Renesas Synergy™ Platform

SPI Framework Module Guide

Introduction

This module guide will enable you to effectively use a module in your own design. Upon completion of this guide, you will be able to add this module to your own design, configure it correctly for the target application and write code, using the included application project code as a reference and efficient starting point. References to more detailed API descriptions and suggestions of other application projects that illustrate more advanced uses of the module are available on the Renesas Synergy Knowledge Base (as described in the References section at the end of this document), and should be valuable resources for creating more complex designs.

The SPI Framework module provides a ThreadX-aware framework API and handles the integration and synchronization of multiple SPI peripherals on an SPI bus (including chip-select handling and its level activation). With the SPI Framework, one or more SPI buses can be created and multiple SPI peripherals can be connected to the SPI bus. The SPI Framework module uses a single interface to access both SCI SPI and RSPI drivers. The SPI Framework module uses the SCI and RSPI peripherals on the Synergy MCU.

Contents

1. SPI Framework Module Features

The SPI Framework module uses either the SCI in SPI mode (together with the SCI common lower-level modules) or the RSPI lower-level driver module to communicate with the SPI peripherals on the Synergy microcontroller.

- Supports multiple devices on a bus
- Provides high-level APIs for initialization, transfers, and closing the module
- Supports synchronized transfers
- Supports chip-select operations
- Supports bus-locking

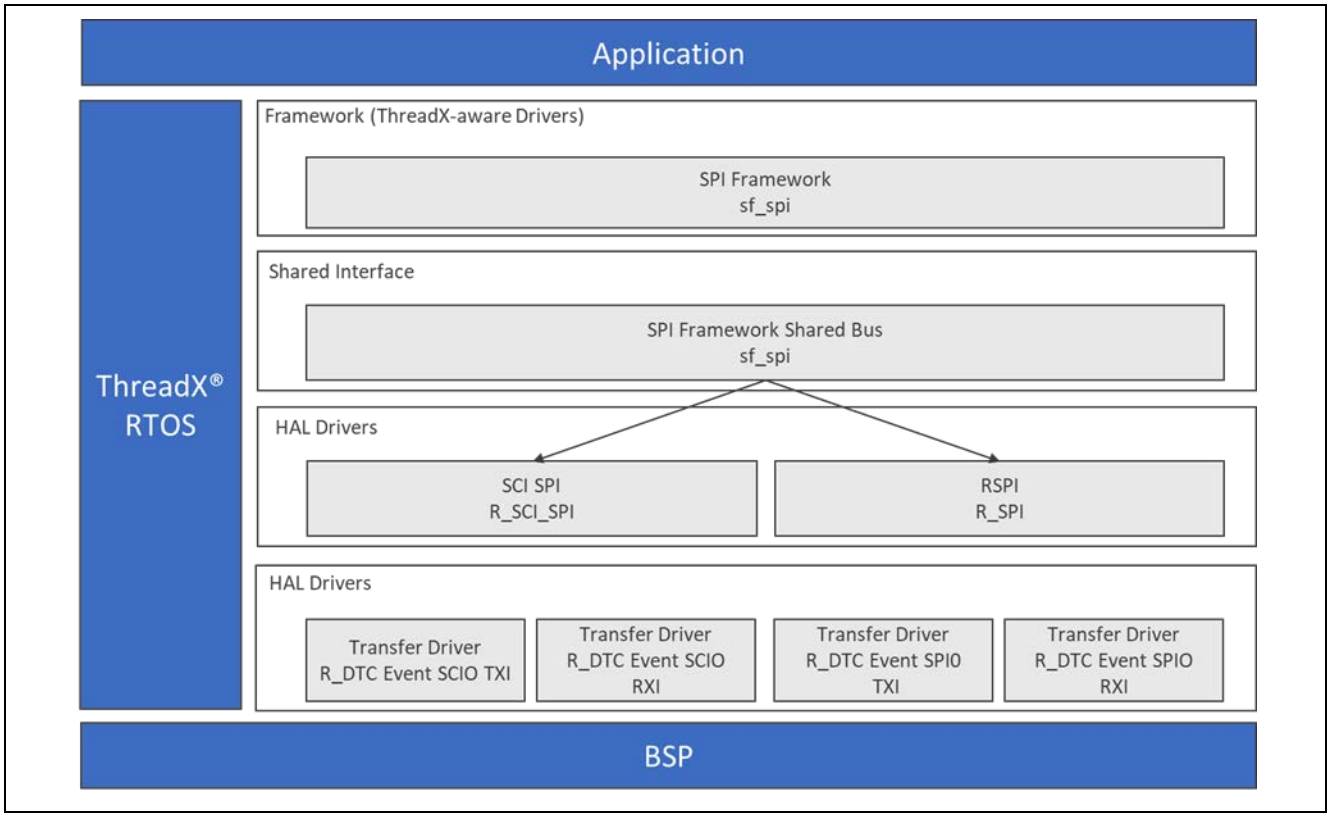

Figure 1. SPI Framework Module Block Diagram

2. SPI Framework Module APIs Overview

The SPI Framework module defines APIs for opening, closing, reading, writing and other useful functions. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Table 1. SPI Framework Module API Summary

Function Name	Example API Call and Description
.open	`g_sf_spi_device0.p_api->open(g_sf_spi_device0.p_cntl, g_sf_spi_device0.p_cfg);` Open a designated SPI device on a bus.
.read	`g_sf_spi_device0.p_api->read(g_sf_spi_device0.p_cntl, dst8, length, SPI_BIT_WIDTH_8_BITS, TX_WAIT_FOREVER);` Receive data from SPI device.
.write	`g_sf_spi_device0.p_api->write(g_sf_spi_device0.p_cntl, src8, length, SPI_BIT_WIDTH_8_BITS, TX_WAIT_FOREVER);` Transmit data to SPI device.

Function Name	Example API Call and Description
.writeRead	`g_sf_spi_device0.p_api->writeRead (g_sf_spi_device0.p_cntl, &source, &destination, length, SPI_BIT_WIDTH_8_BITS, TX_WAIT_FOREVER);` Simultaneously transmits data to an SPI device while receiving data from an SPI device (full duplex). The writeread API gets a mutex object, handles the SPI data transmission at SPI HAL layer, and receives data from the SPI HAL layer. The API uses the event flag wait to synchronize to completion of data transfer.
.close	`g_sf_spi_device0.p_api->close(g_sf_spi_device0.p_cntl);` Disable the SPI device designated by the control handle and close the RTOS services used by the bus, if no devices are connected to the bus. This function removes power to the SPI channel designated by the handle and disables the associated interrupts.
.lock	`g_sf_spi_device0.p_api->lock(g_sf_spi_device0.p_cntl);` Lock the bus for a device. The locking allows devices to reserve a bus to themselves for a given period of time (such as between lock and unlock). This allows devices to complete several reads and writes on the bus without an interrupt.
.unlock	`g_sf_spi_device0.p_api->unlock(g_sf_spi_device0.p_cntl);` Unlock the bus for a particular device and make the bus usable for other devices.
.versionGet	`g_sf_spi_device0.p_api->versionGet(&version);` Retrieve the API version with the version pointer.

Note: Details on operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the *SSP User's Manual* API References for the associated module.

Table 2. Status Return Values

Name	Description
SSP_SUCCESS	Function completed successfully
SSP_ERR_INVALID_MODE	Invalid mode
SSP_ERR_INVALID_CHANNEL	Invalid channel
SSP_ERR_IN_USE	In-use error
SSP_ERR_INVALID_ARGUMENT	Invalid argument
SSP_ERR_QUEUE_UNAVAILABLE	Queue unavailable
SSP_ERR_INVALID_POINTER	Invalid pointer
SSP_ERR_INTERNAL	Internal error
SSP_ERR_TRANSFER_ABORTED	Transfer aborted
SSP_ERR_MODE_FAULT	Mode fault
SSP_ERR_READ_OVF	Read overflow
SSP_ERR_PARITY	Parity error
SSP_ERR_OVERRUN	Overrun error
SSP_ERR_UNDEF	Unknown error
SSP_ERR_TIMEOUT	Timeout error
SSP_ERR_NOT_OPEN	Device not opened

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

3. SPI Framework Module Operational Overview

The SPI Framework module complies with the layered-driver architecture of the SSP. It uses either the SCI on SPI module or the RSPI module to communicate with the SPI peripherals on the Synergy microcontroller.

3.1 Multiple Slave Devices on the Same Bus

The SPI framework module uses a “bus” and “device on bus” architecture. Only one device is configured to the lower level driver at a time, and the other devices are reconfigured upon a read or write operation as required. The lower level driver can only be reconfigured when the bus is not locked. Every slave device is linked to the bus to which it will be connected and shares the bus with all other slave devices.

The user must configure the SPI framework shared-bus and the lower-level SPI HAL layer for each SPI framework module connecting to the bus. The user can add the existing framework shared-bus module when configuring multiple devices on the same bus. Each SPI framework module must be configured with a unique name in the ISDE configurator.

A common start and stop procedure is used for all SPI data-transfer operations (spi_api_t::read, spi_api_t::write, and spi_api_t::writeRead). During the start process, the SPI framework module checks whether reconfiguration is required. Chip select is asserted during the transfer-start process and de-asserted during the transfer-end process if the bus is not locked. The user must configure the chip-select IO pin and the chip-select active level.

3.2 Bus Locking

The SPI Framework module supports bus-locking functionality, meaning that the bus can be locked for a given slave peripheral. The locking allows slave devices to reserve a bus to themselves for the period between the lock and unlock commands. This allows devices to complete several reads and writes on the bus without interruption (which can be required in some situations). The chip select becomes active during lock and becomes inactive when unlocked. Writes and reads in between the lock and unlock do not alter the chip-select line.

3.3 SPI Framework Module Important Operational Notes and Limitations

3.3.1 SPI Framework Module Operational Notes

- Multiple SPI devices can be configured to share a common bus. Once the SPI Framework bus module is configured, different SPI peripherals (devices) can be connected to that bus.
- For each SPI device connected to the bus, one SPI HAL module and one SPI Framework device module must be added.
- User-defined callback is not required as the framework takes care of internally.
- Setting the interrupts to different priority levels could result in improper operation.

3.3.2 SPI Framework Module Limitations

- Refer to the MCU specification manual for identifying SPI bus compatibility. Device compatibility with the SPI bus is not checked in the framework hence incompatible SPI device may result in improper operation.
- Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4. Including the SPI Framework Module in an Application

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

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

To add the SPI Framework module to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the SPI Framework is g_sf_spi_device0. This name can be changed in the associated **Properties** window.)

Table 3. SPI Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_spi_device0 on sf_spi	Threads	New Stack> Framework> Connectivity> SPI Framework Device on sf_spi

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

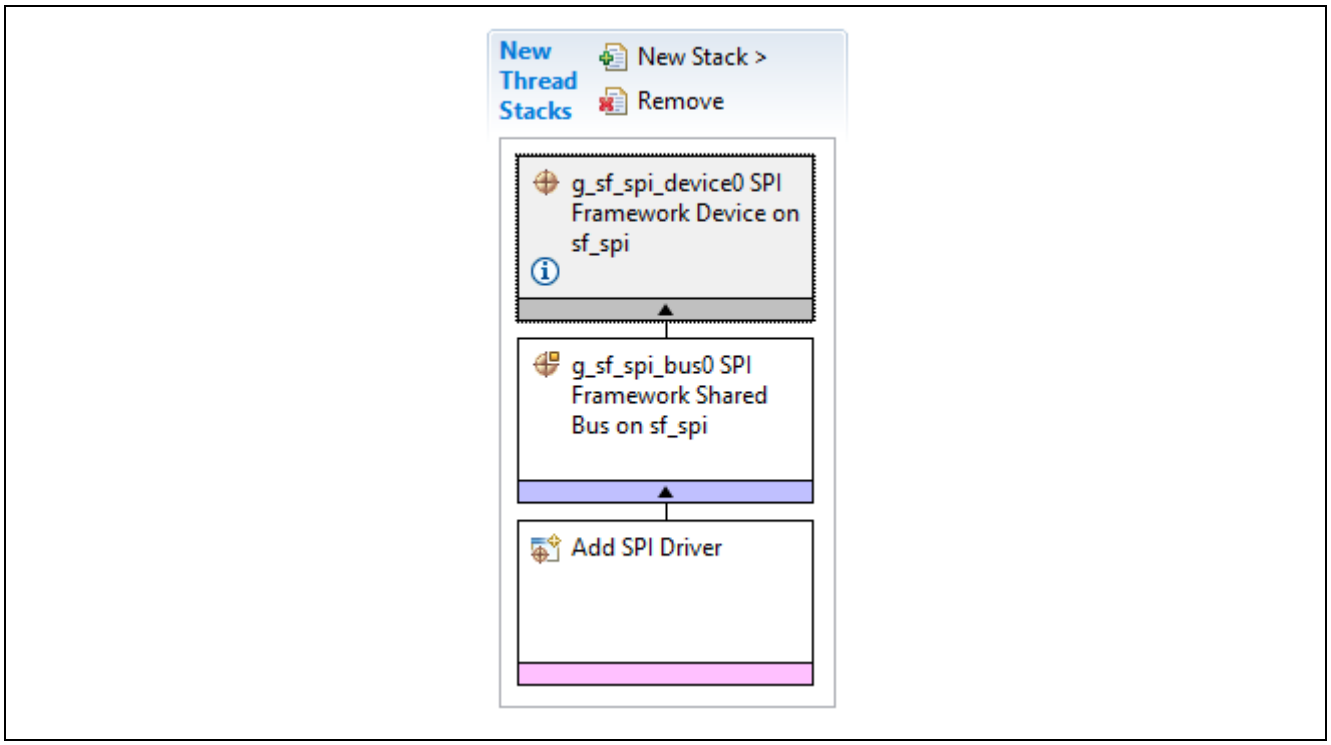

Figure 2. SPI Framework Module Stack

5. Configuring the SPI Framework Module

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

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

Table 4. Configuration Settings for the SPI Framework Module on sf_spi

Parameter	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build.
Name	g_sf_spi_device0	Module name.
Clock Phase	Data sampling on odd edge, data variation on even edge/Data sampling on even edge, data variation on odd edge Default: Data sampling on odd edge, data variation on even edge	Select the clock phase.
Clock Polarity	Low when idle, High when idle Default: Low when idle	Select the clock polarity.
Chip Select Pin	00 thru 15 Default: 00	Select GPIO pin used for the chip select.
Chip Select Pin	00 thru 15 (Default: 00)	Select GPIO pin used for the chip select.
Chip Select Active Level	Low, High Default: Low	Polarity of the Chip Select signal, active High or Low

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

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

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

5.1 Configuration the SPI Framework Lower-Level Modules

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

Table 5. Configuration Settings for the SPI Framework Shard Bus on sf_spi

ISDE Property	Value	Description
Name	g_sf_spi_bus0	Module name

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

Table 6. Configuration Settings for the RSPI HAL Driver on r_rspi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	If selected code for parameter checking is included in the build.
Name	g_spi0	Module name.
Channel	0	SCI or SPI Channel number to which the device has been connected.
Operating Mode	Master, Slave Default: Master	Configure as a Master or Slave device. Note: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on odd edge, data variation on even edge	Data sampling on odd or even clock edge.

ISDE Property	Value	Description
Clock Polarity	Low when idle	Clock level when idle.
Mode Fault Error	Enable, Disable Default: Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First, LSB First (Default: MSB First)	Select transmit order MSB/LSB first.
Bitrate	500000	Transmission or reception rate. Bits per second.
Callback	NULL	Optional Callback function pointer.
SPI Mode	SPI Operation, Clock synchronous operation Default: SPI Operation	Select spi or clock syn mode operation.
Slave Select Polarity(SSL1)	Active Low, Active High Default: Active Low	Select SSL1 signal polarity.
Slave Select Polarity(SSL2)	Active Low, Active High Default: Active Low	Select SSL2 signal polarity.
Slave Select Polarity(SSL3)	Active Low, Active High Default: Active Low	Select SSL3 signal polarity.
Select Loopback1	Normal, Inverted Default: Normal	Select the data mode for loopback 1.
Select Loopback2	Normal, Inverted Default: Normal	Select the data mode for loopback 2.
Enable MOSI Idle	Enable, Disable Default: Disable	Select MOSI idle fixed value and selection
MOSI Idle State	MOSI Low, MOSI High Default: MOSI Low	Select MOSI idle fixed value and selection
Enable Parity	Enable, Disable Default: Disable	Enable/disable parity
Parity Mode	Parity Odd, Parity Even Default: Parity Odd	Select parity
Select SSL(Slave Select)	SSL0, SSL1, SSL2, SSL3 Default: SSL0	Select which slave to use; 0-SSL0; 1-SSL1; 2-SSL2; 3-SSL3
Select SSL Level After Transfer	SSL Level Keep, SSL Level Do Not Keep Default: SSL Level Do Not Keep	Select SSL level after transfer completion; 0-negate; 1-keep
Clock Delay Enable	Clock Delay Enable, Clock Delay Disable Default: Clock Delay Disable	Clock delay enable selection
Clock Delay Count	Clock Delay 1 thru 8 RSPCK Default: Clock Delay 1 RSPCK	Clock delay count selection
SSL Negation Delay Enable	Negation Delay Enable, Negation Delay Disable Default: Negation Delay Disable	SSL negation delay enable selection
Negation Delay Count	Negation Delay 1 thru 8 RSPCK Default: Negation Delay 1 RSPCK	Negation delay count selection
Next Access Delay Enable	Next Access Delay Enable, Next Access Delay Disable Default: Next Access Delay Disable	Next access delay enable selection
Next Access Delay Count	Next Access Delay 1 thru 8 RSPCK Default: Next Access Delay 1 RSPCK	Next access delay count selection

ISDE Property	Value	Description
Receive Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Receive interrupt priority selection
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit interrupt priority selection
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit interrupt priority selection
Error Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Error interrupt priority selection

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

Table 7. Configuration Settings for the Transfer Driver on r_dtc Event SPI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	2 Bytes	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SPI0 TXI	Activation source selection
Auto Enable	False	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	ELC Software Event interrupt priority selection.

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

Table 8. Configuration Settings for the Transfer Driver on r_dtc Event SPI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection
Link section to keep DTC vector table	.ssp_dtc_vector_table	Link section to keep DTC vector table selection
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	2 Bytes	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SPI0 RXI	Activation source selection
Auto Enable	False	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	ELC Software Event interrupt priority selection.

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

Table 9. Configuration Settings for the SPI Driver on r_sci_spi

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_spi0	Module name
Channel	0	SCI or SPI Channel number to which the device has been connected.
Operating Mode	Master, Slave Default: Master	Configure as a Master or Slave device. Note: Current version of SSP supports only SPI Master mode.
Clock Phase	Data sampling on odd edge, data variation on even edge	Data sampling on odd or even clock edge.
Clock Polarity	Low when idle	Clock level when idle.
Mode Fault Error	Enable, Disable Default: Disable	Indicates Mode fault error (master/slave conflict) flag.
Bit Order	MSB First, LSB First Default: MSB First	Select transmit order MSB/LSB first
Bitrate	100000	Transmission or reception rate. Bits per second.

ISDE Property	Value	Description
Bit Rate Modulation Enable	Enable, Disable Default: Enable	Bitrate Modulation Function enable or disable
Callback	NULL	Optional Call back function pointer.
Receive Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Bitrate Modulation Function enable or disable. Note: This is applicable only for SCI SPI.
Transmit Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Transmit end interrupt priority selection
Error Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX) Default: Priority 12	Error interrupt priority selection

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

Table 10. Configuration Settings for the Transfer Driver on r_dtc Event SCI0 TXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection
Name	g_transfer0	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Fixed	Destination address mode selection
Source Address Mode	Incremented	Source address mode selection
Repeat Area (Unused in Normal Mode	Source	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 TXI	Activation source selection
Auto Enable	False	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	ELC Software Event interrupt priority selection.

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

Table 11. Configuration Settings for the Transfer Driver on r_dtc Event SCI0 RXI

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Selects if code for parameter checking is to be included in the build
Software Start	Enabled, Disabled Default: Disabled	Software start selection
Linker section to keep DTC vector table	.ssp_dtc_vector_table	Linker section to keep DTC vector table selection
Name	g_transfer1	Module name
Mode	Normal	Mode selection
Transfer Size	1 Byte	Transfer size selection
Destination Address Mode	Incremented	Destination address mode selection
Source Address Mode	Fixed	Source address mode selection
Repeat Area (Unused in Normal Mode	Destination	Repeat area selection
Interrupt Frequency	After all transfers have completed	Interrupt frequency selection
Destination Pointer	NULL	Destination pointer selection
Source Pointer	NULL	Source pointer selection
Number of Transfers	0	Number of transfers selection
Number of Blocks (Valid only in Block Mode)	0	Number of blocks selection
Activation Source (Must enable IRQ)	Event SCI0 RXI	Activation source selection
Auto Enable	False	Auto enable selection
Callback (Only valid with Software start)	NULL	Callback selection
ELC Software Event Interrupt Priority	Priority 0 (highest), Priority 1:14 Priority 15 (lowest - not valid if using ThreadX), Disabled Default: Disabled	ELC Software Event interrupt priority selection.

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

5.2 SPI Framework Module Clock Configuration

The SPI peripheral module uses PCLKB as its clock source. The PCLKB frequency is set by using the SSP configurator clock tab prior to a build, or by using the CGC Interface at run-time.

5.3 SPI Framework Module Pin Configuration

The SPI peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the SPI pins.

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

Table 12. Pin Selection for the SPI Framework Module

Resource	ISDE Tab	Pin selection Sequence
SCI	Pins	Select Peripherals > Connectivity: SCI> SCI1
RSPI	Pins	Select Peripherals > Connectivity: SPI > SPI0

Note: The top selection sequence assumes SCI1 and SPI0 are the desired hardware targets for the driver and the bottom selection sequence assumes SPI0 is the desired target.

Table 13. Pin Configuration Settings for the SPI Framework Module

Property	Settings	Description
Operation Mode	Disabled, Asynchronous UART, Synchronous UART, Simple I2C, Simple SPI, SmartCard Default: Disabled	Select Simple SPI as the Operation Mode for SPI on SCI
CTS0_RTS0_SS0	None, P103, P413 Default: None	SS0 Pin selection
RXD0_SCL0_MISO0	None, P100, P410 Default: None	MISO0 Pin selection
SCK0	None, P102, P412 Default: None	SCK0 Pin selection
TXD1_SDA1_MOSI0	None, P100, P410 Default: None	MOSI0 Pin selection

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

5.4 SPI Framework Module Additional Settings

If external chip selects are being used, configure the chip select pins as GPIO outputs.

6. Using the SPI Framework Module in an Application

A common application for the SPI framework module requires multiple slave devices on a single bus. The implementation for this common application is described below. A second implementation shows two busses each with two slave devices attached.

Implementation Steps for Two Slave Devices on a Single Shared Bus

When using the SPI framework module to create a single bus with multiple slave devices, create two thread stacks each with an I2C framework instance. These instances will use the same shared bus instance. Follow the steps below to see how this is done within the SSP Configurator.

Note: The following steps assume some familiarity with the use of the SSP development environment. If any of the following steps are confusing, read over the first few chapters of the SSP User's Manual to become familiar with the SSP development environment.

Step 1: Add the first SPI framework device module to a new or existing thread. This creates the SPI master stack. A shared bus on sf_spi is added along with the I2C driver. The SPI driver can be selected for implementation on r_rspi or r_sci_spi. The DTC transfer driver is also added by default. This can be removed if the CPU transfer mode is needed instead.

The resulting module stack is shown in the following figure. Example configuration settings are given in the tables that follow the figure.

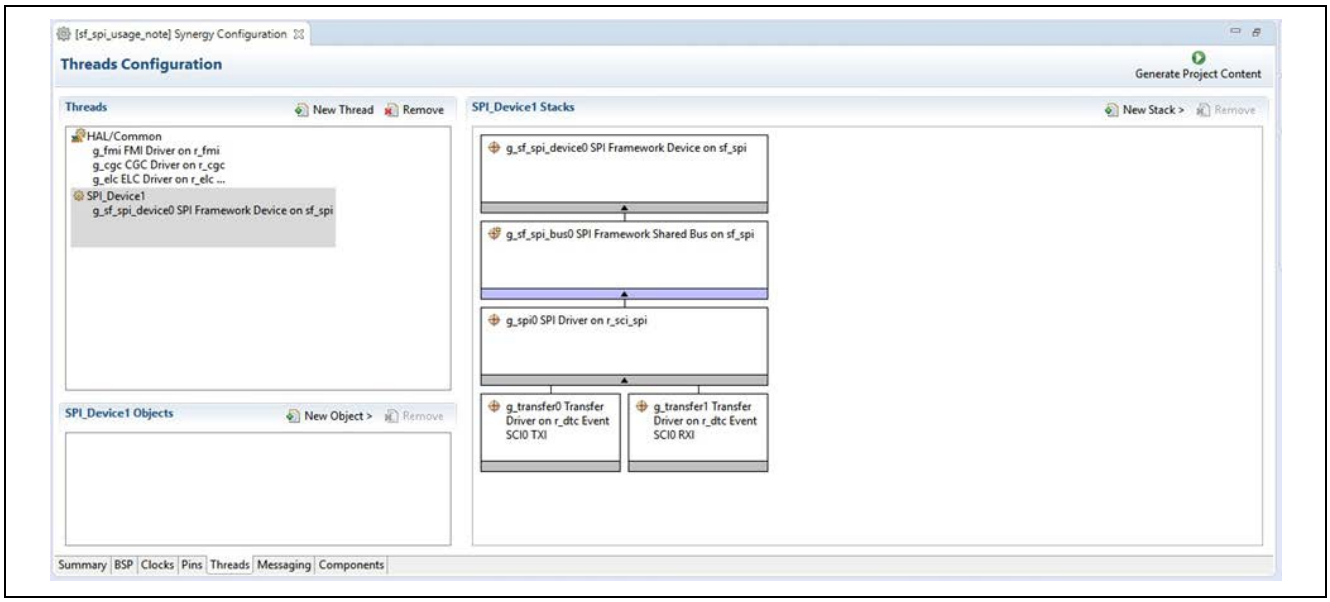

Figure 3. Resulting Module Stack

Example configuration settings for the key first thread stack modules for Slave Device #1 are as follows:

Table 14. Configuration Settings for the SPI Framework Module on sf_spi

Property	Settings	Description
Parameter Checking	Disabled	Enable or Disable Parameter Checking.
Name	g_sf_spi_device1	Give a name to identify the SPI Framework device. API, Config and Control instances will be created based on this name.
Clock Phase	Data sampling on odd edge	Specify the clock phase for data variation and data sampling
Clock Polarity	Low when idle	Select the clock polarity when clock is idle.
Clock Select Port	01	Select GPIO port used for the chip select.
Chip Select Pin	04	Select GPIO pin used for the chip select.
Chip Select Active Level	Low	Select Polarity of the chip select signal.

Table 15. Configuration Settings for the SPI Framework Shared Bus on sf_spi

Property	Settings	Description
Name	g_sf_spi_bus0	Give a name to identify the SPI Framework shared bus. This shared bus will be shared by multiple SPI Framework Devices

Table 16. Configuration Settings for the SPI Driver on r_rspi

Property	Settings	Description
Parameter Checking	BSP	Enable or Disable Parameter Checking.
Name	g_spi0	Give a name to identify the SPI Driver device. This will be used by Framework internally.
Channel	0	Channel number
Operating Mode	Master	Operating mode selection
Clock Phase	Data sampling on odd edge/ data variation on edge	Clock phase selection. This field will be locked as these is already set in the SPI Framework Device on sf_spi module.
Clock Polarity	Low when idle	Clock polarity selection. This field will be locked as these is already set in the SPI Framework Device on sf_spi module.
Mode Fault Error	Enable	Mode fault error selection
Bit Order	MSB First	Bit order selection
Bitrate	500000	Bit rate selection
Callback	NULL	Callback function name
SPI Mode	SPI Operation	SPI mode selection
Slave Select Polarity (SSL0)	Active Low	Slave select polarity selection 0
Slave Select Polarity (SSL1)	Active Low	Slave select polarity selection 1
Slave Select Polarity (SSL2)	Active Low	Slave select polarity selection 2
Slave Select Polarity (SSL3)	Active Low	Slave select polarity selection 3
Select Loopback 1	Normal	Loopback 1 selection
Select Loopback 2	Normal	Loopback 2 selection
Enable MOSI Idle	Disable	Enable MOSI idle selection.
MOSI Idle State	MOSI Low	Enable MOSI idle state selection.
Enable Parity	Disable	Enable parity selection
Parity Mode	Parity Odd	Enable parity mode selection
Select SSL (Slave Select)	SSL0	Select SSL selection
Select SSL Level After Transfer	SSL Level Keep	Select SSL level after transfer selection
Clock Delay Enable	Disable	Clock delay enable selection
Clock Delay Count	Clock Delay 1 RSPCK	Clock delay count selection
SSL Negation Delay Enable	Disable	SSL Negation Delay Enable selection.
Negation Delay Count	Clock Delay 1 RSPCK	Negation Delay Count selection
Next Access Delay Enable	Disable	Next Access Delay Enable selection
Next Access Delay Count	Clock Delay 1 RSPCK	Next Access Delay Count selection
Receive Interrupt Priority	Priority 2	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 2	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 2	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 2	Error interrupt priority selection.

Table 17. Configuration Settings for the SPI Driver on r_sci_spi

Property	Settings	Description
Parameter Checking	BSP	Enable or Disable Parameter Checking.
Name	g_spi0	Give a name to identify the SPI Driver device. This will be used by Framework internally.
Channel	0	Channel number
Operating Mode	Master	Operating mode selection.
Clock Phase	Data sampling on odd edge/ data variation on even edge	Clock phase selection. This field will be locked as these is already set in the SPI Framework Device on sf_spi module.
Clock Polarity	Standard	Clock polarity selection. This field will be locked as these is already set in the SPI Framework Device on sf_spi module.
Mode Fault Error	Disable	Mode fault error selection.
Bit Order	MSB First	Bit order selection
Bitrate	500000	Bit rate selection
Bit Rate Modulation Enable	Enable	Enables/Disable the bit rate modulation.
Callback	NULL	Callback function name. This field will be locked as callback is handled internally in the framework.
Receive Interrupt Priority	Priority 2	Receive interrupt priority selection.
Transmit Interrupt Priority	Priority 2	Transmit interrupt priority selection.
Transmit End Interrupt Priority	Priority 2	Transmit end interrupt priority selection.
Error Interrupt Priority	Priority 2	Error interrupt priority selection.

Note: DTC configuration settings are not shown as a simplification.

Step 2: Add the second SPI Framework Device to a different thread. The SPI Framework Shared Bus on sf_spi is not added automatically. To add it, select the option to use the existing shared bus. The configurator will then automatically add the SPI Framework Shared Bus on sf_spi and the remaining modules. The lower level modules will automatically be configured to be consistent with the previously defined settings from the first SPI framework instance. This ensures that the SPI driver configurations are the same for both devices except for the Clock Phase, Clock Polarity, Chip Select Pin and Port, and Chip Select Active Level properties, as these are defined under the SPI Framework Device module and can be different for each slave device.

The resulting module stack is shown in the following figure:

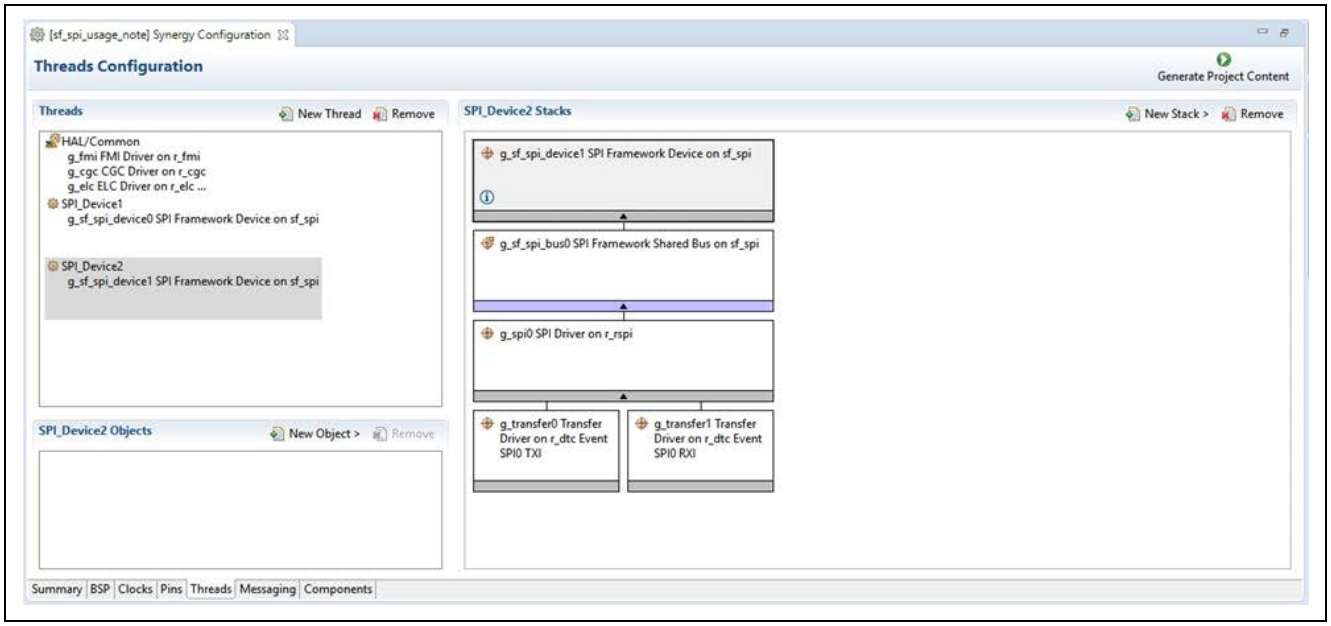

Figure 4. Resulting Module Stack

The only differences in the configuration parameters for the second stack are the name for the second SPI framework device module, and any differences in the non-shared slave settings (Clock Phase, Clock Polarity, Clock Select Port, Chip Select Pin and Chip Select Active Level). Example settings are shown in the following table:

Table 18. Configuration Settings for the SPI Framework Device on sf_spi (Slave #2)

Property	Settings	Description
Parameter Checking	BSP	Enable or Disable Parameter Checking.
Name	g_sf_spi_device2	Give a name to identify the SPI Framework device. API, Config and Control instances will be created based on this name.
Clock Phase	Data sampling on odd edge/ data variation on even edge	Specify the clock phase for data variation and data sampling
Clock Polarity	High when idle	Select the clock polarity when clock is idle.
Clock Select Port	05	Select GPIO port used for the chip select.
Chip Select Pin	01	Select GPIO pin used for the chip select.
Chip Select Active Level	Low	Select Polarity of the chip select signal.

6.1 Implementation Steps for Two Slave Devices on Two Shared Busses

When using the SPI framework module to create a single bus with multiple slave devices create two thread stacks each with an I2C framework instance. These instances will use the same shared bus instance. Follow the steps below to see how this is done within the SSP Configurator.

Note: The following steps assume some familiarity with the use of the SSP development environment. If any of the following steps are confusing, read over the first few chapters of the SSP User's Manual to become familiar with the SSP development environment.

6.2 Adding Another Shared Bus

To add another shared bus just follow the below steps. The previous example is used as the starting point.

1. The SPI framework module which will use a second shared bus can be added to any thread. Starting with the previous example, if it is added to the SPI_Device1 thread, then the module stack would appear as shown below. Available options for the shared bus are **New** or **Use**.

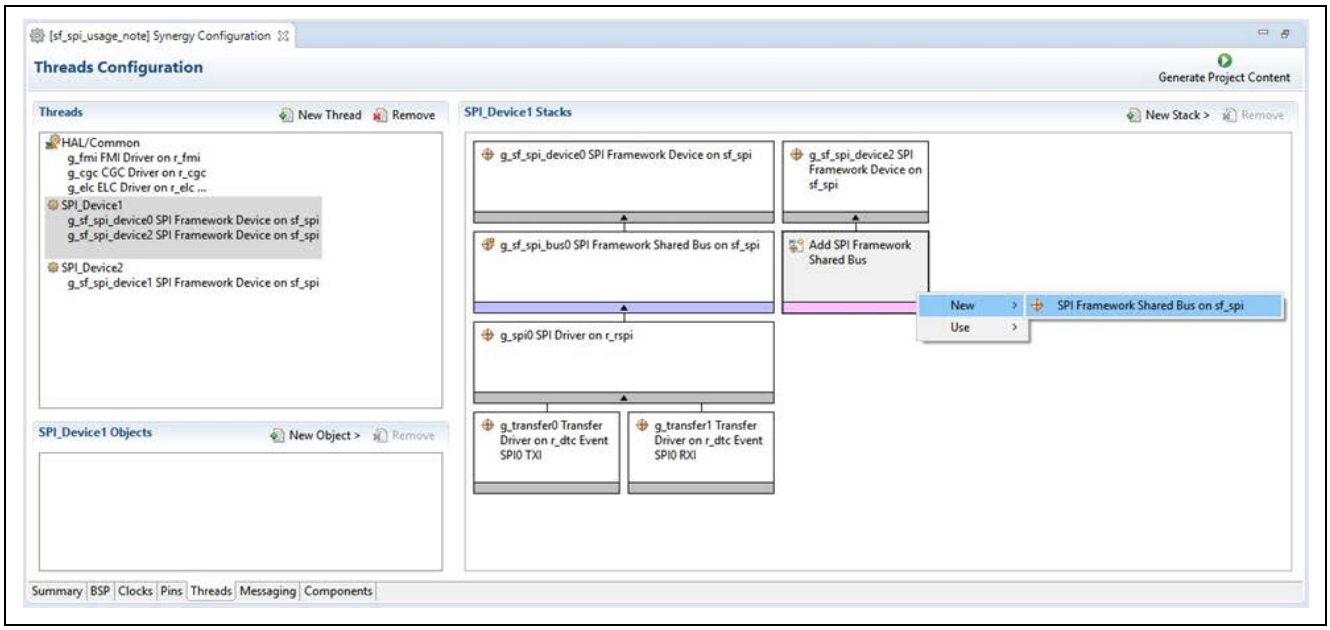

Figure 5. Resulting Module Stack

2. Select **New** to add another SPI Framework Shared Bus on sf_spi module. Configure the shared bus properties as needed for the application. Select the desired low-level SPI driver. The channel number for the g_spi1 SPI driver module, must be different from the channel number for the g_spi0 SPI driver module. The resulting thread stack is shown below:

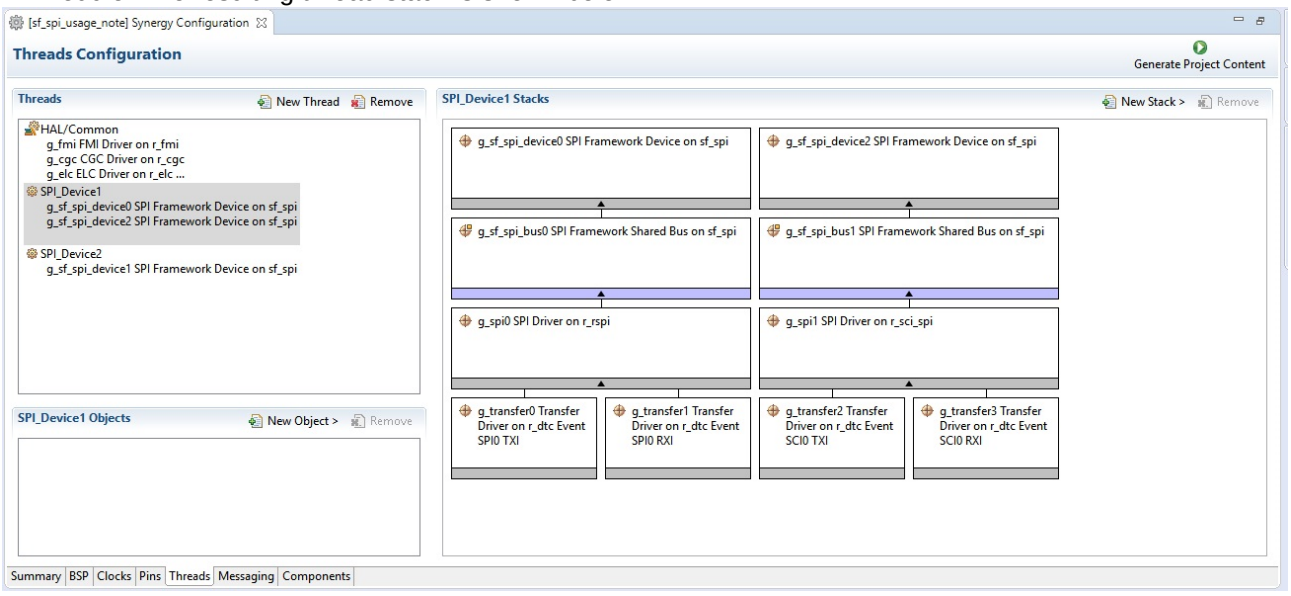

Figure 6. Resulting Module Stack

3. A second device can be added in the SPI_Device2 thread using the same steps described above. The resulting thread stack is shown below:

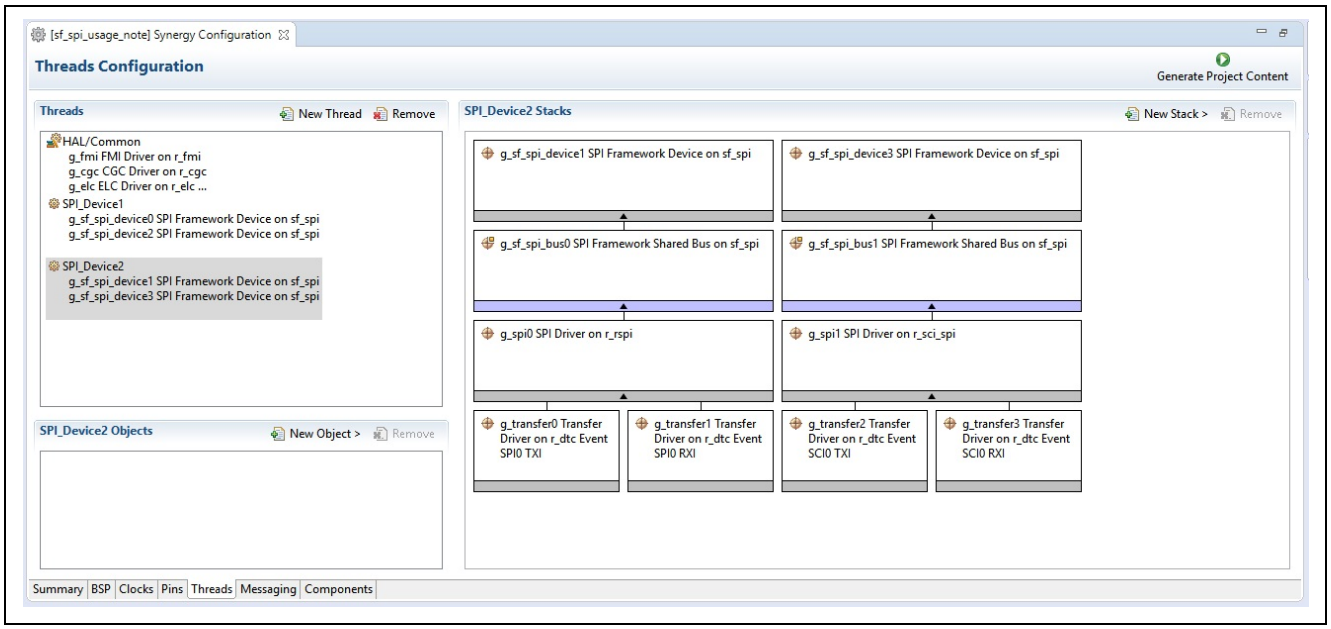

Figure 7. Resulting Module Stack

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

1. Initialize the SPI Framework device module using the `sf_spi_api_t::open` API function. Each SPI framework device module needs to call the `spi_api_t::open` API function at least once before performing any operations on the bus.
2. Lock the bus for continuous transfer using the `sf_spi_api_t::lock` API function for a particular SPI Framework device module. Once Bus is locked by a particular SPI Framework device module it cannot be used by any other SPI Framework device module on the bus. This ensures that ownership of the bus remains with the locked module until it explicitly unlocks it. Any kind of operation from other SPI Framework device modules on the bus will return a fail status during this period. It is not mandatory to lock the bus before any read/write operations on the bus. It is optional.
3. Read data using the `sf_spi_api_t::read` API function. The read operation will not be successful if the bus is already locked by any other SPI Framework device module.
4. Write data using the `sf_spi_api_t::write` API function. The write operation will not be successful if the bus is already locked by any other SPI Framework device module.
5. Write and read data simultaneously using the `sf_spi_api_t::writeRead` API function. The simultaneous read and write operation will not be successful if the bus is already locked by any other SPI Framework device module.
6. Unlock the bus from continuous transfer using the `sf_spi_api_t::unlock` API function if it is already locked by the same device. Once the bus is unlocked other SPI Framework device modules can use it. It is necessary to unlock the locked bus after the intended read/write operation is completed.
7. Close the SPI Framework device module using the `sf_spi_api_t::close` API function. Each SPI Framework device module can call the `sf_spi_api_t::close` API function after all read/write operations on the bus are over. These common steps are illustrated in a typical operational flow diagram in the following figure:

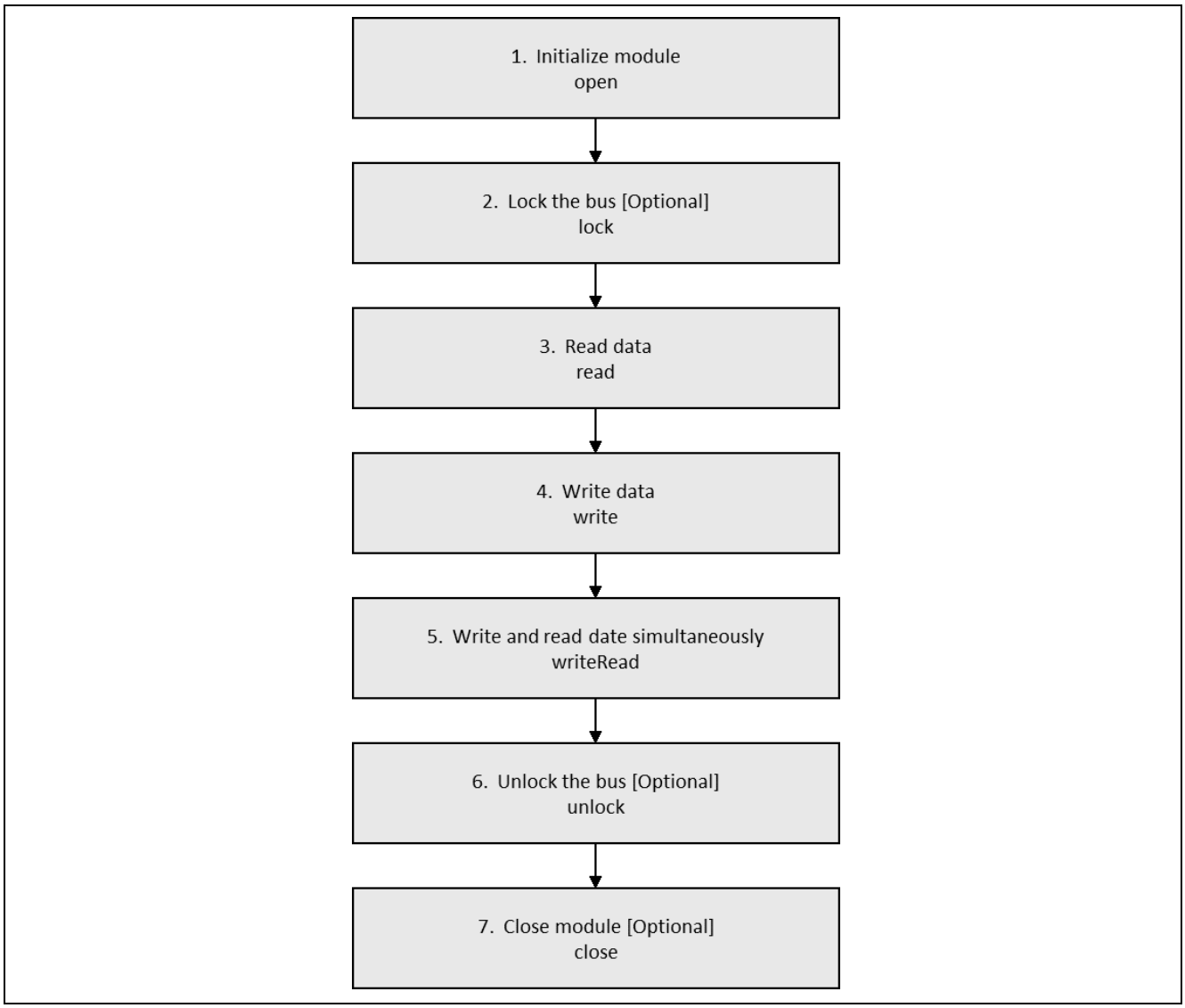

Figure 8. Flow Diagram of a Typical SPI Framework Application

7. SPI Framework Module Application Project

The application project associated with this module guide demonstrates the aforementioned steps in a full design; the project can be found using the link provided in the References section at the end of this document. You may want to import and open the application project within the ISDE and view the configuration settings for the SPI Framework module. You can also read over the code (in `temperature_thread_entry.c`) used to illustrate the SPI Framework module APIs in a complete design.

The application project demonstrates the typical use of the SPI Framework module APIs. Temperature sensor MAX31723 is plugged into PMODA. It works as an SPI slave device to read the current environment temperature. The LEDs are lit basing on temperature difference; the temperature is measured periodically in 1-second intervals.

Table 19. Software and Hardware Resources Used by the Application Project

Resource	Revision	Description
e² studio	v7.3.0 or later	Integrated Solution Development Environment
IAR EW for Synergy	v8.23.3 or later	IAR Embedded Workbench® for Renesas Synergy™
SSP	v1.6.0 or later	Synergy Software Platform
SSC	v7.3.0 or later	Synergy Standalone Configurator
SK-S7G2	v3.0 to v3.3	Starter Kit

The following figure illustrates a simple flow diagram of the application project.

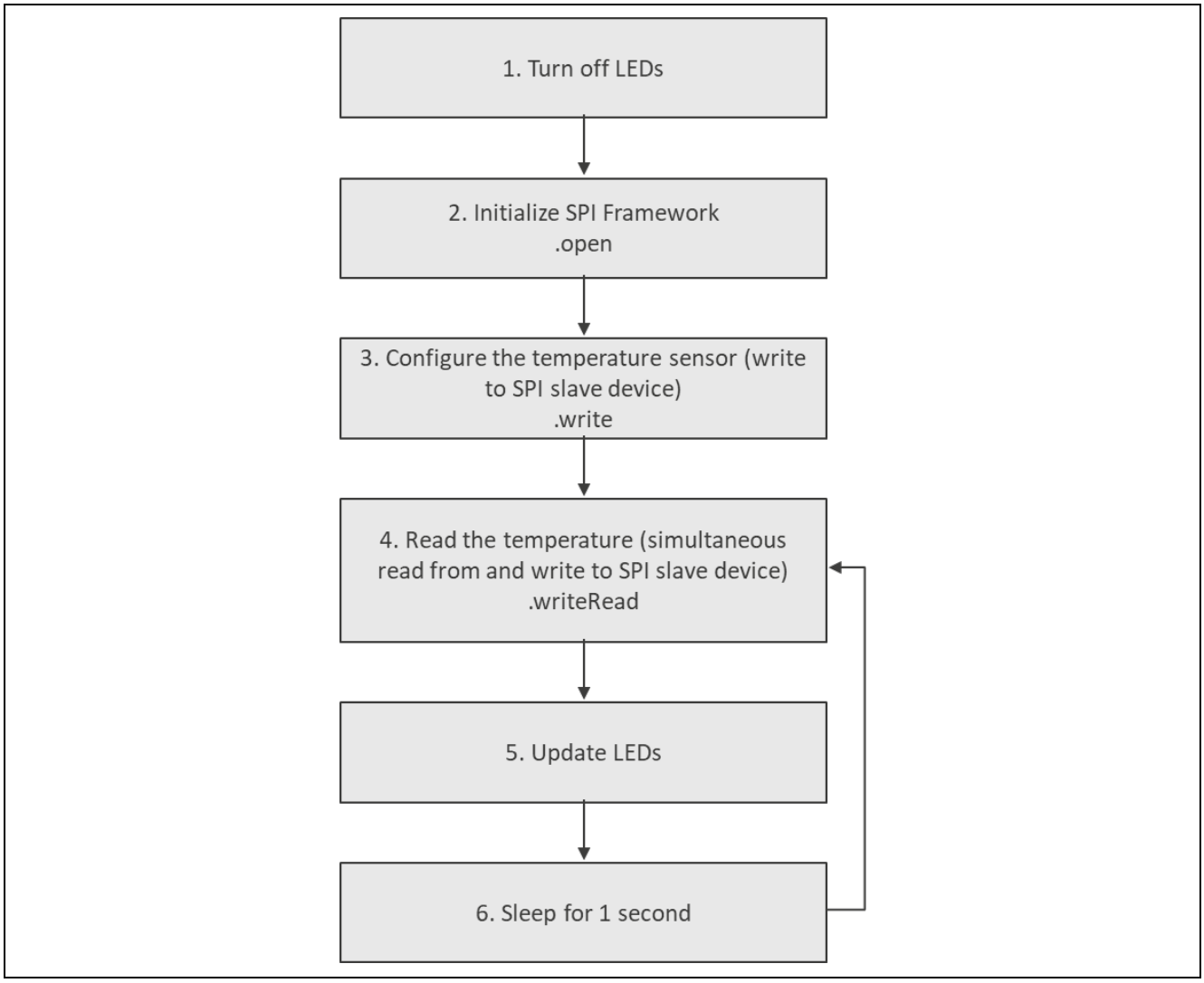

Figure 9. SPI Framework Module Application Project Flow Diagram

The `temperature_thread_entry.c` file is located in the project once it has been imported into the ISDE. You can open this file within the ISDE and follow along with the description provided to help identify key uses of APIs.

The first section of `temperature_thread_entry.c` has the header files which reference the SPI instance structure and the math functions that are used to perform floating-point calculation of the temperature. If enabled via a `#define`, a code section which allows semi-hosting to display results using `printf()` is included. This is followed by the global-variable definitions used within the application and the function prototypes.

The entry function for the thread is `temperature_thread_entry()`. Within the function, local variables for temperature calculation are defined, as well as data arrays containing configuration data for configuring the temperature sensor and a storage location for the received temperature data.

As the application project illuminates the LEDs according to the calculated temperature, the LEDs are set to their starting state of all-off.

The next stage is to open the SPI Framework. If this successfully opens, the next step is to configure the temperature sensor. The configuration data is written to the temperature sensor using the `write` API. The data configures the temperature sensor for 12-bit resolution. A successful completion of the write function returns SSP_SUCCESS.

The application now enters the thread `while(1)` loop. Here the temperature is read using the `writeRead` API. Again, successful completion of the `writeRead` function returns SSP_SUCCESS. The data that is

written is the address from where the temperature data can be read from. The temperature data is 12 bits in size, so a minimum of 2 bytes of data have to be read. The storage location for temperature data is 3 bytes in size, as the process of writing the address will receive dummy data.

The temperature is then calculated using the valid 2 bytes received.

The first temperature that is calculated by the application is stored as the reference temperature. All subsequent temperature calculations that occur every second are compared to this reference temperature. If the new temperature differs from the reference temperature, then the LEDs are illuminated accordingly. 2^{O}C delta – Green LED, 3^{O}C delta – Green & Orange LEDs, 4^{O}C delta – Green & Orange & Red LEDs.

If enabled, the measured temperatures will be shown on the Debug Console.

Note: The preceding description assumes you are familiar with using `printf()` with the Debug Console in the Synergy Software Package. If you are unfamiliar with this, see *How do I Use Printf() with the Debug Console in the Synergy Software Package*, given in the References section at the end of this document. Alternatively, the user can see results via the watch variables in the debug mode.

A few key properties are configured in this application project to support the required operations and the physical properties of the target board and MCU. The properties with the values set for this specific project are listed in the following tables; you can also open the application project and view these settings in the Properties window as a hands-on exercise.

Table 20. SPI Framework Module Configuration Settings for the Application Project

ISDE Property	Value Set
Name	g_sf_spi_device0
Clock Phase	Data sampling on even edge, data variation on odd edge
Clock Polarity	High when idle
Chip Select Port	01
Chip Select Pin	03
Chip Select Active Level	High

Note: This configuration assumes that pin 103 is configured as a GPIO pin, mode: Output mode (Initial Low) on the Pins tab.

Table 21. SPI Framework Module Shared Bus Settings for the Application Project

ISDE Property	Value Set
Name	g_sf_spi_bus0

Table 22. SPI HAL Module (r_rspi) Configuration Settings for the Application Project

ISDE Property	Value Set
Name	g_spi
Channel	0
Operating Mode	Master
Clock Phase	Data sampling on odd edge, data variation on even edge
Clock Polarity	Low when idle
Mode Fault Error	Disable
Bit Order	MSB First
Bitrate	500000
Callback	NULL
SPI Mode	SPI Operation
Slave Select Polarity (SSL0)	Active Low
Slave Select Polarity (SSL1)	Active Low
Slave Select Polarity (SSL2)	Active Low
Slave Select Polarity (SSL3)	Active Low
Select Loopback1	Normal
Select Loopback2	Normal
Enable MOSI Idle	Disable

ISDE Property	Value Set
MOSI Idle State	MOS Low
Enable Parity	Disable
Parity Mode	Parity Odd
Select SSL (Slave Select)	SSL0
Select SSL Level After Transfer	SSL Level Do Not Keep
Clock Delay Enable	Clock Delay Disable
Clock Delay Count	Clock Delay 1 RSPCK
SSL Negation Delay Enable	Negation Delay Disable
Negation Delay Count	Negation Delay 1 RSPCK
Next Access Delay Enable	Next Access Delay Disable
Next Access Delay Count	Next Access Delay 1 RSPCK
Receive Interrupt Priority	Priority 2
Transmit Interrupt Priority	Priority 2
Transmit End Interrupt Priority	Priority 2
Error Interrupt Priority	Priority 2

Note: The SPI HAL module on r_rspi has no DTC HAL modules configured (nor for transmission, nor for reception) due to SPI data-width (8 bits, not supported by DTC.)

8. Customizing the SPI Framework Module for a Target Application

The user can modify the channel used for SPI transmission, bitrate, and other parameters depending on the slave device. The most significant change can be performed by using alternative SPI implementation: Simple SPI on SCI. The SPI Framework device should be configured with an SPI HAL module on `r_sci_spi`. PMODB should be used for the temperature sensor. The following tables list all the details:

Table 23. SPI Framework Module Configuration Settings for SCI SPI option

ISDE Property	Value Set
Name	g_sf_spi_device0
Clock Phase	Data sampling on even edge, data variation on odd edge
Clock Polarity	High when idle
Chip Select Port	04
Chip Select Pin	13
Chip Select Active Level	High

Note: This configuration assumes that pin 413 is configured as a GPIO pin mode: Output mode (Initial Low) on the Pins tab.

Table 24. SPI Framework Module Shared Bus Settings for SCI SPI option

ISDE Property	Value Set
Name	g_sf_spi_bus0

Table 25. SPI HAL Module (r_sci_spi) Configuration Settings for SCI SPI option

ISDE Property	Value Set
Name	g_spi0
Channel	0
Operating Mode	Master
Clock Phase	Data sampling on odd edge, data variation on even edge
Clock Polarity	Low when idle
Mode Fault Error	Disable
Bit Order	MSB First
Bitrate	100000
Bit Rate Modulation Enable	Enable
Callback	NULL

ISDE Property	Value Set
Receive Interrupt Priority	Priority 8 (CM4: valid, CM0+: invalid)
Transmit Interrupt Priority	Priority 8 (CM4: valid, CM0+: invalid)
Transmit End Interrupt Priority	Priority 8 (CM4: valid, CM0+: invalid)
Error Interrupt Priority	Priority 8 (CM4: valid, CM0+: invalid)

The SPI HAL module on `r_sci_spi` should have no DTC drivers configured (not for transmission, and not for reception) due to SPI data-width (8 bits, not supported by DTC.)

Additionally, another SPI device could be included and the user could experiment with sharing an SPI bus.

9. Running the SPI Framework Module Application Project

To run the SPI Framework application project and to see it executed on a target kit, you can simply import it into your ISDE, compile, and run debug.

To implement the SPI Framework application in a new project, follow the steps for defining, configuring, auto-generating files, adding code, compiling, and debugging on the target kit. Following these steps is a hands-on approach that can help make the development process with SSP more practical, while just reading over this guide tends to be more theoretical.

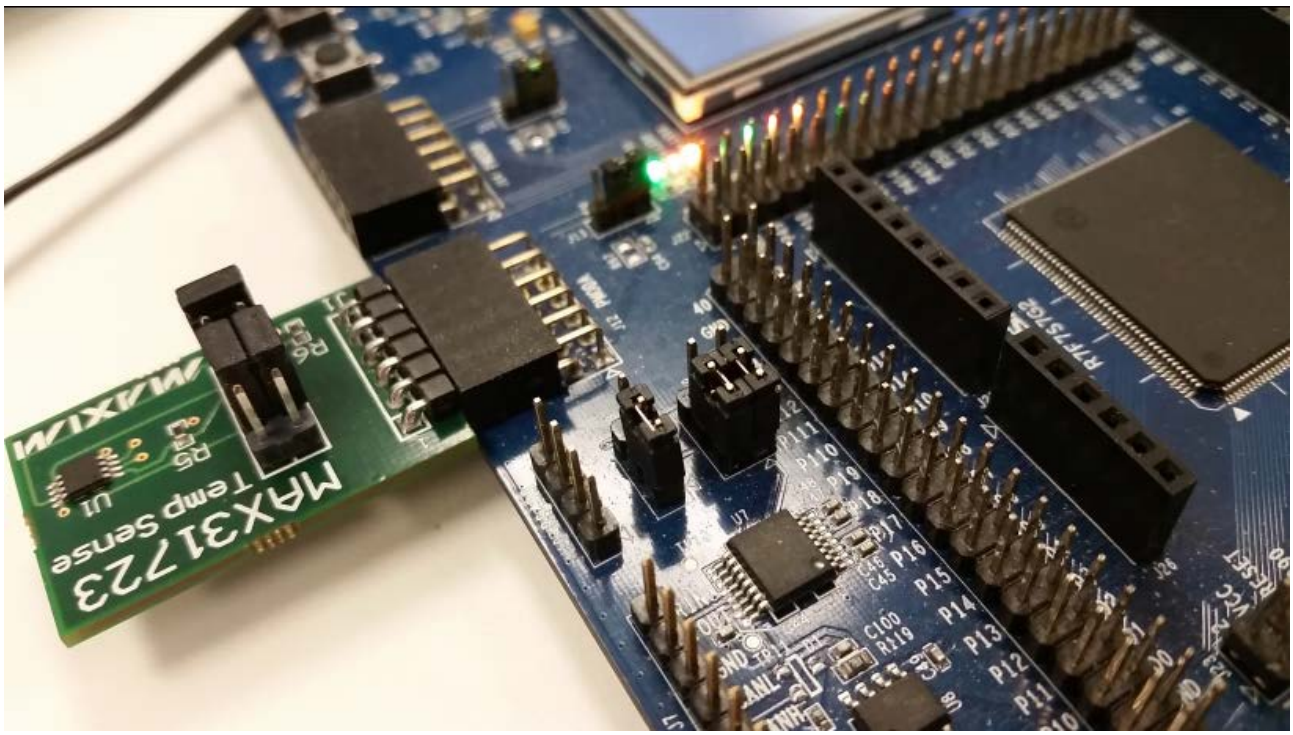

Figure 10. Temperature Sensor Connected to PMODA

Note: The following steps provide sufficient detail for someone experienced with the basic flow through the Synergy development process. If these steps are not familiar, refer to the first few sections in the *SSP User's Manual*, available as described in the References section at the end of this document.

To create and run the SPI Framework application project, simply follow these steps:

1. Connect the temperature sensor as shown in Figure 10.
2. Create a new Renesas Synergy project for the SK-S7G2 board called **SPI_Framework_MG_AP**.
3. Select the **Threads** tab.
4. Add a new thread called
   Symbol: **temperature_thread**
   Name: **Temperature Thread**
5. Add to **Temperature Thread the SPI Framework** and configure it.
6. Click on the **Generate Project Content** button.

7. Add the code from the supplied project file `temperature_thread_entry.c` or copy over the generated `temperature_thread_entry.c` file.
8. Connect to the host PC via a micro USB cable to J19 on SK-S7G2 Kit.
9. Start to debug the application.
10. Touch the temperature sensor (change the temperature) and watch the LEDs being lit.

Expected output:

```
Init temperature: 25.50
Current temperature: 25.06
Init temperature: 25.50
Current temperature: 25.00
Init temperature: 25.50
Current temperature: 25.00
Init temperature: 25.50
Current temperature: 25.00
```

If the temperature is over 100, the temperature sensor is either not connected or broken.

10. SPI Framework Module Conclusion

This module guide has provided all the background information needed to select, add, configure and use the module in an example project. Many of these steps were time consuming and error-prone activities in previous generations of embedded systems. The Renesas Synergy Platform makes these steps much less time consuming and removes the common errors, like conflicting configuration settings or the incorrect selection of lower-level drivers. The use of high-level APIs (as demonstrated in the application project) illustrates additional development time savings by allowing work to begin at a high level and avoiding the time required in older development environments to use or, in some cases, create, lower-level drivers.

11. SPI Framework Module Next Steps

After you have mastered a simple SPI Framework module project, you may want to review a more complex example, probably using other SPI Slave devices. Experiment with SCI SPI implementation. Compare the accompanying application project with SCI SPI and RSPI application projects to see the benefits of using SSP frameworks.

12. SPI Framework Module Reference Information

SSP User Manual: Available in html format in the SSP distribution package and as a pdf from the Synergy Gallery.

Links to all the most up-to-date sf_spi module reference materials and resources are available on the Synergy Knowledge Base: https://en-us.knowledgebase.renesas.com/English_Content/Renesas_Synergy%E2%84%A2_Platform/Renesas_Synergy_Knowledge_Base/SF_SPI_Module_Guide_References.

Website and Support

Visit the following vanity URLs to learn about key elements of the Synergy Platform, download components and related documentation, and get support.

Synergy Software	www.renesas.com/synergy/software
Synergy Software Package	www.renesas.com/synergy/ssp
Software add-ons	www.renesas.com/synergy/addons
Software glossary	www.renesas.com/synergy/softwareglossary
Development tools	www.renesas.com/synergy/tools
Synergy Hardware	www.renesas.com/synergy/hardware
Microcontrollers	www.renesas.com/synergy/mcus
MCU glossary	www.renesas.com/synergy/mcuglossary
Parametric search	www.renesas.com/synergy/parametric
Kits	www.renesas.com/synergy/kits
Synergy Solutions Gallery	www.renesas.com/synergy/solutionsgallery
Partner projects	www.renesas.com/synergy/partnerprojects
Application projects	www.renesas.com/synergy/applicationprojects
Self-service support resources:	
Documentation	www.renesas.com/synergy/docs
Knowledgebase	www.renesas.com/synergy/knowledgebase
Forums	www.renesas.com/synergy/forum
Training	www.renesas.com/synergy/training
Videos	www.renesas.com/synergy/videos
Chat and web ticket	www.renesas.com/synergy/resourcelibrary

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jun.14.17	-	Initial version
1.01	Aug.30.17	7	Update to Hardware and Software Resources Table
1.02	Feb.06.19	-	Updated for SSP v1.5.0
1.10	Apr.29.19	-	Updated for SSP v1.6.0

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.