RX Family

How to Change Transfer Data Length During RSPI Communication Using a DTC

Introduction

This application note describes how to change the transfer data length during serial peripheral interface (RSPI) communication using a data transfer controller (DTC), using the RX660 group as an example.

Target Devices

RX Family

Confirmed Devices

RX660 Group

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Contents

1. SSL Signal When the RSPI Transfer Data Length Changes

1.1 SSL Signal Controlled by Hardware

During RSPI communication, the transfer data length can be changed only after the current transfer finishes.

When the RSPI communication terminates, the SSL signal is negated by hardware control. Therefore, as shown in Figure 1.1, assertion and negation of the SSL signal occur when the transfer data length changes.

If you want the SSL signal to remain asserted throughout the data communication, you need to control the SSL signal by using a general-purpose port.

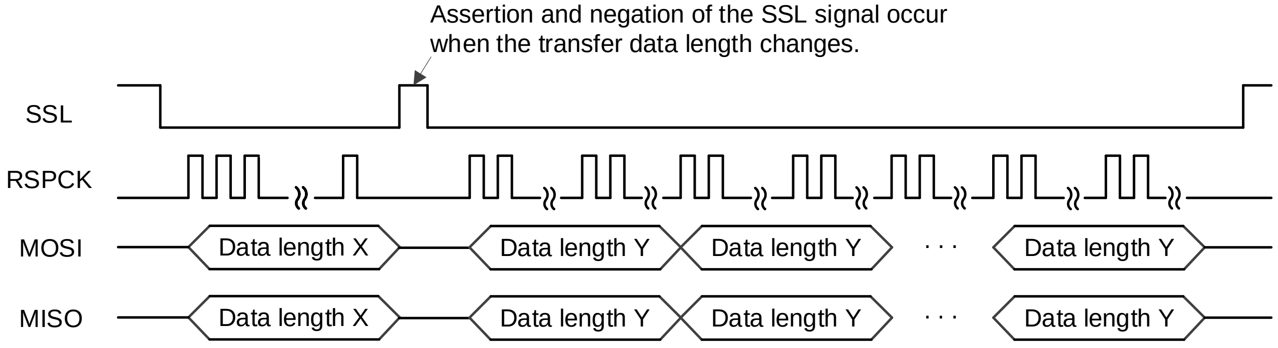


Figure 1.1 SSL Signal Controlled by Hardware When the Transfer Data Length Changes

1.2 SSL Signal Generation Controlled by a General-Purpose Port

In this application note, a general-purpose port is used to control the SSL signal by software rather than by hardware.

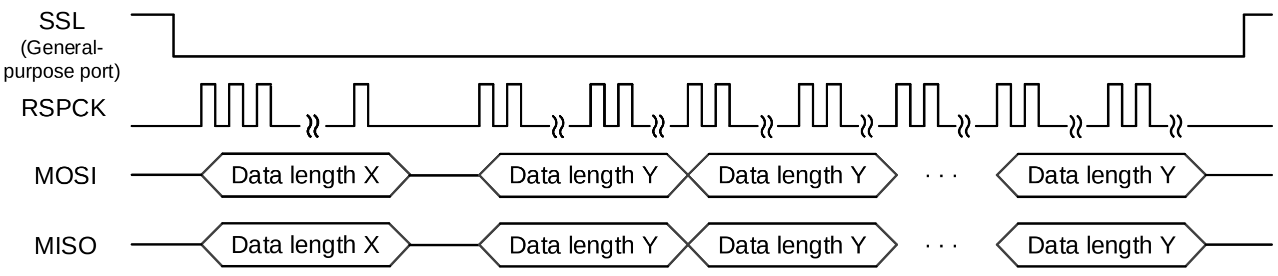


Figure 1.2 SSL Signal Controlled Using a General-Purpose Port When the Transfer Data Length Changes

2. Hardware Configuration

Figure 2.1 shows Hardware Configuration.

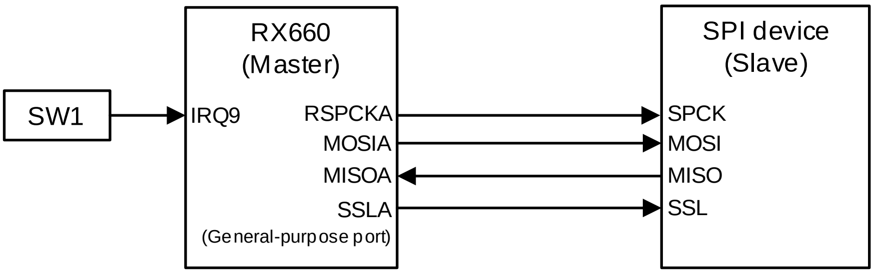


Figure 2.1 Hardware Configuration

Table 2.1 shows RSPI Pins Used for Connecting RX660 and the SPI Slave Device.

Table 2.1 RSPI Pins Used for Connecting RX660 and the SPI Slave Device

Pin Name	I/O	Port Used	Function
RSPCKA	Output	PA5	Clock I/O
MOSIA	Output	PA6	Master transmit data I/O
MISOA	Input	PA7	Slave transmit data I/O
SSLA	Output	PA2	Slave selection output by general-purpose port control

In this application note, SW1 installed on the Renesas Starter Kit+ for RX660 (RSK) board is used to start RSPI communication.

Table 2.2 shows External Pin Interrupt Assigned to SW1 Input.

Table 2.2 External Pin Interrupt Assigned to SW1 Input

Pin Name	Port Used	Function
IRQ9	P91	Detects that SW1 is pressed and then starts RSPI communication.

3. Operation Confirmation Conditions

Table 3.1 Operation Confirmation Conditions

Item	Description
MCU used	R5F56609HDFB (RX660 Group)
Operating frequency	• Main clock: 24 MHz • PLL: 240 MHz (Main clock, divided by 1, multiplied by 10) • System clock (ICLK): 120 MHz (PLL divided by 2) • Peripheral module clock A (PCLKA): 120 MHz (PLL divided by 2) • Peripheral module clock B (PCLKB): 60 MHz (PLL divided by 4)
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics e2 studio Version 2024-01 (24.1.0)
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.06.00
	Compile options -lang = c99
iodefine.h version	Version 1.00
Endian order	Little endian
Operating mode	Single-chip mode
Processor mode	Supervisor mode
Sample program version	Version 1.00
Board used	Renesas Starter Kit for RX660 (Product No.: RTK556609xxxxxxxxx)

4. Description of Software

In this application note, when SW1 installed on RSK is pressed, 1-frame communication is performed using a data length of 24 bits. Then, the transfer data length is changed to 16 bits and 8-frame communication is performed.

SW1 is connected to IRQ9. Table 4.1 shows Setting of IRQ9 (Used to Detect Pressing of SW1).

Table 4.1 Setting of IRQ9 (Used to Detect Pressing of SW1)

Item	Settings
Detection type	Falling edge
Digital filter setting	PCLK/64

Table 4.2 and Table 4.3 show the RSPI settings for communication when the transfer data length is 24 bits and 16 bits, respectively.

Table 4.2 RSPI Settings for Communication When the Transfer Data Length Is 24 Bits

Item	Settings
RSPCK clock	125 kHz
Bit length	24 bits
Number of frames	1 frame
Format	MSB first
RSPCK phase	Data variation on odd edge, data sampling on even edge.
RSPCK polarity	Low when idle
SSL polarity	Active Low
SSL negation operation	Keeps the SSL signal level from transfer end until next access start
RSLCK delay	1 RSPCK
SSL negation delay	1 RSPCK
Next access delay	1 RSPCK + 2 PCLK

Table 4.3 Settings for Communication When the Transfer Data Length Is 16 Bits

Item	Settings
RSPCK clock	125 kHz
Bit length	16 bits
Number of frames	1 frame
Format	MSB first
RSPCK phase	Data variation on odd edge, data sampling on even edge.
RSPCK polarity	Low when idle
SSL polarity	Active Low
SSL negate operation	Keeps the SSL signal level from transfer end until next access start
RSLCK delay	1 RSPCK
SSL negation delay	1 RSPCK
Next access delay	1 RSPCK + 2 PCLK

Table 4.4 shows RSPI Interrupts Used.

Table 4.4 RSPI Interrupts Used

Interrupt	Description
SPTI0	Transmit buffer empty interrupt
SPRI0	Receive data full interrupt
SPEI0	Error interrupt
SPCI0	Communication end interrupt

In this application note, both data transmission and reception use the DTC.

Because the DTC settings differ depending on the transfer data length and whether data is to be sent or received, the DTC settings are classified as follows.

DTC transfer A: DTC settings for data transmission (transfer data length: 24 bits)

DTC transfer B: DTC settings for data reception (transfer data length: 24 bits)

DTC transfer C: DTC settings for data transmission (transfer data length: 16 bits)

DTC transfer D: DTC settings for data reception (transfer data length: 16 bits)

Table 4.5 to Table 4.8 show the respective DTC settings.

Table 4.5 DTC Transfer A: DTC Settings for Data Transmission (Transfer Data Length: 24 Bits)

Item	Description
Activation Source	SPTI0 interrupt
Transfer mode	Normal transfer
Transfer data size	32 bits
Interrupt settings	An interrupt request to the CPU is generated when specified data transfer is completed.
Source address	Address fixed
Destination address	Address fixed
Source register (SAR)	0x3000 (RAM area address)
Destination register (DAR)	SPDR register address
Transfer count register A (CRA)	0x0001
Transfer count register B (CRB)	0x0000

Table 4.6 DTC Transfer B: DTC Settings for Data Reception (Transfer Data Length: 24 Bits)

Item	Description
Activation Source	SPRI0 interrupt
Transfer mode	Normal transfer
Transfer data size	32 bits
Interrupt settings	An interrupt request to the CPU is generated when specified data transfer is completed.
Source address	Address fixed
Destination address	Address fixed
Source register (SAR)	SPDR register address
Destination register (DAR)	0x2000 (RAM area address)
Transfer count register A (CRA)	0x0001
Transfer count register B (CRB)	0x0000

Table 4.7 DTC Transfer C: DTC Settings for Data Transmission (Transfer Data Length: 16 Bits)

Item	Description
Activation Source	SPTI0 interrupt
Transfer mode	Normal transfer
Transfer data size	16 bits
Interrupt settings	An interrupt request to the CPU is generated when specified data transfer is completed.
Source address	Address fixed
Destination address	Address fixed
Source register (SAR)	Address of g_w16_data
Destination register (DAR)	SPDR register address
Transfer count register A (CRA)	0x0008
Transfer count register B (CRB)	0x0000

Table 4.8 DTC Transfer D: DTC Settings for Data Reception (Transfer Data Length: 16 Bits)

Item	Description
Activation Source	SPRI0 interrupt
Transfer mode	Normal transfer
Transfer data size	16 bits
Interrupt settings	An interrupt request to the CPU is generated when specified data transfer is completed.
Source address	Address fixed
Destination address	Address incremented
Source register (SAR)	SPDR register address
Destination register (DAR)	Address of g_r16_data[0]
Transfer count register A (CRA)	0x0008
Transfer count register B (CRB)	0x0000

4.1 Description of Operation

4.1.1 Communication When the Transfer Data Length is 24 Bits

When SW1 is pressed, an IRQ9 interrupt request is generated. The IRQ9 interrupt handler changes the SSL signal controlled by the general-purpose port (PA2) to the Low level, and then starts communication for 24-bit transfer data length.

The communication for 24-bit transfer data length sends and receives one frame.

Figure 4.1 shows Timing Chart for Communication When the Transfer Data Length Is 24 Bits.

Data transmission:

Using an SPTI0 interrupt as an activation source, DTC transfer A in Table 4.5 is performed to write 24-bit data to the SPDR register (the DTC transfers 32 bits, of which the lower 24 bits will be valid data).
After DTC transfer A is performed, an SPTI0 interrupt request to the CPU is generated.
The SPTI0 interrupt handler disables the SPTI0 interrupt (SPCR.SPTIE=0).

Data reception:

Using an SPRI0 interrupt as an activation source, DTC transfer B in Table 4.6 is performed to read 24-bit data from the SPDR register (the DTC transfers 32 bits, of which the lower 24 bits will be valid data).
After DTC transfer B is performed, an SPRI0 interrupt request to the CPU is generated.
The SPRI0 interrupt handler enables the SPCI0 interrupt (SPCR3.SPCIE=1).

Communication completion processing:

The SPCI0 interrupt handler changes the transfer data length to 16 bits, and then changes the DTC settings according to DTC transfer C for sending 16-bit data (Table 4.7) and DTC transfer D for receiving 16-bit data (Table 4.8). Then, communication for 16-bit transfer length starts.

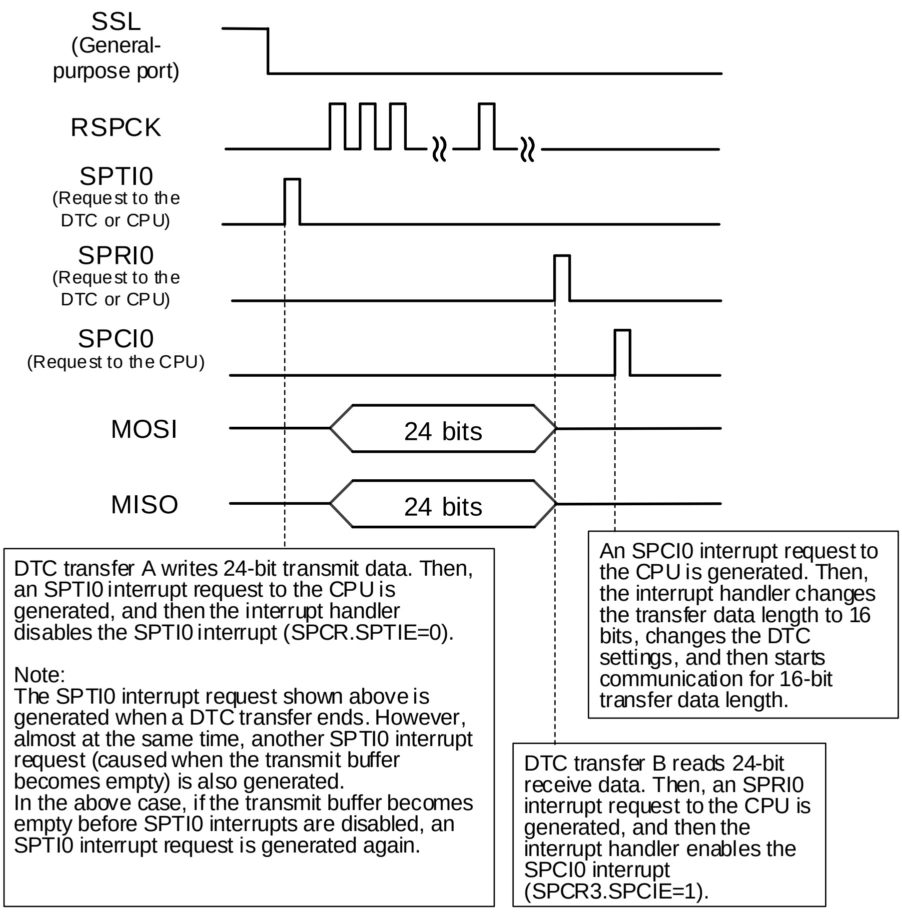


Figure 4.1 Timing Chart for Communication When the Transfer Data Length Is 24 Bits

4.1.2 Communication When the Transfer Data Length Is 16 Bits

When communication for 24-bit transfer data length finishes, communication for 16-bit transfer data length starts.

The communication for 16-bit transfer data length sends and receives eight frames.

Figure 4.2 shows Timing Chart for Communication When the Transfer Data Length is 16 Bits.

Data transmission:

Using an SPTI0 interrupt as an activation source, DTC transfer C in Table 4.7 is performed to write 16-bit data to the SPDR register.
When DTC transfer C activated by an SPTI0 interrupt request has transferred eight frames, another SPTI0 interrupt request to the CPU is generated.
The SPTI0 interrupt handler disables the SPTI0 interrupt (SPCR.SPTIE=0).

Data reception:

Using an SPRI0 interrupt as an activation source, DTC transfer D in Table 4.8 is performed to read 16-bit data from the SPDR register.
When DTC transfer D activated by an SPRI0 interrupt request has transferred eight frames, another SPRI0 interrupt request to the CPU is generated.
The SPRI0 interrupt handler enables the SPCI0 interrupt (SPCR3.SPCIE=1).

Communication completion processing:

The SPCI0 interrupt handler changes the SSL signal controlled by the general-purpose port (PA2) to High level, changes the transfer data length to 24 bits, and then changes the DTC settings according to DTC transfer A for sending 24-bit data (Table 4.5) and DTC transfer B for receiving 24-bit data (Table 4.6). Then, when SW1 is pressed, communication for 24-bit transfer data length starts as described in 4.1.1 Communication When the Transfer Data Length is 24 Bits.

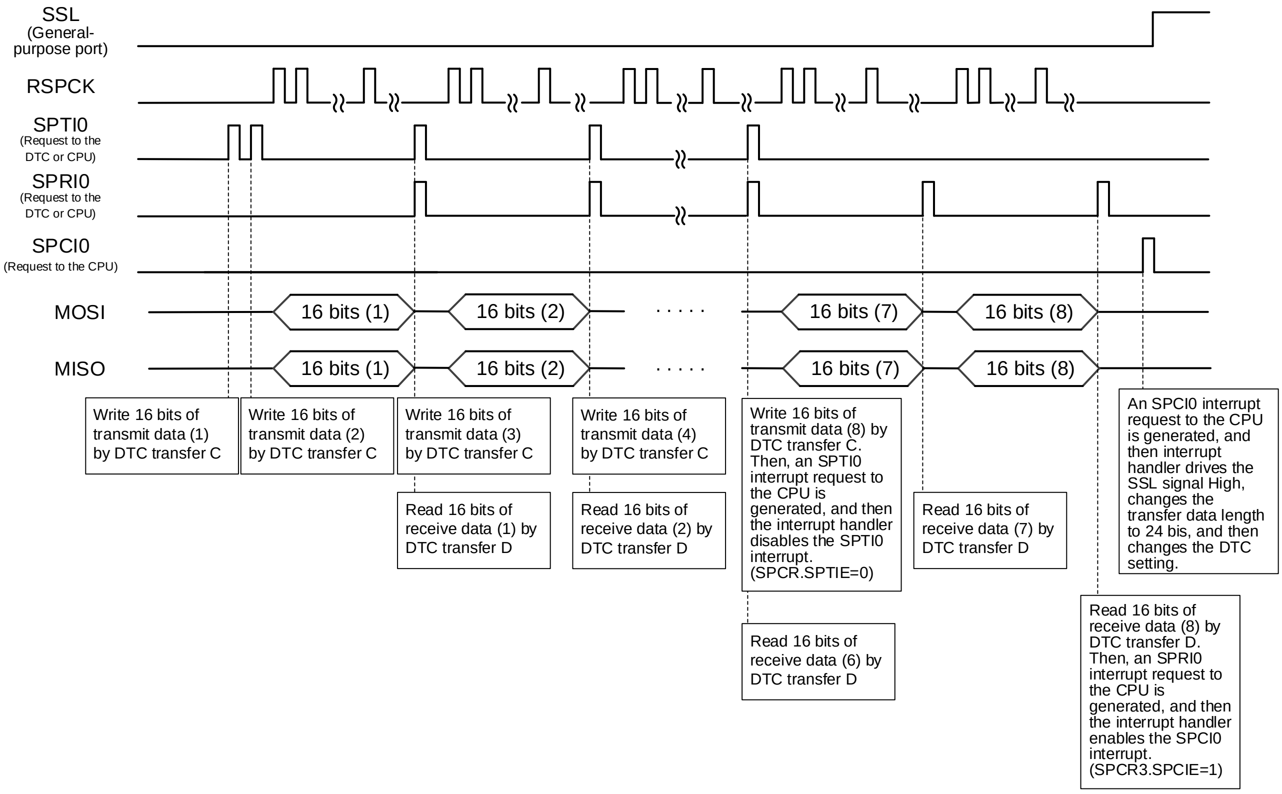


Figure 4.2 Timing Chart for Communication When the Transfer Data Length is 16 Bits

4.2 Components Used for Firmware Integration Technology (FIT) Modules and Code Generation

Table 4.9 shows Components Used for FIT Modules and Code Generation.

Table 4.9 Components Used for FIT Modules and Code Generation

Component	Category	Use
Board Support Packages (BSP)	FIT module	Provides all codes from reset to the main() function
Interrupt controller	Code generation	ICU settings
Data transfer controller	Code generation	DTC settings
SPI operation mode (4-wire method)	Code generation	RSPI settings

4.2.1 Smart Configurator (SC) Settings for FIT Module Component

This application note uses BSP modules that are automatically generated when a new project is created, but the SC settings for the BSP are not changed.

Table 4.10 shows the major clock settings only.

Table 4.10 BSP Module Clock Settings

Item	Settings
System clock settings	Clock source: PLL circuit output 240 MHz System clock (ICLK): x1/2 ··· 120 (MHz) Peripheral module clock (PCLKA): x1/2 ··· 120 (MHz) Peripheral module clock (PCLKB): x1/4 ··· 60 (MHz) Peripheral module clock (PCLKD): x1/4 ··· 60 (MHz) Bus clock (BCLK): x1/4 ··· 60 (MHz) FlashIF clock (FCLK): x1/4 ··· 60 (MHz)
Sub-clock oscillator setting	Operating (Default setting. Sub-clocks are not used.)
HOCO clock setting	Stop
LOCO clock setting	Stop
IWDT-dedicated clock setting	Stop

4.2.2 SC Settings for Code Generation Components

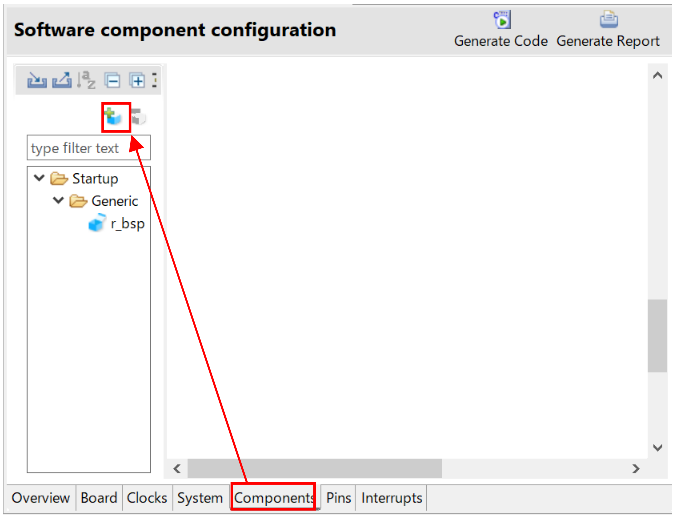
The first communication after the power is turned on uses 24 bits for the transfer data length.

Therefore, the SC settings described in this section apply to communication for 24-bit transfer data length.

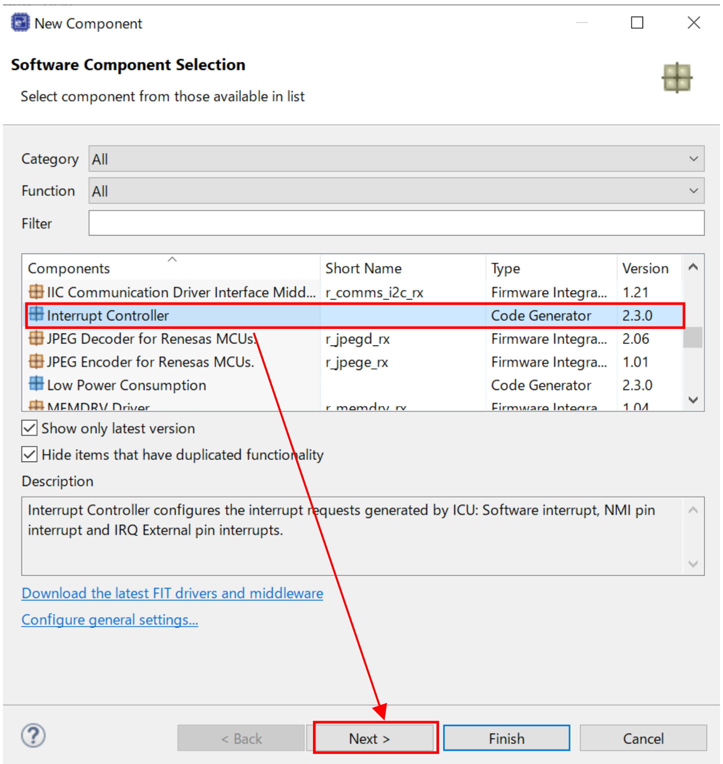
Note that the settings necessary for communication for 16-bit transfer data length must be manually added to the code generated by SC.

4.2.2.1 Interrupt controller configuration

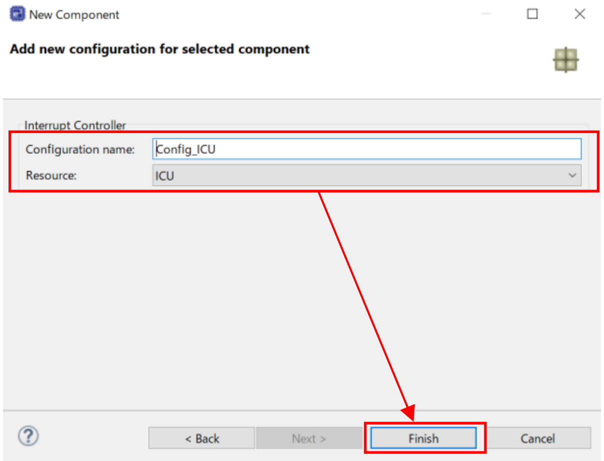
(1) Open the [Components] tab, and then click the icon for adding a component.



(2) In the [Software Component Selection] window, select [Interrupt Controller], and then click [Next].

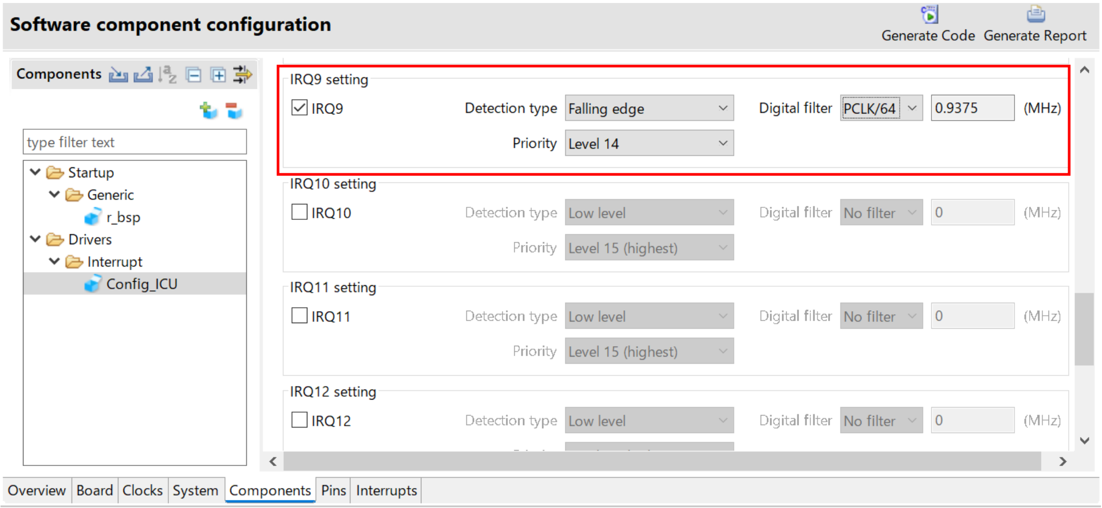


(3) In the window for adding the component, click [Finish].



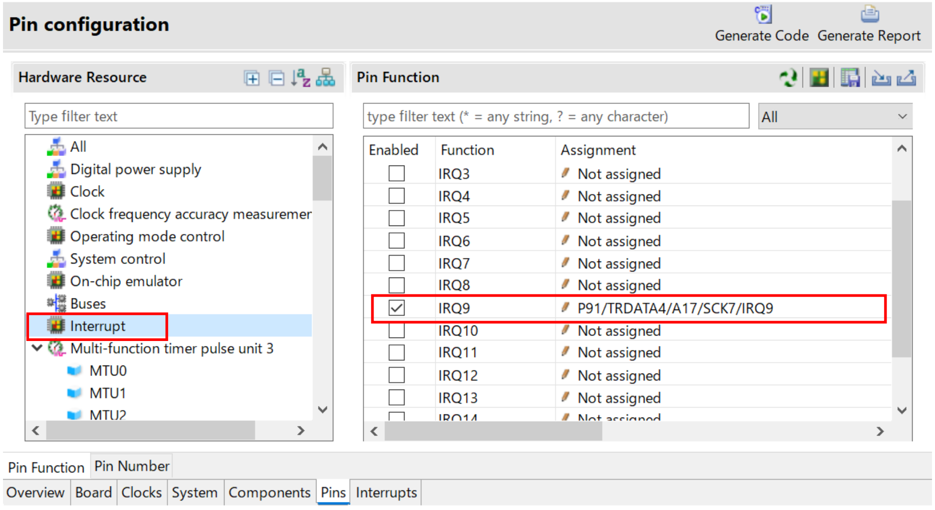
(4) Software component configuration

Assign the pin (P91) connected to SW1 of RSK to IRQ9.



(5) Pin configuration

Open the [Pins] tab, select [Interrupt] from under [Hardware Resource], and then assign P91 to IRQ9 included in the pin functions.

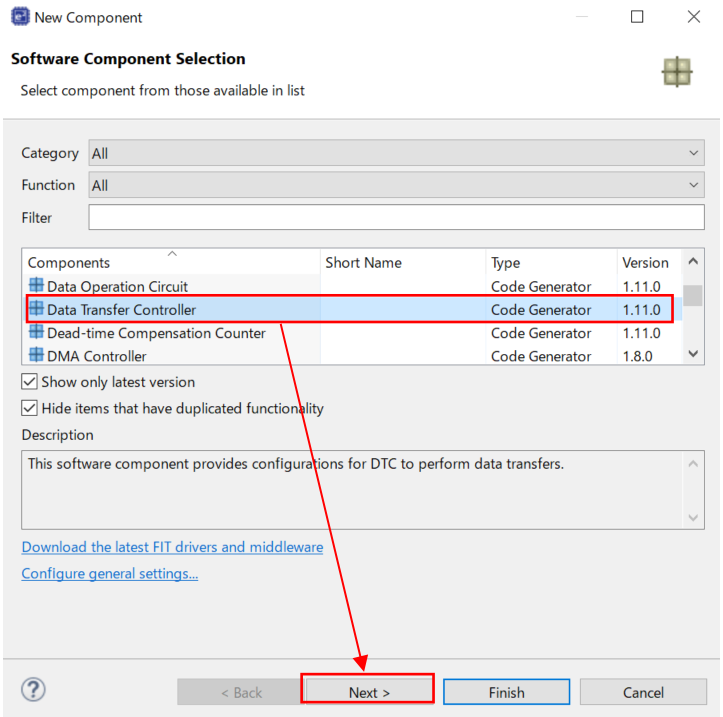


4.2.2.2 Data transfer controller configuration (DTC settings for data transmission (transfer data length is 24 bits))

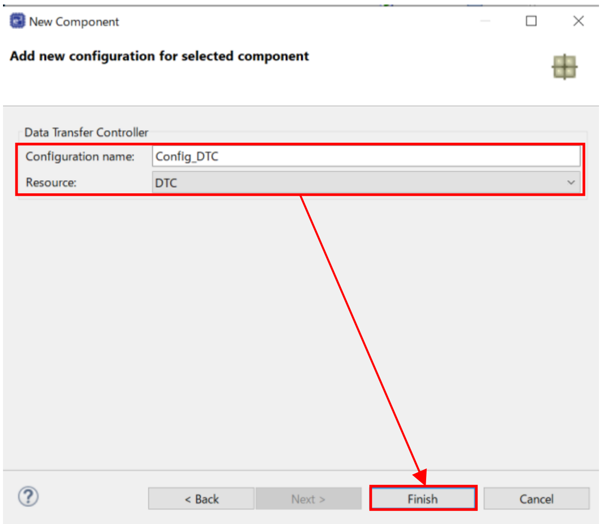
(1) Open the [Components] tab, and then click the icon for adding a component.

For details about the selection window, see 4.2.2.1(1).

(2) In the [Software Component Selection] window, select [Data Transfer Controller], and then click [Next].



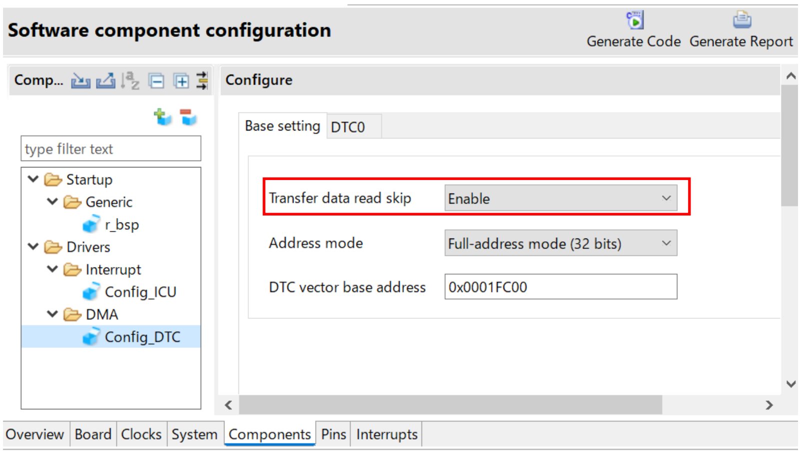
(3) In the window for adding the component, click [Finish].



(4) Software component configuration

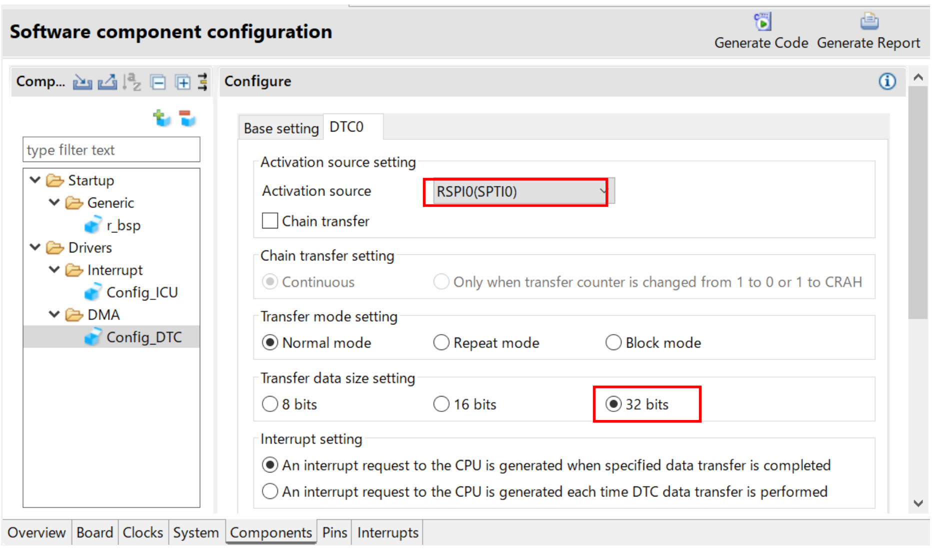
In this application note, the items on the [Base setting] tab are specified as follows.

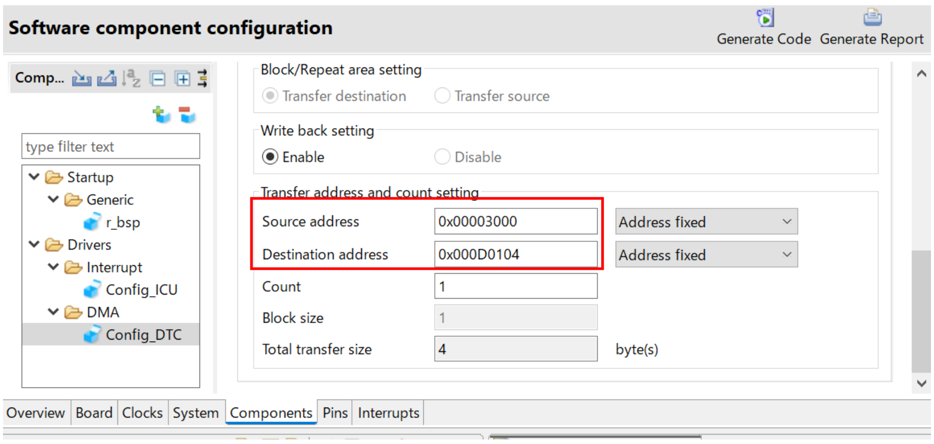
The setting in the red frame has been changed from the default.



The items on the [DTC0] tab are specified as follows.

The settings in the red frames have been changed from the defaults.





4.2.2.3 Data transfer controller configuration (DTC settings for data reception (transfer data length is 24 bits))

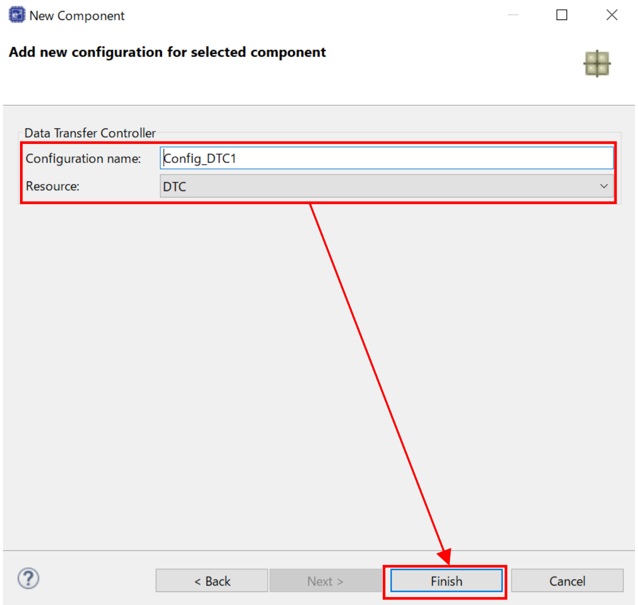
(1) Open the [Components] tab, and then click the icon for adding a component.

For details about the selection window, see 4.2.2.1(1).

(2) In the [Software Component Selection] window, select [Data Transfer Controller], and then click [Next].

For details about the selection window, see 4.2.2.2 (2).

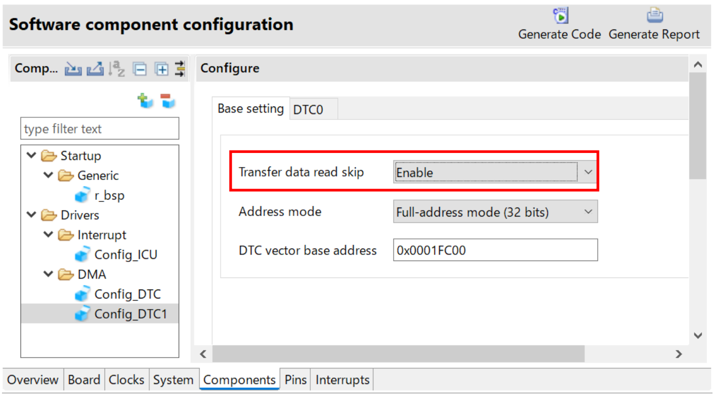
(3) In the window for adding the component, click [Finish].



(4) Software component configuration

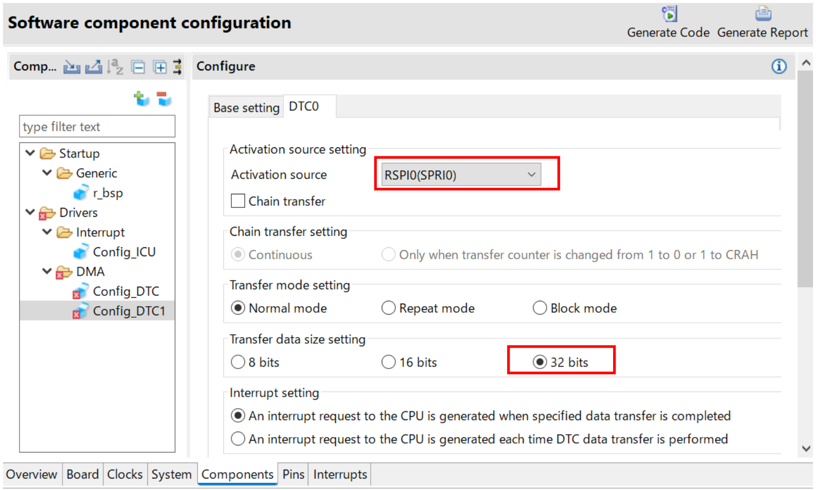
In this application note, the items on the [Base setting] tab are specified as follows.

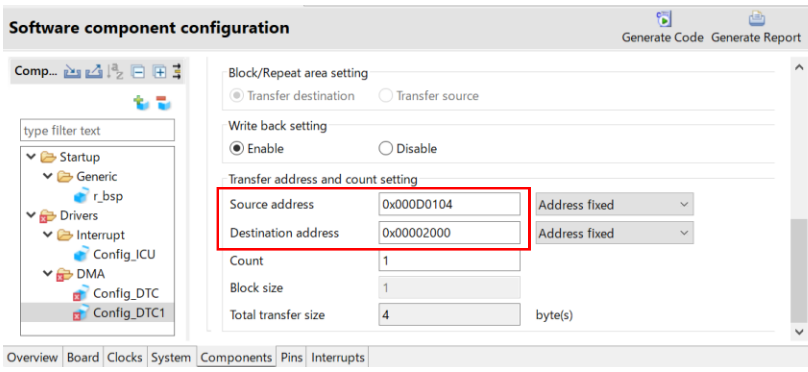
The setting in the red frame has been changed from the default.



The items on the [DTC0] tab are specified as follows.

The settings in the red frames have been changed from the defaults.



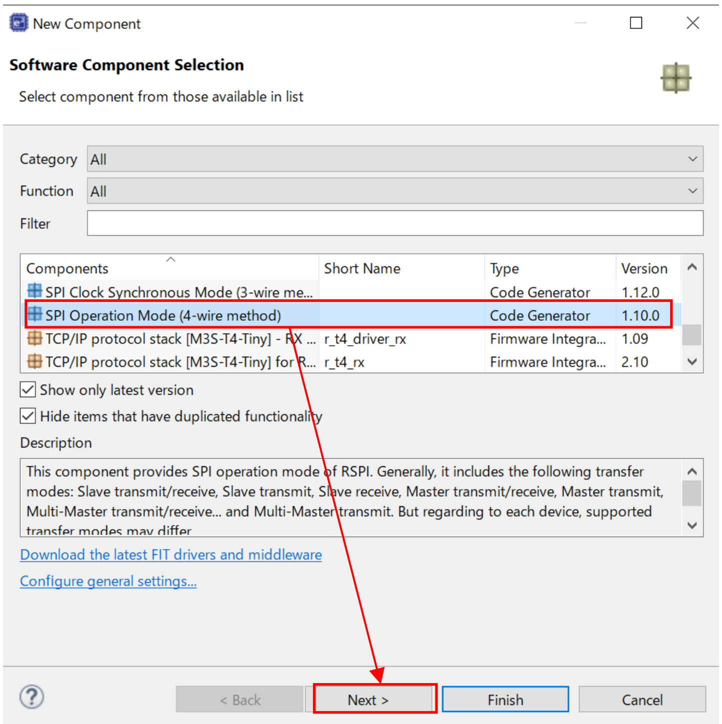


4.2.2.4 SPI operation mode (4-wire method) configuration

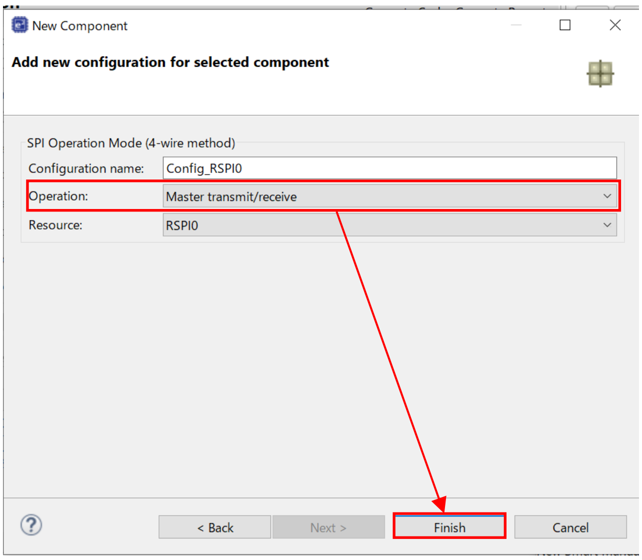
(1) Open the [Components] tab, and then click the icon for adding a component.

For details about the selection window, see 4.2.2.1(1).

(2) In the [Software Component Selection] window, select [SPI Operation Mode (4-wire method)], and then click [Next].



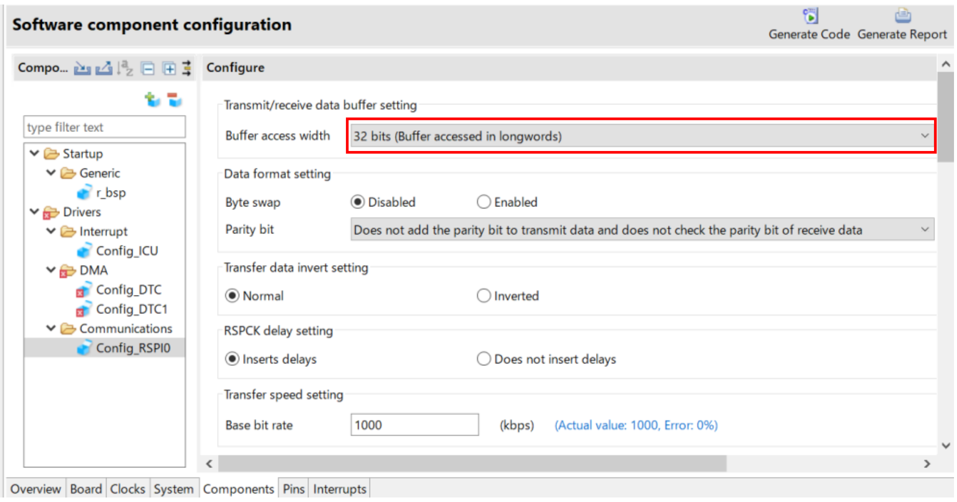
(3) In the window for adding the component, select [Master transmit/receive] for [Operation], and then click [Finish].

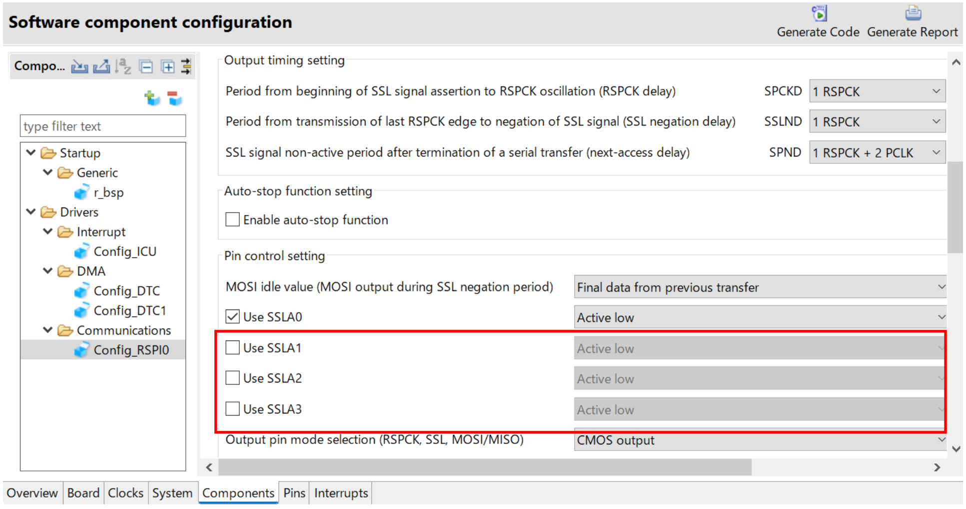


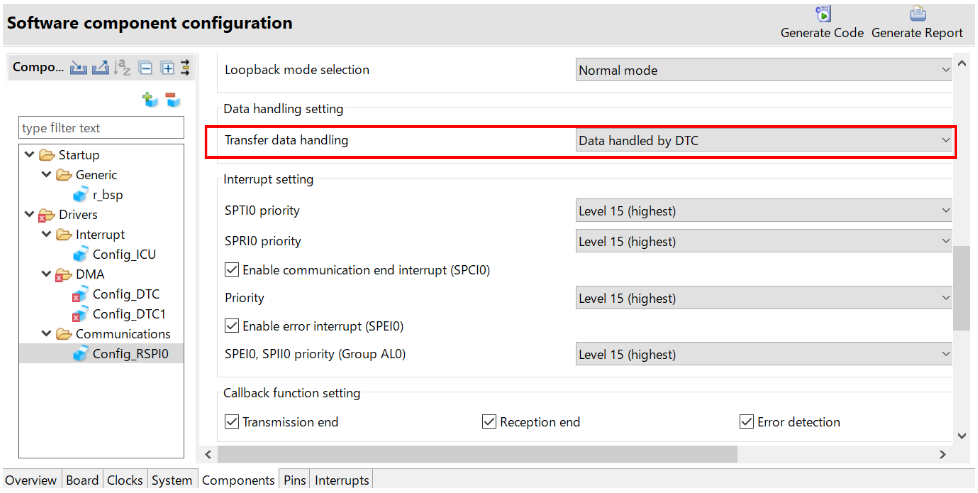
(4) Software component configuration

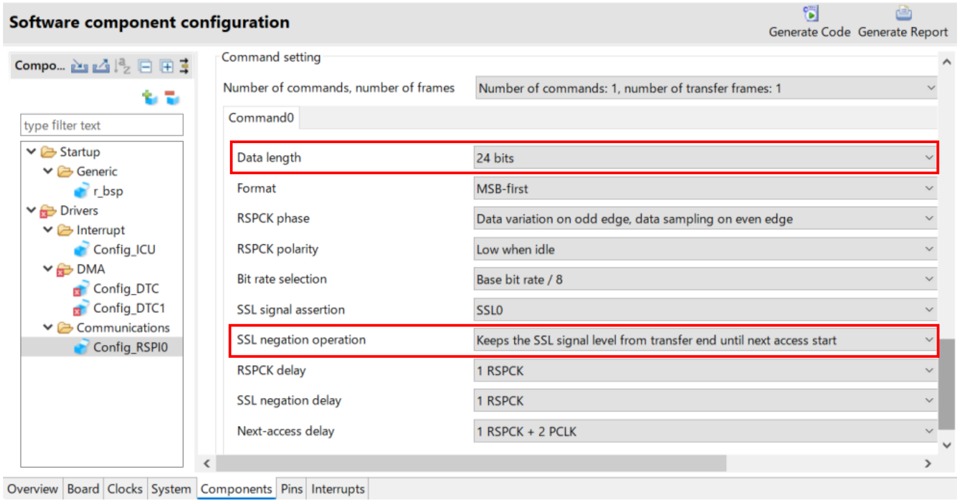
In this application note, the items are specified as follows.

The settings in red frames have been changed from the defaults.









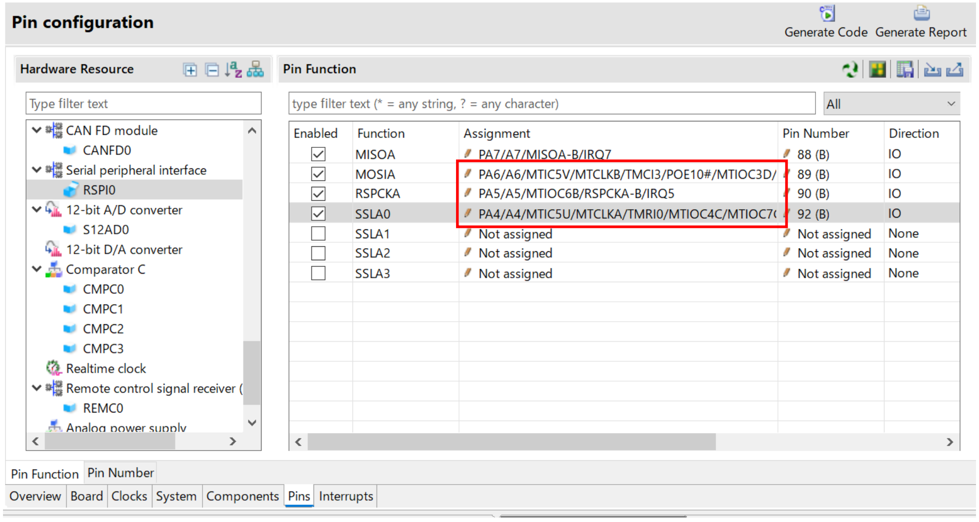
(5) Pin configuration

Open the [Pins] tab, select [RSPI0] from the hardware resources, and then assign the pins to be used for RSPI.

In this application note, pins that are open on the RSK are assigned.

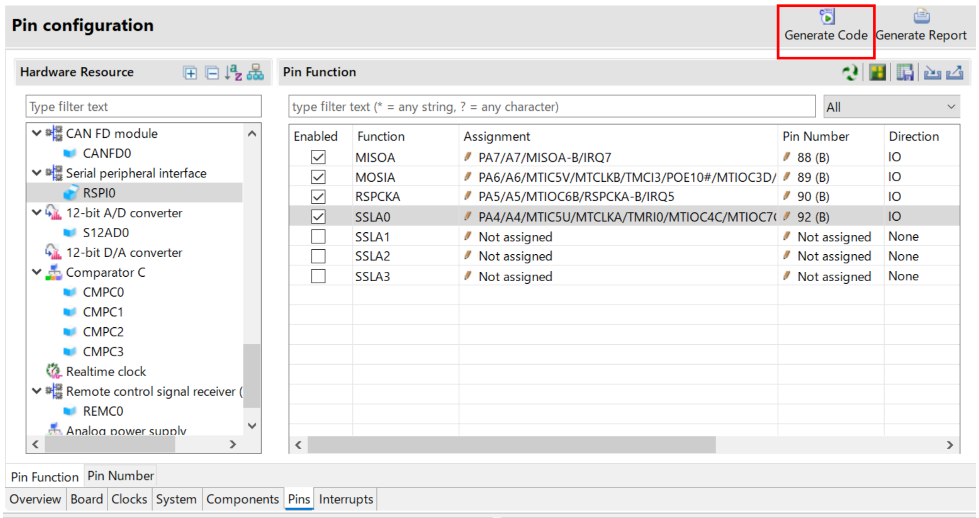
Note that the SSLA0 pin is not used.

The settings in the red frame have been changed from the defaults.

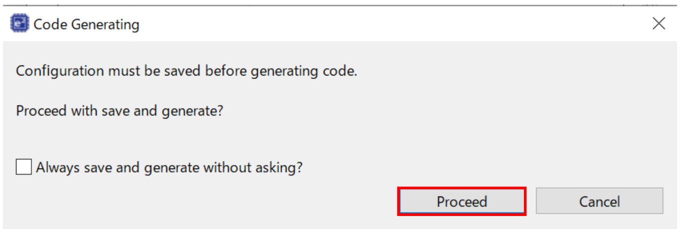


4.2.3 Generating Code

When all the SC settings are complete, click [Generate Code] to generate code.



When the following dialog box appears, click [Proceed].



4.2.4 Adding Code to the SC-Generated Code

You can add user code between the following lines in a source file or header file:
/* Start user … */
/* End user … */

4.2.4.1 Additional processing to the SC-generated code

To the code generated by SC, add the processing that is necessary for 16-bit data transfer (such as changing the transfer data length and DTC settings) and the IRQ9 interrupt handler to be run after SW1 is pressed.

Table 4.11 shows Additions to SC-Generated Code.

Table 4.11 Additions to SC-Generated Code

Folder	File	Changed or Added Function	Description
src	data_length_change_sample_for_rspi_dtc.c	void main(void)	The processing to drive the SSL signal High using general-purpose port control, clear the 24-bit communication completion check flag (g_length_check), and enable IRQ9 was added.
src\sm_gen\Config_ICU	Config_ICU_user.c	static void r_Config_ICU_irq9_interrupt(void)	The processing to initialize transmit data, initialize the receive data storage RAM, start the DTC, assert the SSL signal, and start RSPI communication was added.
src\sm_gen\Config_DTC	Config_DTC.c	void set_16bit_data_transfer_mode(void)	The void set_16bit_data_transfer_mode (void) function was added. This function sets the DTC transfer information shown in Table 4.7 DTC Transfer C: DTC Settings for Data Transmission (Transfer Data Length: 16 Bits).
	Config_DTC.h	-	The prototype declaration of the void set_16bit_data_transfer_mode(void) function was added. The value of the transfer count register A (CRA) to be specified for this function was added as a macro constant.
src\sm_gen\Config_DTC1	Config_DTC1.c	void set_16bit_data_receive_mode(void)	The void set_16bit_data_receive_mode(void) function was added. This function sets the DTC transfer information shown in Table 4.8 DTC Transfer D: DTC Settings for Data Reception (Transfer Data Length: 16 Bits).
	Config_DTC1.h	-	The prototype declaration of the void set_16bit_data_receive_mode(void) function was added. The value of the transfer count register A (CRA) to be specified for this function was added as a macro constant.
src\sm_gen\Config_RSPI0	Config_RSPI0_user.c	static void r_Config_RSPI0_communication_end_interrupt(void)	The following processing was added: • When 24-bit communication is complete Processing to change the transfer data length to 16 bits, change the DTC settings, start the DTC, and start RSPI communication • When communication of 16 bits x 8 frames is complete Processing to negate the SSL signal, change the transfer data length to 24 bits, and change the DTC settings
		static void r_Config_RSPI0_callback_transmitend(void)	The processing to disable SPTI0 interrupts was added.
		static void r_Config_RSPI0_callback_receiveend(void)	The processing to enable SPCI0 interrupts was added.
	Config_RSPI0.h	-	The macro definition of the general-purpose port used for SSL control was added.

4.2.4.2 Constants added to the SC-generated code

Table 4.12 shows List of Constants Added to the SC-Generated Code.

Table 4.12 List of Constants Added to the SC-Generated Code

Constant Name	Setting	Description
SSL_PORT_PODR_BIT	PORTA.PODR.BIT.B2	PODR register bits of the general-purpose port used for SSL control
SSL_PORT_PDR_BIT	PORTA.PDR.BIT.B2	PDR register bits of the general-purpose port used for SSL control
CRA_16BIT_TRANSFER	0x0008	Value of the DTC transfer count register A (CRA) for 16-bit transmission
CRA_16BIT_RECEIVE	0x0008	Value of the DTC transfer count register A (CRA) for 16-bit reception

4.2.4.3 Variables added to the SC-generated code

Table 4.13 shows List of Variables Added to the SC-Generated Code.

Table 4.13 List of Variables Added to the SC-Generated Code

Type	Variable Name	Description	Functions Using the Variable
volatile uint8_t	g_length_check	Transfer data length check flag 0: Transfer data length is 24 bits 1: Transfer data length is 16 bits	main() r_Config_ICU_irq9_interrupt() r_Config_RSPI0_communication_end_interrupt()
volatile uint32_t	g_w24_data	Stores 24-bit transmit data	r_Config_ICU_irq9_interrupt()
volatile uint32_t	g_r24_data	Stores 24-bit receive data	r_Config_ICU_irq9_interrupt()
volatile uint16_t	g_w16_data	Stores 16-bit transmit data	r_Config_ICU_irq9_interrupt() set_16bit_data_transfer_mode()
volatile uint16_t	g_r16_data[8]	Stores 16-bit receive data	r_Config_ICU_irq9_interrupt() set_16bit_data_receive_mode()

4.2.4.4 Functions added to the SC-generated code

Table 4.14 shows List of Functions Added to the SC-Generated Code.

Table 4.14 List of Functions Added to the SC-Generated Code

Type	Variable Name	Argument	Description
void	set_16bit_data_transfer_mode	void	Sets DTC transfer information for 16-bit transmission as shown in Table 4.7 DTC Transfer C: DTC Settings for Data Transmission (Transfer Data Length: 16 Bits)
void	set_16bit_data_receive_mode	void	Sets DTC transfer information for 16-bit reception as shown in Table 4.8 DTC Transfer D: DTC Settings for Data Reception (Transfer Data Length: 16 Bits).

4.2.4.5 Adding code to the main routine

Add code to the main() function of the main routine in the data_length_change_sample_for_rspi_dtc.c file.

Figure 4.3 shows Outline Flow of the main() Function.

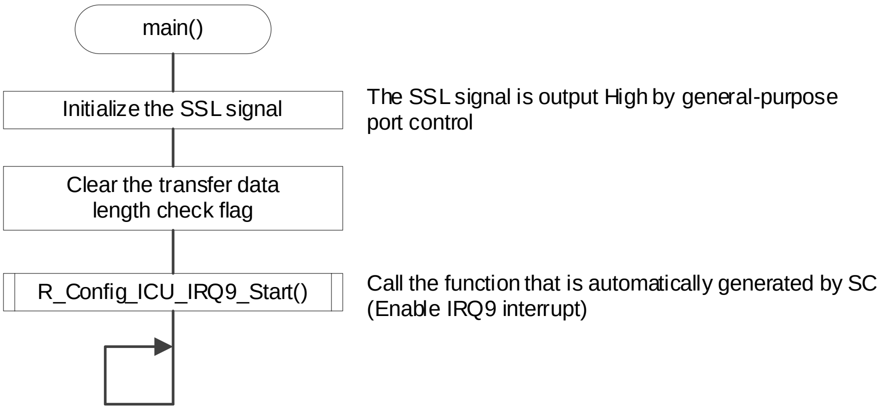


Figure 4.3 Outline Flow of the main() Function

Add code to the Config_RSPI0.h file.

Adding code to the Config_RSPI0.h file

```
/* Start user code for function. Do not edit comment generated here */
/***********************************************************************************************************
Macro definitions
***********************************************************************************************************/
#define SSL_PORT_PODR_BIT  (PORTA.PODR.BIT.B2)    ← General-purpose port PA2 controls the
#define SSL_PORT_PDR_BIT   (PORTA.PDR.BIT.B2)       SSL signal.
/* End user code. Do not edit comment generated here */
```

Add code to the main routine.

Adding code to the main() function

```
#include "r_smc_entry.h"              ← Includes the header files automatically created by SC.

volatile uint8_t g_length_check;      ← Definition of the transfer data length check flag

void main(void);

void main(void)
{
   /* Set General-purpose port for SSL control */
   SSL_PORT_PODR_BIT = 1U;            ← The SSL signal is output High by general-purpose port control.
   SSL_PORT_PDR_BIT = 1U;

   g_length_check = 0U;               ← Clears the transfer data length check flag.
   R_Config_ICU_IRQ9_Start();         ← Enables IRQ9.

   while(1U){
    /* do nothing */
   }
}
```

4.2.4.6 Adding code to the IRQ9 interrupt handler

Add code to the r_Config_ICU_irq9_interrupt() function, which is the IRQ9 interrupt handler.

Figure 4.4 shows Outline Flow of the r_Config_ICU_irq9_interrupt() Function.

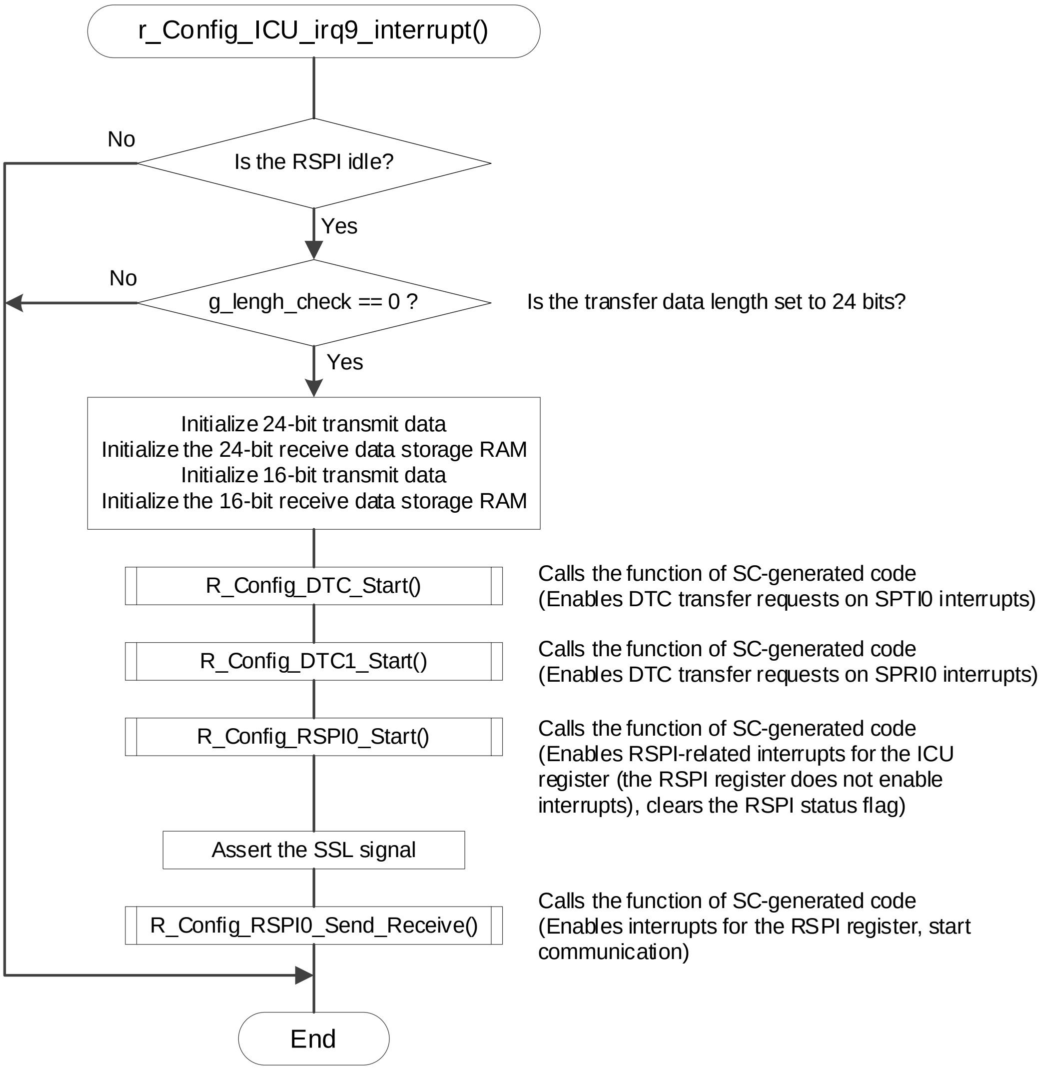


Figure 4.4 Outline Flow of the r_Config_ICU_irq9_interrupt() Function

Add code to the “Includes” and “Global variables and functions” sections in the Config_ICU_user.c file.

Adding code to the “Includes” and “Global variables and functions” sections in the Config_ICU_user.c file

```
/***********************************************************************************************************************
Includes
***********************************************************************************************************************/
#include "r_cg_macrodriver.h"
#include "Config_ICU.h"
/* Start user code for include. Do not edit comment generated here */
#include "r_smc_entry.h"          ← Includes the header files automatically created by SC.
/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"

/***********************************************************************************************************************
Global variables and functions
***********************************************************************************************************************/
/* Start user code for global. Do not edit comment generated here */
extern volatile uint8_t g_length_check;          ← extern statement of the transfer data length check
                                                   flag

#pragma address (g_w24_data=0x03000U)           ← Defines the addresses according to the source
#pragma address (g_r24_data=0x02000U)             address and destination address specified in
                                                   4.2.2.2 and 4.2.2.3.

volatile uint32_t  g_w24_data;
volatile uint32_t  g_r24_data;                    ← Defines the variables for 24-bit transmit, 24-bit
volatile uint16_t  g_w16_data;                      receive data storage RAM, 16-bit transmit data,
volatile uint16_t  g_r16_data[8];                   and 16-bit receive data storage RAM.

/* End user code. Do not edit comment generated here */
```

Add code to the r_Config_ICU_irq9_interrupt() function.

Adding code to the r_Config_ICU_irq9_interrupt () function

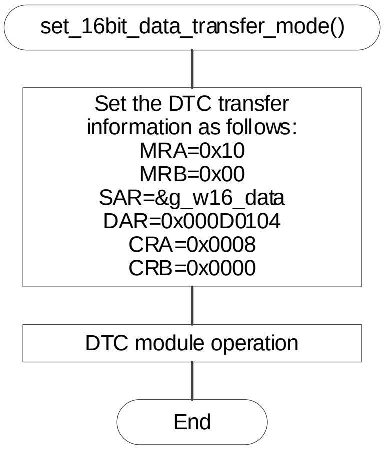
```
static void r_Config_ICU_irq9_interrupt(void)
{
/* Start user code for r_Config_ICU_irq9_interrupt. Do not edit comment generated here */
   uint32_t i;

   if (0U != RSPI0.SPSR.BIT.IDLNF)           ← Checks whether the RSPI is idle.
   {
     /* do nothing */
   }
   else
   {
     If (0U == g_length_check)        ← Checks whether the initial setting of the transfer data length is
                                         24 bits.
     {
        g_w24_data = 0x123456U;                      ← Initializes the RAM areas that store 24-bit
        g_r24_data = 0xFFFFFFFFU;                       transmit data, 24-bit receive data, 16-bit
                                                        transmit data, and 16-bit receive data.
        g_w16_data = 0x789AU;

        for (i=0U; i<8U; i++)
        {
             g_r16_data[i] = 0xFFFFU;
        }

        R_Config_DTC_Start();                ← Enables DTC transfer requests on SPTI0 interrupts or
        R_Config_DTC1_Start();                  SPRI0 interrupts.
        R_Config_RSPI0_Start();                 Enables RSPI-related interrupts for the ICU register.
                                                (Interrupts for the RSPI register are not enabled.)
                                                Clears the RSPI status flag.

        /* SSL assert */
        SSL_PORT_PODR_BIT = 0U;              ← Asserts the SSL signal.

        R_Config_RSPI0_Send_Receive(NULL, 24U, NULL);    ← Enables interrupts for the RSPI
     }                                                      register, start communication
   }
   /* End user code. Do not edit comment generated here */
}
```

4.2.4.7 Adding the set_16bit_data_transfer_mode() function to the Config_DTC.c file

The set_16bit_data_transfer_mode() function has been added to the Config_DTC.c file.

This function sets DTC transfer information for 16-bit transmission as shown in Table 4.7 DTC Transfer C: DTC Settings for Data Transmission (Transfer Data Length: 16 Bits).

Figure 4.5 shows Outline Flow of the set_16bit_data_transfer_mode() Function.



For details about the settings, see Table 4.7 DTC Transfer C: DTC Settings for Data Transmission (Transfer Data Length: 16 Bits).

Figure 4.5 Outline Flow of the set_16bit_data_transfer_mode() Function

Add the prototype declaration and constant definition of the set_16bit_data_transfer_mode() function to the Config_DTC.h file.

Adding code to the Config_DTC.h file

```
/* Start user code for function. Do not edit comment generated here */
void set_16bit_data_transfer_mode(void);          ← Prototype declaration
#define CRA_16BIT_TRANSFER       (0x0008U)        ← Value of the CRA register for 16-bit transmission
/* End user code. Do not edit comment generated here */
```

Add the extern statement of the transmit data storage variable for 16-bit communication to the “Global variables and functions” section in the Config_DTC.c file.

Adding code to the “Global variables and functions” section in the Config_DTC.c file

```
/*******************************************************************************************************************
Global variables and functions
*******************************************************************************************************************/
#pragma address dtc_vector39=0x0001FC9CUL
volatile uint32_t dtc_vector39;
volatile st_dtc_data_t dtc_transferdata_vector39;
/* Start user code for global. Do not edit comment generated here */
extern volatile uint16_t g_w16_data;          ← extern statement of the transmit data storage
                                                variable for 16-bit communication
/* End user code. Do not edit comment generated here */
```

The following shows the set_16bit_data_transfer_mode() function added to the Config_DTC.c file.

set_16bit_data_transfer_mode() function added to the Config_DTC.c file

```
/* Start user code for adding. Do not edit comment generated here */
/*******************************************************************************************************************
* Function Name: set_16bit_data_transfer_mode
* Description  : This function initializes the DTC module for 16bit transmission.
* Arguments    : None
* Return Value : None
*******************************************************************************************************************/
void set_16bit_data_transfer_mode(void)
{
  /* Set DTC transfer data */
  dtc_transferdata_vector39.mra_mrb =((uint32_t)(_00_DTC_WRITE_BACK_ENABLE |
                                                 _00_DTC_SRC_ADDRESS_FIXED |
                                                 _10_DTC_TRANSFER_SIZE_16BIT |
                                                 _00_DTC_TRANSFER_MODE_NORMAL)<<24U) |
                                                 ((uint32_t)(_00_DTC_DST_ADDRESS_FIXED |
                                                 _00_DTC_INTERRUPT_COMPLETED)<<16U);
  dtc_transferdata_vector39.sar = (uint32_t) &g_w16_data;
  dtc_transferdata_vector39.dar = _000D0104_DTC0_DST_ADDRESS;
  dtc_transferdata_vector39.cra_crb = (uint32_t)(CRA_16BIT_TRANSFER) << 16U;

  /* Enable DTC module start */
    DTC.DTCST.BYTE = _01_DTC_MODULE_START;
}
/* End user code. Do not edit comment generated here */
```

4.2.4.8 Adding the set_16bit_data_receive_mode() function to the Config_DTC1.c file

The set_16bit_data_receive_mode() function has been added to the Config_DTC1.c file.

This function sets DTC transfer information for 16-bit reception to the settings in Table 4.8 DTC Transfer D: DTC Settings for Data Reception (Transfer Data Length: 16 Bits).

Figure 4.6 shows Outline Flow of the set_16bit_data_receive_mode() Function.

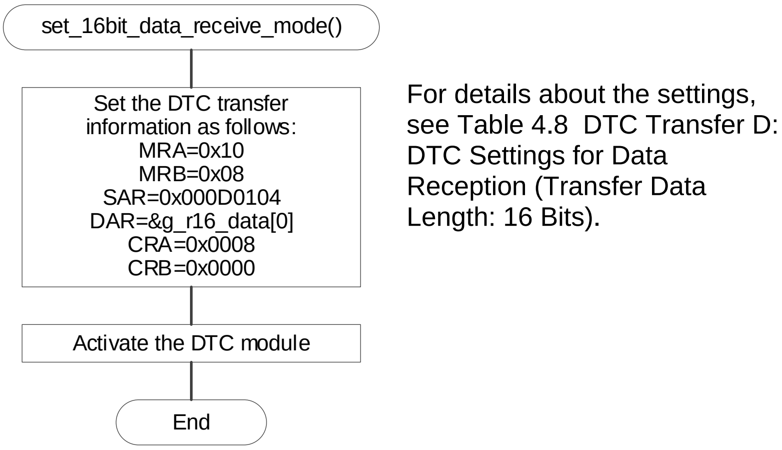


Figure 4.6 Outline Flow of the set_16bit_data_receive_mode() Function

Add the prototype declaration and constant definition of the set_16bit_data_receive_mode() function to the Config_DTC1.h file.

Adding code to the Config_DTC1.h file

```
/* Start user code for function. Do not edit comment generated here */
void set_16bit_data_receive_mode(void);          ← Prototype declaration
#define CRA_16BIT_RECEIVE      (0x0008U)         ← Value of the CRA register for 16-bit reception
/* End user code. Do not edit comment generated here */
```

Add the extern statement of the receive data storage variable for 16-bit communication to "Global variables and functions" in the Config_DTC1.c file.

Adding code to "Global variables and functions" in the Config_DTC1.c file

```
/***********************************************************************************************************************
Global variables and functions
***********************************************************************************************************************/
#pragma address dtc_vector38=0x0001FC98UL
volatile uint32_t dtc_vector38;
volatile st_dtc_data_t dtc_transferdata_vector38;
/* Start user code for global. Do not edit comment generated here */
extern volatile uint16_t g_r16_data[8];          ← extern statement of the receive data storage
                                                   variable for 16-bit communication
/* End user code. Do not edit comment generated here */
```

The following shows how the set_16bit_data_receive_mode() function is added to the Config_DTC1.c file.

set_16bit_data_receive_mode() function added to the Config_DTC1.c file

```
/* Start user code for adding. Do not edit comment generated here */
/***********************************************************************************************************************
* Function Name: set_16bit_data_receive_mode
* Description  : This function initializes the DTC module for 16bit reception.
* Arguments    : None
* Return Value : None
***********************************************************************************************************************/
void set_16bit_data_receive_mode(void)
{
     /* Set DTC transfer data */
  dtc_transferdata_vector38.mra_mrb = ((uint32_t)(_00_DTC_WRITE_BACK_ENABLE |
                                                   _00_DTC_SRC_ADDRESS_FIXED |
                                                   _10_DTC_TRANSFER_SIZE_16BIT |
                                                   _00_DTC_TRANSFER_MODE_NORMAL)<<24U) |
                                                   ((uint32_t)(_08_DTC_DST_ADDRESS_INCREMENTED |
                                                   _00_DTC_REPEAT_DST_SIDE |
                                                   _00_DTC_INTERRUPT_COMPLETED)<<16U);
  dtc_transferdata_vector38.sar = _000D0104_DTC0_SRC_ADDRESS;
  dtc_transferdata_vector38.dar = (uint32_t) &g_r16_data[0];
  dtc_transferdata_vector38.cra_crb = (uint32_t)(CRA_16BIT_RECEIVE) << 16U;

  /* Enable DTC module start */
    DTC.DTCST.BYTE = _01_DTC_MODULE_START;
}
/* End user code. Do not edit comment generated here */
```

4.2.4.9 Adding code to the SPTI0 interrupt handler

Code has been added to the SPTI0 interrupt callback function (r_Config_RSPI0_callback_transmitend()).

Figure 4.7 shows Outline Flow of the r_Config_RSPI0_callback_transmitend() Function.

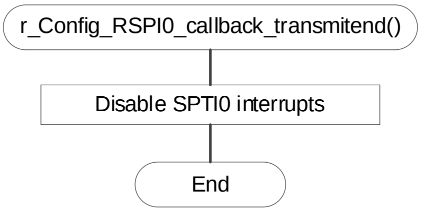


Figure 4.7 Outline Flow of the r_Config_RSPI0_callback_transmitend() Function

Code has been added to the r_Config_RSPI0_callback_transmitend() function of the Config_RSPI0_user.c file.

Code added to the r_Config_RSPI0_callback_transmitend() function

```
/* Start user code for r_Config_RSPI0_callback_transmitend. Do not edit comment generated here */
/* Disable Transmit buffer empty interrupt */
RSPI0.SPCR.BIT.SPTIE = 0U;                    ← Disables SPTI0 interrupts.
/* End user code. Do not edit comment generated here */
```

4.2.4.10 Adding code to the SPRI0 interrupt handler

Code has been added to the SPRI0 interrupt callback function (r_Config_RSPI0_callback_receiveend()).

Figure 4.8 shows Outline Flow of the r_Config_RSPI0_callback_reveiveend() Function.

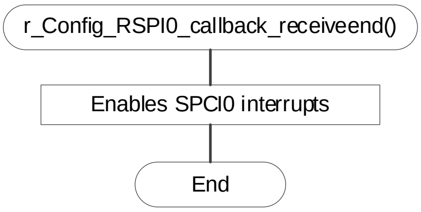


Figure 4.8 Outline Flow of the r_Config_RSPI0_callback_reveiveend() Function

Code has been added to the r_Config_RSPI0_callback_receiveend() function of the Config_RSPI0_user.c file.

Code added to the r_Config_RSPI0_callback_receiveend() function

```
/* Start user code for r_Config_RSPI0_callback_transmitend. Do not edit comment generated here */
/* Enable communication end interrupt */
RSPI0.SPCR3.BIT.SPCIE = 1U;                    ← Enables SPCI0 interrupts.
/* End user code. Do not edit comment generated here */
```

4.2.4.11 Adding code to the SPCI0 interrupt handler

Code has been added to the SPCI0 interrupt callback function (r_Config_RSPI0_communication_end_interrupt()).

Figure 4.9 shows Outline Flow of the r_Config_RSPI0_communication_end_interrupt() Function.

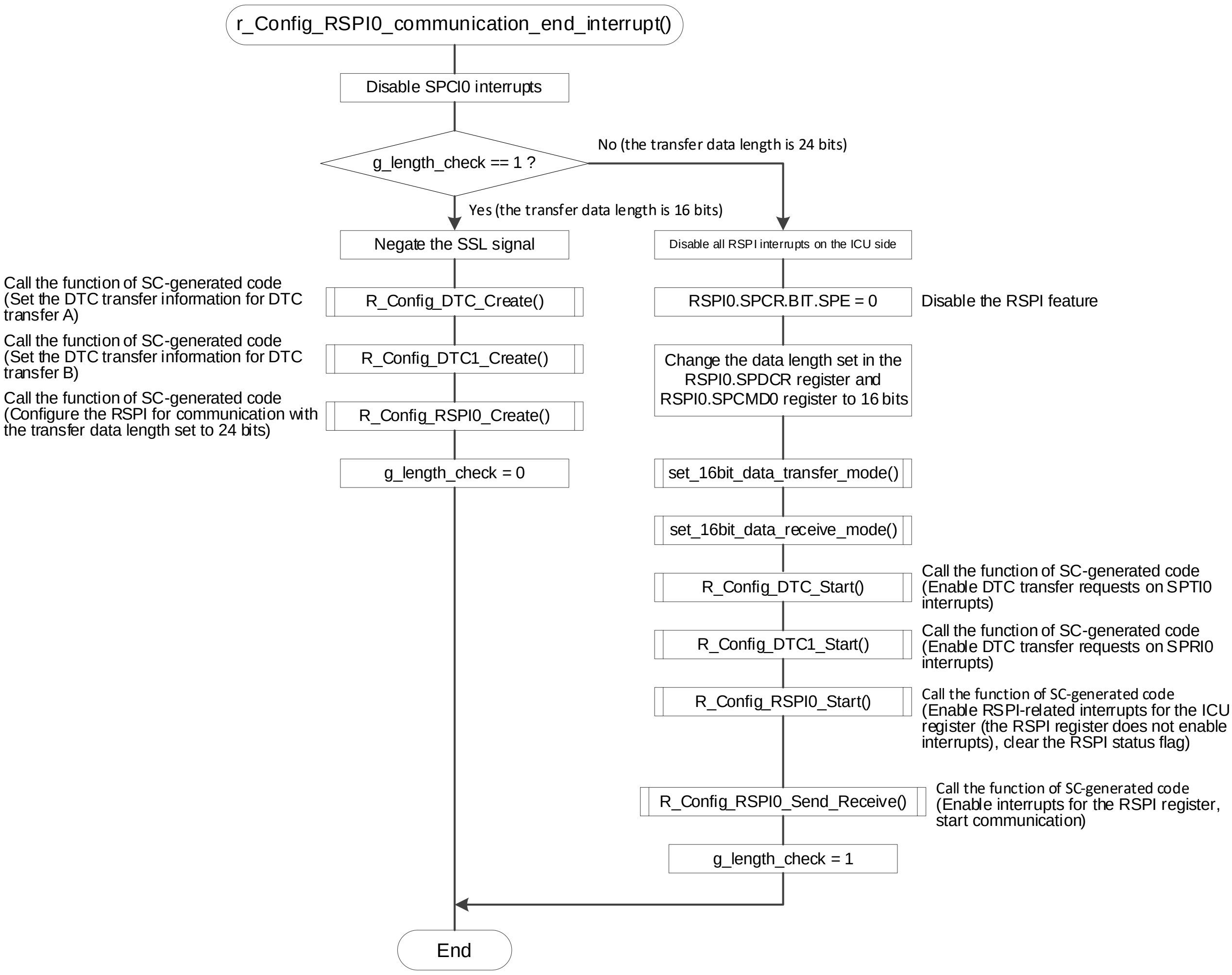


Figure 4.9 Outline Flow of the r_Config_RSPI0_communication_end_interrupt() Function

Add the definition statement to the “Includes” and “Global variables and functions” sections in the Config_RSPI0_user.c file.

Adding code to the “Includes” and “Global variables and functions” sections in the Config_RSPI0_user.c file

```
/***********************************************************************************************************************
Includes
***********************************************************************************************************************/
#include "r_cg_macrodriver.h"
#include "Config_RSPI0.h"
/* Start user code for include. Do not edit comment generated here */
#include "r_smc_entry.h"                ← Includes the header files automatically created by SC
/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"

/***********************************************************************************************************************
Global variables and functions
***********************************************************************************************************************/
/* Start user code for global. Do not edit comment generated here */
extern volatile uint8_t g_length_check;        ← extern statement of the transfer data length check flag
/* End user code. Do not edit comment generated here */
```

Code has been added to the r_Config_RSPI0_communication_end_interrupt() function of the Config_RSPI0_user.c file.

Code added to the r_Config_RSPI0_communication_end_interrupt() function

```
/* Start user code for r_Config_RSPI0_communication_end_interrupt. Do not edit comment generated here
*/
   /* Disable communication end interrupt */
   RSPI0.SPCR3.BIT.SPCIE = 0U;
   /* Processing when 16bit x 8frames is received */
      If (1U == g_length_check)
      {
         /*SSL negate */
         SSL_PORT_PODR_BIT = 1U;

         /* Return DTC and RSPI settings to default settings (for 24-bit settings) */
         R_Config_DTC_Create();
         R_Config_DTC1_Create();
         R_Config_RSPI0_Create();

         g_length_check = 0U;
      }
      else
      {
         /* Processing when 24bit x 1frame is received */
         /* Disable RSPI interrupts */
         IEN(RSPI0,SPTI0) = 0U;
         IEN(RSPI0,SPRI0) = 0U;
         EN(RSPI0,SPEI0) = 0U;
         EN(RSPI0,SPII0) = 0U;
         IEN(RSPI0, SPCI0) = 0U;

         /* Disable RSPI function */
         RSPI0.SPCR.BIT.SPE = 0U;

         /* Change RSPI settings for 16bit length */
         RSPI0.SPDCR.BIT.SPLW = 0U;
         RSPI0.SPDCR.BIT.SPBYT = 0U;
         RSPI0.SPDCR.BIT.SPFC = 0x00U;
         RSPI0.SPCMD0.WORD = _0001_RSPI_RSPCK_SAMPLING_EVEN |
                             _0000_RSPI_RSPCK_POLARITY_LOW |
                             _000C_RSPI_BASE_BITRATE_8 |
                             _0000_RSPI_SIGNAL_ASSERT_SSL0 |
                             _0080_RSPI_SSL_KEEP_ENABLE |
                             _0F00_RSPI_DATA_LENGTH_BITS_16 |
                             _0000_RSPI_MSB_FIRST |
                             _0000_RSPI_NEXT_ACCESS_DELAY_DISABLE |
                             _0000_RSPI_NEGATION_DELAY_DISABLE |
                             _0000_RSPI_RSPCK_DELAY_DISABLE;
```

← This processing negates the SSL signal and resets the transfer data length to 24 bits after completion of 16-bit communication.

This processing starts communication with the transfer data length changed to 16 bits after completion of 24-bit communication. ↓

```
        /* Change DTC settings for 16bit x 8frames */
        set_16bit_data_transfer_mode();
        set_16bit_data_receive_mode();

    /* DTC, RSPI start */
    R_Config_DTC_Start();
    R_Config_DTC1_Start();
    R_Config_RSPI0_Start();
    R_Config_RSPI0_Send_Receive(NULL, 16U, NULL);

        g_length_check = 1U;
}
/* End user code. Do not edit comment generated here */
```

5. Importing a Project

The sample programs are distributed in e^2 studio project format. This section shows how to import a project into e^2 studio or CS+. After importing the sample project, make sure to confirm build and debugger setting.

5.1 Importing a Project into e^2 studio

To use sample programs in e^2 studio, follow the steps below to import them into e^2 studio.

In projects managed by e^2 studio, do not use space codes, multibyte characters, and symbols such as "$", "#", "%" in folder names or paths to them.

(Note that depending on the version of e^2 studio you are using, the interface may appear somewhat different from the screenshots below.)

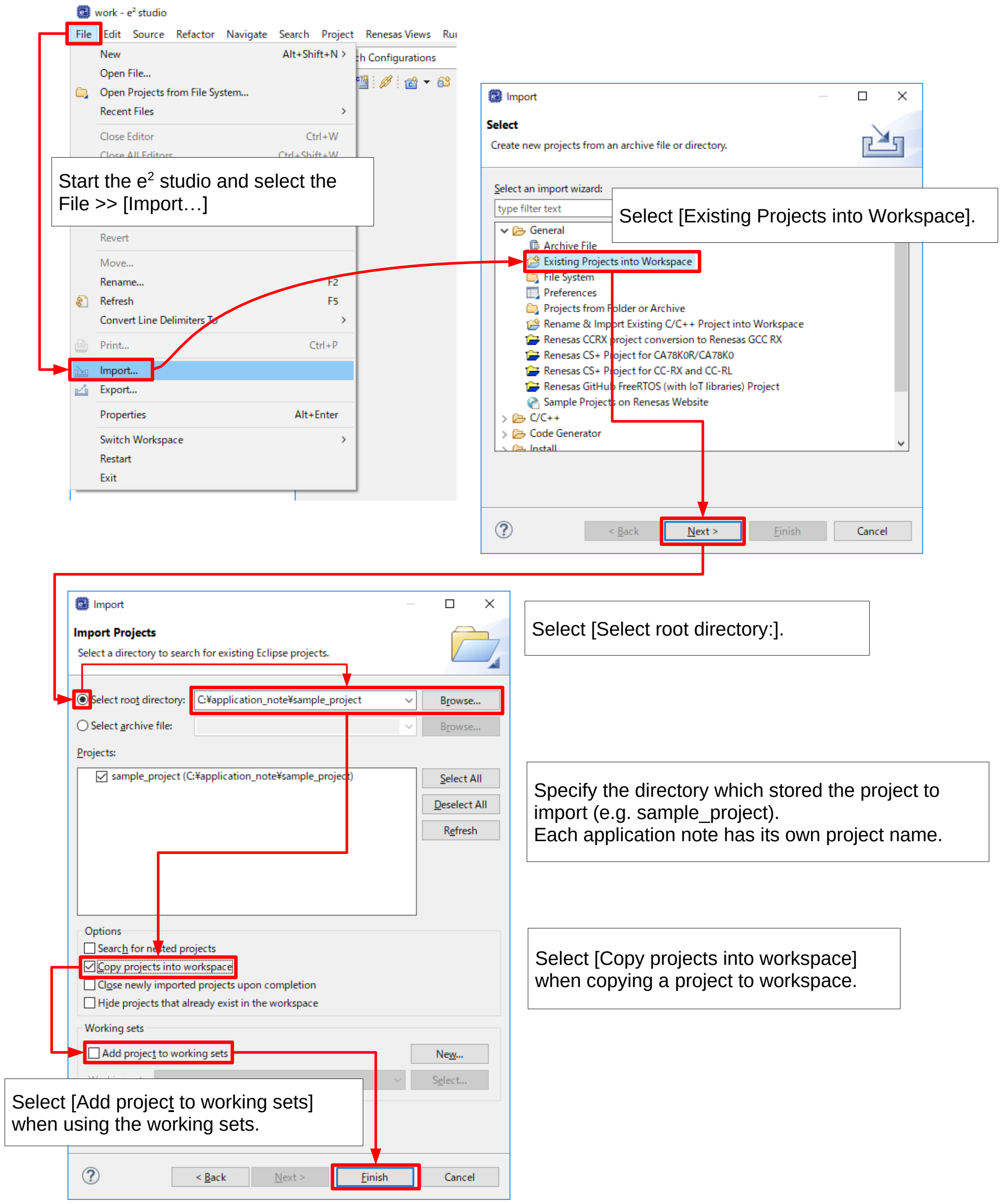


Figure 5.1 Importing a Project into e^2 studio

5.2 Importing a Project into CS+

To use sample programs in CS+, follow the steps below to import them into CS+.

In projects managed by CS+, do not use space codes, multibyte characters, and symbols such as "$", "#", "%" in folder names or paths to them.

(Note that depending on the version of CS+ you are using, the interface may appear somewhat different from the screenshots below.)

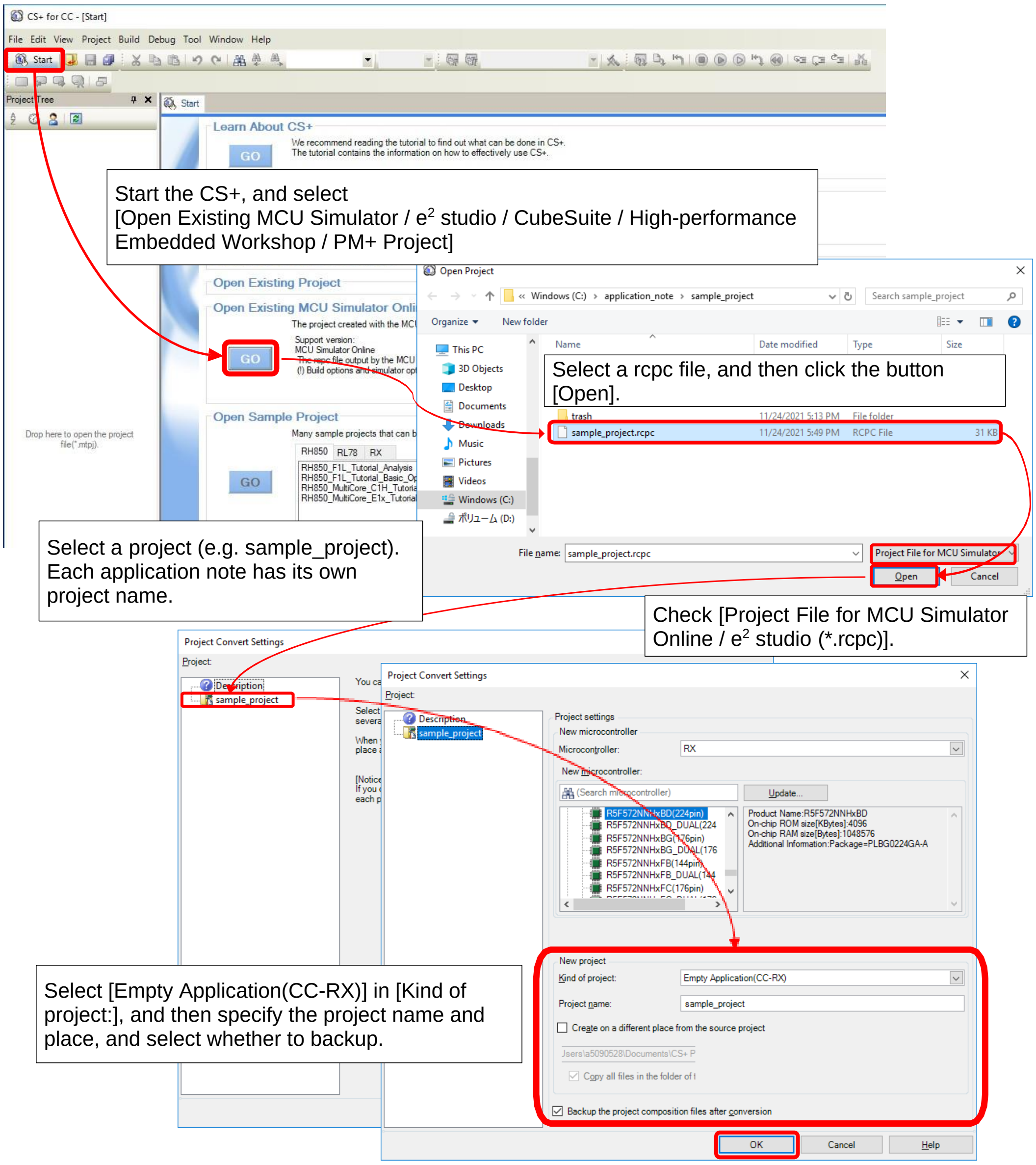


Figure 5.2 Importing a Project into CS+

6. Notes

6.1 Notes on Bit Manipulation Instructions

If the target of bit manipulation is an 8-bit I/O register, depending on the code of the C language, bit manipulation instructions that involve memory access may not be output.

Refer to "C language code example" shown below. In this example, a variable is used on the right side of the expression for bit manipulation on a general-purpose port. If this code is compiled, it may be expanded to the instructions shown in "Instruction expansion example".

In this case, if there is an interrupt that changes the settings of other bits of the same I/O register and the interrupt occurs around the time at which A3 occurs, the changes made by the interrupt are not applied.

C language code example:

```
unsigned char i;
i=1;
PORTD.PODR.BIT.B6 = i;
```

Instruction expansion example:

```
A1 : mov.l      #0x8c02d, r14
A2 : mov.b      [r14], r15
A3 : bset       #6, r15
A4 : mov.b      r15, [r14]
```

A possible solution is to use an immediate value (instead of a variable) on the right side of the expression as shown below. This solution allows you to output bit manipulation instructions that involve memory access (with CC-RX V2.06 or later).

```
if( 0 == i)
{
  PORTD.PODR.BIT.B6 = 0;
}
else
{
  PORTD.PODR.BIT.B6 = 1;
}
```

This problem may also be prevented by using intrinsic functions provided by the CC-RX compiler.

For details, refer to the "CC-RX Compiler User's Manual" (R20UT3248).

7. Reference Documents

User's Manual: Hardware

RX660 Group User's Manual: Hardware (R01UH0937)

(The latest version can be downloaded from the Renesas Electronics website.)

Application Note

RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)

(The latest version can be downloaded from the Renesas Electronics website.)

User's Guide: Smart Configurator

Smart Configurator User's Guide: e^2 studio (R20AN0451)

(The latest version can be downloaded from the Renesas Electronics website.)

User's Manual: Compiler

CC-RX Compiler User's Manual (R20UT3248)

(The latest version can be downloaded from the Renesas Electronics website.)

User's Manual: RSK

Renesas Starter Kit for RX660 User's Manual (R20UT5017)

(The latest version can be downloaded from the Renesas Electronics website.)

Schematic Diagram: RSK

Renesas Starter Kit for RX660 CPU Board Schematics (R20UT5016)

(The latest version can be downloaded from the Renesas Electronics website.)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jan. 22, 2024	—	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

Corporate Headquarters

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

Trademarks

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/.