

RX Family

SCI SPI FIT to FSP Migration Guide

Introduction

This application note provides a comprehensive guide on migrating an existing application that uses the Serial Communication Interface (SCI) Firmware Integration Technology (FIT) module (SPI mode) to the SCI SPI FSP module (Renesas Flexible Software Package). It includes detailed instructions, migration steps, and demonstrations to assist users throughout the migration process.

Target Modules

- FIT Module: r_sci_rx
- FSP Module: r_sci_b_spi

Contents

1. Overview	3
1.1 Stack Diagram	3
1.2 API Functions	4
2. Migrating Project FIT to FSP.....	5
2.1 Creating a new FSP Project	5
2.2 Migrating Configuration	6
2.3 Generate Code	11
2.4 Migrating Application Code	14
2.5 Build and Run the project	18
3. Reference Documents	19
4. Appendices.....	19
4.1 FIT & FSP Environment	19
4.2 Reference FSP Application Code.....	20
4.3 Comparison Reference	27
4.3.1 Compare Functions (Details).....	27
4.3.2 Compare Configuration (Details).....	31
Revision History	32

1. Overview

1.1 Stack Diagram

The figure below illustrates the stack diagram of the SCI FIT module and the SCI SPI FSP module.

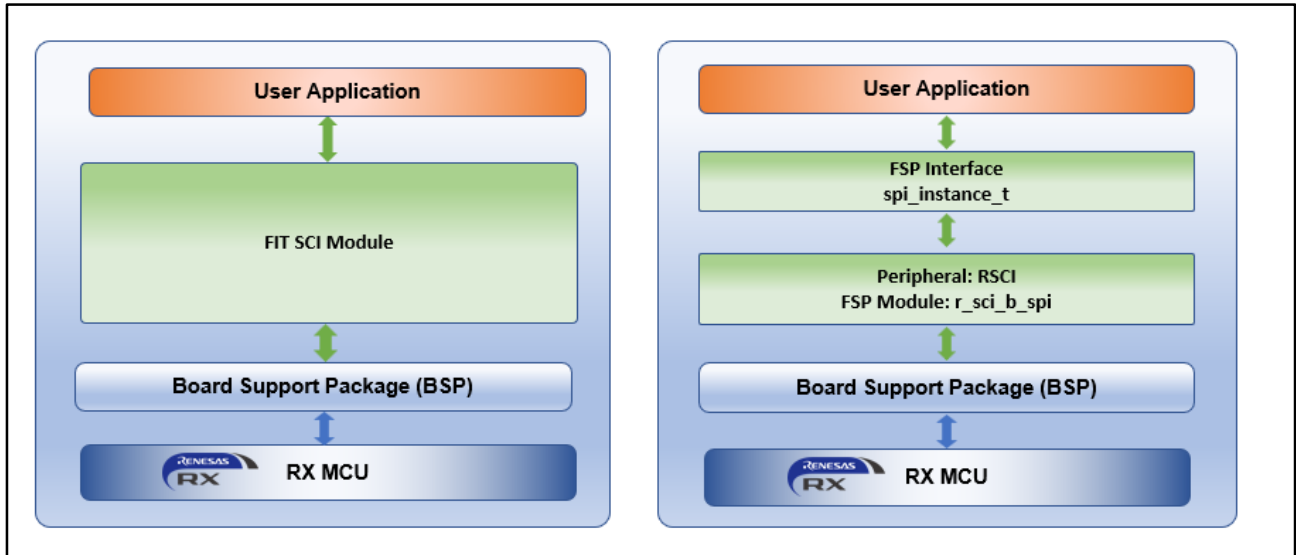


Figure 1-1 SCI FIT and SCI SPI FSP stack diagram

1.2 API Functions

This section summarizes the API Functions of the SCI FIT module (SPI mode) and the SPI FSP module.

FIT	FSP
R_SCI_Open	open
R_SCI_Close	close
R_SCI_Send	write
R_SCI_Receive	read
R_SCI_SendReceive	writeRead
R_SCI_GetVersion	Not supported in specific modules FSP has common function to get FSP pack version. R_FSP_VersionGet (in bsp_api.h)
R_SCI_Control Use Commands to configure and control.	Not supported Control functions. Settings in Properties of FSP Configuration
Callback function set in R_SCI_Open	callbackSet

For more details, please refer to the sections:

[Compare Functions \(Details\)](#) : comparison FIT and FSP.

[Migrate Application Code](#) : example of using FSP SPI API in an application.

2. Migrating Project FIT to FSP

2.1 Creating a new FSP Project

Refer to the reference document “RX FSP Migration Guide” for creating RX FSP project.

To create a new project in e² Studio, navigate to the top bar menu and select:

File > New > Renesas C/C++ Project > Renesas RX.

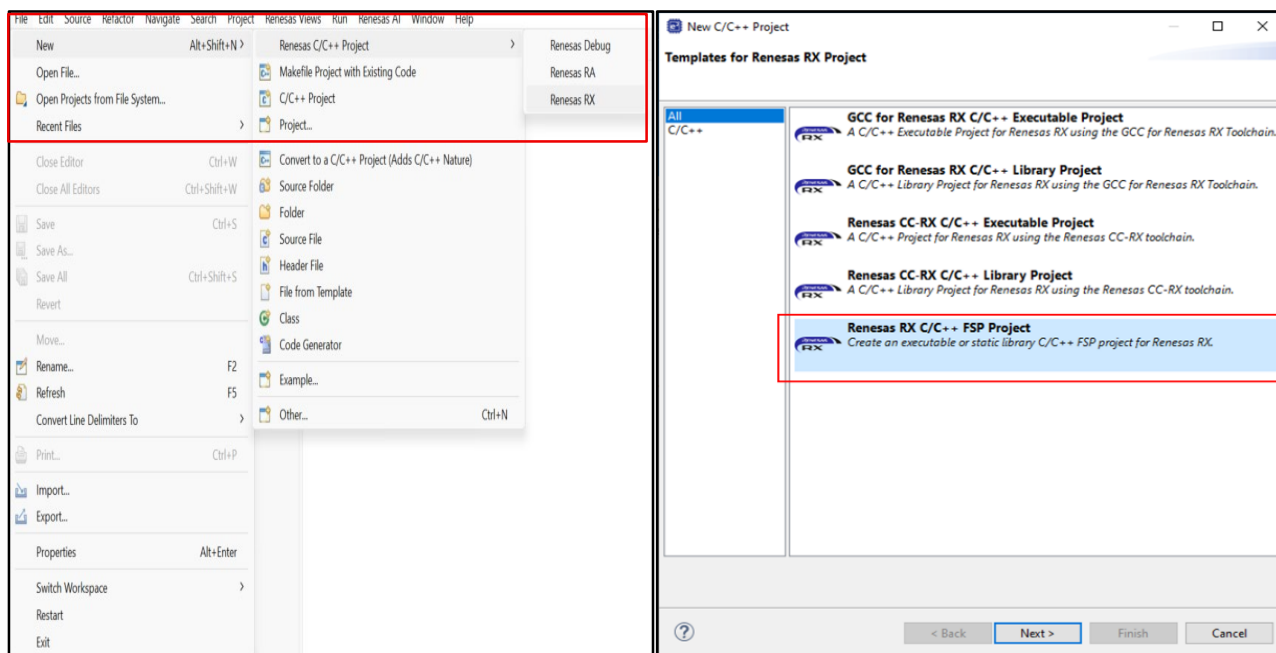


Figure 2-1 Create a new Renesas RX C/C++ FSP Project.

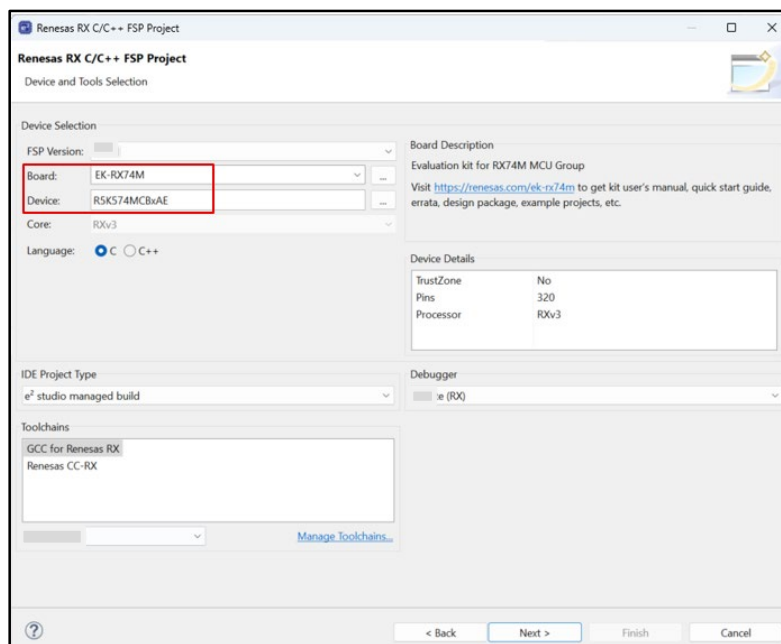


Figure 2-2 Select Board.

2.2 Migrating Configuration

This section describes migration configuration.

Configuration for SCI SPI in this application notes shown as table below:

Configuration	Value
Operating Mode	Master
Channel	5
Pins Setting - MOSI - MISO - SCK - SS	Pin Setting for SCI5 (SPI mode) - TXD5 - RXD5 - SCK5 - GPIO (P009)
Bit Rate	8000000
Clock Source	SCISPICK
Clock Phase	Data sampling on odd edge
Clock Polarity	Low when idle
Bit order	MSB first
Interrupts Priority	Priority 3
Callback	my_sci_callback (user's function name)

Note: Channel, Pins setting depends on User's board.

Steps of updating configuration on FSP Project.

On Configuration Stacks Tab

Click on New Stack -> Connectivity > SPI (r_sci_b_spi).

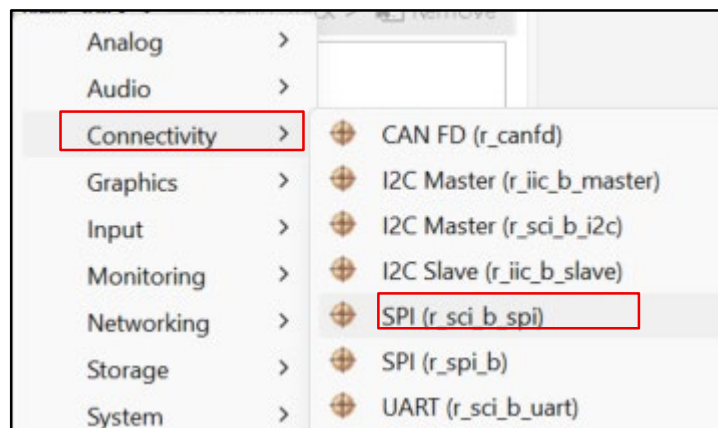


Figure 2-3 Add module SPI in Stack

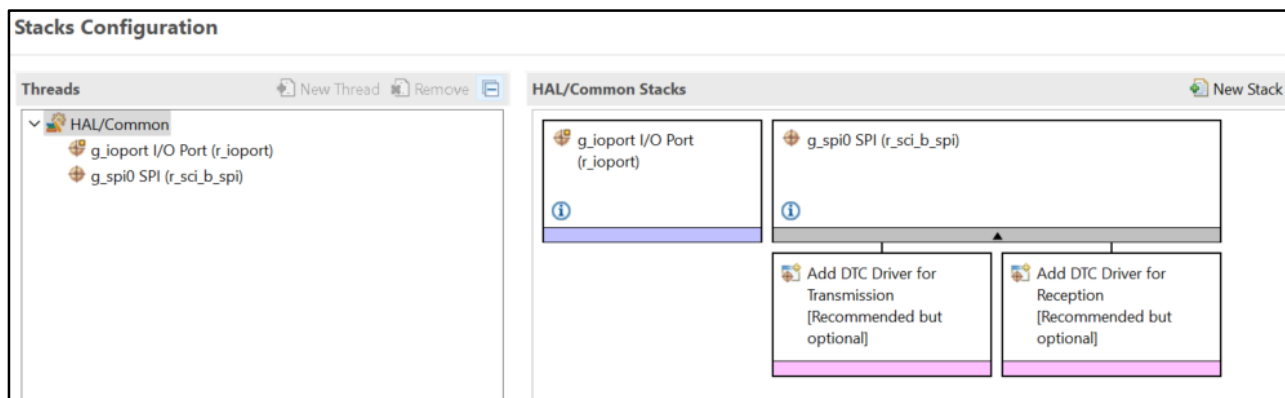


Figure 2-4 SCI SPI in Stack

Click on the "**g_spi0 SPI (r_sci_b_spi)**" module once it appears.

The "Properties" window will display its properties.

Configure SCI SPI FSP according to configuration above.

g_spi0 SPI (r_sci_b_spi)		
Settings	Property	Value
	▼ Common	
	Parameter Checking	Default (BSP)
	DTC Support	Disabled
	▼ Module g_spi0 SPI (r_sci_b_spi)	
	Name	g_spi0
	Channel	5
	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
	Clock Source	SCISPICK
	Callback	my_sci_callback
	Receive Interrupt Priority	Priority 3
	Transmit Interrupt Priority	Priority 3
	Transmit End Interrupt Priority	Priority 3
	Error Interrupt Priority	Priority 3
	Bitrate	8000000
	▼ Pins	
	CTS_RT55#	<unavailable>
	RXD5	P213
	SCK5	P212
	TXD5	P214

Figure 2-5 Configure SPI Properties

The following sections show the Pin settings configured using the GUI (FSP Configurator window).

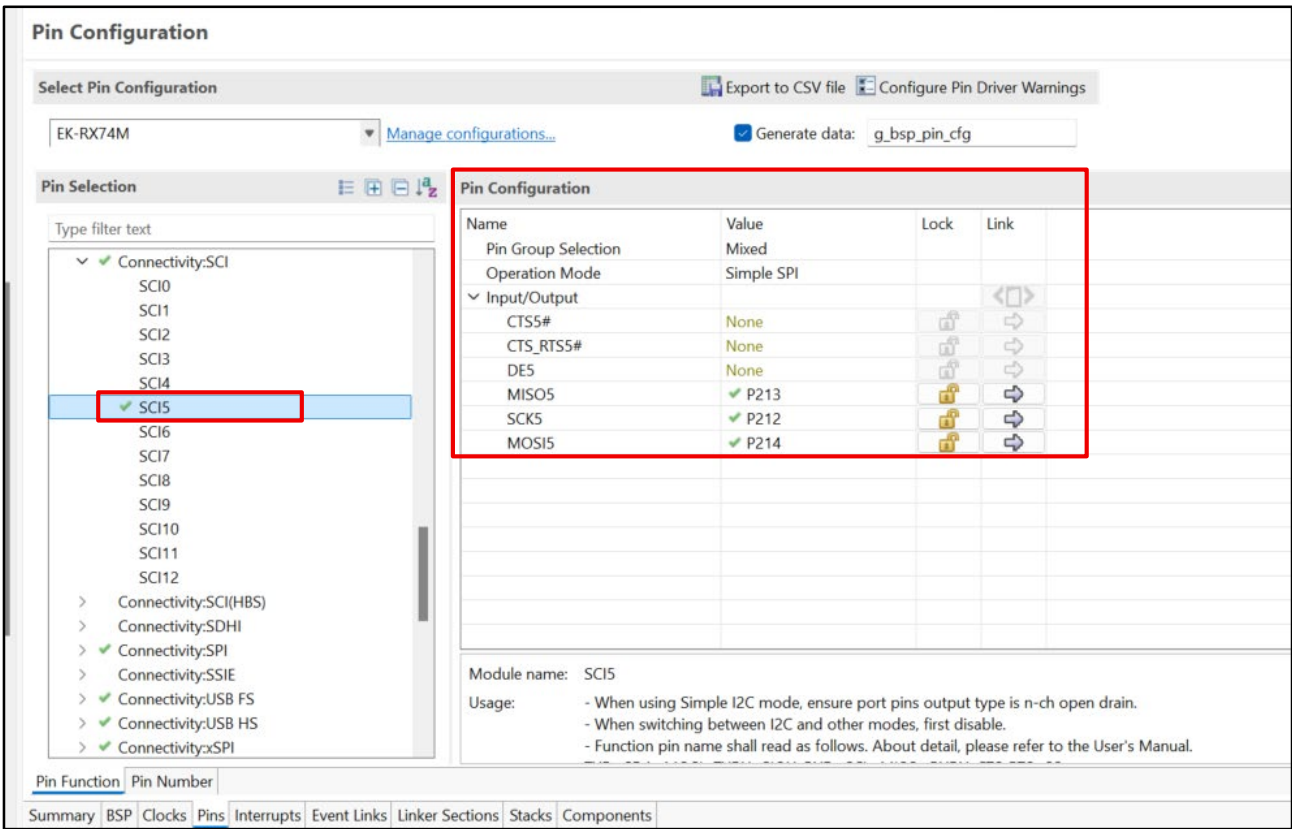


Figure 2-6 Pin Configuration.

The "Pin Configuration" tab shows the application's pin configuration.

Notes: If there is any conflict on pin setting, change the pin mode to Peripheral mode in Pin Configuration

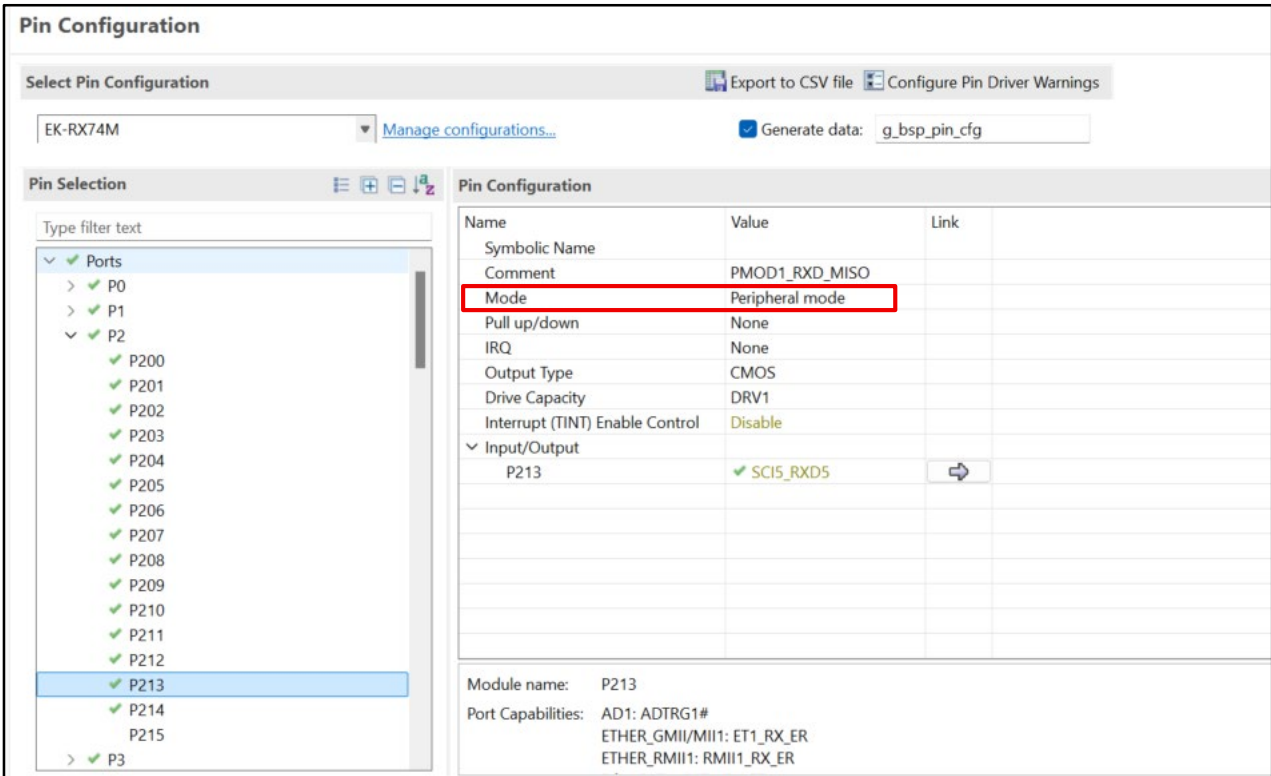


Figure 2-7 Pin Configuration

2.3 Generate Code

After completing stacks and HAL configurations and making user selections, click Generate Project Content. This generates the configuration code for selected modules. You can find generated code using the Project Explorer window in e² studio. The configuration.xml file, containing all settings, can be edited in the GUI-based configuration editor for any necessary reconfigurations.

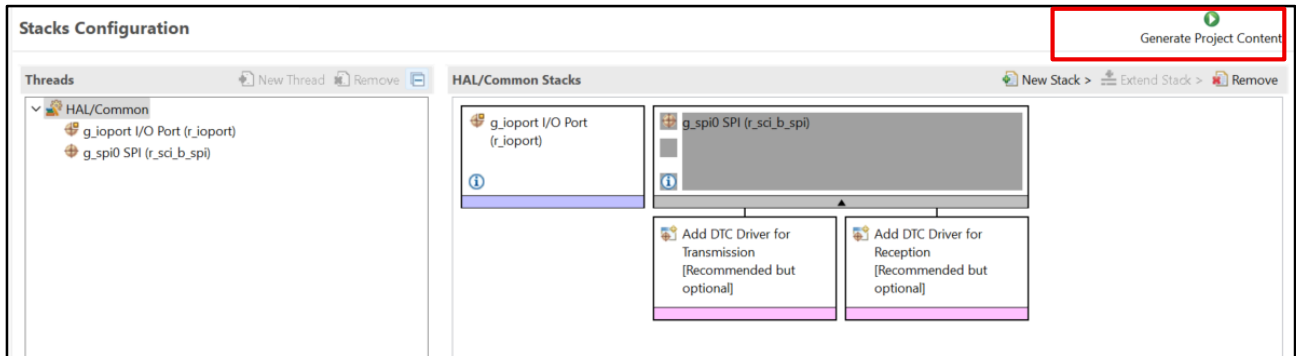


Figure 2-8 Generate Project Content

Once the module has been configured and the files generated, the SCI SPI FSP module is ready to be used in an application.

Generated code in `rx_gen/hal_data.c`.

The following figure shows the code generated for SCI SPI FSP according to the configuration in the Properties.

```

1  /* generated HAL source file - do not edit */
2  #include "hal_data.h"
3  sci_b_spi_instance_ctrl_t g_spi0_ctrl;
4
5  /** SPI extended configuration */
6  const sci_b_spi_extended_cfg_t g_spi0_cfg_extend =
7  {
8      .clk_div =
9      /* Actual calculated bitrate: 7500000. */.cks = 0,
10     .brr = 7, .bgdm = 1, },
11     .clock_source = (sci_b_spi_clock_source_t) 0, };
12
13     const spi_cfg_t g_spi0_cfg =
14     { .channel = 5, .operating_mode = SPI_MODE_MASTER, .clk_phase = SPI_CLK_PHASE_EDGE_ODD, .clk_polarity =
15       SPI_CLK_POLARITY_LOW,
16       .mode_fault = SPI_MODE_FAULT_ERROR_DISABLE, .bit_order = SPI_BIT_ORDER_MSB_FIRST,
17       #define FSP_NOT_DEFINED (1)
18       #if (FSP_NOT_DEFINED == FSP_NOT_DEFINED)
19         .p_transfer_tx = NULL,
20       #else
21       #if (FSP_NOT_DEFINED == FSP_NOT_DEFINED)
22       #else
23       #undef FSP_NOT_DEFINED
24
25       .p_callback = my_sci_callback,
26       .p_context = NULL, .rx_irq = VECTOR_NUMBER_RSCI5_RXI, .tx_irq = VECTOR_NUMBER_RSCI5_TXI, .tei_irq =
27         VECTOR_NUMBER_RSCI5_TEI,
28       .eri_irq = VECTOR_NUMBER_RSCI5_ERI, .rx_ipl = (3), .tx_ipl = (3), .tei_ipl = (3), .eri_ipl = (3), .p_extend =
29         &g_spi0_cfg_extend, };
30
31     /* Instance structure to use this module. */
32     const spi_instance_t g_spi0 =
33     { .p_ctrl = &g_spi0_ctrl, .p_cfg = &g_spi0_cfg, .p_api = &g_spi_on_sci_b };
34
35     void g_hal_init(void)
36     {
37         g_common_init ();
38     }

```

Figure 2-9 Generated file for SCI SPI FSP

Generated code in file **pin_data.c**

The following figure shows the generated code according to the Pin Configuration.



```

88 | (uint32_t) IOPORT_PERIPHERAL_SPI },
89 { .pin = BSP_IO_PORT_02_PIN_11, .pin_cfg = ((uint32_t) IOPORT_CFG_DRIVE_1 | (uint32_t) IOPORT_CFG_PERIPHERAL_PIN
90 | (uint32_t) IOPORT_PERIPHERAL_SPI },
91 { .pin = BSP_IO_PORT_02_PIN_12, .pin_cfg = ((uint32_t) IOPORT_CFG_DRIVE_1 | (uint32_t) IOPORT_CFG_PERIPHERAL_PIN
92 | (uint32_t) IOPORT_PERIPHERAL_SCI1_3_5_7_9_11_0D },
93 { .pin = BSP_IO_PORT_02_PIN_13, .pin_cfg = ((uint32_t) IOPORT_CFG_DRIVE_1 | (uint32_t) IOPORT_CFG_PERIPHERAL_PIN
94 | (uint32_t) IOPORT_PERIPHERAL_SCI1_3_5_7_9_11_0D },
95 { .pin = BSP_IO_PORT_02_PIN_14, .pin_cfg = ((uint32_t) IOPORT_CFG_DRIVE_1 | (uint32_t) IOPORT_CFG_PERIPHERAL_PIN
96 | (uint32_t) IOPORT_PERIPHERAL_SCI1_3_5_7_9_11_0D },
97 { .pin = BSP_IO_PORT_03_PIN_00, .pin_cfg = ((uint32_t) IOPORT_CFG_DRIVE_1 | (uint32_t) IOPORT_CFG_PERIPHERAL_PIN
98 | (uint32_t) IOPORT_PERIPHERAL_DEBUG },
99 { .pin = BSP_IO_PORT_03_PIN_01, .pin_cfg = ((uint32_t) IOPORT_CFG_DRIVE_1 | (uint32_t) IOPORT_CFG_PERIPHERAL_PIN
100 | (uint32_t) IOPORT_PERIPHERAL_DEBUG },
101 { .pin = BSP_IO_PORT_03_PIN_02, .pin_cfg = ((uint32_t) IOPORT_CFG_DRIVE_1 | (uint32_t) IOPORT_CFG_PERIPHERAL_PIN
102 | (uint32_t) IOPORT_PERIPHERAL_DEBUG },
103 { .pin = BSP_IO_PORT_03_PIN_03, .pin_cfg = ((uint32_t) IOPORT_CFG_DRIVE_1 | (uint32_t) IOPORT_CFG_PERIPHERAL_PIN
104 | (uint32_t) IOPORT_PERIPHERAL_DEBUG },
105 { .pin = BSP_IO_PORT_03_PIN_04, .pin_cfg = ((uint32_t) IOPORT_CFG_DRIVE_1 | (uint32_t) IOPORT_CFG_PERIPHERAL_PIN
106 | (uint32_t) IOPORT_PERIPHERAL_CLKOUT },

```

Figure 2-10 Generated files for Pin Configuration

2.4 Migrating Application Code

The typical steps in using the SCI SPI FSP module in an application:

1. Configure module by using FSP Configuration as described in section 2.2.
2. Initialize the module using the .open API.
3. Read and write data using the .read and .write APIs and callbacks.
4. Close the module using the .close API.

```
void hal_entry(void)
{
    /* OPEN SSPI CHANNEL */
    err = g_spi0.p_api->open(&g_spi0_ctrl, &g_spi0_cfg);

    init_demo_data(); // Prepare demo data buffers.

    /* Activate write protection for flash. */
    sci_spi_flash_set_write_protect();

    err = g_spi0.p_api->write(&g_spi0_ctrl, &demo_data_source[0], DEMO_DATA_SIZE, SPI_BIT_WIDTH_8_BITS);
    /* If there were an error this would demonstrate error detection of API calls. */
    if (FSP_SUCCESS != err)
    {
        __asm volatile ("nop"); // Your error handling code would go here.
    }
    sci_spi_event_check();
    CS_DE_ASSERT;

    R_BSP_SoftwareDelay(PAGE_PROGRAM_TIME, BSP_DELAY_UNITS_MILLISECONDS);

    /* Activate write protection for flash. */
    sci_spi_flash_set_write_protect();

    /* Read data from flash */
    err = g_spi0.p_api->write(&g_spi0_ctrl, &data[0], 4, SPI_BIT_WIDTH_8_BITS);

    /* If there were an error this would demonstrate error detection of API calls. */
    if (FSP_SUCCESS != err)
    {
        sci_spi_event_check();
    }

    err = g_spi0.p_api->read(&g_spi0_ctrl, &demo_data_dest[0], DEMO_DATA_SIZE, SPI_BIT_WIDTH_8_BITS);
    /* If there were an error this would demonstrate error detection of API calls. */
    if (FSP_SUCCESS != err)
    {
        sci_spi_event_check();
        CS_DE_ASSERT;
    }

    /* Check that contents of read destination buffer match the transmit buffer. */
    if (false != verify_data(demo_data_source, demo_data_dest, DEMO_DATA_SIZE))
    {
        else
    {

    }

    /* Done using the SCI SPI. We can shut it down now if desired to save power. */
    err = g_spi0.p_api->close(&g_spi0_ctrl);
    if (FSP_SUCCESS != err)
    {

    }

    /* Demo complete. Stay here. */
    while(1)
    {
        /* Do Nothing*/
    }
}
```

Figure 2-11 Overall SPI FSP Application Code

Initialization

FIT

FSP

FIT	FSP
<pre> void main(void) { sci_cfg_t my_sci_config; sci_err_t my_sci_err; uint32_t my_version; /* Initialize CS */ CS_ASSERT_PMR = 0; CS_ASSERT_PDR = 1; CS_DE_ASSERT; /* Array to store the version string */ uint8_t version_str[12] = {0}; /* Data buffer for SPI operations */ uint8_t data[4] = { 0 }; /* Set up the configuration data structure for SPI operation. */ my_sci_config.sync.spi_mode = SCI_SPI_MODE_3; my_sci_config.sync.bit_rate = 8000000; my_sci_config.sync.msb_first = true; my_sci_config.sync.invert_data = false; my_sci_config.sync.int_priority = 3; /* OPEN SSPI CHANNEL * Provide address of the config structure, * the callback function to be assigned, * and the location for the handle to be stored.*/ my_sci_err = R_SCI_Open(SCI_CH7, SCI_MODE_SSPI, &my_sci_config, my_sci_callback, &g_my_sci_handle); /* If there were an error this would demonstrate error detection of API calls. */ if (SCI_SUCCESS != my_sci_err) { R_BSP_NOP(); // Your error handling code would go here. } /* Initialize the I/O port pins for communication on this SCI channel. */ R_SCI_PinSet_SCI7(); </pre>	<pre> void hal_entry(void) { /* TODO: add your own code here */ fsp_err_t err = FSP_SUCCESS; /* Array to store the version string */ uint8_t version_str[12] = {0}; /* Data buffer for SPI operations */ uint8_t data[4] = { 0 }; /* OPEN SSPI CHANNEL * Provide address of the config structure, * the callback function to be assigned, * and the location for the handle to be stored.*/ err = g_spi0.p_api->open(&g_spi0_ctrl, &g_spi0_cfg); /* If there were an error this would demonstrate error detection of API calls. */ if (FSP_SUCCESS != err) { __asm volatile ("nop"); // Your error handling code would go here. } </pre>

Figure 2-12 Initialization Code

FIT: Set configuration module in application code as figure above.

FSP: Set configuration module in FSP Configuration as shown in Figure 2-5 Configure SPI Properties

Send & Receive Message

FIT	FSP
<pre> my_sci_err = R_SCI_Send(g_my_sci_handle, &data[0], 4); /* If there were an error this would demonstrate error detection of API calls. */ if (SCI_SUCCESS != my_sci_err) { R_BSP_NOP(); // Your error handling code would go here. } sci_spi_event_check(); </pre>	<pre> err = g_spi0.p_api->write(&g_spi0_ctrl, &data[0], 4, SPI_BIT_WIDTH_8_BITS); /* If there were an error this would demonstrate error detection of API calls. */ if (FSP_SUCCESS != err) { __asm volatile ("nop"); // Your error handling code would go here. } sci_spi_event_check(); </pre>

Figure 2-13 Send a message.

FIT	FSP
<pre> my_sci_err = R_SCI_Receive(g_my_sci_handle, &demo_data_dest[0], DEMO_DATA_SIZE); /* If there were an error this would demonstrate error detection of API calls. */ if (SCI_SUCCESS != my_sci_err) { R_BSP_NOP(); // Your error handling code would go here. } sci_spi_event_check(); </pre>	<pre> err = g_spi0.p_api->read(&g_spi0_ctrl, &demo_data_dest[0], DEMO_DATA_SIZE, SPI_BIT_WIDTH_8_BITS); /* If there were an error this would demonstrate error detection of API calls. */ if (FSP_SUCCESS != err) { __asm volatile ("nop"); // Your error handling code would go here. } sci_spi_event_check(); </pre>

Figure 2-14 Receive a message

Callback**FIT****FSP**

```
void my_sci_callback(void *pArgs)
{
    sci_cb_args_t *args;

    args = (sci_cb_args_t *)pArgs;

    if (SCI_EVT_XFER_DONE == args->event)
    {
        g_master_event_flag = SCI_EVT_XFER_DONE;
    }
    else
    {
        /* Updating the flag here to capture and handle all other error
        events */
        R_BSP_NOP();
    }
}
```

```
void my_sci_callback(spi_callback_args_t *p_args)
{
    if (SPI_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
        g_master_event_flag = SPI_EVENT_TRANSFER_COMPLETE;
    }
    else
    {
        /* Updating the flag here to capture and handle all other error
        events */
        __asm volatile ("nop");
    }
}
```

Figure 2-15 Callback Code

For more details about FSP sample code, please refer to this section [Reference FSP Application Code](#)

2.5 Build and Run the project

1. Hardware setup.

Connect the PMOD SPI Flash (PMOD MX25L3233F) to the PMOD1 ports on the EK-RX74M Board.

Note: Configure PMOD1 in SPI mode: J14, J15: Pin 1-2. E38: Shorted. E39: Open.

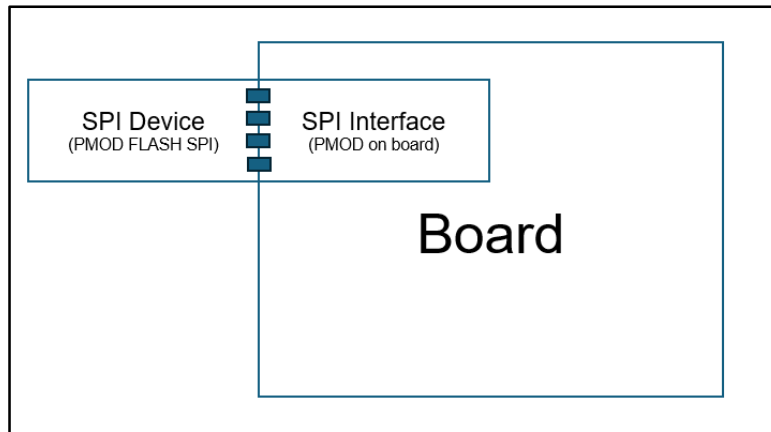


Figure 2-16 Board Connection Diagram

2. Build the sample application, download it to the board, and execute the application.

3. Check result

The project writes data to flash memory, then reads it back and compares it with the data that was originally written. The figure below shows the result of the output after execution.

If there is a match, "Success" will be displayed on the virtual console.

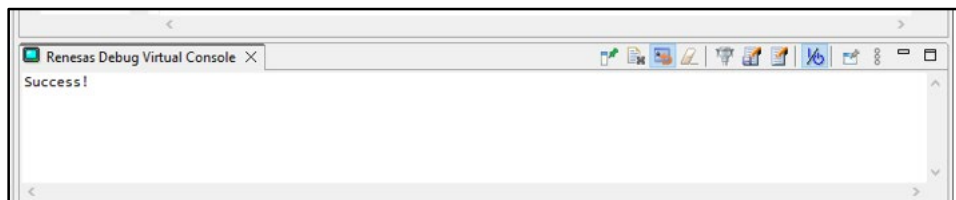


Figure 2-17 Output Renesas Debug Virtual Console

3. Reference Documents

RX Family SCI Module Using Firmware Integration Technology(R01AN1815)
Renesas Flexible Software Package (FSP)
RX FSP Migration Guide

Technical Update/Technical News
The latest information can be downloaded from the Renesas Electronics website.

4. Appendices

4.1 FIT & FSP Environment

This section describes FIT & FSP environment.

Item	Contents
FIT Module	r_sci_rx v5.00
FSP Module	RX FSP v1.0.0 Module: r_sci_b_spi
Board used	Renesas EK-RX74M (product No.: RTKEK74Mxxxxxxxxx)
Integrated development environment	Renesas Electronics e ² studio
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family GCC for Renesas RX

4.2 Reference FSP Application Code

Add this FSP sample code to file “hal_entry.c”

```
#include "hal_data.h"
#include <stdio.h>

FSP_CPP_HEADER
void R BSP WarmStart(bsp warm start event t event);
FSP_CPP_FOOTER

/*****
Macro definitions
*****/
#define DEMO_DATA_SIZE (32)
#define LOOP_COUNT (5)
#define RESET_VALUE (0x00)

#define SECTOR_ERASE_TIME (200)
#define PAGE_PROGRAM_TIME (2)

#define FLASH_CMD_WRITE_ENABLE (uint8_t) (0x06) /* Write Enable */
#define FLASH_CMD_PP (uint8_t) (0x02) /* Page Program */
#define FLASH_CMD_SE (uint8_t) (0x20) /* Sector Erase */

#define FLASH_CMD_READ (uint8_t) (0x03) /* Normal Read */
#define FLASH_CMD_READ_STATUS (uint8_t) (0x05) /* Read Status Register*/

/*****
Private global variables and functions
*****/
static bool verify_data(uint8_t * p_data_source, uint8_t * p_data_dest, uint16_t length);
static void init_demo_data(void);
static void tst_trap(void);

/* Function to check occurrence of event after data transfer */
static void sci_spi_event_check(void);
/* Function to set write protect for flash */
static void sci_spi_flash_set_write_protect(void);

static uint8_t demo_data_source[DEMO_DATA_SIZE] = {0};
static uint8_t demo_data_dest[DEMO_DATA_SIZE] = {0};

/* Master Transfer Event completion flag */
static spi_event_t g_master_event_flag;
/* Wait counter for wait operation monitoring */
static volatile int32_t g_wait_count = INT32_MAX;

/* Slave Select pin for all boards to enable data transfer */
#define CS_PIN BSP_IO_PORT_00_PIN_09

/* Function macros to assert and de-assert Slave Select pins */
#define CS_ASSERT R_IOPORT_PiWrite(&g_ioport_ctrl, CS_PIN, BSP_IO_LEVEL_LOW);
#define CS_DE_ASSERT R_IOPORT_PiWrite(&g_ioport_ctrl, CS_PIN, BSP_IO_LEVEL_HIGH);

/*****
*****
* main() is generated by the RX Configuration editor and is used to generate threads if an RTOS is
used. This function
* is called by main() when no RTOS is used.
*****
*****
void hal_entry(void)
{
    /* TODO: add your own code here */
    fsp_err_t err = FSP_SUCCESS;

    /* Array to store the version string */
    uint8_t version_str[12] = {0};
    /* Data buffer for SPI operations */
    uint8_t data[4] = { 0 };

```

```

/* OPEN SSPI CHANNEL
 * Provide address of the config structure,
 * the callback function to be assigned,
 * and the location for the handle to be stored.*/
err = g_spi0.p_api->open(&g_spi0_ctrl, &g_spi0_cfg);

/* If there were an error this would demonstrate error detection of API calls. */
if (FSP_SUCCESS != err)
{
    asm volatile ("nop"); // Your error handling code would go here.
}

init_demo_data(); // Prepare demo data buffers.

/* Activate write protection for flash. */
sci_spi_flash_set_write_protect();

/* Block erase */
data[0] = FLASH_CMD_SE;
data[1] = 0;
data[2] = 0;
data[3] = 0;

CS_ASSERT;
err = g_spi0.p_api->write(&g_spi0_ctrl, &data[0], 4, SPI_BIT_WIDTH_8_BITS);
/* If there were an error this would demonstrate error detection of API calls. */
if (FSP_SUCCESS != err)
{
    __asm volatile ("nop"); // Your error handling code would go here.
}
sci_spi_event_check();
CS_DE_ASSERT;

R_BSP_SoftwareDelay(SECTOR_ERASE_TIME, BSP_DELAY_UNITS_MILLISECONDS);

/* Activate write protection for flash. */
sci_spi_flash_set_write_protect();

/* Write data to flash */
data[0] = FLASH_CMD_PP;
data[1] = 0x00;
data[2] = 0x00;
data[3] = 0x00;

CS_ASSERT;
err = g_spi0.p_api->write(&g_spi0_ctrl, &data[0], 4, SPI_BIT_WIDTH_8_BITS);
/* If there were an error this would demonstrate error detection of API calls. */
if (FSP_SUCCESS != err)
{
    asm volatile ("nop"); // Your error handling code would go here.
}
sci_spi_event_check();
err = g_spi0.p_api->write(&g_spi0_ctrl, &demo_data_source[0], DEMO_DATA_SIZE,
SPI_BIT_WIDTH_8_BITS);
/* If there were an error this would demonstrate error detection of API calls. */
if (FSP_SUCCESS != err)
{
    __asm volatile ("nop"); // Your error handling code would go here.
}
sci_spi_event_check();
CS_DE_ASSERT;

R_BSP_SoftwareDelay(PAGE_PROGRAM_TIME, BSP_DELAY_UNITS_MILLISECONDS);

/* Activate write protection for flash. */
sci_spi_flash_set_write_protect();

/* Read data from flash */
data[0] = FLASH_CMD_READ;
data[1] = 0x00;
data[2] = 0x00;
data[3] = 0x00;

```

```

    CS_ASSERT;
    err = g_spi0.p_api->write(&g_spi0_ctrl, &data[0], 4, SPI_BIT_WIDTH_8_BITS);
    /* If there were an error this would demonstrate error detection of API calls. */
    if (FSP_SUCCESS != err)
    {
        __asm volatile ("nop"); // Your error handling code would go here.
    }
    sci_spi_event_check();
    err = g_spi0.p_api->read(&g_spi0_ctrl, &demo_data_dest[0], DEMO_DATA_SIZE,
SPI BIT WIDTH 8 BITS);
    /* If there were an error this would demonstrate error detection of API calls. */
    if (FSP_SUCCESS != err)
    {
        __asm volatile ("nop"); // Your error handling code would go here.
    }
    sci_spi_event_check();
    CS_DE_ASSERT;

    /* Check that contents of read destination buffer match the transmit buffer. */
    if (false != verify_data(demo_data_source, demo_data_dest, DEMO_DATA_SIZE))
    {
        printf("Success!\n"); // View stdout in the Renesas Virtual Debug Console window.
    }
    else
    {
        printf("Failed.\n"); // View stdout in the Renesas Virtual Debug Console window.
    }

    /* Done using the SCI SPI. We can shut it down now if desired to save power. */
    err = g_spi0.p_api->close(&g_spi0_ctrl);
    if (FSP_SUCCESS != err)
    {
        __asm volatile ("nop"); // Your error handling code would go here.
    }

    /* Demo complete. Stay here. */
    while(1)
    {
        /* Do Nothing*/
    }
}

/*****
*****
* Function Name : init_demo_data
* Description    : Put some varying data in source buffer to be transmitted and clear the destination
buffer.
* Arguments      : None
* Return value   : None
*****
*****/
static void init_demo_data(void)
{
    uint16_t i;
    for (i = 0; i < DEMO_DATA_SIZE; i++)
    {
        demo_data_source[i] = (uint8_t)(i); // Provide some sample data for transmission.
        demo_data_dest[i] = 0;              // Clear the receive data buffer.
    }
}
/* End of function init_demo_data. */

/*****
*****
* Function Name: my_sci_callback
* Description   : This is a template for an SCI SPI Mode callback function.
* Arguments     : p_args
* Return Value  : none
*****
*****/
void my_sci_callback(spi_callback_args_t *p_args)
{
    if (SPI_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
        g_master_event_flag = SPI_EVENT_TRANSFER_COMPLETE;
    }
}

```

```

    else
    {
        /* Updating the flag here to capture and handle all other error events */
        asm volatile ("nop");
    }
}

/*****
*****//**
* Function Name: sci spi event check
* Description : This function checks occurrence of data transfer complete event until timeout
occurs and handles errors.
* Arguments : none
* Return Value : none
*****
*****/
static void sci_spi_event_check(void)
{
    while(SPI_EVENT_TRANSFER_COMPLETE != g_master_event_flag)
    {
        g_wait_count--;
        if (RESET_VALUE >= g_wait_count)
        {
            printf("SPI module blocked in Write/Read operation.\r\n");
            tst_trap();
        }
    }
    g_wait_count = INT32_MAX;
    g_master_event_flag = RESET_VALUE;
}

/*****
*****//**
* @brief Sets the write protect for the SPI flash.
* @details This function sends commands to the SPI flash to set the write protect status.
* It checks the Write Enable Latch (WEL) and Write In Progress (WIP) bits to ensure
* that the flash is ready for write operations and waits until the flash is ready.
* @param[in] None
* @retval None
*****
*****/
static void sci_spi_flash_set_write_protect(void)
{
    fsp_err_t err;
    uint8_t write_data[2] = { 0 };
    uint8_t read_data[1] = { 0 };
    /* Check WEL(bit2)=1 & WIP(bit0)=0)*/
    write_data[0] = FLASH_CMD_WRITE_ENABLE;
    write_data[1] = FLASH_CMD_READ_STATUS;
    do
    {
        CS_ASSERT;
        err = g_spi0.p_api->write(&g_spi0_ctrl, &write_data[0], 1, SPI_BIT_WIDTH_8_BITS);
        /* If there were an error this would demonstrate error detection of API calls. */
        if (FSP_SUCCESS != err)
        {
            __asm volatile ("nop"); // Your error handling code would go here.
        }
        sci_spi_event_check();
        CS_DE_ASSERT;

        CS_ASSERT;
        err = g_spi0.p_api->write(&g_spi0_ctrl, &write_data[1], 1, SPI_BIT_WIDTH_8_BITS);
        /* If there were an error this would demonstrate error detection of API calls. */
        if (FSP_SUCCESS != err)
        {
            __asm volatile ("nop"); // Your error handling code would go here.
        }
        sci_spi_event_check();

        err = g_spi0.p_api->read(&g_spi0_ctrl, &read_data[0], 1, SPI_BIT_WIDTH_8_BITS);
        /* If there were an error this would demonstrate error detection of API calls. */
        if (FSP_SUCCESS != err)

```

```

    {
        __asm volatile ("nop"); // Your error handling code would go here.
    }
    sci spi event check();
    CS DE ASSERT;

    g_wait_count--;
    if (RESET_VALUE >= g_wait_count)
    {
        printf("SPI module blocked in Set write protect operation.\r\n");
        tst trap();
    }

    } while (0x02 != (read_data[0] & 0x03));
    g_wait_count = INT32_MAX;
}
/*****
*****
* Function Name : verify_data
* Description    : Compares contents of destination buffer against source buffer.
* Arguments      : p data source - pointer to transmitted data buffer.
*                  p data dest  - pointer to received data buffer.
*                  length       - Number of bytes to check.
* Return value   : true  - data is verified
*                  false - data mismatch.
*****
*****/
static bool verify_data(uint8_t * p_data_source, uint8_t * p_data_dest, uint16_t length)
{
    uint16_t i;

    for (i = 0; i < length; i++)
    {
        if(p_data_source[i] != p_data_dest[i])
        {
            return false;
        }
    }

    return true;
}
/* End of function verify_data. */

/*****
*****
* Function Name: tst_trap
* Description   : It's a trap function
* Arguments     : None
* Return Value  : None
*****
*****/
static void tst_trap(void)
{
    while(1){/*Do nothing*/}
}
/* End of function tst trap. */

/*****
*****
* This function is called at various points during the startup process. This implementation uses
the event that is
* called right before main() to set up the pins.
*
* @param[in] event Where at in the start up process the code is currently at
*****
*****/
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 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_ioport.p_cfg);
    }
}
```

```

/*****
* Function Name: my sci callback
* Description : This is a template for an SCI Async Mode callback function.
* Arguments : pArgs -
*             pointer to sci_cb_args_t structure cast to a void. Structure
*             contains event and associated data.
* Return Value : none
*****/
void my sci callback(SPI callback args t *p args)
{
    /* Receive complete */
    if (p_args->event == SPI_EVENT_RX_COMPLETE)
    {
        get event flag value = SPI EVENT RX COMPLETE;
        /* This nop is for optimization suppression. */
        __asm volatile ("nop");
    }
    /* OVERFLOW error occur */
    else if (p args->event == SPI_EVENT_ERR_OVERFLOW)
    {
        /* This nop is for optimization suppression. */
        __asm volatile ("nop");
    }
    /* FRAMING error occur */
    else if (p args->event == SPI_EVENT_ERR_FRAMING)
    {
        /* This nop is for optimization suppression. */
        __asm volatile ("nop");
    }
    /* PARITY error occur */
    else if (p args->event == SPI_EVENT_ERR_PARITY)
    {
        /* This nop is for optimization suppression. */
        __asm volatile ("nop");
    }
}
/*****
*****//**
* This function is called at various points during the startup process. This implementation uses
the event that is
* called right before main() to set up the pins.
*
* @param[in] event Where at in the start up process the code is currently at
*****
*****/
void R_BSP_WarmStart(bsp_warm_start_event_t event)
{
    if (BSP_WARM_START_RESET == event)
    {
        /* Enable reading from data flash. */
        R_FACI_LP->DFLCTL = 1U;

        /* Would normally have to wait 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. */
#ifdef
    }

    if (BSP_WARM_START_POST_C == event)
    {
        /* C runtime environment and system clocks are setup. */

        /* Configure pins. */
        R_IOPORT_Open (&g_ioport_ctrl, g_ioport.p_cfg);
    }
}

```

4.3 Comparison Reference

4.3.1 Compare Functions (Details)

Compare API functions (details)

FIT		FSP	
FIT API (Public API)		FSP Interface API (Public API)	FSP Instance API (Private API)
R_SCI_Open		open	R_SCI_B_SPI_Open
R_SCI_Close		close	R_SCI_B_SPI_Close
R_SCI_Send		write	R_SCI_B_SPI_Write
R_SCI_Receive		read	R_SCI_B_SPI_Read
R_SCI_SendReceive		writeRead	R_SCI_B_SPI_WriteRead
R_SCI_GetVersion		Not supported in specific modules FSP has common function to get FSP pack version. R_FSP_VersionGet (in bsp_api.h)	Not supported
R_SCI_Control Use Commands to configure and control.		Not supported Control functions. Settings in Properties of FSP Configuration	Not supported
Callback function set in R_SCI_Open		callbackSet	R_SCI_B_SPI_CallbackSet

Compare functions in details

Open Function	
Description	Open and configure the module.
FIT	R_SCI_Open (uint8_t const chan, sci_mode_t const mode, sci_cfg_t * const p_cfg, void (* const p_callback)(void *p_args), sci_hdl_t * const p_hdl) Initialize the SCI in SPI mode. Set callback function in this function.
FSP	open (spi_ctrl_t * p_ctrl, spi_cfg_t const * const p_cfg) SPI mode set when creating the SCI SPI module in FSP Configuration. Set callback function in FSP Configuration.

Callback Set Function	
Description	Set callback function.
FIT	R_SCI_Open (uint8_t const chan, sci_mode_t const mode, sci_cfg_t * const p_cfg, void (* const p_callback)(void *p_args), sci_hdl_t * const p_hdl) The callback function is specified during the open operation.
FSP	callbackSet (spi_ctrl_t *const p_ctrl, void(*) (spi_callback_args_t *) p_callback, void const *const p_context, spi_callback_args_t *const p_callback_memory) User can specify the callback function in the FSP Configuration.

Close Function	
Description	Close operation.
FIT	R_SCI_Close (sci_hdl_t const hdl)
FSP	close (spi_ctrl_t *const p_ctrl) FSP is similar to FIT.

Receive Function	
Description	Receive data.
FIT	R_SCI_Receive (sci_hdl_t const hdl, uint8_t *p_dst, uint16_t const length) Specify pointer to data to receive and length (number of bytes).
FSP	read (spi_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const length, spi_bit_width_t const bit_width) FSP is similar to FIT, specify length for data receive FSP needs to specify Data bit width to be received

Send Function	
Description	Send data.
FIT	R_SCI_Send (sci_hdl_t const hdl, uint8_t *p_src, uint16_t const length) Specify pointer to data to transmit and length (number of bytes).
FSP	write (spi_ctrl_t *const p_ctrl, void const * p_src, uint32_t const length, spi_bit_width_t const bit_width) FSP is similar to FIT, specify length for data transfer FSP needs to specify Data bit width to be transferred

Send & Receive Function	
Description	Send & Receive data.
FIT	R_SCI_SendReceive (sci_hdl_t const hdl, uint8_t *p_src, uint8_t *p_dst, int16_t const length) Transmits and receives data simultaneously.
FSP	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) Transmit data to an SPI device while receiving data from an SPI device (full-duplex). In FSP, must specify the data bit width to be transferred.

Additional Functions	
Description	List additional functions or specific function which only have in FIT but don't have in FSP and vice versa.
FIT	Get Version Function: R_SCI_GetVersion(void) FIT provides a function to return the driver version number at runtime, not present in FSP.
FSP	None.

4.3.2 Compare Configuration (Details)

Compare FIT and FSP Configuration:

FIT Configuration	FSP Configuration
Parameter Checking	Common> Parameter Checking
Enable DTC/DMACA for TX/RX channel	Common> DTC Support Note: DMAC is not supported
Use SSPI Mode	Not supported. Note: FSP: SPI = SPI driver/module (stack)
Operating Mode: Default Master mode (SCI FIT only supported Master mode)	Module SPI>Operating Mode
Include software support for SCI channel	Module SPI> Channel
Configure module in user application code: • Bitrate • Bit Order • Clock source • Clock Phase • Clock Polarity • Callback function • Interrupts priority	Set in FSP Configuration>Module SPI > Bitrate > Bit Order > Clock Source > Clock Phase > Clock Polarity > Callback > Receive Interrupt Priority > Transmit Interrupt Priority
Setting priority level for ERI, TEI interrupt	Set in FSP Configuration>Module SPI > Transmit End Interrupt Priority > Error Interrupt Priority
Include software support for nested interrupt (TXI/RXI/TEI/ERI)	Not supported
Not supported	Module SPI> Mode Fault Error

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Sep 30, 2026	-	First edition issued

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

2. Processing at power-on

The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.

3. Input of signal during power-off state

Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.

4. Handling of unused pins

Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.

5. Clock signals

After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.

6. Voltage application waveform at input pin

Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).

7. Prohibition of access to reserved addresses

Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

8. Differences between products

Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

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 be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. 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.
6. 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.

7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENASAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENASAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENASAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENASAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENASAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(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.5.0-1 October 2020)

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/.