Renesas e2 studio

R20AN0480EJ0100
Rev.1.00
Nov 01, 2017

Porting projects produced with the Code Generator to projects for use with the Smart Configurator

Introduction

This application note describes how to port projects produced with the Code Generator to projects for use with the Smart Configurator.

Target Device

- RX64M Group

If you are applying the information in this application note to another MCU, do so in a way that suits the given MCU and evaluate the results.

Reference Documents

Renesas e^2 studio Smart Configurator User Guide (R20AN0451)

e^2 studio Integrated Development Environment User's Manual: Getting Started Guide (R20UT2771)

RSK+RX64M Code Generator Tutorial Manual (R20UT2930)

RX64M Renesas Starter Kit Sample Code for CubeSuite+ (R01AN2219)

Contents

1. Overview

1.1 Purpose of This Document

Using sample source code, this application note concretely describes how to port projects produced with the Code Generator to projects for use with the Smart Configurator in terms of the differences in methods of settings and in the names of functions that are generated.

For the usage of the e^2 studio, refer to the e^2 studio Integrated Development Environment User's Manual: Getting Started Guide.

1.2 Operating Environment

Table 1.1 Operating Environment

Target Device	RX64M Group
Emulator	E1
IDE	e^2 studio v.6.0.0 and later versions
Toolchain	Renesas C/C++ compiler package for RX family
Toolchain version	CC-RX V2.07.00

2. Porting Projects Produced with the Code Generator to Projects for Use with the Smart Configurator

Figure 2.1 shows the steps in porting projects produced with the Code Generator to projects for use with the Smart Configurator.

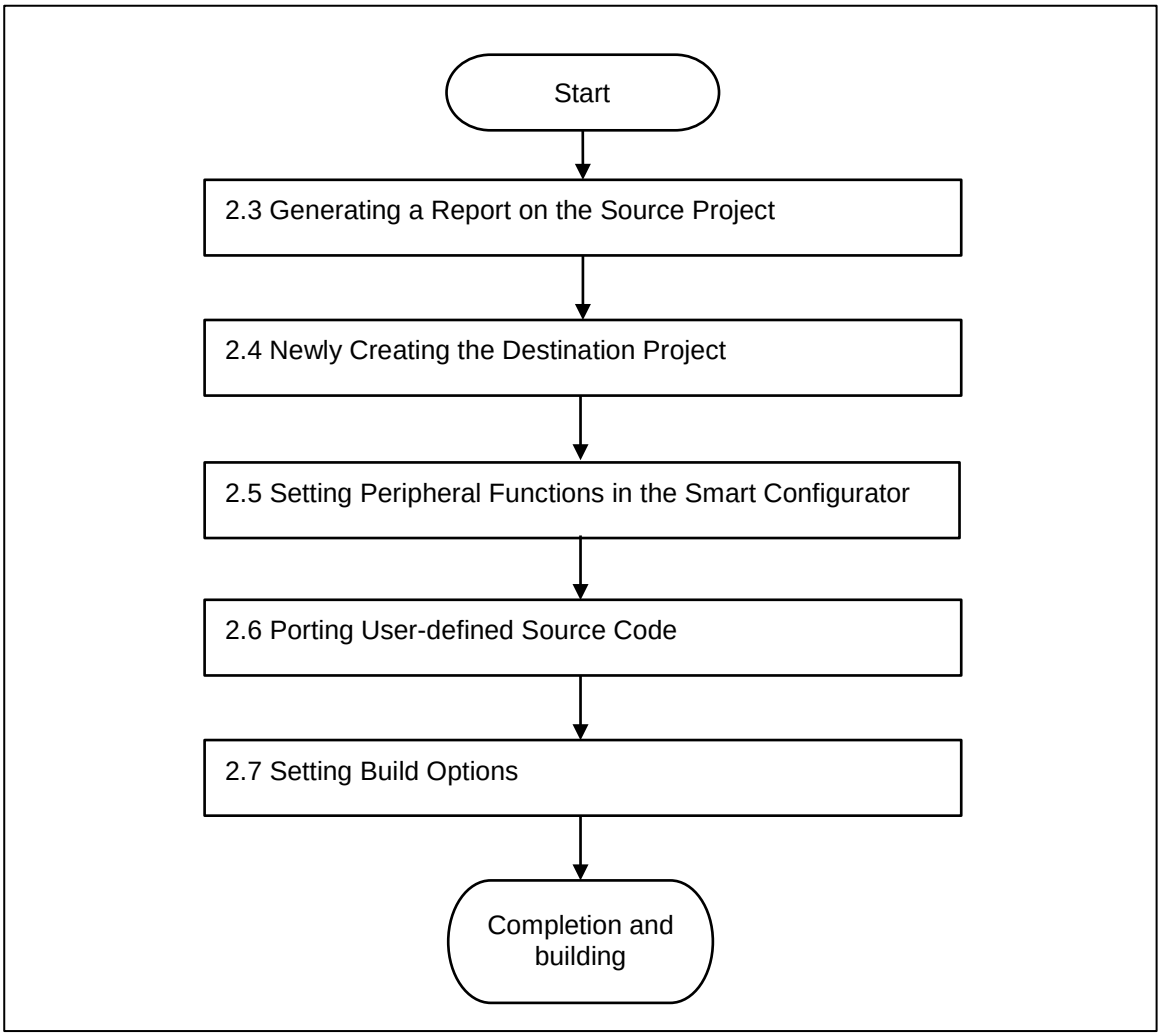


Figure 2.1 Steps in Porting Projects Produced with the Code Generator to Projects for Use with the Smart Configurator

2.1 Projects Used in This Application Note

The following two projects are used in this application note.

A project for the RSK+RX64M, which is a tool for evaluating Renesas MCUs, is used as the source project. The destination project is newly created.

Table 2.1 Projects Used in This Application Note

Project Name	Description
RSK+RX64M_Tutorial	A project for the RSK+RX64M produced with the use of the Code Generator serves as the source project. This project is used to generate a report to provide guidance on the setting of peripheral functions and the copying of user-created source code.
RSK_RX64M_Tutorial_SC	A destination project which is newly created for use with the Smart Configurator. In this project, the settings of peripheral functions and user-created source code in the source project are modified and reflected in the Smart Configurator according to the steps in Figure 2.1.

2.2 Downloading the Source Project

You can download the RSK+RX64M project, which is used as the source project in this application note, from the Web site of Renesas Electronics.

Note: To download the project, you need to register a My Renesas account.

(1) From the top page of the Web site of Renesas (https://www.renesas.com/ja-jp/), select [Boards and Kits] under the [Products] menu, then [See more] under [Renesas Starter Kits].

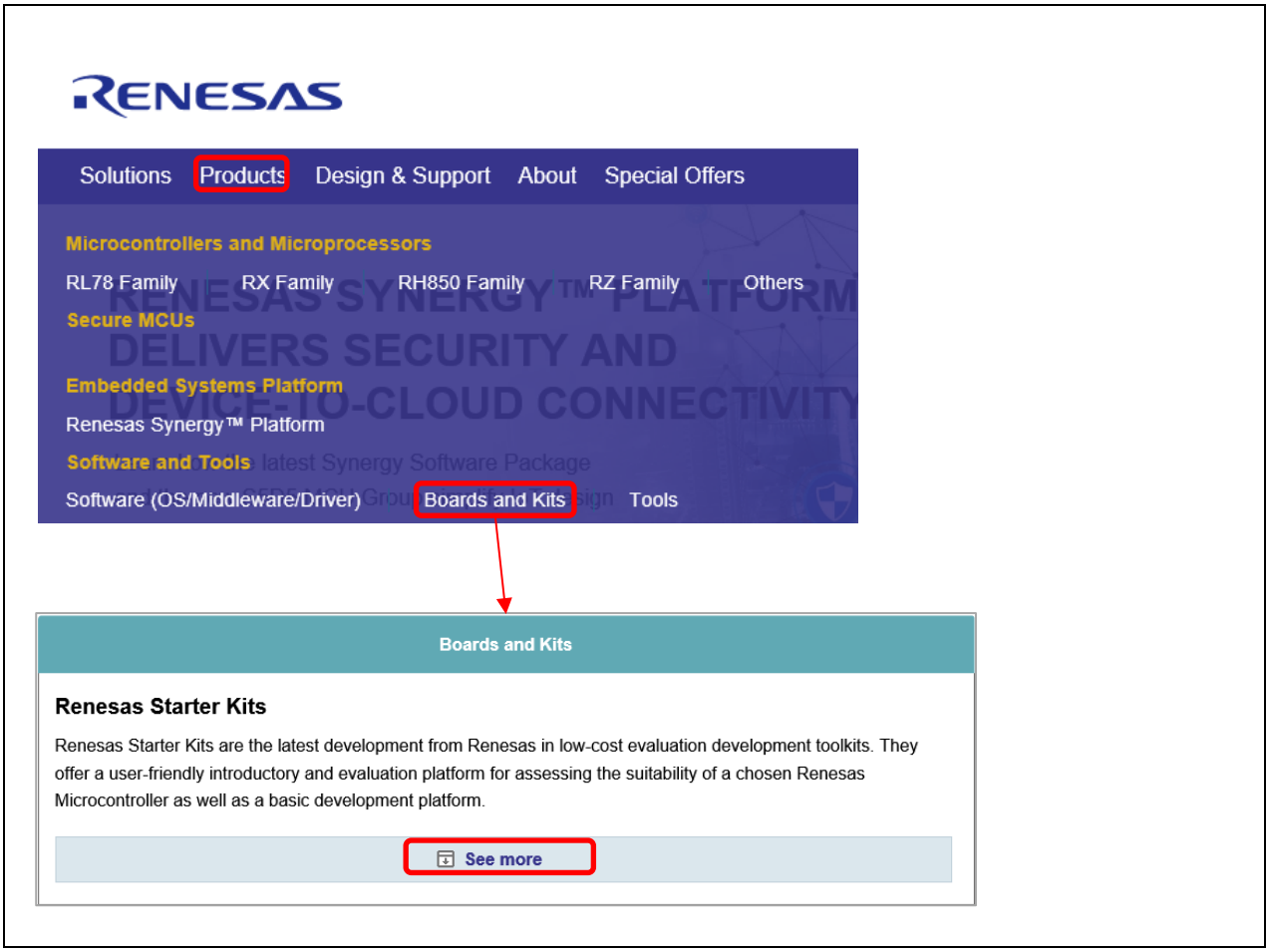


Figure 2.2 Downloading the Source Project (1)

(2) Select [Renesas Starter Kit+ for RX64M] from the list of Renesas Starter Kits.

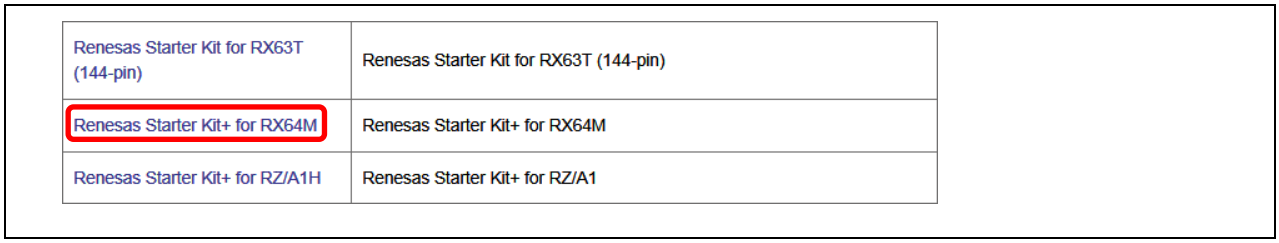


Figure 2.3 Downloading the Source Project (2)

(3) Select [RX64M Renesas Starter Kit Sample Code for CubeSuite+] in the list under [Product Name] on the [Download] tabbed page, then click on the [Download] button at the bottom of the page to proceed with downloading.

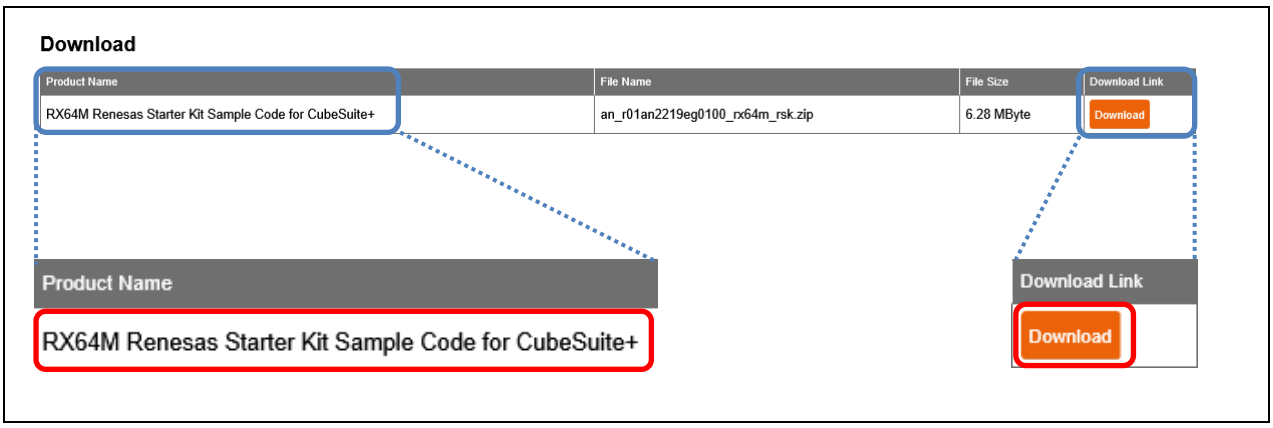


Figure 2.4 Downloading the Source Project (3)

2.3 Generating a Report on the Source Project

Use the function for generating reports from the Code Generator to output a report on the source project in the form of a list of peripheral functions. Refer to this report to set peripheral functions in the Smart Configurator for the destination project.

2.3.1 Generating the Report

- From CS+

(1) Start CS+ and open the source project [RSK+RX64M_Tutorial] that uses the Code Generator. Expand [Code Generator] under [Project Tree] and double-click on [Peripheral Functions].

(2) Select [Save Code Generator Report] from the [File] menu to generate the report.

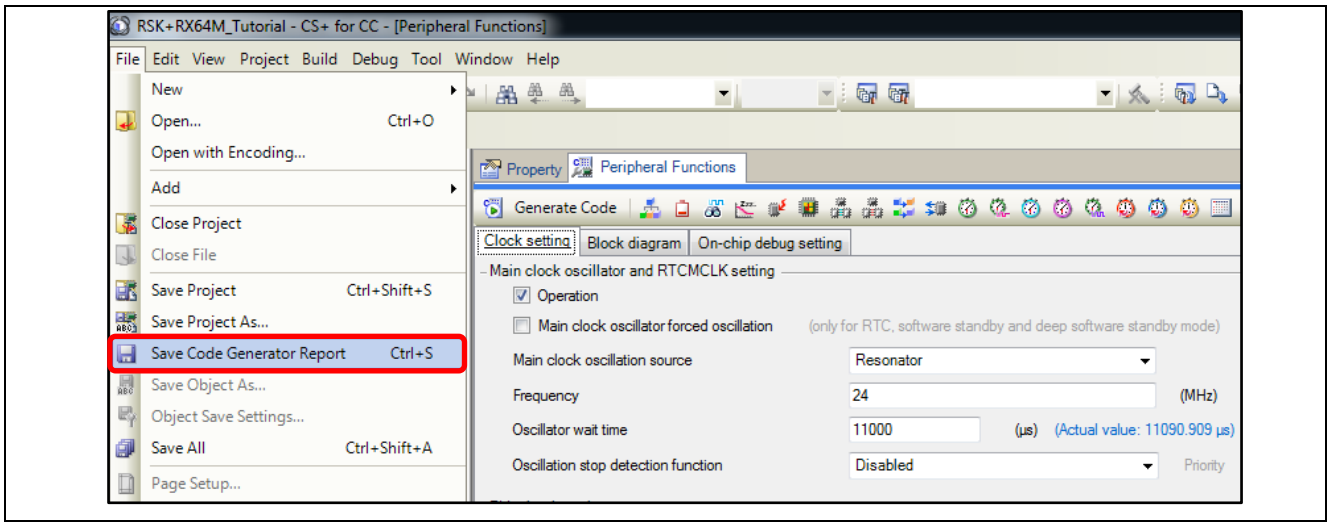


Figure 2.5 Generating the Report from CS+

(3) When the report has been generated, two files, Function.html and Macro.html, are output to the project folder.

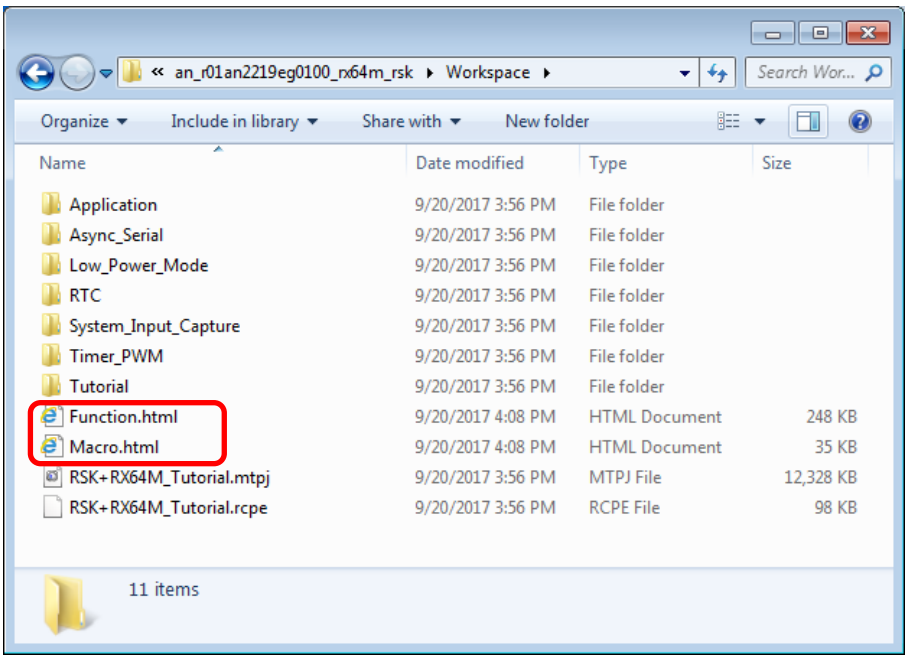


Figure 2.6 Report Files Output by the Report Function of the Code Generator for CS+

- From the e^2 studio

(1) Start the e^2 studio and open the source project [RSK+RX64M_Tutorial] for which the Code Generator was used. Expand [Code Generator] under [Project Tree] and double-click on [Peripheral Functions].

(2) Click on the [Generate Report] button to generate the report.

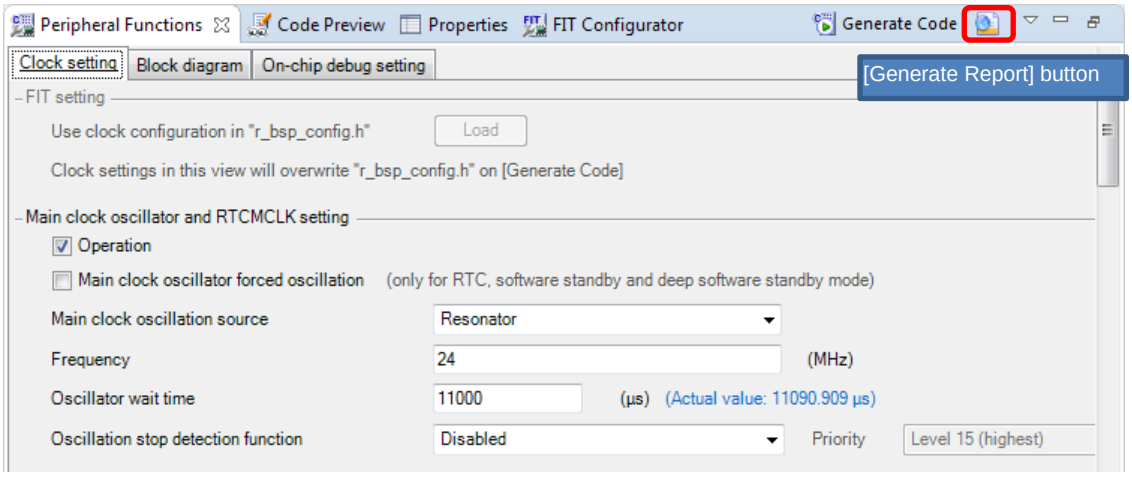


Figure 2.7 Generating a Report from the e^2 studio

(3) When the report has been generated, two files, Function.html and Macro.html, are output to the doc folder.

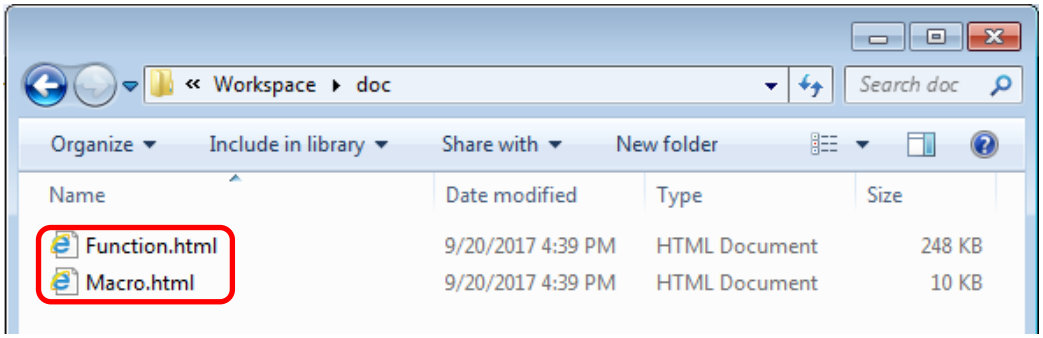


Figure 2.8 Report Files Output by the Report Function of the Code Generator for the e² studio

Table 2.2 Report Files Output by the Report Function of the Code Generator

File Name	Description
Function.html	A list of API functions generated by the Code Generator.
Macro.html	A list of peripheral functions set by the Code Generator.

2.4 Newly Creating the Destination Project

Newly create a C project as the destination project for use with the Smart Configurator. Regarding how to create a project, refer to section 2, Generating a Project, in the Renesas e^2 studio Smart Configurator User Guide.

2.5 Setting Peripheral Functions in the Smart Configurator

2.5.1 Correspondence between the Code Generator and the Smart Configurator

Table 2.3 shows the correspondence of the peripheral functions which are to be set in the RSK+RX64M project between those in the Code Generator and those in the Smart Configurator.

Table 2.3 Correspondence of Peripheral Functions between the Code Generator and the Smart Configurator (1)

Code Generator			**Smart Configurator**			
Peripheral functions	Setting items		Tabs	Peripheral functions	Setting items	
Interrupt Controller Unit	IRQ2 setting	—	Components	Interrupt Controller	IRQ2 setting	—
			Pins	Pin function	Interrupt controller unit	IRQ2
	IRQ5 setting	—	Components	Interrupt Controller	IRQ5 setting	—
			Pins	Pin function	Interrupt controller unit	IRQ5
	Group BL0 setting	—	Interrupts	GROUPBL0	—	—
Compare Match Timer	CMT0	—	Components	Compare Match Timer	CMT0	—
	CMT1	—			CMT1	—
	CMT2	—			CMT2	—
12-Bit A/D Converter	Single scan mode	Analog input channel setting	Components	Single Scan Mode S12AD	Basic setting	Analog input channel setting
		Conversion start trigger setting				Conversion start trigger setting
		ADTRGn# pin selection	Pins	Pin function	12-bit A/D converter	—

Table 2.4 Correspondence of Peripheral Functions between the Code Generator and the Smart Configurator (2)

Code Generator			**Smart Configurator**			
Peripheral functions	Setting items		Tabs	Peripheral functions	Setting items	
Serial Communications Interface	Simple SPI bus	Master transmit only	Components	SPI Clock Synchronous Mode	—	Master transmit only
		Transfer direction setting			—	Transfer direction setting
		Transfer rate setting			—	Transfer speed setting
		Pin setting	Pins	Pin function	Serial communications interface	—
	Asynchronous mode	Transmission /reception	Components	SCI/SCIF Asynchronous Mode	—	Transmission /Reception
		Start bit edge detection setting			—	Start bit edge detection setting
		Transfer rate setting			—	Transfer rate setting
I/O Port	Port0	P03	Components	ポート	PORT0	P03
		P05				P05
	Port2	P26			PORT2	P26
		P27				P27
	Port4	P45			PORT4	P45
		P46				P46
		P47				P47

Set the Smart Configurator with the project that has been created in section 2.4, Newly Creating the Destination Project, with reference to the report that was output in section 2.3, Generating a Report on the Source Project.

This section describes settings of the clock generator, compare-match timer, and serial communications interface. Set other peripheral functions according to the same procedure.

2.5.2 Setting the Clock Generator

Set the clock generator.

(1) Open the Macro.html file of the report that was output in section 2.3, Generating a Report on the Source Project, and display the parts to be set for the clock generator.

MCU name: RX64M_4MB(4MB)
Chip name: R5F564MLCxFC

Peripheral function	Macro	SubMacro	Setting	Status
Clock Generator				Used
	CGC			Used
			Main clock oscillator forced oscillation	Unused
			Main clock oscillation source	Resonator
			Main clock oscillation source Frequency	24(MHz)
			Oscillator wait time	11000(μs),(Actual value: 11090.909 μs)
			Oscillation stop detection function	Disabled
			PLL Operation	Used
			PLL clock source	Main clock oscillator
			Input frequency division ratio	x 1
			Frequency multiplication factor	x 10.0
			Frequency	240 (MHz)
			SubCLK Operation	Unused

Figure 2.9 Report on the Clock Generator Output by the Code Generator

(2) Open the window for setting the Smart Configurator for the project that was created in section 2.4, Newly Creating the Destination Project, and select the [Clocks] tabbed page.

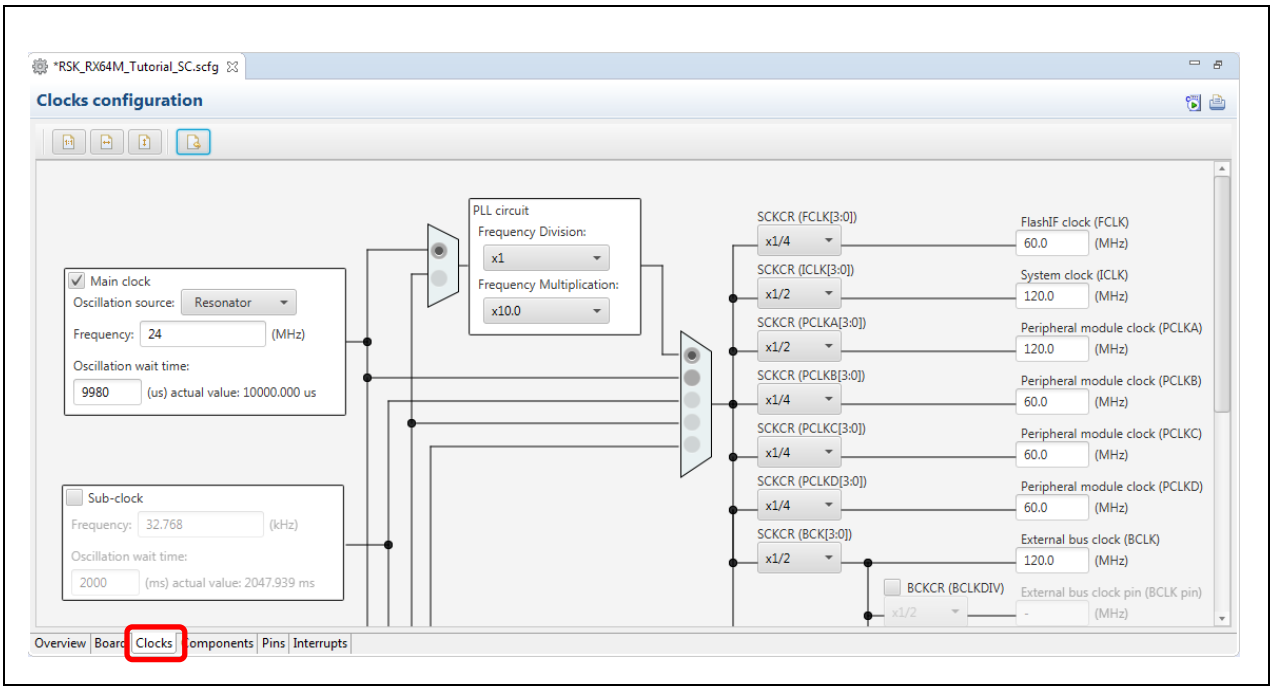
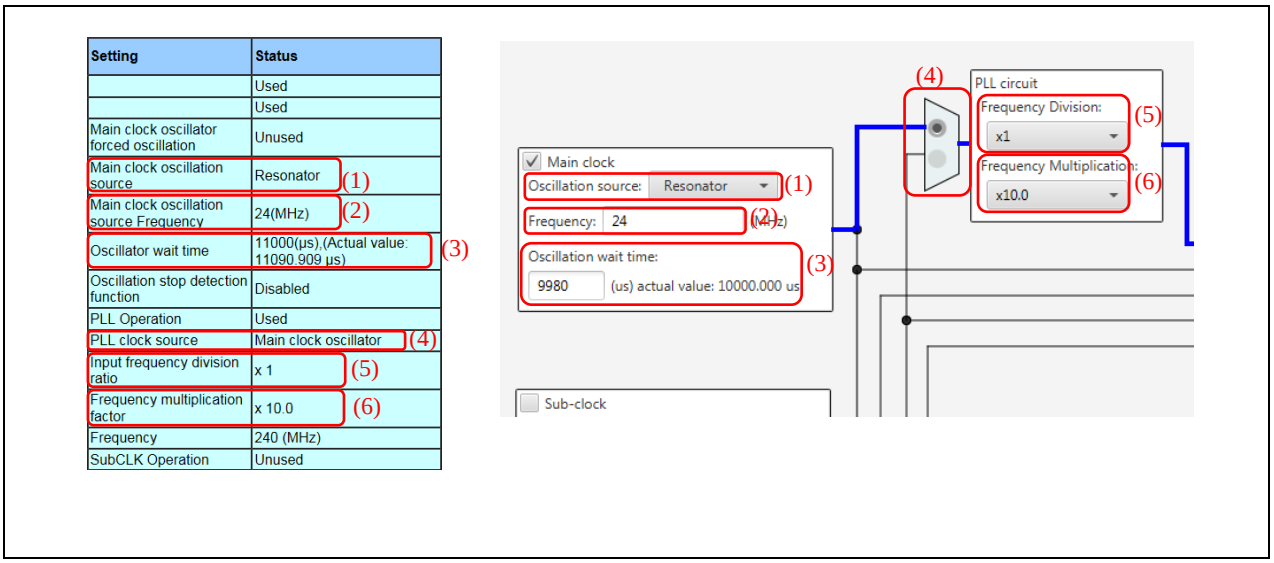


Figure 2.10 Window for Using the Smart Configurator to Make Clock Settings

(3) Reflect the items in the [Setting] and [Status] columns in Macro.html of the report in the settings of the Smart Configurator.



Setting	Status
	Used
	Used
Main clock oscillator forced oscillation	Unused
Main clock oscillation source	Resonator (1)
Main clock oscillation source Frequency	24(MHz) (2)
Oscillator wait time	11000(μs),(Actual value: 11090.909 μs) (3)
Oscillation stop detection function	Disabled
PLL Operation	Used
PLL clock source	Main clock oscillator (4)
Input frequency division ratio	x 1 (5)
Frequency multiplication factor	x 10.0 (6)
Frequency	240 (MHz)
SubCLK Operation	Unused

Figure 2.11 Setting Clocks in the Smart Configurator (1)

SubCLK Operation	Unused	(7)
HOCO Operation	Unused	(8)
LOCO Operation	Used	(9)
Low speed clock oscillator (LOCO) setting	240 (kHz)	(10)
IWDT Operation	Unused	(11)
RTC clock setting	Unused	
Clock source	PLL circuit	(12)
System clock (ICLK)	x 1/2 120 (MHz)	(13)
Peripheral module clock (PCLKA)	x 1/2 120 (MHz)	
Peripheral module clock (PCLKB)	x 1/4 60 (MHz)	
Peripheral module clock (PCLKC)	x 1/4 60 (MHz)	
Peripheral module clock (PCLKD)	x 1/4 60 (MHz)	
External bus clock (BCLK)	x 1/4 60 (MHz)	
Flash IF clock (FCLK)	x 1/4 60 (MHz)	
USB clock (UCLK)	x 1/5 48 (MHz)	(14)
BCLK Operation	Unused	(15)
SDCLK Operation	Unused	(16)

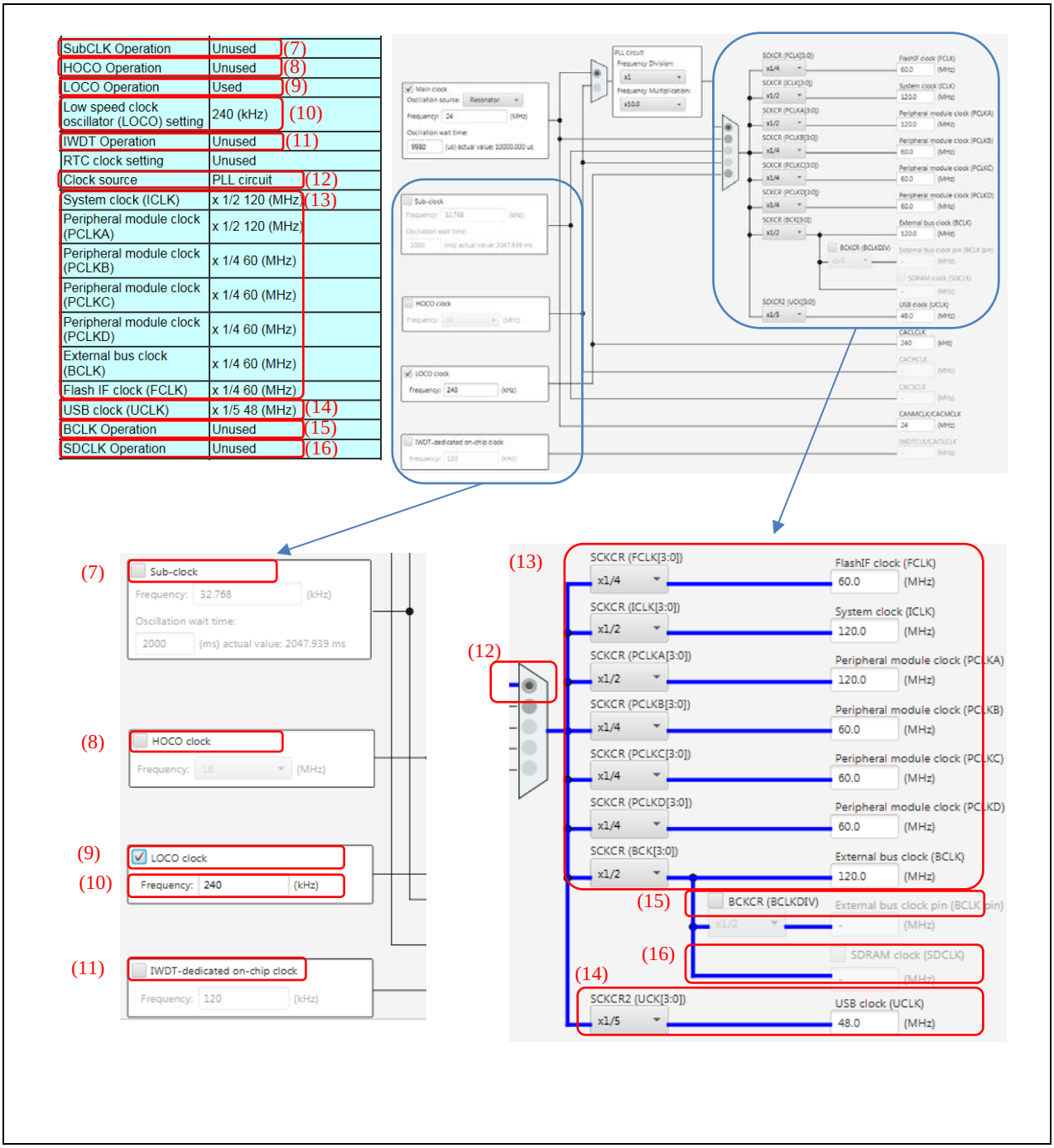


Figure 2.12 Setting Clocks in the Smart Configurator (2)

Table 2.5 Settings of the Clock Generator (1)

	Code Generator		Smart Configurator	
	Item to be Set ([Macro] or [Setting] in Macro.html)	**Setting ([Status] in Macro.html)**	**Item to be Set**	**Setting**
(1)	Main clock oscillation source	Resonator	[Main clock] Oscillation source	Resonator
(2)	Main clock oscillation source Frequency	24(MHz)	[Main clock] Frequency	24(MHz)
(3)	Oscillator wait time	11000(us) (Actual value: 11090.909 us)	[Main clock] Oscillation wait time	11000(us) (Actual value: 11090.909 us)
(4)	PLL clock source	Main clock oscillator	Check that the PLL clock source is selected as [Main clock].	
(5)	Input frequency division ratio	× 1	[PLL circuit] Frequency Division	× 1
(6)	Frequency multiplication factor	× 10.0	[PLL circuit] Frequency Multiplication	× 10.0
(7)	SubCLK Operation	Unused	Sub-clock	Not selected
(8)	HOCO Operation	Unused	HOCO clock	Not selected
(9)	LOCO Operation	Used	LOCO clock	Selected
(10)	Low speed clock oscillator (LOCO) setting	240 (kHz)	Frequency	240 (kHz)
(11)	IWDT operation	Unused	IWDT-dedicated on-chip clock	Not selected
(12)	Clock source	PLL circuit	Check that the clock source is selected as [PLL circuit].	

Table 2.6 Settings of the Clock Generator (2)

	Code Generator		Smart Configurator	
	Item to be Set ([Macro] or [Setting] in Macro.html)	**Setting ([Status] in Macro.html)**	**Item to be Set**	**Setting**
(13)	System clock (ICLK)	× 1/2 120 (MHz)	SCKCR (ICLK[3:0])	× 1/2
			System clock (ICLK)	120.0 (MHz)
	Peripheral module clock (PCLKA)	× 1/2 120 (MHz)	SCKCR (PCLKA[3:0])	× 1/2
			Peripheral module clock A (PCLKA)	120.0 (MHz)
	Peripheral module clock (PCLKB)	× 1/4 60 (MHz)	SCKCR (PCLKB[3:0])	× 1/4
			Peripheral module clock B (PCLKB)	60.0 (MHz)
	Peripheral module clock (PCLKC)	× 1/4 60 (MHz)	SCKCR (PCLKC[3:0])	× 1/4
			Peripheral module clock C (PCLKC)	60.0 (MHz)
	Peripheral module clock (PCLKD)	× 1/4 60 (MHz)	SCKCR (PCLKD[3:0])	× 1/4
			Peripheral module clock D (PCLKD)	60.0 (MHz)
	Peripheral module clock (BCLK)	× 1/4 60 (MHz)	SCKCR (BCK[3:0])	× 1/4
			External bus clock (BCLK)	60.0 (MHz)
	Flash lF clock (FCLK)	× 1/4 60 (MHz)	SCKCR (FCLK[3:0])	× 1/4
			FlashlF clock (FCLK)	60.0 (MHz)
(14)	USB clock (UCLK)	× 1/5 48 (MHz)	SCKCR2 (UCK[3:0])	× 1/5
			USB clock (UCLK)	48.0 (MHz)
(15)	BCLK Operation	Unused	BCKCR (BCLKDIV)	Not selected
(16)	SDCLK Operation	Unused	SDRAM clock (SDCLK)	Not selected

2.5.3 Setting the Compare Match Timers

Set the compare-match timers.

(1) Refer to '(1) To add a Code Generator component' under section 3.3.1, Add a software component into the project, in the Renesas e^2 studio Smart Configurator User Guide, and add the compare match timers as components of the project.
In the [Add new configuration for selected component] dialog box, use the default names as the names of the configurations of the resources, as listed below.

Table 2.7 Correspondence between Resources and the Configuration Names of the Compare Match Timers

Component Type	Component	Resource	Configuration Name
Code Generator	Compare match timer	CMT0	Config_CMT0 (default)
		CMT1	Config_CMT1 (default)
		CMT2	Config_CMT2 (default)

(2) Display the parts showing the settings of the compare match timers in the Macro.html file of the report that was output in section 2.3, Generating a Report on the Source Project.

Compare Match Timer				Used
	CMT0			Used
			Compare match timer operation setting	Used
			Count clock setting	PCLK/8
			Interval value setting	1ms,(Actual value: 1)
			Enable compare match interrupt (CMI0)	Used
			Priority	Level 10
	CMT1			Used
			Compare match timer operation setting	Used
			Count clock setting	PCLK/32
			Interval value setting	20ms,(Actual value: 20)
			Enable compare match interrupt (CMI1)	Used
			Priority	Level 10
	CMT2			Used
			Compare match timer operation setting	Used
			Count clock setting	PCLK/512
			Interval value setting	200ms,(Actual value: 200.004267)
			Enable compare match interrupt (CMI2)	Used
			Priority	Level 10

Figure 2.13 Report on the Compare Match Timers Output by the Code Generator

(3) Open the window for setting compare match timer CMT0 that was created in step (1).

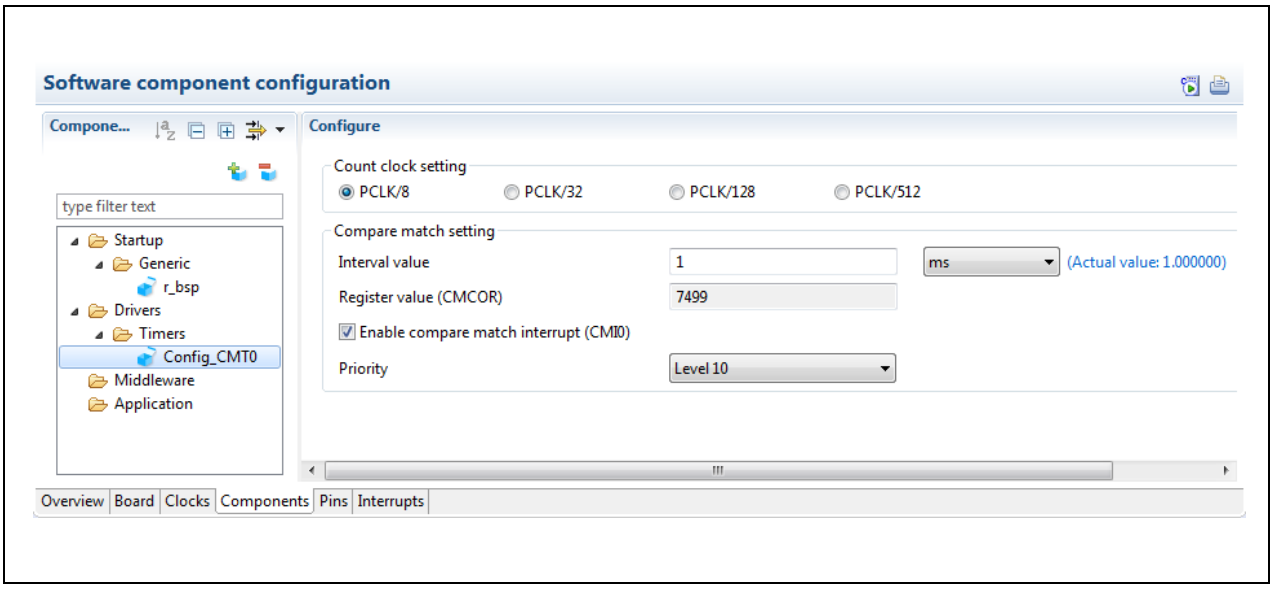


Figure 2.14 Window for Setting the Compare Match Timer (CMT0) in the Smart Configurator

(4) Reflect the settings of the compare match timers in Macro.html in those for CMT0 in the Smart Configurator.

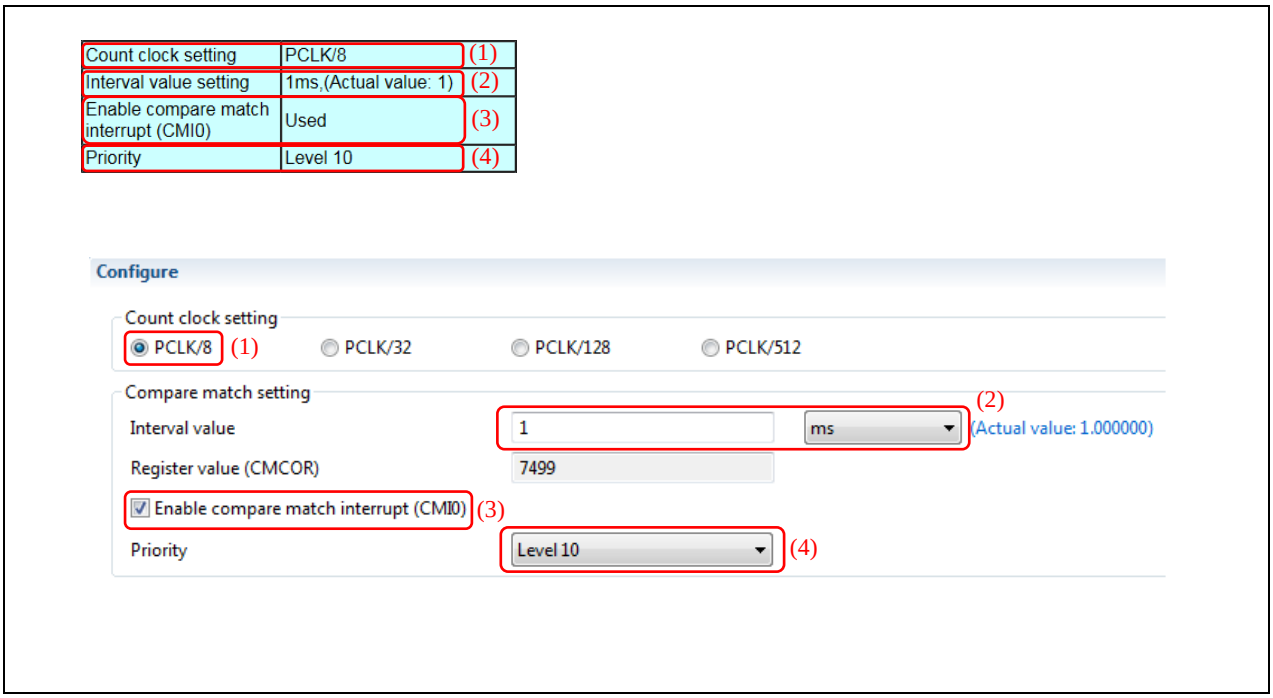


Figure 2.15 Settings of the Compare Match Timer (CMT0) in the Smart Configurator

Table 2.8 Settings of the Compare Match Timer (CMT0)

	Code Generator		Smart Configurator	
	Item to be Set ([Macro] or [Setting] in Macro.html)	**Setting ([Status] in Macro.html)**	**Item to be Set**	**Setting**
(1)	Count clock setting	PCLK/8	Count clock setting	PCLK/8
(2)	Interval value setting	1 ms (Actual value: 1)	[Compare match setting] Interval value	1 ms (Actual value: 1.000000)
(3)	Enable compare match interrupt (CMI0)	Used	[Compare match setting] Enable compare match interrupt (CMI0)	Selected
(4)	Priority	Level 10	[Compare match setting] Priority	Level 10

(5) Similarly, add CMT1 and CMT2 as components and make their settings.

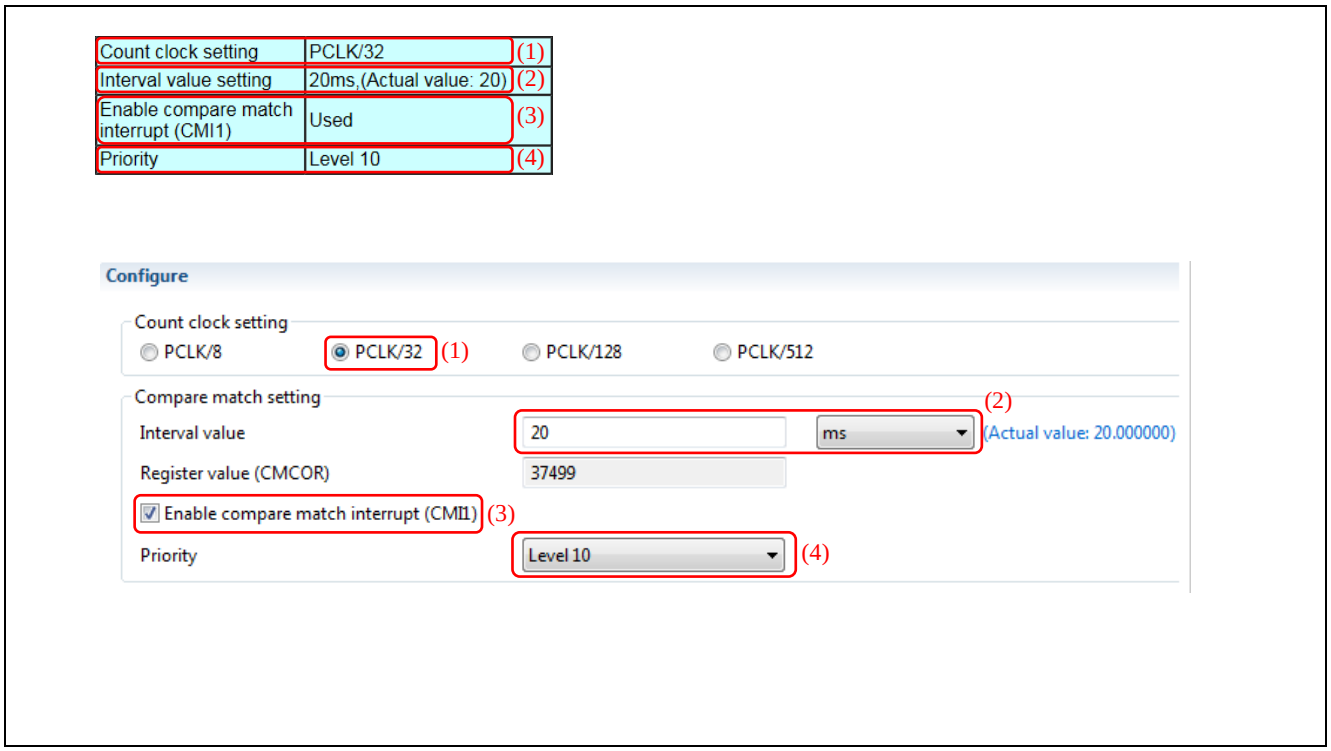


Figure 2.16 Settings of the Compare Match Timer (CMT1) in the Smart Configurator

Table 2.9 Settings of the Compare Match Timer (CMT1)

	Code Generator		Smart Configurator	
	Item to be Set ([Macro] or [Setting] in Macro.html)	**Setting ([Status] in Macro.html)**	**Item to be Set**	**Setting**
(1)	Count clock setting	PCLK/32	Count clock setting	PCLK/32
(2)	Interval value setting	20 ms, (Actual value: 20)	[Compare match setting] Interval value	20 ms (Actual value: 20.000000)
(3)	Enable compare match interrupt (CMI1)	Used	[Compare match setting] Enable compare match interrupt (CMI1)	Selected
(4)	Priority	Level 10	[Compare match setting] Priority	Level 10

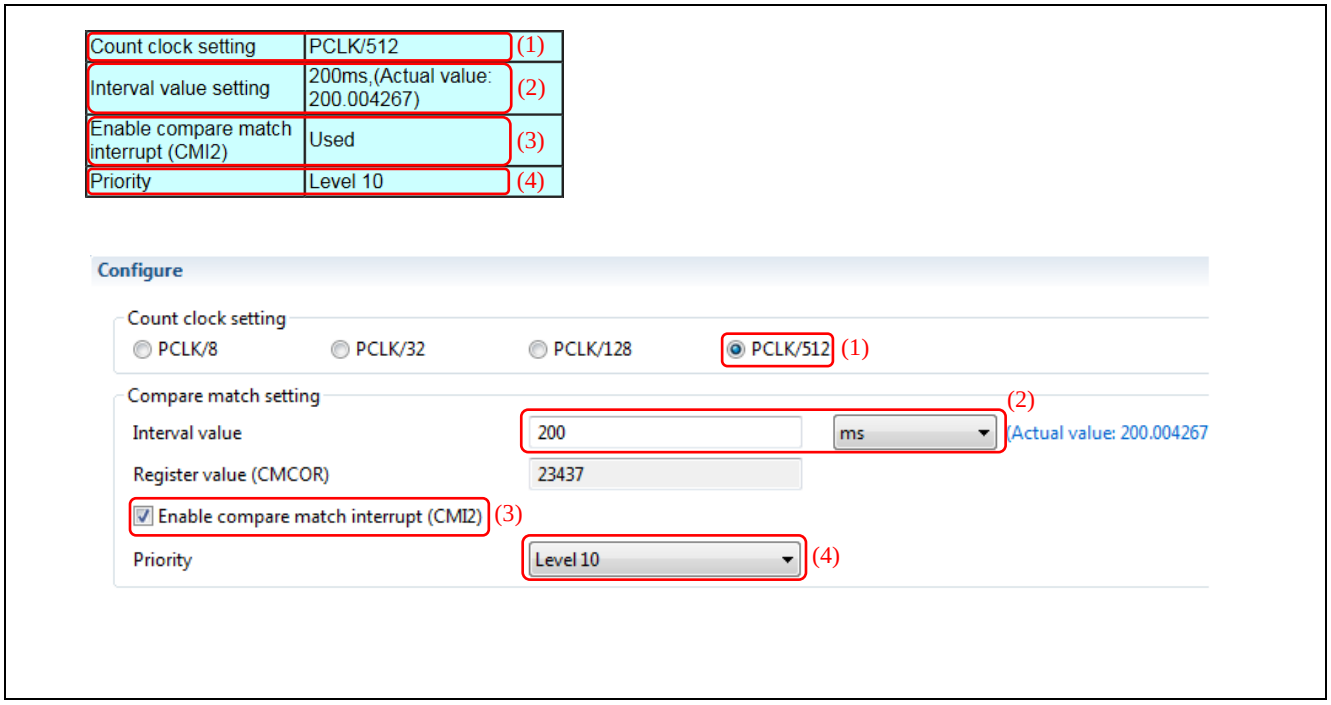


Figure 2.17 Settings of the Compare Match Timer (CMT2) in the Smart Configurator

Table 2.10 Settings of the Compare Match Timer (CMT2)

	Code Generator		Smart Configurator	
	Item to be Set ([Macro] or [Setting] in Macro.html)	**Setting ([Status] in Macro.html)**	**Item to be Set**	**Setting**
(1)	Count clock setting	PCLK/512	Count clock setting	PCLK/512
(2)	Interval value setting	200 ms (Actual value: 200.004267)	[Compare match setting] Interval value	200 ms (Actual value: 200.004267)
(3)	Enable compare match interrupt (CMI2)	Used	[Compare match setting] Enable compare match interrupt (CMI2)	Selected
(4)	Priority	Level 10	[Compare match setting] Priority	Level 10

2.5.4 Setting the Serial Communications Interfaces

Set the serial communications interfaces.

(1) Refer to '(1) To add a Code Generator component' under section 3.3.1, Add a software component into the project, in the Renesas e^2 studio Smart Configurator User Guide, and add the compare match timers as components of the project.
Since SCI6 and SCI7 are used in the simple SPI mode and SCI asynchronous mode, respectively, set the component, resource, and operation/work mode as listed below, using the default configuration names.

Table 2.11 Correspondence between Resources and the Configuration Names of the Serial Communications Interfaces

Component Type	Component	Resource	Configuration Name	Operation/ Work Mode
Code Generator	SPI clock synchronous mode	SCI6	Config_SCI6 (default)	Master transmission
	SCI (SCIF) asynchronous mode	SCI7	Config_SCI7 (default)	Transmission and reception

(2) Display the parts showing the settings of the serial communications interfaces in the Macro.html file of the report that was output in section 2.3, Generating a Report on the Source Project.

Serial Communications Interface				Used
	SCI6			Used
			Function setting	Simple SPI bus (Master transmit only)
			SMOSI6	P00
		SimpleSPIMode_Master_Transmit6		Used
			Transfer direction setting	MSB-first
			Data inversion setting	Normal
			Transfer clock	Internal clock
			Bit rate	1500000 (bps)
			Enable modulation duty correction	Unused
			SCK6 pin function selection	Clock output
			SCK6	P02
			Clock delay	Clock is not delayed
			Enable clock polarity inversion	Unused
			Transmit data handling	Data handled in interrupt service routine
			TXI6 priority	Level 15 (highest)
			TEI6, ERI6 priority (Group BL0)	Level 15 (highest)
			Transmission end	Used
	SCI7			Used
			Function setting	Asynchronous mode (Transmission/reception)
			TXD7	P90
			RXD7	P92
		AsynchronousMode_TransmitReceive7		Used
			Start bit edge detection setting	Falling edge on RXD7 pin
			Data length setting	8 bits
			Parity setting	None
			Stop bit length setting	1 bit
			Transfer direction setting	LSB-first
			Transfer clock	Internal clock
			Bit rate	19200 (bps)
			Enable modulation duty correction	Used
			SCK7 pin function	SCK7 is not used
			Enable noise filter	Unused
			Hardware flow control setting	None
			Transmit data handling	Data handled in interrupt service routine
			Receive data handling	Data handled in interrupt service routine
			Enable error interrupt (ERI7)	Used
			TXI7 priority	Level 15 (highest)
			RXI7 priority	Level 15 (highest)
			TEI7, ERI7 priority (Group BL0)	Level 15 (highest)
			Transmission end	Used
			Reception end	Used
			Reception error	Used

Figure 2.18 Report on the Serial Communications Interfaces Output by the Code Generator

(3) Open the window for setting serial communications interface SCI6 that was created in step (1).

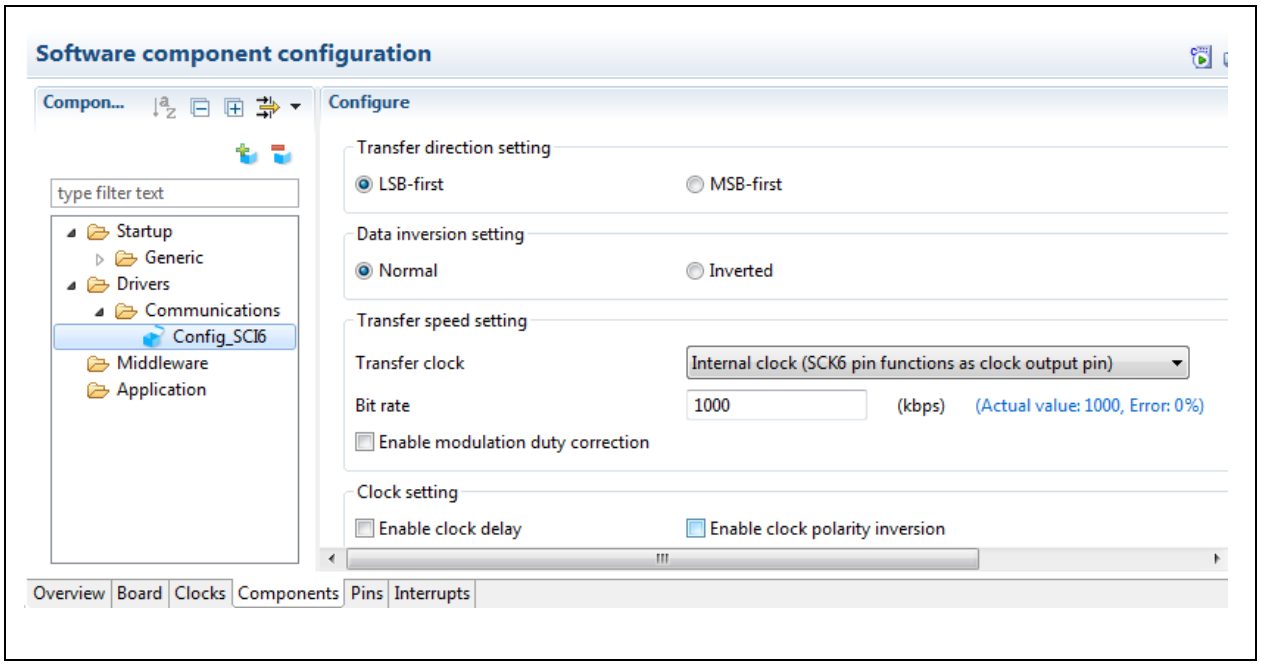


Figure 2.19 Window for Setting the Serial Communications Interface (SCI6) in the Smart Configurator

(4) Reflect the settings of the serial communications interfaces in Macro.html in those for SCI6 in the Smart Configurator.
Pins to handle SMOSI6 and SCK6 are set on the [Pins] tabbed page.

Function setting	Simple SPI bus (Master transmit only)
SMOSI6	P00
	Used
Transfer direction setting	MSB-first (1)
Data inversion setting	Normal (2)
Transfer clock	Internal clock (3)
Bit rate	1500000 (bps) (4)
Enable modulation duty correction	Unused (5)
SCK6 pin function selection	Clock output
SCK6	P02
Clock delay	Clock is not delayed (6)
Enable clock polarity inversion	Unused (7)
Transmit data handling	Data handled in interrupt service routine (8)
TXI6 priority	Level 15 (highest) (9)
TEI6, ERI6 priority (Group BL0)	Level 15 (highest) (10)
Transmission end	Used (11)

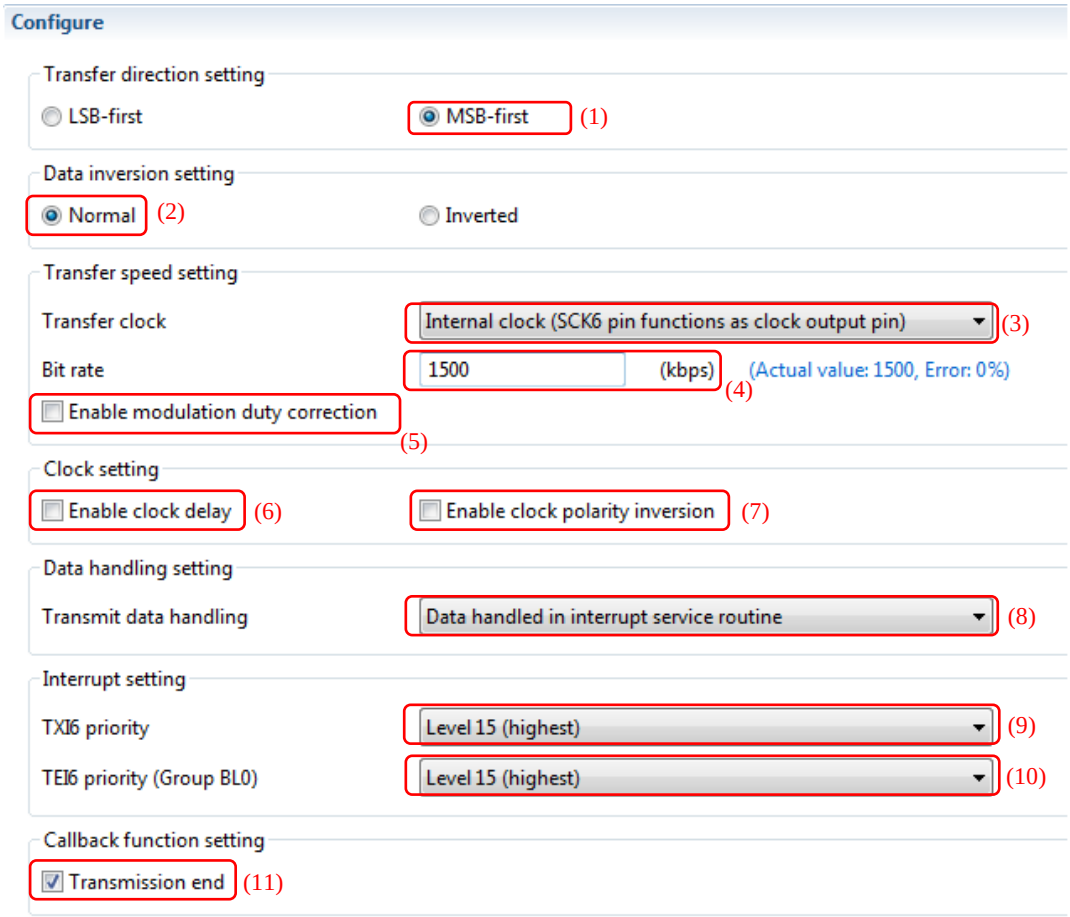


Figure 2.20 Settings of the Serial Communications Interface (SCI6) in the Smart Configurator

Table 2.12 Settings of the Serial Communications Interface (SCI6)

	Code Generator		Smart Configurator	
	Item to be Set ([Macro] or [Setting] in Macro.html)	**Setting ([Status] in Macro.html)**	**Item to be Set**	**Setting**
(1)	Transfer direction setting	MSB-first	Transfer direction setting	MSB-first
(2)	Data inversion setting	Normal	Data inversion setting	Normal
(3)	Transfer clock	Internal clock	[Transfer speed setting] Transfer clock	Internal clock (SCK6 pin functions as clock output pin)
(4)	Bit rate	1500000 (bps)	[Transfer speed setting] Bit rate	1500 (kbps)
(5)	Enable modulation duty correction	Unused	[Transfer speed setting] Enable modulation duty correction	Not selected
(6)	Clock delay	Clock is not delayed	[Clock setting] Enable clock delay	Not selected
(7)	Enable clock polarity inversion	Unused	[Clock setting] Enable clock polarity inversion	Not selected
(8)	Transmit data handling	Data handled in interrupt service routine	[Data handling setting] Transmit data handling	Data handled in interrupt service routine
(9)	TXI6 priority	Level 15 (highest)	[Interrupt setting] TXI6 priority	Level 15 (highest)
(10)	TEI6, ERI6 priority (Group BL0)	Level 15 (highest)	[Interrupt setting] TEI6 priority (Group BL0)	Level 15 (highest)
(11)	Transmission end	Used	[Callback function setting] Transmission end	Selected

(5) Open the window for setting serial communications interface SCI7 that was created in step (1).

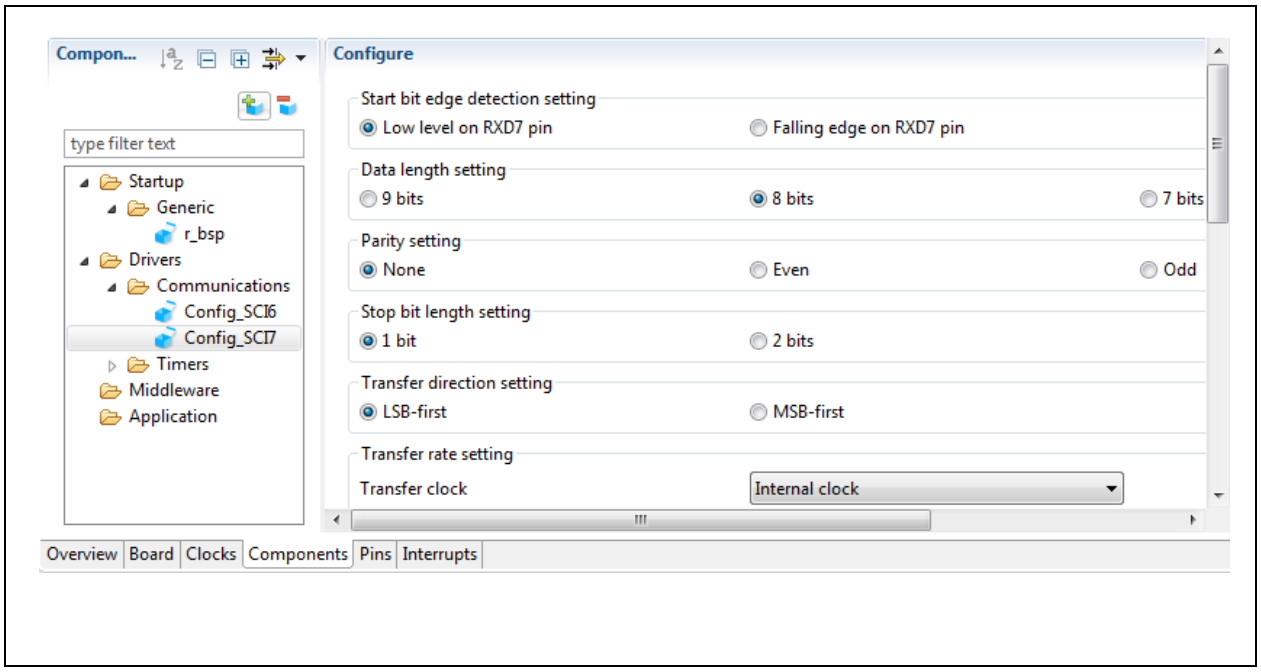


Figure 2.21 Window for Setting the Serial Communications Interface (SCI7) in the Smart Configurator

(6) Reflect the settings of the serial communications interfaces in Macro.html to those for SCI7 in the Smart Configurator. Pins to handle TXD7 and RXD7 are set on the [Pins] tabbed page.

	TXD7	P90
	RXD7	P92
AsynchronousMode_TransmitReceive7		Used
	Start bit edge detection setting	Falling edge on RXD7 pin (1)
	Data length setting	8 bits (2)
	Parity setting	None (3)
	Stop bit length setting	1 bit (4)
	Transfer direction setting	LSB-first (5)
	Transfer clock	Internal clock (6)
	Bit rate	19200 (bps) (7)
	Enable modulation duty correction	Used (8)
	SCK7 pin function	SCK7 is not used (9)
	Enable noise filter	Unused (10)

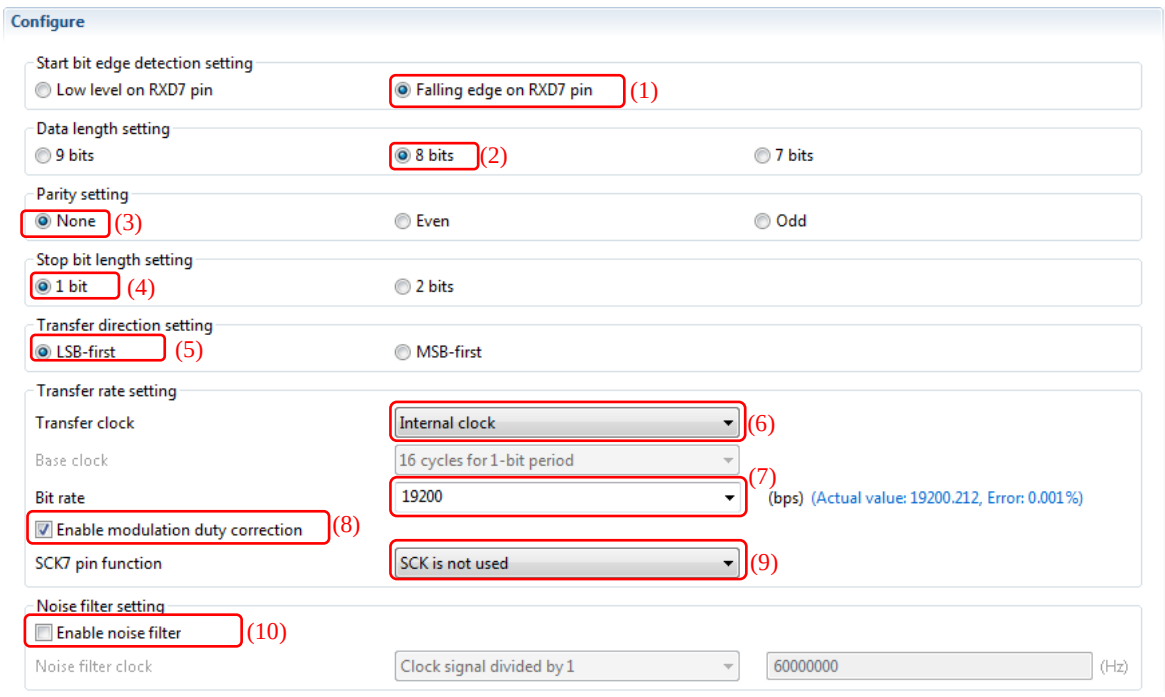


Figure 2.22 Settings of the Serial Communications Interface (SCI7) in the Smart Configurator (1)

(11)	Hardware flow control setting	None
(12)	Transmit data handling	Data handled in interrupt service routine
(13)	Receive data handling	Data handled in interrupt service routine
(14)	Enable error interrupt (ERI7)	Used
(15)	TXI7 priority	Level 15 (highest)
(16)	RXI7 priority	Level 15 (highest)
(17)	TEI7, ERI7 priority (Group BL0)	Level 15 (highest)
(18)	Transmission end	Used
(19)	Reception end	Used
(20)	Reception error	Used

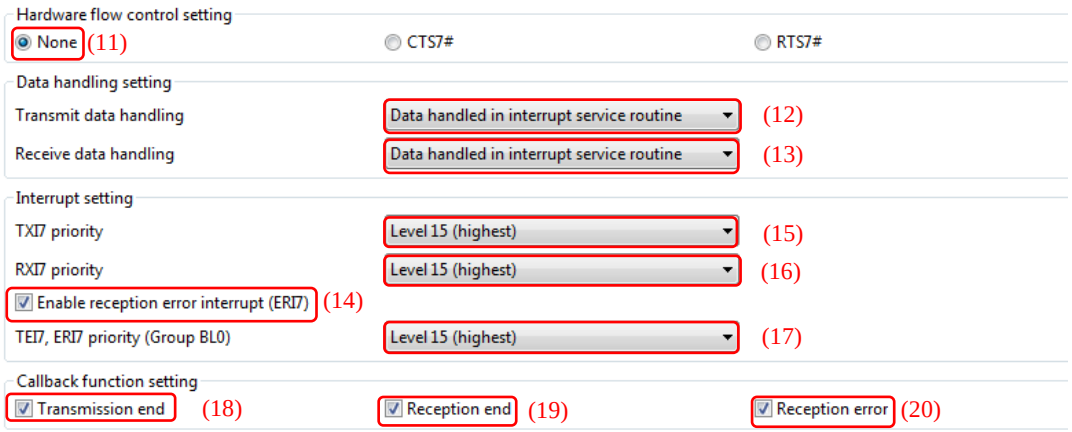


Figure 2.23 Settings of the Serial Communications Interface (SCI7) in the Smart Configurator (2)

Table 2.13 Settings of the Serial Communications Interface (SCI7)

	Code Generator		Smart Configurator	
	Item to be Set ([Macro] or [Setting] in Macro.html)	**Setting ([Status] in Macro.html)**	**Item to be Set**	**Setting**
(1)	Start bit edge detection setting	Falling edge on RXD7 pin	Start bit edge detection setting	Falling edge on RXD7 pin
(2)	Data length setting	8 bits	Data length setting	8 bits
(3)	Parity setting	None	Parity setting	None
(4)	Stop bit length setting	1 bit	Stop bit length setting	1 bit
(5)	Transfer direction setting	LSB-first	Transfer direction setting	LSB-first
(6)	Transfer clock	Internal clock	[Transfer rate setting] Transfer clock	Internal clock
(7)	Bit rate	19200 (bps)	[Transfer rate setting] Bit rate	19200 (bps)
(8)	Enable modulation duty correction	Used	[Transfer rate setting] Enable modulation duty correction	Selected
(9)	SCK7 pin function	SCK7 is not used	[Transfer rate setting] SCK7 pin function	SCK7 is not used
(10)	Enable noise filter	Unused	[Noise filter setting] Enable noise filter	Not selected
(11)	Hardware flow control setting	None	Hardware flow control setting	None
(12)	Transmit data handling	Data handled in interrupt service routine	[Data handling setting] Transmit data handling	Data handled in interrupt service routine
(13)	Receive data handling	Data handled in interrupt service routine	[Data handling setting] Receive data handling	Data handled in interrupt service routine
(14)	Enable error interrupt (ERI7)	Used	[Interrupt setting] Enable reception error interrupt (ERI7)	Selected
(15)	TXI7 priority	Level 15 (highest)	[Interrupt setting] TXI7 priority	Level 15 (highest)
(16)	RXI7 priority	Level 15 (highest)	[Interrupt setting] RXI7 priority	Level 15 (highest)
(17)	TEI7, ERI7 priority (Group BL0)	Level 15 (highest)	[Interrupt setting] TEI7, ERI7 priority (Group BL0)	Level 15 (highest)
(18)	Transmission end	Used	[Callback function setting] Transmission end	Selected

(19)	Reception end	Used	[Callback function setting] Reception end	Selected
(20)	Reception error	Used	[Callback function setting] Reception error	Selected

(7) Make settings of pins. Select the [Pins] and [Pin Function] tabs. When [SCI6] or [SCI7] under [Serial communications interface] is selected in the left pane, a list of pin functions that may be used is displayed in the right [Pin Function] pane.

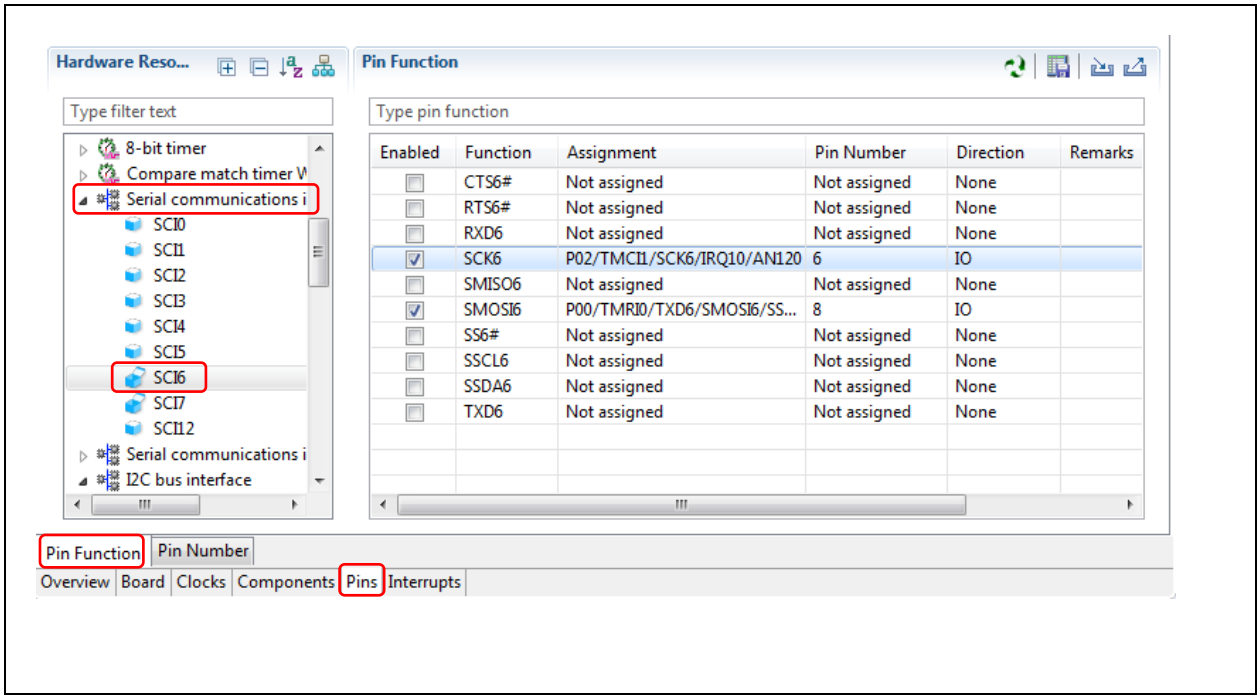
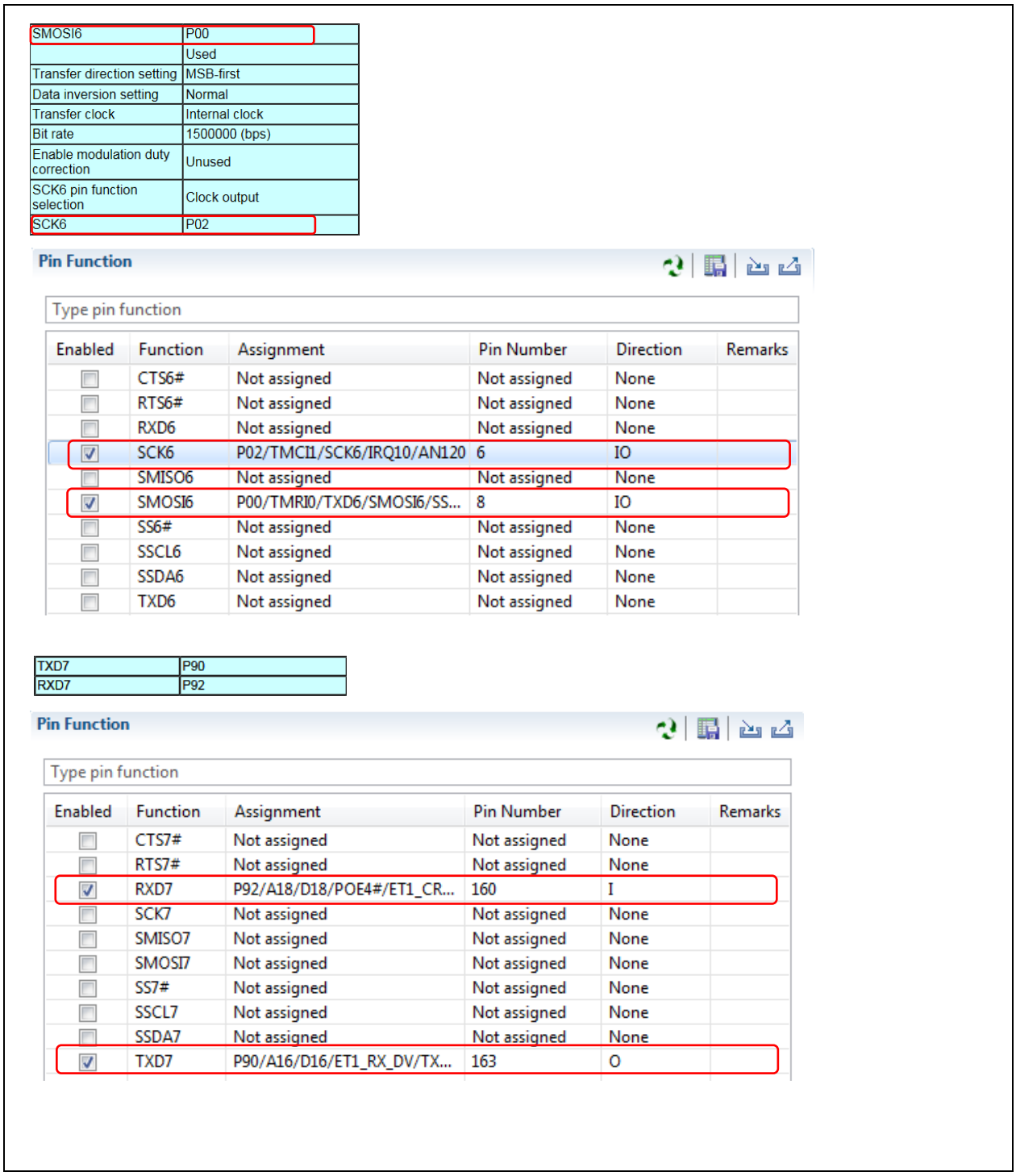


Figure 2.24 Settings of Pins for a Serial Communications Interface in the Smart Configurator (1)

(8) Assign the SMOSI6 and SCK6 functions to pins.



SMOSI6	P00
	Used
Transfer direction setting	MSB-first
Data inversion setting	Normal
Transfer clock	Internal clock
Bit rate	1500000 (bps)
Enable modulation duty correction	Unused
SCK6 pin function selection	Clock output
SCK6	P02

Pin Function

Type pin function

Enabled	Function	Assignment	Pin Number	Direction	Remarks
☐	CTS6#	Not assigned	Not assigned	None	
☐	RTS6#	Not assigned	Not assigned	None	
☐	RXD6	Not assigned	Not assigned	None	
☑	SCK6	P02/TMCI1/SCK6/IRQ10/AN120	6	IO	
☐	SMISO6	Not assigned	Not assigned	None	
☑	SMOSI6	P00/TMRI0/TXD6/SMOSI6/SS...	8	IO	
☐	SS6#	Not assigned	Not assigned	None	
☐	SSCL6	Not assigned	Not assigned	None	
☐	SSDA6	Not assigned	Not assigned	None	
☐	TXD6	Not assigned	Not assigned	None	

TXD7	P90
RXD7	P92

Pin Function

Type pin function

Enabled	Function	Assignment	Pin Number	Direction	Remarks
☐	CTS7#	Not assigned	Not assigned	None	
☐	RTS7#	Not assigned	Not assigned	None	
☑	RXD7	P92/A18/D18/POE4#/ET1_CR...	160	I	
☐	SCK7	Not assigned	Not assigned	None	
☐	SMISO7	Not assigned	Not assigned	None	
☐	SMOSI7	Not assigned	Not assigned	None	
☐	SS7#	Not assigned	Not assigned	None	
☐	SSCL7	Not assigned	Not assigned	None	
☐	SSDA7	Not assigned	Not assigned	None	
☑	TXD7	P90/A16/D16/ET1_RX_DV/TX...	163	O	

Figure 2.25 Settings of Pins for a Serial Communications Interface in the Smart Configurator (2)

Table 2.14 Settings of Pins for the Serial Communications Interfaces (SCI6 and SCI7)

Code Generator		Smart Configurator	
Item to be Set ([Macro] or [Setting] in Macro.html)	**Setting ([Status] in Macro.html)**	**Item to be Set**	**Setting**
SCK6	P02	SCK6	P02/TMCI1/SCK6/IRQ10/AN120
SMOSI6	P00	SMOSI6	P00/TMRI0/TXD6/SMOSI6/SSDA6/IRQ8/AN118
TXD7	P90	TXD7	P90/A16/D16/ET1_RX_DV/TXD7/SMOSI7/SSDA7/AN114
RXD7	P92	RXD7	P92/A18/D18/POE4#/ET1_CRS/RMII1_CRS_DV/RXD7/SMOSO7/SSCL7/AN116

2.5.5 Setting Other Peripheral Functions

For settings of the 12-bit A/D converter, interrupt functions, and port pins, refer to the steps described in Table 2.2, Report Files Output by the Report Function of the Code Generator, section 2.5.2, Setting the Clock Generator, section 2.5.3, Setting the Compare Match Timers, and section 2.5.4, Setting the Serial Communications Interfaces, and set the Smart Configurator in the equivalent ways.

2.5.6 Generating Code

When all settings are finished, save the project and click on the [Generate Code] button  to make the Smart Configurator generate the code.

2.6 Porting User-defined Source Code

2.6.1 Overview

The user-created source files or user-defined source code written in the source files which were generated by the Code Generator in the source project created by using the Code Generator must be copied to the destination project created by using the Smart Configurator.

Figure 2.26 shows the procedure for porting user-defined source code.

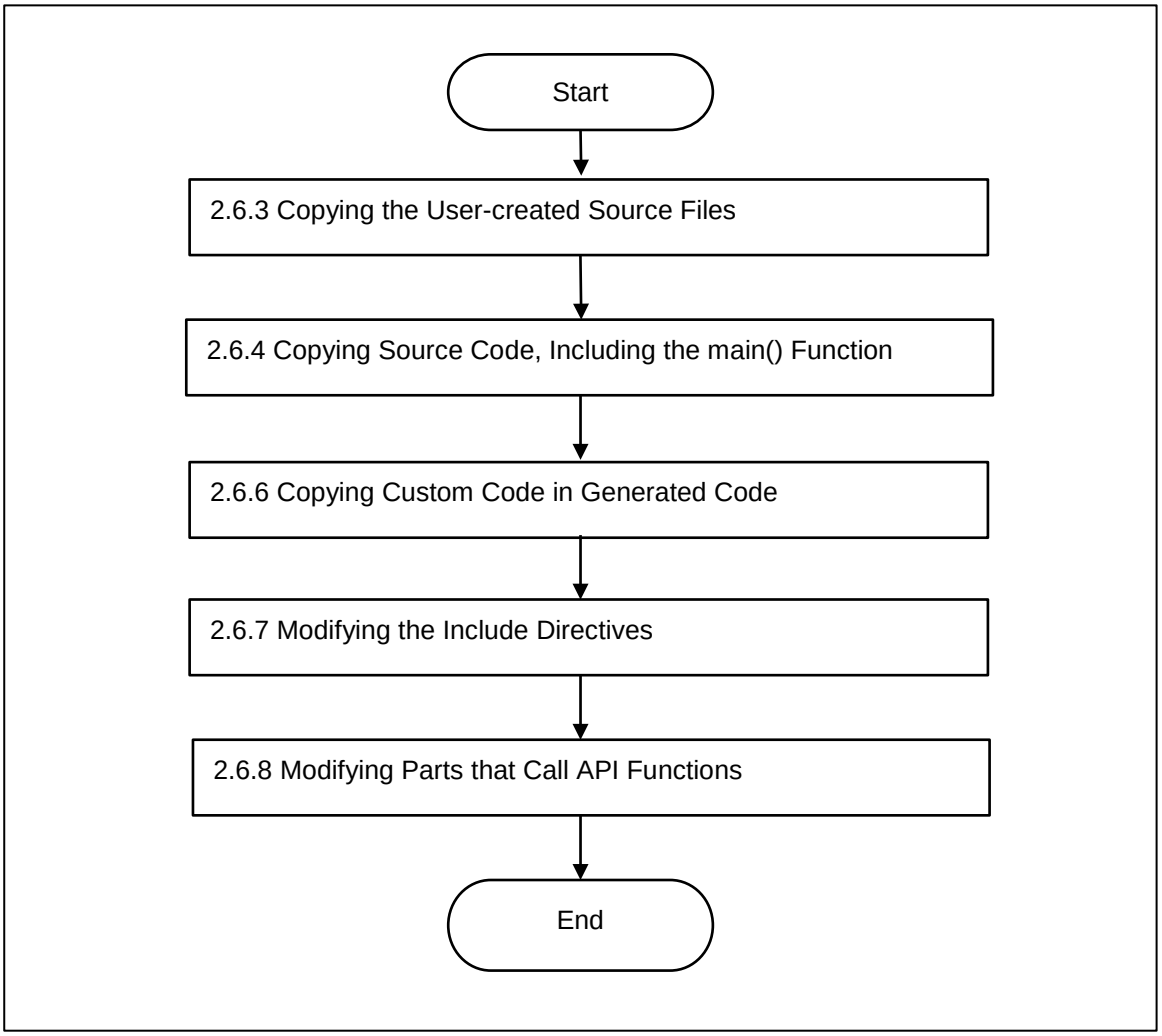


Figure 2.26 Procedure for Porting User-defined Source Code

2.6.2 Areas for Writing User-defined Source Code

Files generated by the Code Generator and Smart Configurator include areas where the user can freely add code. Areas for custom code are indicated by comments as shown below.

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

In the comments above, the part ‘xxxxxx’ depends on the area where custom code is to be added. For example, it is the word ‘include’ in the part where include declarations are to be written and the word ‘global’ in the part where global variables are to be defined.

Custom code located between these comments must be copied from the source projects to the destination projects.

2.6.3 Copying the User-created Source Files

Copy the user-created source files other than the source files output by the Code Generator from the source project.

As shown below, copy the source files and header files from the folder other than ‘cg_src’ in the source project to the ‘src’ folder in the destination project.

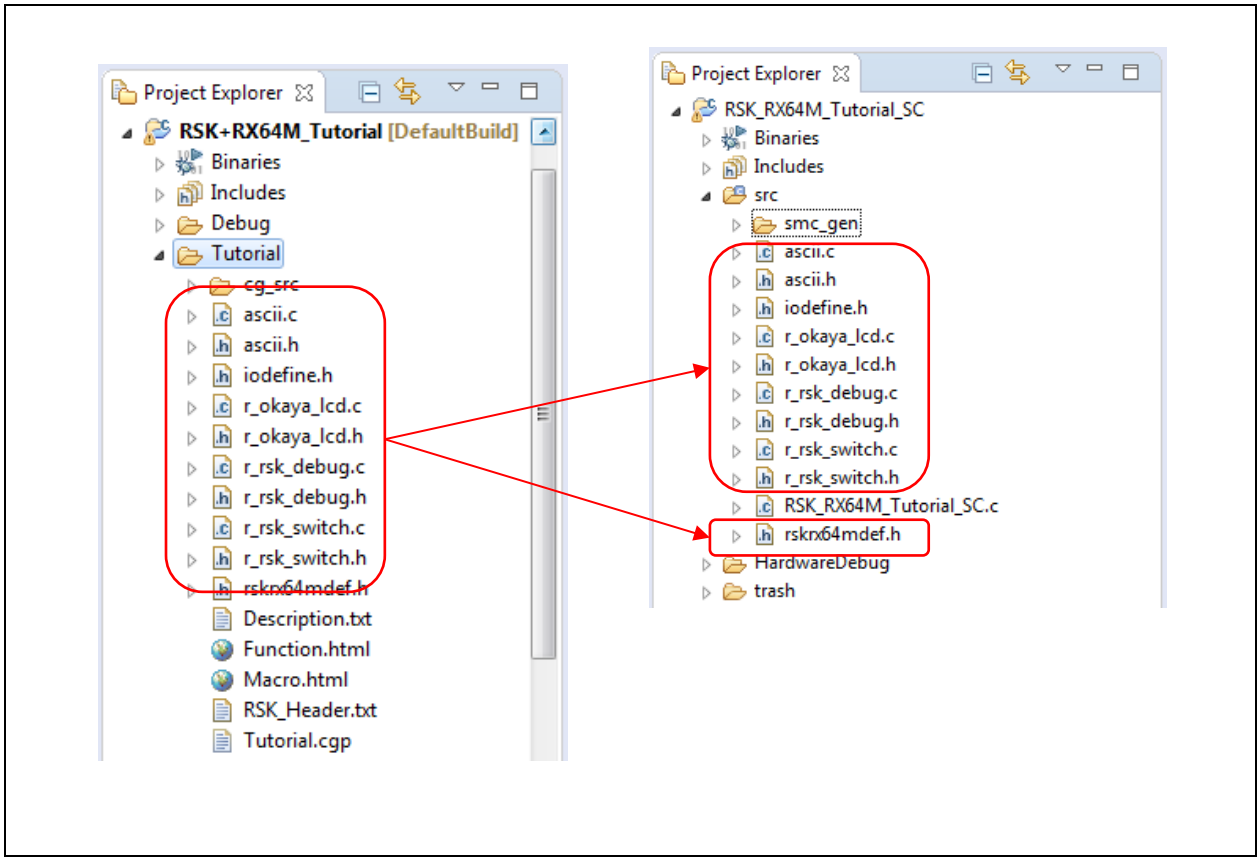


Figure 2.27 Copying the User-created Source Files

Since the copied source files will use the names of the API functions generated by the Code Generator, these names must be modified to those generated by the Smart Configurator. In addition, the header files to be included must also be modified as required. For modifying the names of the API functions, refer to section 2.6.8, Modifying Parts that Call API Functions. For modifying the include directives, refer to section 2.6.7, Modifying the Include Directives.

The ‘src’ folder must be added to the include directory since the ‘src’ folder will include header files to be included. Add the include directory through the following steps.

(1) Right-click on [RSK_RX64M_Tutorial_SC], which is the destination project, to open [Properties for RSK_RX64M_Tutorial_SC]. Select [Settings] under [C/C++ Build] in the left pane. Select the [Tool Settings] tabbed page in the right window. Then select [Source] under [Compiler] and click on the [Add] button in the [Include file directories] category.

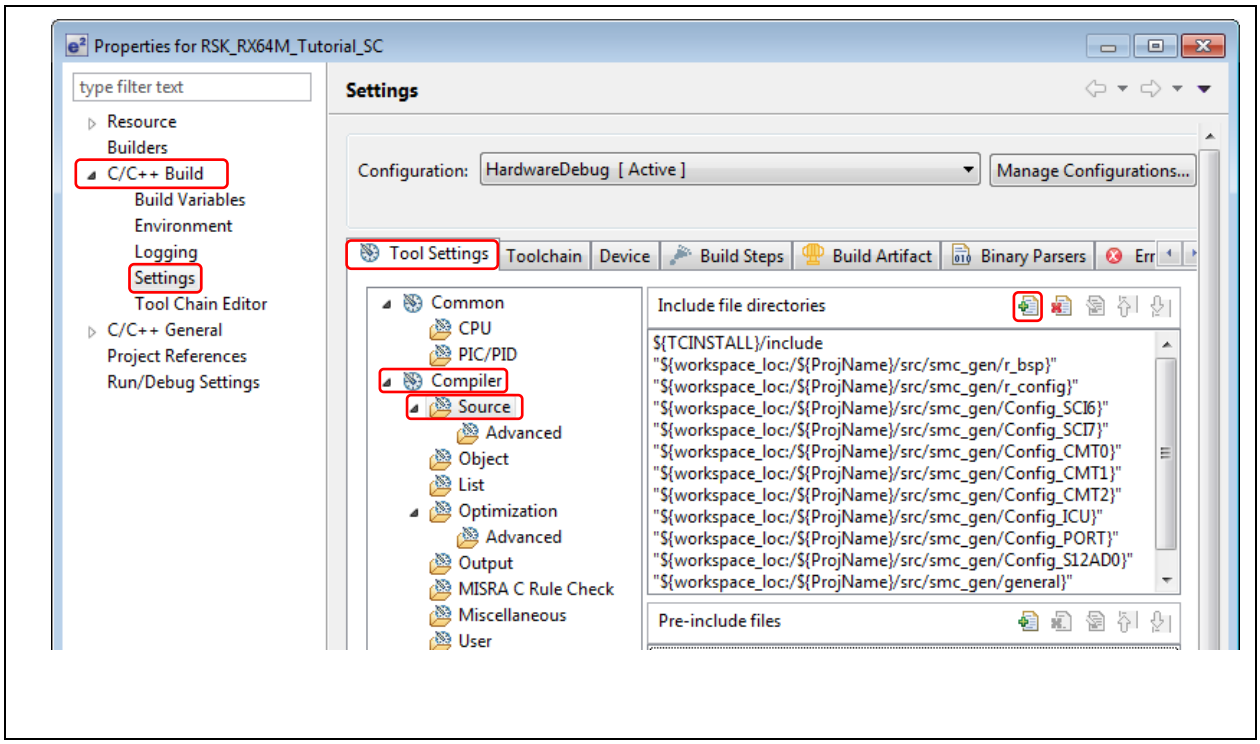


Figure 2.28 Adding the Include Directory (1)

(2) Select [Workspace] in the [Add directory path] dialog box. In the [Folder selection] dialog box, select the folder (e.g. ‘src’) to be added as the include directory and click on [OK]. Check that the folder specified for [Directory] has been added to the [Add directory path] dialog box and click on [OK].

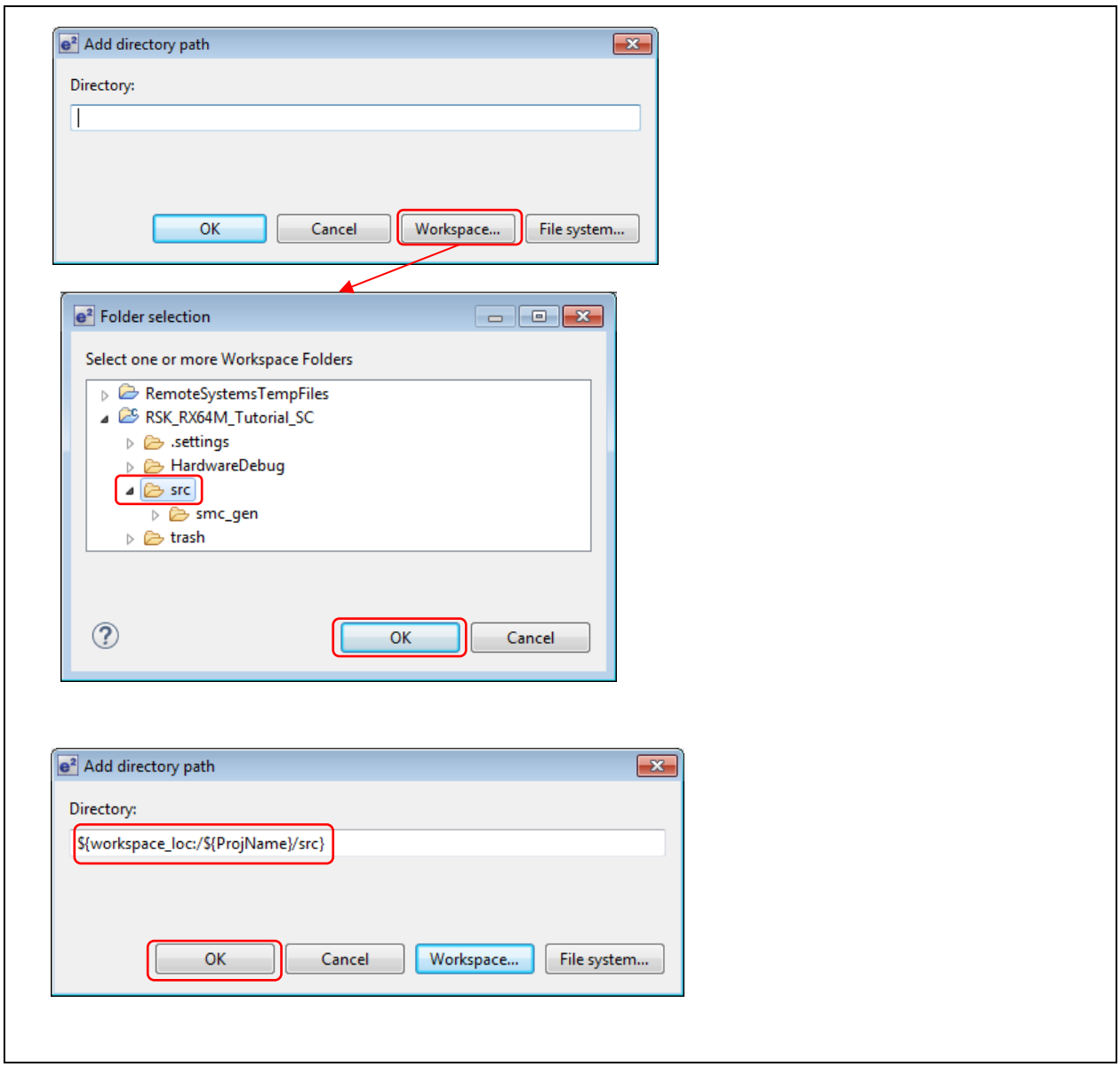


Figure 2.29 Adding the Include Directory (2)

2.6.4 Copying Source Code, Including the main() Function

Copy user-defined source code from the source file including the main() function.

In the source project, the main() function will be in the ‘r_cg_main.c’ file in the ‘cg_src’ folder. Since ‘r_cg_main.c’ is a source file generated by the Code Generator, the user-defined source code will be in the area between comments.

In the destination project, the file that includes the main() function is not among the files generated by the Smart Configurator. Instead, the main() function is in the {ProjName}.c file, which is automatically generated when a new project is created. Since the name of the destination project is ‘RSK_RX64M_Tutorial_SC’ in this application note, the main() function will be in ‘RSK_RX64M_Tutorial_SC.c’. All source code in ‘{ProjName}.c’ is user-defined.

Open ‘r_cg_main.c’ and copy all source code written between comments of the type shown in section 2.6.2, Areas for Writing User-defined Source Code.

The following explains how to copy the include directives as an example.

For the include files, the source code between comments of the type shown in section 2.6.2, Areas for Writing User-defined Source Code, will generally be copied. Header files that contain user-defined source code (e.g. ‘r_cg_userdefine.h’) are also copied.

An include directive for ‘r_smc_entry.h’ is automatically written in ‘{ProjName}.c’ when this source file is generated during creation of the new project.

Preprocessor directives for the inclusion of other header files written in ‘r_cg_main.c’ (e.g. ‘r_cg_macrodriver.h’ through ‘r_cg_s12ad.h’) need not be copied unless these header files contain user-defined source code.

The statements to be copied are highlighted in yellow below.

r_cg_main.c

```
/***********************************************************************************************
************************************
Includes
***********************************************************************************************
************************************/
#include "r_cg_macrodriver.h"
#include "r_cg_cgc.h"
#include "r_cg_icu.h"
#include "r_cg_port.h"
#include "r_cg_cmt.h"
#include "r_cg_sci.h"
#include "r_cg_s12ad.h"
/* Start user code for include. Do not edit comment generated here */
#include "r_okaya_lcd.h"
#include "r_rsk_switch.h"
#include "r_rsk_debug.h"
#include "rskrx64mdef.h"
/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"
```

{ProjName}.c

```
#include "r_smc_entry.h"
#include "r_okaya_lcd.h"
#include "r_rsk_switch.h"
#include "r_rsk_debug.h"
#include "rskrx64mdef.h"
#include "r_cg_userdefine.h"
```

Source code written between comments of the type shown in section 2.6.2, Areas for Writing User-defined Source Code, such as the user-defined prototype declarations and function definitions, is copied to '{ProjName}.c', preserving the original order.

In the example, copy the user-defined prototype and variable declarations highlighted in yellow on the next page.

r_cg_main.c

```
/***********************************************************************************************
***********************************
Global variables and functions
***********************************************************************************************
***********************************/
/* Start user code for global. Do not edit comment generated here */

/* Prototype declaration for cb_switch_press */
static void cb_switch_press (void);

/* Prototype declaration for get_adc */
static uint16_t get_adc (void);

/* Prototype declaration for lcd_display_adc */
static void lcd_display_adc (const uint16_t adc_result);

/* Prototype declaration for uart_display_adc */
static void uart_display_adc (const uint8_t adc_count, const uint16_t adc_result);

/* Variable to store the A/D conversion count for user display */
static uint8_t adc_count = 0;

/* Prototype declaration for led_display_count */
static void led_display_count (const uint8_t count);

/* Variable for flagging user requested ADC conversion */
volatile uint8_t g_adc_trigger = FALSE;

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

{ProjName}.c

```
void main(void);

/* Prototype declaration for cb_switch_press */
static void cb_switch_press (void);

/* Prototype declaration for get_adc */
static uint16_t get_adc (void);

/* Prototype declaration for lcd_display_adc */
static void lcd_display_adc (const uint16_t adc_result);

/* Prototype declaration for uart_display_adc */
static void uart_display_adc (const uint8_t adc_count, const uint16_t adc_result);

/* Prototype declaration for led_display_count */
static void led_display_count (const uint8_t count);

/* Variable to store the A/D conversion count for user display */
static uint8_t adc_count = 0;

/* Variable for flagging user requested ADC conversion */
volatile uint8_t g_adc_trigger = FALSE;
```

Copy the function calls and other code in the main() function that is highlighted in yellow below. All sections indicated as '- Omitted -' must also be copied. Code related to functions generated by the Code Generator, such as 'R_MAIN_UserInit()', need not be copied unless user-defined code has been added to the functions.

r_cg_main.c

```
void R_MAIN_UserInit(void);

/***********************************************************************************************
***********************************
* Function Name: main
* Description  : This function implements main function.
* Arguments    : None
* Return Value : None
***********************************************************************************************
***********************************/
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */

    /* Initialise the switch module */
    R_SWITCH_Init();


    /* Set the call back function when SW1 or SW2 is pressed */
    R_SWITCH_SetPressCallback(cb_switch_press);

    - Omitted -

    /* Set up SCI7 receive buffer and callback function */
    R_SCI7_Serial_Receive((uint8_t *)&g_rx_char, 1);

    /* Enable SCI7 operations */
    R_SCI7_Start();

    while (1U)
    {
        uint16_t adc_result;

        /* Wait for user requested A/D conversion flag to be set (SW1 or SW2) */
        if (TRUE == g_adc_trigger)
        {
            /* Call the function to perform an A/D conversion */
            adc_result = get_adc();

    - Omitted -

            /* Send the result to the UART */
            uart_display_adc(adc_count, adc_result);

            /* Reset the flag */
            g_adc_complete = FALSE;
        }
        else
        {
            /* do nothing */
        }
    }
    /* End user code. Do not edit comment generated here */
}
```

{ProjName}.c

```
void main(void)
{
    /* Initialise the switch module */
    R_SWITCH_Init();

    /* Set the call back function when SW1 or SW2 is pressed */
    R_SWITCH_SetPressCallback(cb_switch_press);

    - Omitted -

    while (1U)
    {
        uint16_t adc_result;

        /* Wait for user requested A/D conversion flag to be set (SW1 or SW2) */
        if (TRUE == g_adc_trigger)
        {
            /* Call the function to perform an A/D conversion */
            adc_result = get_adc();

    - Omitted -

            /* Send the result to the UART */
            uart_display_adc(adc_count, adc_result);

            /* Reset the flag */
            g_adc_complete = FALSE;
        }
        else
        {
            /* do nothing */
        }
    }
}
```

Copy the function calls and other code in the private function that is highlighted in yellow below. All sections indicated as '- Omitted -' must also be copied. Code related to functions generated by the Code Generator, such as 'R_MAIN_UserInit()', need not be copied unless user-defined code has been added to the functions.

r_cg_main.c

```
/***********************************************************************************************
***********************************
* Function Name: R_MAIN_UserInit
* Description  : This function adds user code before implementing main function.
* Arguments    : None
* Return Value : None
***********************************************************************************************
***********************************/
void R_MAIN_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    /* End user code. Do not edit comment generated here */
}

/* Start user code for adding. Do not edit comment generated here */

/***********************************************************************************
* Function Name : cb_switch_press
* Description   : Switch press callback function. Sets g_adc_trigger flag.
* Argument      : none
* Return value  : none
***********************************************************************************/
static void cb_switch_press (void)
{
    /* Check if switch 1 or 2 was pressed */
    if (g_switch_flag & (SWITCHPRESS_1 | SWITCHPRESS_2))
    {
        /* set the flag indicating a user requested A/D conversion is required */
        g_adc_trigger = TRUE;

        /* Clear flag */
        g_switch_flag = 0x0;
    }
}

/***********************************************************************************
* End of function cb_switch_press
***********************************************************************************/

    - Omitted -

/***********************************************************************************
* Function Name : led_display_count
* Description   : Converts count to binary and displays on 4 LEDS0-3
* Argument      : uint8_t count
* Return value  : none
***********************************************************************************/
static void led_display_count (const uint8_t count)
{
    /* Set LEDs according to lower nibble of count parameter */
    LED0 = (uint8_t)((count & 0x01) ? LED_ON : LED_OFF);
    LED1 = (uint8_t)((count & 0x02) ? LED_ON : LED_OFF);
    LED2 = (uint8_t)((count & 0x04) ? LED_ON : LED_OFF);
    LED3 = (uint8_t)((count & 0x08) ? LED_ON : LED_OFF);
}

/***********************************************************************************
* End of function led_display_count
***********************************************************************************/


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

{ProjName}.c

```
/***********************************************************************
* Function Name : cb_switch_press
* Description   : Switch press callback function. Sets g_adc_trigger flag.
* Argument      : none
* Return value  : none
***********************************************************************/
static void cb_switch_press (void)
{
    /* Check if switch 1 or 2 was pressed */
    if (g_switch_flag & (SWITCHPRESS_1 | SWITCHPRESS_2))
    {
        /* set the flag indicating a user requested A/D conversion is required */
        g_adc_trigger = TRUE;

        /* Clear flag */
        g_switch_flag = 0x0;
    }
}

/***********************************************************************
* End of function cb_switch_press
***********************************************************************/


    - Omitted -

/***********************************************************************
* Function Name : led_display_count
* Description   : Converts count to binary and displays on 4 LEDS0-3
* Argument      : uint8_t count
* Return value  : none
***********************************************************************/
static void led_display_count (const uint8_t count)
{
    /* Set LEDs according to lower nibble of count parameter */
    LED0 = (uint8_t)((count & 0x01) ? LED_ON : LED_OFF);
    LED1 = (uint8_t)((count & 0x02) ? LED_ON : LED_OFF);
    LED2 = (uint8_t)((count & 0x04) ? LED_ON : LED_OFF);
    LED3 = (uint8_t)((count & 0x08) ? LED_ON : LED_OFF);
}

/***********************************************************************
* End of function led_display_count
***********************************************************************/
```

2.6.5 Correspondences between Code Generated by the Code Generator and by the Smart Configurator

Since files generated by the Code Generator and the Smart Configurator are not paired and are output in different folder structures, copying between the appropriate files is required.

The following lists the main files and output folders for which user-created code must be copied from the source project to the destination project.

Table 2.15 Correspondences between Code Generated by the Code Generator and by the Smart Configurator

Code Generator		Smart Configurator		Note
Output Folder	**Source File**	**Output Folder**	**Source File**	
cg_src	r_cg_main.c	src	{ProjName}.c	File that contains main().
cg_src	r_cg_userdefine.h	src¥smc_gen¥general	r_cg_userdefine.h	Header file for user-defined code that is used in common with peripheral functions.
cg_src	r_cg_xxx.c	src¥smc_gen¥ Config_XXX	Config_XXX.c	Source file for initializing and operating peripheral functions. With the Smart Configurator, one file is output for each resource.
	r_cg_xxx_user.c	src¥smc_gen¥ Config_XXX	Config_XXX_user.c	Source file for writing user-defined code or interrupt callback functions after peripheral functions have been initialized. With the Smart Configurator, one file is output for each resource.
	r_cg_xxx.h	src¥smc_gen¥ general	r_cg_xxx.h	Header file including macro definitions for setting the SFR registers. These files are used in common with the peripheral functions.
		src¥smc_gen¥ Config_XXX	Config_XXX.h	Header file for Config_XXX.c.

Note: ‘xxx’ and ‘XXX’ represent the names of peripheral functions.

For files requiring the porting of custom code in the RSK+RX64M sample code, the following shows the correspondences between the locations in code generated by the Code Generator and by the Smart Configurator. In this example, the porting of custom code is not required for files on a gray background. The other files have custom code that requires porting.

Table 2.16 Files that Require Porting of Custom Code in the RSK+RX64M Sample Code (1)

Peripheral Function	Code Generator		Smart Configurator	
	Output Folder	Source File	Output Folder	Source File
File including main()	cg_src	r_cg_main.c	src	{ProjName}.c
General settings	cg_src	r_cg_userdefine.h	src¥smc_gen¥general	r_cg_userdefine.h
Interrupt controller	cg_src	r_cg_icu.c	src¥smc_gen¥ Config_ICU	Config_ICU.c
			src¥smc_gen¥ general	r_smc_interrupt.c
		r_cg_icu_user.c	src¥smc_gen¥ Config_ICU	Config_ICU_user.c
		r_cg_icu.h	src¥smc_gen¥ general	r_cg_icu.h
			src¥smc_gen¥ Config_ICU	Config_ICU.h
			src¥smc_gen¥ general	r_smc_interrupt.h
I/O port	cg_src	r_cg_port.c	src¥smc_gen¥ Config_PORT	Config_PORT.c
		r_cg_port_user.c	src¥smc_gen¥ Config_PORT	Config_PORT_user.c
		r_cg_port.h	src¥smc_gen¥ general	r_cg_port.h
			src¥smc_gen¥ Config_PORT	Config_PORT.h
Compare match timer	cg_src	r_cg_cmt.c	src¥smc_gen¥ Config_CMT0	Config_CMT0.c
			src¥smc_gen¥ Config_CMT1	Config_CMT1.c
			src¥smc_gen¥ Config_CMT2	Config_CMT2.c
		r_cg_cmt_user.c	src¥smc_gen¥ Config_CMT0	Config_CMT0_user.c
			src¥smc_gen¥ Config_CMT1	Config_CMT1_user.c
			src¥smc_gen¥ Config_CMT2	Config_CMT2_user.c
		r_cg_cmt.h	src¥smc_gen¥ general	r_cg_cmt.h
			src¥smc_gen¥ Config_CMT0	Config_CMT0.h
			src¥smc_gen¥ Config_CMT1	Config_CMT1.h
			src¥smc_gen¥ Config_CMT2	Config_CMT2.h

Table 2.17 Files that Require Porting of Custom Code in the RSK+RX64M Sample Code (2)

Peripheral Function	Code Generator		Smart Configurator	
	Output Folder	Source File	Output Folder	Source File
Serial communi-cations interface	cg_src	r_cg_sci.c	src¥smc_gen¥ Config_SCI6	Config_SCI6.c
			src¥smc_gen¥ Config_SCI7	Config_SCI7.c
		r_cg_sci_user.c	src¥smc_gen¥ Config_SCI6	Config_SCI6_user.c
			src¥smc_gen¥ Config_SCI7	Config_SCI7_user.c
		r_cg_sci.h	src¥smc_gen¥ general	r_cg_sci.h
			src¥smc_gen¥ Config_SCI6	Config_SCI6.h
			src¥smc_gen¥ Config_SCI7	Config_SCI7.h
12-bit A/D converter	cg_src	r_cg_s12ad.c	src¥smc_gen¥ Config_S12AD0	Config_S12AD0.c
		r_cg_s12ad_user.c	src¥smc_gen¥ Config_S12AD0	Config_S12AD0_user.c
		r_cg_s12ad.h	src¥smc_gen¥ general	r_cg_s12ad.h
			src¥smc_gen¥ Config_S12AD0	Config_S12AD0.h

2.6.6 Copying Custom Code in Generated Code

The following explains how to copy custom code from the files in the source project to the destination project according to the correspondences listed in Table 2.16, taking the case of the SCI6 serial communications interface (in use for SPI master transmission) as an example.

Firstly, copy the custom code for SCI6 that is highlighted in yellow below from ‘r_cg_sci.h’ to ‘Config_SCI6.h’.

r_cg_sci.h

```
/* Start user code for function. Do not edit comment generated here */
/* Exported functions used to transmit a number of bytes and wait for completion */
MD_STATUS R_SCI6_SPIMasterTransmit(uint8_t * const tx_buf, const uint16_t tx_num);
MD_STATUS R_SCI7_AsyncTransmit(uint8_t * const tx_buf, const uint16_t tx_num);
```

Config_SCI6.h

```
/* Start user code for function. Do not edit comment generated here */
/* Exported functions used to transmit a number of bytes and wait for completion */
MD_STATUS R_SCI6_SPIMasterTransmit(uint8_t * const tx_buf, const uint16_t tx_num);
/* End user code. Do not edit comment generated here */
```

After that, copy the custom code for SCI6 that is highlighted in yellow below from ‘r_cg_sci_user.c’ to ‘Config_SCI6_user.c’.

r_cg_sci_user.c

```
/* Start user code for global. Do not edit comment generated here */
/* Flag used locally to detect transmission complete */

/* Global used to receive a character from the PC terminal */
uint8_t g_rx_char;

/* Flag used to control transmission to PC terminal */
volatile uint8_t g_tx_flag = FALSE;

/* Flag used locally to detect transmission complete */
static volatile uint8_t sci6_txdone;
static volatile uint8_t sci7_txdone;

- Omitted -

static void r_sci6_callback_transmitend(void)
{
    /* Start user code. Do not edit comment generated here */
    sci6_txdone = TRUE;
    /* End user code. Do not edit comment generated here */
}

- Omitted -
/* Start user code for adding. Do not edit comment generated here */
/***********************************************************************
* Function Name: R_SCI6_SPIMasterTransmit
* Description  : This function sends SPI6 data to slave device.
* Arguments    : tx_buf -
*                    transfer buffer pointer
*                tx_num -
*                    buffer size
* Return Value : status -
*                    MD_OK or MD_ARGERROR
***********************************************************************/
MD_STATUS R_SCI6_SPIMasterTransmit (uint8_t * const tx_buf,
                                    const uint16_t tx_num)
{
    MD_STATUS status = MD_OK;

    /* clear the flag before initiating a new transmission */
    sci6_txdone = FALSE;

    /* Send the data using the API */
    status = R_SCI6_SPI_Master_Send(tx_buf, tx_num);

    /* Wait for the transmit end flag */
    while (FALSE == sci6_txdone)
    {
        /* Wait */
    }

    return (status);
}

/***********************************************************************
* End of function R_SCI6_SPIMasterTransmit
***********************************************************************/
```

Since ‘R_SCI6_SPI_Master_Send’, highlighted in blue, is the name of the API function from the Code Generator, the name must be modified to that from the Smart Configurator. Details of the steps for this modification are described in section 2.6.8, Modifying Parts that Call API Functions.

Config_SCI6_user.c

```
extern uint8_t *gp_sci6_tx_address;             /* SCI6 transmit buffer address */
extern uint16_t g_sci6_tx_count;                /* SCI6 transmit data number */
/* Start user code for global. Do not edit comment generated here */
/* Flag used locally to detect transmission complete */
static volatile uint8_t sci6_txdone;
/* End user code. Do not edit comment generated here */

- Omitted -

static void r_Config_SCI6_callback_transmitend(void)
{
    /* Start user code for r_Config_SCI6_callback_transmitend. Do not edit comment
generated here */
        sci6_txdone = TRUE;
        /* End user code. Do not edit comment generated here */
}

- Omitted -

/* Start user code for adding. Do not edit comment generated here */
/***********************************************************************************
* Function Name: R_SCI6_SPIMasterTransmit
* Description  : This function sends SPI6 data to slave device.
* Arguments    : tx_buf -
*                    transfer buffer pointer
*                tx_num -
*                    buffer size
* Return Value : status -
*                    MD_OK or MD_ARGERROR
***********************************************************************************/
MD_STATUS R_SCI6_SPIMasterTransmit (uint8_t * const tx_buf,
                                    const uint16_t tx_num)
{
    MD_STATUS status = MD_OK;

    /* clear the flag before initiating a new transmission */
    sci6_txdone = FALSE;

    /* Send the data using the API */
    status = R_SCI6_SPI_Master_Send(tx_buf, tx_num);

    /* Wait for the transmit end flag */
    while (FALSE == sci6_txdone)
    {
        /* Wait */
    }

    return (status);
}

/***********************************************************************************
* End of function R_SCI6_SPIMasterTransmit
***********************************************************************************/
/* End user code. Do not edit comment generated here */
```

2.6.7 Modifying the Include Directives

The Code Generator mainly outputs files with names of the form ‘r_cg_xxx.h’ per peripheral function. The Smart Configurator outputs more than one header files by dividing them into ‘r_cg_xxx.h’ that are common to peripheral functions and ‘Config_XXX.h’ for resources of the component. Accordingly, the description of source code including header files must be modified to the appropriate names of header files. (‘xxx’ and ‘XXX’ represent the names of peripheral functions.)

For example, we search to find files that include ‘r_cg_sci.h’, which was copied in section 2.6.6, Copying Custom Code in Generated Code, and modify them to have the appropriate include directives.

After a search for files that contain ‘#include “r_cg_sci.h”’ in the source project, the results are as follows.

```
Tutorial
├──  cg_src
│      ├──  r_cg_hardware_setup.c
│      ├──  r_cg_main.c
│      ├──  r_cg_sci_user.c
│      └──  r_cg_sci.c
├──  r_okaya_lcd.c
└──  r_rsk_debug.h
```

Among these files that contain ‘#include “r_cg_sci.h”’, since files other than ‘r_cg_main.c’ ({ProjName}.c for the destination project) in the ‘cg_src’ folder include appropriate header files from the Smart Configurator, the include directives need not be modified.

For ‘r_okaya_lcd.c’ and ‘r_rsk_debug.h’, the include directives in the corresponding source files of the destination project require modification.

- For the source file (r_okaya_lcd.c)
  Open ‘r_okaya_lcd.c’ in the destination project and find the part where the SCI function is called. In ‘r_okaya_lcd.c’, the following two functions are called.
  — R_Config_SCI6_Start()
  — R_SCI6_SPIMasterTransmit()

  Since the prototype declarations are in ‘Config_SCI6.h’, the directives are modified so that this header file is included.

r_okaya_lcd.c (before modification)

```
/* SPI Driver Layer */
#include "r_cg_sci.h"
```

r_okaya_lcd.c (after modification)

```
/* SPI Driver Layer */
#include "Config_SCI6.h"
```

- For the header file (r_rsk_debug.h)
  Open ‘r_rsk_debug.h’ in the destination project and check if it has a function name or global variable declaration for use with SCI6 or SCI7. Since the file has a macro definition in which the R_SCI7_AsyncTransmit() function is used, the description is modified so that the header file of ‘Config_SCI7.h’ is to be included.

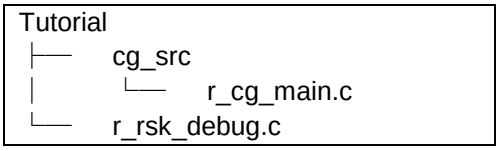
r_rsk_debug.h (before modification)

```
#include "r_cg_macrodriver.h"
#include "r_cg_sci.h"
```

r_rsk_debug.h (after modification)

```
#include "r_cg_macrodriver.h"
#include "Config_SCI7.h"
```

We then search for the files that include ‘r_rsk_debug.h’ in the source project and find the following two files.



A search for calls of functions for SCI6 or SCI7 in ‘r_cg_main.c’ ({ProjName}.c for the destination project) shows calls of the following two functions.

— R_SCI7_Start ()
— R_SCI7_Serial_Receive()

Since ‘Config_SCI7.h’ including the description of those prototype declarations has already been included in ‘r_rsk_debug.h’, the modification is already complete.
Regarding the calls of the R_SCI7_Start () or R_SCI7_Serial_Receive() API functions, refer to section 2.6.8, Modifying Parts that Call API Functions, and modify the statements in {ProjName}.c of the destination project.

2.6.8 Modifying Parts that Call API Functions

Parts of the source code copied in section 2.6.3, Copying the User-created Source Files, will have calls of API functions from the Code Generator. These names of the API functions must be modified to those from the Smart Configurator.

The user-created source file ‘r_okaya_lcd.c’ was copied in accord with section 2.6.3, Copying the User-created Source Files. The definition of the R_LCD_Init() function is in the file. We take this as an example.

R_LCD_Init() calls two user-defined functions and one API function. The user-defined functions are init_pmod_lcd() and R_LCD_ClearDisplay(), and the declarations are included in r_okaya_lcd.c. These functions need not be modified.

The API function R_SCI6_Start() generated by the Code Generator is called in r_okaya_lcd.c before modification. Since this function is R_Config_SCI6_Start() generated by the Smart Configurator (when the default configuration name is used during addition of the component), the name of the API function where it is called must be modified.

r_okaya_lcd.c (before modification)

```
void R_LCD_Init (void)
{
    /* Start SPI comm channel to LCD Display */
    R_SCI6_Start();

    /* initialise Standard PMOD display */
    init_pmod_lcd();

    /* clear the display before use */
    R_LCD_ClearDisplay(back_colour);
}
```

r_okaya_lcd.c (after modification)

```
void R_LCD_Init (void)
{
    /* Start SPI comm channel to LCD Display */
    R_Config_SCI6_Start();

    /* initialise Standard PMOD display */
    init_pmod_lcd();

    /* clear the display before use */
    R_LCD_ClearDisplay(back_colour);
}
```

Table 2.18 shows the correspondences between the names of API functions generated by the Code Generator and by the Smart Configurator. According to the table, modify the part where the API function is called.

The names of the API functions from the Smart Configurator listed in Table 2.18 are those when the default configuration names are set during the addition of the component. Since users are able to set configuration names, the names of the API functions may differ with the setting for the configuration name.

For the API functions from the Smart Configurator, refer to [Help - e^2 studio] - [e^2 studio User Guide] - [Building Projects] - [Smart Configurator] - [API reference].

Table 2.18 Correspondences between the Names of API Functions Generated by the Code Generator and by the Smart Configurator (1)

	Code Generator		Smart Configurator	
	File Name	**Name of the API Function**	**File Name**	**Name of the API Function**
Clock generator				
	r_cg_cgc.c	R_CGC_Create	r_smc_cgc.c	R_CGC_Create
Compare match timer				
	r_cg_cmt.c	R_CMT0_Create	Config_CMT0.c	R_Config_CMT0_Create
		R_CMT0_Start		R_Config_CMT0_Start
		R_CMT0_Stop		R_Config_CMT0_Stop
		R_CMT1_Create	Config_CMT1.c	R_Config_CMT1_Create
		R_CMT1_Start		R_Config_CMT1_Start
		R_CMT1_Stop		R_Config_CMT1_Stop
		R_CMT2_Create	Config_CMT2.c	R_Config_CMT2_Create
		R_CMT2_Start		R_Config_CMT2_Start
		R_CMT2_Stop		R_Config_CMT2_Stop
	r_cg_cmt_user.c	–	Config_CMT0_user.c	R_Config_CMT0_Create_UserInit
		r_cmt_cmi0_interrupt		r_Config_CMT0_cmi0_interrupt
		–	Config_CMT1_user.c	R_Config_CMT1_Create_UserInit
		r_cmt_cmi1_interrupt		r_Config_CMT1_cmi1_interrupt
		–	Config_CMT2_user.c	R_Config_CMT2_Create_UserInit
		r_cmt_cmi2_interrupt		r_Config_CMT2_cmi2_interrupt
Interrupt controller				
	r_cg_icu.c	R_ICU_Create	Config_ICU.c	R_Config_ICU_Create
		R_ICU_IRQ2_Start		R_Config_ICU_IRQ2_Start
		R_ICU_IRQ2_Stop		R_Config_ICU_IRQ2_Stop
		R_ICU_IRQ5_Start		R_Config_ICU_IRQ5_Start
		R_ICU_IRQ5_Stop		R_Config_ICU_IRQ5_Stop
	r_cg_icu_user.c	–	Config_ICU_user.c	R_Config_ICU_Create_UserInit
		r_icu_irq2_interrupt		r_Config_ICU_irq2_interrupt
		r_icu_irq5_interrupt		r_Config_ICU_irq5_interrupt
I/O port				
	r_cg_port.c	R_PORT_Create	Config_PORT.c	R_Config_PORT_Create
	r_cg_port_user.c	–	Config_PORT_user.c	R_Config_PORT_Create_UserInit

Table 2.19 Correspondences between the Names of API Functions Generated by the Code Generator and by the Smart Configurator (2)

	Code Generator		Smart Configurator	
	File Name	Name of the API Function	File Name	Name of the API Function
12-bit A/D converter				
	r_cg_s12ad.c	R_S12AD0_Create	Config_S12AD0.c	R_Config_S12AD0_Create
		R_S12AD0_Start		R_Config_S12AD0_Start
		R_S12AD0_Start		R_Config_S12AD0_Stop
		R_S12AD0_Get_ValueResult		R_Config_S12AD0_Get_ValueResult
		R_S12AD0_Set_CompareValue		R_Config_S12AD0_Set_CompareValue
	r_cg_s12ad_user.c	–	Config_S12AD0_user.c	R_Config_S12AD0_Create_UserInit
		r_s12ad0_interrupt		r_Config_S12AD0_interrupt
		r_s12ad0_compare_interrupt		r_Config_S12AD0_compare_interrupt
Serial communications interface				
	r_cg_sci.c	R_SCI6_Create	Config_SCI6.c	R_Config_SCI6_Create
		R_SCI6_Start		R_Config_SCI6_Start
		R_SCI6_Stop		R_Config_SCI6_Stop
		R_SCI6_SPI_Master_Send		R_Config_SCI6_SPI_Master_Send
		R_SCI7_Create	Config_SCI7.c	R_Config_SCI7_Create
		R_SCI7_Start		R_Config_SCI7_Start
		R_SCI7_Stop		R_Config_SCI7_Stop
		R_SCI7_Serial_Receive		R_Config_SCI7_Serial_Receive
		R_SCI7_Serial_Send		R_Config_SCI7_Serial_Send
	r_cg_sci_user.c	–	Config_SCI6_user.c	R_Config_SCI6_Create_UserInit
		r_sci6_transmit_interrupt		r_Config_SCI6_transmit_interrupt
		r_sci6_transmitend_interrupt		r_Config_SCI6_transmitend_interrupt
		r_sci6_callback_transmitend		r_Config_SCI6_callback_transmitend
		–	Config_SCI7_user.c	R_Config_SCI7_Create_UserInit
		r_sci7_transmit_interrupt		r_Config_SCI7_transmit_interrupt
		r_sci7_transmitend_interrupt		r_Config_SCI7_transmitend_interrupt
		r_sci7_receive_interrupt		r_Config_SCI7_receive_interrupt
		r_sci7_receiveerror_interrupt		r_Config_SCI7_receiveerror_interrupt
		r_sci7_callback_transmitend		r_Config_SCI7_callback_transmitend
		r_sci7_callback_receiveend		r_Config_SCI7_callback_receiveend
		r_sci7_callback_receiveerror		r_Config_SCI7_callback_receiveerror

2.7 Setting Build Options

The default build options are applied for the newly created destination project. Accordingly, build options in the source project must be reflected in the destination project.

Refer to section 4.1, Build Option Settings, in the e^2 studio Integrated Development Environment User's Manual: Getting Started Guide, and set the build options of the source project for the destination project.

3. Reference Documents

User's Manual: Hardware

The latest versions can be downloaded from the Renesas Electronics website.

Technical Update/Technical News

The latest information can be downloaded from the Renesas Electronics website.

User's Manual: Development Tools

RX Family C/C++ Compiler CC-RX User's Manual (R20UT3248)

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

e^2 studio Integrated Development Environment User's Manual: Getting Started Guide (R20UT2771)

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

Website and Support

Renesas Electronics Website

http://www.renesas.com/

Inquiries

http://www.renesas.com/contact/

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Nov 01, 2017	–	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. 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 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. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment 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 moment 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 moment 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 moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has 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. Moreover, 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.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the 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 disputes 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, drawing, chart, program, algorithm, application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or otherwise misappropriate 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, copy or otherwise misappropriation of Renesas Electronics products.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and 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.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (space and 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 third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. When using the 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 radiation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions or failure or accident arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please ensure to implement safety measures to guard them against the possibility of bodily injury, injury or damage caused by fire, and social damage in the event of 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 by your own responsibility as warranty for your products/system. Because the evaluation of microcomputer software alone is very difficult and not practical, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please investigate applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive carefully and sufficiently and use Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall not use Renesas Electronics products or technologies for (1) any purpose relating to the development, design, manufacture, use, stockpiling, etc., of weapons of mass destruction, such as nuclear weapons, chemical weapons, or biological weapons, or missiles (including unmanned aerial vehicles (UAVs)) for delivering such weapons, (2) any purpose relating to the development, design, manufacture, or use of conventional weapons, or (3) any other purpose of disturbing international peace and security, and you shall not sell, export, lease, transfer, or release Renesas Electronics products or technologies to any third party whether directly or indirectly with knowledge or reason to know that the third party or any other party will engage in the activities described above. When exporting, selling, transferring, etc., Renesas Electronics products or technologies, you shall comply with any applicable export control laws and regulations promulgated and administered by the governments of the countries asserting jurisdiction over the parties or transactions.
10. Please acknowledge and agree that you shall bear all the losses and damages which are incurred from the misuse or violation of the terms and conditions described in this document, including this notice, and hold Renesas Electronics harmless, if such misuse or violation results from your resale or making Renesas Electronics products available any third party.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.3.0-1 November 2016)

SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 6.0