

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

- RL78/G23 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

RL78 Smart Configurator User's Guide: e² studio (R20AN0579)

e² studio Integrated Development Environment User's Manual: Getting Started Guide (R20UT4819)

RL78/G13 Renesas Starter Kit Sample Code (CS+ for CC-RL) (R01AN2899)

Contents

1. Overview	3
1.1 Purpose of This Document	3
1.2 Operating Environment	3
2. Porting Projects Produced with the Code Generator to Projects for Use with the Smart Configurator.....	4
2.1 Projects Used in This Application Note	5
2.2 Downloading the Source Project	6
2.3 Generating a Report on the Source Project	7
2.3.1 Generating the Report	7
2.4 Newly Creating the Destination Project.....	10
2.5 Setting Peripheral Functions in the Smart Configurator	10
2.5.1 Correspondence between the Code Generator and the Smart Configurator	10
2.5.2 Setting the Clock Generator	11
2.5.3 Setting the Timer	18
2.5.4 Setting Other Peripheral Functions	20
2.5.5 Generating Code	20

2.6	Porting User-defined Source Code	21
2.6.1	Overview.....	21
2.6.2	Areas for Writing User-defined Source Code.....	21
2.6.3	Copying the User-created Source Files	22
2.6.4	Copying Source Code, Including the main() Function.....	25
2.6.5	Correspondences between Code Generated by the Code Generator and by the Smart Configurator	31
2.6.6	Copying Custom Code in Generated Code.....	33
2.6.7	Modifying the Include Directives.....	36
2.6.8	Modifying Parts that Call API Functions	37
2.7	Setting Build Options	39
3.	Reference Documents	40
	Revision History	41

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 for RL78/G13 to projects for use with the Smart Configurator target for same package RL78/G23 device in terms of the differences in methods of settings and in the names of functions that are generated.

For the usage of the e² studio, refer to the e² studio Integrated Development Environment User's Manual: Getting Started Guide.

1.2 Operating Environment

Table 1.1 Operating Environment

Target Device	RL78/G23 Group
Emulator	E2 lite or E2
IDE	e ² studio 2021-04 and later versions
Toolchain	Renesas C/C++ compiler package for RL78 family
Toolchain version	CC-RL78 V1.10.00

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.

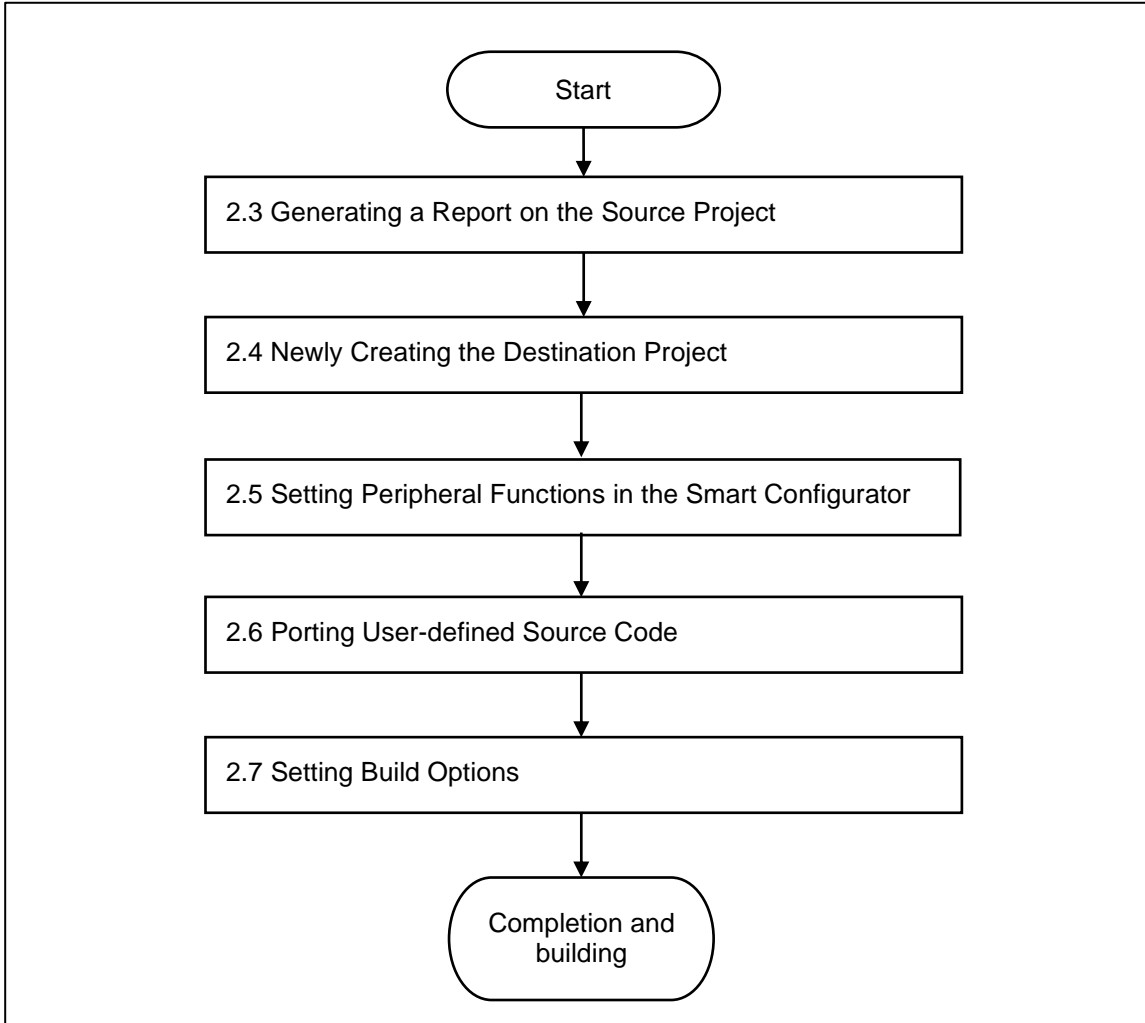


Figure 2.1 Steps in Porting Projects Produced with the Code Generator to Projects for Use with the Smart Configurator

2.1 Projects Used in This Application Note

The following two projects are used in this application note.

A project for the RL78/G13 Renesas Starter Kit Sample Code (CS+ for CC-RL), 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
RSKRL78G13_Tutorial	A project for the RL78/G13 Renesas Starter Kit Sample Code (CS+ for CC-RL), 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.
RSKRL78G23_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 RL78/G13 Renesas Starter Kit Sample Code (CS+ for CC-RL) 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>), select [RL78 Low Power 8 & 16-bit MCUs] under the [Products] menu, then select [RL78/G13] under [Portfolio] in [RL78 Low Power 8 & 16-bit MCUs] page.

The screenshot shows the Renesas website navigation menu with 'PRODUCTS' selected. Under 'PRODUCTS', 'MICROCONTROLLERS & MICROPROCESSORS' is expanded, and 'RL78 Low Power 8 & 16-bit MCUs' is highlighted with a red box. A red arrow points from this box to the 'Portfolio' section below. In the 'Portfolio' section, under 'General Purpose', the 'Standard' category is expanded, and 'RL78/G13 Standard' is highlighted with a red box. Other products shown include RL78/G23, RL78/G12, RL78/G10, RL78/G11, RL78/G13A, RL78/G1A, and RL78/G1P.

Figure 2.2 Downloading the Source Project (1)

- (2) Select [RL78/G13-Starter-Kit] from the list of [Boards & Kits] section in [RL78/G13] page.

Boards & Kits			
Part Number	Title	Type	Company
QB-R5F100LE-TB	RL78/G13 (R5F100LE) Target Board	Evaluation	Renesas
QB-R5F100SL-TB	RL78/G13 (R5F100SLAFB) Target Board	Evaluation	Renesas
RL78/G13-Starter-Kit	Renesas Starter Kit for RL78/G13	Starter	Renesas
RTK0EE0007D02001Bj	CPX3 Evaluation Kit for DC under 48V (J80D2 with RL78/G13)	Evaluation	Renesas

Figure 2.3 Downloading the Source Project (2)

- (3) Click [RL78/G13 Renesas Starter Kit Sample Code (CS+ for CC-RL) Rev.1.00 Sample Code] under [Downloads] section in the [RL78/G13-Starter-Kit] page to proceed with downloading.

Title, start typing to filter	All types	Format	File Size	Date
RSK RL78/G13 Design Data	日本語 PCB Design Files	ZIP	1.21 MB	Jan 25, 2013
RL78/G13 Group Sensirion environmental sensor module control sample software Rev.2.00 - Sample Code	日本語 Sample Code	ZIP	1.17 MB	Jul 31, 2019
RL78/G13 Renesas Starter Kit Sample Code (CS+ for CC-RL) Rev.1.00 - Sample Code	日本語 Sample Code	ZIP	2.08 MB	Aug 4, 2015
RL78/G13 Renesas Starter Kit Sample Code for Cubesuite + Toolchain Rev.1.00 - Sample Code	日本語 Sample Code	ZIP	4.44 MB	Jan 27, 2012
RL78/G13 Renesas Starter Kit Sample Code for IAR Toolchain Rev.1.00 - Sample Code	- Sample Code	ZIP	1.91 MB	Jan 27, 2012
Renesas Starter Kit for RL78/G13 Installer V.1.00 (CubeSuite+ Version)	- Software & Tools - Other	ZIP	546.41 MB	Jun 9, 2014
Renesas Starter Kit for RL78/G13 Installer V.2.00 (IAR Embedded Workbench Version)	- Software & Tools - Other	ZIP	338.02 MB	Jun 9, 2014
Product Activation Code for the RL78/L13 Renesas Starter Kit	日本語 Software & Tools - Other	ZIP	0 KB	Oct 21, 2013

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+
 - Start CS+ and open the source project [RSKRL78G13_Tutorial] that uses the Code Generator. Expand [Code Generator] under [Project Tree] and double-click on [Peripheral Functions].
 - Select [Save Code Generator Report] from the [File] menu to generate the report.

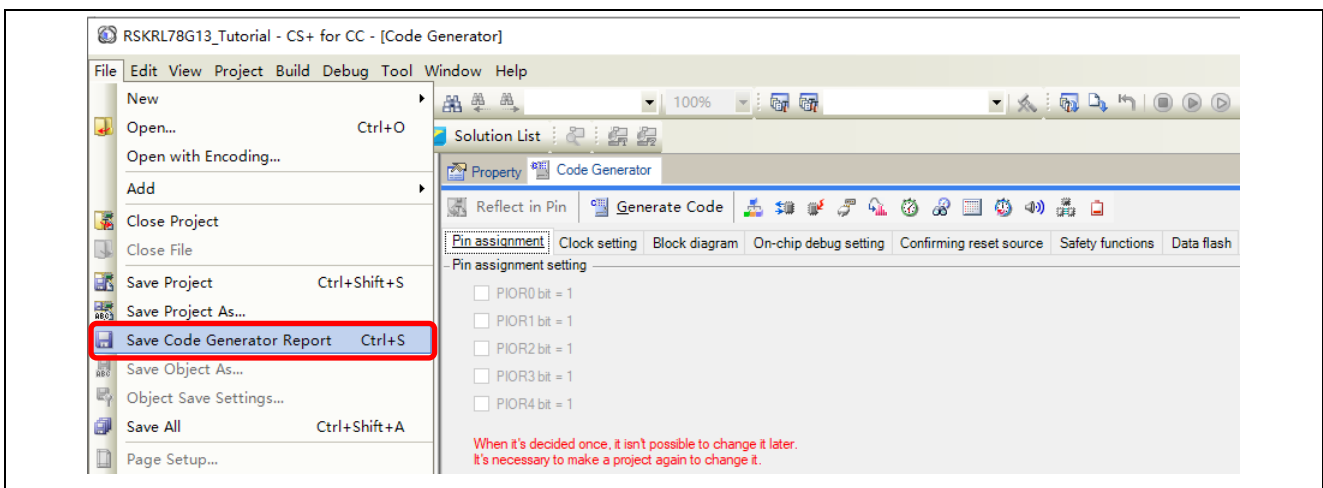


Figure 2.5 Generating the Report from CS+

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

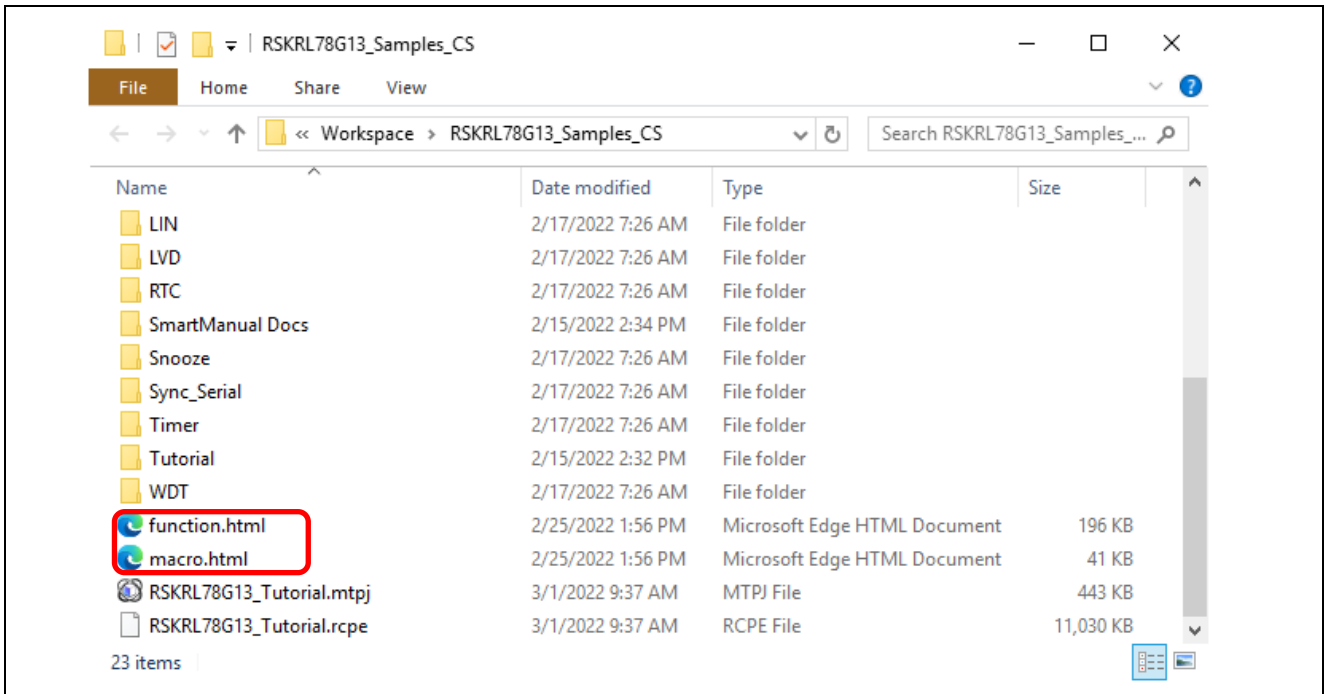


Figure 2.6 Report Files Output by the Report Function of the Code Generator for CS+

- From the e² studio
 - Start the e² studio and open the source project [RSKRL78G13_Tutorial] for which the Code Generator was used. Expand [Code Generator] under [Project Tree] and double-click on [Peripheral Functions].
 - Click on the [Generate Report] button to generate the report.

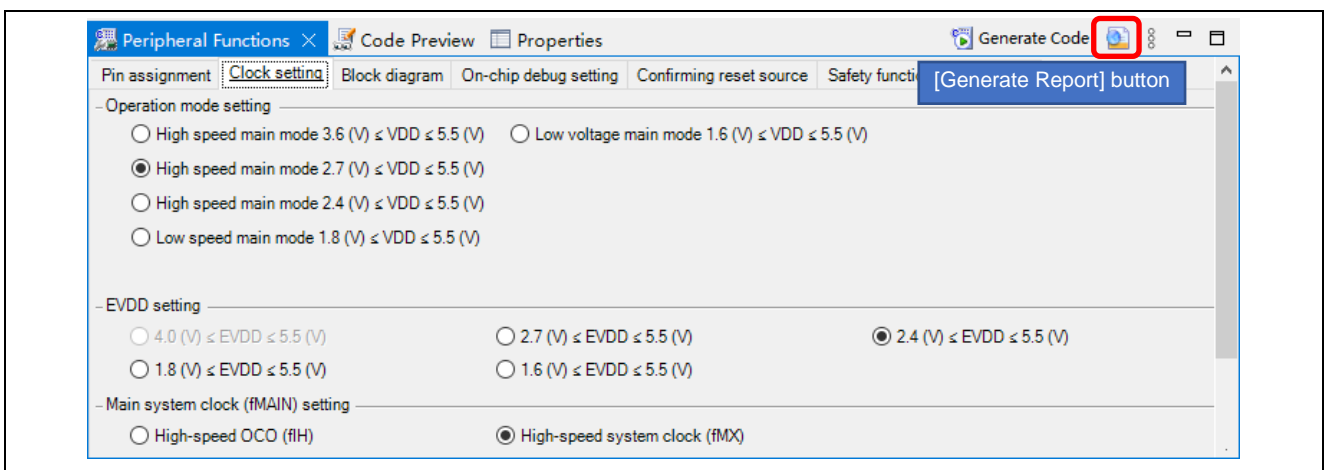


Figure 2.7 Generating a Report from the e² studio

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

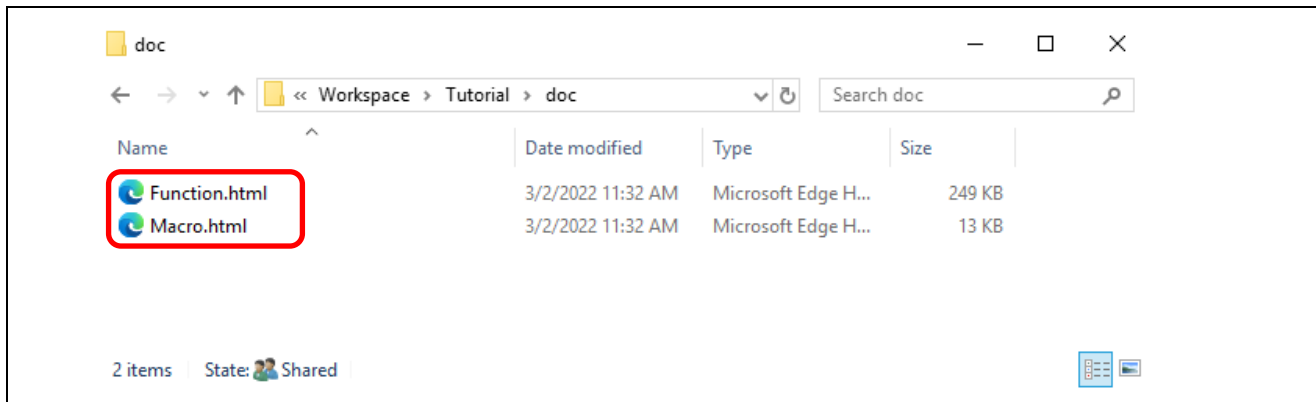


Figure 2.8 Report Files Output by the Report Function of the Code Generator for the e² studio

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 target device with R7F100GLGxFA as the destination project for use with the Smart Configurator. Regarding how to create a project, refer to section 2, Generating a C Project, in the Renesas e² 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 RSKRL78G13_Tutorial 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

Code Generator			Smart Configurator						
Peripheral functions	Setting items		Tabs	Peripheral functions	Setting items				
Port	Port5	P52	Components	Ports	PORT5	P52			
		P53				P53			
		P54				P54			
		P55				P55			
	Port6	P62			PORT6	P62			
		P63				P63			
	Port7	P70			PORT7	P70			
		P71				P71			
		P72				P72			
		P73				P73			
	Interrupt	INTP1			—	Components	Interrupt Controller	INTP1	—
		INTP2			—	Components	Interrupt Controller	INTP2	—
INTP4		—	Components	Interrupt Controller	INTP4	—			
A/D Converter	Operation mode setting	—	Components	A/D Converter	Operation mode setting	—			
	A/D channel selection	—	Components	A/D Converter	A/D channel selection	—			
Timer	Interval timer	Interval value (16 bits)	Components	Interval Timer	16 bit count mode	Interval value (16 bits)			

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 and timer. Set other peripheral functions according to the same procedure.

2.5.2 Setting the Clock Generator

Set the clock generator.

- 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: RL78/G13(ROM:64KB)				
Chip name: R5F100LE				
Module	Macro	Sub	Setting	Status
Clock Generator				Used
	CGC			Used
			Pin assignment setting-PIOR0 bit = 1	Unused
			Pin assignment setting-PIOR1 bit = 1	Unused
			Pin assignment setting-PIOR2 bit = 1	Unused
			Pin assignment setting-PIOR3 bit = 1	Unused
			Pin assignment setting-PIOR4 bit = 1	Unused
			Operation mode setting	High speed main mode $2.7 (V) \leq VDD \leq 5.5 (V)$
			EVDD setting	$2.7 (V) \leq EVDD \leq 5.5 (V)$
			Main system clock (fMAIN) setting	High-speed system clock (fMX)
			fIH operation	Unused
			fMX operation	Used
			High-speed system clock setting	X1 oscillation (fX)
			fMX frequency	20(MHz)
			Stable time	$6553.6 (2^{17}/fX)(\mu s)$
			fSUB operation	Used

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.

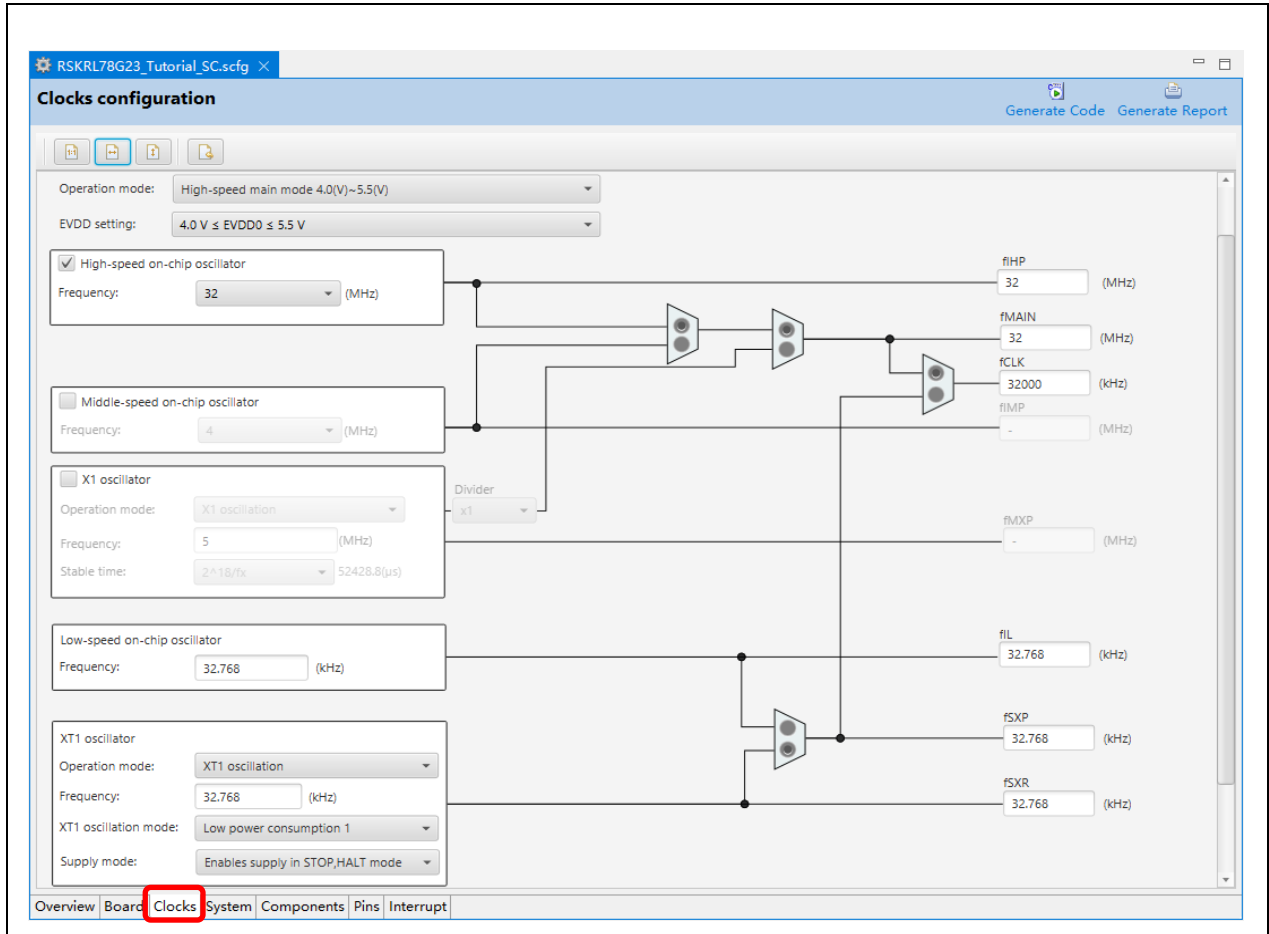


Figure 2.10 Window for Using the Smart Configurator to Make Clock Settings

(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
Pin assignment setting-PIOR0 bit = 1	Unused
Pin assignment setting-PIOR1 bit = 1	Unused
Pin assignment setting-PIOR2 bit = 1	Unused
Pin assignment setting-PIOR3 bit = 1	Unused
Pin assignment setting-PIOR4 bit = 1	Unused
Operation mode setting	High speed main mode 2.7 (V) ≤ VDD ≤ 5.5 (V) (1)
EVDD setting	2.7 (V) ≤ EVDD ≤ 5.5 (V) (2)
Main system clock (fMAIN) setting	High-speed system clock (fMX)
fIH operation	Unused (3)
fMX operation	Used (4)
High-speed system clock setting	X1 oscillation (fX) (5)
fMX frequency	20(MHz) (6)
Stable time	6553.6 (2 ¹⁷ /fX)(μs) (7)

Figure 2.11 Setting Clocks in the Smart Configurator (1)

fSUB frequency	32.768(kHz)	(8)
XT1 oscillator oscillation mode setting	Low power consumption	(9)
Subsystem clock in STOP, HALT mode setting	Enables supply	(10)
fiL Frequency	15(kHz)	
RTC and interval timer operation clock	32.768 (fSUB)(kHz)	(11)
CPU and peripheral clock (fCLK)	20000 (fMX)(kHz)	(12)

Low-speed on-chip oscillator

Frequency: 32.768 (kHz)

XT1 oscillator

Operation mode: XT1 oscillation

Frequency: 32.768 (kHz) (8)

XT1 oscillation mode: Low power consumption 1 (9)

Supply mode: Enables supply in STOP,HALT mode (10)

fMAIN: 20 (MHz)

fCLK: 20000 (kHz) (12)

fIMP: - (MHz)

fMXP: 20 (MHz)

fiL: 32.768 (kHz)

fSXP: 32.768 (kHz) (11)

fSXR: 32.768 (kHz)

Figure 2.12 Setting Clocks in the Smart Configurator (2)

(4) Select the [System] tabbed page.

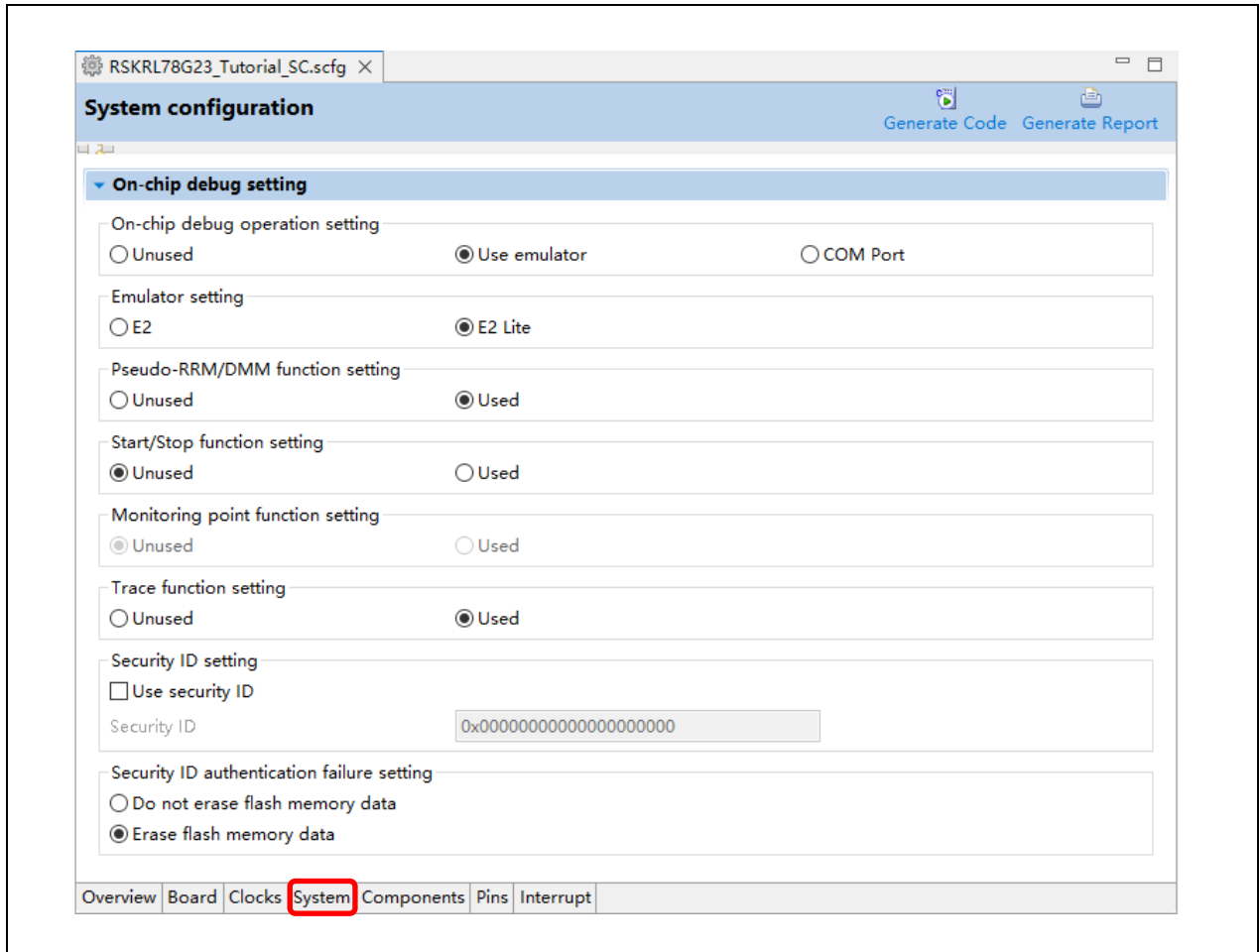


Figure 2.13 Window for Using the Smart Configurator to Make Clock Settings

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

On-chip debug operation setting	Used	(13)	On-chip debug setting
Security ID setting	Unused		On-chip debug operation setting
Security ID authentication failure setting	Erase flash memory data	(14)	<input type="radio"/> Unused <input checked="" type="radio"/> Use emulator (13) <input type="radio"/> COM Port
Emulator setting	E1/E20		Emulator setting
Pseudo-RRM/DMM function setting	Used	(15)	<input type="radio"/> E2 <input checked="" type="radio"/> E2 Lite
Start/Stop function setting	Unused	(16)	Pseudo-RRM/DMM function setting
Monitoring point function setting	Unused	(17)	<input type="radio"/> Unused <input checked="" type="radio"/> Used (15)
Output the function for confirming reset source	Used		Start/Stop function setting
Illegal memory access detection function setting	Unused		<input checked="" type="radio"/> Unused (16) <input type="radio"/> Used
RAM guard function setting	Unused		Monitoring point function setting
Port register guard function setting	Unused		<input checked="" type="radio"/> Unused (17) <input type="radio"/> Used
Interrupt register guard function setting	Unused		Trace function setting
Chip state control register guard function setting	Unused		<input type="radio"/> Unused <input checked="" type="radio"/> Used
Data flash access control setting	Disables data flash access		Security ID setting
Setting of data flash library	Unused		<input type="checkbox"/> Use security ID
			Security ID <input type="text" value="0x000000000000000000000000"/>
			Security ID authentication failure setting
			<input type="radio"/> Do not erase flash memory data
			<input checked="" type="radio"/> Erase flash memory data (14)

Figure 2.14 Setting System in the Smart Configurator

Table 2.4 Settings of the Clock Generator

	Code Generator		Smart Configurator – [Clocks] tabbed page	
	Item to be Set ([Macro] or [Setting] in Macro.html)	Setting ([Status] in Macro.html)	Item to be Set	Setting
(1)	Operation mode setting	High speed main mode $2.7\text{ (V)} \leq \text{VDD} \leq 5.5\text{ (V)}$	Operation mode	High speed main mode $2.7\text{ (V)} \sim 5.5\text{ (V)}$
(2)	EVDD setting	$2.7\text{ (V)} \leq \text{EVDD} \leq 5.5\text{ (V)}$	EVDD setting	$2.7\text{ V} \leq \text{EVDD0} \leq 5.5\text{ V}$
(3)	fIH operation	Unused	High-speed on-chip oscillator	Not selected
(4)	fMX operation	Used	Check that the fMX clock source is selected.	
(5)	High-speed system clock setting	X1 oscillation (fX)	X1 oscillator	Selected
(6)	fMX frequency	20(MHz)	[X1] Frequency	20
(7)	Stable time	$6553.6\text{ (}2^{17}/\text{fX)}\text{ (}\mu\text{s)}$	Stable time	$2^{17}/\text{fx}$
(8)	HOCO Operation	Unused	HOCO clock	Not selected
(9)	fSUB frequency	32.768 (fSUB)(kHz)	[XT1] Frequency	32.768
(10)	Subsystem clock in STOP, HALT mode setting	Enables supply	Supply mode	Enable supply in STOP, HALT mode
(11)	RTC and interval timer operation clock	32.768 (fSUB)(kHz)	Check that the fSXR clock source is selected.	
(12)	CPU and peripheral clock(fCLK)	20000 (fMX)(kHz)	Check that the fMAIN clock source is selected.	
(13)	On-chip debug operation setting	Used	On-chip debug operation setting	Used
(14)	Security ID authentication failure setting	Erase flash memory data	Security ID authentication failure setting	Erase flash memory data
(15)	Pseudo-RRM/DMM function setting	Used	Pseudo-RRM/DMM function setting	Used
(16)	Start/Stop function setting	Unused	Start/Stop function setting	Unused
(17)	Monitoring point function setting	Unused	Monitoring point function setting	Unused

2.5.3 Setting the Timer

Set the timer.

- (1) Refer to 'a-1 To add a Code Generator component' under section 4.4.2, Adding a software component into the project, in the Renesas e² 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 2.5 Correspondence between Resources and the Configuration Names of the Compare Match Timers

Component Type	Component	Resource	Configuration Name	Operation/Work Mode
Code Generator	Interval timer	TAU0_1	Config_TAU0_1 (default)	16bit count mode

- (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.

Timer				Used
	TAU0			Used
		Channel1		
			Channel 1	Interval timer
			Operation mode setting	16 bits
			Interval value (16 bits)	100ms, (Actual value: 100)
			Generates INTTM01 when counting is started	Unused
			End of timer channel 1 count, generate an interrupt (INTTM01)	Used
			Priority (INTTM01)	Low

Figure 2.15 Report on the Timer Output by the Code Generator

(3) Open the window for setting Interval timer TAU0_1 that was created in step (1).

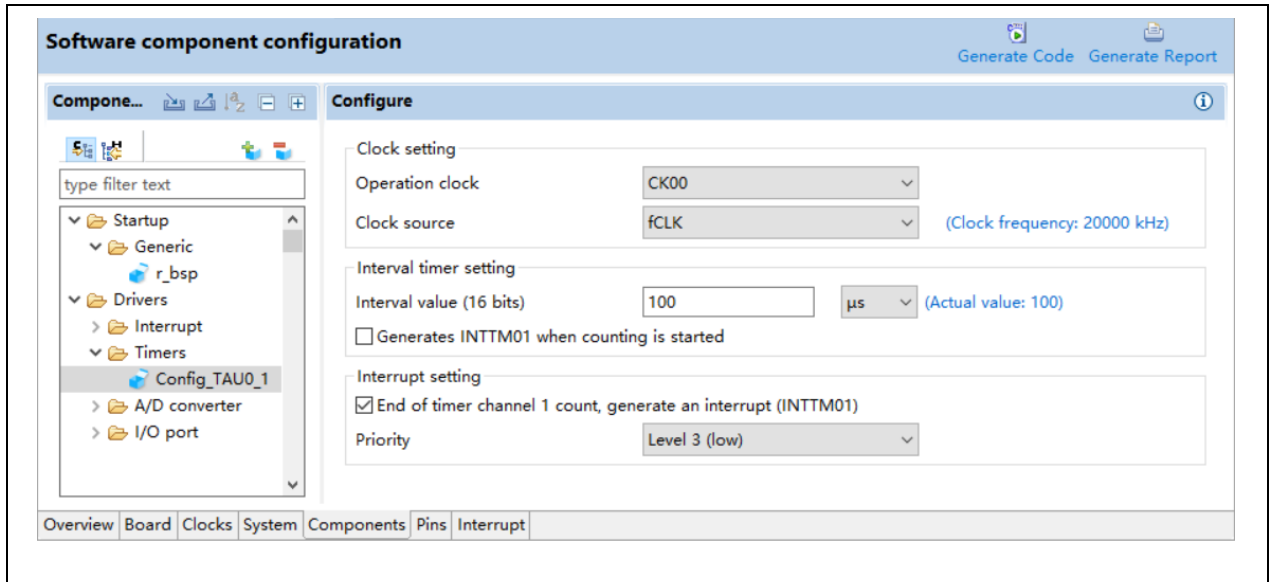


Figure 2.16 Window for Setting the Interval Timer (TAU0_1) in the Smart Configurator

(4) Reflect the settings of the Timer in Macro.html in those for TAU0 channel1 in the Smart Configurator.

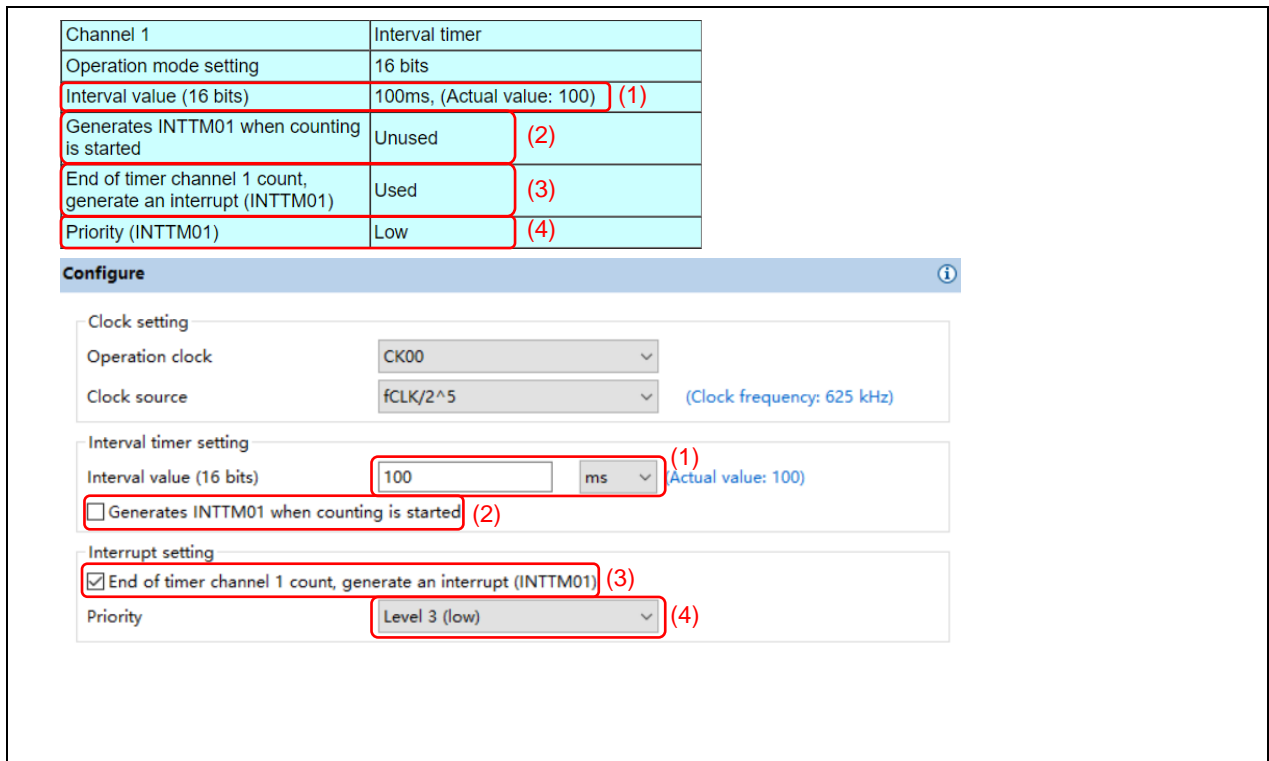


Figure 2.17 Settings of the Interval Timer (TAU0_1) in the Smart Configurator


Table 2.6 Settings of the Timer (TAU0 Channel1)

	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)	Interval value (16 bits)	100 ms	Interval value (16 bits)	100 ms
(2)	Generates INTTM01 when counting is started	Unused	Generates INTTM01 when counting is started	Unused
(3)	End of timer channel 1 count, generate an interrupt (INTTM01)	Used	End of timer channel 1 count, generate an interrupt (INTTM01)	Used
(4)	Priority (INTTM01)	Low	Priority (INTTM01)	Level3 (Low)

2.5.4 Setting Other Peripheral Functions

For settings of the PORT and A/D converter, 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 Timer, and set the Smart Configurator in the equivalent ways.

2.5.5 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 2.18 shows the procedure for porting user-defined source code.

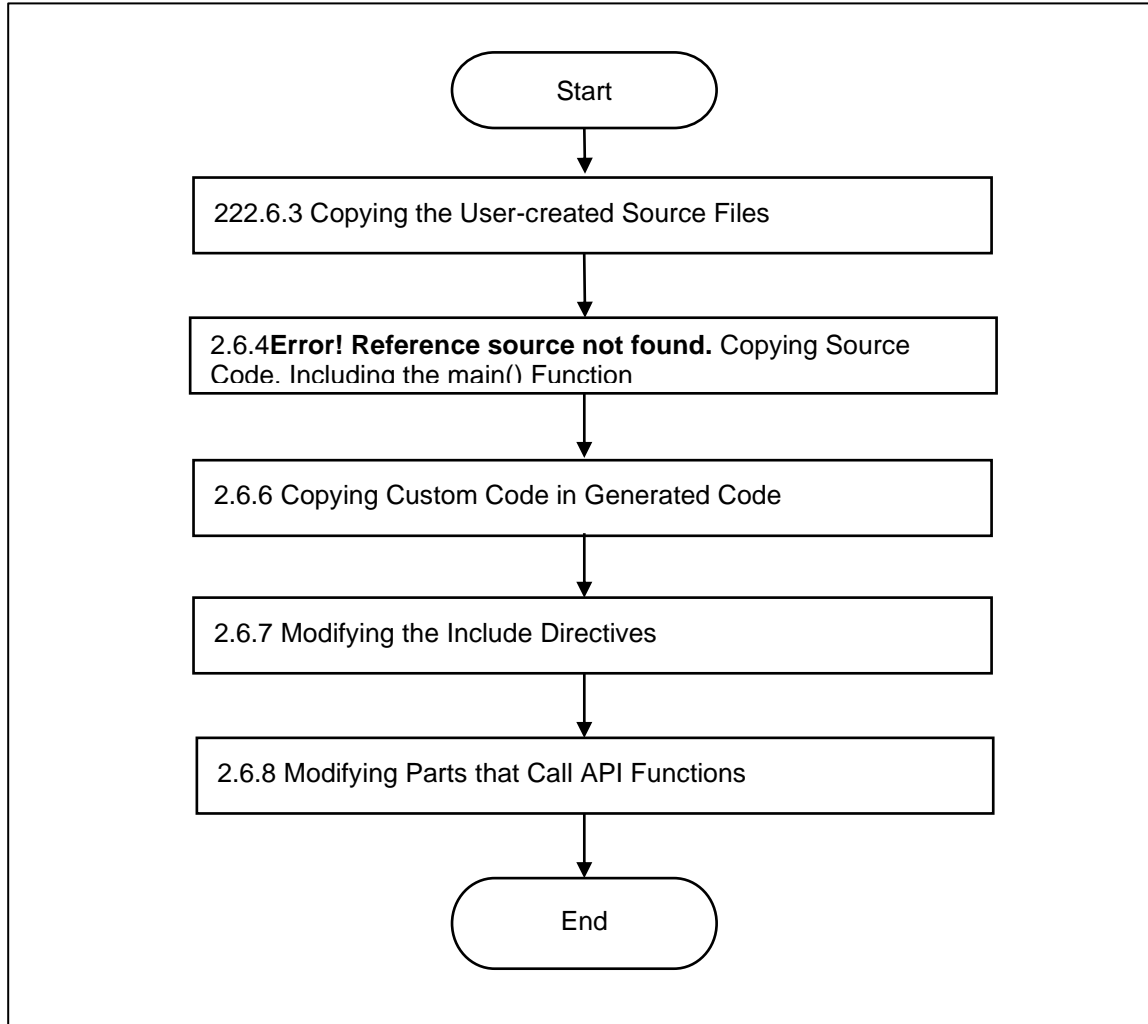


Figure 2.18 Procedure for Porting User-defined Source Code

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 specified source files and header files from the folder 'Tutorial' in the source project to the 'src' folder in the destination project.

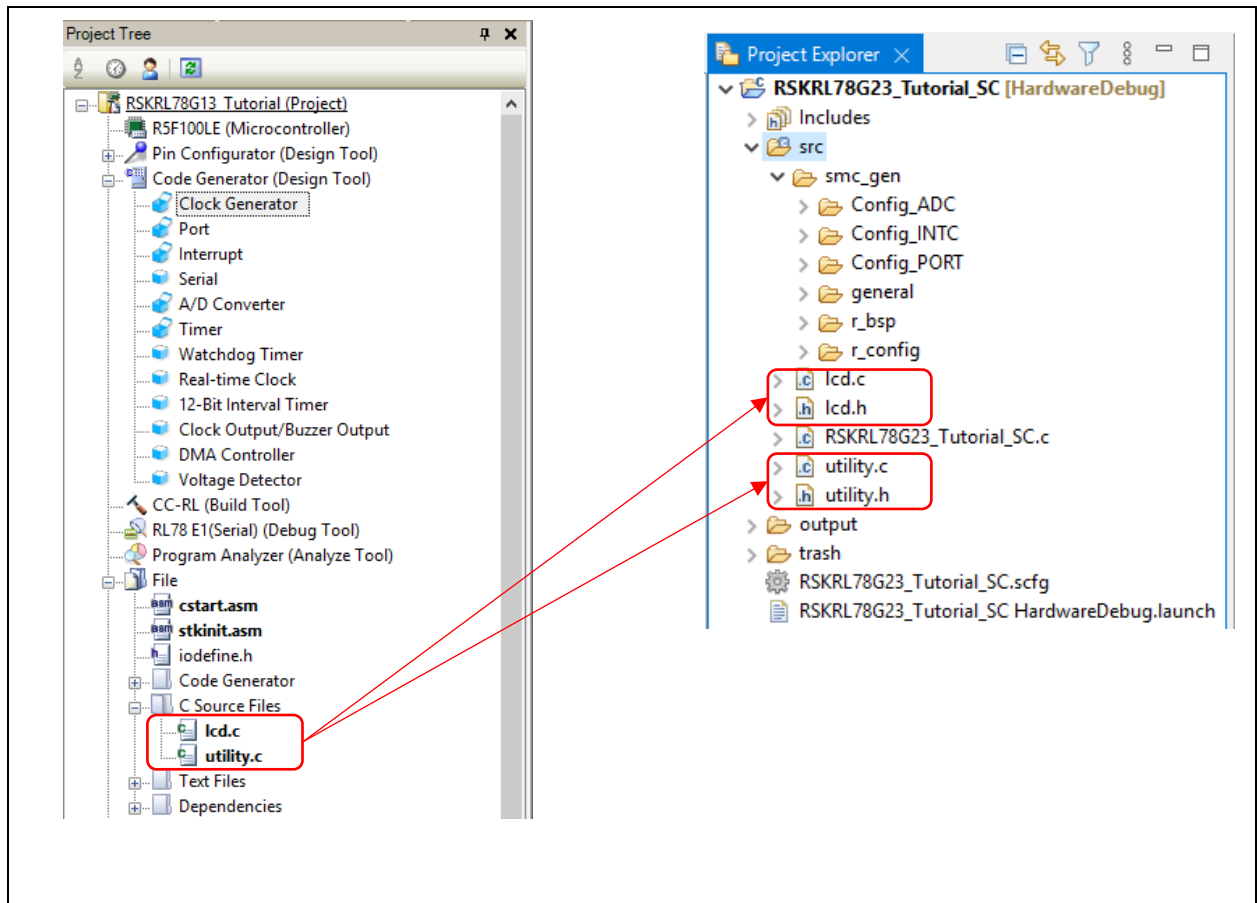


Figure 2.19 Copying the User-created Source Files

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 [RSKRL78G23_Tutorial_SC], which is the destination project, to open [Properties for RSKRL78_Tutorial_SC]. Select [Settings] under [C/C++ Project Settings] 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.

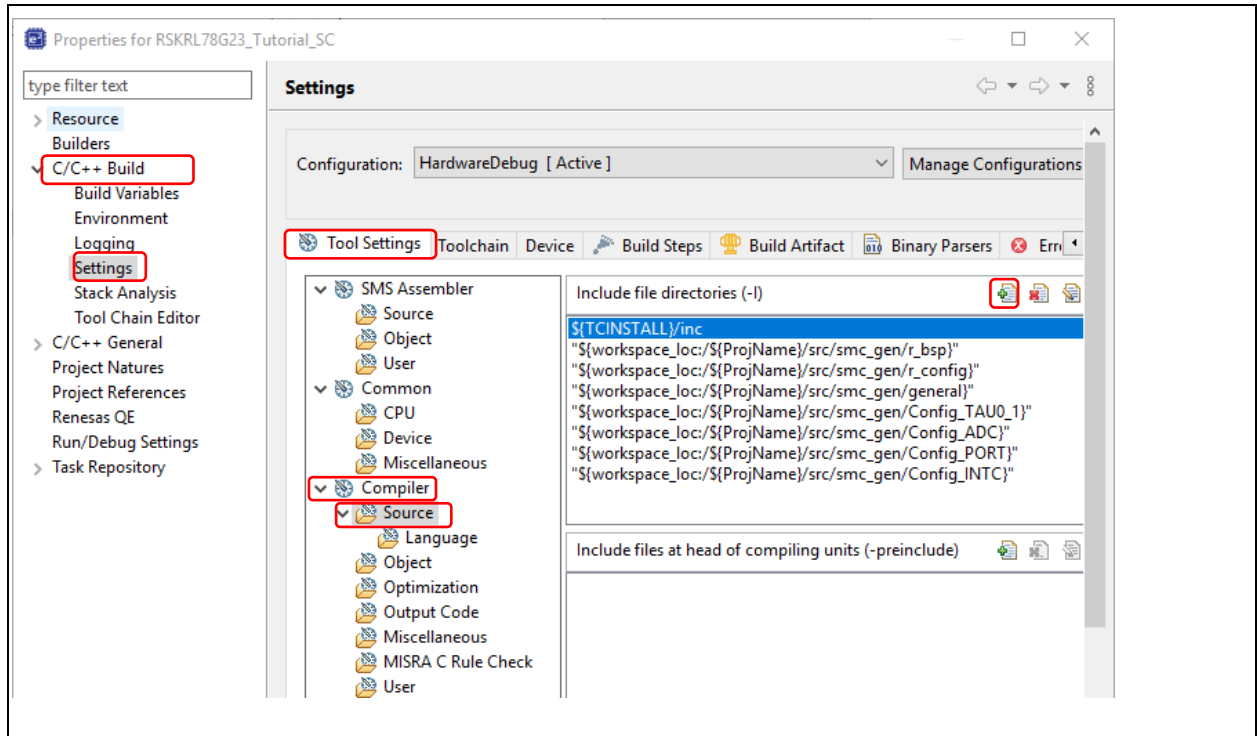


Figure 2.20 Adding the Include Directory (1)

- (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].

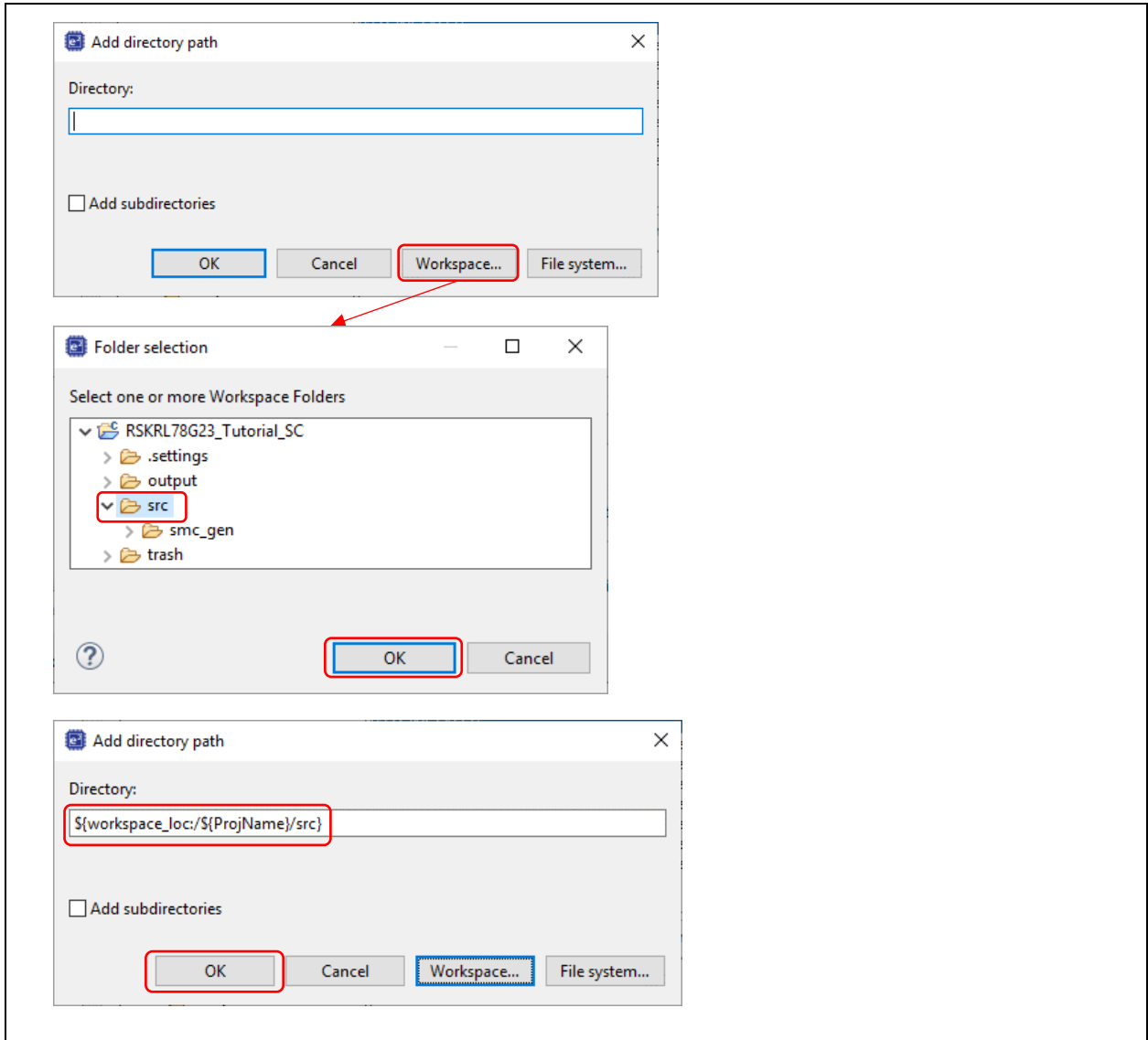


Figure 2.21 Adding the Include Directory (2)

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_main.c' file in the 'Tutorial' folder. Since 'r_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 'RSKRL78G23_Tutorial_SC' in this application note, the main() function will be in 'RSKRL78G23_Tutorial_SC.c'. All source code in '{ProjName}.c' is user-defined.

Open 'r_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_main.c' (e.g. 'r_cg_macrodriver.h' through 'r_cg_adc.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_port.h"
#include "r_cg_intc.h"
#include "r_cg_adc.h"
#include "r_cg_timer.h"
/* Start user code for include. Do not edit comment generated here */

/* Following header file provides common defines for widely used items. */
#include "rskrl78g13def.h"
/* Following header file provides useful macros and function prototypes for
controlling the LCD interface. */
#include "lcd.h"
/* Header file with integer to string conversion functions */
#include "utility.h"

/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"

```

{ProjName}.c

```

#include "r_smc_entry.h"
/* Following header file provides useful macros and function prototypes for
controlling the LCD interface. */
#include "lcd.h"
/* Header file with integer to string conversion functions */
#include "utility.h"
#include "r_cg_userdefine.h"

```

Source code written between comments of the type shown in section **Error! Reference source not found.**, 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_main.c

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

/* Define the RSK short name */
#define NICKNAME "RL78G13 "

/* Global initialised variable*/
int8_t ucStr[9]=" STATIC ";
/* Constant Data for replacement */
const int8_t ucReplace[] = "TESTTEST";
/* Global variable changed by pressing switches */
volatile int8_t gSwitchFlag = 0;

/* Static test function prototype */
void Statics_Test(void);

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

{ProjName}.c

```
int main(void);

/* Define the RSK short name */
#define NICKNAME "RL78G13 "

/* Global initialised variable*/
int8_t ucStr[9]=" STATIC ";
/* Constant Data for replacement */
const int8_t ucReplace[] = "TESTTEST";
/* Global variable changed by pressing switches */
volatile int8_t gSwitchFlag = 0;

/* Static test function prototype */
void Statics_Test(void);
```

Copy the function calls and other code in the main() function that is highlighted in yellow below.

r_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 LCD module. */
    InitialiseDisplay();

    /* Display information on the debug LCD. */
    DisplayString(LCD_LINE1, (int8_t*)"Renesas");
    DisplayString(LCD_LINE2, (int8_t*)NICKNAME);

    /* Flash the user LEDs for some time or until a push button is pressed. */
    FlashLEDs();

    /* Flash the user LEDs at a rate set by the user potentiometer (ADC) using
       interrupts. */
    TimerADC();

    /* Demonstration of initialised variables. Use this function with the
       debugger.*/
    Statics_Test();

    /* Halt program in an infinite while loop */
    while (1U)
    {
        ;
    }

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

```

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.

{ProjName}.c

```
int main(void)
{
    EI();
    /* Initialise the LCD module. */
    InitialiseDisplay();

    /* Display information on the debug LCD. */
    DisplayString(LCD_LINE1, (int8_t*)"Renesas");
    DisplayString(LCD_LINE2, (int8_t*)NICKNAME);

    /* Flash the user LEDs for some time or until a push button is pressed. */
    FlashLEDs();

    /* Flash the user LEDs at a rate set by the user potentiometer (ADC) using
    interrupts. */
    TimerADC();

    /* Demonstration of initialised variables. Use this function with the
    debugger.*/
    Statics_Test();
    return 0;
}
```

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_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 */

    EI();

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

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

void Statics_Test(void)
{
    /* Declare loop count variable */
    uint8_t uiCount;
    /* Declare string variable to hold the string to be copied */
    uint8_t ucStr[] = "STATIC \0";
    /* Declare variable buffer to store the copied string */
    uint8_t ucReplace[] = "TESTTEST\0";

    /* Declare a delay count variable */
    uint32_t ulDelay;

    /* Write ucStr variable, "STATIC" to LCD */
    DisplayString(LCD_LINE2, (int8_t*)ucStr);

    /* Begin for loop which writes one letter of ucReplace to the LCD at a time
    The nested while loops generate the delay bewteen each letter change */
    for (uiCount=0; uiCount<8; uiCount++)
    {
        /* Replace letter number uiCount of ucStr from ucReplace */
        ucStr[uiCount] = ucReplace[uiCount];

        /* Display the character on the debug LCD */
        DisplayString(LCD_LINE2, (int8_t*)ucStr);

        /* LED Flashing Delay */
        for (ulDelay=0; ulDelay<700000; ulDelay++)
        {
            /* Delay */
        }
    }

    /* Clear LCD Display */
    ucStr[uiCount] = '\0';

    /* Write MCU nickname to LCD again */
    DisplayString(LCD_LINE2, (int8_t*)NICKNAME);
}
/*****
End of function Statics_Test
*****/

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

```

{ProjName}.c

```

void Statics_Test(void)
{
    /* Declare loop count variable */
    uint8_t uiCount;
    /* Declare string variable to hold the string to be copied */
    uint8_t ucStr[] = "STATIC \0";
    /* Declare variable buffer to store the copied string */
    uint8_t ucReplace[] = "TESTTEST\0";

    /* Declare a delay count variable */
    uint32_t ulDelay;

    /* Write ucStr variable, "STATIC" to LCD */
    DisplayString(LCD_LINE2, (int8_t*)ucStr);

    /* Begin for loop which writes one letter of ucReplace to the LCD at a time
    The nested while loops generate the delay bewteen each letter change */
    for (uiCount=0; uiCount<8; uiCount++)
    {
        /* Replace letter number uiCount of ucStr from ucReplace */
        ucStr[uiCount] = ucReplace[uiCount];

        /* Display the character on the debug LCD */
        DisplayString(LCD_LINE2, (int8_t*)ucStr);

        /* LED Flashing Delay */
        for (ulDelay=0; ulDelay<700000; ulDelay++)
        {
            /* Delay */
        }
    }

    /* Clear LCD Display */
    ucStr[uiCount] = '\0';

    /* Write MCU nickname to LCD again */
    DisplayString(LCD_LINE2, (int8_t*)NICKNAME);
}
/*****
End of function Statics_Test
*****/

```

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 2.7 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	
Tutorial	r_main.c	src	{ProjName}.c	File that contains main().
Tutorial	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.
Tutorial	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 RSKRL78G13_Tutorial 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 2.8 Files that Require Porting of Custom Code in the RSKRL78G13_Tutorial Sample Code

Peripheral Function	Code Generator		Smart Configurator	
	Output Folder	Source File	Output Folder	Source File
File including main()	Tutorial	r_main.c	src	{ProjName}.c
General settings	Tutorial	r_cg_userdefine.h	src\smc_gen\general	r_cg_userdefine.h
Interrupt	Tutorial	r_cg_intc.c	src\smc_gen\Config_INTC	Config_INTC.c
		r_cg_intc_user.c	src\smc_gen\Config_INTC	Config_INTC_user.c
		r_cg_intc.h	src\smc_gen\general	r_cg_intc.h
			src\smc_gen\Config_INTC	Config_INTC.h
PORT	Tutorial	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
Timer	Tutorial	r_cg_timer.c	src\smc_gen\Config_TAU0_1	Config_TAU0_1.c
			src\smc_gen\general	r_cg_tau_common.c
		r_cg_timer_user.c	src\smc_gen\Config_TAU0_1	Config_TAU0_1_user.c
		r_cg_timer.h	src\smc_gen\general	r_cg_tau_common.h
			src\smc_gen\general	r_cg_tau.h
src\smc_gen\Config_TAU0_1	Config_TAU0_1.h			
A/D converter	Tutorial	r_cg_adc.c	src\smc_gen\Config_ADC	r_cg_ad_common.c
			src\smc_gen\Config_ADC	Config_ADC.c
		r_cg_adc_user.c	src\smc_gen\Config_ADC	Config_ADC_user.c
		r_cg_adc.h	src\smc_gen\general	r_cg_ad_common.h
			src\smc_gen\general	r_cg_ad.h
src\smc_gen\Config_ADC	Config_ADC.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 2.8, taking the case of the Timer (in use for interval timer) as an example.

Firstly, copy the custom code for Timer that is highlighted in yellow below from 'r_cg_timer.h' to 'Config_TAU0_1.h'.

r_cg_timer.h

```
/* Start user code for function. Do not edit comment generated here */  
/* Declare TimerADC function prototype */  
void TimerADC(void);  
/* End user code. Do not edit comment generated here */
```

Config_TAU0_1.h

```
/* Start user code for function. Do not edit comment generated here */  
/* Declare TimerADC function prototype */  
void TimerADC(void);  
/* End user code. Do not edit comment generated here */
```

After that, copy the custom code for Timer that is highlighted in yellow below from 'r_cg_timer_user.c' to 'Config_TAU0_1_user.c'.

r_cg_timer_user.c

```

/* Start user code for include. Do not edit comment generated here */
#include "r_cg_adc.h"
/* rskrl78gl3def.h provides common defines for widely used items. */
#include "rskrl78gl3def.h"
/* Following header file provides function prototypes for LCD controlling
functions & macro defines */
#include "lcd.h"
/* Header file with integer to string conversion functions */
#include "utility.h"
/* End user code. Do not edit comment generated here */

/* Start user code for global. Do not edit comment generated here */
/* Declare a variable for storing the ADC value */
volatile uint16_t gTimerADCperiod = 0;
/* End user code. Do not edit comment generated here */ -

static void __near r_tau0_channell_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Toggle user LEDs */
    LED0 = ~LED0;
    LED1 = ~LED1;
    LED2 = ~LED2;
    LED3 = ~LED3;

    /* Store the ADC value into the lower 12 bits of the variable */
    gTimerADCperiod = ADCR >> 6;

    /* Ensure that the timer period is never set below 0x75 */
    if(gTimerADCperiod < 0x0075)
    {
        gTimerADCperiod = 0x0075;
    }

    /* Update timer period with respect to adc value */
    TDR01 = gTimerADCperiod * 58;

    /* Clear TM01 interrupt flag */
    TMIF01 = 0;
    /* End user code. Do not edit comment generated here */
}

/* Start user code for adding. Do not edit comment generated here */
/*****
* Function Name: TimerADC
* Description : Uses the ADC to change the duty of the timer, used to flash
the LEDs.
* Arguments : None
* Return Value : None
*****/
void TimerADC(void)
{
    /* Start ADC operations */
    R_ADC_Start();
    /* Start timer TM01 operations */
    R_TAU0_Channell_Start();
}
/*****
End of function TimerADC
*****/
/* End user code. Do not edit comment generated here */

```

Config_TAU0_1_user.c

```

/* Start user code for include. Do not edit comment generated here */
#include "Config_ADC.h"
/* rskrl78g13def.h provides common defines for widely used items. */
#include "rskrl78g13def.h"
/* Following header file provides function prototypes for LCD controlling
functions & macro defines */
#include "lcd.h"
/* Header file with integer to string conversion functions */
#include "utility.h"
/* End user code. Do not edit comment generated here */

/* Start user code for global. Do not edit comment generated here */
/* Declare a variable for storing the ADC value */
volatile uint16_t gTimerADCperiod = 0;
/* End user code. Do not edit comment generated here */
static void __near r_Config_TAU0_1_interrupt(void)
{
    /* Start user code for r_Config_TAU0_1_interrupt. Do not edit comment
generated here */
    /* Toggle user LEDs */
    LED0 = ~LED0;
    LED1 = ~LED1;
    LED2 = ~LED2;
    LED3 = ~LED3;

    /* Store the ADC value into the lower 12 bits of the variable */
    gTimerADCperiod = ADCR >> 6;

    /* Ensure that the timer period is never set below 0x75 */
    if(gTimerADCperiod < 0x0075)
    {
        gTimerADCperiod = 0x0075;
    }

    /* Update timer period with respect to adc value */
    TDR01 = gTimerADCperiod * 58;

    /* Clear TM01 interrupt flag */
    TMIF01 = 0;
    /* End user code. Do not edit comment generated here */
}
/* Start user code for adding. Do not edit comment generated here */

/*****
* Function Name: TimerADC
* Description : Uses the ADC to change the duty of the timer, used to flash the
LEDs.
* Arguments : None
* Return Value : None
*****/
void TimerADC(void)
{
    /* Start ADC operations */
    R_Config_ADC_Start();

    /* Start timer TM01 operations */
    R_Config_TAU0_1_Start();
}
- Omitted -

```

Since 'R_Config_ADC_Start', 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 **Error! Reference source not found.**, Modifying Parts that Call API Functions.

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, find files that include 'r_cg_adc.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_adc.h"' in the source project, the results are as follows.

Tutorial	
—	r_systeminit.c
—	r_main.c
—	r_cg_adc.c
—	r_cg_adc_user.c
—	r_cg_timer_user.c

Among these files that contain '#include "r_cg_adc.h"', since files other than 'r_main.c' ({ProjName}.c for the destination project) and 'r_cg_timer_user.c' (Config_TAU0_1_user.c for the destination project) include appropriate header files from the Smart Configurator, the include directives need not be modified.

For 'r_cg.timer_user.c' (Config_TAU0_1_user.c for the destination project), the include directives in the corresponding source files of the destination project require modification.

- For the source file (Config_TAU0_1_user.c)

Open 'Config_TAU0_1_user.c' in the destination project and find the part where the ADC function is called. In 'Config_TAU0_1_user.c', the following function is called.

— R_Config_ADC_Start()

Since the prototype declarations are in 'Config_ADC.h', the directives are modified so that this header file is included.

<p>r_cg_timer_user.c (before modification)</p> <pre>/* Start user code for include. Do not edit comment generated here */ #include "r_cg_adc.h"</pre>
<p>Config_TAU0_1_user.c (after modification)</p> <pre>/* Start user code for include. Do not edit comment generated here */ #include "Config_ADC.h"</pre>

2.6.8 Modifying Parts that Call API Functions

Parts of the custom code copied in section 2.6.6, Copying Custom Code in Generated Code, 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 custom code in 'r_cg_timer_user.c' was copied in accordance with section 2.6.6, Copying Custom Code in Generated Code. The definition of the TimerADC() function is in the file.

Example.

Two API functions: R_ADC_Start() and R_TAU0_Channel1_Start() generated by the Code Generator are called in 'TimerADC' before modification. Since these function are R_Config_ADC_Start() and R_Config_TAU0_1_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_cg_timer_user.c (before modification)

```
void TimerADC(void)
{
    /* Start ADC operations */
    R_ADC_Start();

    /* Start timer TM01 operations */
    R_TAU0_Channel1_Start();
}
```

Config_TAU0_1_user.c (after modification)

```
void TimerADC(void)
{
    /* Start ADC operations */
    R_Config_ADC_Start();

    /* Start timer TM01 operations */
    R_Config_TAU0_1_Start();
}
```

Table 2.9 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 2.9 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 - e² studio] - [e² studio User Guide] - [Building Projects] - [Smart Configurator] - [API reference].

Table 2.9 Correspondences between the Names of API Functions Generated by the Code Generator and by the Smart Configurator

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	mcu_clocks.c	mcu_clock_setup
Timer			
r_cg_timer.c	R_TAU0_Create	Config_TAU0_1.c	R_Config_TAU0_1_Create
	R_TAU0_Channel1_Start		R_Config_TAU0_1_Start
	R_TAU0_Channel1_Stop		R_Config_TAU0_1_Stop
r_cg_timer_user.c	–	Config_TAU0_1_user.c	R_Config_TAU0_1_Create_UserInit
	r_tau0_channel1_interrupt		r_tau0_channel1_interrupt
Interrupt			
r_cg_intc.c	R_INTC_Create	Config_INTC.c	R_Config_INTC_Create
	R_INTC1_Start		R_Config_INTC_INTP1_Start
	R_INTC1_Stop		R_Config_INTC_INTP1_Stop
	R_INTC2_Start		R_Config_INTC_INTP2_Start
	R_INTC2_Stop		R_Config_INTC_INTP2_Stop
	R_INTC4_Start		R_Config_INTC_INTP2_Start
	R_INTC4_Stop		R_Config_INTC_INTP2_Stop
r_cg_intc_user.c	–	Config_INTC_user.c	R_Config_ICU_Create_UserInit
	r_intc1_interrupt		r_Config_INTC_intp1_interrupt
	r_intc2_interrupt		r_Config_INTC_intp2_interrupt
	r_intc4_interrupt		r_Config_INTC_intp4_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
A/D converter			
r_cg_adc.c	R_ADC_Create	Config_ADC.c	R_Config_ADC_Create
	R_ADC_Start		R_Config_ADC_Start
	R_ADC_Stop		R_Config_ADC_Stop
	R_ADC_Set_OperationOn		R_Config_ADC_Set_OperationOn
	R_ADC_Set_OperationOff		R_Config_ADC_Set_OperationOff
	R_ADC_Get_Result		R_Config_ADC_Get_Result_10bit
	–		R_Config_ADC_Set_SnoozeOn
	–		R_Config_ADC_Set_SnoozeOff
	–		R_Config_ADC_Set_ADChannel
	–		R_Config_ADC_Set_TestChannel
r_cg_adc_user.c	–	Config_ADC_user.c	R_Config_ADC_Create_UserInit
	r_adc_interrupt		r_Config_ADC_interrupt

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 e² 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

CC-RL Compiler User's Manual (R20UT3123)

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

e² studio Integrated Development Environment User's Manual: Getting Started Guide (R20UT4819)

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/>

All trademarks and registered trademarks are the property of their respective owners.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Apr 5, 2022	–	First edition issued

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

2. Processing at power-on

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

3. Input of signal during power-off state

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

4. Handling of unused pins

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

5. Clock signals

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

6. Voltage application waveform at input pin

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

7. Prohibition of access to reserved addresses

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

8. Differences between products

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

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
 - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
 - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
7. Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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