnetmask[4] | Current Subnet Mask of module. |
| uint8_t | curr_gateway[4] | Current GAteway of module. |
| SemaphoreHandle_t | tx_sem | Transmit binary semaphore handle. |
| SemaphoreHandle_t | rx_sem | Receive binary semaphore handle. |
| uint8_t | last_data[WIFI_ONCHIP_SILEX_RETURN_TEXT_LENGTH] | Tailing buffer used for command parser. |
| uart_instance_t * | uart_instance_objects[WIFI_ONCHIP_SILEX_CFG_MAX_NUMBER_UART_PORTS] | UART instance objects. |
| uart_state_t | uart_state_info[WIFI_ONCHIP_SILEX_CFG_MAX_NUMBER_UART_PORTS] | UART instance state information. |
| ulpgn_socket_t | sockets[WIFI_ONCHIP_SILEX_CFG_NUM_CREATEABLE_SOCKETS] | Internal socket instances. |

## Enumeration Type Documentation

### ◆ sx_ulpgn_security_t

| enum sx_ulpgn_security_t |
|---|
| Silex ULPGN Wifi security types |

### ◆ sx_ulpgn_socket_status_t

| enum sx_ulpgn_socket_status_t |
|---|
| Silex ULPGN Wifi socket status types |

### ◆ sx_ulpgn_socket_rw

| enum sx_ulpgn_socket_rw |
|---|
| Silex socket shutdown channels |

## Function Documentation

### ◆ rm_wifi_onchip_silex_open()

fsp_err_t rm_wifi_onchip_silex_open ( wifi_onchip_silex_cfg_t const *const *p_cfg*)

Opens and configures the WIFI_ONCHIP_SILEX Middleware module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_cfg | Pointer to pin configuration structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WIFI_ONCHIP_SILEX successfully configured. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_OUT_OF_MEMORY | There is no more heap memory available. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |

### ◆ rm_wifi_onchip_silex_version_get()

fsp_err_t rm_wifi_onchip_silex_version_get ( fsp_version_t *const *p_version*)

Returns the WIFI_ONCHIP_SILEX Middleware module versions.

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Memory address to return version information to. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |

### ◆ rm_wifi_onchip_silex_close()

fsp_err_t rm_wifi_onchip_silex_close ( )

Disables WIFI_ONCHIP_SILEX.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WIFI_ONCHIP_SILEX closed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ rm_wifi_onchip_silex_connect()

fsp_err_t rm_wifi_onchip_silex_connect ( const char * *p_ssid*, uint32_t *security*, const char * *p_passphrase* )

Connects to the specified Wifi Access Point.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ssid | Pointer to SSID of Wifi Access Point. |
| [in] | security | Security type to use for connection. |
| [in] | p_passphrase | Pointer to the passphrase to use for connection. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_INVALID_ARGUMENT | No commas are accepted in the SSID or Passphrase. |

### ◆ rm_wifi_onchip_silex_mac_addr_get()

fsp_err_t rm_wifi_onchip_silex_mac_addr_get ( uint8_t * *p_macaddr*)

Get MAC address.

**Parameters**

| | | |
|---|---|---|
| [out] | p_macaddr | Pointer array to hold mac address. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_scan()

fsp_err_t rm_wifi_onchip_silex_scan ( WIFIScanResult_t * *p_results*, uint32_t *maxNetworks* )

Get the information about local Wifi Access Points.

**Parameters**

| | | |
|---|---|---|
| [out] | p_results | Pointer to a structure array holding scanned Access Points. |
| [in] | maxNetworks | Size of the structure array for holding APs. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_ping()

fsp_err_t rm_wifi_onchip_silex_ping ( uint8_t * *p_ip_addr*, uint32_t *count*, uint32_t *interval_ms* )

Ping an IP address on the network.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ip_addr | Pointer to IP address array. |
| [in] | count | Number of pings to attempt. |
| [in] | interval_ms | Interval between ping attempts. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_ip_addr_get()

fsp_err_t rm_wifi_onchip_silex_ip_addr_get ( uint8_t * *p_ip_addr*)

Get the assigned module IP address.

**Parameters**

| | | |
|---|---|---|
| [out] | p_ip_addr | Pointer an array to hold the IP address. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_avail_socket_get()

fsp_err_t rm_wifi_onchip_silex_avail_socket_get ( uint32_t * *p_socket_id*)

Get the next available socket ID.

**Parameters**

| | | |
|---|---|---|
| [out] | p_socket_id | Pointer to an integer to hold the socket ID. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_WIFI_FAILED | Error occured in the execution of this function |

### ◆ rm_wifi_onchip_silex_socket_status_get()

fsp_err_t rm_wifi_onchip_silex_socket_status_get ( uint32_t *socket_no*, uint32_t * *p_socket_status* )

Get the socket status.

**Parameters**

| | | |
|---|---|---|
| [in] | socket_no | Socket ID number. |
| [out] | p_socket_status | Pointer to an integer to hold the socket status |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_socket_create()

fsp_err_t rm_wifi_onchip_silex_socket_create ( uint32_t *socket_no*, uint32_t *type*, uint32_t *ipversion* )

Create a new socket instance.

**Parameters**

| | | |
|---|---|---|
| [in] | socket_no | Socket ID number. |
| [in] | type | Socket type. |
| [in] | ipversion | Socket IP type. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_tcp_connect()

fsp_err_t rm_wifi_onchip_silex_tcp_connect ( uint32_t *socket_no*, uint32_t *ipaddr*, uint32_t *port* )

Connect to a specific IP and Port using socket.

**Parameters**

| | | |
|---|---|---|
| [in] | socket_no | Socket ID number. |
| [in] | ipaddr | IP address for socket connection. |
| [in] | port | Port number for socket connection. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_tcp_send()

int32_t rm_wifi_onchip_silex_tcp_send ( uint32_t *socket_no*, const uint8_t * *p_data*, uint32_t *length*, uint32_t *timeout_ms* )

Send data over TCP to a server.

**Parameters**

| | | |
|---|---|---|
| [in] | socket_no | Socket ID number. |
| [in] | p_data | Pointer to data to send. |
| [in] | length | Length of data to send. |
| [in] | timeout_ms | Timeout to wait for transmit end event |

**Return values**

| | |
|---|---|
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_tcp_recv()

int32_t rm_wifi_onchip_silex_tcp_recv ( uint32_t *socket_no*, uint8_t * *p_data*, uint32_t *length*, uint32_t *timeout_ms* )

Receive data over TCP from a server.

**Parameters**

| | | |
|---|---|---|
| [in] | socket_no | Socket ID number. |
| [out] | p_data | Pointer to data received from socket. |
| [in] | length | Length of data array used for receive. |
| [in] | timeout_ms | Timeout to wait for data to be received from socket. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_tcp_shutdown()

int32_t rm_wifi_onchip_silex_tcp_shutdown ( uint32_t *socket_no*, uint32_t *shutdown_channels* )

Shutdown portion of a socket

**Parameters**

| | | |
|---|---|---|
| [in] | socket_no | Socket ID number. |
| [in] | shutdown_channels | Specify if read or write channel is shutdown for socket |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_socket_disconnect()

fsp_err_t rm_wifi_onchip_silex_socket_disconnect ( uint32_t *socket_no*)

Disconnect a specific socket connection.

**Parameters**

| | | |
|---|---|---|
| [in] | socket_no | Socket ID to disconnect |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_INVALID_ARGUMENT | Bad parameter value was passed into function. |

### ◆ rm_wifi_onchip_silex_disconnect()

fsp_err_t rm_wifi_onchip_silex_disconnect ( )

Disconnects from connected AP.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WIFI_ONCHIP_SILEX disconnected successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ rm_wifi_onchip_silex_dns_query()

fsp_err_t rm_wifi_onchip_silex_dns_query ( const char * *p_textstring*, uint8_t * *p_ip_addr* )

Initiate a DNS lookup for a given URL.

**Parameters**

| | | |
|---|---|---|
| [in] | p_textstring | Pointer to array holding URL to query from DNS. |
| [out] | p_ip_addr | Pointer to IP address returned from look up. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_INVALID_ARGUMENT | The URL passed in is to long. |

### ◆ rm_wifi_onchip_silex_socket_connected()

fsp_err_t rm_wifi_onchip_silex_socket_connected ( fsp_err_t * *p_status*)

Check if a specific socket instance is connected.

**Parameters**

| | | |
|---|---|---|
| [out] | p_status | Pointer to integer holding the socket connection status. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |

### ◆ rm_wifi_onchip_silex_uart_callback()

void rm_wifi_onchip_silex_uart_callback ( uart_callback_args_t * *p_args*)

Callback function for first UART port in command mode. Used specifically for the SCI UART driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_args | Pointer to callback arguments structure. |

### ◆ SOCKETS_Socket()

Socket_t SOCKETS_Socket ( int32_t *lDomain*, int32_t *lType*, int32_t *lProtocol* )

Creates a TCP socket.

This call allocates memory and claims a socket resource.

**See also**

SOCKETS_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | lDomain | Must be set to SOCKETS_AF_INET. See SocketDomains. |
| [in] | lType | Set to SOCKETS_SOCK_STREAM to create a TCP socket. No other value is valid. See SocketTypes. |
| [in] | lProtocol | Set to SOCKETS_IPPROTO_TCP to create a TCP socket. No other value is valid. See Protocols. |

**Returns**

- If a socket is created successfully, then the socket handle is returned
- SOCKETS_INVALID_SOCKET is returned if an error occurred.

### ◆ SOCKETS_Connect()

int32_t SOCKETS_Connect ( Socket_t *xSocket*, SocketsSockaddr_t * *pxAddress*, Socklen_t *xAddressLength* )

Connects the socket to the specified IP address and port.

The socket must first have been successfully created by a call to SOCKETS_Socket().

**Parameters**

| | | |
|---|---|---|
| [in] | xSocket | The handle of the socket to be connected. |
| [in] | pxAddress | A pointer to a SocketsSockaddr_t structure that contains the the address to connect the socket to. |
| [in] | xAddressLength | Should be set to sizeof( SocketsSockaddr_t ). |

**Returns**

- SOCKETS_ERROR_NONE if a connection is established.
- If an error occured, a negative value is returned.

### ◆ SOCKETS_Recv()

int32_t SOCKETS_Recv ( Socket_t *xSocket*, void * *pvBuffer*, size_t *xBufferLength*, uint32_t *ulFlags* )

Receive data from a TCP socket.

The socket must have already been created using a call to SOCKETS_Socket() and connected to a remote socket using SOCKETS_Connect().

**Parameters**

| | | |
|---|---|---|
| [in] | xSocket | The handle of the socket from which data is being received. |
| [out] | pvBuffer | The buffer into which the received data will be placed. |
| [in] | xBufferLength | The maximum number of bytes which can be received. pvBuffer must be at least xBufferLength bytes long. |
| [in] | ulFlags | Not currently used. Should be set to 0. |

**Returns**

- If the receive was successful then the number of bytes received (placed in the buffer pointed to by pvBuffer) is returned.
- If a timeout occurred before data could be received then 0 is returned (timeout is set using SOCKETS_SO_RCVTIMEO).
- If an error occured, a negative value is returned.

### ◆ SOCKETS_Send()

int32_t SOCKETS_Send ( Socket_t *xSocket*, const void * *pvBuffer*, size_t *xDataLength*, uint32_t *ulFlags* )

Transmit data to the remote socket.

The socket must have already been created using a call to SOCKETS_Socket() and connected to a remote socket using SOCKETS_Connect().

**Parameters**

| | | |
|---|---|---|
| [in] | xSocket | The handle of the sending socket. |
| [in] | pvBuffer | The buffer containing the data to be sent. |
| [in] | xDataLength | The length of the data to be sent. |
| [in] | ulFlags | Not currently used. Should be set to 0. |

**Returns**

- On success, the number of bytes actually sent is returned.
- If an error occured, a negative value is returned.

### ◆ SOCKETS_Shutdown()

int32_t SOCKETS_Shutdown ( Socket_t *xSocket*, uint32_t *ulHow* )

Closes all or part of a full-duplex connection on the socket.

**Parameters**

| | | |
|---|---|---|
| [in] | xSocket | The handle of the socket to shutdown. |
| [in] | ulHow | SOCKETS_SHUT_RD, SOCKETS_SHUT_WR or SOCKETS_SHUT_RDWR. ShutdownFlags |

**Returns**

- If the operation was successful, 0 is returned.
- If an error occured, a negative value is returned.

### ◆ SOCKETS_Close()

int32_t SOCKETS_Close ( Socket_t *xSocket*)

Closes the socket and frees the related resources.

**Parameters**

| | | |
|---|---|---|
| [in] | xSocket | The handle of the socket to close. |

**Returns**

- On success, 0 is returned.
- If an error occurred, a negative value is returned.

### ◆ SOCKETS_SetSockOpt()

int32_t SOCKETS_SetSockOpt ( Socket_t *xSocket*, int32_t *lLevel*, int32_t *lOptionName*, const void * *pvOptionValue*, size_t *xOptionLength* )

Manipulates the options for the socket.

**Parameters**

| | | |
|---|---|---|
| [in] | xSocket | The handle of the socket to set the option for. |
| [in] | lLevel | Not currently used. Should be set to 0. |
| [in] | lOptionName | See SetSockOptOptions. |
| [in] | pvOptionValue | A buffer containing the value of the option to set. |
| [in] | xOptionLength | The length of the buffer pointed to by pvOptionValue. |

***Note***

*Socket option support and possible values vary by port. Please see PORT_SPECIFIC_LINK to check the valid options and limitations of your device.*

- Berkeley Socket Options
  - SOCKETS_SO_RCVTIMEO
    - Sets the receive timeout
    - pvOptionValue (TickType_t) is the number of milliseconds that the receive function should wait before timing out.
    - Setting pvOptionValue = 0 causes receive to wait forever.
    - See PORT_SPECIFIC_LINK for device limitations.
  - SOCKETS_SO_SNDTIMEO
    - Sets the send timeout
    - pvOptionValue (TickType_t) is the number of milliseconds that the send function should wait before timing out.
    - Setting pvOptionValue = 0 causes send to wait forever.

        - See PORT_SPECIFIC_LINK for device limitations.
- Non-Standard Options
    - SOCKETS_SO_NONBLOCK
        - Makes a socket non-blocking.
        - pvOptionValue is ignored for this option.
- Security Sockets Options
    - SOCKETS_SO_REQUIRE_TLS
        - Use TLS for all connect, send, and receive on this socket.
        - This socket options MUST be set for TLS to be used, even if other secure socket options are set.
        - pvOptionValue is ignored for this option.
    - SOCKETS_SO_TRUSTED_SERVER_CERTIFICATE
        - Set the root of trust server certificiate for the socket.
        - This socket option only takes effect if SOCKETS_SO_REQUIRE_TLS is also set. If SOCKETS_SO_REQUIRE_TLS is not set, this option will be ignored.
        - pvOptionValue is a pointer to the formatted server certificate. TODO: Link to description of how to format certificates with
        - xOptionLength (BaseType_t) is the length of the certificate in bytes.
    - SOCKETS_SO_SERVER_NAME_INDICATION
        - Use Server Name Indication (SNI)
        - This socket option only takes effect if SOCKETS_SO_REQUIRE_TLS is also set. If SOCKETS_SO_REQUIRE_TLS is not set, this option will be ignored.
        - pvOptionValue is a pointer to a string containing the hostname
        - xOptionLength is the length of the hostname string in bytes.

**Returns**

- On success, 0 is returned.
- If an error occured, a negative value is returned.

### ◆ SOCKETS_GetHostByName()

uint32_t SOCKETS_GetHostByName ( const char * *pcHostName*)

Resolve a host name using Domain Name Service.

**Parameters**

| | | |
|---|---|---|
| [in] | pcHostName | The host name to resolve. |

**Returns**

- The IPv4 address of the specified host.
- If an error has occured, 0 is returned.

### ◆ SOCKETS_Init()

BaseType_t SOCKETS_Init ( void )

Secure Sockets library initialization function.

This function does general initialization and setup. It must be called once and only once before calling any other function.

**Returns**

- pdPASS if everything succeeds
- pdFAIL otherwise.

### ◆ ulApplicationGetNextSequenceNumber()

uint32_t ulApplicationGetNextSequenceNumber ( uint32_t *ulSourceAddress*, uint16_t *usSourcePort*, uint32_t *ulDestinationAddress*, uint16_t *usDestinationPort* )

Generate a TCP Initial Sequence Number that is reasonably difficult to predict, per https://tools.ietf.org/html/rfc6528.

### ◆ WIFI_On()

WIFIReturnCode_t WIFI_On ( void )

Turns on Wi-Fi.

This function turns on Wi-Fi module,initializes the drivers and must be called before calling any other Wi-Fi API

**Returns**

eWiFiSuccess if Wi-Fi module was successfully turned on, failure code otherwise.

### ◆ WIFI_Off()

WIFIReturnCode_t WIFI_Off ( void )

Turns off Wi-Fi.

This function turns off the Wi-Fi module. The Wi-Fi peripheral should be put in a low power or off state in this routine.

**Returns**

eWiFiSuccess if Wi-Fi module was successfully turned off, failure code otherwise.

### ◆ WIFI_ConnectAP()

WIFIReturnCode_t WIFI_ConnectAP ( const WIFINetworkParams_t *const *pxNetworkParams*)

Connects to the Wi-Fi Access Point (AP) specified in the input.

The Wi-Fi should stay connected when the same Access Point it is currently connected to is specified. Otherwise, the Wi-Fi should disconnect and connect to the new Access Point specified. If the new Access Point specifed has invalid parameters, then the Wi-Fi should be disconnected.

**Parameters**

| | | |
|---|---|---|
| [in] | pxNetworkParams | Configuration to join AP. |

**Returns**

eWiFiSuccess if connection is successful, failure code otherwise.

```
WIFINetworkParams_t xNetworkParams;
WIFIReturnCode_t xWifiStatus;
xNetworkParams.pcSSID = "SSID String";
xNetworkParams.ucSSIDLength = SSIDLen;
xNetworkParams.pcPassword = "Password String";
xNetworkParams.ucPasswordLength = PassLength;
xNetworkParams.xSecurity = eWiFiSecurityWPA2;
xWifiStatus = WIFI_ConnectAP( &( xNetworkParams ) );
if(xWifiStatus == eWiFiSuccess)
{
 //Connected to AP.
}
```

**See also**

WIFINetworkParams_t

### ◆ WIFI_Disconnect()

WIFIReturnCode_t WIFI_Disconnect ( void )

Disconnects from the currently connected Access Point.

**Returns**

eWiFiSuccess if disconnection was successful or if the device is already disconnected, failure code otherwise.

### ◆ WIFI_Reset()

WIFIReturnCode_t WIFI_Reset ( void )

Resets the Wi-Fi Module.

**Returns**

eWiFiSuccess if Wi-Fi module was successfully reset, failure code otherwise.

### ◆ WIFI_Scan()

WIFIReturnCode_t WIFI_Scan ( WIFIScanResult_t * *pxBuffer*, uint8_t *ucNumNetworks* )

Perform a Wi-Fi network Scan.

**Parameters**

| | | |
|---|---|---|
| [in] | pxBuffer | - Buffer for scan results. |
| [in] | ucNumNetworks | - Number of networks to retrieve in scan result. |

**Returns**

eWiFiSuccess if the Wi-Fi network scan was successful, failure code otherwise.

***Note***

*The input buffer will have the results of the scan.*

```
const uint8_t ucNumNetworks = 10; //Get 10 scan results
WIFIScanResult_t xScanResults[ ucNumNetworks ];
WIFI_Scan( xScanResults, ucNumNetworks );
```

### ◆ WIFI_Ping()

WIFIReturnCode_t WIFI_Ping ( uint8_t * *pucIPAddr*, uint16_t *usCount*, uint32_t *ulIntervalMS* )

Ping an IP address in the network.

**Parameters**

| | | |
|---|---|---|
| [in] | pucIPAddr | IP Address array to ping. |
| [in] | usCount | Number of times to ping |
| [in] | ulIntervalMS | Interval in milliseconds for ping operation |

**Returns**

eWiFiSuccess if ping was successful, other failure code otherwise.

### ◆ WIFI_GetIP()

WIFIReturnCode_t WIFI_GetIP ( uint8_t * *pucIPAddr*)

Retrieves the Wi-Fi interface's IP address.

**Parameters**

| | | |
|---|---|---|
| [out] | pucIPAddr | IP Address buffer. |

**Returns**

eWiFiSuccess if successful and IP Address buffer has the interface's IP address, failure code otherwise.

```
uint8_t ucIPAddr[ 4 ];

WIFI_GetIP( &ucIPAddr[0] );
```

### ◆ WIFI_GetMAC()

WIFIReturnCode_t WIFI_GetMAC ( uint8_t * *pucMac*)

Retrieves the Wi-Fi interface's MAC address.

**Parameters**

| | | |
|---|---|---|
| [out] | pucMac | MAC Address buffer sized 6 bytes. |

```
uint8_t ucMacAddressVal[ wificonfigMAX_BSSID_LEN ];

WIFI_GetMAC( &ucMacAddressVal[0] );
```

**Returns**

eWiFiSuccess if the MAC address was successfully retrieved, failure code otherwise. The returned MAC address must be 6 consecutive bytes with no delimitters.

### ◆ WIFI_GetHostIP()

WIFIReturnCode_t WIFI_GetHostIP ( char * *pcHost*, uint8_t * *pucIPAddr* )

Retrieves the host IP address from a host name using DNS.

**Parameters**

| | | |
|---|---|---|
| [in] | pcHost | - Host (node) name. |
| [in] | pucIPAddr | - IP Address buffer. |

**Returns**

eWiFiSuccess if the host IP address was successfully retrieved, failure code otherwise.

```
uint8_t ucIPAddr[ 4 ];

WIFI_GetHostIP( "amazon.com", &ucIPAddr[0] );
```

### ◆ WIFI_IsConnected()

BaseType_t WIFI_IsConnected ( void )

Check if the Wi-Fi is connected.

**Returns**

pdTRUE if the link is up, pdFalse otherwise.

## 4.2.69 AWS Secure Sockets

Modules

This module provides the AWS Secure Sockets implementation.

# Overview

### Features

Information about the features provided by the AWS Secure Sockets Library is available in the FreeRTOS Libraries User Guide.

The FSP implementation supports using Secure Sockets with either Ethernet or WiFi. These stacks can be added in FSP via the RA Configuration editor under FreeRTOS | Secure Sockets.

### Dependencies

The Secure Sockets library has two dependencies:

1. A TCP/IP implementation
2. A TLS implementation

For TCP/IP, AWS have provided the FreeRTOS TCP/IP implementation. For TLS, AWS have chosen mbedTLS, but use PKCS11 for storage and invoking the crypto portion of mbedTLS. For more information about AWS Secure Sockets, refer to the AWS documentation. An example of Secure Sockets usage is on the same page.

**mbedTLS**

mbedTLS is ARM's implementation of the TLS and SSL protocols as well as the cryptographic primitives required by those implementations. mbedTLS is also solely used for its cryptographic features even if the TLS/SSL portions are not used. With PSA, ARM have created a separate API for cryptography. Starting with mbedTLS3, crypto implementation has been moved out to a new module called mbedCrypto (PSA Crypto API) and a build time configuration can direct the mbedTLS3 implementation to use either the old mbedtls cryptography functions or use the new PSA Crypto API. Since the current version of mbedCrypto (PSA Crypto API) implements both the old mbedtls crypto API as well as the new PSA Crypto API, either option is functional for now.

**CipherSuites**

During the TLS connection setup stage, the client has to indicate to the server the type of cryptographic operations that it supports. This is referred to as the ciphersuite. The entire list of ciphersuites supported by mbedTLS can be found in mbedtls/ssl_ciphersuites.h.

**Configuration**

In FSP, Secure Sockets can be added as a new stack via FreeRTOS | Secure Sockets | Secure Sockets on WiFi or Secure Sockets on FreeRTOS Plus TCP. All required dependant modules, except heap, are automatically added. To complete the configuration,

- Add a heap instance and use the same one for all dependencies.
- Resolve the module configuration requirements.

# Usage Notes

For detailed documentation on Secure Sockets consult the AWS documentation.

# Examples

**Basic Example**

This is a basic example of using the Secure Sockets API with Ethernet. The message "hello, world!" is sent to a remote socket.

```
#define SECURE_SOCKETS_EXAMPLE_BUFFER_SIZE (64)

static const char SERVER_CERTIFICATE_PEM[] =

 "-----BEGIN CERTIFICATE-----\n"

 "MIIDazCCAlOgAwIBAgIURabL79ayIywQv0y8SPnbZ1FYDRIwDQYJKoZIhvcNAQEL\n"

 "BQAwRTELMAkGA1UEBhMCQVUxEzARBgNVBAgMClNvbWUtU3RhdGUxITAfBgNVBAoM\n"
```

```
 "GEludGVybmV0IFdpZGdpdHMgUHR5IEx0ZDAeFw0xOTA5MTEyMTIyMjZaFw0yMDA5\n"
 "MTAyMTIyMjZaMEUxCzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEw\n"
 "HwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQwggEiMA0GCSqGSIb3DQEB\n"
 "AQUAA4IBDwAwggEKAoIBAQDSA3h+5sT58FHgnovnQzsVHQ0H/3TsnEKwVzyBwTQl\n"
 "s4PbG6VXCWyyJWjdJ4XMH1oU8gAlxauFbwOO98Aquei4K3Pi/ynKNBeX4VJcLyE5\n"
 "Azq7nRIIwt4+OoZ5kV7v8JIoLY5i+Ktn3zq1t0y1ZmK6Uk/rRPonb+Kx7wQPx7jq\n"
 "ZIZGda+CgF6ZedidPcABuggqD1y3U2gLiRPoBhe9nN2hG60rRp7vhbWMF0pzTDXu\n"
 "BKF7XSTbhYz3pl6NeOCLh5E3t8x908Ui5W1zDN3iOysrcwQFtCiGTvzNtxSfli1+\n"
 "PugIt9Q2vlYmuz5qI+juxHftJSXO86M5SV7exqUOXP9RAgMBAAGjUzBRMB0GA1Ud\n"
 "DgQWBBQG8VNJEJUjpTKMjmrOY3XApNp5lDAfBgNVHSMEGDAWgBQG8VNJEJUjpTKM\n"
 "jmrOY3XApNp5lDAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3DQEBCwUAA4IBAQAt\n"
 "CabfjsYUnG8tt3/GDdhjsuG+SfeQe11S73pZi3+L616bPH5MNUv+LkgR/1AFEqt5\n"
 "WadKVTgzW5Ork1t7CfkYwrOHbyhyaaDPzERjMCfCcl8lQluBy6vE/lEb0hWq6XlO\n"
 "f6+8i+VKxWkSIXs2ZQqqYSOTTzAjHSsiiuE5WsC00ErvCvnC7uD6+3Y7W1uQRkFZ\n"
 "uSd9AN1ixPvAFi69FF/ymlJv6vII5GXOVDrIwdr50bMNuezMEx6qMNDADRH8iEaL\n"
 "JaSgfklczGiI1i7MPD4JTtsXOgKwxcBDAa0zQDVA5uBGEIOhva3m5X70N4iO7W0V\n"
 "eEhZekKeg3Fl3t/CXi8l\n"
 "-----END CERTIFICATE-----";
#define keyCLIENT_CERTIFICATE_PEM \
  "-----BEGIN CERTIFICATE-----\n" \
  "MIIDETCCAfkCFHwd2yn8zn5qB2ChYUT9Mvbi9Xp1MA0GCSqGSIb3DQEBCwUAMEUx\n" \
  "CzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRl\n" \
  "cm5ldCBXaWRnaXRzIFB0eSBMdGQwHhcNMTkwOTExMjEyMjU0WhcNMjAwOTEwMjEy\n" \
  "MjU0WjBFMQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UE\n" \
  "CgwYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMIIBIjANBgkqhkiG9w0BAQEFAAOC\n" \
  "AQ8AMIIBCgKCAQEAo8oThJXSMDo41oL7HTpC4TX8NalBvnkFw30Av67dl/oZDjVA\n" \
  "iXPnZkhVppLnj++0/Oed0M7UwNUO2nurQt6yTYrvW7E8ZPjAlC7ueJcGYZhOaVv2\n" \
  "bhSmigjFQru2lw5odSuYy5+22CCgxft58nrRCo5Bk+GwWgZmcrxe/BzutRHQ7X4x\n" \
  "dYJhyhBOi2R1Kt8XsbuWilfgfkVhhkVklFeKqiypdQM6cnPWo/G4DyW34jOXzzEM\n" \
  "FLWvQOQLCKUZOgjJBnFdbx8oOOwMkYCChbV7gqPE6cw0Zy26CvlLQiINyonLPbNT\n" \
  "c64sS/ZBGPZFOPJmb4tG2nipYgZ1hO/r++jCbwIDAQABMA0GCSqGSIb3DQEBCwUA\n" \
  "A4IBAQCdqq59ubdRY9EiV3bleKXeqG7+8HgBHdm0X9dgq10nD37p00YLyuZLE9NM\n" \
  "066G/VcflGrx/Nzw+/UuI7/UuBbBS/3ppHRnsZqBIl8nnr/ULrFQy8z3vKtL1q3C\n" \
  "DxabjPONlPO2keJeTTA71N/RCEMwJoa8i0XKXGdu/hQo6x4n+Gq73fEiGCl99xsc\n" \
```

```
  "4tIO4yPS4lv+uXBzEUzoEy0CLIkiDesnT5lLeCyPmUNoU89HU95IusZT7kygCHHd\n" \
  "72am1ic3X8PKc268KT3ilr3VMhK67C+iIIkfrM5AiU+oOIRrIHSC/p0RigJg3rXA\n" \
  "GBIRHvt+OYF9fDeG7U4QDJNCfGW+\n" \
  "-----END CERTIFICATE-----"
#define keyCLIENT_PRIVATE_KEY_PEM \
  "-----BEGIN RSA PRIVATE KEY-----\n" \
  "MIIEowIBAAKCAQEAo8oThJXSMDo41oL7HTpC4TX8NalBvnkFw30Av67dl/oZDjVA\n" \
  "iXPnZkhVppLnj++0/Oed0M7UwNUO2nurQt6yTYrvW7E8ZPjAlC7ueJcGYZhOaVv2\n" \
  "bhSmigjFQru2lw5odSuYy5+22CCgxft58nrRCo5Bk+GwWgZmcrxe/BzutRHQ7X4x\n" \
  "dYJhyhBOi2R1Kt8XsbuWilfgfkVhhkVklFeKqiypdQM6cnPWo/G4DyW34jOXzzEM\n" \
  "FLWvQOQLCKUZOgjJBnFdbx8oOOwMkYCChbV7gqPE6cw0Zy26CvlLQiINyonLPbNT\n" \
  "c64sS/ZBGPZFOPJmb4tG2nipYgZ1hO/r++jCbwIDAQABAoIBAQCGR2hC/ZVJhqIM\n" \
  "c2uuJZKpElpIIBBPOObZwwS3IYR4UUjzVgMn7UbbmxflLXD8lzfZU4YVp0vTH5lC\n" \
  "07qvYuXpHqtnj+GEok837VYCtUY9AuHeDM/2paV3awNV15E1PFG1Jd3pqnH7tJw6\n" \
  "VBZBDiGNNt1agN/UnoSlMfvpU0r8VGPXCBNxe3JY5QyBJPI1wF4LcxRI+eYmr7Ja\n" \
  "/cjn97DZotgz4B7gUNu8XIEkUOTwPabZINY1zcLWiXTMA+8qTniPVk653h14Xqt4\n" \
  "4o4D4YCTpwJcmxSV1m21/6+uyuXr9SIKAE+Ys2cYLA46x+rwLaW5fUoQ5hHa0Ytb\n" \
  "RYJ4SrtBAoGBANWtwlE69N0hq5xDPckSbNGubIeG8P4mBhGkJxIqYoqugGLMDiGX\n" \
  "4bltrjr2TPWaxTo3pPavLJiBMIsENA5KU+c/r0jLkxgEp9MIVJrtNgkCiDQqogBG\n" \
  "j4IJL2iQwXoLCqk2tx/dh9Mww+7SETE7EPNrv4UrYaGN5AEvpf5W+NHPAoGBAMQ6\n" \
  "wVa0Mx1PlA4enY2rfE3WXP8bzjleSOwR75JXqG2WbPC0/cszwbyPWOEqRpBZfvD/\n" \
  "QFkKx06xp1C09XwiQanr2gDucYXHeEKg/9iuJV1UkMQp95ojlhtSXdRZV7/l4pmN\n" \
  "fpB2vcAptX/4gY4tDrWMO08JNnRjE7duC+rmmk1hAoGAS4L0QLCNB/h2JOq+Uuhn\n" \
  "/FGfmOVfFPFrA6D3DbxcxpWUWVWzSLvb0SOphryzxbfEKyau7V5KbDp7ZSU/IC20\n" \
  "KOygjSEkAkDi7fjrrTRW/Cgg6g6G4YIOBO4qCtHdDbwJMHNdk6096qw5EZS67qLp\n" \
  "Apz5OZ5zChySjri/+HnTxJECgYBysGSP6IJ3fytplTtAshnU5JU2BWpi3ViBoXoE\n" \
  "bndilajWhvJO8dEqBB5OfAcCF0y6TnWtlT8oH21LHnjcNKlsRw0Dvllbd1oylybx\n" \
  "3da41dRG0sCEtoflMB7nHdDLt/DZDnoKtVvyFG6gfP47utn+Ahgn+Zp6K+46J3eP\n" \
  "s3g8AQKBgE/PJiaF8pbBXaZOuwRRA9GOMSbDIF6+jBYTYp4L9wk4+LZArKtyI+4k\n" \
  "Md2DUvHwMC+ddOtKqjYnLm+V5cSbvu7aPvBZtwxghzTUDcf7EvnA3V/bQBh3R0z7\n" \
  "pVsxTyGRmBSeLdbUWACUbX9LXdpudarPAJ59daWmP3mBEVmWdzUw\n" \
  "-----END RSA PRIVATE KEY-----"
const uint8_t g_ip_address[4] = {169, 254, 57, 49};
const uint8_t g_net_mask[4] = {255, 255, 0, 0};
```

```
const uint8_t g_gateway_address[4] = {169, 254, 57, 49};
const uint8_t g_dns_address[4] = {8, 8, 8, 8};
const uint8_t g_mac_address[6] = {0x66, 0x66, 0x66, 0x66, 0x66, 0x66};
static uint8_t g_buffer[SECURE_SOCKETS_EXAMPLE_BUFFER_SIZE];
/*******************************************************************************
**********************************
 * Refer to the following link for detailed API information:
 * https://docs.aws.amazon.com/freertos/latest/lib-
ref/html2/secure_sockets/secure_sockets_function_primary.html
 *******************************************************************************
**********************************/
void secure_sockets_ethernet_example (void)
{
 /* Initialize the crypto hardware acceleration. */
 mbedtls_platform_setup(NULL);
    xLoggingTaskInitialize(256, 1, 10); // NOLINT(readability-magic-numbers)
    ProvisioningParams_t params;
 /* Write the keys into a secure region in data flash. */
    params.pucClientPrivateKey       = (uint8_t *) keyCLIENT_PRIVATE_KEY_PEM;
    params.pucClientCertificate      = (uint8_t *) keyCLIENT_CERTIFICATE_PEM;
    params.ulClientPrivateKeyLength = 1 + strlen((const char *)
params.pucClientPrivateKey);
    params.ulClientCertificateLength = 1 + strlen((const char *)
params.pucClientCertificate);
    params.pucJITPCertificate       = NULL;
    params.ulJITPCertificateLength   = 0;
    vAlternateKeyProvisioning(&params);
 /* Start up the network stack. */
    FreeRTOS_IPInit(g_ip_address, g_net_mask, g_gateway_address, g_dns_address,
g_mac_address);
 while (pdFALSE == FreeRTOS_IsNetworkUp())
    {
       vTaskDelay(1);
    }
```

```
    Socket_t socket = SOCKETS_Socket(SOCKETS_AF_INET, SOCKETS_SOCK_STREAM,
SOCKETS_IPPROTO_TCP);
 if (SOCKETS_INVALID_SOCKET == socket)
    {
 /* Could not create socket. */
       __BKPT(0);
    }
 /* Enable TLS and configure the server certificate. */
 SOCKETS_SetSockOpt(socket, 0, SOCKETS_SO_REQUIRE_TLS, NULL, (size_t) 0);
 SOCKETS_SetSockOpt(socket, 0, SOCKETS_SO_TRUSTED_SERVER_CERTIFICATE,
SERVER_CERTIFICATE_PEM,
 sizeof(SERVER_CERTIFICATE_PEM));
 /* Connect to a remote server */
    SocketsSockaddr_t server_addr;
    server_addr.usPort    = SOCKETS_htons(9001);
    server_addr.ulAddress = SOCKETS_inet_addr_quick(192, 168, 0, 3);
 if (0 != SOCKETS_Connect(socket, &server_addr, sizeof(server_addr)))
    {
 /* Could not connect to server. */
       __BKPT(0);
    }
 /* Send a message and check that the correct number of bytes were transferred */
 const char msg[] = "hello, world!\n";
 if (sizeof(msg) != SOCKETS_Send(socket, msg, sizeof(msg), 0))
    {
 /* Failed to send data. */
       __BKPT(0);
    }
 if (0 != SOCKETS_Shutdown(socket, SOCKETS_SHUT_RDWR))
    {
       __BKPT(0);
    }
 /* Follow socket shutdown example:
  * https://freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_TCP/API/close.html
```

```
  */
 while (0 <= SOCKETS_Recv(socket, g_buffer, sizeof(g_buffer), 0))
    {
       vTaskDelay(10);
    }
 SOCKETS_Close(socket);
}
const char * pcApplicationHostnameHook(void);
const char * pcApplicationHostnameHook (void)
{
 /* Assign the name "FreeRTOS" to this network node. This function will
  * be called during the DHCP: the machine will be registered with an IP
  * address plus this name. */
 return "FreeRTOS";
}
void vApplicationIPNetworkEventHook (eIPCallbackEvent_t eNetworkEvent)
{
 FSP_PARAMETER_NOT_USED(eNetworkEvent);
}
```

# 4.3 Interfaces

### Detailed Description

The FSP interfaces provide APIs for common functionality. They can be implemented by one or more modules. Modules can use other modules as dependencies using this interface layer.

### Modules

| | |
|---|---|
| | ADC Interface<br>Interface for A/D Converters. |
| | BLE Interface<br>Interface for Bluetooth Low Energy functions. |
| | |

| | |
|---|---|
| | CAC Interface<br>Interface for clock frequency accuracy measurements. |
| | CAN Interface<br>Interface for CAN peripheral. |
| | CGC Interface<br>Interface for clock generation. |
| | Comparator Interface<br>Interface for comparators. |
| | CRC Interface<br>Interface for cyclic redundancy checking. |
| | CTSU Interface<br>Interface for Capacitive Touch Sensing Unit (CTSU) functions. |
| | DAC Interface<br>Interface for D/A converters. |
| | Display Interface<br>Interface for LCD panel displays. |
| | DOC Interface<br>Interface for the Data Operation Circuit. |
| | ELC Interface<br>Interface for the Event Link Controller. |
| | Ethernet Interface<br>Interface for Ethernet functions. |
| | Ethernet PHY Interface |

| | |
|---|---|
| | Interface for Ethernet PHY functions. |
| | External IRQ Interface<br>Interface for detecting external interrupts. |
| | Flash Interface<br>Interface for the Flash Memory. |
| | I2C Master Interface<br>Interface for I2C master communication. |
| | I2C Slave Interface<br>Interface for I2C slave communication. |
| | I2S Interface<br>Interface for I2S audio communication. |
| | I/O Port Interface<br>Interface for accessing I/O ports and configuring I/O functionality. |
| | JPEG Codec Interface<br>Interface for JPEG functions. |
| | Key Matrix Interface<br>Interface for key matrix functions. |
| | Low Power Modes Interface<br>Interface for accessing low power modes. |
| | Low Voltage Detection Interface<br>Interface for Low Voltage Detection. |
| | OPAMP Interface<br>Interface for Operational Amplifiers. |

| | |
|---|---|
| | POEG Interface<br>Interface for the Port Output Enable for GPT. |
| | RTC Interface<br>Interface for accessing the Realtime Clock. |
| | SD/MMC Interface<br>Interface for accessing SD, eMMC, and SDIO devices. |
| | SLCDC Interface<br>Interface for Segment LCD controllers. |
| | SPI Interface<br>Interface for SPI communications. |
| | SPI Flash Interface<br>Interface for accessing external SPI flash devices. |
| | Three-Phase Interface<br>Interface for three-phase timer functions. |
| | Timer Interface<br>Interface for timer functions. |
| | Transfer Interface<br>Interface for data transfer functions. |
| | UART Interface<br>Interface for UART communications. |
| | USB Interface<br>Interface for USB functions. |
| | |

| | |
|---|---|
| | USB HCDC Interface<br>Interface for USB HCDC functions. |
| | USB HHID Interface<br>Interface for USB HHID functions. |
| | USB HMSC Interface<br>Interface for USB HMSC functions. |
| | USB PCDC Interface<br>Interface for USB PCDC functions. |
| | USB PHID Interface<br>Interface for USB PHID functions. |
| | USB PMSC Interface<br>Interface for USB PMSC functions. |
| | WDT Interface<br>Interface for watch dog timer functions. |
| | BLE ABS Interface<br>Interface for Bluetooth Low Energy Abstraction functions. |
| | Block Media Interface<br>Interface for block media memory access. |
| | FreeRTOS+FAT Port Interface<br>Interface for FreeRTOS+FAT port. |
| | LittleFS Interface<br>Interface for LittleFS access. |
| | Touch Middleware Interface |

| | |
|---|---|
| | Interface for Touch Middleware functions. |

## 4.3.1 ADC Interface

Interfaces

### Detailed Description

Interface for A/D Converters.

# 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: Analog to Digital Converter (r_adc)

### Data Structures

| | |
|---|---|
| struct | adc_status_t |
| struct | adc_callback_args_t |
| struct | adc_info_t |
| struct | adc_cfg_t |
| struct | adc_api_t |
| struct | adc_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | adc_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | adc_mode_t |
| enum | adc_resolution_t |
| enum | adc_alignment_t |
| enum | adc_trigger_t |

| | |
|---|---|
| enum | adc_event_t |
| enum | adc_channel_t |
| enum | adc_state_t |

## Data Structure Documentation

### ◆ adc_status_t

| struct adc_status_t | | |
|---|---|---|
| ADC status. | | |
| Data Fields | | |
| adc_state_t | state | Current state. |

### ◆ adc_callback_args_t

| struct adc_callback_args_t | | |
|---|---|---|
| ADC callback arguments definitions | | |
| Data Fields | | |
| uint16_t | unit | ADC device in use. |
| adc_event_t | event | ADC callback event. |
| void const * | p_context | Placeholder for user data. |
| adc_channel_t | channel | Channel of conversion result. Only valid for ADC_EVENT_CONVERSION_COMPLETE. |

### ◆ adc_info_t

| struct adc_info_t | | |
|---|---|---|
| ADC Information Structure for Transfer Interface | | |
| Data Fields | | |
| __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) for reference voltage. |
| int16_t | slope_microvolts | Temperature sensor slope in microvolts/degrees C. |
| bool | calibration_ongoing | Calibration is in progress. |

### ◆ adc_cfg_t

| struct adc_cfg_t | |
|---|---|
| ADC general configuration | |
| **Data Fields** | |
| uint16_t | unit |
| | ADC unit to be used. |
| | |
| adc_mode_t | mode |
| | ADC operation mode. |
| | |
| adc_resolution_t | resolution |
| | ADC resolution. |
| | |
| adc_alignment_t | alignment |
| | Specify left or right alignment; ignored if addition used. |
| | |
| adc_trigger_t | trigger |
| | Default and Group A trigger source. |
| | |
| IRQn_Type | scan_end_irq |
| | Scan end IRQ number. |
| | |
| IRQn_Type | scan_end_b_irq |
| | Scan end group B IRQ number. |
| | |

| | |
|---|---|
| uint8_t | scan_end_ipl |
| | Scan end interrupt priority. |
| | |
| uint8_t | scan_end_b_ipl |
| | Scan end group B interrupt priority. |
| | |
| 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_callback_args_t. |
| | |
| void const * | p_extend |
| | Extension parameter for hardware specific settings. |
| | |

### ◆ adc_api_t

| | |
|---|---|
| struct adc_api_t | |
| ADC functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(adc_ctrl_t *const p_ctrl, adc_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | scanCfg )(adc_ctrl_t *const p_ctrl, void const *const p_extend) |
| | |
| fsp_err_t(* | scanStart )(adc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | scanStop )(adc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | scanStatusGet )(adc_ctrl_t *const p_ctrl, adc_status_t *p_status) |
| | |
| fsp_err_t(* | read )(adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint16_t |

| | |
|---|---|
| | *const p_data) |
| | |
| fsp_err_t(* | read32 )(adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data) |
| | |
| fsp_err_t(* | calibrate )(adc_ctrl_t *const p_ctrl, void *const p_extend) |
| | |
| fsp_err_t(* | offsetSet )(adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t const offset) |
| | |
| fsp_err_t(* | close )(adc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | infoGet )(adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* adc_api_t::open) (adc_ctrl_t *const p_ctrl, adc_cfg_t const *const p_cfg)

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()

**Precondition**

Configure peripheral clocks, ADC pins and IRQs prior to calling this function.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | p_cfg | Pointer to configuration structure |

### ◆ scanCfg

fsp_err_t(* adc_api_t::scanCfg) (adc_ctrl_t *const p_ctrl, void const *const p_extend)

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_ScanCfg()
- R_SDADC_ScanCfg()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | p_extend | See implementation for details |

### ◆ scanStart

fsp_err_t(* adc_api_t::scanStart) (adc_ctrl_t *const p_ctrl)

Start the scan (in case of a software trigger), or enable the hardware trigger.

**Implemented as**

- R_ADC_ScanStart()
- R_SDADC_ScanStart()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |

### ◆ scanStop

fsp_err_t(* adc_api_t::scanStop) (adc_ctrl_t *const p_ctrl)

Stop the ADC scan (in case of a software trigger), or disable the hardware trigger.

**Implemented as**

- R_ADC_ScanStop()
- R_SDADC_ScanStop()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |

### ◆ scanStatusGet

fsp_err_t(* adc_api_t::scanStatusGet) (adc_ctrl_t *const p_ctrl, adc_status_t *p_status)

Check scan status.

**Implemented as**

- R_ADC_StatusGet()
- R_SDADC_StatusGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [out] | p_status | Pointer to store current status in |

### ◆ read

fsp_err_t(* adc_api_t::read) (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data)

Read ADC conversion result.

**Implemented as**

- R_ADC_Read()
- R_SDADC_Read()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | reg_id | ADC channel to read (see enumeration adc_channel_t) |
| [in] | p_data | Pointer to variable to load value into. |

### ◆ read32

fsp_err_t(* adc_api_t::read32) (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data)

Read ADC conversion result into a 32-bit word.

**Implemented as**

- R_SDADC_Read32()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | reg_id | ADC channel to read (see enumeration adc_channel_t) |
| [in] | p_data | Pointer to variable to load value into. |

### ◆ calibrate

fsp_err_t(* adc_api_t::calibrate) (adc_ctrl_t *const p_ctrl, void *const p_extend)

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()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | p_extend | Pointer to implementation specific arguments |

### ◆ offsetSet

fsp_err_t(* adc_api_t::offsetSet) (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t const offset)

Set offset for input PGA configured for differential input. Not supported for all implementations. See implementation for details.

**Implemented as**

- R_SDADC_OffsetSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | reg_id | ADC channel to read (see enumeration adc_channel_t) |
| [in] | offset | See implementation for details. |

### ◆ close

fsp_err_t(* adc_api_t::close) (adc_ctrl_t *const p_ctrl)

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()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |

### ◆ infoGet

fsp_err_t(* adc_api_t::infoGet) (adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info)

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()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [out] | p_adc_info | Pointer to ADC information structure |

### ◆ versionGet

fsp_err_t(* adc_api_t::versionGet) (fsp_version_t *const p_version)

Retrieve the API version.

**Implemented as**

- R_ADC_VersionGet()
- R_SDADC_VersionGet()

**Precondition**

This function retrieves the API version.

**Parameters**

| | | |
|---|---|---|
| [in] | p_version | Pointer to version structure |

### ◆ adc_instance_t

struct adc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| 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. |
| void 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. |

## Typedef Documentation

### ◆ adc_ctrl_t

| typedef void adc_ctrl_t |
|---|
| ADC control block. Allocate using driver instance control structure from driver instance header file. |

## Enumeration Type Documentation

### ◆ adc_mode_t

| enum adc_mode_t | |
|---|---|
| ADC operation mode definitions | |
| Enumerator | |
| 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. |

### ◆ adc_resolution_t

| enum adc_resolution_t | |
|---|---|
| ADC data resolution definitions | |
| Enumerator | |
| 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 |

### ◆ adc_alignment_t

| enum adc_alignment_t | |
|---|---|
| ADC data alignment definitions | |
| Enumerator | |
| ADC_ALIGNMENT_RIGHT | Data alignment right. |
| ADC_ALIGNMENT_LEFT | Data alignment left. |

### ◆ adc_trigger_t

| enum adc_trigger_t | |
|---|---|
| ADC trigger mode definitions | |
| Enumerator | |
| ADC_TRIGGER_SOFTWARE | Software trigger; not for group modes. |
| ADC_TRIGGER_SYNC_ELC | Synchronous trigger via ELC. |
| ADC_TRIGGER_ASYNC_EXTERNAL | External asynchronous trigger; not for group modes. |

### ◆ adc_event_t

| enum adc_event_t | |
|---|---|
| ADC callback event definitions | |
| Enumerator | |
| 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. |
| ADC_EVENT_CONVERSION_COMPLETE | Conversion complete. |

### ◆ adc_channel_t

| enum adc_channel_t | |
|---|---|
| ADC channels | |
| Enumerator | |
| ADC_CHANNEL_0 | ADC channel 0. |
| ADC_CHANNEL_1 | ADC channel 1. |
| ADC_CHANNEL_2 | ADC channel 2. |
| ADC_CHANNEL_3 | ADC channel 3. |
| ADC_CHANNEL_4 | ADC channel 4. |
| ADC_CHANNEL_5 | ADC channel 5. |
| ADC_CHANNEL_6 | ADC channel 6. |
| ADC_CHANNEL_7 | ADC channel 7. |
| ADC_CHANNEL_8 | ADC channel 8. |
| ADC_CHANNEL_9 | ADC channel 9. |
| ADC_CHANNEL_10 | ADC channel 10. |
| ADC_CHANNEL_11 | ADC channel 11. |
| ADC_CHANNEL_12 | ADC channel 12. |
| ADC_CHANNEL_13 | ADC channel 13. |
| ADC_CHANNEL_14 | ADC channel 14. |
| ADC_CHANNEL_15 | ADC channel 15. |
| ADC_CHANNEL_16 | ADC channel 16. |
| ADC_CHANNEL_17 | ADC channel 17. |
| ADC_CHANNEL_18 | ADC channel 18. |
| ADC_CHANNEL_19 | ADC channel 19. |
| ADC_CHANNEL_20 | ADC channel 20. |

| | |
|---|---|
| ADC_CHANNEL_21 | ADC channel 21. |
| ADC_CHANNEL_22 | ADC channel 22. |
| ADC_CHANNEL_23 | ADC channel 23. |
| ADC_CHANNEL_24 | ADC channel 24. |
| ADC_CHANNEL_25 | ADC channel 25. |
| ADC_CHANNEL_26 | ADC channel 26. |
| ADC_CHANNEL_27 | ADC channel 27. |
| ADC_CHANNEL_DUPLEX_A | Data duplexing register A. |
| ADC_CHANNEL_DUPLEX_B | Data duplexing register B. |
| ADC_CHANNEL_DUPLEX | Data duplexing register. |
| ADC_CHANNEL_TEMPERATURE | Temperature sensor output. |
| ADC_CHANNEL_VOLT | Internal reference voltage. |

### ◆ adc_state_t

| enum adc_state_t | |
|---|---|
| ADC states. | |
| Enumerator | |
| ADC_STATE_IDLE | ADC is idle. |
| ADC_STATE_SCAN_IN_PROGRESS | ADC scan in progress. |

## 4.3.2 BLE Interface

Interfaces

### Detailed Description

Interface for Bluetooth Low Energy functions.

# Summary

The BLE interface for the Bluetooth Low Energy (BLE) peripheral provides Bluetooth Low Energy functionality.

The Bluetooth Low Energy interface can be implemented by:

- Bluetooth Low Energy Library (r_ble)

## Macros

| | |
|---|---|
| #define | BLE_VERSION_MAJOR |
| #define | BLE_VERSION_MINOR |
| #define | BLE_LIB_ALL_FEATS |
| #define | BLE_LIB_BALANCE |
| #define | BLE_LIB_COMPACT |

## Macro Definition Documentation

### ◆ BLE_VERSION_MAJOR

| |
|---|
| #define BLE_VERSION_MAJOR |
| BLE Module Major Version. |

### ◆ BLE_VERSION_MINOR

| |
|---|
| #define BLE_VERSION_MINOR |
| BLE Module Minor Version. |

### ◆ BLE_LIB_ALL_FEATS

| |
|---|
| #define BLE_LIB_ALL_FEATS |
| BLE Protocol Stack Library All Features type. |

### ◆ BLE_LIB_BALANCE

| |
|---|
| #define BLE_LIB_BALANCE |
| BLE Protocol Stack Library Balance type. |

### ◆ BLE_LIB_COMPACT

| #define BLE_LIB_COMPACT |
|---|
| BLE Protocol Stack Library Compacy type. |

## 4.3.3 CAC Interface

Interfaces

### Detailed Description

Interface for clock frequency accuracy measurements.

# 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 to be measured.

Implemented by: Clock Frequency Accuracy Measurement Circuit (r_cac)

### Data Structures

| | |
|---|---|
| struct | cac_ref_clock_config_t |
| struct | cac_meas_clock_config_t |
| struct | cac_callback_args_t |
| struct | cac_cfg_t |
| struct | cac_api_t |
| struct | cac_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | cac_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | cac_event_t |
| enum | cac_clock_type_t |
| enum | cac_clock_source_t |

| | |
|---|---|
| enum | cac_ref_divider_t |
| enum | cac_ref_digfilter_t |
| enum | cac_ref_edge_t |
| enum | cac_meas_divider_t |

## Data Structure Documentation

### ◆ cac_ref_clock_config_t

| struct cac_ref_clock_config_t | | |
|---|---|---|
| Structure defining the settings that apply to reference clock configuration. | | |
| Data Fields | | |
| 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. |

### ◆ cac_meas_clock_config_t

| struct cac_meas_clock_config_t | | |
|---|---|---|
| Structure defining the settings that apply to measurement clock configuration. | | |
| Data Fields | | |
| cac_meas_divider_t | divider | Divider specification for the Measurement clock. |
| cac_clock_source_t | clock | Clock source for the Measurement clock. |

### ◆ cac_callback_args_t

| struct cac_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| cac_event_t | event | The event can be used to identify what caused the callback. |
| void const * | p_context | Value provided in configuration structure. |

### ◆ cac_cfg_t

| struct cac_cfg_t | |
|---|---|
| CAC Configuration | |
| **Data Fields** | |
| 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. |
| | |
| IRQn_Type | mendi_irq |
| | Measurement End IRQ number. |
| | |
| IRQn_Type | ovfi_irq |
| | Measurement Overflow IRQ number. |
| | |
| IRQn_Type | ferri_irq |
| | Frequency Error IRQ number. |
| | |
| uint8_t | mendi_ipl |
| | Measurement end interrupt priority. |
| | |
| uint8_t | ovfi_ipl |

| | |
|---|---|
| | Overflow interrupt priority. |
| | |
| uint8_t | ferri_ipl |
| | Frequency error interrupt priority. |
| | |
| void(* | p_callback )(cac_callback_args_t *p_args) |
| | Callback provided when a CAC interrupt ISR occurs. |
| | |
| void const * | p_context |
| | Passed to user callback in cac_callback_args_t. |
| | |
| void const * | p_extend |
| | CAC hardware dependent configuration */. |
| | |

### ◆ cac_api_t

| | |
|---|---|
| struct cac_api_t | |
| CAC functions implemented at the HAL layer API | |
| **Data Fields** | |
| fsp_err_t(* | open )(cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | startMeasurement )(cac_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stopMeasurement )(cac_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | read )(cac_ctrl_t *const p_ctrl, uint16_t *const p_counter) |
| | |
| fsp_err_t(* | close )(cac_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* cac_api_t::open) (cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)

Open function for CAC device.

**Parameters**

| | | |
|---|---|---|
| [out] | p_ctrl | Pointer to CAC device control. Must be declared by user. |
| [in] | cac_cfg_t | Pointer to CAC configuration structure. |

### ◆ startMeasurement

fsp_err_t(* cac_api_t::startMeasurement) (cac_ctrl_t *const p_ctrl)

Begin a measurement for the CAC peripheral.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CAC device control. |

### ◆ stopMeasurement

fsp_err_t(* cac_api_t::stopMeasurement) (cac_ctrl_t *const p_ctrl)

End a measurement for the CAC peripheral.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CAC device control. |

### ◆ read

fsp_err_t(* cac_api_t::read) (cac_ctrl_t *const p_ctrl, uint16_t *const p_counter)

Read function for CAC peripheral.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control for the CAC device context. |
| [in] | p_counter | Pointer to variable in which to store the current CACNTBR register contents. |

### ◆ close

| fsp_err_t(* cac_api_t::close) (cac_ctrl_t *const p_ctrl) |
|---|

Close function for CAC device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CAC device control. |

### ◆ versionGet

| fsp_err_t(* cac_api_t::versionGet) (fsp_version_t *p_version) |
|---|

Get the CAC API and code version information.

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | is value returned. |

### ◆ cac_instance_t

| struct cac_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| 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. |

## Typedef Documentation

### ◆ cac_ctrl_t

| typedef void cac_ctrl_t |
|---|

CAC control block. Allocate an instance specific control block to pass into the CAC API calls.

**Implemented as**

- cac_instance_ctrl_t

## Enumeration Type Documentation

### ◆ cac_event_t

| enum cac_event_t | |
|---|---|
| Event types returned by the ISR callback when used in CAC interrupt mode | |
| Enumerator | |
| CAC_EVENT_FREQUENCY_ERROR | Frequency error. |
| CAC_EVENT_MEASUREMENT_COMPLETE | Measurement complete. |
| CAC_EVENT_COUNTER_OVERFLOW | Counter overflow. |

### ◆ cac_clock_type_t

| enum cac_clock_type_t | |
|---|---|
| Enumeration of the two possible clocks. | |
| Enumerator | |
| CAC_CLOCK_MEASURED | Measurement clock. |
| CAC_CLOCK_REFERENCE | Reference clock. |

### ◆ cac_clock_source_t

| enum cac_clock_source_t | |
|---|---|
| Enumeration of the possible clock sources for both the reference and measurement clocks. | |
| Enumerator | |
| 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 (Low speed on chip oscillator) |
| CAC_CLOCK_SOURCE_PCLKB | PCLKB (Peripheral Clock B) |
| CAC_CLOCK_SOURCE_IWDT | IWDT-dedicated on-chip oscillator. |
| CAC_CLOCK_SOURCE_EXTERNAL | Externally supplied measurement clock on CACREF pin. |

### ◆ cac_ref_divider_t

| enum cac_ref_divider_t | |
|---|---|
| Enumeration of available dividers for the reference clock. | |
| Enumerator | |
| 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. |

### ◆ cac_ref_digfilter_t

| enum cac_ref_digfilter_t | |
|---|---|
| Enumeration of available digital filter settings for an external reference clock. | |
| Enumerator | |
| 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. |

### ◆ cac_ref_edge_t

| enum cac_ref_edge_t | |
|---|---|
| Enumeration of available edge detect settings for the reference clock. | |
| Enumerator | |
| CAC_REF_EDGE_RISE | Rising edge detect for the Reference clock. |
| CAC_REF_EDGE_FALL | Falling edge detect for the Reference clock. |
| CAC_REF_EDGE_BOTH | Both Rising and Falling edges detect for the Reference clock. |

### ◆ cac_meas_divider_t

| enum cac_meas_divider_t | |
|---|---|
| Enumeration of available dividers for the measurement clock | |
| Enumerator | |
| 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. |

## 4.3.4 CAN Interface

Interfaces

### Detailed Description

Interface for CAN peripheral.

# 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

Implemented by: Controller Area Network (r_can)

### Data Structures

| | |
|---|---|
| struct | can_bit_timing_cfg_t |
| struct | can_frame_t |
| struct | can_mailbox_t |
| struct | can_callback_args_t |

| | |
|---|---|
| struct | can_cfg_t |
| struct | can_api_t |
| struct | can_instance_t |

### Typedefs

| | |
|---|---|
| typedef uint32_t | can_id_t |
| typedef void | can_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | can_event_t |
| enum | can_status_t |
| enum | can_error_t |
| enum | can_operation_mode_t |
| enum | can_test_mode_t |
| enum | can_id_mode_t |
| enum | can_frame_type_t |
| enum | can_message_mode_t |
| enum | can_clock_source_t |
| enum | can_time_segment1_t |
| enum | can_time_segment2_t |
| enum | can_sync_jump_width_t |
| enum | can_mailbox_send_receive_t |

## Data Structure Documentation

### ◆ can_bit_timing_cfg_t

| struct can_bit_timing_cfg_t | | |
|---|---|---|
| CAN bit rate configuration. | | |
| Data Fields | | |
| 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. |

### ◆ can_frame_t

| struct can_frame_t | | |
|---|---|---|
| CAN data Frame | | |
| Data Fields | | |
| 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. |

### ◆ can_mailbox_t

| struct can_mailbox_t | | |
|---|---|---|
| CAN Mailbox | | |
| Data Fields | | |
| 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. |

### ◆ can_callback_args_t

| struct can_callback_args_t | | |
|---|---|---|
| CAN callback parameter definition | | |
| Data Fields | | |
| uint32_t | channel | Device channel number. |
| can_event_t | event | Event code. |
| uint32_t | mailbox | Mailbox number of interrupt source. |
| can_frame_t * | p_frame | Pointer to the received frame. |
| void const * | p_context | Context provided to user during callback. |

### ◆ can_cfg_t

| struct can_cfg_t |
|---|
| CAN Configuration |

| Data Fields | |
|---|---|
| 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. |
| | |
| can_operation_mode_t | operation_mode |
| | CAN operation mode. |
| | |
| can_test_mode_t | test_mode |
| | CAN operation mode. |
| | |
| 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 | ipl |
| | Error/Transmit/Receive interrupt priority. |
| | |
| 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. |
| | |

### ◆ can_api_t

| | |
|---|---|
| struct can_api_t | |
| Shared Interface definition for CAN | |
| **Data Fields** | |
| fsp_err_t(* | open )(can_ctrl_t *const p_ctrl, can_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | write )(can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t *const p_frame) |
| | |
| fsp_err_t(* | close )(can_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | modeTransition )(can_ctrl_t *const p_api_ctrl, can_operation_mode_t |

| | |
|---|---|
| | operation_mode, can_test_mode_t test_mode) |
| | |
| fsp_err_t(* | infoGet )(can_ctrl_t *const p_ctrl, can_info_t *const p_info) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* can_api_t::open) (can_ctrl_t *const p_ctrl, can_cfg_t const *const p_cfg)

Open function for CAN device

**Implemented as**

- R_CAN_Open()

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to the CAN control block. Must be declared by user. Value set here. |
| [in] | can_cfg_t | Pointer to CAN configuration structure. All elements of this structure must be set by user. |

### ◆ write

fsp_err_t(* can_api_t::write) (can_ctrl_t *const p_ctrl, uint32_t mailbox, can_frame_t *const p_frame)

Write function for CAN device

**Implemented as**

- R_CAN_Write()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the CAN control block. |
| [in] | mailbox | Mailbox (number) to write to. |
| [in] | p_frame | Pointer for frame of CAN ID, DLC, data and frame type to write. |

### ◆ close

fsp_err_t(* can_api_t::close) (can_ctrl_t *const p_ctrl)

Close function for CAN device

**Implemented as**

- R_CAN_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the CAN control block. |

### ◆ modeTransition

fsp_err_t(* can_api_t::modeTransition) (can_ctrl_t *const p_api_ctrl, can_operation_mode_t operation_mode, can_test_mode_t test_mode)

Mode Transition function for CAN device

**Implemented as**

- R_CAN_ModeTransition()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the CAN control block. |
| [in] | operation_mode | Destination CAN operation state. |
| [in] | test_mode | Destination CAN test state. |

### ◆ infoGet

fsp_err_t(* can_api_t::infoGet) (can_ctrl_t *const p_ctrl, can_info_t *const p_info)

Get CAN channel info.

**Implemented as**

- R_CAN_InfoGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Handle for channel (pointer to channel control block) |
| [out] | p_info | Memory address to return channel specific data to. |

### ◆ versionGet

| fsp_err_t(* can_api_t::versionGet) (fsp_version_t *const p_version) |
|---|

Version get function for CAN device

**Implemented as**

- R_CAN_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_version | Pointer to the memory to store the version information |

### ◆ can_instance_t

| struct can_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| 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. |

## Typedef Documentation

### ◆ can_id_t

| typedef uint32_t can_id_t |
|---|
| CAN Id |

### ◆ can_ctrl_t

| typedef void can_ctrl_t |
|---|

CAN control block. Allocate an instance specific control block to pass into the CAN API calls.

**Implemented as**

- can_instance_ctrl_t

## Enumeration Type Documentation

### ◆ can_event_t

| enum can_event_t | |
|---|---|
| CAN event codes | |
| Enumerator | |
| CAN_EVENT_ERR_WARNING | Error Warning event. |
| CAN_EVENT_ERR_PASSIVE | Error Passive event. |
| CAN_EVENT_ERR_BUS_OFF | Bus Off event. |
| CAN_EVENT_BUS_RECOVERY | Bus Off Recovery event. |
| CAN_EVENT_MAILBOX_MESSAGE_LOST | Mailbox has been overrun. |
| CAN_EVENT_RX_COMPLETE | Receive complete event. |
| CAN_EVENT_TX_COMPLETE | Transmit complete event. |

### ◆ can_status_t

| enum can_status_t | |
|---|---|
| CAN Status | |
| Enumerator | |
| CAN_STATUS_NEW_DATA | New Data status flag. |
| CAN_STATUS_SENT_DATA | Sent Data status flag. |
| CAN_STATUS_RECEIVE_FIFO | Receive FIFO status flag (Not supported) |
| CAN_STATUS_TRANSMIT_FIFO | Transmit FIFO status flag (Not supported) |
| CAN_STATUS_NORMAL_MBOX_MESSAGE_LOST | Normal mailbox message lost status flag. |
| CAN_STATUS_FIFO_MBOX_MESSAGE_LOST | FIFO mailbox message lost status flag (Not Supported) |
| CAN_STATUS_TRANSMISSION_ABORT | Transmission abort status flag. |
| CAN_STATUS_ERROR | Error status flag. |
| CAN_STATUS_RESET_MODE | Reset mode status flag. |
| CAN_STATUS_HALT_MODE | Halt mode status flag. |
| CAN_STATUS_SLEEP_MODE | Sleep mode status flag. |
| CAN_STATUS_ERROR_PASSIVE | Error-passive status flag. |
| CAN_STATUS_BUS_OFF | Bus-off status flag. |

### ◆ can_error_t

| enum can_error_t | |
|---|---|
| CAN Error Code | |
| Enumerator | |
| CAN_ERROR_STUFF | Stuff Error. |
| CAN_ERROR_FORM | Form Error. |
| CAN_ERROR_ACK | ACK Error. |
| CAN_ERROR_CRC | CRC Error. |
| CAN_ERROR_BIT_RECESSIVE | Bit Error (recessive) Error. |
| CAN_ERROR_BIT_DOMINANT | Bit Error (dominant) Error. |
| CAN_ERROR_ACK_DELIMITER | ACK Delimiter Error. |
| CAN_ERROR_ERROR_DISPLAY_MODE | Error Display mode. |

### ◆ can_operation_mode_t

| enum can_operation_mode_t | |
|---|---|
| CAN Operation modes | |
| Enumerator | |
| CAN_OPERATION_MODE_NORMAL | CAN Normal Operation Mode. |
| CAN_OPERATION_MODE_RESET | CAN Reset Operation Mode. |
| CAN_OPERATION_MODE_HALT | CAN Halt Operation Mode. |
| CAN_OPERATION_MODE_SLEEP | CAN SLEEP Operation Mode. |

### ◆ can_test_mode_t

| enum can_test_mode_t | |
|---|---|
| CAN Test modes | |
| Enumerator | |
| CAN_TEST_MODE_DISABLED | CAN Test Mode Disabled. |
| CAN_TEST_MODE_LISTEN | CAN Test Listen Mode. |
| CAN_TEST_MODE_LOOPBACK_EXTERNAL | CAN Test External Loopback Mode. |
| CAN_TEST_MODE_LOOPBACK_INTERNAL | CAN Test Internal Loopback Mode. |

### ◆ can_id_mode_t

| enum can_id_mode_t | |
|---|---|
| CAN ID modes | |
| Enumerator | |
| CAN_ID_MODE_STANDARD | Standard IDs of 11 bits used. |
| CAN_ID_MODE_EXTENDED | Extended IDs of 29 bits used. |

### ◆ can_frame_type_t

| enum can_frame_type_t | |
|---|---|
| CAN frame types | |
| Enumerator | |
| CAN_FRAME_TYPE_DATA | Data frame type. |
| CAN_FRAME_TYPE_REMOTE | Remote frame type. |

### ◆ can_message_mode_t

| enum can_message_mode_t | |
|---|---|
| CAN Message Modes | |
| Enumerator | |
| 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. |

### ◆ can_clock_source_t

| enum can_clock_source_t | |
|---|---|
| CAN Source Clock | |
| Enumerator | |
| CAN_CLOCK_SOURCE_PCLKB | PCLKB is the source of the CAN Clock. |
| CAN_CLOCK_SOURCE_CANMCLK | CANMCLK is the source of the CAN Clock. |

### ◆ can_time_segment1_t

| enum can_time_segment1_t | |
|---|---|
| CAN Time Segment 1 Time Quanta | |
| Enumerator | |
| 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. |

### ◆ can_time_segment2_t

| enum can_time_segment2_t | |
|---|---|
| CAN Time Segment 2 Time Quanta | |
| Enumerator | |
| 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. |
| 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. |

### ◆ can_sync_jump_width_t

| enum can_sync_jump_width_t | |
|---|---|
| CAN Synchronization Jump Width Time Quanta | |
| Enumerator | |
| 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. |

### ◆ can_mailbox_send_receive_t

| enum can_mailbox_send_receive_t | |
|---|---|
| CAN Mailbox type | |
| Enumerator | |
| CAN_MAILBOX_RECEIVE | Mailbox is for receiving. |
| CAN_MAILBOX_TRANSMIT | Mailbox is for sending. |

## 4.3.5 CGC Interface

Interfaces

### Detailed Description

Interface for clock generation.

# 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.

The CGC interface is implemented by:

- Clock Generation Circuit (r_cgc)

### Data Structures

| | |
|---|---|
| struct | cgc_callback_args_t |
| struct | cgc_pll_cfg_t |
| union | cgc_divider_cfg_t |
| struct | cgc_cfg_t |
| struct | cgc_clocks_cfg_t |

| | |
|---|---|
| struct | cgc_api_t |
| struct | cgc_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | cgc_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | cgc_event_t |
| enum | cgc_clock_t |
| enum | cgc_pll_div_t |
| enum | cgc_pll_mul_t |
| enum | cgc_sys_clock_div_t |
| enum | cgc_usb_clock_div_t |
| enum | cgc_clock_change_t |

## Data Structure Documentation

### ◆ cgc_callback_args_t

| struct cgc_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| cgc_event_t | event | The event can be used to identify what caused the callback. |
| void const * | p_context | Placeholder for user data. |

### ◆ cgc_pll_cfg_t

| struct cgc_pll_cfg_t | | |
|---|---|---|
| Clock configuration structure - Used as an input parameter to the cgc_api_t::clockStart function for the PLL clock. | | |
| Data Fields | | |
| cgc_clock_t | source_clock | PLL source clock (main oscillator or HOCO) |
| cgc_pll_div_t | divider | PLL divider. |
| cgc_pll_mul_t | multiplier | PLL multiplier. |

### ◆ cgc_divider_cfg_t

| union cgc_divider_cfg_t | | |
|---|---|---|
| Clock configuration structure - Used as an input parameter to the cgc_api_t::systemClockSet and cgc_api_t::systemClockGet functions. | | |
| Data Fields | | |
| uint32_t | sckdivcr_w | (@ 0x4001E020) System clock Division control register |
| struct cgc_divider_cfg_t | __unnamed__ | |

### ◆ cgc_cfg_t

| struct cgc_cfg_t |
|---|
| Configuration options. |

### ◆ cgc_clocks_cfg_t

| struct cgc_clocks_cfg_t | | |
|---|---|---|
| Clock configuration | | |
| Data Fields | | |
| cgc_clock_t | system_clock | System clock source enumeration. |
| cgc_pll_cfg_t | pll_cfg | PLL configuration structure. |
| cgc_divider_cfg_t | divider_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 | mainosc_state | State of Main oscillator. |
| cgc_clock_change_t | pll_state | State of PLL. |

### ◆ cgc_api_t

| struct cgc_api_t | |
|---|---|
| CGC functions implemented at the HAL layer follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(cgc_ctrl_t *const p_ctrl, cgc_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | clocksCfg )(cgc_ctrl_t *const p_ctrl, cgc_clocks_cfg_t const *const p_clock_cfg) |
| | |
| fsp_err_t(* | clockStart )(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, |

| | |
|---|---|
| | cgc_pll_cfg_t const *const p_pll_cfg) |
| | |
| fsp_err_t(* | clockStop )(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source) |
| | |
| fsp_err_t(* | clockCheck )(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source) |
| | |
| fsp_err_t(* | systemClockSet )(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_divider_cfg_t const *const p_divider_cfg) |
| | |
| fsp_err_t(* | systemClockGet )(cgc_ctrl_t *const p_ctrl, cgc_clock_t *const p_clock_source, cgc_divider_cfg_t *const p_divider_cfg) |
| | |
| fsp_err_t(* | oscStopDetectEnable )(cgc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | oscStopDetectDisable )(cgc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | oscStopStatusClear )(cgc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | close )(cgc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* cgc_api_t::open) (cgc_ctrl_t *const p_ctrl, cgc_cfg_t const *const p_cfg)

Initial configuration

**Implemented as**

- R_CGC_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | p_cfg | Pointer to configuration |

### ◆ clocksCfg

fsp_err_t(* cgc_api_t::clocksCfg) (cgc_ctrl_t *const p_ctrl, cgc_clocks_cfg_t const *const p_clock_cfg)

Configure all system clocks.

**Implemented as**

- R_CGC_ClocksCfg()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | p_clock_cfg | Pointer to desired configuration of system clocks |

### ◆ clockStart

fsp_err_t(* cgc_api_t::clockStart) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_pll_cfg_t const *const p_pll_cfg)

Start a clock.

**Implemented as**

- R_CGC_ClockStart()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | clock_source | Clock source to start |
| [in] | p_pll_cfg | Pointer to PLL configuration, can be NULL if clock_source is not CGC_CLOCK_PLL |

### ◆ clockStop

fsp_err_t(* cgc_api_t::clockStop) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)

Stop a clock.

**Implemented as**

- R_CGC_ClockStop()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | clock_source | The clock source to stop |

### ◆ clockCheck

fsp_err_t(* cgc_api_t::clockCheck) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)

Check the stability of the selected clock.

**Implemented as**

- R_CGC_ClockCheck()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | clock_source | Which clock source to check for stability |

### ◆ systemClockSet

fsp_err_t(* cgc_api_t::systemClockSet) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_divider_cfg_t const *const p_divider_cfg)

Set the system clock.

**Implemented as**

- R_CGC_SystemClockSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | clock_source | Clock source to set as system clock |
| [in] | p_divider_cfg | Pointer to the clock divider configuration |

### ◆ systemClockGet

fsp_err_t(* cgc_api_t::systemClockGet) (cgc_ctrl_t *const p_ctrl, cgc_clock_t *const p_clock_source, cgc_divider_cfg_t *const p_divider_cfg)

Get the system clock information.

**Implemented as**

- R_CGC_SystemClockGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [out] | p_clock_source | Returns the current system clock |
| [out] | p_divider_cfg | Returns the current system clock dividers |

### ◆ oscStopDetectEnable

fsp_err_t(* cgc_api_t::oscStopDetectEnable) (cgc_ctrl_t *const p_ctrl)

Enable and optionally register a callback for Main Oscillator stop detection.

**Implemented as**

- R_CGC_OscStopDetectEnable()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | p_callback | 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. |
| [in] | enable | Enable/disable Oscillation Stop Detection |

### ◆ oscStopDetectDisable

fsp_err_t(* cgc_api_t::oscStopDetectDisable) (cgc_ctrl_t *const p_ctrl)

Disable Main Oscillator stop detection.

**Implemented as**

- R_CGC_OscStopDetectDisable()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |

### ◆ oscStopStatusClear

fsp_err_t(* cgc_api_t::oscStopStatusClear) (cgc_ctrl_t *const p_ctrl)

Clear the oscillator stop detection flag.

**Implemented as**

- R_CGC_OscStopStatusClear()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |

### ◆ close

fsp_err_t(* cgc_api_t::close) (cgc_ctrl_t *const p_ctrl)

Close the CGC driver.

**Implemented as**

- R_CGC_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |

### ◆ versionGet

fsp_err_t(* cgc_api_t::versionGet) (fsp_version_t *p_version)

Gets the CGC driver version.

**Implemented as**

- R_CGC_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version used |

### ◆ cgc_instance_t

struct cgc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| cgc_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| cgc_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. |

## Typedef Documentation

### ◆ cgc_ctrl_t

typedef void cgc_ctrl_t

CGC control block. Allocate an instance specific control block to pass into the CGC API calls.

**Implemented as**

- cgc_instance_ctrl_t

## Enumeration Type Documentation

### ◆ cgc_event_t

enum cgc_event_t

Events that can trigger a callback function

| Enumerator | |
|---|---|
| CGC_EVENT_OSC_STOP_DETECT | Oscillator stop detection has caused the event. |

### ◆ cgc_clock_t

| enum cgc_clock_t | |
|---|---|
| System clock source identifiers - The source of ICLK, BCLK, FCLK, PCLKS A-D and UCLK prior to the system clock divider | |
| Enumerator | |
| 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. |

### ◆ cgc_pll_div_t

| enum cgc_pll_div_t | |
|---|---|
| PLL divider values | |
| Enumerator | |
| CGC_PLL_DIV_1 | PLL divider of 1. |
| CGC_PLL_DIV_2 | PLL divider of 2. |
| CGC_PLL_DIV_3 | PLL divider of 3 (S7, S5 only) |
| CGC_PLL_DIV_4 | PLL divider of 4 (S3 only) |

### ◆ cgc_pll_mul_t

| enum cgc_pll_mul_t | |
|---|---|
| PLL multiplier values | |
| Enumerator | |
| CGC_PLL_MUL_8_0 | PLL multiplier of 8.0. |
| CGC_PLL_MUL_9_0 | PLL multiplier of 9.0. |
| CGC_PLL_MUL_10_0 | PLL multiplier of 10.0. |
| CGC_PLL_MUL_10_5 | PLL multiplier of 10.5. |
| CGC_PLL_MUL_11_0 | PLL multiplier of 11.0. |
| CGC_PLL_MUL_11_5 | PLL multiplier of 11.5. |
| CGC_PLL_MUL_12_0 | PLL multiplier of 12.0. |
| CGC_PLL_MUL_12_5 | PLL multiplier of 12.5. |
| CGC_PLL_MUL_13_0 | PLL multiplier of 13.0. |
| CGC_PLL_MUL_13_5 | PLL multiplier of 13.5. |
| CGC_PLL_MUL_14_0 | PLL multiplier of 14.0. |
| CGC_PLL_MUL_14_5 | PLL multiplier of 14.5. |
| CGC_PLL_MUL_15_0 | PLL multiplier of 15.0. |
| CGC_PLL_MUL_15_5 | PLL multiplier of 15.5. |
| CGC_PLL_MUL_16_0 | PLL multiplier of 16.0. |
| CGC_PLL_MUL_16_5 | PLL multiplier of 16.5. |
| CGC_PLL_MUL_17_0 | PLL multiplier of 17.0. |
| CGC_PLL_MUL_17_5 | PLL multiplier of 17.5. |
| CGC_PLL_MUL_18_0 | PLL multiplier of 18.0. |
| CGC_PLL_MUL_18_5 | PLL multiplier of 18.5. |
| CGC_PLL_MUL_19_0 | PLL multiplier of 19.0. |

| | |
|---|---|
| CGC_PLL_MUL_19_5 | PLL multiplier of 19.5. |
| CGC_PLL_MUL_20_0 | PLL multiplier of 20.0. |
| CGC_PLL_MUL_20_5 | PLL multiplier of 20.5. |
| CGC_PLL_MUL_21_0 | PLL multiplier of 21.0. |
| CGC_PLL_MUL_21_5 | PLL multiplier of 21.5. |
| CGC_PLL_MUL_22_0 | PLL multiplier of 22.0. |
| CGC_PLL_MUL_22_5 | PLL multiplier of 22.5. |
| CGC_PLL_MUL_23_0 | PLL multiplier of 23.0. |
| CGC_PLL_MUL_23_5 | PLL multiplier of 23.5. |
| CGC_PLL_MUL_24_0 | PLL multiplier of 24.0. |
| CGC_PLL_MUL_24_5 | PLL multiplier of 24.5. |
| CGC_PLL_MUL_25_0 | PLL multiplier of 25.0. |
| CGC_PLL_MUL_25_5 | PLL multiplier of 25.5. |
| CGC_PLL_MUL_26_0 | PLL multiplier of 26.0. |
| CGC_PLL_MUL_26_5 | PLL multiplier of 26.5. |
| CGC_PLL_MUL_27_0 | PLL multiplier of 27.0. |
| CGC_PLL_MUL_27_5 | PLL multiplier of 27.5. |
| CGC_PLL_MUL_28_0 | PLL multiplier of 28.0. |
| CGC_PLL_MUL_28_5 | PLL multiplier of 28.5. |
| CGC_PLL_MUL_29_0 | PLL multiplier of 29.0. |
| CGC_PLL_MUL_29_5 | PLL multiplier of 29.5. |
| CGC_PLL_MUL_30_0 | PLL multiplier of 30.0. |
| CGC_PLL_MUL_31_0 | PLL multiplier of 31.0. |

### ◆ cgc_sys_clock_div_t

| enum cgc_sys_clock_div_t | |
|---|---|
| System clock divider vlues - The individually selectable divider of each of the system clocks, ICLK, BCLK, FCLK, PCLKS A-D. | |
| Enumerator | |
| 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. |

### ◆ cgc_usb_clock_div_t

| enum cgc_usb_clock_div_t | |
|---|---|
| USB clock divider values | |
| Enumerator | |
| 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. |

### ◆ cgc_clock_change_t

| enum cgc_clock_change_t | |
|---|---|
| Clock options | |
| Enumerator | |
| CGC_CLOCK_CHANGE_START | Start the clock. |
| CGC_CLOCK_CHANGE_STOP | Stop the clock. |
| CGC_CLOCK_CHANGE_NONE | No change to the clock. |

## 4.3.6 Comparator Interface

Interfaces

### Detailed Description

Interface for comparators.

# Summary

The comparator interface provides standard comparator functionality, including generating an event when the comparator result changes.

Implemented by:

- High-Speed Analog Comparator (r_acmphs)
- Low-Power Analog Comparator (r_acmplp)

### Data Structures

| | |
|---|---|
| struct | comparator_info_t |
| struct | comparator_status_t |
| struct | comparator_callback_args_t |
| struct | comparator_cfg_t |
| struct | comparator_api_t |
| struct | comparator_instance_t |

### Macros

| | |
|---|---|
| #define | COMPARATOR_API_VERSION_MAJOR |

### Typedefs

| | |
|---|---|
| typedef void | comparator_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | comparator_mode_t |
| enum | comparator_trigger_t |
| enum | comparator_polarity_invert_t |
| enum | comparator_pin_output_t |
| enum | comparator_filter_t |
| enum | comparator_state_t |

## Data Structure Documentation

### ◆ comparator_info_t

| struct comparator_info_t | | |
|---|---|---|
| Comparator information. | | |
| Data Fields | | |
| uint32_t | min_stabilization_wait_us | Minimum stabilization wait time in microseconds. |

### ◆ comparator_status_t

| struct comparator_status_t | | |
|---|---|---|
| Comparator status. | | |
| Data Fields | | |
| comparator_state_t | state | Current comparator state. |

### ◆ comparator_callback_args_t

| struct comparator_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| 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. |

### ◆ comparator_cfg_t

| struct comparator_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| uint8_t | channel |
| | Hardware channel used. |
| | |
| 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. |
| | |
| uint8_t | vref_select |
| | Internal Vref Select. |
| | |
| uint8_t | ipl |
| | Interrupt priority. |
| | |
| | |

| | |
|---|---|
| IRQn_Type | irq |
| | NVIC interrupt number. |
| | |
| void(* | p_callback )(comparator_callback_args_t *p_args) |
| | |
| void const * | p_context |
| | |
| void const * | p_extend |
| | Comparator hardware dependent configuration. |
| | |

## Field Documentation

### ◆ p_callback

void(* comparator_cfg_t::p_callback) (comparator_callback_args_t *p_args)

Callback called when comparator event occurs.

### ◆ p_context

void const* comparator_cfg_t::p_context

Placeholder for user data. Passed to the user callback in comparator_callback_args_t.

### ◆ comparator_api_t

struct comparator_api_t

Comparator functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(comparator_ctrl_t *const p_ctrl, comparator_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | outputEnable )(comparator_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | infoGet )(comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info) |
| | |
| fsp_err_t(* | statusGet )(comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status) |
| | |
| | |

| | |
|---|---|
| fsp_err_t(* | close )(comparator_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* comparator_api_t::open) (comparator_ctrl_t *const p_ctrl, comparator_cfg_t const *const p_cfg)

Initialize the comparator.

**Implemented as**

- R_ACMPHS_Open()
- R_ACMPLP_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | p_cfg | Pointer to configuration |

### ◆ outputEnable

fsp_err_t(* comparator_api_t::outputEnable) (comparator_ctrl_t *const p_ctrl)

Start the comparator.

**Implemented as**

- R_ACMPHS_OutputEnable()
- R_ACMPLP_OutputEnable()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |

### ◆ infoGet

fsp_err_t(* comparator_api_t::infoGet) (comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info)

Provide information such as the recommended minimum stabilization wait time.

**Implemented as**

- R_ACMPHS_InfoGet()
- R_ACMPLP_InfoGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [out] | p_info | Comparator information stored here |

### ◆ statusGet

fsp_err_t(* comparator_api_t::statusGet) (comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status)

Provide current comparator status.

**Implemented as**

- R_ACMPHS_StatusGet()
- R_ACMPLP_StatusGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [out] | p_status | Status stored here |

### ◆ close

fsp_err_t(* comparator_api_t::close) (comparator_ctrl_t *const p_ctrl)

Stop the comparator.

**Implemented as**

- R_ACMPHS_Close()
- R_ACMPLP_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |

### ◆ versionGet

fsp_err_t(* comparator_api_t::versionGet) (fsp_version_t *const p_version)

Retrieve the API version.

**Implemented as**

- R_ACMPHS_VersionGet()
- R_ACMPLP_VersionGet()

**Precondition**

This function retrieves the API version.

**Parameters**

| | | |
|---|---|---|
| [in] | p_version | Pointer to version structure |

### ◆ comparator_instance_t

struct comparator_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| 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. |

## Macro Definition Documentation

### ◆ COMPARATOR_API_VERSION_MAJOR

#define COMPARATOR_API_VERSION_MAJOR

Includes board and MCU related header files. Version Number of API.

## Typedef Documentation

### ◆ comparator_ctrl_t

typedef void comparator_ctrl_t

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

## Enumeration Type Documentation

### ◆ comparator_mode_t

| enum comparator_mode_t | |
|---|---|
| Select whether to invert the polarity of the comparator output. | |
| Enumerator | |
| COMPARATOR_MODE_NORMAL | Normal mode. |
| COMPARATOR_MODE_WINDOW | Window mode, not supported by all implementations. |

### ◆ comparator_trigger_t

| enum comparator_trigger_t | |
|---|---|
| Trigger type: rising edge, falling edge, both edges, low level. | |
| Enumerator | |
| COMPARATOR_TRIGGER_RISING | Rising edge trigger. |
| COMPARATOR_TRIGGER_FALLING | Falling edge trigger. |
| COMPARATOR_TRIGGER_BOTH_EDGE | Both edges trigger. |

### ◆ comparator_polarity_invert_t

| enum comparator_polarity_invert_t | |
|---|---|
| Select whether to invert the polarity of the comparator output. | |
| Enumerator | |
| COMPARATOR_POLARITY_INVERT_OFF | Do not invert polarity. |
| COMPARATOR_POLARITY_INVERT_ON | Invert polarity. |

### ◆ comparator_pin_output_t

| enum comparator_pin_output_t | |
|---|---|
| Select whether to include the comparator output on the output pin. | |
| Enumerator | |
| COMPARATOR_PIN_OUTPUT_OFF | Do not include comparator output on output pin. |
| COMPARATOR_PIN_OUTPUT_ON | Include comparator output on output pin. |

### ◆ comparator_filter_t

| enum comparator_filter_t | |
|---|---|
| Comparator digital filtering sample clock divisor settings. | |
| Enumerator | |
| 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. |

### ◆ comparator_state_t

| enum comparator_state_t | |
|---|---|
| Current comparator state. | |
| Enumerator | |
| COMPARATOR_STATE_OUTPUT_LOW | VCMP < VREF if polarity is not inverted, VCMP > VREF if inverted. |
| COMPARATOR_STATE_OUTPUT_HIGH | VCMP > VREF if polarity is not inverted, VCMP < VREF if inverted. |
| COMPARATOR_STATE_OUTPUT_DISABLED | comparator_api_t::outputEnable() has not been called |

## 4.3.7 CRC Interface

Interfaces

### Detailed Description

Interface for cyclic redundancy checking.

# 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.

Implemented by:

- Cyclic Redundancy Check (CRC) Calculator (r_crc)

### Data Structures

| | |
|---|---|
| struct | crc_input_t |
| struct | crc_cfg_t |
| struct | crc_api_t |
| struct | crc_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | crc_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | crc_polynomial_t |
| enum | crc_bit_order_t |
| enum | crc_snoop_direction_t |
| enum | crc_snoop_address_t |

### Data Structure Documentation

#### ◆ crc_input_t

| |
|---|
| struct crc_input_t |
| Structure for CRC inputs |

### ◆ crc_cfg_t

| struct crc_cfg_t | | |
|---|---|---|
| User configuration structure, used in open function | | |
| Data Fields | | |
| crc_polynomial_t | polynomial | CRC Generating Polynomial Switching (GPS) |
| crc_bit_order_t | bit_order | CRC Calculation Switching (LMS) |
| crc_snoop_address_t | snoop_address | Register Snoop Address (CRCSA) |
| void const * | p_extend | CRC Hardware Dependent Configuration. |

### ◆ crc_api_t

| struct crc_api_t | |
|---|---|
| CRC driver structure. General CRC functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(crc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | crcResultGet )(crc_ctrl_t *const p_ctrl, uint32_t *crc_result) |
| | |
| fsp_err_t(* | snoopEnable )(crc_ctrl_t *const p_ctrl, uint32_t crc_seed) |
| | |
| fsp_err_t(* | snoopDisable )(crc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | calculate )(crc_ctrl_t *const p_ctrl, crc_input_t *const p_crc_input, uint32_t *p_crc_result) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* crc_api_t::open) (crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)

Open the CRC driver module.

**Implemented as**

- R_CRC_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CRC device handle. |
| [in] | p_cfg | Pointer to a configuration structure. |

### ◆ close

fsp_err_t(* crc_api_t::close) (crc_ctrl_t *const p_ctrl)

Close the CRC module driver

**Implemented as**

- R_CRC_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to crc device handle |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration was successful. |

### ◆ crcResultGet

fsp_err_t(* crc_api_t::crcResultGet) (crc_ctrl_t *const p_ctrl, uint32_t *crc_result)

Return the current calculated value.

**Implemented as**

- R_CRC_CalculatedValueGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CRC device handle. |
| [out] | crc_result | The calculated value from the last CRC calculation. |

### ◆ snoopEnable

fsp_err_t(* crc_api_t::snoopEnable) (crc_ctrl_t *const p_ctrl, uint32_t crc_seed)

Configure and Enable snooping.

**Implemented as**

- R_CRC_SnoopEnable()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CRC device handle. |
| [in] | crc_seed | CRC seed. |

### ◆ snoopDisable

fsp_err_t(* crc_api_t::snoopDisable) (crc_ctrl_t *const p_ctrl)

Disable snooping.

**Implemented as**

- R_CRC_SnoopDisable()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CRC device handle. |

### ◆ calculate

fsp_err_t(* crc_api_t::calculate) (crc_ctrl_t *const p_ctrl, crc_input_t *const p_crc_input, uint32_t *p_crc_result)

Perform a CRC calculation on a block of data.

**Implemented as**

- R_CRC_Calculate()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to crc device handle. |
| [in] | p_crc_input | A pointer to structure for CRC inputs |
| [out] | crc_result | The calculated value of the CRC calculation. |

### ◆ versionGet

| fsp_err_t(* crc_api_t::versionGet) (fsp_version_t *version) |
|---|
| Get the driver version based on compile time macros.<br><br>**Implemented as**<br>◦ R_CRC_VersionGet() |

### ◆ crc_instance_t

| struct crc_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| 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. |

## Typedef Documentation

### ◆ crc_ctrl_t

| typedef void crc_ctrl_t |
|---|
| CRC control block. Allocate an instance specific control block to pass into the CRC API calls.<br><br>**Implemented as**<br>◦ crc_instance_ctrl_t |

## Enumeration Type Documentation

### ◆ crc_polynomial_t

| enum crc_polynomial_t | |
|---|---|
| CRC Generating Polynomial Switching (GPS). | |
| Enumerator | |
| CRC_POLYNOMIAL_CRC_8 | 8-bit CRC-8 (X^8 + X^2 + X + 1) |
| CRC_POLYNOMIAL_CRC_16 | 16-bit CRC-16 (X^16 + X^15 + X^2 + 1) |
| CRC_POLYNOMIAL_CRC_CCITT | 16-bit CRC-CCITT (X^16 + X^12 + X^5 + 1) |
| CRC_POLYNOMIAL_CRC_32 | 32-bit CRC-32 (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_POLYNOMIAL_CRC_32C | 32-bit CRC-32C (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_bit_order_t

| enum crc_bit_order_t | |
|---|---|
| CRC Calculation Switching (LMS) | |
| Enumerator | |
| CRC_BIT_ORDER_LMS_LSB | Generates CRC for LSB first communication. |
| CRC_BIT_ORDER_LMS_MSB | Generates CRC for MSB first communication. |

### ◆ crc_snoop_direction_t

| enum crc_snoop_direction_t | |
|---|---|
| Snoop-On-Write/Read Switch (CRCSWR) | |
| Enumerator | |
| CRC_SNOOP_DIRECTION_RECEIVE | Snoop-on-read. |
| CRC_SNOOP_DIRECTION_TRANSMIT | Snoop-on-write. |

### ◆ crc_snoop_address_t

| enum crc_snoop_address_t | |
|---|---|
| Snoop SCI register Address (lower 14 bits) | |
| Enumerator | |
| CRC_SNOOP_ADDRESS_NONE | Snoop mode disabled. |
| CRC_SNOOP_ADDRESS_SCI0_TDR | Snoop SCI0 transmit data register. |
| CRC_SNOOP_ADDRESS_SCI1_TDR | Snoop SCI1 transmit data register. |
| CRC_SNOOP_ADDRESS_SCI2_TDR | Snoop SCI2 transmit data register. |
| CRC_SNOOP_ADDRESS_SCI3_TDR | Snoop SCI3 transmit data register. |
| CRC_SNOOP_ADDRESS_SCI4_TDR | Snoop SCI4 transmit data register. |
| CRC_SNOOP_ADDRESS_SCI5_TDR | Snoop SCI5 transmit data register. |
| CRC_SNOOP_ADDRESS_SCI6_TDR | Snoop SCI6 transmit data register. |
| CRC_SNOOP_ADDRESS_SCI7_TDR | Snoop SCI7 transmit data register. |
| CRC_SNOOP_ADDRESS_SCI8_TDR | Snoop SCI8 transmit data register. |
| CRC_SNOOP_ADDRESS_SCI9_TDR | Snoop SCI9 transmit data register. |
| CRC_SNOOP_ADDRESS_SCI0_FTDRL | Snoop SCI0 transmit FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI1_FTDRL | Snoop SCI1 transmit FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI2_FTDRL | Snoop SCI2 transmit FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI3_FTDRL | Snoop SCI3 transmit FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI4_FTDRL | Snoop SCI4 transmit FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI5_FTDRL | Snoop SCI5 transmit FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI6_FTDRL | Snoop SCI6 transmit FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI7_FTDRL | Snoop SCI7 transmit FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI8_FTDRL | Snoop SCI8 transmit FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI9_FTDRL | Snoop SCI9 transmit FIFO data register. |

| | |
|---|---|
| CRC_SNOOP_ADDRESS_SCI0_RDR | Snoop SCI0 receive data register. |
| CRC_SNOOP_ADDRESS_SCI1_RDR | Snoop SCI1 receive data register. |
| CRC_SNOOP_ADDRESS_SCI2_RDR | Snoop SCI2 receive data register. |
| CRC_SNOOP_ADDRESS_SCI3_RDR | Snoop SCI3 receive data register. |
| CRC_SNOOP_ADDRESS_SCI4_RDR | Snoop SCI4 receive data register. |
| CRC_SNOOP_ADDRESS_SCI5_RDR | Snoop SCI5 receive data register. |
| CRC_SNOOP_ADDRESS_SCI6_RDR | Snoop SCI6 receive data register. |
| CRC_SNOOP_ADDRESS_SCI7_RDR | Snoop SCI7 receive data register. |
| CRC_SNOOP_ADDRESS_SCI8_RDR | Snoop SCI8 receive data register. |
| CRC_SNOOP_ADDRESS_SCI9_RDR | Snoop SCI9 receive data register. |
| CRC_SNOOP_ADDRESS_SCI0_FRDRL | Snoop SCI0 receive FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI1_FRDRL | Snoop SCI1 receive FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI2_FRDRL | Snoop SCI2 receive FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI3_FRDRL | Snoop SCI3 receive FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI4_FRDRL | Snoop SCI4 receive FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI5_FRDRL | Snoop SCI5 receive FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI6_FRDRL | Snoop SCI6 receive FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI7_FRDRL | Snoop SCI7 receive FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI8_FRDRL | Snoop SCI8 receive FIFO data register. |
| CRC_SNOOP_ADDRESS_SCI9_FRDRL | Snoop SCI9 receive FIFO data register. |

### 4.3.8 CTSU Interface

Interfaces

### Detailed Description

Interface for Capacitive Touch Sensing Unit (CTSU) functions.

# Summary

The CTSU interface provides CTSU functionality.

The CTSU interface can be implemented by:

- Capacitive Touch Sensing Unit (r_ctsu)

### Data Structures

| | |
|---|---|
| struct | ctsu_callback_args_t |
| struct | ctsu_element_cfg_t |
| struct | ctsu_cfg_t |
| struct | ctsu_api_t |
| struct | ctsu_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | ctsu_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | ctsu_event_t |
| enum | ctsu_cap_t |
| enum | ctsu_txvsel_t |
| enum | ctsu_txvsel2_t |
| enum | ctsu_atune1_t |
| enum | ctsu_atune12_t |
| enum | ctsu_md_t |
| enum | ctsu_posel_t |
| enum | ctsu_ssdiv_t |

### Data Structure Documentation

#### ◆ ctsu_callback_args_t

| struct ctsu_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| ctsu_event_t | event | The event can be used to identify what caused the callback. |
| void const * | p_context | Placeholder for user data. Set in ctsu_api_t::open function in ctsu_cfg_t. |

### ◆ ctsu_element_cfg_t

| struct ctsu_element_cfg_t | | |
|---|---|---|
| CTSU Configuration parameters. Element Configuration | | |
| Data Fields | | |
| ctsu_ssdiv_t | ssdiv | CTSU Spectrum Diffusion Frequency Division Setting (CTSU Only) |
| uint16_t | so | CTSU Sensor Offset Adjustment. |
| uint8_t | snum | CTSU Measurement Count Setting. |
| uint8_t | sdpa | CTSU Base Clock Setting. |

### ◆ ctsu_cfg_t

| struct ctsu_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| ctsu_cap_t | cap |
| | CTSU Scan Start Trigger Select. |
| | |
| ctsu_txvsel_t | txvsel |
| | CTSU Transmission Power Supply Select. |
| | |
| ctsu_txvsel2_t | txvsel2 |
| | CTSU Transmission Power Supply Select 2 (CTSU2 Only) |
| | |
| ctsu_atune1_t | atune1 |

| | |
|---|---|
| | CTSU Power Supply Capacity Adjustment (CTSU Only) |
| | |
| ctsu_atune12_t | atune12 |
| | CTSU Power Supply Capacity Adjustment (CTSU2 Only) |
| | |
| ctsu_md_t | md |
| | CTSU Measurement Mode Select. |
| | |
| ctsu_posel_t | posel |
| | CTSU Non-Measured Channel Output Select (CTSU2 Only) |
| | |
| uint8_t | ctsuchac0 |
| | TS00-TS07 enable mask. |
| | |
| uint8_t | ctsuchac1 |
| | TS08-TS15 enable mask. |
| | |
| uint8_t | ctsuchac2 |
| | TS16-TS23 enable mask. |
| | |
| uint8_t | ctsuchac3 |
| | TS24-TS31 enable mask. |
| | |
| uint8_t | ctsuchac4 |
| | TS32-TS39 enable mask. |
| | |
| uint8_t | ctsuchtrc0 |
| | TS00-TS07 mutual-tx mask. |

| | |
|---|---|
| | |
| uint8_t | ctsuchtrc1 |
| | TS08-TS15 mutual-tx mask. |
| | |
| uint8_t | ctsuchtrc2 |
| | TS16-TS23 mutual-tx mask. |
| | |
| uint8_t | ctsuchtrc3 |
| | TS24-TS31 mutual-tx mask. |
| | |
| uint8_t | ctsuchtrc4 |
| | TS32-TS39 mutual-tx mask. |
| | |
| ctsu_element_cfg_t const * | p_elements |
| | Pointer to elements configuration array. |
| | |
| uint8_t | num_rx |
| | Number of receive terminals. |
| | |
| uint8_t | num_tx |
| | Number of transmit terminals. |
| | |
| uint16_t | num_moving_average |
| | Number of moving average for measurement data. |
| | |
| bool | tunning_enable |
| | Initial offset tuning flag. |
| | |

| | |
|---|---|
| void(* | p_callback )(ctsu_callback_args_t *p_args) |
| | Callback provided when CTSUFN ISR occurs. |
| | |
| transfer_instance_t const * | p_transfer_tx |
| | DTC instance for transmit at CTSUWR. Set to NULL if unused. |
| | |
| transfer_instance_t const * | p_transfer_rx |
| | DTC instance for receive at CTSURD. Set to NULL if unused. |
| | |
| adc_instance_t const * | p_adc_instance |
| | ADC instance for temperature correction. |
| | |
| IRQn_Type | write_irq |
| | CTSU_CTSUWR interrupt vector. |
| | |
| IRQn_Type | read_irq |
| | CTSU_CTSURD interrupt vector. |
| | |
| IRQn_Type | end_irq |
| | CTSU_CTSUFN interrupt vector. |
| | |
| void const * | p_context |
| | User defined context passed into callback function. |
| | |
| void const * | p_extend |
| | Pointer to extended configuration by instance of interface. |
| | |

### ◆ ctsu_api_t

struct ctsu_api_t

Functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(ctsu_ctrl_t *const p_ctrl, ctsu_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | scanStart )(ctsu_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | dataGet )(ctsu_ctrl_t *const p_ctrl, uint16_t *p_data) |
| | |
| fsp_err_t(* | close )(ctsu_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_data) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* ctsu_api_t::open) (ctsu_ctrl_t *const p_ctrl, ctsu_cfg_t const *const p_cfg)

Open driver.

**Implemented as**

- R_CTSU_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ scanStart

fsp_err_t(* ctsu_api_t::scanStart) (ctsu_ctrl_t *const p_ctrl)

Scan start.

**Implemented as**

- R_CTSU_ScanStart()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ dataGet

fsp_err_t(* ctsu_api_t::dataGet) (ctsu_ctrl_t *const p_ctrl, uint16_t *p_data)

Data get.

**Implemented as**

- R_CTSU_DataGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_data | Pointer to get data array. |

### ◆ close

fsp_err_t(* ctsu_api_t::close) (ctsu_ctrl_t *const p_ctrl)

Close driver.

**Implemented as**

- R_CTSU_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ versionGet

fsp_err_t(* ctsu_api_t::versionGet) (fsp_version_t *const p_data)

Return the version of the driver.

**Implemented as**

- R_CTSU_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_data | Memory address to return version information to. |

### ◆ ctsu_instance_t

struct ctsu_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| ctsu_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| ctsu_cfg_t const * | p_cfg | Pointer to the configuration |

| | | |
|---|---|---|
| | | structure for this instance. |
| ctsu_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ ctsu_ctrl_t

| typedef void ctsu_ctrl_t |
|---|
| CTSU Control block. Allocate an instance specific control block to pass into the API calls.<br><br>**Implemented as**<br><br>◦ ctsu_instance_ctrl_t |

## Enumeration Type Documentation

### ◆ ctsu_event_t

| enum ctsu_event_t | |
|---|---|
| CTSU Events for callback function | |
| Enumerator | |
| CTSU_EVENT_SCAN_COMPLETE | Normal end. |
| CTSU_EVENT_OVERFLOW | Sensor counter overflow (CTSUST.CTSUSOVF set) |
| CTSU_EVENT_ICOMP | Abnormal TSCAP voltage (CTSUERRS.CTSUICOMP set) |
| CTSU_EVENT_ICOMP1 | Abnormal sensor current (CTSUSR.ICOMP1 set) |

### ◆ ctsu_cap_t

| enum ctsu_cap_t | |
|---|---|
| CTSU Scan Start Trigger Select | |
| Enumerator | |
| CTSU_CAP_SOFTWARE | Scan start by software trigger. |
| CTSU_CAP_EXTERNAL | Scan start by external trigger. |

### ◆ ctsu_txvsel_t

| enum ctsu_txvsel_t | |
|---|---|
| CTSU Transmission Power Supply Select | |
| Enumerator | |
| CTSU_TXVSEL_VCC | VCC selected. |
| CTSU_TXVSEL_INTERNAL_POWER | Internal logic power supply selected. |

### ◆ ctsu_txvsel2_t

| enum ctsu_txvsel2_t | |
|---|---|
| CTSU Transmission Power Supply Select 2 (CTSU2 Only) | |
| Enumerator | |
| CTSU_TXVSEL_MODE | Follow TXVSEL setting. |
| CTSU_TXVSEL_VCC_PRIVATE | VCC private selected. |

### ◆ ctsu_atune1_t

| enum ctsu_atune1_t | |
|---|---|
| CTSU Power Supply Capacity Adjustment (CTSU Only) | |
| Enumerator | |
| CTSU_ATUNE1_NORMAL | Normal output (40uA) |
| CTSU_ATUNE1_HIGH | High-current output (80uA) |

### ◆ ctsu_atune12_t

| enum ctsu_atune12_t | |
|---|---|
| CTSU Power Supply Capacity Adjustment (CTSU2 Only) | |
| Enumerator | |
| CTSU_ATUNE12_80UA | High-current output (80uA) |
| CTSU_ATUNE12_40UA | Normal output (40uA) |
| CTSU_ATUNE12_20UA | Low-current output (20uA) |
| CTSU_ATUNE12_160UA | Very high-current output (160uA) |

### ◆ ctsu_md_t

| enum ctsu_md_t | |
|---|---|
| CTSU Measurement Mode Select | |
| Enumerator | |
| CTSU_MODE_SELF_MULTI_SCAN | Self-capacitance multi scan mode. |
| CTSU_MODE_MUTUAL_FULL_SCAN | Mutual capacitance full scan mode. |
| CTSU_MODE_MUTUAL_CFC_SCAN | Mutual capacitance cfc scan mode (CTSU2 Only) |
| CTSU_MODE_CURRENT_SCAN | Current scan mode (CTSU2 Only) |
| CTSU_MODE_CORRECTION_SCAN | Correction scan mode (CTSU2 Only) |

### ◆ ctsu_posel_t

| enum ctsu_posel_t | |
|---|---|
| CTSU Non-Measured Channel Output Select (CTSU2 Only) | |
| Enumerator | |
| CTSU_POSEL_LOW_GPIO | Output low through GPIO. |
| CTSU_POSEL_HI_Z | Hi-Z. |
| CTSU_POSEL_LOW | Output low through the power setting by the TXVSEL[1:0] bits. |
| CTSU_POSEL_SAME_PULSE | Same phase pulse output as transmission channel through the power setting by the TXVSEL[1:0] bits. |

### ◆ ctsu_ssdiv_t

| enum ctsu_ssdiv_t | |
|---|---|
| CTSU Spectrum Diffusion Frequency Division Setting (CTSU Only) | |
| Enumerator | |
| CTSU_SSDIV_4000 | 4.00 <= Base clock frequency (MHz) |
| CTSU_SSDIV_2000 | 2.00 <= Base clock frequency (MHz) < 4.00 |
| CTSU_SSDIV_1330 | 1.33 <= Base clock frequency (MHz) < 2.00 |
| CTSU_SSDIV_1000 | 1.00 <= Base clock frequency (MHz) < 1.33 |
| CTSU_SSDIV_0800 | 0.80 <= Base clock frequency (MHz) < 1.00 |
| CTSU_SSDIV_0670 | 0.67 <= Base clock frequency (MHz) < 0.80 |
| CTSU_SSDIV_0570 | 0.57 <= Base clock frequency (MHz) < 0.67 |
| CTSU_SSDIV_0500 | 0.50 <= Base clock frequency (MHz) < 0.57 |
| CTSU_SSDIV_0440 | 0.44 <= Base clock frequency (MHz) < 0.50 |
| CTSU_SSDIV_0400 | 0.40 <= Base clock frequency (MHz) < 0.44 |
| CTSU_SSDIV_0360 | 0.36 <= Base clock frequency (MHz) < 0.40 |
| CTSU_SSDIV_0330 | 0.33 <= Base clock frequency (MHz) < 0.36 |
| CTSU_SSDIV_0310 | 0.31 <= Base clock frequency (MHz) < 0.33 |
| CTSU_SSDIV_0290 | 0.29 <= Base clock frequency (MHz) < 0.31 |
| CTSU_SSDIV_0270 | 0.27 <= Base clock frequency (MHz) < 0.29 |
| CTSU_SSDIV_0000 | 0.00 <= Base clock frequency (MHz) < 0.27 |

## 4.3.9 DAC Interface

Interfaces

### Detailed Description

Interface for D/A converters.

# 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.

Implemented by:

- Digital to Analog Converter (r_dac)
- Digital to Analog Converter (r_dac8)

### Data Structures

| | |
|---|---|
| struct | dac_info_t |
| struct | dac_cfg_t |
| struct | dac_api_t |
| struct | dac_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | dac_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | dac_data_format_t |

## Data Structure Documentation

### ◆ dac_info_t

| struct dac_info_t | | |
|---|---|---|
| DAC information structure to store various information for a DAC | | |
| Data Fields | | |
| uint8_t | bit_width | Resolution of the DAC. |

### ◆ dac_cfg_t

| struct dac_cfg_t | | |
|---|---|---|
| DAC Open API configuration parameter | | |
| Data Fields | | |
| uint8_t | channel | ID associated with this DAC channel. |
| bool | ad_da_synchronized | AD/DA synchronization. |
| void const * | p_extend | |

### ◆ dac_api_t

struct dac_api_t

DAC driver structure. General DAC functions implemented at the HAL layer follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(dac_ctrl_t *const p_ctrl, dac_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(dac_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | write )(dac_ctrl_t *const p_ctrl, uint16_t value) |
| | |
| fsp_err_t(* | start )(dac_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stop )(dac_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* dac_api_t::open) (dac_ctrl_t *const p_ctrl, dac_cfg_t const *const p_cfg)

Initial configuration.

**Implemented as**

- R_DAC_Open()
- R_DAC8_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ close

fsp_err_t(* dac_api_t::close) (dac_ctrl_t *const p_ctrl)

Close the D/A Converter.

**Implemented as**

- R_DAC_Close()
- R_DAC8_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in dac_api_t::open call for this timer. |

### ◆ write

fsp_err_t(* dac_api_t::write) (dac_ctrl_t *const p_ctrl, uint16_t value)

Write sample value to the D/A Converter.

**Implemented as**

- R_DAC_Write()
- R_DAC8_Write()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in dac_api_t::open call for this timer. |
| [in] | value | Sample value to be written to the D/A Converter. |

### ◆ start

fsp_err_t(* dac_api_t::start) (dac_ctrl_t *const p_ctrl)

Start the D/A Converter if it has not been started yet.

**Implemented as**

- R_DAC_Start()
- R_DAC8_Start()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in dac_api_t::open call for this timer. |

### ◆ stop

| fsp_err_t(* dac_api_t::stop) (dac_ctrl_t *const p_ctrl) |
|---|

Stop the D/A Converter if the converter is running.

**Implemented as**

- R_DAC_Stop()
- R_DAC8_Stop()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in dac_api_t::open call for this timer. |

### ◆ versionGet

| fsp_err_t(* dac_api_t::versionGet) (fsp_version_t *p_version) |
|---|

Get version and store it in provided pointer p_version.

**Implemented as**

- R_DAC_VersionGet()
- R_DAC8_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version used. |

### ◆ dac_instance_t

| struct dac_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| 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. |

## Typedef Documentation

### ◆ dac_ctrl_t

| typedef void dac_ctrl_t |
|---|
| DAC control block. Allocate an instance specific control block to pass into the DAC API calls.<br><br>**Implemented as**<br><br>◦ dac_instance_ctrl_t<br>◦ dac8_instance_ctrl_t |

#### Enumeration Type Documentation

### ◆ dac_data_format_t

| enum dac_data_format_t | |
|---|---|
| DAC Open API data format settings. | |
| Enumerator | |
| 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. |

## 4.3.10 Display Interface

Interfaces

#### Detailed Description

Interface for LCD panel displays.

# 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: Graphics LCD Controller (r_glcdc)

#### Data Structures

| | |
|---|---|
| struct | display_timing_t |
| struct | display_color_t |
| struct | display_coordinate_t |
| struct | display_brightness_t |
| struct | display_contrast_t |
| struct | display_correction_t |
| struct | gamma_correction_t |
| struct | display_gamma_correction_t |
| struct | display_clut_t |
| struct | display_input_cfg_t |
| struct | display_output_cfg_t |
| struct | display_layer_t |
| struct | display_callback_args_t |
| struct | display_cfg_t |
| struct | display_runtime_cfg_t |
| struct | display_clut_cfg_t |
| struct | display_status_t |
| struct | display_api_t |
| struct | display_instance_t |

## Typedefs

| | |
|---|---|
| typedef void | display_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | display_frame_layer_t |
| enum | display_state_t |
| enum | display_event_t |

| | |
|---|---|
| enum | display_in_format_t |
| enum | display_out_format_t |
| enum | display_endian_t |
| enum | display_color_order_t |
| enum | display_signal_polarity_t |
| enum | display_sync_edge_t |
| enum | display_fade_control_t |
| enum | display_fade_status_t |

## Data Structure Documentation

### ◆ display_timing_t

| struct display_timing_t | | |
|---|---|---|
| Display signal timing setting | | |
| Data Fields | | |
| 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 porch cycles or lines. |
| uint16_t | sync_width | Sync signal asserting width. |
| display_signal_polarity_t | sync_polarity | Sync signal polarity. |

### ◆ display_color_t

| struct display_color_t |
|---|
| RGB Color setting |

### ◆ display_coordinate_t

| struct display_coordinate_t | | |
|---|---|---|
| Contrast (gain) correction setting | | |
| Data Fields | | |
| int16_t | x | Coordinate X, this allows to set signed value. |
| int16_t | y | Coordinate Y, this allows to set signed value. |

### ◆ display_brightness_t

| struct display_brightness_t | | |
|---|---|---|
| Brightness (DC) correction setting | | |
| Data Fields | | |
| 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. |

### ◆ display_contrast_t

| struct display_contrast_t | | |
|---|---|---|
| Contrast (gain) correction setting | | |
| Data Fields | | |
| 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. |

### ◆ display_correction_t

| struct display_correction_t | | |
|---|---|---|
| Color correction setting | | |
| Data Fields | | |
| display_brightness_t | brightness | Brightness. |
| display_contrast_t | contrast | Contrast. |

### ◆ gamma_correction_t

| struct gamma_correction_t | | |
|---|---|---|
| Gamma correction setting for each color | | |
| Data Fields | | |
| bool | enable | Gamma Correction On/Off. |
| uint16_t * | gain | Gain adjustment. |
| uint16_t * | threshold | Start threshold. |

### ◆ display_gamma_correction_t

| struct display_gamma_correction_t | | |
|---|---|---|
| Gamma correction setting | | |
| Data Fields | | |
| 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. |

### ◆ display_clut_t

| struct display_clut_t | | |
|---|---|---|
| CLUT setting | | |
| Data Fields | | |
| 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) |

### ◆ display_input_cfg_t

| struct display_input_cfg_t | | |
|---|---|---|
| Graphics plane input configuration structure | | |
| Data Fields | | |
| 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. |

### ◆ display_output_cfg_t

| struct display_output_cfg_t |
|---|
| Display output configuration structure |
| Data Fields |

| | | |
|---|---|---|
| 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. |

### ◆ display_layer_t

| struct display_layer_t | | |
|---|---|---|
| Graphics layer blend setup parameter structure | | |
| Data Fields | | |
| 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. |

### ◆ display_callback_args_t

| struct display_callback_args_t | | |
|---|---|---|
| Display callback parameter definition | | |
| Data Fields | | |
| display_event_t | event | Event code. |
| void const * | p_context | Context provided to user during callback. |

### ◆ display_cfg_t

| struct display_cfg_t | |
|---|---|
| Display main configuration structure | |
| **Data Fields** | |
| display_input_cfg_t | input [2] |

| | |
|---|---|
| | Graphics input frame setting. More... |
| | |
| display_output_cfg_t | output |
| | Graphics output frame setting. |
| | |
| display_layer_t | layer [2] |
| | 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 2 interrupt priority. |
| | |
| IRQn_Type | line_detect_irq |
| | Line detect interrupt vector. |
| | |
| IRQn_Type | underflow_1_irq |
| | Underflow 1 interrupt vector. |
| | |
| IRQn_Type | underflow_2_irq |
| | Underflow 2 interrupt vector. |
| | |
| void(* | p_callback )(display_callback_args_t *p_args) |
| | Pointer to callback function. More... |

| | |
|---|---|
| void const * | p_context |
| | User defined context passed into callback function. |
| void const * | p_extend |
| | Display hardware dependent configuration. More... |

## Field Documentation

### ◆ input

display_input_cfg_t display_cfg_t::input[2]

Graphics input frame setting.

Generic configuration for display devices

### ◆ p_callback

void(* display_cfg_t::p_callback) (display_callback_args_t *p_args)

Pointer to callback function.

Configuration for display event processing

### ◆ p_extend

void const* display_cfg_t::p_extend

Display hardware dependent configuration.

Pointer to display peripheral specific configuration

### ◆ display_runtime_cfg_t

struct display_runtime_cfg_t

Display main configuration structure

| Data Fields | | |
|---|---|---|
| display_input_cfg_t | input | Graphics input frame setting.<br><br>Generic configuration for display devices |
| display_layer_t | layer | Graphics layer alpha blending setting. |

### ◆ display_clut_cfg_t

struct display_clut_cfg_t

| Display CLUT configuration structure | | |
|---|---|---|
| Data Fields | | |
| 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. |

### ◆ display_status_t

| struct display_status_t | | |
|---|---|---|
| Display Status | | |
| Data Fields | | |
| display_state_t | state | Status of GLCDC module. |
| display_fade_status_t | fade_status[ DISPLAY_FRAME_LAYER_2+1] | Status of fade-in/fade-out status. |

### ◆ display_api_t

| struct display_api_t | |
|---|---|
| Shared Interface definition for display peripheral | |
| **Data Fields** | |
| fsp_err_t(* | open )(display_ctrl_t *const p_ctrl, display_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(display_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | start )(display_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stop )(display_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | layerChange )(display_ctrl_t const *const p_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t frame) |
| | |
| fsp_err_t(* | bufferChange )(display_ctrl_t const *const p_ctrl, uint8_t *const framebuffer, display_frame_layer_t frame) |
| | |
| fsp_err_t(* | correction )(display_ctrl_t const *const p_ctrl, display_correction_t const *const p_param) |
| | |

| | |
|---|---|
| fsp_err_t(* | clut )(display_ctrl_t const *const p_ctrl, display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t layer) |
| | |
| fsp_err_t(* | clutEdit )(display_ctrl_t const *const p_ctrl, display_frame_layer_t layer, uint8_t index, uint32_t color) |
| | |
| fsp_err_t(* | statusGet )(display_ctrl_t const *const p_ctrl, display_status_t *const p_status) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* display_api_t::open) (display_ctrl_t *const p_ctrl, display_cfg_t const *const p_cfg)

Open display device.

**Implemented as**

- R_GLCDC_Open()

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to display interface control block. Must be declared by user. Value set here. |
| [in] | p_cfg | Pointer to display configuration structure. All elements of this structure must be set by user. |

### ◆ close

fsp_err_t(* display_api_t::close) (display_ctrl_t *const p_ctrl)

Close display device.

**Implemented as**

- R_GLCDC_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |

### ◆ start

fsp_err_t(* display_api_t::start) (display_ctrl_t *const p_ctrl)

Display start.

**Implemented as**

- R_GLCDC_Start()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |

### ◆ stop

fsp_err_t(* display_api_t::stop) (display_ctrl_t *const p_ctrl)

Display stop.

**Implemented as**

- R_GLCDC_Stop()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |

### ◆ layerChange

fsp_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)

Change layer parameters at runtime.

**Implemented as**

- R_GLCDC_LayerChange()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | p_cfg | Pointer to run-time layer configuration structure. |
| [in] | frame | Number of graphic frames. |

### ◆ bufferChange

fsp_err_t(* display_api_t::bufferChange) (display_ctrl_t const *const p_ctrl, uint8_t *const framebuffer, display_frame_layer_t frame)

Change layer framebuffer pointer.

**Implemented as**

- R_GLCDC_BufferChange()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | framebuffer | Pointer to desired framebuffer. |
| [in] | frame | Number of graphic frames. |

### ◆ correction

fsp_err_t(* display_api_t::correction) (display_ctrl_t const *const p_ctrl, display_correction_t const *const p_param)

Color correction.

**Implemented as**

- R_GLCDC_ColorCorrection()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | param | Pointer to color correction configuration structure. |

### ◆ clut

fsp_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 layer)

Set CLUT for display device.

**Implemented as**

- R_GLCDC_ClutUpdate()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | p_clut_cfg | Pointer to CLUT configuration structure. |
| [in] | layer | Layer number corresponding to the CLUT. |

### ◆ clutEdit

fsp_err_t(* display_api_t::clutEdit) (display_ctrl_t const *const p_ctrl, display_frame_layer_t layer, uint8_t index, uint32_t color)

Set CLUT element for display device.

**Implemented as**

- R_GLCDC_ClutEdit()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | layer | Layer number corresponding to the CLUT. |
| [in] | index | CLUT element index. |
| [in] | color | Desired CLUT index color. |

### ◆ statusGet

fsp_err_t(* display_api_t::statusGet) (display_ctrl_t const *const p_ctrl, display_status_t *const p_status)

Get status for display device.

**Implemented as**

- R_GLCDC_StatusGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | status | Pointer to display interface status structure. |

### ◆ versionGet

fsp_err_t(* display_api_t::versionGet) (fsp_version_t *p_version)

Get version.

**Implemented as**

- R_GLCDC_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_version | Pointer to the memory to store the version information. |

### ◆ display_instance_t

struct display_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| 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. |

## Typedef Documentation

### ◆ display_ctrl_t

| typedef void display_ctrl_t |
|---|
| Display control block. Allocate an instance specific control block to pass into the display API calls.<br><br>**Implemented as**<br><br>◦ glcdc_instance_ctrl_tDisplay control block |

## Enumeration Type Documentation

### ◆ display_frame_layer_t

| enum display_frame_layer_t | |
|---|---|
| Display frame number | |
| Enumerator | |
| DISPLAY_FRAME_LAYER_1 | Frame layer 1. |
| DISPLAY_FRAME_LAYER_2 | Frame layer 2. |

### ◆ display_state_t

| enum display_state_t | |
|---|---|
| Display interface operation state | |
| Enumerator | |
| DISPLAY_STATE_CLOSED | Display closed. |
| DISPLAY_STATE_OPENED | Display opened. |
| DISPLAY_STATE_DISPLAYING | Displaying. |

### ◆ display_event_t

| enum display_event_t | |
|---|---|
| Display event codes | |
| Enumerator | |
| DISPLAY_EVENT_GR1_UNDERFLOW | Graphics frame1 underflow occurs. |
| DISPLAY_EVENT_GR2_UNDERFLOW | Graphics frame2 underflow occurs. |
| DISPLAY_EVENT_LINE_DETECTION | Designated line is processed. |

### ◆ display_in_format_t

| enum display_in_format_t | |
|---|---|
| Input format setting | |
| Enumerator | |
| 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. |

### ◆ display_out_format_t

| enum display_out_format_t | |
|---|---|
| Output format setting | |
| Enumerator | |
| 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. |

### ◆ display_endian_t

| enum display_endian_t | |
|---|---|
| Data endian select | |
| Enumerator | |
| DISPLAY_ENDIAN_LITTLE | Little-endian. |
| DISPLAY_ENDIAN_BIG | Big-endian. |

### ◆ display_color_order_t

| enum display_color_order_t | |
|---|---|
| RGB color order select | |
| Enumerator | |
| DISPLAY_COLOR_ORDER_RGB | Color order RGB. |
| DISPLAY_COLOR_ORDER_BGR | Color order BGR. |

### ◆ display_signal_polarity_t

| enum display_signal_polarity_t | |
|---|---|
| Polarity of a signal select | |
| Enumerator | |
| DISPLAY_SIGNAL_POLARITY_LOACTIVE | Low active signal. |
| DISPLAY_SIGNAL_POLARITY_HIACTIVE | High active signal. |

### ◆ display_sync_edge_t

| enum display_sync_edge_t | |
|---|---|
| Signal synchronization edge select | |
| Enumerator | |
| DISPLAY_SIGNAL_SYNC_EDGE_RISING | Signal is synchronized to rising edge. |
| DISPLAY_SIGNAL_SYNC_EDGE_FALLING | Signal is synchronized to falling edge. |

### ◆ display_fade_control_t

| enum display_fade_control_t | |
|---|---|
| Fading control | |
| Enumerator | |
| DISPLAY_FADE_CONTROL_NONE | Applying no fading control. |
| DISPLAY_FADE_CONTROL_FADEIN | Applying fade-in control. |
| DISPLAY_FADE_CONTROL_FADEOUT | Applying fade-out control. |

### ◆ display_fade_status_t

| enum display_fade_status_t | |
|---|---|
| Fading status | |
| Enumerator | |
| 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_PENDING | Fade-in/fade-out is configured but not yet started. |

## 4.3.11 DOC Interface

Interfaces

### Detailed Description

Interface for the Data Operation Circuit.

Defines the API and data structures for the DOC implementation of the Data Operation Circuit (DOC) interface.

# Summary

This module implements the DOC_API using the Data Operation Circuit (DOC).

Implemented by: Data Operation Circuit (r_doc)

### Data Structures

| | |
|---|---|
| struct | doc_status_t |
| struct | doc_callback_args_t |
| struct | doc_cfg_t |
| struct | doc_api_t |
| struct | doc_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | doc_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | doc_event_t |

## Data Structure Documentation

### ◆ doc_status_t

| struct doc_status_t |
|---|
| DOC status |

### ◆ doc_callback_args_t

| struct doc_callback_args_t | | |
|---|---|---|
| Callback function parameter data. | | |
| Data Fields | | |
| void const * | p_context | Set in doc_api_t::open function in doc_cfg_t.<br><br>Placeholder for user data. |

### ◆ doc_cfg_t

| struct doc_cfg_t | |
|---|---|
| User configuration structure, used in the open function. | |
| **Data Fields** | |
| doc_event_t | event |
| | Select enumerated value from doc_event_t. |
| | |
| uint16_t | doc_data |
| | Initial/reference value for DODSR register. |
| | |
| uint8_t | ipl |
| | DOC interrupt priority. |
| | |
| IRQn_Type | irq |
| | NVIC interrupt number assigned to this instance. |
| | |

| | |
|---|---|
| void(* | p_callback )(doc_callback_args_t *p_args) |
| | |
| void const * | p_context |
| | |

## Field Documentation

### ◆ p_callback

void(* doc_cfg_t::p_callback) (doc_callback_args_t *p_args)

Callback provided when a DOC ISR occurs.

### ◆ p_context

void const* doc_cfg_t::p_context

Placeholder for user data. Passed to the user callback in doc_callback_args_t.

### ◆ doc_api_t

struct doc_api_t

Data Operation Circuit (DOC) API structure. DOC functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(doc_ctrl_t *const p_ctrl, doc_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(doc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | statusGet )(doc_ctrl_t *const p_ctrl, doc_status_t *p_status) |
| | |
| fsp_err_t(* | write )(doc_ctrl_t *const p_ctrl, uint16_t data) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* doc_api_t::open) (doc_ctrl_t *const p_ctrl, doc_cfg_t const *const p_cfg)

Initial configuration.

**Implemented as**

- R_DOC_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ close

fsp_err_t(* doc_api_t::close) (doc_ctrl_t *const p_ctrl)

Allow the driver to be reconfigured. Will reduce power consumption.

**Implemented as**

- R_DOC_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in doc_api_t::open call. |

### ◆ statusGet

fsp_err_t(* doc_api_t::statusGet) (doc_ctrl_t *const p_ctrl, doc_status_t *p_status)

Gets the result of addition/subtraction and stores it in the provided pointer p_data.

**Implemented as**

- R_DOC_StatusGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in doc_api_t::open call. |
| [out] | p_data | Provides the 16 bit result of the addition/subtraction operation at the user defined location. |

### ◆ write

fsp_err_t(* doc_api_t::write) (doc_ctrl_t *const p_ctrl, uint16_t data)

Write to the DODIR register.

**Implemented as**

- R_DOC_Write()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in doc_api_t::open call. |
| [in] | data | data to be written to DOC DODIR register. |

### ◆ versionGet

fsp_err_t(* doc_api_t::versionGet) (fsp_version_t *const p_version)

Get version and stores it in provided pointer p_version.

**Implemented as**

- R_DOC_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version used. |

### ◆ doc_instance_t

struct doc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| 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. |

## Typedef Documentation

### ◆ doc_ctrl_t

| typedef void doc_ctrl_t |
|---|
| DOC control block. Allocate an instance specific control block to pass into the DOC API calls.<br><br>**Implemented as**<br><br>◦ doc_instance_ctrl_t |

### Enumeration Type Documentation

### ◆ doc_event_t

| enum doc_event_t | |
|---|---|
| Event that can trigger a callback function. | |
| Enumerator | |
| 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. |

## 4.3.12 ELC Interface

Interfaces

### Detailed Description

Interface for the Event Link Controller.

### Data Structures

| | |
|---|---|
| struct | elc_cfg_t |
| struct | elc_api_t |
| struct | elc_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | elc_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | elc_peripheral_t |
| enum | elc_software_event_t |

## Data Structure Documentation

### ◆ elc_cfg_t

| struct elc_cfg_t | | |
|---|---|---|
| Main configuration structure for the Event Link Controller | | |
| Data Fields | | |
| elc_event_t const | link[ELC_PERIPHERAL_NUM] | Event link register (ELSR) settings. |

### ◆ elc_api_t

| struct elc_api_t | |
|---|---|
| ELC driver structure. General ELC functions implemented at the HAL layer follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(elc_ctrl_t *const p_ctrl, elc_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(elc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | softwareEventGenerate )(elc_ctrl_t *const p_ctrl, elc_software_event_t event_num) |
| | |
| fsp_err_t(* | linkSet )(elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral, elc_event_t signal) |
| | |
| fsp_err_t(* | linkBreak )(elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral) |
| | |
| fsp_err_t(* | enable )(elc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | disable )(elc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* elc_api_t::open) (elc_ctrl_t *const p_ctrl, elc_cfg_t const *const p_cfg)

Initialize all links in the Event Link Controller.

**Implemented as**

- R_ELC_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* elc_api_t::close) (elc_ctrl_t *const p_ctrl)

Disable all links in the Event Link Controller and close the API.

**Implemented as**

- R_ELC_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ softwareEventGenerate

fsp_err_t(* elc_api_t::softwareEventGenerate) (elc_ctrl_t *const p_ctrl, elc_software_event_t event_num)

Generate a software event in the Event Link Controller.

**Implemented as**

- R_ELC_SoftwareEventGenerate()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | eventNum | Software event number to be generated. |

### ◆ linkSet

fsp_err_t(* elc_api_t::linkSet) (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral, elc_event_t signal)

Create a single event link.

**Implemented as**

- R_ELC_LinkSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | peripheral | The peripheral block that will receive the event signal. |
| [in] | signal | The event signal. |

### ◆ linkBreak

fsp_err_t(* elc_api_t::linkBreak) (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral)

Break an event link.

**Implemented as**

- R_ELC_LinkBreak()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | peripheral | The peripheral that should no longer be linked. |

### ◆ enable

fsp_err_t(* elc_api_t::enable) (elc_ctrl_t *const p_ctrl)

Enable the operation of the Event Link Controller.

**Implemented as**

- R_ELC_Enable()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ disable

fsp_err_t(* elc_api_t::disable) (elc_ctrl_t *const p_ctrl)

Disable the operation of the Event Link Controller.

**Implemented as**

- R_ELC_Disable()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ versionGet

fsp_err_t(* elc_api_t::versionGet) (fsp_version_t *const p_version)

Get the driver version based on compile time macros.

**Implemented as**

- R_ELC_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_version | is value returned. |

### ◆ elc_instance_t

struct elc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| elc_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| 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. |

## Typedef Documentation

### ◆ elc_ctrl_t

typedef void elc_ctrl_t

ELC control block. Allocate an instance specific control block to pass into the ELC API calls.

**Implemented as**

- elc_instance_ctrl_t

### Enumeration Type Documentation

#### ◆ elc_peripheral_t

| enum elc_peripheral_t |
|---|
| Possible peripherals to be linked to event signals (not all available on all MCUs) |

#### ◆ elc_software_event_t

| enum elc_software_event_t | |
|---|---|
| Software event number | |
| Enumerator | |
| ELC_SOFTWARE_EVENT_0 | Software event 0. |
| ELC_SOFTWARE_EVENT_1 | Software event 1. |

## 4.3.13 Ethernet Interface

Interfaces

### Detailed Description

Interface for Ethernet functions.

# Summary

The Ethernet interface provides Ethernet functionality. The Ethernet interface supports the following features:

- Transmit/receive processing (Blocking and Non-Blocking)
- Callback function with returned event code
- Magic packet detection mode support
- Auto negotiation support
- Flow control support
- Multicast filtering support

Implemented by:

- Ethernet (r_ether)

### Data Structures

| | |
|---|---|
| struct | ether_instance_descriptor_t |

| | |
|---|---|
| struct | ether_callback_args_t |
| struct | ether_cfg_t |
| struct | ether_api_t |
| struct | ether_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | ether_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | ether_wake_on_lan_t |
| enum | ether_flow_control_t |
| enum | ether_multicast_t |
| enum | ether_promiscuous_t |
| enum | ether_zerocopy_t |
| enum | ether_event_t |

## Data Structure Documentation

### ◆ ether_instance_descriptor_t

| struct ether_instance_descriptor_t |
|---|
| EDMAC descriptor as defined in the hardware manual. Structure must be packed at 1 byte. |

### ◆ ether_callback_args_t

| struct ether_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| uint32_t | channel | Device channel number. |
| ether_event_t | event | Event code. |
| uint32_t | status_ecsr | ETHERC status register for interrupt handler. |
| uint32_t | status_eesr | ETHERC/EDMAC status register for interrupt handler. |
| void const * | p_context | Placeholder for user data. Set in ether_api_t::open function in |

| | | |
|---|---|---|
| | | ether_cfg_t. |

### ◆ ether_cfg_t

| struct ether_cfg_t | |
|---|---|
| Configuration parameters. | |
| **Data Fields** | |
| uint8_t | channel |
| | Channel. |
| | |
| ether_zerocopy_t | zerocopy |
| | Zero copy enable or disable in Read/Write function. |
| | |
| ether_multicast_t | multicast |
| | Multicast enable or disable. |
| | |
| ether_promiscuous_t | promiscuous |
| | Promiscuous mode enable or disable. |
| | |
| ether_flow_control_t | flow_control |
| | Flow control functionally enable or disable. |
| | |
| uint32_t | broadcast_filter |
| | Limit of the number of broadcast frames received continuously. |
| | |
| uint8_t * | p_mac_address |
| | Pointer of MAC address. |
| | |
| ether_instance_descriptor_t * | p_rx_descriptors |
| | Receive descriptor buffer pool. |

| | |
|---|---|
| | |
| ether_instance_descriptor_t * | p_tx_descriptors |
| | Transmit descriptor buffer pool. |
| | |
| uint8_t | num_tx_descriptors |
| | Number of transmission descriptor. |
| | |
| uint8_t | num_rx_descriptors |
| | Number of receive descriptor. |
| | |
| uint8_t ** | pp_ether_buffers |
| | Transmit and receive buffer. |
| | |
| uint32_t | ether_buffer_size |
| | Size of transmit and receive buffer. |
| | |
| IRQn_Type | irq |
| | NVIC interrupt number. |
| | |
| uint32_t | interrupt_priority |
| | NVIC interrupt priority. |
| | |
| void(* | p_callback )(ether_callback_args_t *p_args) |
| | Callback provided when an ISR occurs. |
| | |
| ether_phy_instance_t const * | p_ether_phy_instance |
| | Pointer to ETHER_PHY instance. |

| | |
|---|---|
| void const * | p_context |
| | Placeholder for user data. More... |
| void const * | p_extend |
| | Placeholder for user extension. |

## Field Documentation

### ◆ p_context

void const* ether_cfg_t::p_context

Placeholder for user data.

Placeholder for user data. Passed to the user callback in ether_callback_args_t.

### ◆ ether_api_t

struct ether_api_t

Functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(ether_ctrl_t *const p_api_ctrl, ether_cfg_t const *const p_cfg) |
| fsp_err_t(* | close )(ether_ctrl_t *const p_api_ctrl) |
| fsp_err_t(* | read )(ether_ctrl_t *const p_api_ctrl, void *const p_buffer, uint32_t *const length_bytes) |
| fsp_err_t(* | bufferRelease )(ether_ctrl_t *const p_api_ctrl) |
| fsp_err_t(* | write )(ether_ctrl_t *const p_api_ctrl, void *const p_buffer, uint32_t const frame_length) |
| fsp_err_t(* | linkProcess )(ether_ctrl_t *const p_api_ctrl) |
| fsp_err_t(* | wakeOnLANEnable )(ether_ctrl_t *const p_api_ctrl) |

| fsp_err_t(* | versionGet )(fsp_version_t *const p_data) |
|---|---|

## Field Documentation

### ◆ open

fsp_err_t(* ether_api_t::open) (ether_ctrl_t *const p_api_ctrl, ether_cfg_t const *const p_cfg)

Open driver.

**Implemented as**

- R_ETHER_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ close

fsp_err_t(* ether_api_t::close) (ether_ctrl_t *const p_api_ctrl)

Close driver.

**Implemented as**

- R_ETHER_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |

### ◆ read

fsp_err_t(* ether_api_t::read) (ether_ctrl_t *const p_api_ctrl, void *const p_buffer, uint32_t *const length_bytes)

Read packet if data is available.

**Implemented as**

- R_ETHER_Read()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | p_buffer | Pointer to where to store read data. |
| [in] | length_bytes | Number of bytes in buffer |

### ◆ bufferRelease

fsp_err_t(* ether_api_t::bufferRelease) (ether_ctrl_t *const p_api_ctrl)

Release rx buffer from buffer pool process in zero-copy read operation.

**Implemented as**

- R_ETHER_BufferRelease()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |

### ◆ write

fsp_err_t(* ether_api_t::write) (ether_ctrl_t *const p_api_ctrl, void *const p_buffer, uint32_t const frame_length)

Write packet.

**Implemented as**

- R_ETHER_Write()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | p_buffer | Pointer to data to write. |
| [in] | frame_length | Send ethernet frame size (without 4 bytes of CRC data size). |

### ◆ linkProcess

fsp_err_t(* ether_api_t::linkProcess) (ether_ctrl_t *const p_api_ctrl)

Process link.

**Implemented as**

- R_ETHER_LinkProcess()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |

### ◆ wakeOnLANEnable

fsp_err_t(* ether_api_t::wakeOnLANEnable) (ether_ctrl_t *const p_api_ctrl)

Enable magic packet detection.

**Implemented as**

- R_ETHER_WakeOnLANEnable()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |

### ◆ versionGet

fsp_err_t(* ether_api_t::versionGet) (fsp_version_t *const p_data)

Return the version of the driver.

**Implemented as**

- R_ETHER_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_data | Memory address to return version information to. |

### ◆ ether_instance_t

struct ether_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| ether_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| ether_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| ether_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ ether_ctrl_t

typedef void ether_ctrl_t

Control block. Allocate an instance specific control block to pass into the API calls.

**Implemented as**

- ether_instance_ctrl_t

## Enumeration Type Documentation

### ◆ ether_wake_on_lan_t

| enum ether_wake_on_lan_t | |
|---|---|
| Wake on LAN | |
| Enumerator | |
| ETHER_WAKE_ON_LAN_DISABLE | Disable Wake on LAN. |
| ETHER_WAKE_ON_LAN_ENABLE | Enable Wake on LAN. |

### ◆ ether_flow_control_t

| enum ether_flow_control_t | |
|---|---|
| Flow control functionality | |
| Enumerator | |
| ETHER_FLOW_CONTROL_DISABLE | Disable flow control functionality. |
| ETHER_FLOW_CONTROL_ENABLE | Enable flow control functionality with pause frames. |

### ◆ ether_multicast_t

| enum ether_multicast_t | |
|---|---|
| Multicast Filter | |
| Enumerator | |
| ETHER_MULTICAST_DISABLE | Disable reception of multicast frames. |
| ETHER_MULTICAST_ENABLE | Enable reception of multicast frames. |

### ◆ ether_promiscuous_t

| enum ether_promiscuous_t | |
|---|---|
| Promiscuous Mode | |
| Enumerator | |
| ETHER_PROMISCUOUS_DISABLE | Only receive packets with current MAC address, multicast, and broadcast. |
| ETHER_PROMISCUOUS_ENABLE | Receive all packets. |

### ◆ ether_zerocopy_t

| enum ether_zerocopy_t | |
|---|---|
| Zero copy | |
| Enumerator | |
| ETHER_ZEROCOPY_DISABLE | Disable zero copy in Read/Write function. |
| ETHER_ZEROCOPY_ENABLE | Enable zero copy in Read/Write function. |

### ◆ ether_event_t

| enum ether_event_t | |
|---|---|
| Event code of callback function | |
| Enumerator | |
| ETHER_EVENT_WAKEON_LAN | Magic packet detection event. |
| ETHER_EVENT_LINK_ON | Link up detection event. |
| ETHER_EVENT_LINK_OFF | Link down detection event. |
| ETHER_EVENT_INTERRUPT | Interrupt event. |

## 4.3.14 Ethernet PHY Interface

Interfaces

### Detailed Description

Interface for Ethernet PHY functions.

# Summary

The Ethernet PHY module (r_ether_phy) provides an API for standard Ethernet PHY communications applications that use the ETHERC peripheral.

The Ethernet PHY interface supports the following features:

- Auto negotiation support
- Flow control support
- Link status check support

Implemented by:

- Ethernet PHY (r_ether_phy)

### Data Structures

| | |
|---|---|
| struct | ether_phy_cfg_t |
| struct | ether_phy_api_t |
| struct | ether_phy_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | ether_phy_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | ether_phy_flow_control_t |
| enum | ether_phy_link_speed_t |
| enum | ether_phy_mii_type_t |

## Data Structure Documentation

### ◆ ether_phy_cfg_t

| struct ether_phy_cfg_t | | |
|---|---|---|
| Configuration parameters. | | |
| Data Fields | | |
| uint8_t | channel | Channel. |
| uint8_t | phy_lsi_address | Address of PHY-LSI. |
| uint32_t | phy_reset_wait_time | Wait time for PHY-LSI reboot. |
| int32_t | mii_bit_access_wait_time | Wait time for MII/RMII access. |
| | | |

| | | |
|---|---|---|
| ether_phy_flow_control_t | flow_control | Flow control functionally enable or disable. |
| ether_phy_mii_type_t | mii_type | Interface type is MII or RMII. |
| void const * | p_context | Placeholder for user data. Passed to the user callback in ether_phy_callback_args_t. |
| void const * | p_extend | Placeholder for user extension. |

### ◆ ether_phy_api_t

struct ether_phy_api_t

Functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(ether_phy_ctrl_t *const p_api_ctrl, ether_phy_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(ether_phy_ctrl_t *const p_api_ctrl) |
| | |
| fsp_err_t(* | startAutoNegotiate )(ether_phy_ctrl_t *const p_api_ctrl) |
| | |
| fsp_err_t(* | linkPartnerAbilityGet )(ether_phy_ctrl_t *const p_api_ctrl, uint32_t *const p_line_speed_duplex, uint32_t *const p_local_pause, uint32_t *const p_partner_pause) |
| | |
| fsp_err_t(* | linkStatusGet )(ether_phy_ctrl_t *const p_api_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_data) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* ether_phy_api_t::open) (ether_phy_ctrl_t *const p_api_ctrl, ether_phy_cfg_t const *const p_cfg)

Open driver.

**Implemented as**

- R_ETHER_PHY_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ close

fsp_err_t(* ether_phy_api_t::close) (ether_phy_ctrl_t *const p_api_ctrl)

Close driver.

**Implemented as**

- R_ETHER_PHY_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |

### ◆ startAutoNegotiate

fsp_err_t(* ether_phy_api_t::startAutoNegotiate) (ether_phy_ctrl_t *const p_api_ctrl)

Start auto negotiation.

**Implemented as**

- R_ETHER_PHY_StartAutoNegotiate()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |

### ◆ linkPartnerAbilityGet

fsp_err_t(* ether_phy_api_t::linkPartnerAbilityGet) (ether_phy_ctrl_t *const p_api_ctrl, uint32_t *const p_line_speed_duplex, uint32_t *const p_local_pause, uint32_t *const p_partner_pause)

Get the partner ability.

**Implemented as**

- R_ETHER_PHY_LinkPartnerAbilityGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [out] | p_line_speed_duplex | Pointer to the location of both the line speed and the duplex. |
| [out] | p_local_pause | Pointer to the location to store the local pause bits. |
| [out] | p_partner_pause | Pointer to the location to store the partner pause bits. |

### ◆ linkStatusGet

fsp_err_t(* ether_phy_api_t::linkStatusGet) (ether_phy_ctrl_t *const p_api_ctrl)

Get Link status from PHY-LSI interface.

**Implemented as**

- R_ETHER_PHY_LinkStatusGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |

### ◆ versionGet

fsp_err_t(* ether_phy_api_t::versionGet) (fsp_version_t *const p_data)

Return the version of the driver.

**Implemented as**

- R_ETHER_PHY_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_data | Memory address to return version information to. |

### ◆ ether_phy_instance_t

struct ether_phy_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| ether_phy_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| ether_phy_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| ether_phy_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ ether_phy_ctrl_t

typedef void ether_phy_ctrl_t

Control block. Allocate an instance specific control block to pass into the API calls.

**Implemented as**

- ether_phy_instance_ctrl_t

## Enumeration Type Documentation

### ◆ ether_phy_flow_control_t

enum ether_phy_flow_control_t

Flow control functionality

| Enumerator | |
|---|---|
| ETHER_PHY_FLOW_CONTROL_DISABLE | Disable flow control functionality. |
| ETHER_PHY_FLOW_CONTROL_ENABLE | Enable flow control functionality with pause frames. |

### ◆ ether_phy_link_speed_t

| enum ether_phy_link_speed_t | |
|---|---|
| Link speed | |
| Enumerator | |
| ETHER_PHY_LINK_SPEED_NO_LINK | Link is not established. |
| ETHER_PHY_LINK_SPEED_10H | Link status is 10Mbit/s and half duplex. |
| ETHER_PHY_LINK_SPEED_10F | Link status is 10Mbit/s and full duplex. |
| ETHER_PHY_LINK_SPEED_100H | Link status is 100Mbit/s and half duplex. |
| ETHER_PHY_LINK_SPEED_100F | Link status is 100Mbit/s and full duplex. |

### ◆ ether_phy_mii_type_t

| enum ether_phy_mii_type_t | |
|---|---|
| Media-independent interface | |
| Enumerator | |
| ETHER_PHY_MII_TYPE_MII | MII. |
| ETHER_PHY_MII_TYPE_RMII | RMII. |

## 4.3.15 External IRQ Interface

Interfaces

### Detailed Description

Interface for detecting external interrupts.

# Summary

The External IRQ Interface is for configuring interrupts to fire when a trigger condition is detected on an external IRQ pin.

The External IRQ Interface can be implemented by:

- Interrupt Controller Unit (r_icu)

## Data Structures

| | |
|---|---|
| struct | external_irq_callback_args_t |
| struct | external_irq_cfg_t |
| struct | external_irq_api_t |
| struct | external_irq_instance_t |

## Macros

| | |
|---|---|
| #define | EXTERNAL_IRQ_API_VERSION_MAJOR<br>EXTERNAL IRQ API version number (Major) |
| #define | EXTERNAL_IRQ_API_VERSION_MINOR<br>EXTERNAL IRQ API version number (Minor) |

## Typedefs

| | |
|---|---|
| typedef void | external_irq_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | external_irq_trigger_t |
| enum | external_irq_pclk_div_t |

## Data Structure Documentation

### ◆ external_irq_callback_args_t

| struct external_irq_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| 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. |

### ◆ external_irq_cfg_t

| struct external_irq_cfg_t |
|---|
| User configuration structure, used in open function |
| **Data Fields** |

| | |
|---|---|
| uint8_t | channel |
| | Hardware channel used. |
| | |
| uint8_t | ipl |
| | Interrupt priority. |
| | |
| IRQn_Type | irq |
| | NVIC interrupt number assigned to this instance. |
| | |
| external_irq_trigger_t | trigger |
| | Trigger setting. |
| | |
| external_irq_pclk_div_t | pclk_div |
| | Digital filter clock divisor setting. |
| | |
| bool | filter_enable |
| | Digital filter enable/disable setting. |
| | |
| void(* | p_callback )(external_irq_callback_args_t *p_args) |
| | |
| void const * | p_context |
| | |
| void const * | p_extend |
| | External IRQ hardware dependent configuration. |
| | |

## Field Documentation

### ◆ p_callback

void(* external_irq_cfg_t::p_callback) (external_irq_callback_args_t *p_args)

Callback provided external input trigger occurs.

### ◆ p_context

void const* external_irq_cfg_t::p_context

Placeholder for user data. Passed to the user callback in external_irq_callback_args_t.

### ◆ external_irq_api_t

struct external_irq_api_t

External interrupt driver structure. External interrupt functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(external_irq_ctrl_t *const p_ctrl, external_irq_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | enable )(external_irq_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | disable )(external_irq_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | close )(external_irq_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* external_irq_api_t::open) (external_irq_ctrl_t *const p_ctrl, external_irq_cfg_t const *const p_cfg)

Initial configuration.

**Implemented as**

- R_ICU_ExternalIrqOpen()

**Parameters**

| | | |
|---|---|---|
| [out] | p_ctrl | Pointer to control block. Must be declared by user. Value set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of the structure must be set by user. |

### ◆ enable

fsp_err_t(* external_irq_api_t::enable) (external_irq_ctrl_t *const p_ctrl)

Enable callback when an external trigger condition occurs.

**Implemented as**

- R_ICU_ExternalIrqEnable()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in Open call for this external interrupt. |

### ◆ disable

fsp_err_t(* external_irq_api_t::disable) (external_irq_ctrl_t *const p_ctrl)

Disable callback when external trigger condition occurs.

**Implemented as**

- R_ICU_ExternalIrqDisable()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in Open call for this external interrupt. |

### ◆ close

| fsp_err_t(* external_irq_api_t::close) (external_irq_ctrl_t *const p_ctrl) |
|---|

Allow driver to be reconfigured. May reduce power consumption.

**Implemented as**

- R_ICU_ExternalIrqClose()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in Open call for this external interrupt. |

### ◆ versionGet

| fsp_err_t(* external_irq_api_t::versionGet) (fsp_version_t *const p_version) |
|---|

Get version and store it in provided pointer p_version.

**Implemented as**

- R_ICU_ExternalIrqVersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version used. |

### ◆ external_irq_instance_t

| struct external_irq_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| 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. |

## Typedef Documentation

### ◆ external_irq_ctrl_t

| typedef void external_irq_ctrl_t |
|---|
| External IRQ control block. Allocate an instance specific control block to pass into the external IRQ API calls.<br><br>**Implemented as**<br><br>◦ icu_instance_ctrl_t |

## Enumeration Type Documentation

### ◆ external_irq_trigger_t

| enum external_irq_trigger_t | |
|---|---|
| Condition that will trigger an interrupt when detected. | |
| Enumerator | |
| 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. |

### ◆ external_irq_pclk_div_t

| enum external_irq_pclk_div_t | |
|---|---|
| External IRQ input pin digital filtering sample clock divisor settings. The digital filter rejects trigger conditions that are shorter than 3 periods of the filter clock. | |
| Enumerator | |
| 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. |

## 4.3.16 Flash Interface

Interfaces

### Detailed Description

Interface for the Flash Memory.

## Summary

The Flash interface provides the ability to read, write, erase, and blank check the code flash and data flash regions.

The Flash interface is implemented by:

- Low-Power Flash Driver (r_flash_lp)

### Data Structures

| | |
|---|---|
| struct | flash_block_info_t |
| struct | flash_regions_t |
| struct | flash_info_t |
| struct | flash_callback_args_t |
| struct | flash_cfg_t |
| struct | flash_api_t |
| struct | flash_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | flash_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | flash_result_t |
| enum | flash_startup_area_swap_t |
| enum | flash_event_t |
| enum | flash_id_code_mode_t |
| enum | flash_status_t |

### Data Structure Documentation

#### ◆ flash_block_info_t

| struct flash_block_info_t | | |
|---|---|---|
| Flash block details stored in factory flash. | | |
| Data Fields | | |
| 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. |

### ◆ flash_regions_t

| struct flash_regions_t | | |
|---|---|---|
| Flash block details | | |
| Data Fields | | |
| uint32_t | num_regions | Length of block info array. |
| flash_block_info_t const * | p_block_array | Block info array base address. |

### ◆ flash_info_t

| struct flash_info_t | | |
|---|---|---|
| Information about the flash blocks | | |
| Data Fields | | |
| flash_regions_t | code_flash | Information about the code flash regions. |
| flash_regions_t | data_flash | Information about the code flash regions. |

### ◆ flash_callback_args_t

| struct flash_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| 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. |

### ◆ flash_cfg_t

| struct flash_cfg_t |
|---|
| FLASH Configuration |

| Data Fields | |
|---|---|
| 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 | ipl |
| | Flash ready interrupt priority. |
| | |
| IRQn_Type | irq |
| | Flash ready interrupt number. |
| | |
| uint8_t | err_ipl |
| | Flash error interrupt priority (unused in r_flash_lp) |
| | |
| IRQn_Type | err_irq |
| | Flash error interrupt number (unused in r_flash_lp) |
| | |

### ◆ flash_api_t

| struct flash_api_t |
|---|
| Shared Interface definition for FLASH |

| **Data Fields** | |
|---|---|
| fsp_err_t(* | open )(flash_ctrl_t *const p_ctrl, flash_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | write )(flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes) |
| | |
| fsp_err_t(* | erase )(flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_blocks) |
| | |
| fsp_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) |
| | |
| fsp_err_t(* | infoGet )(flash_ctrl_t *const p_ctrl, flash_info_t *const p_info) |
| | |
| fsp_err_t(* | close )(flash_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | statusGet )(flash_ctrl_t *const p_ctrl, flash_status_t *const p_status) |
| | |
| fsp_err_t(* | accessWindowSet )(flash_ctrl_t *const p_ctrl, uint32_t const start_addr, uint32_t const end_addr) |
| | |
| fsp_err_t(* | accessWindowClear )(flash_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | idCodeSet )(flash_ctrl_t *const p_ctrl, uint8_t const *const p_id_bytes, flash_id_code_mode_t mode) |
| | |
| fsp_err_t(* | reset )(flash_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | updateFlashClockFreq )(flash_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | startupAreaSelect )(flash_ctrl_t *const p_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* flash_api_t::open) (flash_ctrl_t *const p_ctrl, flash_cfg_t const *const p_cfg)

Open FLASH device.

**Implemented as**

- R_FLASH_LP_Open()
- R_FLASH_HP_Open()

**Parameters**

| | | |
|---|---|---|
| [out] | p_ctrl | Pointer to FLASH device control. Must be declared by user. Value set here. |
| [in] | flash_cfg_t | Pointer to FLASH configuration structure. All elements of this structure must be set by the user. |

### ◆ write

fsp_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)

Write FLASH device.

**Implemented as**

- R_FLASH_LP_Write()
- R_FLASH_HP_Write()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control for the FLASH device context. |
| [in] | src_address | Address of the buffer containing the data to write to Flash. |
| [in] | flash_address | Code Flash or Data Flash address to write. The address must be on a programming line boundary. |
| [in] | num_bytes | 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. |

**Warning**

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.

### ◆ erase

fsp_err_t(* flash_api_t::erase) (flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_blocks)

Erase FLASH device.

**Implemented as**

- R_FLASH_LP_Erase()
- R_FLASH_HP_Erase()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control for the FLASH device. |
| [in] | address | The block containing this address is the first block erased. |
| [in] | num_blocks | Specifies the number of blocks to be erased, the starting block determined by the block_erase_address. |

### ◆ blankCheck

fsp_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)

Blank check FLASH device.

**Implemented as**

- R_FLASH_LP_BlankCheck()
- R_FLASH_HP_BlankCheck()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control for the FLASH device context. |
| [in] | address | The starting address of the Flash area to blank check. |
| [in] | num_bytes | Specifies the number of bytes that need to be checked. See the specific handler for details. |
| [out] | p_blank_check_result | 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. |

### ◆ infoGet

fsp_err_t(* flash_api_t::infoGet) (flash_ctrl_t *const p_ctrl, flash_info_t *const p_info)

Close FLASH device.

**Implemented as**

- R_FLASH_LP_InfoGet()
- R_FLASH_HP_InfoGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |
| [out] | p_info | Pointer to FLASH info structure. |

### ◆ close

fsp_err_t(* flash_api_t::close) (flash_ctrl_t *const p_ctrl)

Close FLASH device.

**Implemented as**

- R_FLASH_LP_Close()
- R_FLASH_HP_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |

### ◆ statusGet

fsp_err_t(* flash_api_t::statusGet) (flash_ctrl_t *const p_ctrl, flash_status_t *const p_status)

Get Status for FLASH device.

**Implemented as**

- R_FLASH_LP_StatusGet()
- R_FLASH_HP_StatusGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |
| [out] | p_ctrl | Pointer to the current flash status. |

### ◆ accessWindowSet

fsp_err_t(* flash_api_t::accessWindowSet) (flash_ctrl_t *const p_ctrl, uint32_t const start_addr, uint32_t const end_addr)

Set Access Window for FLASH device.

**Implemented as**

- R_FLASH_LP_AccessWindowSet()
- R_FLASH_HP_AccessWindowSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |
| [in] | start_addr | Determines the Starting block for the Code Flash access window. |
| [in] | end_addr | Determines the Ending block for the Code Flash access window. This address will not be within the access window. |

### ◆ accessWindowClear

fsp_err_t(* flash_api_t::accessWindowClear) (flash_ctrl_t *const p_ctrl)

Clear any existing Code Flash access window for FLASH device.

**Implemented as**

- R_FLASH_LP_AccessWindowClear()
- R_FLASH_HP_AccessWindowClear()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |
| [in] | start_addr | Determines the Starting block for the Code Flash access window. |
| [in] | end_addr | Determines the Ending block for the Code Flash access window. |

### ◆ idCodeSet

fsp_err_t(* flash_api_t::idCodeSet) (flash_ctrl_t *const p_ctrl, uint8_t const *const p_id_bytes, flash_id_code_mode_t mode)

Set ID Code for FLASH device. Setting the ID code can restrict access to the device. The ID code will be required to connect to the device. Bits 126 and 127 are set based on the mode.

For example, uint8_t id_bytes[] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0x00}; with mode FLASH_ID_CODE_MODE_LOCKED_WITH_ALL_ERASE_SUPPORT will result in an ID code of 00112233445566778899aabbccddeec0

With mode FLASH_ID_CODE_MODE_LOCKED, it will result in an ID code of 00112233445566778899aabbccddee80

**Implemented as**

- R_FLASH_LP_IdCodeSet()
- R_FLASH_HP_IdCodeSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |
| [in] | p_id_bytes | Ponter to the ID Code to be written. |
| [in] | mode | Mode used for checking the ID code. |

### ◆ reset

fsp_err_t(* flash_api_t::reset) (flash_ctrl_t *const p_ctrl)

Reset function for FLASH device.

**Implemented as**

- R_FLASH_LP_Reset()
- R_FLASH_HP_Reset()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |

### ◆ updateFlashClockFreq

fsp_err_t(* flash_api_t::updateFlashClockFreq) (flash_ctrl_t *const p_ctrl)

Update Flash clock frequency (FCLK) and recalculate timeout values

**Implemented as**

- R_FLASH_LP_UpdateFlashClockFreq()
- R_FLASH_HP_UpdateFlashClockFreq()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |

### ◆ startupAreaSelect

fsp_err_t(* flash_api_t::startupAreaSelect) (flash_ctrl_t *const p_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary)

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()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |
| [in] | swap_type | FLASH_STARTUP_AREA_BLOCK0, FLASH_STARTUP_AREA_BLOCK1 or FLASH_STARTUP_AREA_BTFLG. |
| [in] | is_temporary | 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_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. |

### ◆ versionGet

fsp_err_t(* flash_api_t::versionGet) (fsp_version_t *p_version)

Get Flash driver version.

**Implemented as**

- R_FLASH_LP_VersionGet()
- R_FLASH_HP_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Returns version. |

### ◆ flash_instance_t

struct flash_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| 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. |

## Typedef Documentation

### ◆ flash_ctrl_t

typedef void flash_ctrl_t

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

## Enumeration Type Documentation

### ◆ flash_result_t

| enum flash_result_t | |
|---|---|
| Result type for certain operations | |
| Enumerator | |
| 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. |

### ◆ flash_startup_area_swap_t

| enum flash_startup_area_swap_t | |
|---|---|
| Parameter for specifying the startup area swap being requested by startupAreaSelect() | |
| Enumerator | |
| FLASH_STARTUP_AREA_BTFLG | Startup area will be set based on the value of the BTFLG. |
| FLASH_STARTUP_AREA_BLOCK0 | Startup area will be set to Block 0. |
| FLASH_STARTUP_AREA_BLOCK1 | Startup area will be set to Block 1. |

### ◆ flash_event_t

| enum flash_event_t | |
|---|---|
| Event types returned by the ISR callback when used in Data Flash BGO mode | |
| Enumerator | |
| 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. |
| FLASH_EVENT_ERR_ONE_BIT | A 1-bit error has been corrected when reading the flash memory area by the sequencer. |

### ◆ flash_id_code_mode_t

| enum flash_id_code_mode_t | |
|---|---|
| ID Code Modes for writing to ID code registers | |
| Enumerator | |
| FLASH_ID_CODE_MODE_UNLOCKED | ID code is ignored. |
| FLASH_ID_CODE_MODE_LOCKED_WITH_ALL_ERASE_SUPPORT | ID code is checked. All erase is available. |
| FLASH_ID_CODE_MODE_LOCKED | ID code is checked. |

### ◆ flash_status_t

| enum flash_status_t | |
|---|---|
| Flash status | |
| Enumerator | |
| FLASH_STATUS_IDLE | The flash is idle. |
| FLASH_STATUS_BUSY | The flash is currently processing a command. |

## 4.3.17 I2C Master Interface

Interfaces

### Detailed Description

Interface for I2C master communication.

# 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:

- I2C Master on IIC (r_iic_master)

### Data Structures

| | |
|---|---|
| struct | i2c_master_callback_args_t |
| struct | i2c_master_cfg_t |
| struct | i2c_master_api_t |
| struct | i2c_master_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | i2c_master_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | i2c_master_rate_t |
| enum | i2c_master_addr_mode_t |
| enum | i2c_master_event_t |

## Data Structure Documentation

### ◆ i2c_master_callback_args_t

| struct i2c_master_callback_args_t | | |
|---|---|---|
| I2C callback parameter definition | | |
| Data Fields | | |
| void const *const | p_context | Pointer to user-provided context. |
| i2c_master_event_t const | event | Event code. |

### ◆ i2c_master_cfg_t

| struct i2c_master_cfg_t | |
|---|---|
| I2C configuration block | |
| **Data Fields** | |
| uint8_t | channel |
| | Identifier recognizable by implementation. More... |
| | |
| i2c_master_rate_t | rate |
| | Device's maximum clock rate from enum i2c_rate_t. |
| | |
| uint32_t | slave |
| | The address of the slave device. |
| | |
| i2c_master_addr_mode_t | addr_mode |
| | Indicates how slave fields should be interpreted. |
| | |
| uint8_t | ipl |
| | Interrupt priority level. Same for RXI, TXI, TEI and ERI. |

| | |
|---|---|
| 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. |
| transfer_instance_t const * | p_transfer_tx |
| | DTC instance for I2C transmit.Set to NULL if unused. More... |
| transfer_instance_t const * | p_transfer_rx |
| | DTC instance for I2C receive. Set to NULL if unused. |
| void(* | p_callback )(i2c_master_callback_args_t *p_args) |
| | Pointer to callback function. More... |
| void const * | p_context |
| | Pointer to the user-provided context. |
| void const * | p_extend |
| | Any configuration data needed by the hardware. More... |

## Field Documentation

### ◆ channel

uint8_t i2c_master_cfg_t::channel

Identifier recognizable by implementation.

Generic configuration

### ◆ p_transfer_tx

transfer_instance_t const* i2c_master_cfg_t::p_transfer_tx

DTC instance for I2C transmit.Set to NULL if unused.

DTC support

### ◆ p_callback

void(* i2c_master_cfg_t::p_callback) (i2c_master_callback_args_t *p_args)

Pointer to callback function.

Parameters to control software behavior

### ◆ p_extend

void const* i2c_master_cfg_t::p_extend

Any configuration data needed by the hardware.

Implementation-specific configuration

### ◆ i2c_master_api_t

struct i2c_master_api_t

Interface definition for I2C access as master

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(i2c_master_ctrl_t *const p_ctrl, i2c_master_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | read )(i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart) |
| | |
| fsp_err_t(* | write )(i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart) |
| | |
| fsp_err_t(* | abort )(i2c_master_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | slaveAddressSet )(i2c_master_ctrl_t *const p_ctrl, uint32_t const |

| | |
|---|---|
| | slave, i2c_master_addr_mode_t const addr_mode) |
| | |
| fsp_err_t(* | close )(i2c_master_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* i2c_master_api_t::open) (i2c_master_ctrl_t *const p_ctrl, i2c_master_cfg_t const *const p_cfg)

Opens the I2C Master driver and initializes the hardware.

**Implemented as**

- R_IIC_MASTER_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements are set here. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ read

fsp_err_t(* i2c_master_api_t::read) (i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)

Performs a read operation on an I2C Master device.

**Implemented as**

- R_IIC_MASTER_Read()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block set in i2c_api_master_t::open call. |
| [in] | p_dest | Pointer to the location to store read data. |
| [in] | bytes | Number of bytes to read. |
| [in] | restart | Specify if the restart condition should be issued after reading. |

### ◆ write

fsp_err_t(* i2c_master_api_t::write) (i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)

Performs a write operation on an I2C Master device.

**Implemented as**

- R_IIC_MASTER_Write()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block set in i2c_api_master_t::open call. |
| [in] | p_src | Pointer to the location to get write data from. |
| [in] | bytes | Number of bytes to write. |
| [in] | restart | Specify if the restart condition should be issued after writing. |

### ◆ abort

fsp_err_t(* i2c_master_api_t::abort) (i2c_master_ctrl_t *const p_ctrl)

Performs a reset of the peripheral.

**Implemented as**

- R_IIC_MASTER_Abort()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block set in i2c_api_master_t::open call. |

### ◆ slaveAddressSet

fsp_err_t(* i2c_master_api_t::slaveAddressSet) (i2c_master_ctrl_t *const p_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode)

Sets address of the slave device without reconfiguring the bus.

**Implemented as**

- R_IIC_MASTER_SlaveAddressSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block set in i2c_api_master_t::open call. |
| [in] | slave_address | Address of the slave device. |
| [in] | address_mode | Addressing mode. |

### ◆ close

fsp_err_t(* i2c_master_api_t::close) (i2c_master_ctrl_t *const p_ctrl)

Closes the driver and releases the I2C Master device.

**Implemented as**

- R_IIC_MASTER_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block set in i2c_api_master_t::open call. |

### ◆ versionGet

fsp_err_t(* i2c_master_api_t::versionGet) (fsp_version_t *const p_version)

Gets version information and stores it in the provided version struct.

**Implemented as**

- R_IIC_MASTER_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version used. |

### ◆ i2c_master_instance_t

struct i2c_master_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| i2c_master_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| i2c_master_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| i2c_master_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ i2c_master_ctrl_t

typedef void i2c_master_ctrl_t

I2C control block. Allocate an instance specific control block to pass into the I2C API calls.

**Implemented as**

- iic_master_instance_ctrl_t

## Enumeration Type Documentation

### ◆ i2c_master_rate_t

enum i2c_master_rate_t

Communication speed options

| Enumerator | |
|---|---|
| I2C_MASTER_RATE_STANDARD | 100 kHz |
| I2C_MASTER_RATE_FAST | 400 kHz |
| I2C_MASTER_RATE_FASTPLUS | 1 MHz |

### ◆ i2c_master_addr_mode_t

enum i2c_master_addr_mode_t

Addressing mode options

| Enumerator | |
|---|---|
| I2C_MASTER_ADDR_MODE_7BIT | Use 7-bit addressing mode. |
| I2C_MASTER_ADDR_MODE_10BIT | Use 10-bit addressing mode. |

### ◆ i2c_master_event_t

| enum i2c_master_event_t | |
|---|---|
| Callback events | |
| Enumerator | |
| I2C_MASTER_EVENT_ABORTED | A transfer was aborted. |
| I2C_MASTER_EVENT_RX_COMPLETE | A receive operation was completed successfully. |
| I2C_MASTER_EVENT_TX_COMPLETE | A transmit operation was completed successfully. |

## 4.3.18 I2C Slave Interface

Interfaces

### Detailed Description

Interface for I2C slave communication.

# Summary

The I2C slave interface provides a common API for I2C HAL drivers. The I2C slave interface supports:

- Interrupt driven transmit/receive processing
- Callback function support which returns a event codes

Implemented by:

- I2C Slave on IIC (r_iic_slave)

### Data Structures

| | |
|---|---|
| struct | i2c_slave_callback_args_t |
| struct | i2c_slave_cfg_t |
| struct | i2c_slave_api_t |
| struct | i2c_slave_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | i2c_slave_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | i2c_slave_rate_t |
| enum | i2c_slave_addr_mode_t |
| enum | i2c_slave_event_t |

## Data Structure Documentation

### ◆ i2c_slave_callback_args_t

| struct i2c_slave_callback_args_t | | |
|---|---|---|
| I2C callback parameter definition | | |
| Data Fields | | |
| void const *const | p_context | Pointer to user-provided context. |
| uint32_t const | bytes | Number of received/transmitted bytes in buffer. |
| i2c_slave_event_t const | event | Event code. |

### ◆ i2c_slave_cfg_t

| struct i2c_slave_cfg_t | |
|---|---|
| I2C configuration block | |
| **Data Fields** | |
| uint8_t | channel |
| | Identifier recognizable by implementation. More... |
| | |
| i2c_slave_rate_t | rate |
| | Device's maximum clock rate from enum i2c_rate_t. |
| | |
| uint16_t | slave |
| | The address of the slave device. |
| | |
| i2c_slave_addr_mode_t | addr_mode |
| | Indicates how slave fields should be interpreted. |
| | |

| | |
|---|---|
| bool | general_call_enable |
| | Allow a General call from master. |
| | |
| 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 | ipl |
| | Interrupt priority level. |
| | |
| void(* | p_callback )(i2c_slave_callback_args_t *p_args) |
| | Pointer to callback function. More... |
| | |
| void const * | p_context |
| | Pointer to the user-provided context. |
| | |
| void const * | p_extend |
| | Any configuration data needed by the hardware. More... |

## Field Documentation

### ◆ channel

uint8_t i2c_slave_cfg_t::channel

Identifier recognizable by implementation.

Generic configuration

### ◆ p_callback

void(* i2c_slave_cfg_t::p_callback) (i2c_slave_callback_args_t *p_args)

Pointer to callback function.

Parameters to control software behavior

### ◆ p_extend

void const* i2c_slave_cfg_t::p_extend

Any configuration data needed by the hardware.

Implementation-specific configuration

### ◆ i2c_slave_api_t

struct i2c_slave_api_t

Interface definition for I2C access as slave

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(i2c_slave_ctrl_t *const p_ctrl, i2c_slave_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | read )(i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| | |
| fsp_err_t(* | write )(i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes) |
| | |
| fsp_err_t(* | close )(i2c_slave_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* i2c_slave_api_t::open) (i2c_slave_ctrl_t *const p_ctrl, i2c_slave_cfg_t const *const p_cfg)

Opens the I2C Slave driver and initializes the hardware.

**Implemented as**

- R_IIC_SLAVE_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements are set here. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ read

fsp_err_t(* i2c_slave_api_t::read) (i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

Performs a read operation on an I2C Slave device.

**Implemented as**

- R_IIC_SLAVE_Read()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block set in i2c_slave_api_t::open call. |
| [in] | p_dest | Pointer to the location to store read data. |
| [in] | bytes | Number of bytes to read. |

### ◆ write

fsp_err_t(* i2c_slave_api_t::write) (i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes)

Performs a write operation on an I2C Slave device.

**Implemented as**

- R_IIC_SLAVE_Write()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block set in i2c_slave_api_t::open call. |
| [in] | p_src | Pointer to the location to get write data from. |
| [in] | bytes | Number of bytes to write. |

### ◆ close

fsp_err_t(* i2c_slave_api_t::close) (i2c_slave_ctrl_t *const p_ctrl)

Closes the driver and releases the I2C Slave device.

**Implemented as**

- R_IIC_SLAVE_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block set in i2c_slave_api_t::open call. |

### ◆ versionGet

fsp_err_t(* i2c_slave_api_t::versionGet) (fsp_version_t *const p_version)

Gets version information and stores it in the provided version struct.

**Implemented as**

- R_IIC_SLAVE_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version used. |

### ◆ i2c_slave_instance_t

struct i2c_slave_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| i2c_slave_ctrl_t * | p_ctrl | Pointer to the control structure |

| | | |
|---|---|---|
| | | for this instance. |
| i2c_slave_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| i2c_slave_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ i2c_slave_ctrl_t

| typedef void i2c_slave_ctrl_t |
|---|
| I2C control block. Allocate an instance specific control block to pass into the I2C API calls.<br><br>**Implemented as**<br><br>◦ iic_slave_instance_ctrl_t |

## Enumeration Type Documentation

### ◆ i2c_slave_rate_t

| enum i2c_slave_rate_t | |
|---|---|
| Communication speed options | |
| Enumerator | |
| I2C_SLAVE_RATE_STANDARD | 100 kHz |
| I2C_SLAVE_RATE_FAST | 400 kHz |
| I2C_SLAVE_RATE_FASTPLUS | 1 MHz |

### ◆ i2c_slave_addr_mode_t

| enum i2c_slave_addr_mode_t | |
|---|---|
| Addressing mode options | |
| Enumerator | |
| I2C_SLAVE_ADDR_MODE_7BIT | Use 7-bit addressing mode. |
| I2C_SLAVE_ADDR_MODE_10BIT | Use 10-bit addressing mode. |

### ◆ i2c_slave_event_t

| enum i2c_slave_event_t | |
|---|---|
| Callback events | |
| Enumerator | |
| I2C_SLAVE_EVENT_ABORTED | A transfer was aborted. |
| I2C_SLAVE_EVENT_RX_COMPLETE | A receive operation was completed successfully. |
| I2C_SLAVE_EVENT_TX_COMPLETE | A transmit operation was completed successfully. |
| I2C_SLAVE_EVENT_RX_REQUEST | A read operation expected from slave. Detected a write from master. |
| I2C_SLAVE_EVENT_TX_REQUEST | A write operation expected from slave. Detected a read from master. |
| I2C_SLAVE_EVENT_RX_MORE_REQUEST | A read operation expected from slave. Master sends out more data than configured to be read in slave. |
| I2C_SLAVE_EVENT_TX_MORE_REQUEST | A write operation expected from slave. Master requests more data than configured to be written by slave. |
| I2C_SLAVE_EVENT_GENERAL_CALL | General Call address received from Master. Detected a write from master. |

## 4.3.19 I2S Interface

Interfaces

### Detailed Description

Interface for I2S audio communication.

# Summary

The I2S (Inter-IC Sound) interface provides APIs and definitions for I2S audio communication.

# Known Implementations

## Serial Sound Interface (r_ssi)

### Data Structures

| | |
|---|---|
| struct | i2s_callback_args_t |
| struct | i2s_status_t |
| struct | i2s_cfg_t |
| struct | i2s_api_t |
| struct | i2s_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | i2s_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | i2s_pcm_width_t |
| enum | i2s_word_length_t |
| enum | i2s_event_t |
| enum | i2s_mode_t |
| enum | i2s_mute_t |
| enum | i2s_ws_continue_t |
| enum | i2s_state_t |

## Data Structure Documentation

### ◆ i2s_callback_args_t

| struct i2s_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| 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). |

### ◆ i2s_status_t

| struct i2s_status_t | | |
|---|---|---|
| I2S status. | | |
| Data Fields | | |
| i2s_state_t | state | Current I2S state. |

### ◆ i2s_cfg_t

| struct i2s_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| uint32_t | channel |
| | |
| 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. |
| | |
| i2s_mode_t | operating_mode |
| | Master or slave mode. |
| | |
| 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 |

| | |
|---|---|
| | 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. |
| | |
| IRQn_Type | txi_irq |
| | Transmit IRQ number. |
| | |
| IRQn_Type | rxi_irq |
| | Receive IRQ number. |
| | |
| IRQn_Type | int_irq |
| | Idle/Error IRQ number. |
| | |

## Field Documentation

### ◆ channel

uint32_t i2s_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

### ◆ p_transfer_tx

transfer_instance_t const* i2s_cfg_t::p_transfer_tx

To use DTC during write, link a DTC instance here. Set to NULL if unused.

### ◆ p_transfer_rx

transfer_instance_t const* i2s_cfg_t::p_transfer_rx

To use DTC during read, link a DTC instance here. Set to NULL if unused.

### ◆ p_callback

void(* i2s_cfg_t::p_callback) (i2s_callback_args_t *p_args)

Callback provided when an I2S ISR occurs. Set to NULL for no CPU interrupt.

### ◆ p_context

void const* i2s_cfg_t::p_context

Placeholder for user data. Passed to the user callback in i2s_callback_args_t.

### ◆ i2s_api_t

struct i2s_api_t

I2S functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | stop )(i2s_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | mute )(i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable) |
| | |
| fsp_err_t(* | write )(i2s_ctrl_t *const p_ctrl, void const *const p_src, uint32_t const bytes) |
| | |
| fsp_err_t(* | read )(i2s_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes) |
| | |
| fsp_err_t(* | writeRead )(i2s_ctrl_t *const p_ctrl, void const *const p_src, void *const p_dest, uint32_t const bytes) |
| | |
| fsp_err_t(* | statusGet )(i2s_ctrl_t *const p_ctrl, i2s_status_t *const p_status) |
| | |
| fsp_err_t(* | close )(i2s_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* i2s_api_t::open) (i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg)

Initial configuration.

**Implemented as**

- R_SSI_Open()

**Precondition**

Peripheral clocks and any required output pins should be configured prior to calling this function.

***Note***

*To reconfigure after calling this function, call i2s_api_t::close first.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ stop

fsp_err_t(* i2s_api_t::stop) (i2s_ctrl_t *const p_ctrl)

Stop communication. Communication is stopped when callback is called with I2S_EVENT_IDLE.

**Implemented as**

- R_SSI_Stop()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i2s_api_t::open call for this instance. |

### ◆ mute

fsp_err_t(* i2s_api_t::mute) (i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)

Enable or disable mute.

**Implemented as**

- R_SSI_Mute()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i2s_api_t::open call for this instance. |
| [in] | mute_enable | Whether to enable or disable mute. |

### ◆ write

fsp_err_t(* i2s_api_t::write) (i2s_ctrl_t *const p_ctrl, void const *const p_src, uint32_t const 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.

**Implemented as**

- R_SSI_Write()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i2s_api_t::open call for this instance. |
| [in] | p_src | Buffer of PCM samples. Must be 4 byte aligned. |
| [in] | bytes | 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. |

### ◆ read

fsp_err_t(* i2s_api_t::read) (i2s_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes)

Read I2S data. Reception is complete when callback is called with I2S_EVENT_RX_EMPTY.

**Implemented as**

- R_SSI_Read()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i2s_api_t::open call for this instance. |
| [in] | p_dest | Buffer to store PCM samples. Must be 4 byte aligned. |
| [in] | bytes | 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. |

### ◆ writeRead

fsp_err_t(* i2s_api_t::writeRead) (i2s_ctrl_t *const p_ctrl, void const *const p_src, void *const p_dest, uint32_t const bytes)

Simultaneously write and read I2S data. Transmission and reception are complete when callback is called with I2S_EVENT_IDLE.

**Implemented as**

- R_SSI_WriteRead()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i2s_api_t::open call for this instance. |
| [in] | p_src | Buffer of PCM samples. Must be 4 byte aligned. |
| [in] | p_dest | Buffer to store PCM samples. Must be 4 byte aligned. |
| [in] | bytes | 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. |

### ◆ statusGet

fsp_err_t(* i2s_api_t::statusGet) (i2s_ctrl_t *const p_ctrl, i2s_status_t *const p_status)

Get current status and store it in provided pointer p_status.

**Implemented as**

- R_SSI_StatusGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i2s_api_t::open call for this instance. |
| [out] | p_status | Current status of the driver. |

### ◆ close

fsp_err_t(* i2s_api_t::close) (i2s_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

**Implemented as**

- R_SSI_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i2s_api_t::open call for this instance. |

### ◆ versionGet

fsp_err_t(* i2s_api_t::versionGet) (fsp_version_t *const p_version)

Get version and store it in provided pointer p_version.

**Implemented as**

- R_SSI_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version used. |

### ◆ i2s_instance_t

struct i2s_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| 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. |

## Typedef Documentation

### ◆ i2s_ctrl_t

| typedef void i2s_ctrl_t |
|---|
| I2S control block. Allocate an instance specific control block to pass into the I2S API calls.<br><br>**Implemented as**<br><br>◦ ssi_instance_ctrl_t |

## Enumeration Type Documentation

### ◆ i2s_pcm_width_t

| enum i2s_pcm_width_t | |
|---|---|
| Audio PCM width | |
| Enumerator | |
| 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. |
| I2S_PCM_WIDTH_32_BITS | Using 24-bit PCM. |

### ◆ i2s_word_length_t

| enum i2s_word_length_t | |
|---|---|
| Audio system word length. | |
| Enumerator | |
| 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. |
| I2S_WORD_LENGTH_48_BITS | Using 48-bit system word length. |
| I2S_WORD_LENGTH_64_BITS | Using 64-bit system word length. |
| I2S_WORD_LENGTH_128_BITS | Using 128-bit system word length. |
| I2S_WORD_LENGTH_256_BITS | Using 256-bit system word length. |

### ◆ i2s_event_t

| enum i2s_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| 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. |

### ◆ i2s_mode_t

| enum i2s_mode_t | |
|---|---|
| I2S communication mode | |
| Enumerator | |
| I2S_MODE_SLAVE | Slave mode. |
| I2S_MODE_MASTER | Master mode. |

### ◆ i2s_mute_t

| enum i2s_mute_t | |
|---|---|
| Mute audio samples. | |
| Enumerator | |
| I2S_MUTE_OFF | Disable mute. |
| I2S_MUTE_ON | Enable mute. |

### ◆ i2s_ws_continue_t

| enum i2s_ws_continue_t | |
|---|---|
| Whether to continue WS (word select line) transmission during idle state. | |
| Enumerator | |
| I2S_WS_CONTINUE_ON | Enable WS continue mode. |
| I2S_WS_CONTINUE_OFF | Disable WS continue mode. |

### ◆ i2s_state_t

| enum i2s_state_t | |
|---|---|
| Possible status values returned by i2s_api_t::statusGet. | |
| Enumerator | |
| I2S_STATE_IN_USE | I2S is in use. |
| I2S_STATE_STOPPED | I2S is stopped. |

## 4.3.20 I/O Port Interface

Interfaces

### Detailed Description

Interface for accessing I/O ports and configuring I/O functionality.

# Summary

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.

IOPORT Interface description: I/O Ports (r_ioport)

### Data Structures

| | |
|---|---|
| struct | ioport_pin_cfg_t |
| struct | ioport_cfg_t |
| struct | ioport_api_t |
| struct | ioport_instance_t |

### Typedefs

| | |
|---|---|
| typedef uint16_t | ioport_size_t<br>IO port size on this device. More... |
| typedef void | ioport_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | ioport_peripheral_t |
| enum | ioport_ethernet_channel_t |
| enum | ioport_ethernet_mode_t |
| enum | ioport_cfg_options_t |
| enum | ioport_pwpr_t |

### Data Structure Documentation

#### ◆ ioport_pin_cfg_t

| struct ioport_pin_cfg_t | | |
|---|---|---|
| Pin identifier and pin PFS pin configuration value | | |
| Data Fields | | |
| uint32_t | pin_cfg | Pin PFS configuration - Use ioport_cfg_options_t parameters to configure. |
| bsp_io_port_pin_t | pin | Pin identifier. |

### ◆ ioport_cfg_t

| struct ioport_cfg_t | | |
|---|---|---|
| Multiple pin configuration data for loading into PFS registers by R_IOPORT_Init() | | |
| Data Fields | | |
| 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. |

### ◆ ioport_api_t

| struct ioport_api_t | |
|---|---|
| IOPort driver structure. IOPort functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg) |
| | |
| fsp_err_t(* | close )(ioport_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | pinsCfg )(ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg) |
| | |
| fsp_err_t(* | pinCfg )(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, uint32_t cfg) |
| | |
| fsp_err_t(* | pinEventInputRead )(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_event) |
| | |
| fsp_err_t(* | pinEventOutputWrite )(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t pin_value) |
| | |
| fsp_err_t(* | pinEthernetModeCfg )(ioport_ctrl_t *const p_ctrl, ioport_ethernet_channel_t channel, ioport_ethernet_mode_t mode) |
| | |

| | |
|---|---|
| fsp_err_t(* | pinRead )(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_value) |
| | |
| fsp_err_t(* | pinWrite )(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t level) |
| | |
| fsp_err_t(* | portDirectionSet )(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t direction_values, ioport_size_t mask) |
| | |
| fsp_err_t(* | portEventInputRead )(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_event_data) |
| | |
| fsp_err_t(* | portEventOutputWrite )(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t event_data, ioport_size_t mask_value) |
| | |
| fsp_err_t(* | portRead )(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_port_value) |
| | |
| fsp_err_t(* | portWrite )(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t value, ioport_size_t mask) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *p_data) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* ioport_api_t::open) (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)

Initialize internal driver data and initial pin configurations. Called during startup. Do not call this API during runtime. Use ioport_api_t::pinsCfg for runtime reconfiguration of multiple pins.

**Implemented as**

- R_IOPORT_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_cfg | Pointer to pin configuration data array. |

### ◆ close

fsp_err_t(* ioport_api_t::close) (ioport_ctrl_t *const p_ctrl)

Close the API.

**Implemented as**

- R_IOPORT_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ pinsCfg

fsp_err_t(* ioport_api_t::pinsCfg) (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)

Configure multiple pins.

**Implemented as**

- R_IOPORT_PinsCfg()

**Parameters**

| | | |
|---|---|---|
| [in] | p_cfg | Pointer to pin configuration data array. |

### ◆ pinCfg

fsp_err_t(* ioport_api_t::pinCfg) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, uint32_t cfg)

Configure settings for an individual pin.

**Implemented as**

- R_IOPORT_PinCfg()

**Parameters**

| | | |
|---|---|---|
| [in] | pin | Pin to be read. |
| [in] | cfg | Configuration options for the pin. |

### ◆ pinEventInputRead

fsp_err_t(* ioport_api_t::pinEventInputRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_event)

Read the event input data of the specified pin and return the level.

**Implemented as**

- R_IOPORT_PinEventInputRead()

**Parameters**

| | | |
|---|---|---|
| [in] | pin | Pin to be read. |
| [in] | p_pin_event | Pointer to return the event data. |

### ◆ pinEventOutputWrite

fsp_err_t(* ioport_api_t::pinEventOutputWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t pin_value)

Write pin event data.

**Implemented as**

- R_IOPORT_PinEventOutputWrite()

**Parameters**

| | | |
|---|---|---|
| [in] | pin | Pin event data is to be written to. |
| [in] | pin_value | Level to be written to pin output event. |

### ◆ pinEthernetModeCfg

fsp_err_t(* ioport_api_t::pinEthernetModeCfg) (ioport_ctrl_t *const p_ctrl, ioport_ethernet_channel_t channel, ioport_ethernet_mode_t mode)

Configure the PHY mode of the Ethernet channels.

**Implemented as**

- R_IOPORT_EthernetModeCfg()

**Parameters**

| | | |
|---|---|---|
| [in] | channel | Channel configuration will be set for. |
| [in] | mode | PHY mode to set the channel to. |

### ◆ pinRead

fsp_err_t(* ioport_api_t::pinRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_value)

Read level of a pin.

**Implemented as**

- R_IOPORT_PinRead()

**Parameters**

| | | |
|---|---|---|
| [in] | pin | Pin to be read. |
| [in] | p_pin_value | Pointer to return the pin level. |

### ◆ pinWrite

fsp_err_t(* ioport_api_t::pinWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t level)

Write specified level to a pin.

**Implemented as**

- R_IOPORT_PinWrite()

**Parameters**

| | | |
|---|---|---|
| [in] | pin | Pin to be written to. |
| [in] | level | State to be written to the pin. |

### ◆ portDirectionSet

fsp_err_t(* ioport_api_t::portDirectionSet) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t direction_values, ioport_size_t mask)

Set the direction of one or more pins on a port.

**Implemented as**

- R_IOPORT_PortDirectionSet()

**Parameters**

| | | |
|---|---|---|
| [in] | port | Port being configured. |
| [in] | direction_values | Value controlling direction of pins on port (1 - output, 0 - input). |
| [in] | mask | Mask controlling which pins on the port are to be configured. |

### ◆ portEventInputRead

fsp_err_t(* ioport_api_t::portEventInputRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_event_data)

Read captured event data for a port.

**Implemented as**

- R_IOPORT_PortEventInputRead()

**Parameters**

| | | |
|---|---|---|
| [in] | port | Port to be read. |
| [in] | p_event_data | Pointer to return the event data. |

### ◆ portEventOutputWrite

fsp_err_t(* ioport_api_t::portEventOutputWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t event_data, ioport_size_t mask_value)

Write event output data for a port.

**Implemented as**

- R_IOPORT_PortEventOutputWrite()

**Parameters**

| | | |
|---|---|---|
| [in] | port | Port event data will be written to. |
| [in] | event_data | Data to be written as event data to specified port. |
| [in] | mask_value | Each bit set to 1 in the mask corresponds to that bit's value in event data. being written to port. |

### ◆ portRead

fsp_err_t(* ioport_api_t::portRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_port_value)

Read states of pins on the specified port.

**Implemented as**

- R_IOPORT_PortRead()

**Parameters**

| | | |
|---|---|---|
| [in] | port | Port to be read. |
| [in] | p_port_value | Pointer to return the port value. |

### ◆ portWrite

fsp_err_t(* ioport_api_t::portWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t value, ioport_size_t mask)

Write to multiple pins on a port.

**Implemented as**

- R_IOPORT_PortWrite()

**Parameters**

| | | |
|---|---|---|
| [in] | port | Port to be written to. |
| [in] | value | Value to be written to the port. |
| [in] | mask | Mask controlling which pins on the port are written to. |

### ◆ versionGet

fsp_err_t(* ioport_api_t::versionGet) (fsp_version_t *p_data)

Return the version of the IOPort driver.

**Implemented as**

- R_IOPORT_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_data | Memory address to return version information to. |

### ◆ ioport_instance_t

struct ioport_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| ioport_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| 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. |

## Typedef Documentation

### ◆ ioport_size_t

typedef uint16_t ioport_size_t

IO port size on this device.

IO port type used with ports

### ◆ ioport_ctrl_t

typedef void ioport_ctrl_t

IOPORT control block. Allocate an instance specific control block to pass into the IOPORT API calls.

**Implemented as**

- ioport_instance_ctrl_t

## Enumeration Type Documentation

### ◆ ioport_peripheral_t

| enum ioport_peripheral_t | |
|---|---|
| Superset of all peripheral functions. | |
| Enumerator | |
| 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 peripheral pin |
| IOPORT_PERIPHERAL_GPT0 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_GPT1 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_SCI0_2_4_6_8 | Pin will function as an SCI peripheral pin |
| IOPORT_PERIPHERAL_SCI1_3_5_7_9 | Pin will function as an SCI peripheral pin |
| IOPORT_PERIPHERAL_SPI | Pin will function as a SPI peripheral pin |
| IOPORT_PERIPHERAL_IIC | Pin will function as a IIC peripheral pin |
| IOPORT_PERIPHERAL_KEY | Pin will function as a KEY peripheral pin |
| IOPORT_PERIPHERAL_CLKOUT_COMP_RTC | Pin will function as a clock/comparator/RTC peripheral pin |
| IOPORT_PERIPHERAL_CAC_AD | Pin will function as a CAC/ADC peripheral pin |
| IOPORT_PERIPHERAL_BUS | Pin will function as a BUS peripheral pin |
| IOPORT_PERIPHERAL_CTSU | Pin will function as a CTSU peripheral pin |
| IOPORT_PERIPHERAL_LCDC | Pin will function as a segment LCD peripheral pin |
| IOPORT_PERIPHERAL_DALI | Pin will function as a DALI peripheral pin |
| IOPORT_PERIPHERAL_CAN | Pin will function as a CAN peripheral pin |
| IOPORT_PERIPHERAL_QSPI | Pin will function as a QSPI peripheral pin |
| IOPORT_PERIPHERAL_SSI | Pin will function as an SSI peripheral pin |
| IOPORT_PERIPHERAL_USB_FS | Pin will function as a USB full speed peripheral pin |

| | |
|---|---|
| IOPORT_PERIPHERAL_USB_HS | Pin will function as a USB high speed peripheral pin |
| IOPORT_PERIPHERAL_SDHI_MMC | Pin will function as an SD/MMC peripheral pin |
| IOPORT_PERIPHERAL_ETHER_MII | Pin will function as an Ethernet MMI peripheral pin |
| IOPORT_PERIPHERAL_ETHER_RMII | Pin will function as an Ethernet RMMI peripheral pin |
| IOPORT_PERIPHERAL_PDC | Pin will function as a PDC peripheral pin |
| IOPORT_PERIPHERAL_LCD_GRAPHICS | Pin will function as a graphics LCD peripheral pin |
| IOPORT_PERIPHERAL_TRACE | Pin will function as a debug trace peripheral pin |
| IOPORT_PERIPHERAL_END | Marks end of enum - used by parameter checking |

### ◆ ioport_ethernet_channel_t

| enum ioport_ethernet_channel_t | |
|---|---|
| Superset of Ethernet channels. | |
| Enumerator | |
| IOPORT_ETHERNET_CHANNEL_0 | Used to select Ethernet channel 0. |
| IOPORT_ETHERNET_CHANNEL_1 | Used to select Ethernet channel 1. |
| IOPORT_ETHERNET_CHANNEL_END | Marks end of enum - used by parameter checking. |

### ◆ ioport_ethernet_mode_t

| enum ioport_ethernet_mode_t | |
|---|---|
| Superset of Ethernet PHY modes. | |
| Enumerator | |
| IOPORT_ETHERNET_MODE_RMII | Ethernet PHY mode set to MII. |
| IOPORT_ETHERNET_MODE_MII | Ethernet PHY mode set to RMII. |
| IOPORT_ETHERNET_MODE_END | Marks end of enum - used by parameter checking. |

### ◆ ioport_cfg_options_t

| enum ioport_cfg_options_t | |
|---|---|
| Options to configure pin functions | |
| Enumerator | |
| 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. |
| 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. |

### ◆ ioport_pwpr_t

| enum ioport_pwpr_t | |
|---|---|
| Enumerator | |
| IOPORT_PFS_WRITE_DISABLE | Disable PFS write access. |
| IOPORT_PFS_WRITE_ENABLE | Enable PFS write access. |

## 4.3.21 JPEG Codec Interface

Interfaces

### Detailed Description

Interface for JPEG functions.

### Data Structures

| | |
|---|---|
| struct | jpeg_encode_image_size_t |
| struct | jpeg_callback_args_t |
| struct | jpeg_cfg_t |
| struct | jpeg_api_t |
| struct | jpeg_instance_t |

### Macros

| | |
|---|---|
| #define | JPEG_API_VERSION_MAJOR |

### Typedefs

| | |
|---|---|
| typedef void | jpeg_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | jpeg_color_space_t |
| enum | jpeg_data_order_t |
| enum | jpeg_status_t |
| enum | jpeg_decode_pixel_format_t |

| | |
|---|---|
| enum | jpeg_decode_subsample_t |

## Data Structure Documentation

### ◆ jpeg_encode_image_size_t

| struct jpeg_encode_image_size_t | | |
|---|---|---|
| Image parameter structure | | |
| Data Fields | | |
| uint16_t | horizontal_stride_pixels | Horizontal stride. |
| uint16_t | horizontal_resolution | Horizontal Resolution in pixels. |
| uint16_t | vertical_resolution | Vertical Resolution in pixels. |

### ◆ jpeg_callback_args_t

| struct jpeg_callback_args_t | | |
|---|---|---|
| Callback status structure | | |
| Data Fields | | |
| jpeg_status_t | status | JPEG status. |
| uint32_t | image_size | JPEG image size. |
| void const * | p_context | Pointer to user-provided context. |

### ◆ jpeg_cfg_t

| struct jpeg_cfg_t | |
|---|---|
| User configuration structure, used in open function. | |
| **Data Fields** | |
| IRQn_Type | jedi_irq |
| | Data transfer interrupt IRQ number. |
| | |
| IRQn_Type | jdti_irq |
| | Decompression interrupt IRQ number. |
| | |
| uint8_t | jdti_ipl |
| | Data transfer interrupt priority. |
| | |

| | |
|---|---|
| uint8_t | jedi_ipl |
| | Decompression interrupt priority. |
| | |
| jpeg_mode_t | default_mode |
| | Mode to use at startup. |
| | |
| jpeg_data_order_t | decode_input_data_order |
| | Input data stream byte order. |
| | |
| jpeg_data_order_t | decode_output_data_order |
| | 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 ARGB8888 format. |
| | |
| void(* | p_decode_callback )(jpeg_callback_args_t *p_args) |
| | User-supplied callback functions. |
| | |
| void const * | p_decode_context |
| | Placeholder for user data. Passed to user callback in jpeg_callback_args_t. |
| | |
| jpeg_data_order_t | encode_input_data_order |
| | Input data stream byte order. |
| | |

| | |
|---|---|
| jpeg_data_order_t | encode_output_data_order |
| | Output data stream byte order. |
| | |
| uint16_t | dri_marker |
| | DRI Marker setting (0 = No DRI or RST marker) |
| | |
| uint16_t | horizontal_resolution |
| | Horizontal resolution of input image. |
| | |
| uint16_t | vertical_resolution |
| | Vertical resolution of input image. |
| | |
| uint16_t | horizontal_stride_pixels |
| | Horizontal stride of input image. |
| | |
| uint8_t const * | p_quant_luma_table |
| | Luma quantization table. |
| | |
| uint8_t const * | p_quant_chroma_table |
| | Chroma quantization 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_chroma_ac_table |

| | |
|---|---|
| | Huffman AC table for chroma. |
| | |
| uint8_t const * | p_huffman_chroma_dc_table |
| | Huffman DC table for chroma. |
| | |
| void(* | p_encode_callback )(jpeg_callback_args_t *p_args) |
| | User-supplied callback functions. |
| | |
| void const * | p_encode_context |
| | Placeholder for user data. Passed to user callback in jpeg_callback_args_t. |
| | |

### ◆ jpeg_api_t

| struct jpeg_api_t | |
|---|---|
| JPEG functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(jpeg_ctrl_t *const p_ctrl, jpeg_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | inputBufferSet )(jpeg_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size) |
| | |
| fsp_err_t(* | outputBufferSet )(jpeg_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size) |
| | |
| fsp_err_t(* | statusGet )(jpeg_ctrl_t *const p_ctrl, jpeg_status_t *const p_status) |
| | |
| fsp_err_t(* | close )(jpeg_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *p_version) |
| | |
| fsp_err_t(* | horizontalStrideSet )(jpeg_ctrl_t *const p_ctrl, uint32_t horizontal_stride) |
| | |

| | |
|---|---|
| fsp_err_t(* | pixelFormatGet )(jpeg_ctrl_t *const p_ctrl, jpeg_color_space_t *const p_color_space) |
| | |
| fsp_err_t(* | imageSubsampleSet )(jpeg_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample) |
| | |
| fsp_err_t(* | linesDecodedGet )(jpeg_ctrl_t *const p_ctrl, uint32_t *const p_lines) |
| | |
| fsp_err_t(* | imageSizeGet )(jpeg_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size) |
| | |
| fsp_err_t(* | imageSizeSet )(jpeg_ctrl_t *const p_ctrl, jpeg_encode_image_size_t *p_image_size) |
| | |
| fsp_err_t(* | modeSet )(jpeg_ctrl_t *const p_ctrl, jpeg_mode_t mode) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* jpeg_api_t::open) (jpeg_ctrl_t *const p_ctrl, jpeg_cfg_t const *const p_cfg)

Initial configuration

**Implemented as**

- R_JPEG_Open()

**Precondition**

none

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ inputBufferSet

fsp_err_t(* jpeg_api_t::inputBufferSet) (jpeg_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)

Assign input data buffer to JPEG codec.

**Implemented as**

- R_JPEG_InputBufferSet()

**Precondition**

the JPEG codec module must have been opened properly.

***Note***

*The buffer starting address must be 8-byte aligned.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [in] | p_buffer | Pointer to the input buffer space |
| [in] | buffer_size | Size of the input buffer |

### ◆ outputBufferSet

fsp_err_t(* jpeg_api_t::outputBufferSet) (jpeg_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)

Assign output buffer to JPEG codec for storing output data.

**Implemented as**

- R_JPEG_OutputBufferSet()

**Precondition**

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.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [in] | p_buffer | Pointer to the output buffer space |
| [in] | buffer_size | Size of the output buffer |

### ◆ statusGet

fsp_err_t(* jpeg_api_t::statusGet) (jpeg_ctrl_t *const p_ctrl, jpeg_status_t *const p_status)

Retrieve current status of the JPEG codec module.

**Implemented as**

- R_JPEG_StatusGet()

**Precondition**

the JPEG codec module must have been opened properly.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [out] | p_status | JPEG module status |

### ◆ close

fsp_err_t(* jpeg_api_t::close) (jpeg_ctrl_t *const p_ctrl)

Cancel an outstanding operation.

**Implemented as**

- R_JPEG_Close()

**Precondition**

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.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |

### ◆ versionGet

fsp_err_t(* jpeg_api_t::versionGet) (fsp_version_t *p_version)

Get version and store it in provided pointer p_version.

**Implemented as**

- R_JPEG_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version used. |

### ◆ horizontalStrideSet

fsp_err_t(* jpeg_api_t::horizontalStrideSet) (jpeg_ctrl_t *const p_ctrl, uint32_t horizontal_stride)

Configure the horizontal stride value.

**Implemented as**

- R_JPEG_DecodeHorizontalStrideSet()

**Precondition**

The JPEG codec module must have been opened properly.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [in] | horizontal_stride | Horizontal stride value to be used for the decoded image data. |
| [in] | buffer_size | Size of the output buffer |

### ◆ pixelFormatGet

fsp_err_t(* jpeg_api_t::pixelFormatGet) (jpeg_ctrl_t *const p_ctrl, jpeg_color_space_t *const p_color_space)

Get the input pixel format.

**Implemented as**

- R_JPEG_DecodePixelFormatGet()

**Precondition**

the JPEG codec module must have been opened properly.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [out] | p_color_space | JPEG input format. |

### ◆ imageSubsampleSet

fsp_err_t(* jpeg_api_t::imageSubsampleSet) (jpeg_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample)

Configure the horizontal and vertical subsample settings.

**Implemented as**

- R_JPEG_DecodeImageSubsampleSet()

**Precondition**

The JPEG codec module must have been opened properly.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [in] | horizontal_subsample | Horizontal subsample value |
| [in] | vertical_subsample | Vertical subsample value |

### ◆ linesDecodedGet

fsp_err_t(* jpeg_api_t::linesDecodedGet) (jpeg_ctrl_t *const p_ctrl, uint32_t *const p_lines)

Return the number of lines decoded into the output buffer.

**Implemented as**

- R_JPEG_DecodeLinesDecodedGet()

**Precondition**

the JPEG codec module must have been opened properly.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [out] | p_lines | Number of lines decoded |

### ◆ imageSizeGet

fsp_err_t(* jpeg_api_t::imageSizeGet) (jpeg_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)

Retrieve image size during decoding operation.

**Implemented as**

- R_JPEG_DecodeImageSizeGet()

**Precondition**

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.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [out] | p_horizontal_size | Image horizontal size, in number of pixels. |
| [out] | p_vertical_size | Image vertical size, in number of pixels. |

### ◆ imageSizeSet

fsp_err_t(* jpeg_api_t::imageSizeSet) (jpeg_ctrl_t *const p_ctrl, jpeg_encode_image_size_t *p_image_size)

Set image parameters to JPEG Codec

**Implemented as**

- R_JPEG_EncodeImageSizeSet()

**Precondition**

The JPEG codec module must have been opened properly.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_image_size | Pointer to the RAW image parameters |

### ◆ modeSet

| fsp_err_t(* jpeg_api_t::modeSet) (jpeg_ctrl_t *const p_ctrl, jpeg_mode_t mode) |
|---|

Switch between encode and decode mode or vice-versa.

**Implemented as**

- R_JPEG_ModeSet()

**Precondition**

The JPEG codec module must have been opened properly. The JPEG Codec can only perform one operation at a time and requires different configuration for encode and decode. This function facilitates easy switching between the two modes in case both are needed in an application.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [in] | mode | Mode to switch to |

### ◆ jpeg_instance_t

| struct jpeg_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| jpeg_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| jpeg_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| jpeg_api_t const * | p_api | Pointer to the API structure for this instance. |

## Macro Definition Documentation

### ◆ JPEG_API_VERSION_MAJOR

| #define JPEG_API_VERSION_MAJOR |
|---|
| Configuration for this module |

## Typedef Documentation

### ◆ jpeg_ctrl_t

| typedef void jpeg_ctrl_t |
|---|
| JPEG decode control block. Allocate an instance specific control block to pass into the JPEG decode API calls.<br><br>**Implemented as**<br><br>◦ jpeg_instance_ctrl_t |

## Enumeration Type Documentation

### ◆ jpeg_color_space_t

| enum jpeg_color_space_t | |
|---|---|
| Image color space definitions | |
| Enumerator | |
| JPEG_COLOR_SPACE_YCBCR444 | Color Space YCbCr 444. |
| JPEG_COLOR_SPACE_YCBCR422 | Color Space YCbCr 422. |
| JPEG_COLOR_SPACE_YCBCR420 | Color Space YCbCr 420. |
| JPEG_COLOR_SPACE_YCBCR411 | Color Space YCbCr 411. |

### ◆ jpeg_data_order_t

| enum jpeg_data_order_t | |
|---|---|
| Multi-byte Data Format | |
| Enumerator | |
| JPEG_DATA_ORDER_NORMAL | (1)(2)(3)(4)(5)(6)(7)(8) Normal byte order |
| JPEG_DATA_ORDER_BYTE_SWAP | (2)(1)(4)(3)(6)(5)(8)(7) Byte Swap |
| JPEG_DATA_ORDER_WORD_SWAP | (3)(4)(1)(2)(7)(8)(5)(6) Word Swap |
| JPEG_DATA_ORDER_WORD_BYTE_SWAP | (4)(3)(2)(1)(8)(7)(6)(5) Word-Byte Swap |
| JPEG_DATA_ORDER_LONGWORD_SWAP | (5)(6)(7)(8)(1)(2)(3)(4) Longword Swap |
| JPEG_DATA_ORDER_LONGWORD_BYTE_SWAP | (6)(5)(8)(7)(2)(1)(4)(3) Longword Byte Swap |
| JPEG_DATA_ORDER_LONGWORD_WORD_SWAP | (7)(8)(5)(6)(3)(4)(1)(2) Longword Word Swap |
| JPEG_DATA_ORDER_LONGWORD_WORD_BYTE_SWAP | (8)(7)(6)(5)(4)(3)(2)(1) Longword Word Byte Swap |

### ◆ jpeg_status_t

| enum jpeg_status_t | |
|---|---|
| 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 | |
| Enumerator | |
| JPEG_STATUS_NONE | JPEG codec module is not initialized. |
| JPEG_STATUS_IDLE | JPEG Codec module is open but not running. |
| JPEG_STATUS_RUNNING | JPEG Codec is running. |
| JPEG_STATUS_HEADER_PROCESSING | JPEG Codec module is reading the JPEG header information. |
| JPEG_STATUS_INPUT_PAUSE | JPEG Codec paused waiting for more input data. |
| JPEG_STATUS_OUTPUT_PAUSE | JPEG Codec paused after it decoded the number of lines specified by user. |
| JPEG_STATUS_IMAGE_SIZE_READY | JPEG decoding operation obtained image size, and paused. |
| JPEG_STATUS_ERROR | JPEG Codec module encountered an error. |
| JPEG_STATUS_OPERATION_COMPLETE | JPEG Codec has completed the operation. |

### ◆ jpeg_decode_pixel_format_t

| enum jpeg_decode_pixel_format_t | |
|---|---|
| Pixel Data Format | |
| Enumerator | |
| JPEG_DECODE_PIXEL_FORMAT_ARGB8888 | Pixel Data ARGB8888 format. |
| JPEG_DECODE_PIXEL_FORMAT_RGB565 | Pixel Data RGB565 format. |

### ◆ jpeg_decode_subsample_t

| enum jpeg_decode_subsample_t | |
|---|---|
| Data type for horizontal and vertical subsample settings. This setting applies only to the decoding operation. | |
| Enumerator | |
| 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. |

## 4.3.22 Key Matrix Interface

Interfaces

### Detailed Description

Interface for key matrix functions.

# Summary

The KEYMATRIX interface provides standard Key Matrix 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.

Implemented by:

- Key Interrupt (r_kint)

### Data Structures

| | |
|---|---|
| struct | keymatrix_callback_args_t |
| struct | keymatrix_cfg_t |
| struct | keymatrix_api_t |

| | |
|---|---|
| struct | keymatrix_instance_t |

### Macros

| | |
|---|---|
| #define | KEYMATRIX_API_VERSION_MAJOR<br>KEY MATRIX API version number (Major) |
| #define | KEYMATRIX_API_VERSION_MINOR<br>KEY MATRIX API version number (Minor) |

### Typedefs

| | |
|---|---|
| typedef void | keymatrix_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | keymatrix_trigger_t |

## Data Structure Documentation

### ◆ keymatrix_callback_args_t

| struct keymatrix_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| void const * | p_context | Holder for user data. Set in keymatrix_api_t::open function in keymatrix_cfg_t. |
| uint32_t | channel_mask | Bit vector representing the physical hardware channel(s) that caused the interrupt. |

### ◆ keymatrix_cfg_t

| struct keymatrix_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| uint32_t | channel_mask |
| | Key Input channel(s). Bit mask of channels to open. |
| | |
| keymatrix_trigger_t | trigger |
| | Key Input trigger setting. |

| | |
|---|---|
| | |
| uint8_t | ipl |
| | Interrupt priority level. |
| | |
| IRQn_Type | irq |
| | NVIC IRQ number. |
| | |
| 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. |
| | |

### ◆ keymatrix_api_t

| | |
|---|---|
| struct keymatrix_api_t | |
| Key Matrix driver structure. Key Matrix functions implemented at the HAL layer will use this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(keymatrix_ctrl_t *const p_ctrl, keymatrix_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | enable )(keymatrix_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | disable )(keymatrix_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | close )(keymatrix_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* keymatrix_api_t::open) (keymatrix_ctrl_t *const p_ctrl, keymatrix_cfg_t const *const p_cfg)

Initial configuration.

**Implemented as**

- R_KINT_Open()

**Parameters**

| | | |
|---|---|---|
| [out] | p_ctrl | Pointer to control block. Must be declared by user. Value set in this function. |
| [in] | p_cfg | Pointer to configuration structure. All elements of the structure must be set by user. |

### ◆ enable

fsp_err_t(* keymatrix_api_t::enable) (keymatrix_ctrl_t *const p_ctrl)

Enable Key interrupt

**Implemented as**

- R_KINT_Enable()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block pointer set in Open call for this Key interrupt. |

### ◆ disable

fsp_err_t(* keymatrix_api_t::disable) (keymatrix_ctrl_t *const p_ctrl)

Disable Key interrupt.

**Implemented as**

- R_KINT_Disable()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block pointer set in Open call for this Key interrupt. |

### ◆ close

fsp_err_t(* keymatrix_api_t::close) (keymatrix_ctrl_t *const p_ctrl)

Allow driver to be reconfigured. May reduce power consumption.

**Implemented as**

- R_KINT_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block pointer set in Open call for this Key interrupt. |

### ◆ versionGet

fsp_err_t(* keymatrix_api_t::versionGet) (fsp_version_t *const p_version)

Get version and store it in provided pointer p_version.

**Implemented as**

- R_KINT_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version used. |

### ◆ keymatrix_instance_t

struct keymatrix_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| 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. |

## Typedef Documentation

### ◆ keymatrix_ctrl_t

| typedef void keymatrix_ctrl_t |
|---|
| Key matrix control block. Allocate an instance specific control block to pass into the key matrix API calls.<br><br>**Implemented as**<br><br>◦ kint_instance_ctrl_t |

### Enumeration Type Documentation

### ◆ keymatrix_trigger_t

| enum keymatrix_trigger_t | |
|---|---|
| Trigger type: rising edge, falling edge | |
| Enumerator | |
| KEYMATRIX_TRIG_FALLING | Falling edge trigger. |
| KEYMATRIX_TRIG_RISING | Rising edge trigger. |

## 4.3.23 Low Power Modes Interface

Interfaces

### Detailed Description

Interface for accessing low power modes.

# Summary

This section defines the API for the LPM (Low Power Mode) Driver. The LPM Driver provides functions for controlling power consumption by configuring and transitioning to a low power mode. The LPM driver supports configuration of MCU low power modes using the LPM hardware peripheral. The LPM driver supports low power modes deep standby, standby, sleep, and snooze.

*Note*

*Not all low power modes are available on all MCUs.*

The LPM interface is implemented by:

- Low Power Modes (r_lpm)

### Data Structures

| | |
|---|---|
| struct | lpm_cfg_t |
| struct | lpm_api_t |
| struct | lpm_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | lpm_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | lpm_mode_t |
| enum | lpm_snooze_request_t |
| enum | lpm_snooze_end_t |
| enum | lpm_snooze_cancel_t |
| enum | lpm_snooze_dtc_t |
| enum | lpm_standby_wake_source_t |
| enum | lpm_io_port_t |
| enum | lpm_power_supply_t |
| enum | lpm_deep_standby_cancel_edge_t |
| enum | lpm_deep_standby_cancel_source_t |
| enum | lpm_output_port_enable_t |

## Data Structure Documentation

### ◆ lpm_cfg_t

| struct lpm_cfg_t | | |
|---|---|---|
| User configuration structure, used in open function | | |
| Data Fields | | |
| lpm_mode_t | low_power_mode | Low Power Mode |
| lpm_standby_wake_source_bits_t | standby_wake_sources | Bitwise list of sources to wake from standby |
| lpm_snooze_request_t | snooze_request_source | Snooze request source |
| lpm_snooze_end_bits_t | snooze_end_sources | Bitwise list of snooze end sources |

| | | |
|---|---|---|
| lpm_snooze_cancel_t | snooze_cancel_sources | List of snooze cancel sources |
| lpm_snooze_dtc_t | dtc_state_in_snooze | State of DTC in snooze mode, enabled or disabled |
| void const * | p_extend | Placeholder for extension. |

### ◆ lpm_api_t

struct lpm_api_t

LPM driver structure. General LPM functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(lpm_ctrl_t *const p_api_ctrl) |
| | |
| fsp_err_t(* | lowPowerReconfigure )(lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | lowPowerModeEnter )(lpm_ctrl_t *const p_api_ctrl) |
| | |
| fsp_err_t(* | ioKeepClear )(lpm_ctrl_t *const p_api_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* lpm_api_t::open) (lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg)

Initialization function

**Implemented as**

- R_LPM_Open()

### ◆ close

fsp_err_t(* lpm_api_t::close) (lpm_ctrl_t *const p_api_ctrl)

Initialization function

**Implemented as**

- R_LPM_Close()

### ◆ lowPowerReconfigure

fsp_err_t(* lpm_api_t::lowPowerReconfigure) (lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg)

Configure a low power mode.

**Implemented as**

- R_LPM_LowPowerReconfigure()

**Parameters**

| | | |
|---|---|---|
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ lowPowerModeEnter

fsp_err_t(* lpm_api_t::lowPowerModeEnter) (lpm_ctrl_t *const p_api_ctrl)

Enter low power mode (sleep/standby/deep standby) using WFI macro. Function will return after waking from low power mode.

**Implemented as**

- R_LPM_LowPowerModeEnter()

### ◆ ioKeepClear

fsp_err_t(* lpm_api_t::ioKeepClear) (lpm_ctrl_t *const p_api_ctrl)

Clear the IOKEEP bit after deep software standby.

- **Implemented as**

- R_LPM_IoKeepClear()

### ◆ versionGet

fsp_err_t(* lpm_api_t::versionGet) (fsp_version_t *const p_version)

Get the driver version based on compile time macros.

**Implemented as**

- R_LPM_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version used. |

### ◆ lpm_instance_t

struct lpm_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| lpm_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| lpm_cfg_t const *const | p_cfg | Pointer to the configuration structure for this instance. |
| lpm_api_t const *const | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ lpm_ctrl_t

typedef void lpm_ctrl_t

LPM control block. Allocate an instance specific control block to pass into the LPM API calls.

**Implemented as**

- lpm_instance_ctrl_t

## Enumeration Type Documentation

### ◆ lpm_mode_t

enum lpm_mode_t

Low power modes

| Enumerator | |
|---|---|
| LPM_MODE_SLEEP | Sleep mode. |
| LPM_MODE_STANDBY | Software Standby mode. |
| LPM_MODE_STANDBY_SNOOZE | Software Standby mode with Snooze mode enabled. |
| LPM_MODE_DEEP | Deep Software Standby mode. |

### ◆ lpm_snooze_request_t

enum lpm_snooze_request_t

Snooze request sources

| Enumerator | |
|---|---|
| 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_KEY | Enable KR snooze request. |
| LPM_SNOOZE_REQUEST_ACMPHS0 | Enable High-speed analog comparator 0 snooze request. |
| LPM_SNOOZE_REQUEST_RTC_ALARM | Enable RTC alarm snooze request. |
| 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. |

### ◆ lpm_snooze_end_t

| enum lpm_snooze_end_t | |
|---|---|
| Snooze end control | |
| Enumerator | |
| LPM_SNOOZE_END_STANDBY_WAKE_SOURCES | 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 Channel 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 mismatch. |

### ◆ lpm_snooze_cancel_t

| enum lpm_snooze_cancel_t | |
|---|---|
| Snooze cancel control | |
| Enumerator | |
| LPM_SNOOZE_CANCEL_SOURCE_NONE | No snooze cancel source. |
| LPM_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM | ADC Channel 0 window compare match. |
| LPM_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM | ADC Channel 0 window compare mismatch. |
| LPM_SNOOZE_CANCEL_SOURCE_SCI0_AM | SCI0 address match event. |
| LPM_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI | SCI0 receive error. |
| LPM_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE | DTC transfer completion. |
| LPM_SNOOZE_CANCEL_SOURCE_DOC_DOPCI | Data operation circuit interrupt. |
| LPM_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN | CTSU measurement end interrupt. |

### ◆ lpm_snooze_dtc_t

| enum lpm_snooze_dtc_t | |
|---|---|
| DTC Enable in Snooze Mode | |
| Enumerator | |
| LPM_SNOOZE_DTC_DISABLE | Disable DTC operation. |
| LPM_SNOOZE_DTC_ENABLE | Enable DTC operation. |

### ◆ lpm_standby_wake_source_t

| enum lpm_standby_wake_source_t | |
|---|---|
| Wake from standby mode sources, does not apply to sleep or deep standby modes | |
| Enumerator | |
| LPM_STANDBY_WAKE_SOURCE_IRQ0 | IRQ0. |
| LPM_STANDBY_WAKE_SOURCE_IRQ1 | IRQ1. |
| LPM_STANDBY_WAKE_SOURCE_IRQ2 | IRQ2. |
| LPM_STANDBY_WAKE_SOURCE_IRQ3 | IRQ3. |
| LPM_STANDBY_WAKE_SOURCE_IRQ4 | IRQ4. |
| LPM_STANDBY_WAKE_SOURCE_IRQ5 | IRQ5. |
| LPM_STANDBY_WAKE_SOURCE_IRQ6 | IRQ6. |
| LPM_STANDBY_WAKE_SOURCE_IRQ7 | IRQ7. |
| LPM_STANDBY_WAKE_SOURCE_IRQ8 | IRQ8. |
| LPM_STANDBY_WAKE_SOURCE_IRQ9 | IRQ9. |
| LPM_STANDBY_WAKE_SOURCE_IRQ10 | IRQ10. |
| LPM_STANDBY_WAKE_SOURCE_IRQ11 | IRQ11. |
| LPM_STANDBY_WAKE_SOURCE_IRQ12 | IRQ12. |
| LPM_STANDBY_WAKE_SOURCE_IRQ13 | IRQ13. |
| LPM_STANDBY_WAKE_SOURCE_IRQ14 | IRQ14. |
| LPM_STANDBY_WAKE_SOURCE_IRQ15 | IRQ15. |
| LPM_STANDBY_WAKE_SOURCE_IWDT | Independent watchdog interrupt. |
| LPM_STANDBY_WAKE_SOURCE_KEY | Key interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD1 | Low Voltage Detection 1 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD2 | Low Voltage Detection 2 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_VBATT | VBATT Monitor interrupt. |

| | |
|---|---|
| LPM_STANDBY_WAKE_SOURCE_ACMPHS0 | Analog Comparator High-speed 0 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ACMPLP0 | Analog Comparator Low-speed 0 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTCALM | RTC Alarm interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTCPRD | RTC Period interrupt. |
| LPM_STANDBY_WAKE_SOURCE_USBHS | USB High-speed interrupt. |
| LPM_STANDBY_WAKE_SOURCE_USBFS | USB Full-speed interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1UD | AGT1 underflow interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1CA | AGT1 compare match A interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1CB | AGT1 compare match B interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IIC0 | I2C 0 interrupt. |

### ◆ lpm_io_port_t

| enum lpm_io_port_t | |
|---|---|
| I/O port state after Deep Software Standby mode | |
| Enumerator | |
| 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 |

### ◆ lpm_power_supply_t

| enum lpm_power_supply_t | |
|---|---|
| Power supply control | |
| Enumerator | |
| 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 |
| 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 |

### ◆ lpm_deep_standby_cancel_edge_t

| enum lpm_deep_standby_cancel_edge_t | |
|---|---|
| Deep Standby Interrupt Edge | |
| Enumerator | |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_EDGE_NONE | No options for a deep standby cancel source. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_RISING | IRQ0-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_FALLING | IRQ0-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_RISING | IRQ1-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_FALLING | IRQ1-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_RISING | IRQ2-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_FALLING | IRQ2-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_RISING | IRQ3-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_FALLING | IRQ3-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_RISING | IRQ4-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_FALLING | IRQ4-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_RISING | IRQ5-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_FALLING | IRQ5-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_RISING | IRQ6-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_FALLING | IRQ6-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_RISING | IRQ7-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_FALLING | IRQ7-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_RISING | IRQ8-DS Pin Rising Edge. |

| | |
|---|---|
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_FALLING | IRQ8-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_RISING | IRQ9-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_FALLING | IRQ9-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_RISING | IRQ10-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_FALLING | IRQ10-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_RISING | IRQ11-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_FALLING | IRQ11-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_RISING | IRQ12-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_FALLING | IRQ12-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_RISING | IRQ13-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_FALLING | IRQ13-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_RISING | IRQ14-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_FALLING | IRQ14-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD1_RISING | LVD1 Rising Slope. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD1_FALLING | LVD1 Falling Slope. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD2_RISING | LVD2 Rising Slope. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD2_FALLING | LVD2 Falling Slope. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_NMI_RISING | NMI Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_NMI_FALLING | NMI Pin Falling Edge. |

### ◆ lpm_deep_standby_cancel_source_t

| enum lpm_deep_standby_cancel_source_t | |
|---|---|
| Deep Standby cancel sources | |
| Enumerator | |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_RESET_ONLY | Cancel deep standby only by reset. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ0 | IRQ0. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ1 | IRQ1. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ2 | IRQ2. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ3 | IRQ3. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ4 | IRQ4. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ5 | IRQ5. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ6 | IRQ6. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ7 | IRQ7. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ8 | IRQ8. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ9 | IRQ9. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ10 | IRQ10. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ11 | IRQ11. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ12 | IRQ12. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ13 | IRQ13. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ14 | IRQ14. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD1 | LVD1. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD2 | LVD2. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_RTC_INTERVAL | RTC Interval Interrupt. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_RTC_ALARM | RTC Alarm Interrupt. |

| | |
|---|---|
| LPM_DEEP_STANDBY_CANCEL_SOURCE_NMI | NMI. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_USBFS | USBFS Suspend/Resume. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_USBHS | USBHS Suspend/Resume. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_AGT1 | AGT1 Underflow. |

### ◆ lpm_output_port_enable_t

| enum lpm_output_port_enable_t | |
|---|---|
| Output port enable | |
| Enumerator | |
| 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. |

## 4.3.24 Low Voltage Detection Interface

Interfaces

### Detailed Description

Interface for Low Voltage Detection.

# Summary

The LVD driver provides functions for configuring the LVD voltage monitors and detectors.

Implemented by:

- Low Voltage Detection (r_lvd)

### Data Structures

| | |
|---|---|
| struct | lvd_status_t |
| struct | lvd_callback_args_t |
| struct | lvd_cfg_t |
| struct | lvd_api_t |
| struct | lvd_instance_t |

### Macros

| | |
|---|---|
| #define | LVD_API_VERSION_MAJOR |

### Typedefs

| | |
|---|---|
| typedef void | lvd_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | lvd_threshold_t |
| enum | lvd_response_t |
| enum | lvd_voltage_slope_t |
| enum | lvd_sample_clock_t |
| enum | lvd_negation_delay_t |
| enum | lvd_threshold_crossing_t |
| enum | lvd_current_state_t |

## Data Structure Documentation

### ◆ lvd_status_t

| struct lvd_status_t | | |
|---|---|---|
| Current state of a voltage monitor. | | |
| Data Fields | | |
| 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) |

### ◆ lvd_callback_args_t

| struct lvd_callback_args_t |
|---|

LVD callback parameter definition

| Data Fields | | |
|---|---|---|
| uint32_t | monitor_number | Monitor number. |
| lvd_current_state_t | current_state | Current state of the voltage monitor. |
| void const * | p_context | Placeholder for user data. |

### ◆ lvd_cfg_t

struct lvd_cfg_t

LVD configuration structure

**Data Fields**

| | |
|---|---|
| uint32_t | monitor_number |
| lvd_threshold_t | voltage_threshold |
| lvd_response_t | detection_response |
| lvd_voltage_slope_t | voltage_slope |
| lvd_negation_delay_t | negation_delay |
| lvd_sample_clock_t | sample_clock_divisor |
| IRQn_Type | irq |
| uint8_t | monitor_ipl |
| void(* | p_callback )(lvd_callback_args_t *p_args) |
| void const * | p_context |
| void const * | p_extend |

## Field Documentation

### ◆ monitor_number

uint32_t lvd_cfg_t::monitor_number

Monitor number, 1, 2, ...

### ◆ voltage_threshold

lvd_threshold_t lvd_cfg_t::voltage_threshold

Threshold for out of range voltage detection

### ◆ detection_response

lvd_response_t lvd_cfg_t::detection_response

Response on detecting a threshold crossing

### ◆ voltage_slope

lvd_voltage_slope_t lvd_cfg_t::voltage_slope

Direction of voltage crossing that will trigger a detection (Rising Edge, Falling Edge, Both).

### ◆ negation_delay

lvd_negation_delay_t lvd_cfg_t::negation_delay

Negation of LVD signal follows reset or voltage in range

### ◆ sample_clock_divisor

lvd_sample_clock_t lvd_cfg_t::sample_clock_divisor

Sample clock divider, use LVD_SAMPLE_CLOCK_DISABLED to disable digital filtering

### ◆ irq

IRQn_Type lvd_cfg_t::irq

Interrupt number.

### ◆ monitor_ipl

uint8_t lvd_cfg_t::monitor_ipl

Interrupt priority level.

### ◆ p_callback

void(* lvd_cfg_t::p_callback) (lvd_callback_args_t *p_args)

User function to be called from interrupt

### ◆ p_context

void const* lvd_cfg_t::p_context

Placeholder for user data. Passed to the user callback in

### ◆ p_extend

| void const* lvd_cfg_t::p_extend |
|---|
| Extension parameter for hardware specific settings |

### ◆ lvd_api_t

| struct lvd_api_t | |
|---|---|
| LVD driver API structure. LVD driver functions implemented at the HAL layer will adhere to this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(lvd_ctrl_t *const p_ctrl, lvd_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | statusGet )(lvd_ctrl_t *const p_ctrl, lvd_status_t *p_lvd_status) |
| | |
| fsp_err_t(* | statusClear )(lvd_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | close )(lvd_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* lvd_api_t::open) (lvd_ctrl_t *const p_ctrl, lvd_cfg_t const *const p_cfg)

Initializes a low voltage detection driver according to the passed-in configuration structure.

**Implemented as**

- R_LVD_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure for the driver instance |
| [in] | p_cfg | Pointer to the configuration structure for the driver instance |

### ◆ statusGet

fsp_err_t(* lvd_api_t::statusGet) (lvd_ctrl_t *const p_ctrl, lvd_status_t *p_lvd_status)

Get the current state of the monitor, (threshold crossing detected, voltage currently above or below threshold). Must be used if the peripheral was initialized with lvd_response_t set to LVD_RESPONSE_NONE.

**Implemented as**

- R_LVD_StatusGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the control structure for the driver instance |
| [in,out] | p_lvd_status | Pointer to a lvd_status_t structure |

### ◆ statusClear

fsp_err_t(* lvd_api_t::statusClear) (lvd_ctrl_t *const 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.

**Implemented as**

- R_LVD_StatusClear()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the control structure for the driver instance |

### ◆ close

fsp_err_t(* lvd_api_t::close) (lvd_ctrl_t *const p_ctrl)

Disables the LVD peripheral. Closes the driver instance.

**Implemented as**

- R_LVD_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the control structure for the driver instance |

### ◆ versionGet

fsp_err_t(* lvd_api_t::versionGet) (fsp_version_t *const p_version)

Returns the LVD driver version based on compile time macros.

**Implemented as**

- R_LVD_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_version | Pointer to version structure |

### ◆ lvd_instance_t

struct lvd_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| 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. |

## Macro Definition Documentation

### ◆ LVD_API_VERSION_MAJOR

#define LVD_API_VERSION_MAJOR

Register definitions, common services, and error codes.

## Typedef Documentation

### ◆ lvd_ctrl_t

typedef void lvd_ctrl_t

LVD control block. Allocate an instance specific control block to pass into the LVD API calls.

**Implemented as**

- lvd_instance_ctrl_t

## Enumeration Type Documentation

### ◆ lvd_threshold_t

| enum lvd_threshold_t | |
|---|---|
| Voltage detection level The thresholds supported by each MCU are in the MCU User's Manual as well as in the r_lvd module description on the stack tab of the RA project. | |
| Enumerator | |
| LVD_THRESHOLD_MONITOR_1_LEVEL_4_29V | 4.29V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_4_14V | 4.14V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_4_02V | 4.02V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_84V | 3.84V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_10V | 3.10V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_00V | 3.00V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_90V | 2.90V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_79V | 2.79V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_68V | 2.68V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_58V | 2.58V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_48V | 2.48V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_20V | 2.20V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_96V | 1.96V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_86V | 1.86V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_75V | 1.75V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_65V | 1.65V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_99V | 2.99V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_92V | 2.92V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_85V | 2.85V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_4_29V | 4.29V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_4_14V | 4.14V |

| | |
|---|---|
| LVD_THRESHOLD_MONITOR_2_LEVEL_4_02V | 4.02V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_3_84V | 3.84V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_2_99V | 2.99V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_2_92V | 2.92V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_2_85V | 2.85V |

### ◆ lvd_response_t

| enum lvd_response_t | |
|---|---|
| Response types for handling threshold crossing event. | |
| Enumerator | |
| 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. |

### ◆ lvd_voltage_slope_t

| enum lvd_voltage_slope_t | |
|---|---|
| The direction from which Vcc must cross the threshold to trigger a detection (rising, falling, or both). | |
| Enumerator | |
| 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. |

### ◆ lvd_sample_clock_t

| enum lvd_sample_clock_t | |
|---|---|
| Sample clock divider, use LVD_SAMPLE_CLOCK_DISABLED to disable digital filtering | |
| Enumerator | |
| 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. |

### ◆ lvd_negation_delay_t

| enum lvd_negation_delay_t | |
|---|---|
| Negation delay of LVD reset signal follows reset or voltage in range | |
| Enumerator | |
| 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 |

### ◆ lvd_threshold_crossing_t

| enum lvd_threshold_crossing_t | |
|---|---|
| Threshold crossing detection (latched) | |
| Enumerator | |
| LVD_THRESHOLD_CROSSING_NOT_DETECTED | Threshold crossing has not been detected. |
| LVD_THRESHOLD_CROSSING_DETECTED | Threshold crossing has been detected. |

### ◆ lvd_current_state_t

| enum lvd_current_state_t | |
|---|---|
| Instantaneous status of VCC (above or below threshold) | |
| Enumerator | |
| LVD_CURRENT_STATE_BELOW_THRESHOLD | VCC < threshold. |
| LVD_CURRENT_STATE_ABOVE_THRESHOLD | VCC >= threshold or monitor is disabled. |

## 4.3.25 OPAMP Interface

Interfaces

### Detailed Description

Interface for Operational Amplifiers.

# Summary

The OPAMP interface provides standard operational amplifier functionality, including starting and stopping the amplifier.

Implemented by: Operational Amplifier (r_opamp)

### Data Structures

| | |
|---|---|
| struct | opamp_trim_args_t |
| struct | opamp_info_t |
| struct | opamp_status_t |
| struct | opamp_cfg_t |
| struct | opamp_api_t |
| struct | opamp_instance_t |

### Macros

| | |
|---|---|
| #define | OPAMP_API_VERSION_MAJOR |

### Typedefs

| | |
|---|---|
| typedef void | opamp_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | opamp_trim_cmd_t |
| enum | opamp_trim_input_t |

## Data Structure Documentation

### ◆ opamp_trim_args_t

| struct opamp_trim_args_t | | |
|---|---|---|
| OPAMP trim arguments. | | |
| Data Fields | | |
| uint8_t | channel | Channel. |
| opamp_trim_input_t | input | Which input of the channel above. |

### ◆ opamp_info_t

| struct opamp_info_t | | |
|---|---|---|
| OPAMP information. | | |
| Data Fields | | |
| uint32_t | min_stabilization_wait_us | Minimum stabilization wait time in microseconds. |

### ◆ opamp_status_t

| struct opamp_status_t | | |
|---|---|---|
| OPAMP status. | | |
| Data Fields | | |
| uint32_t | operating_channel_mask | Bitmask of channels currently operating. |

### ◆ opamp_cfg_t

| struct opamp_cfg_t | | |
|---|---|---|
| OPAMP general configuration. | | |
| Data Fields | | |
| void const * | p_extend | Extension parameter for hardware specific settings. |

### ◆ opamp_api_t

| struct opamp_api_t |
|---|
| OPAMP functions implemented at the HAL layer will follow this API. |

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(opamp_ctrl_t *const p_ctrl, opamp_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | start )(opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask) |
| | |
| fsp_err_t(* | stop )(opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask) |
| | |
| fsp_err_t(* | trim )(opamp_ctrl_t *const p_ctrl, opamp_trim_cmd_t const cmd, opamp_trim_args_t const *const p_args) |
| | |
| fsp_err_t(* | infoGet )(opamp_ctrl_t *const p_ctrl, opamp_info_t *const p_info) |
| | |
| fsp_err_t(* | statusGet )(opamp_ctrl_t *const p_ctrl, opamp_status_t *const p_status) |
| | |
| fsp_err_t(* | close )(opamp_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* opamp_api_t::open) (opamp_ctrl_t *const p_ctrl, opamp_cfg_t const *const p_cfg)

Initialize the operational amplifier.

**Implemented as**

- R_OPAMP_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | p_cfg | Pointer to configuration |

### ◆ start

fsp_err_t(* opamp_api_t::start) (opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask)

Start the op-amp(s).

**Implemented as**

- R_OPAMP_Start()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | channel_mask | Bitmask of channels to start |

### ◆ stop

fsp_err_t(* opamp_api_t::stop) (opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask)

Stop the op-amp(s).

**Implemented as**

- R_OPAMP_Stop()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | channel_mask | Bitmask of channels to stop |

### ◆ trim

fsp_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)

Trim the op-amp(s). Not supported on all MCUs. See implementation for procedure details.

**Implemented as**

- R_OPAMP_Trim()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | cmd | Trim command |
| [in] | p_args | Pointer to arguments for the command |

### ◆ infoGet

fsp_err_t(* opamp_api_t::infoGet) (opamp_ctrl_t *const p_ctrl, opamp_info_t *const p_info)

Provide information such as the recommended minimum stabilization wait time.

**Implemented as**

- R_OPAMP_InfoGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [out] | p_info | OPAMP information stored here |

### ◆ statusGet

fsp_err_t(* opamp_api_t::statusGet) (opamp_ctrl_t *const p_ctrl, opamp_status_t *const p_status)

Provide status of each op-amp channel.

**Implemented as**

- R_OPAMP_StatusGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [out] | p_status | Status stored here |

### ◆ close

fsp_err_t(* opamp_api_t::close) (opamp_ctrl_t *const p_ctrl)

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()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |

### ◆ versionGet

fsp_err_t(* opamp_api_t::versionGet) (fsp_version_t *const p_version)

Retrieve the API version.

**Implemented as**

- R_OPAMP_VersionGet()

**Precondition**

This function retrieves the API version.

**Parameters**

| | | |
|---|---|---|
| [in] | p_version | Pointer to version structure |

### ◆ opamp_instance_t

struct opamp_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| 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. |

## Macro Definition Documentation

### ◆ OPAMP_API_VERSION_MAJOR

#define OPAMP_API_VERSION_MAJOR

Includes board and MCU related header files. Version Number of API.

## Typedef Documentation

### ◆ opamp_ctrl_t

typedef void opamp_ctrl_t

OPAMP control block. Allocate using driver instance control structure from driver instance header file.

## Enumeration Type Documentation

### ◆ opamp_trim_cmd_t

| enum opamp_trim_cmd_t | |
|---|---|
| Trim command. | |
| Enumerator | |
| 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. |

### ◆ opamp_trim_input_t

| enum opamp_trim_input_t | |
|---|---|
| Trim input. | |
| Enumerator | |
| OPAMP_TRIM_INPUT_PCH | Trim non-inverting (+) input. |
| OPAMP_TRIM_INPUT_NCH | Trim inverting (-) input. |

## 4.3.26 POEG Interface

Interfaces

### Detailed Description

Interface for the Port Output Enable for GPT.

Defines the API and data structures for the Port Output Enable for GPT (POEG) interface.

# Summary

The POEG disables GPT output pins based on configurable events.

Implemented by: Port Output Enable for GPT (r_poeg)

### Data Structures

| | |
|---|---|
| struct | poeg_status_t |
| struct | poeg_callback_args_t |

| | |
|---|---|
| struct | poeg_cfg_t |
| struct | poeg_api_t |
| struct | poeg_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | poeg_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | poeg_state_t |
| enum | poeg_trigger_t |
| enum | poeg_gtetrg_polarity_t |
| enum | poeg_gtetrg_noise_filter_t |

## Data Structure Documentation

### ◆ poeg_status_t

| struct poeg_status_t | | |
|---|---|---|
| POEG status | | |
| Data Fields | | |
| poeg_state_t | state | Current state of POEG. |

### ◆ poeg_callback_args_t

| struct poeg_callback_args_t | | |
|---|---|---|
| Callback function parameter data. | | |
| Data Fields | | |
| void const * | p_context | Placeholder for user data, set in poeg_cfg_t. |

### ◆ poeg_cfg_t

| struct poeg_cfg_t | |
|---|---|
| User configuration structure, used in the open function. | |
| **Data Fields** | |
| poeg_trigger_t | trigger |
| | Select one or more triggers for the POEG. |
| | |

| | |
|---|---|
| poeg_gtetrg_polarity_t | polarity |
| | Select the polarity for the GTETRG pin. |
| | |
| poeg_gtetrg_noise_filter_t | noise_filter |
| | Configure the GTETRG noise filter. |
| | |
| void(* | p_callback )(poeg_callback_args_t *p_args) |
| | |
| void const * | p_context |
| | |
| uint32_t | channel |
| | Channel 0 corresponds to GTETRGA, 1 to GTETRGB, etc. |
| | |
| IRQn_Type | irq |
| | NVIC interrupt number assigned to this instance. |
| | |
| uint8_t | ipl |
| | POEG interrupt priority. |
| | |

## Field Documentation

### ◆ p_callback

void(* poeg_cfg_t::p_callback) (poeg_callback_args_t *p_args)

Callback called when a POEG interrupt occurs.

### ◆ p_context

void const* poeg_cfg_t::p_context

Placeholder for user data. Passed to the user callback in poeg_callback_args_t.

### ◆ poeg_api_t

struct poeg_api_t

Port Output Enable for GPT (POEG) API structure. POEG functions implemented at the HAL layer will follow this API.

| Data Fields | |
|---|---|
| fsp_err_t(* | open )(poeg_ctrl_t *const p_ctrl, poeg_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | statusGet )(poeg_ctrl_t *const p_ctrl, poeg_status_t *p_status) |
| | |
| fsp_err_t(* | outputDisable )(poeg_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | reset )(poeg_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | close )(poeg_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* poeg_api_t::open) (poeg_ctrl_t *const p_ctrl, poeg_cfg_t const *const p_cfg)

Initial configuration.

**Implemented as**

- R_POEG_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ statusGet

fsp_err_t(* poeg_api_t::statusGet) (poeg_ctrl_t *const p_ctrl, poeg_status_t *p_status)

Gets the current driver state.

**Implemented as**

- R_POEG_StatusGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in poeg_api_t::open call. |
| [out] | p_status | Provides the current state of the POEG. |

### ◆ outputDisable

fsp_err_t(* poeg_api_t::outputDisable) (poeg_ctrl_t *const p_ctrl)

Disables GPT output pins by software request.

**Implemented as**

- R_POEG_OutputDisable()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in poeg_api_t::open call. |

### ◆ reset

fsp_err_t(* poeg_api_t::reset) (poeg_ctrl_t *const p_ctrl)

Attempts to clear status flags to reenable GPT output pins. Confirm all status flags are cleared after calling this function by calling poeg_api_t::statusGet().

**Implemented as**

- R_POEG_Reset()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in poeg_api_t::open call. |

### ◆ close

fsp_err_t(* poeg_api_t::close) (poeg_ctrl_t *const p_ctrl)

Disables POEG interrupt.

**Implemented as**

- R_POEG_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in poeg_api_t::open call. |

### ◆ versionGet

fsp_err_t(* poeg_api_t::versionGet) (fsp_version_t *const p_version)

Get version and stores it in provided pointer p_version.

**Implemented as**

- R_POEG_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version used. |

### ◆ poeg_instance_t

struct poeg_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| poeg_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| poeg_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| poeg_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ poeg_ctrl_t

typedef void poeg_ctrl_t

DOC control block. Allocate an instance specific control block to pass into the DOC API calls.

**Implemented as**

- poeg_instance_ctrl_t

## Enumeration Type Documentation

### ◆ poeg_state_t

| enum poeg_state_t | |
|---|---|
| POEG states. | |
| Enumerator | |
| POEG_STATE_NO_DISABLE_REQUEST | GPT output is not disabled by POEG. |
| POEG_STATE_PIN_DISABLE_REQUEST | GPT output disabled due to GTETRG pin level. |
| POEG_STATE_GPT_OR_COMPARATOR_DISABLE_REQUEST | GPT output disabled due to high speed analog comparator or GPT. |
| POEG_STATE_OSCILLATION_STOP_DISABLE_REQUEST | GPT output disabled due to main oscillator stop. |
| POEG_STATE_SOFTWARE_STOP_DISABLE_REQUEST | GPT output disabled due to poeg_api_t::outputDisable() |
| POEG_STATE_PIN_DISABLE_REQUEST_ACTIVE | GPT output disable request active from the GTETRG pin. If a filter is used, this flag represents the state of the filtered input. |

### ◆ poeg_trigger_t

| enum poeg_trigger_t | |
|---|---|
| Triggers that will disable GPT output pins. | |
| Enumerator | |
| POEG_TRIGGER_SOFTWARE | Software disable is always supported with POEG. Select this option if no other triggers are used. |
| POEG_TRIGGER_PIN | Disable GPT output based on GTETRG input level. |
| POEG_TRIGGER_GPT_OUTPUT_LEVEL | Disable GPT output based on GPT output pin levels. |
| POEG_TRIGGER_OSCILLATION_STOP | Disable GPT output based on main oscillator stop. |
| POEG_TRIGGER_ACMPHS0 | Disable GPT output based on ACMPHS0 comparator result. |
| POEG_TRIGGER_ACMPHS1 | Disable GPT output based on ACMPHS1 comparator result. |
| POEG_TRIGGER_ACMPHS2 | Disable GPT output based on ACMPHS2 comparator result. |
| POEG_TRIGGER_ACMPHS3 | Disable GPT output based on ACMPHS3 comparator result. |
| POEG_TRIGGER_ACMPHS4 | Disable GPT output based on ACMPHS4 comparator result. |
| POEG_TRIGGER_ACMPHS5 | Disable GPT output based on ACMPHS5 comparator result. |

### ◆ poeg_gtetrg_polarity_t

| enum poeg_gtetrg_polarity_t | |
|---|---|
| GTETRG polarity. | |
| Enumerator | |
| POEG_GTETRG_POLARITY_ACTIVE_HIGH | Disable GPT output based when GTETRG input level is high. |
| POEG_GTETRG_POLARITY_ACTIVE_LOW | Disable GPT output based when GTETRG input level is low. |

### ◆ poeg_gtetrg_noise_filter_t

| enum poeg_gtetrg_noise_filter_t | |
|---|---|
| GTETRG noise filter. For the input signal to pass through the noise filter, the active level set in poeg_gtetrg_polarity_t must be read 3 consecutive times at the sampling clock selected. | |
| Enumerator | |
| POEG_GTETRG_NOISE_FILTER_DISABLED | No noise filter applied to GTETRG input. |
| POEG_GTETRG_NOISE_FILTER_PCLKB_DIV_1 | Apply noise filter with sample clock PCLKB. |
| POEG_GTETRG_NOISE_FILTER_PCLKB_DIV_8 | Apply noise filter with sample clock PCLKB/8. |
| POEG_GTETRG_NOISE_FILTER_PCLKB_DIV_32 | Apply noise filter with sample clock PCLKB/32. |
| POEG_GTETRG_NOISE_FILTER_PCLKB_DIV_128 | Apply noise filter with sample clock PCLKB/128. |

## 4.3.27 RTC Interface

Interfaces

### Detailed Description

Interface for accessing the Realtime Clock.

# Summary

The RTC Interface is for configuring Real Time Clock (RTC) functionality including alarm, periodic notiification and error adjustment.

The Real Time Clock Interface can be implemented by:

- Realtime Clock (r_rtc)

## Data Structures

| | |
|---|---|
| struct | rtc_callback_args_t |
| struct | rtc_error_adjustment_cfg_t |
| struct | rtc_alarm_time_t |
| struct | rtc_info_t |
| struct | rtc_cfg_t |
| struct | rtc_api_t |
| struct | rtc_instance_t |

## Typedefs

| | |
|---|---|
| typedef struct tm | rtc_time_t |
| typedef void | rtc_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | rtc_event_t |
| enum | rtc_clock_source_t |
| enum | rtc_status_t |
| enum | rtc_error_adjustment_t |
| enum | rtc_error_adjustment_mode_t |
| enum | rtc_error_adjustment_period_t |
| enum | rtc_periodic_irq_select_t |

## Data Structure Documentation

### ◆ rtc_callback_args_t

| struct rtc_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| | | |

| | | |
|---|---|---|
| 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. |

### ◆ rtc_error_adjustment_cfg_t

| struct rtc_error_adjustment_cfg_t | | |
|---|---|---|
| Time error adjustment value configuration | | |
| Data Fields | | |
| rtc_error_adjustment_mode_t | adjustment_mode | Automatic Adjustment Enable/Disable. |
| rtc_error_adjustment_period_t | adjustment_period | Error Adjustment period. |
| rtc_error_adjustment_t | adjustment_type | Time error adjustment setting. |
| uint32_t | adjustment_value | Value of the prescaler for error adjustment. |

### ◆ rtc_alarm_time_t

| struct rtc_alarm_time_t | | |
|---|---|---|
| Alarm time setting structure | | |
| Data Fields | | |
| 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. |

### ◆ rtc_info_t

| struct rtc_info_t |
|---|
| RTC Information Structure for information returned by infoGet() |
| Data Fields |

| | | |
|---|---|---|
| rtc_clock_source_t | clock_source | Clock source for the RTC block. |
| rtc_status_t | status | RTC run status. |

### ◆ rtc_cfg_t

| struct rtc_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| rtc_clock_source_t | clock_source |
| | Clock source for the RTC block. |
| | |
| uint32_t | freq_compare_value_loco |
| | The frequency comparison value for LOCO. |
| | |
| rtc_error_adjustment_cfg_t const *const | p_err_cfg |
| | Pointer to Error Adjustment configuration. |
| | |
| uint8_t | alarm_ipl |
| | Alarm interrupt priority. |
| | |
| IRQn_Type | alarm_irq |
| | Alarm interrupt vector. |
| | |
| uint8_t | periodic_ipl |
| | Periodic interrupt priority. |
| | |
| IRQn_Type | periodic_irq |
| | Periodic interrupt vector. |
| | |
| uint8_t | carry_ipl |
| | |

| | |
|---|---|
| | Carry interrupt priority. |
| | |
| IRQn_Type | carry_irq |
| | Carry interrupt vector. |
| | |
| void(* | p_callback )(rtc_callback_args_t *p_args) |
| | Called from the ISR. |
| | |
| void const * | p_context |
| | User defined context passed into callback function. |
| | |
| void const * | p_extend |
| | RTC hardware dependant configuration. |
| | |

### ◆ rtc_api_t

| struct rtc_api_t | |
|---|---|
| RTC driver structure. General RTC functions implemented at the HAL layer follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rtc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | calendarTimeSet )(rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time) |
| | |
| fsp_err_t(* | calendarTimeGet )(rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time) |
| | |
| fsp_err_t(* | calendarAlarmSet )(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm) |
| | |
| fsp_err_t(* | calendarAlarmGet )(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm) |

| | |
|---|---|
| fsp_err_t(* | periodicIrqRateSet )(rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t const rate) |
| fsp_err_t(* | errorAdjustmentSet )(rtc_ctrl_t *const p_ctrl, rtc_error_adjustment_cfg_t const *const err_adj_cfg) |
| fsp_err_t(* | infoGet )(rtc_ctrl_t *const p_ctrl, rtc_info_t *const p_rtc_info) |
| fsp_err_t(* | versionGet )(fsp_version_t *const version) |

## Field Documentation

### ◆ open

fsp_err_t(* rtc_api_t::open) (rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg)

Open the RTC driver.

**Implemented as**

- R_RTC_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to RTC device handle |
| [in] | p_cfg | Pointer to the configuration structure |

### ◆ close

fsp_err_t(* rtc_api_t::close) (rtc_ctrl_t *const p_ctrl)

Close the RTC driver.

**Implemented as**

- R_RTC_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to RTC device handle. |

### ◆ calendarTimeSet

fsp_err_t(* rtc_api_t::calendarTimeSet) (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)

Set the calendar time and start the calender counter

**Implemented as**

- R_RTC_CalendarTimeSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to RTC device handle |
| [in] | p_time | Pointer to a time structure that contains the time to set |
| [in] | clock_start | Flag that starts the clock right after it is set |

### ◆ calendarTimeGet

fsp_err_t(* rtc_api_t::calendarTimeGet) (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)

Get the calendar time.

**Implemented as**

- R_RTC_CalendarTimeGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to RTC device handle |
| [out] | p_time | Pointer to a time structure that contains the time to get |

### ◆ calendarAlarmSet

fsp_err_t(* rtc_api_t::calendarAlarmSet) (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)

Set the calendar alarm time and enable the alarm interrupt.

**Implemented as**

- R_RTC_CalendarAlarmSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to RTC device handle |
| [in] | p_alarm | Pointer to an alarm structure that contains the alarm time to set |
| [in] | irq_enable_flag | Enable the ALARM irq if set |

### ◆ calendarAlarmGet

fsp_err_t(* rtc_api_t::calendarAlarmGet) (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)

Get the calendar alarm time.

**Implemented as**

- R_RTC_CalendarAlarmGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to RTC device handle |
| [out] | p_alarm | Pointer to an alarm structure to fill up with the alarm time |

### ◆ periodicIrqRateSet

fsp_err_t(* rtc_api_t::periodicIrqRateSet) (rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t const rate)

Set the periodic irq rate

**Implemented as**

- R_RTC_PeriodicIrqRateSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to RTC device handle |
| [in] | rate | Rate of periodic interrupts |

### ◆ errorAdjustmentSet

fsp_err_t(* rtc_api_t::errorAdjustmentSet) (rtc_ctrl_t *const p_ctrl, rtc_error_adjustment_cfg_t const *const err_adj_cfg)

Set time error adjustment.

**Implemented as**

- R_RTC_ErrorAdjustmentSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | err_adj_cfg | Pointer to the Error Adjustment Config |

### ◆ infoGet

| fsp_err_t(* rtc_api_t::infoGet) (rtc_ctrl_t *const p_ctrl, rtc_info_t *const p_rtc_info) |
|---|

Return the currently configure clock source for the RTC

**Implemented as**

- R_RTC_InfoGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [out] | p_rtc_info | Pointer to RTC information structure |

### ◆ versionGet

| fsp_err_t(* rtc_api_t::versionGet) (fsp_version_t *const version) |
|---|

Gets version and stores it in provided pointer p_version.

**Implemented as**

- R_RTC_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version used |

### ◆ rtc_instance_t

| struct rtc_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| 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. |

## Typedef Documentation

### ◆ rtc_time_t

| typedef struct tm rtc_time_t |
|---|
| Date and time structure defined in C standard library <time.h> |

### ◆ rtc_ctrl_t

| typedef void rtc_ctrl_t |
|---|
| RTC control block. Allocate an instance specific control block to pass into the RTC API calls.<br><br>**Implemented as**<br><br>◦ rtc_instance_ctrl_t |

## Enumeration Type Documentation

### ◆ rtc_event_t

| enum rtc_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| RTC_EVENT_ALARM_IRQ | Real Time Clock ALARM IRQ. |
| RTC_EVENT_PERIODIC_IRQ | Real Time Clock PERIODIC IRQ. |

### ◆ rtc_clock_source_t

| enum rtc_clock_source_t | |
|---|---|
| Clock source for the RTC block | |
| Enumerator | |
| RTC_CLOCK_SOURCE_SUBCLK | Sub-clock oscillator. |
| RTC_CLOCK_SOURCE_LOCO | Low power On Chip Oscillator. |

### ◆ rtc_status_t

| enum rtc_status_t | |
|---|---|
| RTC run state | |
| Enumerator | |
| RTC_STATUS_STOPPED | RTC counter is stopped. |
| RTC_STATUS_RUNNING | RTC counter is running. |

### ◆ rtc_error_adjustment_t

| enum rtc_error_adjustment_t | |
|---|---|
| Time error adjustment settings | |
| Enumerator | |
| 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. |

### ◆ rtc_error_adjustment_mode_t

| enum rtc_error_adjustment_mode_t | |
|---|---|
| Time error adjustment mode settings | |
| Enumerator | |
| RTC_ERROR_ADJUSTMENT_MODE_MANUAL | Adjustment mode is set to manual. |
| RTC_ERROR_ADJUSTMENT_MODE_AUTOMATIC | Adjustment mode is set to automatic. |

### ◆ rtc_error_adjustment_period_t

| enum rtc_error_adjustment_period_t | |
|---|---|
| Time error adjustment period settings | |
| Enumerator | |
| 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. |

### ◆ rtc_periodic_irq_select_t

| enum rtc_periodic_irq_select_t | |
|---|---|
| Periodic Interrupt select | |
| Enumerator | |
| 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_SECOND | A periodic irq is generated every 2 seconds. |

## 4.3.28 SD/MMC Interface

Interfaces

### Detailed Description

Interface for accessing SD, eMMC, and SDIO devices.

# Summary

The r_sdhi interface provides standard SD and eMMC media functionality. This interface also supports SDIO.

The SD/MMC interface is implemented by:

- SD/MMC Host Interface (r_sdhi)

## Data Structures

| | |
|---|---|
| struct | sdmmc_status_t |
| struct | sdmmc_device_t |
| struct | sdmmc_callback_args_t |
| struct | sdmmc_cfg_t |
| struct | sdmmc_api_t |
| struct | sdmmc_instance_t |

## Typedefs

| | |
|---|---|
| typedef void | sdmmc_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | sdmmc_card_type_t |
| enum | sdmmc_bus_width_t |
| enum | sdmmc_io_transfer_mode_t |
| enum | sdmmc_io_address_mode_t |
| enum | sdmmc_io_write_mode_t |
| enum | sdmmc_event_t |
| enum | sdmmc_card_detect_t |
| enum | sdmmc_write_protect_t |
| enum | sdmmc_r1_state_t |

## Data Structure Documentation

### ◆ sdmmc_status_t

| struct sdmmc_status_t | | |
|---|---|---|
| Current status. | | |
| Data Fields | | |
| bool | initialized | False if card was removed (only applies if MCU supports card |

| | | |
|---|---|---|
| | | detection and SDnCD pin is connected), true otherwise.<br><br>If ready is false, call sdmmc_api_t::mediaInit to reinitialize it |
| bool | transfer_in_progress | true = Card is busy |
| bool | card_inserted | Card detect status, true if card detect is not used. |

### ◆ sdmmc_device_t

| struct sdmmc_device_t | | |
|---|---|---|
| Information obtained from the media device. | | |
| Data Fields | | |
| sdmmc_card_type_t | card_type | SD, eMMC, or SDIO. |
| bool | write_protected | true = Card is write protected |
| uint32_t | clock_rate | Current clock rate. |
| uint32_t | sector_count | Sector count. |
| uint32_t | sector_size_bytes | Sector size. |
| uint32_t | erase_sector_count | Minimum erasable unit (in 512 byte sectors) |

### ◆ sdmmc_callback_args_t

| struct sdmmc_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| sdmmc_event_t | event | The event can be used to identify what caused the callback. |
| sdmmc_response_t | response | Response from card, only valid if SDMMC_EVENT_RESPONSE is set in event. |
| void const * | p_context | Placeholder for user data. |

### ◆ sdmmc_cfg_t

| struct sdmmc_cfg_t | |
|---|---|
| SD/MMC Configuration | |
| **Data Fields** | |
| uint8_t | channel |
| | Channel of SD/MMC host interface. |

| | |
|---|---|
| | |
| sdmmc_bus_width_t | bus_width |
| | Device bus width is 1, 4 or 8 bits wide. |
| | |
| 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. |
| | |
| uint32_t | block_size |
| | |
| sdmmc_card_detect_t | card_detect |
| | |
| sdmmc_write_protect_t | write_protect |
| | |
| IRQn_Type | access_irq |
| | Access IRQ number. |
| | |
| IRQn_Type | sdio_irq |
| | SDIO IRQ number. |
| | |
| IRQn_Type | card_irq |

| | |
|---|---|
| | Card IRQ number. |
| | |
| IRQn_Type | dma_req_irq |
| | DMA request IRQ number. |
| | |
| 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. |
| | |

## Field Documentation

### ◆ block_size

uint32_t sdmmc_cfg_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.

### ◆ card_detect

sdmmc_card_detect_t sdmmc_cfg_t::card_detect

Whether or not card detection is used.

### ◆ write_protect

sdmmc_write_protect_t sdmmc_cfg_t::write_protect

Select whether or not to use the write protect pin. Select Not Used if the MCU or device does not have a write protect pin.

### ◆ sdmmc_api_t

| struct sdmmc_api_t | |
|---|---|
| SD/MMC functions implemented at the HAL layer API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | mediaInit )(sdmmc_ctrl_t *const p_ctrl, sdmmc_device_t *const p_device) |
| | |
| fsp_err_t(* | read )(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count) |
| | |
| fsp_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) |
| | |
| fsp_err_t(* | readIo )(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address) |
| | |
| fsp_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) |
| | |
| fsp_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) |
| | |
| fsp_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) |
| | |
| fsp_err_t(* | ioIntEnable )(sdmmc_ctrl_t *const p_ctrl, bool enable) |
| | |
| fsp_err_t(* | statusGet )(sdmmc_ctrl_t *const p_ctrl, sdmmc_status_t *const p_status) |
| | |
| fsp_err_t(* | erase )(sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector, uint32_t const sector_count) |
| | |

| | |
|---|---|
| fsp_err_t(* | close )(sdmmc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* sdmmc_api_t::open) (sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const *const p_cfg)

Open the SD/MMC driver.

**Implemented as**

- R_SDHI_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to SD/MMC instance control block. |
| [in] | p_cfg | Pointer to SD/MMC instance configuration structure. |

### ◆ mediaInit

fsp_err_t(* sdmmc_api_t::mediaInit) (sdmmc_ctrl_t *const p_ctrl, sdmmc_device_t *const p_device)

Initializes 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 as**

- R_SDHI_MediaInit()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to SD/MMC instance control block. |
| [out] | p_device | Pointer to store device information. |

### ◆ read

fsp_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)

Read data from an SD/MMC channel. This API is not supported for SDIO devices.

**Implemented as**

- R_SDHI_Read()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [out] | p_dest | Pointer to data buffer to read data to. |
| [in] | start_sector | First sector address to read. |
| [in] | sector_count | Number of sectors to read. All sectors must be in the range of sdmmc_device_t::sector_count. |

### ◆ write

fsp_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)

Write data to SD/MMC channel. This API is not supported for SDIO devices.

**Implemented as**

- R_SDHI_Write()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [in] | p_source | Pointer to data buffer to write data from. |
| [in] | start_sector | First sector address to write to. |
| [in] | sector_count | Number of sectors to write. All sectors must be in the range of sdmmc_device_t::sector_count. |

### ◆ readIo

fsp_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)

Read one byte of I/O data from an SDIO device. This API is not supported for SD or eMMC memory devices.

**Implemented as**

- R_SDHI_ReadIo()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [out] | p_data | Pointer to location to store data byte. |
| [in] | function | SDIO Function Number. |
| [in] | address | SDIO register address. |

### ◆ writeIo

fsp_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)

Write one byte of I/O data to an SDIO device. This API is not supported for SD or eMMC memory devices.

**Implemented as**

- R_SDHI_WriteIo()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [in,out] | p_data | Pointer to data byte to write. Read data is also provided here if read_after_write is true. |
| [in] | function | SDIO Function Number. |
| [in] | address | SDIO register address. |
| [in] | read_after_write | Whether or not to read back the same register after writing |

### ◆ readIoExt

fsp_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)

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 as**

- R_SDHI_ReadIoExt()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [out] | p_dest | Pointer to data buffer to read data to. |
| [in] | function | SDIO Function Number. |
| [in] | address | SDIO register address. |
| [in] | count | Number of bytes or blocks to read, maximum 512 bytes or 511 blocks. |
| [in] | transfer_mode | Byte or block mode |
| [in] | address_mode | Fixed or incrementing address mode |

### ◆ writeIoExt

fsp_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)

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 as**

- R_SDHI_WriteIoExt()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [in] | p_source | Pointer to data buffer to write data from. |
| [in] | function_number | SDIO Function Number. |
| [in] | address | SDIO register address. |
| [in] | count | Number of bytes or blocks to write, maximum 512 bytes or 511 blocks. |
| [in] | transfer_mode | Byte or block mode |
| [in] | address_mode | Fixed or incrementing address mode |

### ◆ ioIntEnable

fsp_err_t(* sdmmc_api_t::ioIntEnable) (sdmmc_ctrl_t *const p_ctrl, bool enable)

Enables SDIO interrupt for SD/MMC instance. This API is not supported for SD or eMMC memory devices.

**Implemented as**

- R_SDHI_IoIntEnable

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [in] | enable | Interrupt enable = true, interrupt disable = false. |

### ◆ statusGet

fsp_err_t(* sdmmc_api_t::statusGet) (sdmmc_ctrl_t *const p_ctrl, sdmmc_status_t *const p_status)

Get SD/MMC device status.

**Implemented as**

- R_SDHI_StatusGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [out] | p_status | Pointer to current driver status. |

### ◆ erase

fsp_err_t(* sdmmc_api_t::erase) (sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector, uint32_t const sector_count)

Erase SD/MMC sectors. The sector size for erase is fixed at 512 bytes. This API is not supported for SDIO devices.

**Implemented as**

- R_SDHI_Erase

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [in] | start_sector | First sector to erase. Must be a multiple of sdmmc_device_t::erase_sector_count. |
| [in] | sector_count | Number of sectors to erase. Must be a multiple of sdmmc_device_t::erase_sector_count. All sectors must be in the range of sdmmc_device_t::sector_count. |

### ◆ close

| fsp_err_t(* sdmmc_api_t::close) (sdmmc_ctrl_t *const p_ctrl) |
|---|

Close open SD/MMC device.

**Implemented as**

- R_SDHI_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |

### ◆ versionGet

| fsp_err_t(* sdmmc_api_t::versionGet) (fsp_version_t *const p_version) |
|---|

Returns the version of the SD/MMC driver.

**Implemented as**

- R_SDHI_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Pointer to return version information to. |

### ◆ sdmmc_instance_t

| struct sdmmc_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| 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. |

## Typedef Documentation

### ◆ sdmmc_ctrl_t

| typedef void sdmmc_ctrl_t |
|---|
| SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls.<br><br>**Implemented as**<br><br>◦ sdmmc_instance_ctrl_t |

## Enumeration Type Documentation

### ◆ sdmmc_card_type_t

| enum sdmmc_card_type_t | |
|---|---|
| SD/MMC media uses SD protocol or MMC protocol. | |
| Enumerator | |
| SDMMC_CARD_TYPE_MMC | The media is an eMMC device. |
| SDMMC_CARD_TYPE_SD | The media is an SD card. |
| SDMMC_CARD_TYPE_SDIO | The media is an SDIO card. |

### ◆ sdmmc_bus_width_t

| enum sdmmc_bus_width_t | |
|---|---|
| SD/MMC data bus is 1, 4 or 8 bits wide. | |
| Enumerator | |
| 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. |

### ◆ sdmmc_io_transfer_mode_t

| enum sdmmc_io_transfer_mode_t | |
|---|---|
| SDIO transfer mode, configurable in SDIO read/write extended commands. | |
| Enumerator | |
| SDMMC_IO_MODE_TRANSFER_BYTE | SDIO byte transfer mode. |
| SDMMC_IO_MODE_TRANSFER_BLOCK | SDIO block transfer mode. |

### ◆ sdmmc_io_address_mode_t

| enum sdmmc_io_address_mode_t | |
|---|---|
| SDIO address mode, configurable in SDIO read/write extended commands. | |
| Enumerator | |
| SDMMC_IO_ADDRESS_MODE_FIXED | Write all data to the same address. |
| SDMMC_IO_ADDRESS_MODE_INCREMENT | Increment destination address after each write. |

### ◆ sdmmc_io_write_mode_t

| enum sdmmc_io_write_mode_t | |
|---|---|
| Controls the RAW (read after write) flag of CMD52. Used to read back the status after writing a control register. | |
| Enumerator | |
| SDMMC_IO_WRITE_MODE_NO_READ | Write only (do not read back) |
| SDMMC_IO_WRITE_READ_AFTER_WRITE | Read back the register after write. |

### ◆ sdmmc_event_t

| enum sdmmc_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| SDMMC_EVENT_CARD_REMOVED | Card removed event. |
| SDMMC_EVENT_CARD_INSERTED | Card inserted event. |
| SDMMC_EVENT_RESPONSE | Response event. |
| SDMMC_EVENT_SDIO | IO event. |
| SDMMC_EVENT_TRANSFER_COMPLETE | Read or write complete. |
| SDMMC_EVENT_TRANSFER_ERROR | Read or write failed. |
| SDMMC_EVENT_ERASE_COMPLETE | Erase completed. |
| SDMMC_EVENT_ERASE_BUSY | Erase timeout, poll sdmmc_api_t::statusGet. |

### ◆ sdmmc_card_detect_t

| enum sdmmc_card_detect_t | |
|---|---|
| Card detection configuration options. | |
| Enumerator | |
| SDMMC_CARD_DETECT_NONE | Card detection unused. |
| SDMMC_CARD_DETECT_CD | Card detection using the CD pin. |

### ◆ sdmmc_write_protect_t

| enum sdmmc_write_protect_t | |
|---|---|
| Write protection configuration options. | |
| Enumerator | |
| SDMMC_WRITE_PROTECT_NONE | Write protection unused. |
| SDMMC_WRITE_PROTECT_WP | Write protection using WP pin. |

### ◆ sdmmc_r1_state_t

| enum sdmmc_r1_state_t | |
|---|---|
| Card state when receiving the prior command. | |
| Enumerator | |
| SDMMC_R1_STATE_IDLE | Idle State. |
| SDMMC_R1_STATE_READY | Ready State. |
| SDMMC_R1_STATE_IDENT | Identification State. |
| SDMMC_R1_STATE_STBY | Stand-by State. |
| SDMMC_R1_STATE_TRAN | Transfer State. |
| SDMMC_R1_STATE_DATA | Sending-data State. |
| SDMMC_R1_STATE_RCV | Receive-data State. |
| SDMMC_R1_STATE_PRG | Programming State. |
| SDMMC_R1_STATE_DIS | Disconnect State (between programming and stand-by) |
| SDMMC_R1_STATE_IO | This is an I/O card and memory states do not apply. |

## 4.3.29 SLCDC Interface

Interfaces

### Detailed Description

Interface for Segment LCD controllers.

### Data Structures

| | |
|---|---|
| struct | slcdc_cfg_t |
| struct | slcdc_api_t |
| struct | slcdc_instance_t |

## Typedefs

| | |
|---|---|
| typedef void | slcdc_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | slcdc_bias_method_t |
| enum | slcdc_time_slice_t |
| enum | slcdc_waveform_t |
| enum | slcdc_drive_volt_gen_t |
| enum | slcdc_display_area_control_blink_t |
| enum | slcdc_display_area_t |
| enum | slcdc_contrast_t |
| enum | slcdc_display_on_off_t |
| enum | slcdc_display_enable_disable_t |
| enum | slcdc_display_clock_t |
| enum | slcdc_clk_div_t |

## Data Structure Documentation

### ◆ slcdc_cfg_t

| struct slcdc_cfg_t | | |
|---|---|---|
| SLCDC configuration block | | |
| Data Fields | | |
| 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_waveform_t | waveform | LCD display waveform select (LWAVE bit) |
| slcdc_drive_volt_gen_t | drive_volt_gen | LCD Drive Voltage Generator Select (MDSET bit) |
| slcdc_contrast_t | contrast | LCD Boost Level (contrast setting) |

### ◆ slcdc_api_t

| struct slcdc_api_t | |
|---|---|
| SLCDC functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(slcdc_ctrl_t *const p_ctrl, slcdc_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | write )(slcdc_ctrl_t *const p_ctrl, uint8_t const start_segment, uint8_t const *p_data, uint8_t const segment_count) |
| | |
| fsp_err_t(* | modify )(slcdc_ctrl_t *const p_ctrl, uint8_t const segment, uint8_t const data_mask, uint8_t const data) |
| | |
| fsp_err_t(* | start )(slcdc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stop )(slcdc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | setContrast )(slcdc_ctrl_t *const p_ctrl, slcdc_contrast_t const contrast) |
| | |
| fsp_err_t(* | setDisplayArea )(slcdc_ctrl_t *const p_ctrl, slcdc_display_area_t const display_area) |
| | |
| fsp_err_t(* | close )(slcdc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* slcdc_api_t::open) (slcdc_ctrl_t *const p_ctrl, slcdc_cfg_t const *const p_cfg)

Open SLCDC.

**Implemented as**

- R_SLCDC_Open()

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to display interface control block. Must be declared by user. |
| [in] | p_cfg | Pointer to display configuration structure. All elements of this structure must be set by the user. |

### ◆ write

fsp_err_t(* slcdc_api_t::write) (slcdc_ctrl_t *const p_ctrl, uint8_t const start_segment, uint8_t const *p_data, uint8_t const segment_count)

Write data to the SLCDC segment data array. Specifies the initial display data. Except when using 8-time slice mode, 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.

**Implemented as**

- R_SLCDC_Write()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | start_segment | Specify the start segment number to be written. |
| [in] | p_data | Pointer to the display data to be written to the specified segments. |
| [in] | segment_count | Number of segments to be written. |

### ◆ modify

fsp_err_t(* slcdc_api_t::modify) (slcdc_ctrl_t *const p_ctrl, uint8_t const segment, uint8_t const data_mask, uint8_t const data)

Rewrite data in the SLCDC segment data array. 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.

**Implemented as**

- R_SLCDC_Modify()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | segment | The segment to be written. |
| [in] | data_mask | 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. |
| [in] | data | Specify display data to rewrite to the specified segment. |

### ◆ start

fsp_err_t(* slcdc_api_t::start) (slcdc_ctrl_t *const p_ctrl)

Enable display signal output. Displays the segment data on the LCD.

**Implemented as**

- R_SLCDC_Start()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |

### ◆ stop

fsp_err_t(* slcdc_api_t::stop) (slcdc_ctrl_t *const p_ctrl)

Disable display signal output. Stops displaying data on the LCD.

**Implemented as**

- R_SLCDC_Stop()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |

### ◆ setContrast

fsp_err_t(* slcdc_api_t::setContrast) (slcdc_ctrl_t *const p_ctrl, slcdc_contrast_t const contrast)

Set the display contrast. This function can be used only when the internal voltage boosting method is used for drive voltage generation.

**Implemented as**

- R_SLCDC_SetContrast()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |

### ◆ setDisplayArea

fsp_err_t(* slcdc_api_t::setDisplayArea) (slcdc_ctrl_t *const p_ctrl, slcdc_display_area_t const display_area)

Set LCD display area. This function sets a specified display area, A-pattern or B-pattern. This function can be used to 'blink' the display between A-pattern and B-pattern area data.

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 IRQ 3) Start RTC counter 4) Enable IRQ, RTC_EVENT_PERIODIC_IRQ Refer to the User's Manual for the detailed procedure.

**Implemented as**

- R_SLCDC_SetDisplayArea()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | display_area | Display area to be used, A-pattern or B-pattern area. |

### ◆ close

fsp_err_t(* slcdc_api_t::close) (slcdc_ctrl_t *const p_ctrl)

Close SLCDC.

**Implemented as**

- R_SLCDC_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |

### ◆ versionGet

fsp_err_t(* slcdc_api_t::versionGet) (fsp_version_t *p_version)

Get version.

**Implemented as**

- R_SLCDC_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_version | Pointer to the memory to store the version information. |

### ◆ slcdc_instance_t

struct slcdc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| 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. |

## Typedef Documentation

### ◆ slcdc_ctrl_t

typedef void slcdc_ctrl_t

SLCDC control block. Allocate an instance specific control block to pass into the SLCDC API calls.

**Implemented as**

- slcdc_instance_ctrl_tSLCDC control block

## Enumeration Type Documentation

### ◆ slcdc_bias_method_t

| enum slcdc_bias_method_t | |
|---|---|
| LCD display bias method. | |
| Enumerator | |
| SLCDC_BIAS_2 | 1/2 bias method |
| SLCDC_BIAS_3 | 1/3 bias method |
| SLCDC_BIAS_4 | 1/4 bias method |

### ◆ slcdc_time_slice_t

| enum slcdc_time_slice_t | |
|---|---|
| Time slice of LCD display. | |
| Enumerator | |
| 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 |

### ◆ slcdc_waveform_t

| enum slcdc_waveform_t | |
|---|---|
| LCD display waveform select. | |
| Enumerator | |
| SLCDC_WAVE_A | Waveform A. |
| SLCDC_WAVE_B | Waveform B. |

### ◆ slcdc_drive_volt_gen_t

| enum slcdc_drive_volt_gen_t | |
|---|---|
| LCD Drive Voltage Generator Select. | |
| Enumerator | |
| SLCDC_VOLT_EXTERNAL | External resistance division method. |
| SLCDC_VOLT_INTERNAL | Internal voltage boosting method. |
| SLCDC_VOLT_CAPACITOR | Capacitor split method. |

### ◆ slcdc_display_area_control_blink_t

| enum slcdc_display_area_control_blink_t | |
|---|---|
| Display Data Area Control | |
| Enumerator | |
| SLCDC_NOT_BLINKING | Display either A-pattern or B-pattern data. |
| SLCDC_BLINKING | Alternately display A-pattern and B-pattern data. |

### ◆ slcdc_display_area_t

| enum slcdc_display_area_t | |
|---|---|
| Display Area data | |
| Enumerator | |
| SLCDC_DISP_A | Display A-pattern data. |
| SLCDC_DISP_B | Display B-pattern data. |
| SLCDC_DISP_BLINK | Blink between A- and B-pattern. |

### ◆ slcdc_contrast_t

| enum slcdc_contrast_t | |
|---|---|
| LCD Boost Level (contrast) settings | |
| Enumerator | |
| SLCDC_CONTRAST_0 | Contrast level 0. |
| SLCDC_CONTRAST_1 | Contrast level 1. |
| SLCDC_CONTRAST_2 | Contrast level 2. |
| SLCDC_CONTRAST_3 | Contrast level 3. |
| SLCDC_CONTRAST_4 | Contrast level 4. |
| SLCDC_CONTRAST_5 | Contrast level 5. |
| SLCDC_CONTRAST_6 | Contrast level 6. |
| SLCDC_CONTRAST_7 | Contrast level 7. |
| SLCDC_CONTRAST_8 | Contrast level 8. |
| SLCDC_CONTRAST_9 | Contrast level 9. |
| SLCDC_CONTRAST_10 | Contrast level 10. |
| SLCDC_CONTRAST_11 | Contrast level 11. |
| SLCDC_CONTRAST_12 | Contrast level 12. |
| SLCDC_CONTRAST_13 | Contrast level 13. |
| SLCDC_CONTRAST_14 | Contrast level 14. |
| SLCDC_CONTRAST_15 | Contrast level 15. |

### ◆ slcdc_display_on_off_t

| enum slcdc_display_on_off_t | |
|---|---|
| LCD Display Enable/Disable | |
| Enumerator | |
| SLCDC_DISP_OFF | Display off. |
| SLCDC_DISP_ON | Display on. |

### ◆ slcdc_display_enable_disable_t

| enum slcdc_display_enable_disable_t | |
|---|---|
| LCD Display output enable | |
| Enumerator | |
| SLCDC_DISP_DISABLE | Output ground level to segment/common pins. |
| SLCDC_DISP_ENABLE | Output enable. |

### ◆ slcdc_display_clock_t

| enum slcdc_display_clock_t | |
|---|---|
| LCD Display clock selection | |
| Enumerator | |
| 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. |

### ◆ slcdc_clk_div_t

| enum slcdc_clk_div_t | |
|---|---|
| LCD clock settings | |
| Enumerator | |
| 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. |
| SLCDC_CLK_DIVISOR_HOCO_524288 | HOCO Clock/524288. |

### 4.3.30 SPI Interface

Interfaces

### Detailed Description

Interface for SPI communications.

# Summary

Provides a common interface for communication using the SPI Protocol.

Implemented by:

- Serial Peripheral Interface (r_spi)
- Serial Communications Interface (SCI) SPI (r_sci_spi)

### Data Structures

| | |
|---|---|
| struct | spi_callback_args_t |
| struct | spi_cfg_t |
| struct | spi_api_t |
| struct | spi_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | spi_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | spi_bit_width_t |
| enum | spi_mode_t |
| enum | spi_clk_phase_t |
| enum | spi_clk_polarity_t |
| enum | spi_mode_fault_t |
| enum | spi_bit_order_t |
| enum | spi_event_t |

### Data Structure Documentation

#### ◆ spi_callback_args_t

| |
|---|
| struct spi_callback_args_t |
| Common callback parameter definition |
| Data Fields |

| | | |
|---|---|---|
| uint32_t | channel | Device channel number. |
| spi_event_t | event | Event code. |
| void const * | p_context | Context provided to user during callback. |

### ◆ spi_cfg_t

| struct spi_cfg_t | |
|---|---|
| SPI interface configuration | |
| **Data Fields** | |
| uint8_t | channel |
| | Channel number to be used. |
| | |
| IRQn_Type | rxi_irq |
| | Receive Buffer Full IRQ number. |
| | |
| IRQn_Type | txi_irq |
| | Transmit Buffer Empty IRQ number. |
| | |
| IRQn_Type | tei_irq |
| | Transfer Complete IRQ number. |
| | |
| IRQn_Type | eri_irq |
| | Error IRQ number. |
| | |
| uint8_t | rxi_ipl |
| | Receive Interrupt priority. |
| | |
| uint8_t | txi_ipl |
| | Transmit Interrupt priority. |
| | |
| | |

| | |
|---|---|
| uint8_t | tei_ipl |
| | Transfer Complete 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. |
| | |
| 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. |
| | |

### ◆ spi_api_t

| | |
|---|---|
| struct spi_api_t | |
| Shared Interface definition for SPI | |
| **Data Fields** | |
| fsp_err_t(* | open )(spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | read )(spi_ctrl_t *const p_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |
| | |
| fsp_err_t(* | write )(spi_ctrl_t *const p_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width) |
| | |
| fsp_err_t(* | writeRead )(spi_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |
| | |
| fsp_err_t(* | close )(spi_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* spi_api_t::open) (spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg)

Initialize a channel for SPI communication mode.

**Implemented as**

- R_SPI_Open()
- R_SCI_SPI_Open()

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to user-provided storage for the control block. |
| [in] | p_cfg | Pointer to SPI configuration structure. |

### ◆ read

fsp_err_t(* spi_api_t::read) (spi_ctrl_t *const p_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)

Receive data from a SPI device.

**Implemented as**

- R_SPI_Read()
- R_SCI_SPI_Read()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the control block for the channel. |
| [in] | length | Number of units of data to be transferred (unit size specified by the bit_width). |
| [in] | bit_width | Data bit width to be transferred. |
| [out] | p_dest | 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. |

### ◆ write

fsp_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)

Transmit data to a SPI device.

**Implemented as**

- R_SPI_Write()
- R_SCI_SPI_Write()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the control block for the channel. |
| [in] | p_src | Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL. |
| [in] | length | Number of units of data to be transferred (unit size specified by the bit_width). |
| [in] | bit_width | Data bit width to be transferred. |

### ◆ writeRead

fsp_err_t(* spi_api_t::writeRead) (spi_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)

Simultaneously transmit data to a SPI device while receiving data from a SPI device (full duplex).

**Implemented as**

- R_SPI_WriteRead()
- R_SCI_SPI_WriteRead()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the control block for the channel. |
| [in] | p_src | Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL. |
| [out] | p_dest | 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. |
| [in] | length | Number of units of data to be transferred (unit size specified by the bit_width). |
| [in] | bit_width | Data bit width to be transferred. |

### ◆ close

fsp_err_t(* spi_api_t::close) (spi_ctrl_t *const p_ctrl)

Remove power to the SPI channel designated by the handle and disable the associated interrupts.

**Implemented as**

- R_SPI_Close()
- R_SCI_SPI_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the control block for the channel. |

### ◆ versionGet

fsp_err_t(* spi_api_t::versionGet) (fsp_version_t *p_version)

Get the version information of the underlying driver.

**Implemented as**

- R_SPI_VersionGet()
- R_SCI_SPI_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | pointer to memory location to return version number |

### ◆ spi_instance_t

struct spi_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| 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. |

## Typedef Documentation

### ◆ spi_ctrl_t

typedef void spi_ctrl_t

SPI control block. Allocate an instance specific control block to pass into the SPI API calls.

**Implemented as**

- sci_spi_instance_ctrl_t
- spi_instance_ctrl_t

## Enumeration Type Documentation

### ◆ spi_bit_width_t

| enum spi_bit_width_t | |
|---|---|
| Data bit width | |
| Enumerator | |
| 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. |

### ◆ spi_mode_t

| enum spi_mode_t | |
|---|---|
| Master or slave operating mode | |
| Enumerator | |
| SPI_MODE_MASTER | Channel operates as SPI master. |
| SPI_MODE_SLAVE | Channel operates as SPI slave. |

### ◆ spi_clk_phase_t

| enum spi_clk_phase_t | |
|---|---|
| Clock phase | |
| Enumerator | |
| 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 |

### ◆ spi_clk_polarity_t

| enum spi_clk_polarity_t | |
|---|---|
| Clock polarity | |
| Enumerator | |
| SPI_CLK_POLARITY_LOW | 0: Clock polarity is low when idle |
| SPI_CLK_POLARITY_HIGH | 1: Clock polarity is high when idle |

### ◆ spi_mode_fault_t

| enum spi_mode_fault_t | |
|---|---|
| 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. | |
| Enumerator | |
| SPI_MODE_FAULT_ERROR_ENABLE | Mode fault error flag on. |
| SPI_MODE_FAULT_ERROR_DISABLE | Mode fault error flag off. |

### ◆ spi_bit_order_t

| enum spi_bit_order_t | |
|---|---|
| Bit order | |
| Enumerator | |
| SPI_BIT_ORDER_MSB_FIRST | Send MSB first in transmission. |
| SPI_BIT_ORDER_LSB_FIRST | Send LSB first in transmission. |

### ◆ spi_event_t

| enum spi_event_t | |
|---|---|
| SPI events | |
| Enumerator | |
| 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. |

## 4.3.31 SPI Flash Interface

Interfaces

### Detailed Description

Interface for accessing external SPI flash devices.

# Summary

The SPI flash API provides an interface that configures, writes, and erases sectors in SPI flash devices.

### Data Structures

| | |
|---|---|
| struct | spi_flash_erase_command_t |
| struct | spi_flash_cfg_t |
| struct | spi_flash_status_t |
| struct | spi_flash_api_t |

| | |
|---|---|
| struct | spi_flash_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | spi_flash_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | spi_flash_read_mode_t |
| enum | spi_flash_protocol_t |
| enum | spi_flash_address_bytes_t |
| enum | spi_flash_data_lines_t |
| enum | spi_flash_dummy_clocks_t |

## Data Structure Documentation

### ◆ spi_flash_erase_command_t

| struct spi_flash_erase_command_t | | |
|---|---|---|
| Structure to define an erase command and associated erase size. | | |
| Data Fields | | |
| uint8_t | command | Erase command. |
| uint32_t | size | Size of erase for associated command, set to SPI_FLASH_ERASE_SIZE_CHIP_ERASE for chip erase. |

### ◆ spi_flash_cfg_t

| struct spi_flash_cfg_t | | |
|---|---|---|
| User configuration structure used by the open function | | |
| Data Fields | | |
| spi_flash_protocol_t | spi_protocol | Initial SPI protocol. SPI protocol can be changed in spi_flash_api_t::spiProtocolSet. |
| spi_flash_read_mode_t | read_mode | Read mode. |
| spi_flash_address_bytes_t | address_bytes | Number of bytes used to represent the address. |
| spi_flash_dummy_clocks_t | dummy_clocks | Number of dummy clocks to use for fast read operations. |
| spi_flash_data_lines_t | page_program_address_lines | Number of lines used to send address for page program |

| | | |
|---|---|---|
| | | command. This should either be 1 or match the number of lines used in the selected read mode. |
| uint32_t | page_size_bytes | Page size in bytes (maximum number of bytes for page program) |
| uint8_t | page_program_command | Page program command. |
| uint8_t | write_enable_command | Command to enable write or erase, typically 0x06. |
| uint8_t | status_command | Command to read the write status. |
| uint8_t | write_status_bit | Which bit determines write status. |
| uint8_t | xip_enter_command | Command to enter XIP mode. |
| uint8_t | xip_exit_command | Command to exit XIP mode. |
| uint8_t | erase_command_list_length | Length of erase command list. |
| spi_flash_erase_command_t const * | p_erase_command_list | List of all erase commands and associated sizes. |
| void const * | p_extend | Pointer to implementation specific extended configurations. |

### ◆ spi_flash_status_t

| struct spi_flash_status_t | | |
|---|---|---|
| Status. | | |
| Data Fields | | |
| bool | write_in_progress | Whether or not a write is in progress. This is determined by reading the spi_flash_cfg_t::write_status_bit from the spi_flash_cfg_t::status_command. |

### ◆ spi_flash_api_t

| struct spi_flash_api_t | |
|---|---|
| SPI flash implementations follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(spi_flash_ctrl_t *p_ctrl, spi_flash_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | directWrite )(spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, |

| | |
|---|---|
| | uint32_t const bytes, bool const read_after_write) |
| | |
| fsp_err_t(* | directRead )(spi_flash_ctrl_t *p_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| | |
| fsp_err_t(* | spiProtocolSet )(spi_flash_ctrl_t *p_ctrl, spi_flash_protocol_t spi_protocol) |
| | |
| fsp_err_t(* | write )(spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint32_t byte_count) |
| | |
| fsp_err_t(* | erase )(spi_flash_ctrl_t *p_ctrl, uint8_t *const p_device_address, uint32_t byte_count) |
| | |
| fsp_err_t(* | statusGet )(spi_flash_ctrl_t *p_ctrl, spi_flash_status_t *const p_status) |
| | |
| fsp_err_t(* | xipEnter )(spi_flash_ctrl_t *p_ctrl) |
| | |
| fsp_err_t(* | xipExit )(spi_flash_ctrl_t *p_ctrl) |
| | |
| fsp_err_t(* | bankSet )(spi_flash_ctrl_t *p_ctrl, uint32_t bank) |
| | |
| fsp_err_t(* | close )(spi_flash_ctrl_t *p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* spi_flash_api_t::open) (spi_flash_ctrl_t *p_ctrl, spi_flash_cfg_t const *const p_cfg)

Open the SPI flash driver module.

**Implemented as**

- R_QSPI_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |
| [in] | p_cfg | Pointer to a configuration structure |

### ◆ directWrite

fsp_err_t(* spi_flash_api_t::directWrite) (spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, uint32_t const bytes, bool const read_after_write)

Write raw data to the SPI flash.

**Implemented as**

- R_QSPI_DirectWrite()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |
| [in] | p_src | Pointer to raw data to write, must include any required command/address |
| [in] | bytes | Number of bytes to write |
| [in] | read_after_write | If true, the slave select remains asserted and the peripheral does not return to direct communications mode. If false, the slave select is deasserted and memory mapped access is possible after this function returns if the device is not busy. |

### ◆ directRead

fsp_err_t(* spi_flash_api_t::directRead) (spi_flash_ctrl_t *p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

Read raw data from the SPI flash. Must follow a call to spi_flash_api_t::directWrite.

**Implemented as**

- R_QSPI_DirectRead()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |
| [out] | p_dest | Pointer to read raw data into |
| [in] | bytes | Number of bytes to read |

### ◆ spiProtocolSet

fsp_err_t(* spi_flash_api_t::spiProtocolSet) (spi_flash_ctrl_t *p_ctrl, spi_flash_protocol_t spi_protocol)

Change the SPI protocol in the driver. The application must change the SPI protocol on the device.

**Implemented as**

- R_QSPI_SpiProtocolSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |
| [in] | spi_protocol | Desired SPI protocol |

### ◆ write

fsp_err_t(* spi_flash_api_t::write) (spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint32_t byte_count)

Program a page of data to the flash.

**Implemented as**

- R_QSPI_Write()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |
| [in] | p_src | The memory address of the data to write to the flash device |
| [in] | p_dest | The location in the flash device address space to write the data to |
| [in] | byte_count | The number of bytes to write |

### ◆ erase

fsp_err_t(* spi_flash_api_t::erase) (spi_flash_ctrl_t *p_ctrl, uint8_t *const p_device_address, uint32_t byte_count)

Erase a certain number of bytes of the flash.

**Implemented as**

- R_QSPI_Erase()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |
| [in] | p_device_address | The location in the flash device address space to start the erase from |
| [in] | byte_count | The number of bytes to erase. Set to SPI_FLASH_ERASE_SIZE_CHIP_ERASE to erase entire chip. |

### ◆ statusGet

fsp_err_t(* spi_flash_api_t::statusGet) (spi_flash_ctrl_t *p_ctrl, spi_flash_status_t *const p_status)

Get the write or erase status of the flash.

**Implemented as**

- R_QSPI_StatusGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |
| [out] | p_status | Current status of the SPI flash device stored here. |

### ◆ xipEnter

fsp_err_t(* spi_flash_api_t::xipEnter) (spi_flash_ctrl_t *p_ctrl)

Enter XIP mode.

**Implemented as**

- R_QSPI_XipEnter()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |

### ◆ xipExit

fsp_err_t(* spi_flash_api_t::xipExit) (spi_flash_ctrl_t *p_ctrl)

Exit XIP mode.

**Implemented as**

- R_QSPI_XipExit()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |

### ◆ bankSet

fsp_err_t(* spi_flash_api_t::bankSet) (spi_flash_ctrl_t *p_ctrl, uint32_t bank)

Select the bank to access. See implementation for details.

**Implemented as**

- R_QSPI_BankSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |
| [in] | bank | The bank number |

### ◆ close

fsp_err_t(* spi_flash_api_t::close) (spi_flash_ctrl_t *p_ctrl)

Close the SPI flash driver module.

**Implemented as**

- R_QSPI_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |

### ◆ versionGet

fsp_err_t(* spi_flash_api_t::versionGet) (fsp_version_t *const p_version)

Get the driver version based on compile time macros.

**Implemented as**

- R_QSPI_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version stored here. |

### ◆ spi_flash_instance_t

| struct spi_flash_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| spi_flash_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| spi_flash_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| spi_flash_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ spi_flash_ctrl_t

typedef void spi_flash_ctrl_t

SPI flash control block. Allocate an instance specific control block to pass into the SPI flash API calls.

**Implemented as**

- qspi_instance_ctrl_t

## Enumeration Type Documentation

### ◆ spi_flash_read_mode_t

| enum spi_flash_read_mode_t | |
|---|---|
| Read mode. | |
| Enumerator | |
| SPI_FLASH_READ_MODE_STANDARD | Standard Read Mode (no dummy cycles) |
| SPI_FLASH_READ_MODE_FAST_READ | Fast Read Mode (dummy cycles between address and data) |
| SPI_FLASH_READ_MODE_FAST_READ_DUAL_OUTPUT | Fast Read Dual Output Mode (data on 2 lines) |
| SPI_FLASH_READ_MODE_FAST_READ_DUAL_IO | Fast Read Dual I/O Mode (address and data on 2 lines) |
| SPI_FLASH_READ_MODE_FAST_READ_QUAD_OUTPUT | Fast Read Quad Output Mode (data on 4 lines) |
| SPI_FLASH_READ_MODE_FAST_READ_QUAD_IO | Fast Read Quad I/O Mode (address and data on 4 lines) |

### ◆ spi_flash_protocol_t

| enum spi_flash_protocol_t | |
|---|---|
| SPI protocol. | |
| Enumerator | |
| SPI_FLASH_PROTOCOL_EXTENDED_SPI | Extended SPI mode (commands on 1 line) |
| SPI_FLASH_PROTOCOL_QPI | QPI mode (commands on 4 lines). Note that the application must ensure the device is in QPI mode. |

### ◆ spi_flash_address_bytes_t

| enum spi_flash_address_bytes_t | |
|---|---|
| Number of bytes in the address. | |
| Enumerator | |
| SPI_FLASH_ADDRESS_BYTES_3 | 3 address bytes |
| SPI_FLASH_ADDRESS_BYTES_4 | 4 address bytes with standard commands. If this option is selected, the application must issue the EN4B command using spi_flash_api_t::directWrite() if required by the device. |
| SPI_FLASH_ADDRESS_BYTES_4_4BYTE_READ_CODE | 4 address bytes using standard 4-byte command set. |

### ◆ spi_flash_data_lines_t

| enum spi_flash_data_lines_t | |
|---|---|
| Number of data lines used. | |
| Enumerator | |
| SPI_FLASH_DATA_LINES_1 | 1 data line |
| SPI_FLASH_DATA_LINES_2 | 2 data lines |
| SPI_FLASH_DATA_LINES_4 | 4 data lines |

### ◆ spi_flash_dummy_clocks_t

| enum spi_flash_dummy_clocks_t | |
|---|---|
| Number of dummy cycles for fast read operations. | |
| Enumerator | |
| SPI_FLASH_DUMMY_CLOCKS_DEFAULT | Default is 6 clocks for Fast Read Quad I/O, 4 clocks for Fast Read Dual I/O, and 8 clocks for other fast read instructions including Fast Read Quad Output, Fast Read Dual Output, and Fast Read. |
| SPI_FLASH_DUMMY_CLOCKS_3 | 3 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_4 | 4 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_5 | 5 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_6 | 6 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_7 | 7 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_8 | 8 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_9 | 9 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_10 | 10 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_11 | 11 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_12 | 12 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_13 | 13 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_14 | 14 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_15 | 15 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_16 | 16 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_17 | 17 dummy clocks |

## 4.3.32 Three-Phase Interface

Interfaces

### Detailed Description

Interface for three-phase timer functions.

# Summary

The Three-Phase interface provides functionality for synchronous start/stop/reset control of three timer channels for use in 3-phase motor control applications.

Implemented by:

- General PWM Timer Three-Phase Motor Control Driver (r_gpt_three_phase)

### Data Structures

| | |
|---|---|
| struct | three_phase_duty_cycle_t |
| struct | three_phase_cfg_t |
| struct | three_phase_api_t |
| struct | three_phase_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | three_phase_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | three_phase_channel_t |
| enum | three_phase_buffer_mode_t |

## Data Structure Documentation

### ◆ three_phase_duty_cycle_t

| struct three_phase_duty_cycle_t | | |
|---|---|---|
| Struct for passing duty cycle values to three_phase_api_t::dutyCycleSet | | |
| Data Fields | | |
| uint32_t | duty[3] | Duty cycle. |
| uint32_t | duty_buffer[3] | Double-buffer for duty cycle values. |

### ◆ three_phase_cfg_t

| struct three_phase_cfg_t |
|---|
| User configuration structure, used in open function |

| Data Fields | | |
|---|---|---|
| three_phase_buffer_mode_t | buffer_mode | Single or double-buffer mode. |
| timer_instance_t const * | p_timer_instance[3] | Pointer to the timer instance structs. |
| uint32_t | channel_mask | Bitmask of timer channels used by this module. |
| 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. |

### ◆ three_phase_api_t

| struct three_phase_api_t | |
|---|---|
| Three-Phase API structure. | |
| **Data Fields** | |
| fsp_err_t(* | open )(three_phase_ctrl_t *const p_ctrl, three_phase_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | start )(three_phase_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stop )(three_phase_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | reset )(three_phase_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | dutyCycleSet )(three_phase_ctrl_t *const p_ctrl, three_phase_duty_cycle_t *const p_duty_cycle) |
| | |
| fsp_err_t(* | close )(three_phase_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* three_phase_api_t::open) (three_phase_ctrl_t *const p_ctrl, three_phase_cfg_t const *const p_cfg)

Initial configuration.

**Implemented as**

- R_GPT_THREE_PHASE_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ start

fsp_err_t(* three_phase_api_t::start) (three_phase_ctrl_t *const p_ctrl)

Start all three timers synchronously.

**Implemented as**

- R_GPT_THREE_PHASE_Start()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in three_phase_api_t::open call for this timer. |

### ◆ stop

fsp_err_t(* three_phase_api_t::stop) (three_phase_ctrl_t *const p_ctrl)

Stop all three timers synchronously.

**Implemented as**

- R_GPT_THREE_PHASE_Stop()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in three_phase_api_t::open call for this timer. |

### ◆ reset

fsp_err_t(* three_phase_api_t::reset) (three_phase_ctrl_t *const p_ctrl)

Reset all three timers synchronously.

**Implemented as**

- R_GPT_THREE_PHASE_Reset()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in three_phase_api_t::open call for this timer. |

### ◆ dutyCycleSet

fsp_err_t(* three_phase_api_t::dutyCycleSet) (three_phase_ctrl_t *const p_ctrl, three_phase_duty_cycle_t *const p_duty_cycle)

Sets the duty cycle match values. If the timer is counting, the updated duty cycle is reflected after the next timer expiration.

**Implemented as**

- R_GPT_THREE_PHASE_DutyCycleSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in three_phase_api_t::open call for this timer. |
| [in] | p_duty_cycle | Duty cycle values for all three timer channels. |

### ◆ close

fsp_err_t(* three_phase_api_t::close) (three_phase_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

**Implemented as**

- R_GPT_THREE_PHASE_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in three_phase_api_t::open call for this timer. |

### ◆ versionGet

fsp_err_t(* three_phase_api_t::versionGet) (fsp_version_t *const p_version)

Get version and store it in provided pointer p_version.

**Implemented as**

- R_GPT_THREE_PHASE_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version used. |

### ◆ three_phase_instance_t

struct three_phase_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| three_phase_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| three_phase_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| three_phase_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ three_phase_ctrl_t

typedef void three_phase_ctrl_t

Three-Phase control block. Allocate an instance specific control block to pass into the timer API calls.

**Implemented as**

- gpt_three_phase_instance_ctrl_t

## Enumeration Type Documentation

### ◆ three_phase_channel_t

| enum three_phase_channel_t | |
|---|---|
| Timer channel indices | |
| Enumerator | |
| THREE_PHASE_CHANNEL_U | U-channel index. |
| THREE_PHASE_CHANNEL_V | V-channel index. |
| THREE_PHASE_CHANNEL_W | W-channel index. |

### ◆ three_phase_buffer_mode_t

| enum three_phase_buffer_mode_t | |
|---|---|
| Buffering mode | |
| Enumerator | |
| THREE_PHASE_BUFFER_MODE_SINGLE | Single-buffer mode. |
| THREE_PHASE_BUFFER_MODE_DOUBLE | Double-buffer mode. |

## 4.3.33 Timer Interface

Interfaces

### Detailed Description

Interface for timer functions.

# Summary

The general timer interface provides standard timer functionality including periodic mode, one-shot mode, PWM output, 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:

- General PWM Timer (r_gpt)
- Asynchronous General Purpose Timer (r_agt)

## Data Structures

| | |
|---|---|
| struct | timer_callback_args_t |
| struct | timer_info_t |
| struct | timer_status_t |
| struct | timer_cfg_t |
| struct | timer_api_t |
| struct | timer_instance_t |

## Typedefs

| | |
|---|---|
| typedef void | timer_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | timer_event_t |
| enum | timer_variant_t |
| enum | timer_state_t |
| enum | timer_mode_t |
| enum | timer_direction_t |
| enum | timer_source_div_t |

## Data Structure Documentation

### ◆ timer_callback_args_t

| struct timer_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| 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. |
| uint32_t | capture | Most recent capture, only valid if event is TIMER_EVENT_CAPTURE_A or TIMER_EVENT_CAPTURE_B. |

### ◆ timer_info_t

| struct timer_info_t | | |
|---|---|---|
| Timer information structure to store various information for a timer resource | | |
| Data Fields | | |
| timer_direction_t | count_direction | Clock counting direction of the timer. |
| uint32_t | clock_frequency | Clock frequency of the timer counter. |
| uint32_t | period_counts | Period in raw timer counts.<br>*Note*<br>*For triangle wave PWM modes, the full period is double this value.* |

### ◆ timer_status_t

| struct timer_status_t | | |
|---|---|---|
| Current timer status. | | |
| Data Fields | | |
| uint32_t | counter | Current counter value. |
| timer_state_t | state | Current timer state (running or stopped) |

### ◆ timer_cfg_t

| struct timer_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| timer_mode_t | mode |
| | Select enumerated value from timer_mode_t. |
| | |
| uint32_t | period_counts |
| | Period in raw timer counts. |
| | |
| timer_source_div_t | source_div |
| | Source clock divider. |
| | |
| uint32_t | duty_cycle_counts |

| | |
|---|---|
| | Duty cycle in counts. |
| | |
| uint8_t | channel |
| | |
| uint8_t | cycle_end_ipl |
| | Cycle end interrupt priority. |
| | |
| IRQn_Type | cycle_end_irq |
| | Cycle end interrupt. |
| | |
| void(* | p_callback )(timer_callback_args_t *p_args) |
| | |
| void const * | p_context |
| | |
| void const * | p_extend |
| | Extension parameter for hardware specific settings. |
| | |

## Field Documentation

### ◆ channel

uint8_t timer_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

### ◆ p_callback

void(* timer_cfg_t::p_callback) (timer_callback_args_t *p_args)

Callback provided when a timer ISR occurs. Set to NULL for no CPU interrupt.

### ◆ p_context

void const* timer_cfg_t::p_context

Placeholder for user data. Passed to the user callback in timer_callback_args_t.

### ◆ timer_api_t

struct timer_api_t

Timer API structure. General timer functions implemented at the HAL layer follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | start )(timer_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stop )(timer_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | reset )(timer_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | enable )(timer_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | disable )(timer_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | periodSet )(timer_ctrl_t *const p_ctrl, uint32_t const period) |
| | |
| fsp_err_t(* | dutyCycleSet )(timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin) |
| | |
| fsp_err_t(* | infoGet )(timer_ctrl_t *const p_ctrl, timer_info_t *const p_info) |
| | |
| fsp_err_t(* | statusGet )(timer_ctrl_t *const p_ctrl, timer_status_t *const p_status) |
| | |
| fsp_err_t(* | close )(timer_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* timer_api_t::open) (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)

Initial configuration.

**Implemented as**

- R_GPT_Open()
- R_AGT_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ start

fsp_err_t(* timer_api_t::start) (timer_ctrl_t *const p_ctrl)

Start the counter.

**Implemented as**

- R_GPT_Start()
- R_AGT_Start()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |

### ◆ stop

fsp_err_t(* timer_api_t::stop) (timer_ctrl_t *const p_ctrl)

Stop the counter.

**Implemented as**

- R_GPT_Stop()
- R_AGT_Stop()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |

### ◆ reset

fsp_err_t(* timer_api_t::reset) (timer_ctrl_t *const p_ctrl)

Reset the counter to the initial value.

**Implemented as**

- R_GPT_Reset()
- R_AGT_Reset()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |

### ◆ enable

fsp_err_t(* timer_api_t::enable) (timer_ctrl_t *const p_ctrl)

Enables input capture.

**Implemented as**

- R_GPT_Enable()
- R_AGT_Enable()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |

### ◆ disable

fsp_err_t(* timer_api_t::disable) (timer_ctrl_t *const p_ctrl)

Disables input capture.

**Implemented as**

- R_GPT_Disable()
- R_AGT_Disable()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |

### ◆ periodSet

fsp_err_t(* timer_api_t::periodSet) (timer_ctrl_t *const p_ctrl, uint32_t const period)

Set the time until the timer expires. See implementation for details of period update timing.

**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 timer_api_t::open.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |
| [in] | p_period | Time until timer should expire. |

### ◆ dutyCycleSet

fsp_err_t(* timer_api_t::dutyCycleSet) (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin)

Sets the number of counts for the pin level to be high. If the timer is counting, the updated duty cycle is reflected after the next timer expiration.

**Implemented as**

- R_GPT_DutyCycleSet()
- R_AGT_DutyCycleSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |
| [in] | duty_cycle_counts | Time until duty cycle should expire. |
| [in] | pin | Which output pin to update. See implementation for details. |

### ◆ infoGet

fsp_err_t(* timer_api_t::infoGet) (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)

Stores timer information in p_info.

**Implemented as**

- R_GPT_InfoGet()
- R_AGT_InfoGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |
| [out] | p_info | Collection of information for this timer. |

### ◆ statusGet

fsp_err_t(* timer_api_t::statusGet) (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status)

Get the current counter value and timer state and store it in p_status.

**Implemented as**

- R_GPT_StatusGet()
- R_AGT_StatusGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |
| [out] | p_status | Current status of this timer. |

### ◆ close

fsp_err_t(* timer_api_t::close) (timer_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

**Implemented as**

- R_GPT_Close()
- R_AGT_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |

### ◆ versionGet

fsp_err_t(* timer_api_t::versionGet) (fsp_version_t *const p_version)

Get version and store it in provided pointer p_version.

**Implemented as**

- R_GPT_VersionGet()
- R_AGT_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version used. |

### ◆ timer_instance_t

struct timer_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| 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. |

## Typedef Documentation

### ◆ timer_ctrl_t

typedef void timer_ctrl_t

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

## Enumeration Type Documentation

### ◆ timer_event_t

| enum timer_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| TIMER_EVENT_CYCLE_END | Requested timer delay has expired or timer has wrapped around. |
| TIMER_EVENT_CREST | Timer crest event (counter is at a maximum, triangle-wave PWM only) |
| TIMER_EVENT_CAPTURE_A | A capture has occurred on signal A. |
| TIMER_EVENT_CAPTURE_B | A capture has occurred on signal B. |
| TIMER_EVENT_TROUGH | Timer trough event (counter is 0, triangle-wave PWM only. |

### ◆ timer_variant_t

| enum timer_variant_t | |
|---|---|
| Timer variant types. | |
| Enumerator | |
| TIMER_VARIANT_32_BIT | 32-bit timer |
| TIMER_VARIANT_16_BIT | 16-bit timer |

### ◆ timer_state_t

| enum timer_state_t | |
|---|---|
| Possible status values returned by timer_api_t::statusGet. | |
| Enumerator | |
| TIMER_STATE_STOPPED | Timer is stopped. |
| TIMER_STATE_COUNTING | Timer is running. |

### ◆ timer_mode_t

| enum timer_mode_t | |
|---|---|
| Timer operational modes | |
| Enumerator | |
| TIMER_MODE_PERIODIC | Timer restarts after period elapses. |
| TIMER_MODE_ONE_SHOT | Timer stops after period elapses. |
| TIMER_MODE_PWM | Timer generates saw-wave PWM output. |
| TIMER_MODE_TRIANGLE_WAVE_SYMMETRIC_PWM | Timer generates symmetric triangle-wave PWM output. |
| TIMER_MODE_TRIANGLE_WAVE_ASYMMETRIC_PWM | Timer generates asymmetric triangle-wave PWM output. |

### ◆ timer_direction_t

| enum timer_direction_t | |
|---|---|
| Direction of timer count | |
| Enumerator | |
| TIMER_DIRECTION_DOWN | Timer count goes up. |
| TIMER_DIRECTION_UP | Timer count goes down. |

### ◆ timer_source_div_t

| enum timer_source_div_t | |
|---|---|
| PCLK divisors | |
| Enumerator | |
| TIMER_SOURCE_DIV_1 | Timer clock source divided by 1. |
| TIMER_SOURCE_DIV_2 | Timer clock source divided by 2. |
| TIMER_SOURCE_DIV_4 | Timer clock source divided by 4. |
| TIMER_SOURCE_DIV_8 | Timer clock source divided by 8. |
| TIMER_SOURCE_DIV_16 | Timer clock source divided by 16. |
| TIMER_SOURCE_DIV_32 | Timer clock source divided by 32. |
| TIMER_SOURCE_DIV_64 | Timer clock source divided by 64. |
| TIMER_SOURCE_DIV_128 | Timer clock source divided by 128. |
| TIMER_SOURCE_DIV_256 | Timer clock source divided by 256. |
| TIMER_SOURCE_DIV_1024 | Timer clock source divided by 1024. |

## 4.3.34 Transfer Interface

Interfaces

### Detailed Description

Interface for data transfer functions.

# Summary

The transfer interface supports background data transfer (no CPU intervention).

Implemented by:

- Data Transfer Controller (r_dtc)
- Direct Memory Access Controller (r_dmac)

### Data Structures

| | |
|---|---|
| struct | transfer_properties_t |
| struct | transfer_info_t |
| struct | transfer_cfg_t |
| struct | transfer_api_t |
| struct | transfer_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | transfer_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | transfer_mode_t |
| enum | transfer_size_t |
| enum | transfer_addr_mode_t |
| enum | transfer_repeat_area_t |
| enum | transfer_chain_mode_t |
| enum | transfer_irq_t |
| enum | transfer_start_mode_t |

## Data Structure Documentation

### ◆ transfer_properties_t

| struct transfer_properties_t | | |
|---|---|---|
| Driver specific information. | | |
| Data Fields | | |
| uint32_t | block_count_max | Maximum number of blocks. |
| uint32_t | block_count_remaining | Number of blocks remaining. |
| uint32_t | transfer_length_max | Maximum number of transfers. |
| uint32_t | transfer_length_remaining | Number of transfers remaining. |

### ◆ transfer_info_t

| struct transfer_info_t |
|---|
| This structure specifies the properties of the transfer. |

**Warning**

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.
When using DTC, do NOT reuse this structure to configure multiple transfers. Each transfer must have a unique transfer_info_t.
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.*

| Data Fields | | |
|---|---|---|
| union transfer_info_t | __unnamed__ | |
| void const *volatile | p_src | Source pointer. |
| void *volatile | p_dest | Destination pointer. |
| volatile 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. |
| volatile uint16_t | length | Length of each transfer. Range limited for TRANSFER_MODE_BLOCK and TRANSFER_MODE_REPEAT, see HAL driver for details. |

### ◆ transfer_cfg_t

struct transfer_cfg_t

Driver configuration set in transfer_api_t::open. All elements except p_extend are required and must be initialized.

| Data Fields | | |
|---|---|---|
| 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. |
| void const * | p_extend | Extension parameter for hardware specific settings. |

### ◆ transfer_api_t

struct transfer_api_t

Transfer functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(transfer_ctrl_t *const p_ctrl, transfer_cfg_t const *const p_cfg) |

| | |
|---|---|
| fsp_err_t(* | reconfigure )(transfer_ctrl_t *const p_ctrl, transfer_info_t *p_info) |
| fsp_err_t(* | reset )(transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const num_transfers) |
| fsp_err_t(* | enable )(transfer_ctrl_t *const p_ctrl) |
| fsp_err_t(* | disable )(transfer_ctrl_t *const p_ctrl) |
| fsp_err_t(* | softwareStart )(transfer_ctrl_t *const p_ctrl, transfer_start_mode_t mode) |
| fsp_err_t(* | softwareStop )(transfer_ctrl_t *const p_ctrl) |
| fsp_err_t(* | infoGet )(transfer_ctrl_t *const p_ctrl, transfer_properties_t *const p_properties) |
| fsp_err_t(* | close )(transfer_ctrl_t *const p_ctrl) |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |

## Field Documentation

### ◆ open

fsp_err_t(* transfer_api_t::open) (transfer_ctrl_t *const p_ctrl, transfer_cfg_t const *const p_cfg)

Initial configuration.

**Implemented as**

- R_DTC_Open()
- R_DMAC_Open()

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ reconfigure

fsp_err_t(* transfer_api_t::reconfigure) (transfer_ctrl_t *const p_ctrl, transfer_info_t *p_info)

Reconfigure the transfer. Enable the transfer if p_info is valid.

**Implemented as**

- R_DTC_Reconfigure()
- R_DMAC_Reconfigure()

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_info | Pointer to a new transfer info structure. |

### ◆ reset

fsp_err_t(* transfer_api_t::reset) (transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const num_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.

**Implemented as**

- R_DTC_Reset()
- R_DMAC_Reset()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in transfer_api_t::open call for this transfer. |
| [in] | p_src | Pointer to source. Set to NULL if source pointer should not change. |
| [in] | p_dest | Pointer to destination. Set to NULL if destination pointer should not change. |
| [in] | num_transfers | Transfer length in normal mode or number of blocks in block mode. In DMAC only, resets number of repeats (initially stored in transfer_info_t::num_blocks) in repeat mode. Not used in repeat mode for DTC. |

### ◆ enable

fsp_err_t(* transfer_api_t::enable) (transfer_ctrl_t *const p_ctrl)

Enable transfer. Transfers occur after the activation source event (or when transfer_api_t::softwareStart is called if ELC_EVENT_ELC_NONE is chosen as activation source).

**Implemented as**

- R_DTC_Enable()
- R_DMAC_Enable()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in transfer_api_t::open call for this transfer. |

### ◆ disable

fsp_err_t(* transfer_api_t::disable) (transfer_ctrl_t *const p_ctrl)

Disable transfer. Transfers do not occur after the activation source event (or when transfer_api_t::softwareStart is called if ELC_EVENT_ELC_NONE is chosen as the DMAC activation source).

***Note***

*If a transfer is in progress, it will be completed. Subsequent transfer requests do not cause a transfer.*

**Implemented as**

- R_DTC_Disable()
- R_DMAC_Disable()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in transfer_api_t::open call for this transfer. |

### ◆ softwareStart

fsp_err_t(* transfer_api_t::softwareStart) (transfer_ctrl_t *const p_ctrl, transfer_start_mode_t mode)

Start transfer in software.

**Warning**

Only works if ELC_EVENT_ELC_NONE is chosen as the DMAC activation source.

***Note***

*Not supported for DTC.*

**Implemented as**

- R_DMAC_SoftwareStart()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in transfer_api_t::open call for this transfer. |
| [in] | mode | Select mode from transfer_start_mode_t. |

### ◆ softwareStop

fsp_err_t(* transfer_api_t::softwareStop) (transfer_ctrl_t *const p_ctrl)

Stop transfer in software. The transfer will stop after completion of the current transfer.

***Note***

*Not supported for DTC.*
*Only applies for transfers started with TRANSFER_START_MODE_REPEAT.*

**Warning**

Only works if ELC_EVENT_ELC_NONE is chosen as the DMAC activation source.

**Implemented as**

- R_DMAC_SoftwareStop()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in transfer_api_t::open call for this transfer. |

### ◆ infoGet

fsp_err_t(* transfer_api_t::infoGet) (transfer_ctrl_t *const p_ctrl, transfer_properties_t *const p_properties)

Provides information about this transfer.

**Implemented as**

- R_DTC_InfoGet()
- R_DMAC_InfoGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in transfer_api_t::open call for this transfer. |
| [out] | p_properties | Driver specific information. |

### ◆ close

fsp_err_t(* transfer_api_t::close) (transfer_ctrl_t *const p_ctrl)

Releases hardware lock. This allows a transfer to be reconfigured using transfer_api_t::open.

**Implemented as**

- R_DTC_Close()
- R_DMAC_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in transfer_api_t::open call for this transfer. |

### ◆ versionGet

fsp_err_t(* transfer_api_t::versionGet) (fsp_version_t *const p_version)

Gets version and stores it in provided pointer p_version.

**Implemented as**

- R_DTC_VersionGet()
- R_DMAC_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version used. |

### ◆ transfer_instance_t

struct transfer_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| 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. |

## Typedef Documentation

### ◆ transfer_ctrl_t

typedef void transfer_ctrl_t

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

## Enumeration Type Documentation

### ◆ transfer_mode_t

| enum transfer_mode_t | |
|---|---|
| Transfer mode describes what will happen when a transfer request occurs. | |
| Enumerator | |
| 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. |
| 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. |

### ◆ transfer_size_t

| enum transfer_size_t | |
|---|---|
| Transfer size specifies the size of each individual transfer. Total transfer length = transfer_size_t * transfer_length_t | |
| Enumerator | |
| TRANSFER_SIZE_1_BYTE | Each transfer transfers a 8-bit value. |
| TRANSFER_SIZE_2_BYTE | Each transfer transfers a 16-bit value. |
| TRANSFER_SIZE_4_BYTE | Each transfer transfers a 32-bit value. |

### ◆ transfer_addr_mode_t

| enum transfer_addr_mode_t | |
|---|---|
| Address mode specifies whether to modify (increment or decrement) pointer after each transfer. | |
| Enumerator | |
| TRANSFER_ADDR_MODE_FIXED | Address pointer remains fixed after each transfer. |
| TRANSFER_ADDR_MODE_OFFSET | Offset is added to the address pointer 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. |

### ◆ transfer_repeat_area_t

| enum transfer_repeat_area_t | |
|---|---|
| Repeat area options (source or destination). In TRANSFER_MODE_REPEAT, the selected pointer returns to its original value after transfer_info_t::length transfers. In TRANSFER_MODE_BLOCK, the selected pointer returns to its original value after each transfer. | |
| Enumerator | |
| 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. |

### ◆ transfer_chain_mode_t

| enum transfer_chain_mode_t | |
|---|---|
| Chain transfer mode options.<br><br>***Note***<br>*Only applies for DTC.* | |
| Enumerator | |
| TRANSFER_CHAIN_MODE_DISABLED | Chain mode not used. |
| 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. |

### ◆ transfer_irq_t

| enum transfer_irq_t | |
|---|---|
| Interrupt options. | |
| Enumerator | |
| 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.<br><br>Warning<br>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.<br><br>*Note*<br>*Not available in all HAL drivers. See HAL driver for details.* |

### ◆ transfer_start_mode_t

| enum transfer_start_mode_t | |
|---|---|
| Select whether to start single or repeated transfer with software start. | |
| Enumerator | |
| TRANSFER_START_MODE_SINGLE | Software start triggers single transfer. |
| TRANSFER_START_MODE_REPEAT | Software start transfer continues until transfer is complete. |

## 4.3.35 UART Interface

Interfaces

### Detailed Description

Interface for UART communications.

# Summary

The UART interface provides common APIs for UART HAL drivers. The UART interface supports the following features:

- Full-duplex UART communication
- 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)

Implemented by:

- Serial Communications Interface (SCI) UART (r_sci_uart)

### Data Structures

| | |
|---|---|
| struct | uart_info_t |
| struct | uart_callback_args_t |
| struct | uart_cfg_t |
| struct | uart_api_t |
| struct | uart_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | uart_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | uart_event_t |
| enum | uart_data_bits_t |
| enum | uart_parity_t |
| enum | uart_stop_bits_t |
| enum | uart_dir_t |

## Data Structure Documentation

### ◆ uart_info_t

| struct uart_info_t |
|---|
| UART driver specific information |
| Data Fields |

| | | |
|---|---|---|
| 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. |

### ◆ uart_callback_args_t

| struct uart_callback_args_t | | |
|---|---|---|
| UART Callback parameter definition | | |
| Data Fields | | |
| 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. |

### ◆ uart_cfg_t

| struct uart_cfg_t | |
|---|---|
| UART Configuration | |
| **Data Fields** | |
| uint8_t | channel |
| | Select a channel corresponding to the channel number of the hardware. |
| | |
| 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) |
| | |
| uint8_t | rxi_ipl |
| | Receive interrupt priority. |
| | |
| IRQn_Type | rxi_irq |
| | Receive interrupt IRQ number. |
| | |
| uint8_t | txi_ipl |
| | Transmit interrupt priority. |
| | |
| IRQn_Type | txi_irq |
| | Transmit interrupt IRQ number. |
| | |
| uint8_t | tei_ipl |
| | Transmit end interrupt priority. |
| | |
| IRQn_Type | tei_irq |
| | Transmit end interrupt IRQ number. |
| | |
| uint8_t | eri_ipl |
| | Error interrupt priority. |
| | |
| IRQn_Type | eri_irq |
| | Error interrupt IRQ number. |
| | |

| | |
|---|---|
| transfer_instance_t const * | p_transfer_rx |
| | |
| transfer_instance_t const * | p_transfer_tx |
| | |
| 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. |
| | |

## Field Documentation

### ◆ p_transfer_rx

transfer_instance_t const* uart_cfg_t::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.

### ◆ p_transfer_tx

transfer_instance_t const* uart_cfg_t::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.

### ◆ uart_api_t

struct uart_api_t

Shared Interface definition for UART

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(uart_ctrl_t *const p_ctrl, uart_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | read )(uart_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| | |
| fsp_err_t(* | write )(uart_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t |

| | |
|---|---|
| | const bytes) |
| | |
| fsp_err_t(* | baudSet )(uart_ctrl_t *const p_ctrl, void const *const p_baudrate_info) |
| | |
| fsp_err_t(* | infoGet )(uart_ctrl_t *const p_ctrl, uart_info_t *const p_info) |
| | |
| fsp_err_t(* | communicationAbort )(uart_ctrl_t *const p_ctrl, uart_dir_t communication_to_abort) |
| | |
| fsp_err_t(* | close )(uart_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* uart_api_t::open) (uart_ctrl_t *const p_ctrl, uart_cfg_t const *const p_cfg)

Open UART device.

**Implemented as**

- R_SCI_UART_Open()

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to the UART control block. Must be declared by user. Value set here. |
| [in] | uart_cfg_t | Pointer to UART configuration structure. All elements of this structure must be set by user. |

### ◆ read

fsp_err_t(* uart_api_t::read) (uart_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

Read from UART device. The read buffer is used until the read is complete. 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_UART_Read()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the UART control block for the channel. |
| [in] | p_dest | Destination address to read data from. |
| [in] | bytes | Read data length. |

### ◆ write

fsp_err_t(* uart_api_t::write) (uart_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes)

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_UART_Write()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the UART control block. |
| [in] | p_src | Source address to write data to. |
| [in] | bytes | Write data length. |

### ◆ baudSet

fsp_err_t(* uart_api_t::baudSet) (uart_ctrl_t *const p_ctrl, void const *const p_baudrate_info)

Change baud rate.

**Warning**

Calling this API aborts any in-progress transmission and disables reception until the new baud settings have been applied.

**Implemented as**

- R_SCI_UART_BaudSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the UART control block. |
| [in] | p_baudrate_info | Pointer to module specific information for configuring baud rate. |

### ◆ infoGet

fsp_err_t(* uart_api_t::infoGet) (uart_ctrl_t *const p_ctrl, uart_info_t *const p_info)

Get the driver specific information.

**Implemented as**

- R_SCI_UART_InfoGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the UART control block. |
| [in] | baudrate | Baud rate in bps. |

### ◆ communicationAbort

fsp_err_t(* uart_api_t::communicationAbort) (uart_ctrl_t *const p_ctrl, uart_dir_t communication_to_abort)

Abort ongoing transfer.

**Implemented as**

- R_SCI_UART_Abort()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the UART control block. |
| [in] | communication_to_abort | Type of abort request. |

### ◆ close

| fsp_err_t(* uart_api_t::close) (uart_ctrl_t *const p_ctrl) |
|---|

Close UART device.

**Implemented as**

- R_SCI_UART_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the UART control block. |

### ◆ versionGet

| fsp_err_t(* uart_api_t::versionGet) (fsp_version_t *p_version) |
|---|

Get version.

**Implemented as**

- R_SCI_UART_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_version | Pointer to the memory to store the version information. |

### ◆ uart_instance_t

| struct uart_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| 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. |

## Typedef Documentation

### ◆ uart_ctrl_t

| typedef void uart_ctrl_t |
|---|
| UART control block. Allocate an instance specific control block to pass into the UART API calls.<br><br>**Implemented as**<br><br>◦ sci_uart_instance_ctrl_t |

## Enumeration Type Documentation

### ◆ uart_event_t

| enum uart_event_t | |
|---|---|
| UART Event codes | |
| Enumerator | |
| UART_EVENT_RX_COMPLETE | Receive complete event. |
| UART_EVENT_TX_COMPLETE | Transmit complete event. |
| UART_EVENT_RX_CHAR | Character received. |
| UART_EVENT_ERR_PARITY | Parity error event. |
| UART_EVENT_ERR_FRAMING | Mode fault error event. |
| UART_EVENT_ERR_OVERFLOW | FIFO Overflow error event. |
| UART_EVENT_BREAK_DETECT | Break detect error event. |
| UART_EVENT_TX_DATA_EMPTY | Last byte is transmitting, ready for more data. |

### ◆ uart_data_bits_t

| enum uart_data_bits_t | |
|---|---|
| UART Data bit length definition | |
| Enumerator | |
| UART_DATA_BITS_8 | Data bits 8-bit. |
| UART_DATA_BITS_7 | Data bits 7-bit. |
| UART_DATA_BITS_9 | Data bits 9-bit. |

### ◆ uart_parity_t

| enum uart_parity_t | |
|---|---|
| UART Parity definition | |
| Enumerator | |
| UART_PARITY_OFF | No parity. |
| UART_PARITY_EVEN | Even parity. |
| UART_PARITY_ODD | Odd parity. |

### ◆ uart_stop_bits_t

| enum uart_stop_bits_t | |
|---|---|
| UART Stop bits definition | |
| Enumerator | |
| UART_STOP_BITS_1 | Stop bit 1-bit. |
| UART_STOP_BITS_2 | Stop bits 2-bit. |

### ◆ uart_dir_t

| enum uart_dir_t | |
|---|---|
| UART transaction definition | |
| Enumerator | |
| UART_DIR_RX_TX | Both RX and TX. |
| UART_DIR_RX | Only RX. |
| UART_DIR_TX | Only TX. |

## 4.3.36 USB Interface

Interfaces

### Detailed Description

Interface for USB functions.

# Summary

The USB interface provides USB functionality.

The USB interface can be implemented by:

- USB (r_usb_basic)

### Data Structures

| | |
|---|---|
| struct | usb_api_t |
| struct | usb_instance_t |

### Macros

| | |
|---|---|
| #define | USB_API_VERSION_MINOR<br>Minor version of the API. |
| #define | USB_API_VERSION_MAJOR<br>Major version of the API. |
| #define | USB_BREQUEST<br>b15-8 |
| #define | USB_GET_STATUS<br>USB Standard request Get Status. |
| #define | USB_CLEAR_FEATURE<br>USB Standard request Clear Feature. |
| #define | USB_REQRESERVED<br>USB Standard request Reqreserved. |
| #define | USB_SET_FEATURE<br>USB Standard request Set Feature. |
| #define | USB_REQRESERVED1 |

| | |
|---|---|
| | USB Standard request Reqreserved1. |
| #define | USB_SET_ADDRESS<br>USB Standard request Set Address. |
| #define | USB_GET_DESCRIPTOR<br>USB Standard request Get Descriptor. |
| #define | USB_SET_DESCRIPTOR<br>USB Standard request Set Descriptor. |
| #define | USB_GET_CONFIGURATION<br>USB Standard request Get Configuration. |
| #define | USB_SET_CONFIGURATION<br>USB Standard request Set Configuration. |
| #define | USB_GET_INTERFACE<br>USB Standard request Get Interface. |
| #define | USB_SET_INTERFACE<br>USB Standard request Set Interface. |
| #define | USB_SYNCH_FRAME<br>USB Standard request Synch Frame. |
| #define | USB_HOST_TO_DEV<br>From host to device. |
| #define | USB_DEV_TO_HOST<br>From device to host. |
| #define | USB_STANDARD<br>Standard Request. |

| | |
|---|---|
| #define | USB_CLASS<br>Class Request. |
| #define | USB_VENDOR<br>Vendor Request. |
| #define | USB_DEVICE<br>Device. |
| #define | USB_INTERFACE<br>Interface. |
| #define | USB_ENDPOINT<br>End Point. |
| #define | USB_OTHER<br>Other. |
| #define | USB_NULL<br>NULL pointer. |
| #define | USB_IP0<br>USB0 module. |
| #define | USB_IP1<br>USB1 module. |
| #define | USB_PIPE0<br>Pipe Number0. |
| #define | USB_PIPE1<br>Pipe Number1. |

#define USB_PIPE2

Pipe Number2.

#define USB_PIPE3

Pipe Number3.

#define USB_PIPE4

Pipe Number4.

#define USB_PIPE5

Pipe Number5.

#define USB_PIPE6

Pipe Number6.

#define USB_PIPE7

Pipe Number7.

#define USB_PIPE8

Pipe Number8.

#define USB_PIPE9

Pipe Number9.

#define USB_EP0

End Point Number0.

#define USB_EP1

End Point Number1.

#define USB_EP2

End Point Number2.

#define USB_EP3

| | |
|---|---|
| | End Point Number3. |
| #define | USB_EP4<br>End Point Number4. |
| #define | USB_EP5<br>End Point Number5. |
| #define | USB_EP6<br>End Point Number6. |
| #define | USB_EP7<br>End Point Number7. |
| #define | USB_EP8<br>End Point Number8. |
| #define | USB_EP9<br>End Point Number9. |
| #define | USB_EP10<br>End Point Number10. |
| #define | USB_EP11<br>End Point Number11. |
| #define | USB_EP12<br>End Point Number12. |
| #define | USB_EP13<br>End Point Number13. |
| #define | USB_EP14<br>End Point Number14. |

| | |
|---|---|
| #define | USB_EP15<br>End Point Number15. |
| #define | USB_DT_DEVICE<br>Device Descriptor. |
| #define | USB_DT_CONFIGURATION<br>Configuration Descriptor. |
| #define | USB_DT_STRING<br>String Descriptor. |
| #define | USB_DT_INTERFACE<br>Interface Descriptor. |
| #define | USB_DT_ENDPOINT<br>Endpoint Descriptor. |
| #define | USB_DT_DEVICE_QUALIFIER<br>Device Qualifier Descriptor. |
| #define | USB_DT_OTHER_SPEED_CONF<br>Other Speed Configuration Descriptor. |
| #define | USB_DT_INTERFACE_POWER<br>Interface Power Descriptor. |
| #define | USB_DT_OTGDESCRIPTOR<br>OTG Descriptor. |
| #define | USB_DT_HUBDESCRIPTOR<br>HUB descriptor. |

| | |
|---|---|
| #define | USB_IFCLS_NOT<br>Un corresponding Class. |
| #define | USB_IFCLS_AUD<br>Audio Class. |
| #define | USB_IFCLS_CDC<br>CDC Class. |
| #define | USB_IFCLS_CDCC<br>CDC-Control Class. |
| #define | USB_IFCLS_HID<br>HID Class. |
| #define | USB_IFCLS_PHY<br>Physical Class. |
| #define | USB_IFCLS_IMG<br>Image Class. |
| #define | USB_IFCLS_PRN<br>Printer Class. |
| #define | USB_IFCLS_MAS<br>Mass Storage Class. |
| #define | USB_IFCLS_HUB<br>HUB Class. |
| #define | USB_IFCLS_CDCD<br>CDC-Data Class. |
| #define | USB_IFCLS_CHIP |

| | |
|---|---|
| | Chip/Smart Card Class. |
| #define | USB_IFCLS_CNT<br>Content-Security Class. |
| #define | USB_IFCLS_VID<br>Video Class. |
| #define | USB_IFCLS_DIAG<br>Diagnostic Device. |
| #define | USB_IFCLS_WIRE<br>Wireless Controller. |
| #define | USB_IFCLS_APL<br>Application-Specific. |
| #define | USB_IFCLS_VEN<br>Vendor-Specific Class. |
| #define | USB_EP_IN<br>In Endpoint. |
| #define | USB_EP_OUT<br>Out Endpoint. |
| #define | USB_EP_ISO<br>Isochronous Transfer. |
| #define | USB_EP_BULK<br>Bulk Transfer. |
| #define | USB_EP_INT<br>Interrupt Transfer. |

| | |
|---|---|
| #define | USB_CF_RESERVED<br>Reserved(set to 1) |
| #define | USB_CF_SELFP<br>Self Powered. |
| #define | USB_CF_BUSP<br>Bus Powered. |
| #define | USB_CF_RWUPON<br>Remote Wake up ON. |
| #define | USB_CF_RWUPOFF<br>Remote Wake up OFF. |
| #define | USB_DD_BLENGTH<br>Device Descriptor Length. |
| #define | USB_CD_BLENGTH<br>Configuration Descriptor Length. |
| #define | USB_ID_BLENGTH<br>Interface Descriptor Length. |
| #define | USB_ED_BLENGTH<br>Endpoint Descriptor Length. |

## Enumerations

| | |
|---|---|
| enum | usb_speed_t |
| enum | usb_setup_status_t |
| enum | usb_status_t |
| enum | usb_class_t |

| | |
|---|---|
| enum | usb_bcport_t |
| enum | usb_onoff_t |
| enum | usb_transfer_t |
| enum | usb_transfer_type_t |
| enum | usb_mode_t |
| enum | usb_compliancetest_status_t |

## Data Structure Documentation

### ◆ usb_api_t

| struct usb_api_t | |
|---|---|
| Functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(usb_ctrl_t *const p_api_ctrl, usb_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(usb_ctrl_t *const p_api_ctrl) |
| | |
| fsp_err_t(* | read )(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t destination) |
| | |
| fsp_err_t(* | write )(usb_ctrl_t *const p_api_ctrl, uint8_t const *const p_buf, uint32_t size, uint8_t destination) |
| | |
| fsp_err_t(* | stop )(usb_ctrl_t *const p_api_ctrl, usb_transfer_t direction, uint8_t destination) |
| | |
| fsp_err_t(* | suspend )(usb_ctrl_t *const p_api_ctrl) |
| | |
| fsp_err_t(* | resume )(usb_ctrl_t *const p_api_ctrl) |
| | |
| fsp_err_t(* | vbusSet )(usb_ctrl_t *const p_api_ctrl, uint16_t state) |
| | |
| fsp_err_t(* | infoGet )(usb_ctrl_t *const p_api_ctrl, usb_info_t *p_info, uint8_t destination) |

| | |
|---|---|
| fsp_err_t(* | pipeRead )(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number) |
| fsp_err_t(* | pipeWrite )(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number) |
| fsp_err_t(* | pipeStop )(usb_ctrl_t *const p_api_ctrl, uint8_t pipe_number) |
| fsp_err_t(* | usedPipesGet )(usb_ctrl_t *const p_api_ctrl, uint16_t *p_pipe, uint8_t destination) |
| fsp_err_t(* | pipeInfoGet )(usb_ctrl_t *const p_api_ctrl, usb_pipe_t *p_info, uint8_t pipe_number) |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| fsp_err_t(* | eventGet )(usb_ctrl_t *const p_api_ctrl, usb_status_t *event) |
| fsp_err_t(* | callback )(usb_callback_t *p_callback) |
| fsp_err_t(* | pullUp )(usb_ctrl_t *const p_api_ctrl, uint8_t state) |
| fsp_err_t(* | hostControlTransfer )(usb_ctrl_t *const p_api_ctrl, usb_setup_t *p_setup, uint8_t *p_buf, uint8_t device_address) |
| fsp_err_t(* | periControlDataGet )(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size) |
| fsp_err_t(* | periControlDataSet )(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size) |
| fsp_err_t(* | periControlStatusSet )(usb_ctrl_t *const p_api_ctrl, usb_setup_status_t status) |
| fsp_err_t(* | remoteWakeup )(usb_ctrl_t *const p_api_ctrl) |

| | |
|---|---|
| fsp_err_t(* | moduleNumberGet )(usb_ctrl_t *const p_api_ctrl, uint8_t *module_number) |
| | |
| fsp_err_t(* | classTypeGet )(usb_ctrl_t *const p_api_ctrl, usb_class_t *class_type) |
| | |
| fsp_err_t(* | deviceAddressGet )(usb_ctrl_t *const p_api_ctrl, uint8_t *device_address) |
| | |
| fsp_err_t(* | pipeNumberGet )(usb_ctrl_t *const p_api_ctrl, uint8_t *pipe_number) |
| | |
| fsp_err_t(* | deviceStateGet )(usb_ctrl_t *const p_api_ctrl, uint16_t *state) |
| | |
| fsp_err_t(* | dataSizeGet )(usb_ctrl_t *const p_api_ctrl, uint32_t *data_size) |
| | |
| fsp_err_t(* | setupGet )(usb_ctrl_t *const p_api_ctrl, usb_setup_t *setup) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* usb_api_t::open) (usb_ctrl_t *const p_api_ctrl, usb_cfg_t const *const p_cfg)

Start the USB module

**Implemented as**

- R_USB_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* usb_api_t::close) (usb_ctrl_t *const p_api_ctrl)

Stop the USB module

**Implemented as**

- R_USB_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |

### ◆ read

fsp_err_t(* usb_api_t::read) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t destination)

Request USB data read

**Implemented as**

- R_USB_Read()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | p_buf | Pointer to area that stores read data. |
| [in] | size | Read request size. |
| [in] | destination | In Host mode, it represents the device address, and in Peripheral mode, it represents the device class. |

### ◆ write

fsp_err_t(* usb_api_t::write) (usb_ctrl_t *const p_api_ctrl, uint8_t const *const p_buf, uint32_t size, uint8_t destination)

Request USB data write

**Implemented as**

- R_USB_Write()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | p_buf | Pointer to area that stores write data. |
| [in] | size | Read request size. |
| [in] | destination | In Host mode, it represents the device address, and in Peripheral mode, it represents the device class. |

### ◆ stop

fsp_err_t(* usb_api_t::stop) (usb_ctrl_t *const p_api_ctrl, usb_transfer_t direction, uint8_t destination)

Stop USB data read/write processing

**Implemented as**

- R_USB_Stop()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | direction | Receive (USB_TRANSFER_READ) or send (USB_TRANSFER_WRITE). |
| [in] | destination | In Host mode, it represents the device address, and in Peripheral mode, it represents the device class. |

### ◆ suspend

fsp_err_t(* usb_api_t::suspend) (usb_ctrl_t *const p_api_ctrl)

Request suspend

**Implemented as**

- R_USB_Suspend()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |

### ◆ resume

fsp_err_t(* usb_api_t::resume) (usb_ctrl_t *const p_api_ctrl)

Request resume

**Implemented as**

- R_USB_Resume()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |

### ◆ vbusSet

fsp_err_t(* usb_api_t::vbusSet) (usb_ctrl_t *const p_api_ctrl, uint16_t state)

Sets VBUS supply start/stop.

**Implemented as**

- R_USB_VbusSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | state | VBUS supply start/stop specification |

### ◆ infoGet

fsp_err_t(* usb_api_t::infoGet) (usb_ctrl_t *const p_api_ctrl, usb_info_t *p_info, uint8_t destination)

Get information on USB device.

**Implemented as**

- R_USB_InfoGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | p_info | Pointer to usb_info_t structure area. |
| [in] | destination | Device address for Host. |

### ◆ pipeRead

fsp_err_t(* usb_api_t::pipeRead) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number)

Request data read from specified pipe

**Implemented as**

- R_USB_PipeRead()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | p_buf | Pointer to area that stores read data. |
| [in] | size | Read request size. |
| [in] | pipe_number | Pipe Number. |

### ◆ pipeWrite

fsp_err_t(* usb_api_t::pipeWrite) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number)

Request data write to specified pipe

**Implemented as**

- R_USB_PipeWrite()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | p_buf | Pointer to area that stores write data. |
| [in] | size | Read request size. |
| [in] | pipe_number | Pipe Number. |

### ◆ pipeStop

fsp_err_t(* usb_api_t::pipeStop) (usb_ctrl_t *const p_api_ctrl, uint8_t pipe_number)

Stop USB data read/write processing to specified pipe

**Implemented as**

- R_USB_PipeStop()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | pipe_number | Pipe Number. |

### ◆ usedPipesGet

fsp_err_t(* usb_api_t::usedPipesGet) (usb_ctrl_t *const p_api_ctrl, uint16_t *p_pipe, uint8_t destination)

Get pipe number

**Implemented as**

- R_USB_UsedPipesGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | p_pipe | Pointer to area that stores the selected pipe number (bit map information). |
| [in] | destination | Device address for Host. |

### ◆ pipeInfoGet

fsp_err_t(* usb_api_t::pipeInfoGet) (usb_ctrl_t *const p_api_ctrl, usb_pipe_t *p_info, uint8_t pipe_number)

Get pipe information

**Implemented as**

- R_USB_PipeInfoGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | p_info | Pointer to usb_pipe_t structure area. |
| [in] | pipe_number | Pipe Number. |

### ◆ versionGet

fsp_err_t(* usb_api_t::versionGet) (fsp_version_t *const p_version)

Get the driver version

**Implemented as**

- R_USB_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Version number. |

### ◆ eventGet

fsp_err_t(* usb_api_t::eventGet) (usb_ctrl_t *const p_api_ctrl, usb_status_t *event)

Return USB-related completed events (OS less only)

**Implemented as**

- R_USB_EventGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [out] | event | Pointer to event. |

### ◆ callback

fsp_err_t(* usb_api_t::callback) (usb_callback_t *p_callback)

Register a callback function to be called upon completion of a USB related event. (RTOS only)

**Implemented as**

- R_USB_Callback()

**Parameters**

| | | |
|---|---|---|
| [in] | p_callback | Pointer to Callback function. |

### ◆ pullUp

fsp_err_t(* usb_api_t::pullUp) (usb_ctrl_t *const p_api_ctrl, uint8_t state)

Pull-up enable/disable setting of D+/D- line.

**Implemented as**

- R_USB_PullUp()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | state | Pull-up enable/disable setting. |

### ◆ hostControlTransfer

fsp_err_t(* usb_api_t::hostControlTransfer) (usb_ctrl_t *const p_api_ctrl, usb_setup_t *p_setup, uint8_t *p_buf, uint8_t device_address)

Performs settings and transmission processing when transmitting a setup packet.

**Implemented as**

- R_USB_HostControlTransfer()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | USB control structure. |
| [in] | p_setup | Setup packet information. |
| [in] | p_buf | Transfer area information. |
| [in] | device_address | Device address information. |

### ◆ periControlDataGet

fsp_err_t(* usb_api_t::periControlDataGet) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size)

Receives data sent by control transfer.

**Implemented as**

- R_USB_PeriControlDataGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | USB control structure. |
| [in] | p_buf | Data reception area information. |
| [in] | size | Data reception size information. |

### ◆ periControlDataSet

fsp_err_t(* usb_api_t::periControlDataSet) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size)

Performs transfer processing for control transfer.

**Implemented as**

- R_USB_PeriControlDataSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | USB control structure. |
| [in] | p_buf | Area information for data transfer. |
| [in] | size | Transfer size information. |

### ◆ periControlStatusSet

fsp_err_t(* usb_api_t::periControlStatusSet) (usb_ctrl_t *const p_api_ctrl, usb_setup_status_t status)

Set the response to the setup packet.

**Implemented as**

- R_USB_PeriControlStatusSet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | USB control structure. |
| [in] | status | USB port startup information. |

### ◆ remoteWakeup

fsp_err_t(* usb_api_t::remoteWakeup) (usb_ctrl_t *const p_api_ctrl)

Sends a remote wake-up signal to the connected Host.

**Implemented as**

- R_USB_RemoteWakeup()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | USB control structure. |

### ◆ moduleNumberGet

fsp_err_t(* usb_api_t::moduleNumberGet) (usb_ctrl_t *const p_api_ctrl, uint8_t *module_number)

This API gets the module number.

**Implemented as**

- R_USB_ModuleNumberGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | USB control structure. |
| [out] | module_number | Module number to get. |

### ◆ classTypeGet

fsp_err_t(* usb_api_t::classTypeGet) (usb_ctrl_t *const p_api_ctrl, usb_class_t *class_type)

This API gets the module number.

**Implemented as**

- R_USB_ClassTypeGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | USB control structure. |
| [out] | class_type | Class type to get. |

### ◆ deviceAddressGet

fsp_err_t(* usb_api_t::deviceAddressGet) (usb_ctrl_t *const p_api_ctrl, uint8_t *device_address)

This API gets the device address.

**Implemented as**

- R_USB_DeviceAddressGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | USB control structure. |
| [out] | device_address | Device address to get. |

### ◆ pipeNumberGet

fsp_err_t(* usb_api_t::pipeNumberGet) (usb_ctrl_t *const p_api_ctrl, uint8_t *pipe_number)

This API gets the pipe number.

**Implemented as**

- R_USB_PipeNumberGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | USB control structure. |
| [out] | pipe_number | Pipe number to get. |

### ◆ deviceStateGet

fsp_err_t(* usb_api_t::deviceStateGet) (usb_ctrl_t *const p_api_ctrl, uint16_t *state)

This API gets the state of the device.

**Implemented as**

- R_USB_DeviceStateGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | USB control structure. |
| [out] | state | Device state to get. |

### ◆ dataSizeGet

fsp_err_t(* usb_api_t::dataSizeGet) (usb_ctrl_t *const p_api_ctrl, uint32_t *data_size)

This API gets the data size.

**Implemented as**

- R_USB_DataSizeGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | USB control structure. |
| [out] | data_size | Data size to get. |

### ◆ setupGet

fsp_err_t(* usb_api_t::setupGet) (usb_ctrl_t *const p_api_ctrl, usb_setup_t *setup)

This API gets the setup type.

**Implemented as**

- R_USB_SetupGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | USB control structure. |
| [out] | setup | Setup type to get. |

### ◆ usb_instance_t

struct usb_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| usb_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| usb_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| usb_api_t const * | p_api | Pointer to the API structure for this instance. |

## Enumeration Type Documentation

### ◆ usb_speed_t

| enum usb_speed_t | |
|---|---|
| USB speed type | |
| Enumerator | |
| USB_SPEED_LS | Low speed operation. |
| USB_SPEED_FS | Full speed operation. |
| USB_SPEED_HS | Hi speed operation. |

### ◆ usb_setup_status_t

| enum usb_setup_status_t | |
|---|---|
| USB request result | |
| Enumerator | |
| USB_SETUP_STATUS_ACK | ACK response. |
| USB_SETUP_STATUS_STALL | STALL response. |

### ◆ usb_status_t

| enum usb_status_t | |
|---|---|
| USB driver status | |
| Enumerator | |
| USB_STATUS_POWERED | Powered State. |
| USB_STATUS_DEFAULT | Default State. |
| USB_STATUS_ADDRESS | Address State. |
| USB_STATUS_CONFIGURED | Configured State. |
| USB_STATUS_SUSPEND | Suspend State. |
| USB_STATUS_RESUME | Resume State. |
| USB_STATUS_DETACH | Detach State. |
| USB_STATUS_REQUEST | Request State. |
| USB_STATUS_REQUEST_COMPLETE | Request Complete State. |
| USB_STATUS_READ_COMPLETE | Read Complete State. |
| USB_STATUS_WRITE_COMPLETE | Write Complete State. |
| USB_STATUS_BC | battery Charge State |
| USB_STATUS_OVERCURRENT | Over Current state. |
| USB_STATUS_NOT_SUPPORT | Device Not Support. |
| USB_STATUS_NONE | None Status. |
| USB_STATUS_MSC_CMD_COMPLETE | MSC_CMD Complete. |

### ◆ usb_class_t

| enum usb_class_t | |
|---|---|
| USB class type | |
| Enumerator | |
| USB_CLASS_PCDC | PCDC Class. |
| USB_CLASS_PCDCC | PCDCC Class. |
| USB_CLASS_PHID | PHID Class. |
| USB_CLASS_PVND | PVND Class. |
| USB_CLASS_HCDC | HCDC Class. |
| USB_CLASS_HCDCC | HCDCC Class. |
| USB_CLASS_HHID | HHID Class. |
| USB_CLASS_HVND | HVND Class. |
| USB_CLASS_HMSC | HMSC Class. |
| USB_CLASS_PMSC | PMSC Class. |
| USB_CLASS_REQUEST | USB Class Request. |
| USB_CLASS_END | USB Class End Code. |

### ◆ usb_bcport_t

| enum usb_bcport_t | |
|---|---|
| USB battery charging type | |
| Enumerator | |
| USB_BCPORT_SDP | SDP port settings. |
| USB_BCPORT_CDP | CDP port settings. |
| USB_BCPORT_DCP | DCP port settings. |

### ◆ usb_onoff_t

| enum usb_onoff_t | |
|---|---|
| USB status | |
| Enumerator | |
| USB_OFF | USB Off State. |
| USB_ON | USB On State. |

### ◆ usb_transfer_t

| enum usb_transfer_t | |
|---|---|
| USB read/write type | |
| Enumerator | |
| USB_TRANSFER_READ | Data Receive communication. |
| USB_TRANSFER_WRITE | Data transmission communication. |

### ◆ usb_transfer_type_t

| enum usb_transfer_type_t | |
|---|---|
| USB transfer type | |
| Enumerator | |
| USB_TRANSFER_TYPE_BULK | Bulk communication. |
| USB_TRANSFER_TYPE_INT | Interrupt communication. |
| USB_TRANSFER_TYPE_ISO | Isochronous communication. |

### ◆ usb_mode_t

| enum usb_mode_t | |
|---|---|
| Enumerator | |
| USB_MODE_HOST | Host mode. |
| USB_MODE_PERI | Peripheral mode. |

### ◆ usb_compliancetest_status_t

| enum usb_compliancetest_status_t | |
|---|---|
| Enumerator | |
| USB_COMPLIANCETEST_ATTACH | Device Attach Detection. |
| USB_COMPLIANCETEST_DETACH | Device Detach Detection. |
| USB_COMPLIANCETEST_TPL | TPL device connect. |
| USB_COMPLIANCETEST_NOTTPL | Not TPL device connect. |
| USB_COMPLIANCETEST_HUB | USB Hub connect. |
| USB_COMPLIANCETEST_OVRC | Over current. |
| USB_COMPLIANCETEST_NORES | Response Time out for Control Read Transfer. |
| USB_COMPLIANCETEST_SETUP_ERR | Setup Transaction Error. |

## 4.3.37 USB HCDC Interface

Interfaces

### Detailed Description

Interface for USB HCDC functions.

# Summary

The USB HCDC interface provides USB HCDC functionality.

The USB HCDC interface can be implemented by:

- USB Host Communications Device Class Driver (r_usb_hcdc)

## 4.3.38 USB HHID Interface

Interfaces

### Detailed Description

Interface for USB HHID functions.

## Summary

The USB HHID interface provides USB HHID functionality.

The USB HHID interface can be implemented by:

- USB Host Human Interface Device Class Driver (r_usb_hhid)

### Data Structures

| | |
|---|---|
| struct | usb_hhid_api_t |

### Macros

| | |
|---|---|
| #define | USB_HID_OTHER<br>Other. |
| #define | USB_HID_KEYBOARD<br>Keyboard. |
| #define | USB_HID_MOUSE<br>Mouse. |
| #define | USB_HID_IN<br>In Transfer. |
| #define | USB_HID_OUT<br>Out Transfer. |

### Data Structure Documentation

#### ◆ usb_hhid_api_t

| struct usb_hhid_api_t | |
|---|---|
| USB HHID functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | typeGet )(usb_ctrl_t *const p_api_ctrl, uint8_t *p_type, uint8_t device_address) |

| | |
|---|---|
| fsp_err_t(* | maxPacketSizeGet )(usb_ctrl_t *const p_api_ctrl, uint16_t *p_size, uint8_t direction, uint8_t device_address) |

## Field Documentation

### ◆ typeGet

fsp_err_t(* usb_hhid_api_t::typeGet) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_type, uint8_t device_address)

Get HID protocol.(USB Mouse/USB Keyboard/Other Type.)

**Implemented as**

- R_USB_HHID_TypeGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | p_type | Pointer to store HID protocol value. |
| [in] | device_address | Device Address. |

### ◆ maxPacketSizeGet

fsp_err_t(* usb_hhid_api_t::maxPacketSizeGet) (usb_ctrl_t *const p_api_ctrl, uint16_t *p_size, uint8_t direction, uint8_t device_address)

Obtains max packet size for the connected HID device. The max packet size is set to the area. Set the direction (USB_HID_IN/USB_HID_OUT).

**Implemented as**

- R_USB_HHID_MaxPacketSizeGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | p_size | Pointer to the area to store the max package size. |
| [in] | direction | Transfer direction. |
| [in] | device_address | Device Address. |

## 4.3.39 USB HMSC Interface

Interfaces

### Detailed Description

Interface for USB HMSC functions.

# Summary

The USB HMSC interface provides USB HMSC functionality.

The USB HMSC interface can be implemented by:

- USB Host Mass Storage Class Driver (r_usb_hmsc)

### Data Structures

| | |
|---|---|
| struct | usb_hmsc_api_t |

### Enumerations

| | |
|---|---|
| enum | usb_atapi_t |
| enum | usb_csw_result_t |

### Data Structure Documentation

#### ◆ usb_hmsc_api_t

| struct usb_hmsc_api_t | |
|---|---|
| USB HMSC functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | storageCommand )(usb_ctrl_t *const p_api_ctrl, uint8_t *buf, uint8_t command, uint8_t destination) |
| | |
| fsp_err_t(* | driveNumberGet )(usb_ctrl_t *const p_api_ctrl, uint8_t *p_drive, uint8_t destination) |
| | |
| fsp_err_t(* | storageReadSector )(uint16_t drive_number, uint8_t *const buff, uint32_t sector_number, uint16_t sector_count) |
| | |
| fsp_err_t(* | storageWriteSector )(uint16_t drive_number, uint8_t const *const buff, uint32_t sector_number, uint16_t sector_count) |
| | |
| fsp_err_t(* | semaphoreGet )(void) |
| | |
| fsp_err_t(* | semaphoreRelease )(void) |
| | |

## Field Documentation

### ◆ storageCommand

fsp_err_t(* usb_hmsc_api_t::storageCommand) (usb_ctrl_t *const p_api_ctrl, uint8_t *buf, uint8_t command, uint8_t destination)

Processing for MassStorage(ATAPI) command.

**Implemented as**

- R_USB_HMSC_StorageCommand()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | *buf | Pointer to the buffer area to store the transfer data. |
| [in] | command | ATAPI command. |
| [in] | destination | Represents a device address. |

### ◆ driveNumberGet

fsp_err_t(* usb_hmsc_api_t::driveNumberGet) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_drive, uint8_t destination)

Get number of Storage drive.

**Implemented as**

- R_USB_HMSC_DriveNumberGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [out] | p_drive | Store address for Drive No. |
| [in] | destination | Represents a device address. |

### ◆ storageReadSector

fsp_err_t(* usb_hmsc_api_t::storageReadSector) (uint16_t drive_number, uint8_t *const buff, uint32_t sector_number, uint16_t sector_count)

Read sector information.

**Implemented as**

- R_USB_HMSC_StorageReadSector()

**Parameters**

| | | |
|---|---|---|
| [in] | drive_number | Drive number. |
| [out] | *buff | Pointer to the buffer area to store the transfer data. |
| [in] | sector_number | The sector number to start with. |
| [in] | sector_count | Transmit with the sector size of the number of times. |

### ◆ storageWriteSector

fsp_err_t(* usb_hmsc_api_t::storageWriteSector) (uint16_t drive_number, uint8_t const *const buff, uint32_t sector_number, uint16_t sector_count)

Write sector information.

**Implemented as**

- R_USB_HMSC_StorageWriteSector()

**Parameters**

| | | |
|---|---|---|
| [in] | drive_number | Drive number. |
| [in] | *buff | Pointer to the buffer area to store the transfer data. |
| [in] | sector_number | The sector number to start with. |
| [in] | sector_count | Transmit with the sector size of the number of times. |

### ◆ semaphoreGet

fsp_err_t(* usb_hmsc_api_t::semaphoreGet) (void)

Get Semaphore.

**Implemented as**

- R_USB_HMSC_SemaphoreGet()

### ◆ semaphoreRelease

fsp_err_t(* usb_hmsc_api_t::semaphoreRelease) (void)

Release Semaphore.

**Implemented as**

- R_USB_HMSC_SemaphoreRelease()

## Enumeration Type Documentation

### ◆ usb_atapi_t

| enum usb_atapi_t | |
|---|---|
| Enumerator | |
| USB_ATAPI_TEST_UNIT_READY | ATAPI command Test Unit Ready. |
| USB_ATAPI_REQUEST_SENSE | ATAPI command Request Sense. |
| USB_ATAPI_FORMAT_UNIT | ATAPI command Format Unit. |
| USB_ATAPI_INQUIRY | ATAPI command Inquiry. |
| USB_ATAPI_MODE_SELECT6 | ATAPI command Mode Select6. |
| USB_ATAPI_MODE_SENSE6 | ATAPI command Mode Sense6. |
| USB_ATAPI_START_STOP_UNIT | ATAPI command Start Stop Unit. |
| USB_ATAPI_PREVENT_ALLOW | ATAPI command Prevent Allow. |
| USB_ATAPI_READ_FORMAT_CAPACITY | ATAPI command Read Format Capacity. |
| USB_ATAPI_READ_CAPACITY | ATAPI command Read Capacity. |
| USB_ATAPI_READ10 | ATAPI command Read10. |
| USB_ATAPI_WRITE10 | ATAPI command Write10. |
| USB_ATAPI_SEEK | ATAPI command Seek. |
| USB_ATAPI_WRITE_AND_VERIFY | ATAPI command Write and Verify. |
| USB_ATAPI_VERIFY10 | ATAPI command Verify10. |
| USB_ATAPI_MODE_SELECT10 | ATAPI command Mode Select10. |
| USB_ATAPI_MODE_SENSE10 | ATAPI command Mode Sense10. |

### ◆ usb_csw_result_t

| enum usb_csw_result_t | |
|---|---|
| Enumerator | |
| USB_CSW_RESULT_SUCCESS | CSW was successful. |
| USB_CSW_RESULT_FAIL | CSW failed. |
| USB_CSW_RESULT_PHASE | CSW has phase error. |

## 4.3.40 USB PCDC Interface

Interfaces

### Detailed Description

Interface for USB PCDC functions.

# Summary

The USB interface provides USB functionality.

The USB PCDC interface can be implemented by:

- USB Peripheral Communication Device Class (r_usb_pcdc)

### Macros

| | |
|---|---|
| #define | USB_PCDC_SET_LINE_CODING<br>Command code for Set Line Coding. |
| #define | USB_PCDC_GET_LINE_CODING<br>Command code for Get Line Coding. |
| #define | USB_PCDC_SET_CONTROL_LINE_STATE<br>Command code for Control Line State. |
| #define | USB_PCDC_SERIAL_STATE<br>Serial State Code. |

| | |
|---|---|
| #define | USB_PCDC_SETUP_TBL_BSIZE<br>Setup packet table size (uint16_t * 5) |

### 4.3.41 USB PHID Interface

Interfaces

#### Detailed Description

Interface for USB PHID functions.

## Summary

The USB interface provides USB functionality.

The USB PHID interface can be implemented by:

- USB Peripheral Human Interface Device Class (r_usb_phid)

### 4.3.42 USB PMSC Interface

Interfaces

#### Detailed Description

Interface for USB PMSC functions.

## Summary

The USB PMSC interface provides USB PMSC functionality.

The USB PMSC interface can be implemented by:

- USB Peripheral Mass Storage Class (r_usb_pmsc)

### 4.3.43 WDT Interface

Interfaces

### Detailed Description

Interface for watch dog timer functions.

# Summary

The WDT interface for the Watchdog Timer (WDT) peripheral provides watchdog functionality including resetting the device or generating an interrupt.

The watchdog timer interface can be implemented by:

- Watchdog Timer (r_wdt)
- Independent Watchdog Timer (r_iwdt)

### Data Structures

| | |
|---|---|
| struct | wdt_callback_args_t |
| struct | wdt_timeout_values_t |
| struct | wdt_cfg_t |
| struct | wdt_api_t |
| struct | wdt_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | wdt_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | wdt_timeout_t |
| enum | wdt_clock_division_t |
| enum | wdt_window_start_t |
| enum | wdt_window_end_t |
| enum | wdt_reset_control_t |
| enum | wdt_stop_control_t |
| enum | wdt_status_t |

## Data Structure Documentation

### ◆ wdt_callback_args_t

struct wdt_callback_args_t

| Callback function parameter data | | |
|---|---|---|
| Data Fields | | |
| void const * | p_context | Placeholder for user data. Set in wdt_api_t::open function in wdt_cfg_t. |

### ◆ wdt_timeout_values_t

| struct wdt_timeout_values_t | | |
|---|---|---|
| WDT timeout data. Used to return frequency of WDT clock and timeout period | | |
| Data Fields | | |
| uint32_t | clock_frequency_hz | Frequency of watchdog clock after divider. |
| uint32_t | timeout_clocks | Timeout period in units of watchdog clock ticks. |

### ◆ wdt_cfg_t

| struct wdt_cfg_t | |
|---|---|
| WDT configuration parameters. | |
| **Data Fields** | |
| 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 |
| | |
| void const * | p_extend |
| | Placeholder for user extension. |
| | |

## Field Documentation

### ◆ p_context

void const* wdt_cfg_t::p_context

Placeholder for user data. Passed to the user callback in wdt_callback_args_t.

### ◆ wdt_api_t

struct wdt_api_t

WDT functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | refresh )(wdt_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | statusGet )(wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status) |
| | |
| fsp_err_t(* | statusClear )(wdt_ctrl_t *const p_ctrl, const wdt_status_t status) |
| | |
| fsp_err_t(* | counterGet )(wdt_ctrl_t *const p_ctrl, uint32_t *const p_count) |
| | |
| fsp_err_t(* | timeoutGet )(wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout) |

| | |
|---|---|
| fsp_err_t(* | versionGet )(fsp_version_t *const p_data) |

## Field Documentation

### ◆ open

fsp_err_t(* wdt_api_t::open) (wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)

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()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ refresh

fsp_err_t(* wdt_api_t::refresh) (wdt_ctrl_t *const p_ctrl)

Refresh the watchdog timer.

**Implemented as**

- R_WDT_Refresh()
- R_IWDT_Refresh()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ statusGet

fsp_err_t(* wdt_api_t::statusGet) (wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)

Read the status of the WDT.

**Implemented as**

- R_WDT_StatusGet()
- R_IWDT_StatusGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_status | Pointer to variable to return status information through. |

### ◆ statusClear

fsp_err_t(* wdt_api_t::statusClear) (wdt_ctrl_t *const p_ctrl, const wdt_status_t status)

Clear the status flags of the WDT.

**Implemented as**

- R_WDT_StatusClear()
- R_IWDT_StatusClear()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | status | Status condition(s) to clear. |

### ◆ counterGet

fsp_err_t(* wdt_api_t::counterGet) (wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)

Read the current WDT counter value.

**Implemented as**

- R_WDT_CounterGet()
- R_IWDT_CounterGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_count | Pointer to variable to return current WDT counter value. |

### ◆ timeoutGet

fsp_err_t(* wdt_api_t::timeoutGet) (wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout)

Read the watchdog timeout values.

**Implemented as**

- R_WDT_TimeoutGet()
- R_IWDT_TimeoutGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_timeout | Pointer to structure to return timeout values. |

### ◆ versionGet

fsp_err_t(* wdt_api_t::versionGet) (fsp_version_t *const p_data)

Return the version of the driver.

**Implemented as**

- R_WDT_VersionGet()
- R_IWDT_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_data | Memory address to return version information to. |

### ◆ wdt_instance_t

struct wdt_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| 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. |

## Typedef Documentation

### ◆ wdt_ctrl_t

| typedef void wdt_ctrl_t |
|---|
| WDT control block. Allocate an instance specific control block to pass into the WDT API calls.<br><br>**Implemented as**<br><br>◦ wdt_instance_ctrl_t<br>◦ iwdt_instance_ctrl_t |

## Enumeration Type Documentation

### ◆ wdt_timeout_t

| enum wdt_timeout_t | |
|---|---|
| WDT time-out periods. | |
| Enumerator | |
| 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 |

### ◆ wdt_clock_division_t

| enum wdt_clock_division_t | |
|---|---|
| WDT clock division ratio. | |
| Enumerator | |
| 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. |

### ◆ wdt_window_start_t

| enum wdt_window_start_t | |
|---|---|
| WDT refresh permitted period window start position. | |
| Enumerator | |
| 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%. |

### ◆ wdt_window_end_t

| enum wdt_window_end_t | |
|---|---|
| WDT refresh permitted period window end position. | |
| Enumerator | |
| 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%. |

### ◆ wdt_reset_control_t

| enum wdt_reset_control_t | |
|---|---|
| WDT Counter underflow and refresh error control. | |
| Enumerator | |
| WDT_RESET_CONTROL_NMI | NMI request when counter underflows. |
| WDT_RESET_CONTROL_RESET | Reset request when counter underflows. |

### ◆ wdt_stop_control_t

| enum wdt_stop_control_t | |
|---|---|
| WDT Counter operation in sleep mode. | |
| Enumerator | |
| 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. |

### ◆ wdt_status_t

| enum wdt_status_t | |
|---|---|
| WDT status | |
| Enumerator | |
| 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 window. |
| WDT_STATUS_UNDERFLOW_AND_REFRESH_ERROR | Underflow and refresh error flags set. |

## 4.3.44 BLE ABS Interface

Interfaces

### Detailed Description

Interface for Bluetooth Low Energy Abstraction functions.

# Summary

The BLE ABS interface for the Bluetooth Low Energy Abstraction (BLE ABS) peripheral provides Bluetooth Low Energy Abstraction functionality.

The Bluetooth Low Energy Abstraction interface can be implemented by:

- Bluetooth Low Energy Abstraction (rm_ble_abs)

### Data Structures

| | |
|---|---|
| struct | ble_device_address_t |
| struct | ble_gap_connection_parameter_t |
| struct | ble_gap_connection_phy_parameter_t |
| struct | ble_gap_scan_phy_parameter_t |
| struct | ble_gap_scan_on_t |

| | |
|---|---|
| struct | ble_abs_callback_args_t |
| struct | ble_abs_pairing_parameter_t |
| struct | ble_abs_gatt_server_callback_set_t |
| struct | ble_abs_gatt_client_callback_set_t |
| struct | ble_abs_legacy_advertising_parameter_t |
| struct | ble_abs_extend_advertising_parameter_t |
| struct | ble_abs_non_connectable_advertising_parameter_t |
| struct | ble_abs_periodic_advertising_parameter_t |
| struct | ble_abs_scan_phy_parameter_t |
| struct | ble_abs_scan_parameter_t |
| struct | ble_abs_connection_phy_parameter_t |
| struct | ble_abs_connection_parameter_t |
| struct | ble_abs_cfg_t |
| struct | ble_abs_api_t |
| struct | ble_abs_instance_t |

## Macros

| | |
|---|---|
| #define | BLE_ABS_ADVERTISING_PHY_LEGACY<br>Non-Connectable Legacy Advertising phy setting. |

## Typedefs

| | |
|---|---|
| typedef void(* | ble_gap_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t *p_event_data) |
| typedef void(* | ble_vendor_specific_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_vs_evt_data_t *p_event_data) |
| typedef void(* | ble_gatt_server_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_gatts_evt_data_t *p_event_data) |
| typedef void(* | ble_gatt_client_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_gattc_evt_data_t *p_event_data) |

| | |
|---|---|
| typedef void | ble_abs_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | ble_abs_advertising_filter_t |

## Data Structure Documentation

### ◆ ble_device_address_t

| struct ble_device_address_t | | |
|---|---|---|
| st_ble_device_address is the type of bluetooth device address(BD_ADDR). | | |
| Data Fields | | |
| uint8_t | addr[BLE_BD_ADDR_LEN] | bluetooth device address. |
| uint8_t | type | the type of bluetooth device address. |

### ◆ ble_gap_connection_parameter_t

| struct ble_gap_connection_parameter_t | | |
|---|---|---|
| ble_gap_connection_parameter_t is Connection parameters included in connection interval, slave latency, supervision timeout, ce length. | | |
| Data Fields | | |
| uint16_t | conn_intv_min | Minimum connection interval. |
| uint16_t | conn_intv_max | Maximum connection interval. |
| uint16_t | conn_latency | Slave latency. |
| uint16_t | sup_to | Supervision timeout. |
| uint16_t | min_ce_length | Minimum CE Length. |
| uint16_t | max_ce_length | Maximum CE Length. |

### ◆ ble_gap_connection_phy_parameter_t

| struct ble_gap_connection_phy_parameter_t | | |
|---|---|---|
| ble_gap_connection_phy_parameter_t is Connection parameters per PHY. | | |
| Data Fields | | |
| uint16_t | scan_intv | Scan interval. |
| uint16_t | scan_window | Scan window. |
| ble_gap_connection_parameter_t * | p_conn_param | Connection interval, slave latency, supervision timeout, and CE length. |

### ◆ ble_gap_scan_phy_parameter_t

| struct ble_gap_scan_phy_parameter_t |
|---|

| Scan parameters per scan PHY. | | |
|---|---|---|
| Data Fields | | |
| uint8_t | scan_type | Scan type. |
| uint16_t | scan_intv | Scan interval. |
| uint16_t | scan_window | Scan window. |

### ◆ ble_gap_scan_on_t

| struct ble_gap_scan_on_t | | |
|---|---|---|
| Parameters configured when scanning starts. | | |
| Data Fields | | |
| uint8_t | proc_type | Procedure type. |
| uint8_t | filter_dups | Filter duplicates. |
| uint16_t | duration | Scan duration. |
| uint16_t | period | Scan period. |

### ◆ ble_abs_callback_args_t

| struct ble_abs_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| uint32_t | channel | Select a channel corresponding to the channel number of the hardware. |
| ble_event_cb_t | ble_abs_event | The event can be used to identify what caused the callback. |
| void const * | p_context | Placeholder for user data. Set in ble_abs_api_t::open function in ble_abs_cfg_t. |

### ◆ ble_abs_pairing_parameter_t

| struct ble_abs_pairing_parameter_t | | |
|---|---|---|
| st_ble_abs_pairing_parameter_t includes the pairing parameters. | | |
| Data Fields | | |
| uint8_t | io_capabilitie_local_device | IO capabilities of local device. |
| uint8_t | mitm_protection_policy | MITM protection policy. |
| uint8_t | secure_connection_only | Determine whether to accept only Secure Connections or not. |
| uint8_t | local_key_distribute | Type of keys to be distributed from local device. |

| | | |
|---|---|---|
| uint8_t | remote_key_distribute | Type of keys which local device requests a remote device to distribute. |
| uint8_t | maximum_key_size | Maximum LTK size. |
| uint8_t | padding[2] | padding |

### ◆ ble_abs_gatt_server_callback_set_t

| struct ble_abs_gatt_server_callback_set_t | | |
|---|---|---|
| GATT Server callback function and the priority. | | |
| Data Fields | | |
| ble_gatt_server_application_callback_t | gatt_server_callback_function | GATT Server callback function. |
| uint8_t | gatt_server_callback_priority | The priority number of GATT Server callback function. |

### ◆ ble_abs_gatt_client_callback_set_t

| struct ble_abs_gatt_client_callback_set_t | | |
|---|---|---|
| GATT Client callback function and the priority. | | |
| Data Fields | | |
| ble_gatt_client_application_callback_t | gatt_client_callback_function | GATT Client callback function. |
| uint8_t | gatt_client_callback_priority | The priority number of GATT Client callback function. |

### ◆ ble_abs_legacy_advertising_parameter_t

| struct ble_abs_legacy_advertising_parameter_t | | |
|---|---|---|
| st_ble_abs_legacy_advertising_parameter_t is the parameters for legacy advertising. | | |
| Data Fields | | |
| ble_device_address_t * | p_peer_address | Peer address. |
| uint8_t * | p_advertising_data | Advertising data. If p_advertising_data is specified as NULL, advertising data is not set. |
| uint8_t * | p_scan_response_data | Scan response data. If p_scan_response_data is specified as NULL, scan response data is not set. |
| uint32_t | fast_advertising_interval | Fast advertising interval. |
| uint32_t | slow_advertising_interval | Slow advertising interval. |
| uint16_t | fast_advertising_period | Fast advertising period. |
| | | |

| | | |
|---|---|---|
| uint16_t | slow_advertising_period | Slow advertising period. |
| uint16_t | advertising_data_length | Advertising data length (in bytes). |
| uint16_t | scan_response_data_length | Scan response data length (in bytes). |
| uint8_t | advertising_channel_map | Channel Map. |
| uint8_t | advertising_filter_policy | Advertising Filter Policy. |
| uint8_t | own_bluetooth_address_type | Own Bluetooth address type. |
| uint8_t | own_bluetooth_address[6] | Own Bluetooth address. |
| uint8_t | padding[3] | padding |

### ◆ ble_abs_extend_advertising_parameter_t

| struct ble_abs_extend_advertising_parameter_t | | |
|---|---|---|
| st_ble_abs_extend_advertising_parameter_t is the parameters for extended advertising. | | |
| Data Fields | | |
| ble_device_address_t * | p_peer_address | Peer address. |
| uint8_t * | p_advertising_data | Advertising data. If p_advertising_data is specified as NULL, advertising data is not set. |
| uint32_t | fast_advertising_interval | Fast advertising interval. |
| uint32_t | slow_advertising_interval | Slow advertising interval. |
| uint16_t | fast_advertising_period | Fast advertising period. |
| uint16_t | slow_advertising_period | Slow advertising period. |
| uint16_t | advertising_data_length | Advertising data length (in bytes). |
| uint8_t | advertising_channel_map | Channel Map. |
| uint8_t | advertising_filter_policy | Advertising Filter Policy. |
| uint8_t | own_bluetooth_address_type | Own Bluetooth address type. |
| uint8_t | own_bluetooth_address[6] | Own Bluetooth address. |
| uint8_t | primary_advertising_phy | Primary ADV PHY. |
| uint8_t | secondary_advertising_phy | Secondary ADV Max Skip. |
| uint8_t | padding[3] | padding |

### ◆ ble_abs_non_connectable_advertising_parameter_t

| struct ble_abs_non_connectable_advertising_parameter_t |
|---|
| st_ble_abs_non_connectable_advertising_parameter_t is the parameters for non-connectable advertising. |

| Data Fields | | |
|---|---|---|
| ble_device_address_t * | p_peer_address | Peer address.<br><br>Peer address. |
| uint8_t * | p_advertising_data | Advertising data. If p_advertising_data is specified as NULL, advertising data is not set. |
| uint32_t | advertising_interval | Advertising interval. |
| uint16_t | advertising_duration | Advertising duration. |
| uint16_t | advertising_data_length | Advertising data length (in bytes). |
| uint8_t | advertising_channel_map | Channel Map. |
| uint8_t | own_bluetooth_address_type | Own Bluetooth address type. |
| uint8_t | own_bluetooth_address[6] | Own Bluetooth address. |
| uint8_t | primary_advertising_phy | Primary ADV PHY. |
| uint8_t | secondary_advertising_phy | Secondary ADV Max Skip. |
| uint8_t | padding[2] | padding |

### ◆ ble_abs_periodic_advertising_parameter_t

| struct ble_abs_periodic_advertising_parameter_t | | |
|---|---|---|
| st_ble_abs_periodic_advertising_parameter_t is the parameters for periodic advertising. | | |
| Data Fields | | |
| ble_abs_non_connectable_advertising_parameter_t | advertising_parameter | Advertising parameters. |
| uint8_t * | p_periodic_advertising_data | Periodic advertising data. If p_perd_adv_data is specified as NULL, periodic advertising data is not set. |
| uint16_t | periodic_advertising_interval | Periodic advertising interval. Time(ms) = perd_intv_min * 1.25. |
| uint16_t | periodic_advertising_data_length | Periodic advertising data length (in bytes). |

### ◆ ble_abs_scan_phy_parameter_t

| struct ble_abs_scan_phy_parameter_t | | |
|---|---|---|
| st_ble_abs_scan_phy_parameter_t is the phy parameters for scan. | | |
| Data Fields | | |
| uint16_t | fast_scan_interval | Fast scan interval. Interval(ms) = fast_scan_interval * 0.625. |

| | | |
|---|---|---|
| uint16_t | slow_scan_interval | Slow scan interval. Interval(ms) = slow_scan_interval * 0.625. |
| uint16_t | fast_scan_window | Fast scan window. Window(ms) = fast_scan_window * 0.625. |
| uint16_t | slow_scan_window | Slow scan window. Window(ms) = slow_scan_window * 0.625. |
| uint8_t | scan_type | Scan type. |
| uint8_t | padding[3] | padding. |

### ◆ ble_abs_scan_parameter_t

| struct ble_abs_scan_parameter_t | | |
|---|---|---|
| st_ble_abs_scan_parameter_t is the parameters for scan. | | |
| Data Fields | | |
| ble_abs_scan_phy_parameter_t * | p_phy_parameter_1M | 1M phy parameters for scan. |
| ble_abs_scan_phy_parameter_t * | p_phy_parameter_coded | Coded phy parameters for scan. |
| uint8_t * | p_filter_data | Filter data. |
| uint16_t | fast_scan_period | Fast scan period (in sec). |
| uint16_t | slow_scan_period | Slow scan period (in sec). |
| uint16_t | filter_data_length | Filter data length. |
| uint8_t | device_scan_filter_policy | Scan Filter Policy. |
| uint8_t | filter_duplicate | Filter duplicates. |
| uint8_t | filter_ad_type | The AD type of the data specified by the p_filter_data parameter. |
| uint8_t | padding[3] | Padding. |

### ◆ ble_abs_connection_phy_parameter_t

| struct ble_abs_connection_phy_parameter_t | | |
|---|---|---|
| st_ble_abs_connection_phy_parameter_t is the phy parameters for create connection. | | |
| Data Fields | | |
| uint16_t | connection_interval | Connection interval. Time(ms) = connection_interval * 1.25. |
| uint16_t | connection_slave_latency | Slave latency. |
| uint16_t | supervision_timeout | Supervision timeout. |
| uint8_t | padding[2] | Padding. |

### ◆ ble_abs_connection_parameter_t

| struct ble_abs_connection_parameter_t | | |
|---|---|---|
| st_ble_abs_connection_parameter_t is the parameters for create connection. | | |
| Data Fields | | |
| ble_abs_connection_phy_parameter_t * | p_connection_phy_parameter_1M | 1M phy parameters for create connection. |
| ble_abs_connection_phy_parameter_t * | p_connection_phy_parameter_2M | 2M phy parameters for create connection. |
| ble_abs_connection_phy_parameter_t * | p_connection_phy_parameter_coded | Coded phy parameters for create connection. |
| ble_device_address_t * | p_device_address | Remote device address. |
| uint8_t | filter_parameter | The filter parameter specifies whether the White List is used or not. |
| uint8_t | connection_timeout | GAP Event parameters. This parameter differs in each GAP Event. |
| uint8_t | padding[2] | Padding. |

### ◆ ble_abs_cfg_t

| struct ble_abs_cfg_t | |
|---|---|
| BLE ABS configuration parameters. | |
| **Data Fields** | |
| uint32_t | channel |
| | Select a channel corresponding to the channel number of the hardware. More... |
| | |
| ble_gap_application_callback_t | gap_callback |
| | GAP callback function. |
| | |
| ble_vendor_specific_application_callback_t | vendor_specific_callback |
| | Vendor Specific callback function. |
| | |
| ble_abs_gatt_server_callback_set_t * | p_gatt_server_callback_list |
| | GATT Server callback set. |

| | |
|---|---|
| | |
| uint8_t | gatt_server_callback_list_number |
| | The number of GATT Server callback functions. |
| | |
| ble_abs_gatt_client_callback_set_t * | p_gatt_client_callback_list |
| | GATT Client callback set. |
| | |
| uint8_t | gatt_client_callback_list_number |
| | The number of GATT Client callback functions. |
| | |
| ble_abs_pairing_parameter_t * | p_pairing_parameter |
| | Pairing parameters. |
| | |
| flash_instance_t const * | p_flash_instance |
| | Pointer to flash instance. |
| | |
| timer_instance_t const * | p_timer_instance |
| | Pointer to timer instance. |
| | |
| void(* | p_callback )(ble_abs_callback_args_t *p_args) |
| | Callback provided when a BLE ISR occurs. |
| | |
| void const * | p_context |
| | Placeholder for user data. Passed to the user callback in ble_abs_callback_args_t. |
| | |
| void const * | p_extend |
| | Placeholder for user extension. |

## Field Documentation

### ◆ channel

uint32_t ble_abs_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

the parameters for initialization.

### ◆ ble_abs_api_t

struct ble_abs_api_t

BLE ABS functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(ble_abs_ctrl_t *const p_ctrl, ble_abs_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(ble_abs_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | reset )(ble_abs_ctrl_t *const p_ctrl, ble_event_cb_t init_callback) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_data) |
| | |
| fsp_err_t(* | startLegacyAdvertising )(ble_abs_ctrl_t *const p_ctrl, ble_abs_legacy_advertising_parameter_t const *const p_advertising_parameter) |
| | |
| fsp_err_t(* | startExtendedAdvertising )(ble_abs_ctrl_t *const p_ctrl, ble_abs_extend_advertising_parameter_t const *const p_advertising_parameter) |
| | |
| fsp_err_t(* | startNonConnectableAdvertising )(ble_abs_ctrl_t *const p_ctrl, ble_abs_non_connectable_advertising_parameter_t const *const p_advertising_parameter) |
| | |
| fsp_err_t(* | startPeriodicAdvertising )(ble_abs_ctrl_t *const p_ctrl, ble_abs_periodic_advertising_parameter_t const *const p_advertising_parameter) |
| | |
| fsp_err_t(* | startScanning )(ble_abs_ctrl_t *const p_ctrl, ble_abs_scan_parameter_t const *const p_scan_parameter) |

| | |
|---|---|
| fsp_err_t(* | createConnection )(ble_abs_ctrl_t *const p_ctrl, ble_abs_connection_parameter_t const *const p_connection_parameter) |
| fsp_err_t(* | setLocalPrivacy )(ble_abs_ctrl_t *const p_ctrl, uint8_t const *const p_lc_irk, uint8_t privacy_mode) |
| fsp_err_t(* | startAuthentication )(ble_abs_ctrl_t *const p_ctrl, uint16_t connection_handle) |

## Field Documentation

### ◆ open

fsp_err_t(* ble_abs_api_t::open) (ble_abs_ctrl_t *const p_ctrl, ble_abs_cfg_t const *const p_cfg)

Initialize the BLE ABS in register start mode.

**Implemented as**

- RM_BLE_ABS_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ close

fsp_err_t(* ble_abs_api_t::close) (ble_abs_ctrl_t *const p_ctrl)

Close the BLE ABS.

**Implemented as**

- RM_BLE_ABS_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ reset

fsp_err_t(* ble_abs_api_t::reset) (ble_abs_ctrl_t *const p_ctrl, ble_event_cb_t init_callback)

Close the BLE ABS.

**Implemented as**

- RM_BLE_ABS_Reset()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | init_callback | callback function to initialize Host Stack. |

### ◆ versionGet

fsp_err_t(* ble_abs_api_t::versionGet) (fsp_version_t *const p_data)

Return the version of the driver.

**Implemented as**

- RM_BLE_ABS_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [out] | p_data | Memory address to return version information to. |

### ◆ startLegacyAdvertising

fsp_err_t(* ble_abs_api_t::startLegacyAdvertising) (ble_abs_ctrl_t *const p_ctrl, ble_abs_legacy_advertising_parameter_t const *const p_advertising_parameter)

Start Legacy Connectable Advertising.

**Implemented as**

- RM_BLE_ABS_StartLegacyAdvertising()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_advertising_parameter | Pointer to Advertising parameters for Legacy Advertising. |

### ◆ startExtendedAdvertising

fsp_err_t(* ble_abs_api_t::startExtendedAdvertising) (ble_abs_ctrl_t *const p_ctrl, ble_abs_extend_advertising_parameter_t const *const p_advertising_parameter)

Start Extended Connectable Advertising.

**Implemented as**

- RM_BLE_ABS_StartExtendedAdvertising()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_advertising_parameter | Pointer to Advertising parameters for extend Advertising. |

### ◆ startNonConnectableAdvertising

fsp_err_t(* ble_abs_api_t::startNonConnectableAdvertising) (ble_abs_ctrl_t *const p_ctrl, ble_abs_non_connectable_advertising_parameter_t const *const p_advertising_parameter)

Start Non-Connectable Advertising.

**Implemented as**

- RM_BLE_ABS_StartNonConnectableAdvertising()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_advertising_parameter | Pointer to Advertising parameters for non-connectable Advertising. |

### ◆ startPeriodicAdvertising

fsp_err_t(* ble_abs_api_t::startPeriodicAdvertising) (ble_abs_ctrl_t *const p_ctrl, ble_abs_periodic_advertising_parameter_t const *const p_advertising_parameter)

Start Periodic Advertising.

**Implemented as**

- RM_BLE_ABS_StartPeriodicAdvertising()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_advertising_parameter | Pointer to Advertising parameters for periodic Advertising. |

### ◆ startScanning

fsp_err_t(* ble_abs_api_t::startScanning) (ble_abs_ctrl_t *const p_ctrl, ble_abs_scan_parameter_t const *const p_scan_parameter)

Start scanning.

**Implemented as**

- RM_BLE_ABS_StartScanning()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_scan_parameter | Pointer to scan parameter. |

### ◆ createConnection

fsp_err_t(* ble_abs_api_t::createConnection) (ble_abs_ctrl_t *const p_ctrl, ble_abs_connection_parameter_t const *const p_connection_parameter)

Request create connection.

**Implemented as**

- RM_BLE_ABS_CreateConnection()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_connection_parameter | Pointer to connection parameter. |

### ◆ setLocalPrivacy

fsp_err_t(* ble_abs_api_t::setLocalPrivacy) (ble_abs_ctrl_t *const p_ctrl, uint8_t const *const p_lc_irk, uint8_t privacy_mode)

Configure local device privacy.

**Implemented as**

- RM_BLE_ABS_SetLocalPrivacy()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_lc_irk | Pointer to IRK to be registered in the resolving list. |
| [in] | privacy_mode | privacy_mode privacy mode. |

### ◆ startAuthentication

fsp_err_t(* ble_abs_api_t::startAuthentication) (ble_abs_ctrl_t *const p_ctrl, uint16_t connection_handle)

Start pairing or encryption.

**Implemented as**

- RM_BLE_ABS_StartAuthentication()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | connection_handle | Connection handle identifying the remote device. |

### ◆ ble_abs_instance_t

struct ble_abs_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| ble_abs_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| ble_abs_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| ble_abs_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ ble_gap_application_callback_t

typedef void(* ble_gap_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t *p_event_data)

ble_gap_application_callback_t is the GAP Event callback function type.

### ◆ ble_vendor_specific_application_callback_t

typedef void(* ble_vendor_specific_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_vs_evt_data_t *p_event_data)

ble_vendor_specific_application_callback_t is the Vendor Specific Event callback function type.

### ◆ ble_gatt_server_application_callback_t

| |
|---|
| typedef void(* ble_gatt_server_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_gatts_evt_data_t *p_event_data) |
| ble_gatt_server_application_callback_t is the GATT Server Event callback function type. |

### ◆ ble_gatt_client_application_callback_t

| |
|---|
| typedef void(* ble_gatt_client_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_gattc_evt_data_t *p_event_data) |
| ble_gatt_client_application_callback_t is the GATT Server Event callback function type. |

### ◆ ble_abs_ctrl_t

| |
|---|
| typedef void ble_abs_ctrl_t |
| BLE ABS control block. Allocate an instance specific control block to pass into the BLE ABS API calls.<br><br>**Implemented as**<br><br>◦ ble_abs_instance_ctrl_t |

#### Enumeration Type Documentation

### ◆ ble_abs_advertising_filter_t

| | |
|---|---|
| enum ble_abs_advertising_filter_t | |
| Advertising Filter Policy | |
| Enumerator | |
| BLE_ABS_ADVERTISING_FILTER_ALLOW_ANY | Receive a connect request from all devices. |
| BLE_ABS_ADVERTISING_FILTER_ALLOW_WHITE_LIST | Receive a connect request from only the devices registered in White List. |

## 4.3.45 Block Media Interface

Interfaces

#### Detailed Description

Interface for block media memory access.

# Summary

The block media interface supports reading, writing, and erasing media devices. All functions are non-blocking if possible. The callback is used to determine when an operation completes.

Implemented by:

- SD/MMC Block Media Implementation (rm_block_media_sdmmc)
- USB HMSC Block Media Implementation (rm_block_media_usb)

### Data Structures

| | |
|---|---|
| struct | rm_block_media_info_t |
| struct | rm_block_media_callback_args_t |
| struct | rm_block_media_cfg_t |
| struct | rm_block_media_status_t |
| struct | rm_block_media_api_t |
| struct | rm_block_media_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | rm_block_media_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_block_media_event_t |

## Data Structure Documentation

### ◆ rm_block_media_info_t

| struct rm_block_media_info_t | | |
|---|---|---|
| Block media device information supported by the instance | | |
| Data Fields | | |
| uint32_t | sector_size_bytes | Sector size in bytes. |
| uint32_t | num_sectors | Total number of sectors. |
| bool | reentrant | True if connected block media driver is reentrant. |

### ◆ rm_block_media_callback_args_t

| struct rm_block_media_callback_args_t |
|---|

| Callback function parameter data | | |
|---|---|---|
| Data Fields | | |
| rm_block_media_event_t | event | The event can be used to identify what caused the callback. |
| void const * | p_context | Placeholder for user data. |

### ◆ rm_block_media_cfg_t

| struct rm_block_media_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| uint32_t | block_size |
| | Block size, must be a power of 2 multiple of sector_size_bytes. |
| | |
| void(* | p_callback )(rm_block_media_callback_args_t *p_args) |
| | Pointer to callback function. |
| | |
| void const * | p_context |
| | User defined context passed into callback function. |
| | |
| void const * | p_extend |
| | Extension parameter for hardware specific settings. |
| | |

### ◆ rm_block_media_status_t

| struct rm_block_media_status_t | | |
|---|---|---|
| Current status | | |
| Data Fields | | |
| bool | initialized | False if rm_block_media_api_t::mediaInit has not been called since media was inserted, true otherwise. |
| bool | busy | True if media is busy with a previous write/erase operation. |
| | | |

| | | |
|---|---|---|
| bool | media_inserted | Media insertion status, true if media is not removable. |

### ◆ rm_block_media_api_t

struct rm_block_media_api_t

Block media interface API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | mediaInit )(rm_block_media_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | read )(rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks) |
| | |
| fsp_err_t(* | write )(rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks) |
| | |
| fsp_err_t(* | erase )(rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks) |
| | |
| fsp_err_t(* | statusGet )(rm_block_media_ctrl_t *const p_ctrl, rm_block_media_status_t *const p_status) |
| | |
| fsp_err_t(* | infoGet )(rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info) |
| | |
| fsp_err_t(* | close )(rm_block_media_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_block_media_api_t::open) (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg)

Initialize block media device. rm_block_media_api_t::mediaInit must be called to complete the intitialization procedure.

**Implemented as**

- RM_BLOCK_MEDIA_SDMMC_Open
- RM_BLOCK_MEDIA_USB_Open

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ mediaInit

fsp_err_t(* rm_block_media_api_t::mediaInit) (rm_block_media_ctrl_t *const p_ctrl)

Initializes a media device. If the device is removable, it must be plugged in prior to calling this API. This function blocks until media initialization is complete.

**Implemented as**

- RM_BLOCK_MEDIA_SDMMC_MediaInit
- RM_BLOCK_MEDIA_USB_MediaInit

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_block_media_api_t::open call. |

### ◆ read

fsp_err_t(* rm_block_media_api_t::read) (rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks)

Reads blocks of data from the specified memory device address to the location specified by the caller.

**Implemented as**

- RM_BLOCK_MEDIA_SDMMC_Read
- RM_BLOCK_MEDIA_USB_Read

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_block_media_api_t::open call. |
| [out] | p_dest_address | Destination to read the data into. |
| [in] | block_address | Block address to read the data from. |
| [in] | num_blocks | Number of blocks of data to read. |

### ◆ write

fsp_err_t(* rm_block_media_api_t::write) (rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks)

Writes blocks of data to the specified device memory address.

**Implemented as**

- RM_BLOCK_MEDIA_SDMMC_Write
- RM_BLOCK_MEDIA_USB_Write

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_block_media_api_t::open call. |
| [in] | p_src_address | Address to read the data to be written. |
| [in] | block_address | Block address to write the data to. |
| [in] | num_blocks | Number of blocks of data to write. |

### ◆ erase

fsp_err_t(* rm_block_media_api_t::erase) (rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks)

Erases blocks of data from the memory device.

**Implemented as**

- RM_BLOCK_MEDIA_SDMMC_Erase
- RM_BLOCK_MEDIA_USB_Erase

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_block_media_api_t::open call. |
| [in] | block_address | Block address to start the erase process at. |
| [in] | num_blocks | Number of blocks of data to erase. |

### ◆ statusGet

fsp_err_t(* rm_block_media_api_t::statusGet) (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_status_t *const p_status)

Get status of connected device.

**Implemented as**

- RM_BLOCK_MEDIA_SDMMC_StatusGet
- RM_BLOCK_MEDIA_USB_StatusGet

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_block_media_api_t::open call. |
| [out] | p_status | Pointer to store current status. |

### ◆ infoGet

fsp_err_t(* rm_block_media_api_t::infoGet) (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info)

Returns information about the block media device.

**Implemented as**

- RM_BLOCK_MEDIA_SDMMC_InfoGet
- RM_BLOCK_MEDIA_USB_InfoGet

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_block_media_api_t::open call. |
| [out] | p_info | Pointer to information structure. All elements of this structure will be set by the function. |

### ◆ close

fsp_err_t(* rm_block_media_api_t::close) (rm_block_media_ctrl_t *const p_ctrl)

Closes the module.

**Implemented as**

- RM_BLOCK_MEDIA_SDMMC_Close
- RM_BLOCK_MEDIA_USB_Close

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_block_media_api_t::open call. |

### ◆ versionGet

fsp_err_t(* rm_block_media_api_t::versionGet) (fsp_version_t *const p_version)

Gets version and stores it in provided pointer p_version.

**Implemented as**

- RM_BLOCK_MEDIA_SDMMC_VersionGet
- RM_BLOCK_MEDIA_USB_VersionGet

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version used. |

### ◆ rm_block_media_instance_t

struct rm_block_media_instance_t

| This structure encompasses everything that is needed to use an instance of this interface. | | |
|---|---|---|
| Data Fields | | |
| rm_block_media_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_block_media_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_block_media_api_t const * | p_api | Pointer to the API structure for this instance. |

### Typedef Documentation

#### ◆ rm_block_media_ctrl_t

typedef void rm_block_media_ctrl_t

Block media API control block. Allocate an instance specific control block to pass into the block media API calls.

**Implemented as**

- rm_block_media_sdmmc_instance_ctrl_t
- rm_block_media_usb_instance_ctrl_t

### Enumeration Type Documentation

#### ◆ rm_block_media_event_t

enum rm_block_media_event_t

Events that can trigger a callback function

| Enumerator | |
|---|---|
| RM_BLOCK_MEDIA_EVENT_MEDIA_REMOVED | Media removed event. |
| RM_BLOCK_MEDIA_EVENT_MEDIA_INSERTED | Media inserted event. |
| RM_BLOCK_MEDIA_EVENT_OPERATION_COMPLETE | Read, write, or erase completed. |
| RM_BLOCK_MEDIA_EVENT_ERROR | Media inserted event. |
| RM_BLOCK_MEDIA_EVENT_POLL_STATUS | Poll rm_block_media_api_t::statusGet for write/erase completion. |

## 4.3.46 FreeRTOS+FAT Port Interface

Interfaces

### Detailed Description

Interface for FreeRTOS+FAT port.

## Summary

The FreeRTOS+FAT port provides notifications for insertion and removal of removable media and provides initialization functions required by FreeRTOS+FAT.

The FreeRTOS+FAT interface can be implemented by: FreeRTOS+FAT Port (rm_freertos_plus_fat)

### Data Structures

| | |
|---|---|
| struct | rm_freertos_plus_fat_callback_args_t |
| struct | rm_freertos_plus_fat_device_t |
| struct | rm_freertos_plus_fat_api_t |
| struct | rm_freertos_plus_fat_instance_t |

### Enumerations

| | |
|---|---|
| enum | rm_freertos_plus_fat_event_t |
| enum | rm_freertos_plus_fat_type_t |

### Data Structure Documentation

#### ◆ rm_freertos_plus_fat_callback_args_t

| struct rm_freertos_plus_fat_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| rm_freertos_plus_fat_event_t | event | The event can be used to identify what caused the callback. |
| void const * | p_context | Placeholder for user data. |

#### ◆ rm_freertos_plus_fat_device_t

| struct rm_freertos_plus_fat_device_t | | |
|---|---|---|
| Information obtained from the media device. | | |
| Data Fields | | |
| uint32_t | sector_count | Sector count. |

| | | |
|---|---|---|
| uint32_t | sector_size_bytes | Sector size in bytes. |

### ◆ rm_freertos_plus_fat_api_t

struct rm_freertos_plus_fat_api_t

FreeRTOS plus Fat functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | mediaInit )(rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_device_t *const p_device) |
| | |
| fsp_err_t(* | diskInit )(rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_disk_cfg_t const *const p_disk_cfg, FF_Disk_t *const p_disk) |
| | |
| fsp_err_t(* | diskDeinit )(rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk) |
| | |
| fsp_err_t(* | infoGet )(rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk, rm_freertos_plus_fat_info_t *const p_info) |
| | |
| fsp_err_t(* | close )(rm_freertos_plus_fat_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_freertos_plus_fat_api_t::open) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_cfg_t const *const p_cfg)

Open media device.

**Implemented as**

- RM_FREERTOS_PLUS_FAT_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ mediaInit

fsp_err_t(* rm_freertos_plus_fat_api_t::mediaInit) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_device_t *const p_device)

Initializes a media device. If the device is removable, it must be plugged in prior to calling this API. This function blocks until media initialization is complete.

**Implemented as**

- RM_FREERTOS_PLUS_FAT_MediaInit

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_device | Pointer to store device information. |

### ◆ diskInit

fsp_err_t(* rm_freertos_plus_fat_api_t::diskInit) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_disk_cfg_t const *const p_disk_cfg, FF_Disk_t *const p_disk)

Initializes a FreeRTOS+FAT FF_Disk_t structure.

**Implemented as**

- RM_FREERTOS_PLUS_FAT_DiskInit

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_disk_cfg | Pointer to disk configurations |
| [out] | p_disk | Pointer to store FreeRTOS+FAT disk structure. |

### ◆ diskDeinit

fsp_err_t(* rm_freertos_plus_fat_api_t::diskDeinit) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk)

Deinitializes a FreeRTOS+FAT FF_Disk_t structure.

**Implemented as**

- RM_FREERTOS_PLUS_FAT_DiskDeinit

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_disk_cfg | Pointer to disk configurations |
| [out] | p_disk | Pointer to store FreeRTOS+FAT disk structure. |

### ◆ infoGet

fsp_err_t(* rm_freertos_plus_fat_api_t::infoGet) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk, rm_freertos_plus_fat_info_t *const p_info)

Returns information about the media device.

**Implemented as**

- RM_FREERTOS_PLUS_FAT_InfoGet

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_info | Pointer to information structure. All elements of this structure will be set by the function. |

### ◆ close

fsp_err_t(* rm_freertos_plus_fat_api_t::close) (rm_freertos_plus_fat_ctrl_t *const p_ctrl)

Close media device.

**Implemented as**

- RM_FREERTOS_PLUS_FAT_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ versionGet

| fsp_err_t(* rm_freertos_plus_fat_api_t::versionGet) (fsp_version_t *const p_version) |
|---|

Get the driver version.

**Implemented as**

- RM_FREERTOS_PLUS_FAT_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ rm_freertos_plus_fat_instance_t

struct rm_freertos_plus_fat_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| rm_freertos_plus_fat_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_freertos_plus_fat_cfg_t const *const | p_cfg | Pointer to the configuration structure for this instance. |
| rm_freertos_plus_fat_api_t const * | p_api | Pointer to the API structure for this instance. |

## Enumeration Type Documentation

### ◆ rm_freertos_plus_fat_event_t

enum rm_freertos_plus_fat_event_t

Events that can trigger a callback function

| Enumerator | |
|---|---|
| RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_REMOVED | Media removed event. |
| RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED | Media inserted event. |

### ◆ rm_freertos_plus_fat_type_t

| enum rm_freertos_plus_fat_type_t | |
|---|---|
| Enumerator | |
| RM_FREERTOS_PLUS_FAT_TYPE_FAT32 | FAT32 disk. |
| RM_FREERTOS_PLUS_FAT_TYPE_FAT16 | FAT16 disk. |
| RM_FREERTOS_PLUS_FAT_TYPE_FAT12 | FAT12 disk. |

## 4.3.47 LittleFS Interface

Interfaces

### Detailed Description

Interface for LittleFS access.

# Summary

The LittleFS Port configures a fail-safe filesystem designed for microcontrollers on top of a lower level storage device.

Implemented by: LittleFS Flash Port (rm_littlefs_flash)

### Data Structures

| | |
|---|---|
| struct | rm_littlefs_cfg_t |
| struct | rm_littlefs_api_t |
| struct | rm_littlefs_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | rm_littlefs_ctrl_t |

### Data Structure Documentation

### ◆ rm_littlefs_cfg_t

| struct rm_littlefs_cfg_t |
|---|
| User configuration structure, used in open function |
| Data Fields |

| | | |
|---|---|---|
| struct lfs_config const * | p_lfs_cfg | Pointer LittleFS configuration structure. |
| void const * | p_extend | Pointer to hardware dependent configuration. |

### ◆ rm_littlefs_api_t

struct rm_littlefs_api_t

LittleFS Port interface API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(rm_littlefs_ctrl_t *const p_ctrl, rm_littlefs_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rm_littlefs_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | versionGet )(fsp_version_t *const p_version) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_littlefs_api_t::open) (rm_littlefs_ctrl_t *const p_ctrl, rm_littlefs_cfg_t const *const p_cfg)

Initialize The lower level storage device.

**Implemented as**

- RM_LITTLEFS_FLASH_Open

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ close

| fsp_err_t(* rm_littlefs_api_t::close) (rm_littlefs_ctrl_t *const p_ctrl) |
|---|

Closes the module and lower level storage device.

**Implemented as**

- RM_LITTLEFS_FLASH_Close

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_littlefs_api_t::open call. |

### ◆ versionGet

| fsp_err_t(* rm_littlefs_api_t::versionGet) (fsp_version_t *const p_version) |
|---|

Gets version and stores it in provided pointer p_version.

**Implemented as**

- RM_LITTLEFS_FLASH_VersionGet

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Code and API version used. |

### ◆ rm_littlefs_instance_t

| struct rm_littlefs_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| rm_littlefs_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_littlefs_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_littlefs_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ rm_littlefs_ctrl_t

| typedef void rm_littlefs_ctrl_t |
|---|
| LittleFS Port API control block. Allocate an instance specific control block to pass into the LittleFS Port API calls.<br><br>**Implemented as**<br><br>◦ rm_littlefs_flash_instance_ctrl_t |

## 4.3.48 Touch Middleware Interface

Interfaces

### Detailed Description

Interface for Touch Middleware functions.

# Summary

The TOUCH interface provides TOUCH functionality.

The TOUCH interface can be implemented by:

- Capacitive Touch Middleware (rm_touch)

### Data Structures

| | |
|---|---|
| struct | touch_button_cfg_t |
| struct | touch_slider_cfg_t |
| struct | touch_wheel_cfg_t |
| struct | touch_cfg_t |
| struct | touch_api_t |
| struct | touch_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | touch_ctrl_t |
| typedef struct st_ctsu_callback_args | touch_callback_args_t |

## Data Structure Documentation

### ◆ touch_button_cfg_t

| struct touch_button_cfg_t | | |
|---|---|---|
| Configuration of each button | | |
| Data Fields | | |
| uint8_t | elem_index | Element number used by this button. |
| uint16_t | threshold | Touch/non-touch judgment threshold. |
| uint16_t | hysteresis | Threshold hysteresis for chattering prevention. |

### ◆ touch_slider_cfg_t

| struct touch_slider_cfg_t | | |
|---|---|---|
| Configuration of each slider | | |
| Data Fields | | |
| uint8_t const * | p_elem_index | Element number array used by this slider. |
| uint8_t | num_elements | Number of elements used by this slider. |
| uint16_t | threshold | Position calculation start threshold value. |

### ◆ touch_wheel_cfg_t

| struct touch_wheel_cfg_t | | |
|---|---|---|
| Configuration of each wheel | | |
| Data Fields | | |
| uint8_t const * | p_elem_index | Element number array used by this wheel. |
| uint8_t | num_elements | Number of elements used by this wheel. |
| uint16_t | threshold | Position calculation start threshold value. |

### ◆ touch_cfg_t

| struct touch_cfg_t | | |
|---|---|---|
| User configuration structure, used in open function | | |
| Data Fields | | |
| touch_button_cfg_t const * | p_buttons | Pointer to array of button |

| | | |
|---|---|---|
| | | configuration. |
| touch_slider_cfg_t const * | p_sliders | Pointer to array of slider configuration. |
| touch_wheel_cfg_t const * | p_wheels | Pointer to array of wheel configuration. |
| uint8_t | num_buttons | Number of buttons. |
| uint8_t | num_sliders | Number of sliders. |
| uint8_t | num_wheels | Number of wheels. |
| uint8_t | on_freq | The cumulative number of determinations of ON. |
| uint8_t | off_freq | The cumulative number of determinations of OFF. |
| uint16_t | drift_freq | Base value drift frequency. [0 : no use]. |
| uint16_t | cancel_freq | Maximum continuous ON. [0 : no use]. |
| uint8_t | number | Configuration number for QE monitor. |
| ctsu_instance_t const * | p_ctsu_instance | Pointer to CTSU instance. |
| uart_instance_t const * | p_uart_instance | Pointer to UART instance. |
| void const * | p_context | User defined context passed into callback function. |
| void const * | p_extend | Pointer to extended configuration by instance of interface. |

### ◆ touch_api_t

| struct touch_api_t | |
|---|---|
| Functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(touch_ctrl_t *const p_ctrl, touch_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | scanStart )(touch_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | dataGet )(touch_ctrl_t *const p_ctrl, uint64_t *p_button_status, uint16_t *p_slider_position, uint16_t *p_wheel_position) |
| | |
| fsp_err_t(* | close )(touch_ctrl_t *const p_ctrl) |
| | |

| | |
|---|---|
| fsp_err_t(* | versionGet )(fsp_version_t *const p_data) |

## Field Documentation

### ◆ open

fsp_err_t(* touch_api_t::open) (touch_ctrl_t *const p_ctrl, touch_cfg_t const *const p_cfg)

Open driver.

**Implemented as**

- RM_TOUCH_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ scanStart

fsp_err_t(* touch_api_t::scanStart) (touch_ctrl_t *const p_ctrl)

Scan start.

**Implemented as**

- RM_TOUCH_ScanStart()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ dataGet

fsp_err_t(* touch_api_t::dataGet) (touch_ctrl_t *const p_ctrl, uint64_t *p_button_status, uint16_t *p_slider_position, uint16_t *p_wheel_position)

Data get.

**Implemented as**

- RM_TOUCH_DataGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_buton_status | Pointer to get data bitmap. |
| [out] | p_slider_position | Pointer to get data array. |
| [out] | p_wheel_position | Pointer to get data array. |

### ◆ close

| fsp_err_t(* touch_api_t::close) (touch_ctrl_t *const p_ctrl) |
|---|

Close driver.

**Implemented as**

- RM_TOUCH_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ versionGet

| fsp_err_t(* touch_api_t::versionGet) (fsp_version_t *const p_data) |
|---|

Return the version of the driver.

**Implemented as**

- RM_TOUCH_VersionGet()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_data | Memory address to return version information to. |

### ◆ touch_instance_t

| struct touch_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| touch_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| touch_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| touch_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ touch_ctrl_t

| typedef void touch_ctrl_t |
| --- |
| Control block. Allocate an instance specific control block to pass into the API calls.<br><br>**Implemented as**<br><br>◦ touch_instance_ctrl_t |

### ◆ touch_callback_args_t

| typedef struct st_ctsu_callback_args touch_callback_args_t |
| --- |
| Callback function parameter data |

# Chapter 5 Copyright

Copyright [2020] Renesas Electronics Corporation and/or its affiliates. All Rights Reserved.

This software and documentation are supplied by Renesas Electronics America Inc. and may only be used with products of Renesas Electronics Corp. and its affiliates ("Renesas"). No other uses are authorized. Renesas products are sold pursuant to Renesas terms and conditions of sale. Purchasers are solely responsible for the selection and use of Renesas products and Renesas assumes no liability. No license, express or implied, to any intellectual property right is granted by Renesas. This software is protected under all applicable laws, including copyright laws. Renesas reserves the right to change or discontinue this software and/or this documentation. THE SOFTWARE AND DOCUMENTATION IS DELIVERED TO YOU "AS IS," AND RENESAS MAKES NO REPRESENTATIONS OR WARRANTIES, AND TO THE FULLEST EXTENT PERMISSIBLE UNDER APPLICABLE LAW, DISCLAIMS ALL WARRANTIES, WHETHER EXPLICITLY OR IMPLICITLY, INCLUDING WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT, WITH RESPECT TO THE SOFTWARE OR DOCUMENTATION. RENESAS SHALL HAVE NO LIABILITY ARISING OUT OF ANY SECURITY VULNERABILITY OR BREACH. TO THE MAXIMUM EXTENT PERMITTED BY LAW, IN NO EVENT WILL RENESAS BE LIABLE TO YOU IN CONNECTION WITH THE SOFTWARE OR DOCUMENTATION (OR ANY PERSON OR ENTITY CLAIMING RIGHTS DERIVED FROM YOU) FOR ANY LOSS, DAMAGES, OR CLAIMS WHATSOEVER, INCLUDING, WITHOUT LIMITATION, ANY DIRECT, CONSEQUENTIAL, SPECIAL, INDIRECT, PUNITIVE, OR INCIDENTAL DAMAGES; ANY LOST PROFITS, OTHER ECONOMIC DAMAGE, PROPERTY DAMAGE, OR PERSONAL INJURY; AND EVEN IF RENESAS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSS, DAMAGES, CLAIMS OR COSTS.

Renesas Flexible Software Package (FSP)
Copyright © (2020) Renesas Electronics Corporation. All Rights Reserved.

User's Manual

Publication Date: Revision 1.00 Apr.5.20

# Renesas FSP v1.1.0

# User's Manual

Renesas Electronics Corporation