Smart Configurator Usage for RL78 LoRa®-based Wireless Software

Reference Guide

Introduction

This application note describes the information on how to use RL78 Smart Configurator (hereafter called the Smart Configurator) for LoRa®-based wireless software (hereafter called the LoRa®-based software).

The LoRa®-based software packages contain the source code of peripheral modules. The source code has been generated according to the configurations for the use with RL78 Fast Prototyping Board by the Smart Configurator.

This application note describes how to generate the source code of the peripheral modules by using the Smart Configurator and how to change the resources of the peripheral modules according to the user hardware system.

Note: Users can choose to generate the source code by using the Smart Configurator or change the source code without using Smart Configurator when to port the source code for user applications.

Target Device

MCU: Renesas RL78/G23 (R7F100GSN, R7F100GLG), RL78/G22 (R7F102GGE), RL78/L23 (R7F100LPL)
Transceiver: Semtech SX1261 or SX1262
Sensor: Renesas HS3001

Tool

RL78 Smart Configurator Ver.1.15.0
e2 studio 2025-10 / CS+ for CC V8.14.00 with CC-RL V1.15.01 tool chain

Contents

1. Overview

This application note contains information on how to use the Smart Configurator for the LoRa®-based software.

The LoRa®-based software packages contain the source code of the peripheral modules. The source code has been generated according to the configurations for the use with RL78 Fast Prototyping Board by the Smart Configurator.

The chapter 2 describes how to use the Smart Configurator from e2 studio project and CS+ project. The chapter 3 describes how to generate the source code of peripheral modules required for the LoRa®-based software by using the Smart Configurator including the peripheral modules settings and the generated source code changes. The chapter 4 describes how to change the resources of the peripheral modules when to port the source code for user applications.

Please also refer to the user's guide for the detail of the general usage of the Smart Configurator ([1], [2] and [3]).

1.1 Related Documentation

Table 1 Related Documentation

	Document No.	Title	Author	Language
[1]	R20AN0579	RL78 Smart Configurator User' Guide: e2 studio	Renesas Electronics	English, Japanese
[2]	R20AN0580	RL78 Smart Configurator User' Guide: CS+	Renesas Electronics	English, Japanese
[3]	R20UT4852	Smart Configurator User's Manual: RL78 API Reference	Renesas Electronics	English, Japanese
[4]	R11AN0227	Radio Driver Reference Guide	Renesas Electronics	English
[5]	R11AN0834	Radio Driver Support Functions for Regional Radio Regulations	Renesas Electronics	English
[6]	R11AN0230	Radio Evaluation Program Commands Reference	Renesas Electronics	English
[7]	R11AN0595	RL78/G23, RL78/G22, RL78/L23, RL78/G14 LoRa®-based Wireless Software Package	Renesas Electronics	English
[8]	R11AN0228	LoRaWAN® Stack Reference Guide	Renesas Electronics	English
[9]	R11AN0231	LoRaWAN® Stack Sample Application	Renesas Electronics	English
[10]	R11AN0829	Private LoRa® Stack Reference Guide	Renesas Electronics	English
[11]	R11AN0830	Private LoRa® Stack Sample Application	Renesas Electronics	English

2. Getting Started

2.1 Using Smart Configurator from e2 studio Project

Smart Configurator for RL78 is equivalent to Smart Configurator for RL78 Plug-in in e2 studio.
Please refer to [1] for the details of starting the Smart Configurator.

2.1.1 Displaying Smart Configurator Perspective

To fully utilize Smart Configurator features, ensure that the Smart Configurator perspective is opened. If it is not opened, select the perspective icon in the upper right corner of the e2 studio window.

2.1.2 Starting Smart Configurator

Double-click on the Smart Configurator project file (*.scfg) file under the project tree of e2 studio and activate the Smart Configurator perspective.

The Smart Configurator loads the *.scfg file, which is at the same level as the project of e2 studio. The Smart Configurator saves the setting information such as the target MCU, build tool, peripheral modules, and pin functions in the *.scfg file.

2.2 Using Smart Configurator from CS+ Project

Please refer to [2] for the detail of starting the Smart Configurator.

2.2.1 Installing Smart Configurator

Download the RL78 Smart Configurator and CS+ RL78 Smart Configurator Communication plug-in from the URL below. The CS+ RL78 Smart Configurator communication plug-in is required for registering source code generated by the Smart Configurator with CS+.

https://www.renesas.com/rl78-smart-configurator

After activating the installer, install the Smart Configurator and the plug-in by following the procedure of the installer. You will require administrator privileges to do this.

2.2.2 Setting CS+ Integrated Development Environment

Select [Plug-in Manager] from [Tool] of CS+ menu and confirm that there is a tick against "Smart Configurator for RL78 Communication Plug-in". Tick it if it is not.

2.2.3 Starting Smart Configurator

Double-click on [Smart Configurator (Design Tool)] under [Project name (Project)] in the Project Tree of CS+ to start the Smart Configurator.

When the Smart Configurator is activated from CS+, the Smart Configurator loads the Smart Configurator project file (*.scfg), which is at the same level as the project file (*.mtpj) of CS+. The Smart Configurator saves the setting information such as the target MCU, build tool, peripheral modules, and pin functions in the *.scfg file.

3. Setting of Peripheral Modules for LoRa®-based Software

This section describes the setting of peripheral modules required for the LoRa®-based software.

The peripheral modules can be configured in the Smart Configurator. After that, the source files of the peripheral modules can be generated by clicking on the [Generate Code] button in the Smart Configurator view. The source files will be stored in `<ProjectDir>\src\smc_gen`, and the source file list in the project tree will be updated.
Some of the code in the generated source files need to be changed for the LoRa®-based software, where the key words `/* Start user code */` and `/* End user code */` should be added before and after the changed code to prevent from being overwritten by the Smart Configurator.
Please refer to [1] and/or [2] for details.

Note: The LoRa®-based software packages contain all the necessary configurations and the source code changes by default.

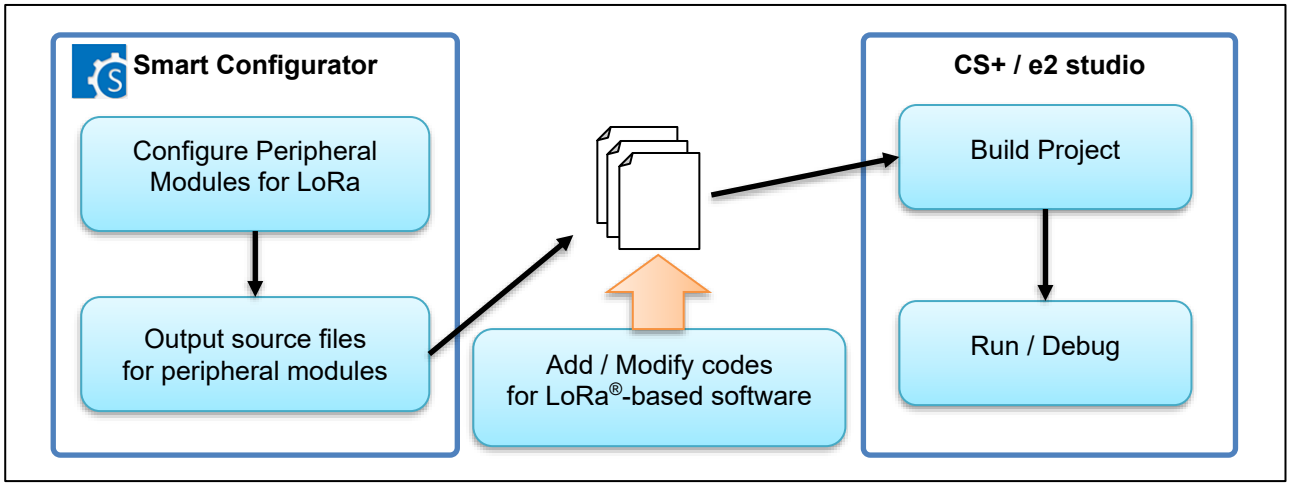


Figure 1 Flow from Configure the Peripheral Modules to Run

3.1 Board Settings

When the Smart Configurator is started for the first time using .scfg file for the LoRa®-based software, [Board] item in [Board] page may be set to [Custom User Board] and warning message (M02000004) is displayed in console view.

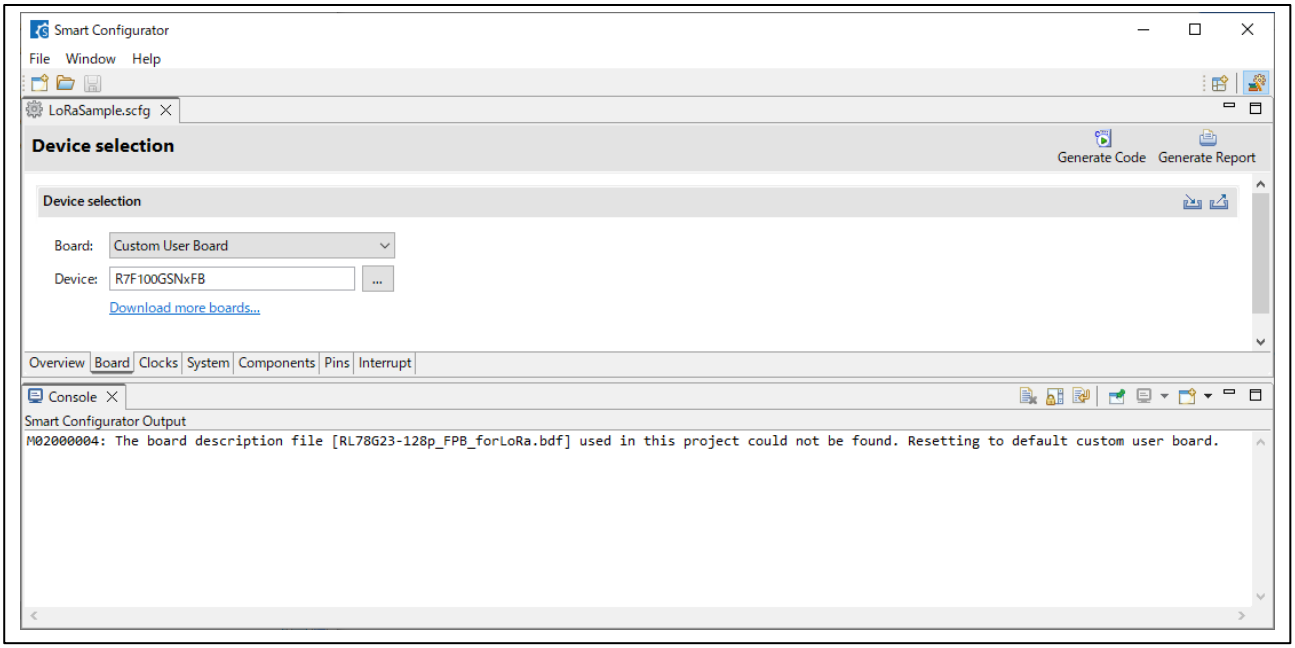


Figure 2 Board Setting and Warning Message at the First Time

In this case, it is necessary to import board setting (.bdf file) for the LoRa®-based software. Refer [1] and/or [2] to import board settings.

Following .bdf files are for RL78 Fast Prototyping Board.

Folder	(package top)\samples\project\	
.bdf File	- RL78G23-128p_FPB_forLoRa.bdf	(for RL78/G23-128p Fast Prototyping Board)
	- RL78G23-64p_FPB_forLoRa.bdf	(for RL78/G23-64p Fast Prototyping Board)
	- RL78G22_FPB_forLoRa.bdf	(for RL78/G22 Fast Prototyping Board)
	- RL78L23_FPB_forLoRa.bdf	(for RL78/L23 Fast Prototyping Board)

3.2 Clock Settings

System clock can be set on the [Clocks] page. Clock Settings for RL78/G23, RL78/G22 and RL78/L23 are shown in Table 2, Figure 3, and Figure 4.

Table 2 Clock Settings

Item	**Setting**	**Note**
Operation mode	Low-speed main mode 1.8(V) ~ 5.5(V)	
EVDD setting	1.8V ≤ EVDD0 ≤ 5.5V	RL78/G23 only
High-speed on-chip oscillator	ON	
Frequency	8 (MHz)	
fHOCO start setting	Normal	
Middle-speed on-chip oscillator	OFF	
Frequency	-	
X1 oscillator	OFF	
Operation mode	-	
Frequency	-	
Stable time	-	
Low-speed on-chip oscillator	-	
Frequency	32.768 (kHz)	
XT1 oscillator	ON	
Operation mode	XT1 oscillation	
Frequency	32.768 (kHz)	

XT1 oscillation mode	Low power consumption 3	
Supply mode	Enables supply in STOP, HALT mode	
Oscillator margin checking	Normal state	RL78/L23 only

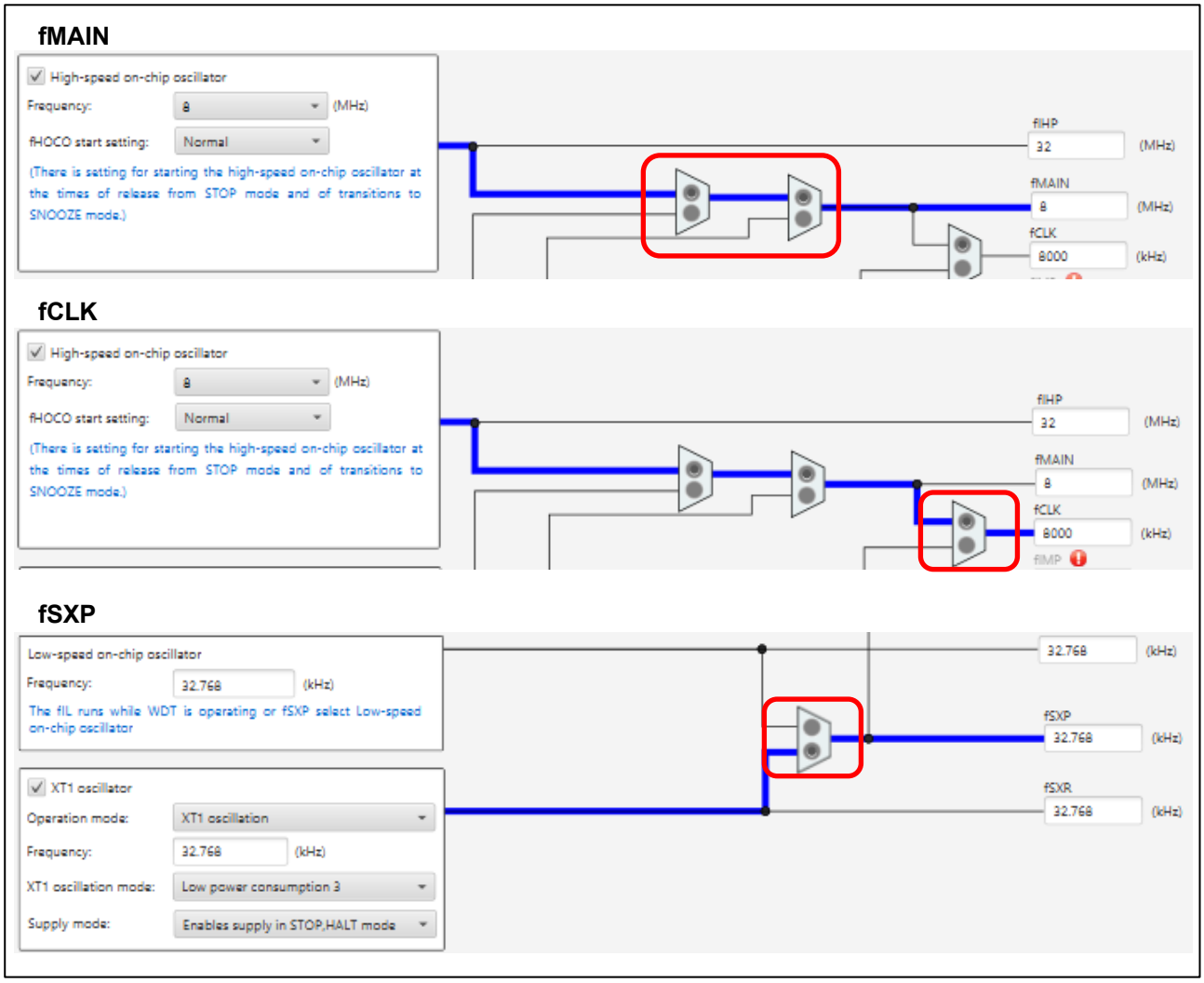


Figure 3 Clock source selection (RL78/G22 and RL78/G23)

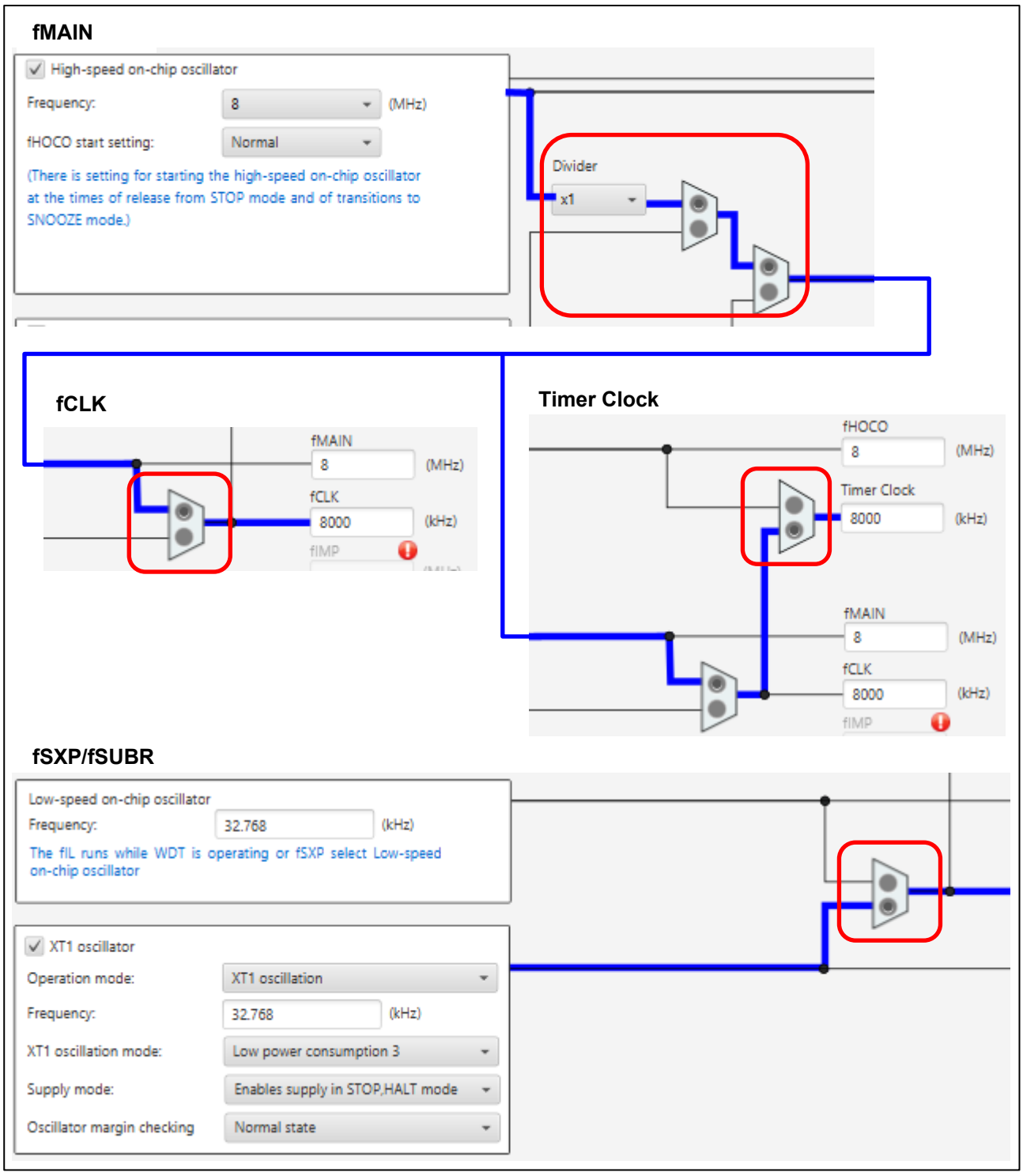


Figure 4 Clock source selection (RL78/L23)

The subsystem clock oscillation stabilization wait time is set to 1.5 seconds by `BSP_CFG_SUBWAITTIME` macro in `<ProjectDir>\src\smc_gen\r_config\r_bsp_config.h`.

```
/* Loop count using the main system clock. */
/* The loop count refers to a loop consisting of a "for" statement that executes a single NOP instruction. */
/* Subsystem clock oscillation stabilization wait time */
/* If the main system clock is 32 MHz, 800000 means 550 ms. */
/* ---> Set stabilization wait time to 1.5 sec. */
#define BSP_CFG_SUBWAITTIME             ( 1500U * 800000U / ( 550U * ( 32U / RP_CPU_CLK ) ) )
```

3.3 System Settings

On-chip debug setting can be set on the [System] page. On-chip debug setting is shown in Table 3.

Table 3 On-chip debug setting

Item	Setting	Note
On-chip debug operation setting	Use emulator	
Emulator setting	E2 Lite	
Pseudo-RRM/DMM function setting	Unused	
Start/Stop function setting	Unused	
Monitoring point function setting	-	
Trace function setting	Unused	RL78/G23, L23
Security ID setting	-	
Use security ID	ON	
Security ID	0x00000000000000000000	
Security ID authentication failure setting	Erase flash memory data	

3.4 Component Settings

3.4.1 Components

Components used by the LoRa®-based software (sample applications) are shown in Table 4.

Table 4 Components used by the LoRa®-based Software

			Sample Application						
Component	Resource	Config name	RadioEvalApp	ping-pong	LoRaSample	LoRaFuotaSample , LoRaFuotaSample_BankSwap	LoRaSensorSample	PrivateLoRaSample	LoRaWanPrivateLoRaComboSample
BSP	-	-	✓	✓	✓	✓	✓	✓	✓
Ports	PORT	SMC_Port	✓	✓	✓	✓	✓	✓	✓
Interrupt Controller	INTC	SMC_INTC	✓	✓	✓	✓	✓	✓	✓
UART Communication	UART0 (RL78/G2x) UARTA0 (RL78/L23)	SMC_UART	✓	✓	✓	✓	✓	✓	✓
SPI (CSI) Communication	CSI11 (RL78/G23) CSI20 (RL78/G22) CSI31 (RL78/L23)	SMC_CSI	✓	✓	✓	✓	✓	✓	✓
IIC Communication (Master mode)	IICA1 (RL78/G23) IIC21 (RL78/G22) IIC10 (RL78/L23)	SMC_IIC	✓	-	-	-	✓	-	-
Interval Timer	ITL000_ITL001	SMC_Timer16bitCapture	✓	✓	✓	✓	✓	✓	✓
	ITL012_ITL013	SMC_Timer16bitInterval	✓	✓	✓	✓	✓	✓	✓
	TAU0_2	SMC_TimerRadioCca	✓	✓	✓	✓	✓	✓	✓

Config names have been changed from the defaults (Config_xxx). The generated function names use them and the LoRa®-based software uses these functions. So, do not change the config names.

3.4.2 BSP Configuration

BSP configuration can be set by clicking [r_bsp] item in the tree view. BSP configuration setting for the LoRa®-based software is shown in Table 5.

Table 5 BSP Configuration

Item	Setting	Note
Start up select	Enable (use BSP startup)	
Control of illicit memory access detection (IAWEN)	Disable	
Protected area in RAM(GRAM1-0)	Disabled	
Protection of the port control register (GPORT)	Disabled	
Protection of the interrupt control register (GINT)	Disabled	
Protection of the clock, voltage detection, and RAM parity error detection control register(GCSC)	Disabled	
Data flash memory area access control (DFLEN)	Disables	RL78/G22
Data flash memory area/extra area access control (DFLEN)	Disables	RL78/G23, L23
Initialization of peripheral functions by Code Generator/Smart Configurator	Enable	
API functions disable (R_BSP_StartClock, R_BSP_StopClock)	Disable	
API functions disable (R_BSP_GetFclkFreqHz)	Enable	
API functions disable (R_BSP_SetClockSource)	Disable	
API functions disable (R_BSP_ChangeClockSetting)	Disable	
API functions disable (R_BSP_SoftwareDelay)	Enable	
Parameter check enable	Disable	
Enable user warm start callback (PRE)	Unused	
Euser warm start callback function name (PRE)	-	
Enable user warm start callback (POST)	Unused	
Euser warm start callback function name (POST)	-	
Watchdog Timer refresh enable	Unused	
Watchdog Timer initialize user function name	-	
Watchdog Timer setting user function name	-	

3.4.3 Ports Component (SMC_Port)

(1) Setting of the Smart Configurator

This component can be set by clicking [SMC_Port] item in the tree view. Port setting is shown in Table 6 (RL78/G23-128p FPB), Table 7 (RL78/G23-64p FPB), Table 8 (RL78/G22 FPB), and Table 9 (FPB-RL78L23).

The LoRa®-based software references the symbolic names described in each table shown below. So, do not change the symbolic names.

Table 6 Port Setting for RL78/G23-128p FPB

Ports Component				**Pin configuration in [Pins] page**
[Port Selection] page		**[PORTx] page**		**Symbolic Name**
Item	**Setting**	**Item**	**Setting**	
			Select (⊙) / Check (☑)	
PORT0	ON	P00 - P07	⊙ Out	
PORT1	ON	P10	⊙ Out	
		P11, P12	⊙ Unused	
		P13 - P17	⊙ Out	
PORT2	ON	P20, P21	⊙ Unused	
		P22 - P27	⊙ Out	
PORT3	ON	P30 - P37	⊙ Out	
PORT4	ON	P40	⊙ Unused	
		P41	⊙ Out	
		P42	⊙ Out	PSYM_ANT_SWITCH_POWER
		P43 - P45	⊙ Out	
		P46	⊙ Out ☑ Output 1	PSYM_RADIO_NSS
		P47	⊙ Out	
PORT5	ON	P50, P51	⊙ Out ☑ Output 1	
		P52 - P57	⊙ Out	
PORT6	ON	P60, P61	⊙ In	
		P62, P63 (*1)	[Not use HS3001] ⊙ Out ☑ Output 1	
			[Use HS3001] ⊙ Unused	
		P64	⊙ Out	
		P65 - P67	⊙ In	
PORT7	ON	P70 - P76	⊙ Out	
		P77 (*2)	⊙ Unused	PSYM_RADIO_DIO_1
PORT8	ON	P80 - P87	⊙ Out	
PORT9	ON	P90 - P94	⊙ Out	
		P95 - P97	⊙ Unused	
PORT10	ON	P100 - P103	⊙ Out	
		P104	⊙ In	
		P105	⊙ Out	
		P106	⊙ In	PSYM_RADIO_BUSY
PORT11	ON	P110 - P114	⊙ Out	
		P115	⊙ Out	PSYM_RADIO_RESET
		P116	⊙ In	
		P117	⊙ In	PSYM_RADIO_DEVICE_SEL
PORT12	ON	P120 - P122	⊙ Out	
		P123, P124	⊙ Unused	
		P125 - P127	⊙ Out	

PORT13	ON	P130	⊙ Out	
		P137 (*3)	[Not use user SW] ⊙ Unused ☑ Input buffer OFF	
			[Use user SW] ⊙ Unused	PSYM_BOARD_SW
PORT14	ON	P140 - P146	⊙ Out	
		P147	⊙ In	PSYM_RADIO_XTAL_SEL
PORT15	ON	P150 - P156	⊙ Out	
Port mode setting	Read Pmn register value			

(*1) HS3001 (the sensor module) is used by RadioEvalApp and LoRaSensorSample, not used otherwise.
(*2) INTP11
(*3) INTP0; User SW on RL78/G23-128p Fast Prototyping Board. See note below.

Table 7 Port Setting for RL78/G23-64p FPB

Ports Component				**Pin configuration in [Pins] page**
[Port Selection] page		**[PORTx] page**		**Symbolic Name**
Item	**Setting**	**Item**	**Setting** **Select (⊙) / Check (☑)**	
PORT0	ON	P00 - P06-	⊙ Out	
PORT1	ON	P10	⊙ Out	
		P11, P12	⊙ Unused	
		P13 - P17	⊙ Out	
PORT2	ON	P20, P21	⊙ Unused	
		P22	⊙ Out	PSYM_RADIO_RESET
		P23	⊙ In	
		P24	⊙ In	PSYM_RADIO_DEVICE_SEL
		P25	⊙ In	PSYM_RADIO_XTAL_SEL
		P26, P27	⊙ Out	
PORT3	ON	P30	⊙ Unused	
		P31	⊙ Out	
PORT4	ON	P40	⊙ Unused	
		P41	⊙ In	
		P42	⊙ In	PSYM_RADIO_BUSY
		P43	⊙ Out	
PORT5	ON	P50, P51	⊙ Unused	
		P52, P53	⊙ Out ☑ Output 1	
		P54, P55	⊙ Out	
PORT6	ON	P60, P61	⊙ In	
		P62, P63 (*1)	[Not use HS3001] ⊙ Out ☑ Output 1	
			[Use HS3001] ⊙ Unused	
PORT7	ON	P70 - P72	⊙ Out	
		P73	⊙ Out	PSYM_ANT_SWITCH_POWER
		P74, P75	⊙ Out	
		P76	⊙ Out ☑ Output 1	PSYM_RADIO_NSS
		P77 (*2)	⊙ Unused	PSYM_RADIO_DIO_1
PORT12	ON	P120 - P122	⊙ Out	
		P123, P124	⊙ Unused	

PORT13	ON	P130	⊙ Out	
		P137 (*3)	[Not use user SW] ⊙ Unused ☑ Input buffer OFF	
			[Use user SW] ⊙ Unused	PSYM_BOARD_SW
PORT14	ON	P140, P141	⊙ Out	
		P146, P147	⊙ Out	
Port mode setting	Read Pmn register value			

(*1) HS3001 (the sensor module) is used by RadioEvalApp, not used otherwise.
(*2) INTP11
(*3) INTP0; User SW on RL78/G23-64p Fast Prototyping Board. See note below.

Table 8 Port Setting for RL78/G22 FPB

Ports Component				**Pin configuration in [Pins] page**
[Port Selection] page		**[PORTx] page**		**Symbolic Name**
Item	**Setting**	**Item**	**Setting** **Select (⊙) / Check (☑)**	
PORT0	ON	P00 - P01	⊙ Out	
PORT1	ON	P10	⊙ Out	
		P11 - P15	⊙ Unused	
		P16	⊙ Out	
		P17	⊙ Out	PSYM_ANT_SWITCH_POWER
PORT2	ON	P20, P21	⊙ Unused	
		P22 - P24	⊙ Out	
		P25	⊙ In	PSYM_RADIO_XTAL_SEL
		P26	⊙ In	PSYM_RADIO_DEVICE_SEL
		P27	⊙ In	
PORT3	ON	P30	⊙ Out	
		P31	⊙ In	PSYM_RADIO_BUSY
PORT4	ON	P40	⊙ Unused	
		P41	⊙ In	
PORT5	ON	P50, P51	⊙ Out	
PORT6	ON	P60, P61	⊙ In	
		P62, P63	⊙ Out ☑ Output 1	
PORT7	ON	P70, P71 (*1)	[Not use HS3001] ⊙ Out ☑ Output 1	
			[Use HS3001] ⊙ Unused	
		P72 - P75	⊙ Out	
PORT12	ON	P120 - P122	⊙ Out	
		P123, P124	⊙ Unused	
PORT13	ON	P130	⊙ Out	
		P137 (*2)	[Not use user SW] ⊙ Unused ☑ Input buffer OFF	
			[Use user SW] ⊙ Unused	PSYM_BOARD_SW
PORT14	ON	P140 (*3)	⊙ In	PSYM_RADIO_DIO_1
		P146	⊙ Out ☑ Output 1	PSYM_RADIO_NSS
		P147	⊙ Out	PSYM_RADIO_RESET
Port mode setting	Read Pmn register value			

(*1) HS3001 (the sensor module) is used by RadioEvalApp and LoRaSensorSample, not used otherwise.
(*2) INTP0; User SW on RL78/G22 Fast Prototyping Board. See note below.

(*3) INTP6

Table 9 Port Setting for FPB-RL78L23

Ports Component				**Pin configuration in [Pins] page**
[Port Selection] page		**[PORTx] page**		**Symbolic Name**
Item	**Setting**	**Item**	**Setting**	
			Select (⊙) / Check (☑)	
PORT0	ON	P00	⊙ Unused ☑ Input buffer OFF	
		P01	⊙ In	SMC_PIN_RADIO_BUSY
		P02 (*1)	⊙ Unused	SMC_PIN_RADIO_DIO_1
		P03	⊙ Out ☑ Input buffer OFF	
		P04 - P07	⊙ Unused	
PORT1	ON	P10 - P14	⊙ Unused	
		P15	⊙ In	
		P16	⊙ Out ☑ Input buffer OFF	
		P17	⊙ Unused	
PORT2	ON	P20, P21	⊙ Unused	
		P22	⊙ Out ☑ Input buffer OFF	
		P23	⊙ Out	
		P24	⊙ In	SMC_PIN_RADIO_XTAL_SEL
		P25	⊙ In	SMC_PIN_RADIO_DEVICE_SEL
		P26	⊙ In	
		P27	⊙ Out ☑ Input buffer OFF	SMC_PIN_RADIO_RESET
PORT3	ON	P30, P31	⊙ Unused	
		P32	⊙ Out	
		P33 - P35	⊙ Out ☑ Input buffer OFF	
PORT4	ON	P40	⊙ Unused	
		P41 (*2)	[Not use HS3001] ⊙ Out ☑ Output 1	
			[Use HS3001] ⊙ Unused	
		P42 - P43	⊙ Out ☑ Input buffer OFF	
		P44 - P47	⊙ Unused	
PORT5	OFF	-	-	
PORT6	ON	P60, P61	⊙ In	
		P62	⊙ Unused	
		P63	⊙ Out	
		P64, P65	⊙ Out ☑ Output 1 ☑ Input buffer OFF	
		P66 (*2)	[Not use HS3001] ⊙ Out ☑ Output 1	
			[Use HS3001] ⊙ Unused	
		P67	⊙ Out	
PORT7	ON	P70 - P75	⊙ Unused	
		P76, P77	⊙ Out ☑ Input buffer OFF	
PORT8	ON	P80	⊙ Out ☑ Input buffer OFF	SMC_PIN_RADIO_NSS
		P81	⊙ Out ☑ Input buffer OFF	SMC_PIN_ANT_SWITCH_POWER

		P82	⊙ Out ☑ Input buffer OFF	
		P83 - P87	⊙ Unused	
PORT9	OFF	-	-	
PORT12	ON	P121, P122	⊙ Out	
		P123 - P127	⊙ Unused	
PORT13	ON	P130	⊙ Unused	
		P137 (*3)	[Not use user SW] ⊙ Unused ☑ Input buffer OFF	
			[Use user SW] ⊙ Unused	PSYM_BOARD_SW
PORT14	ON	P140 - P145	⊙ Unused	
		P146, P147	⊙ Out ☑ Input buffer OFF	
Port mode setting	Read Pmn register value			

(*1) INTP7
(*2) HS3001 (the sensor module) is used by RadioEvalApp and LoRaSensorSample, not used otherwise.
(*3) INTP0; User SW on RL78/L23 Fast Prototyping Board. See note below.

Note: User SW on RL78 Fast Prototyping Board is used for debugging purposes (wake up MCU and receive AT command) by following sample applications.

- LoRaSample
- LoRaFuotaSample
- LoRaFuotaSample_BankSwap
- LoRaSensorSample
- LoRaWanPrivateLoRaComboSample

(2) Modification of the Generated Code

■ ***Added Code to Disable User SW on RL78 Fast Prototyping Board***

File	<ProjectDir>\src\smc_gen\SMC_Port\SMC_Port_user.c
Function	R_SMC_PORT_Create_UserInit() * Refer to `R_{Config_PORT}_Create_UserInit` function in [3].

Code	`/* Start user code for global. Do not edit comment generated here */`
	`#ifndef APP_TEST_LOWPOWER`
	`// Set port digital input disable register to disable input (=1).`
	`#define R_SMC_PORT_INPUT_BUFFER_OFF_HELPER( x, y ) { PDIDIS##x## |= (1 << (y)); }`
	`#define R_SMC_PORT_INPUT_BUFFER_OFF(...) R_SMC_PORT_INPUT_BUFFER_OFF_HELPER(__VA_ARGS__)`
	`#endif`
	`/* End user code. Do not edit comment generated here */`
	
	`void R_SMC_PORT_Create_UserInit(void)`
	`{`
	`/* Start user code for user init. Do not edit comment generated here */`
	`#ifndef APP_TEST_LOWPOWER`
	`/* SW: Disable digital input */`
	`R_SMC_PORT_INPUT_BUFFER_OFF( SMC_PIN_BOARD_SW );`
	`#endif // APP_TEST_LOWPOWER`
	`/* End user code. Do not edit comment generated here */`
	`}`

Target Application	LoRaSample, LoRaFuotaSample, LoRaFuotaSample_BankSwap, LoRaSensorSample, and LoRaWanPrivateLoRaComboSample.

User SW on RL78 Fast Prototyping Board is used for debugging purposes (wake up MCU to receive AT command). But this debugging feature is not usually used. So, the code to disable user SW (digital input) is added to the function described above.

Please see also section 3.4.4 to disable user SW.

3.4.4 Interrupt Controller Component (SMC_INTC)

(1) **Setting of the Smart Configurator**

The LoRa®-based software uses INTP to get the event triggers from SX126x and user SW on the RL78 Fast Prototyping Board.
This component can be set by clicking [SMC_INTC] item in the tree view. Interrupt controller setting is shown in Table 10 (RL78/G23-128p FPB and RL78/G23-64p FPB), Table 11 (RL78/G22 FPB), and Table 12 (FPB-RL78L23).

Table 10 INTP Setting for RL78/G23-128p FPB and RL78/G23-64p FPB

INTP (Port)	**Setting**			**Note**
	ON/OFF	**Valid edge**	**Priority**	
INTP0 (P137)	OFF	-	-	-
	ON	Falling edge	Level 3 (low)	User SW on RL78/G23 Fast Prototyping Board. See Note below
INTP11 (P77)	ON	Rising edge	Level 1	SX126x event trigger
The others	OFF	-	-	-

Table 11 INTP Setting for RL78/G22 FPB

INTP (Port)	**Setting**			**Note**
	ON/OFF	**Valid edge**	**Priority**	
INTP0 (P137)	OFF	-	-	-
	ON	Falling edge	Level 3 (low)	User SW on RL78/G22 Fast Prototyping Board. See Note below
INTP6 (P140)	ON	Rising edge	Level 1	SX126x event trigger
The others	OFF	-	-	-

Table 12 INTP Setting for FPB-RL78L23

INTP (Port)	**Items**	**Setting**	**Note**
INTP0 setting (P137)	INTP0	OFF	-
		ON	User SW on RL78/L23 Fast Prototyping Board. See Note below
	Valid edge	Falling edge	
	Priority	Level 3 (low)	
	Enable the restart or forced output stop signal 2 for timer KB4	OFF	
INTP7 setting (P02)	INTP7	ON	SX126x event trigger
	Valid edge	Rising edge	
	Priority	Level 1	
	Enable the restart or forced output stop signal 2 for timer KB4	OFF	
The others	INTPx (x: 1-6)	OFF	-

Note: User SW on RL78 Fast Prototyping Board is used for debugging purposes (wake up MCU and receive AT command) by following sample applications.
- LoRaSample
- LoRaFuotaSample
- LoRaFuotaSample_BankSwap

- LoRaSensorSample
- LoRaWanPrivateLoRaComboSample

(2) Modification of the Generated Code

■ ***Added API Functions and Code to Call the Callback Function for INTPx Interrupt for SX126x Event***

File	<ProjectDir>\src\smc_gen\SMC_INTC\SMC_INTC_user.c
Added Functions	void R_SMC_INTC_INTP_RadioDio1_SetHandler(GpioIrqHandler *p_irqHandler)

Code	/***
	Global variables and functions
	***/
	/* Start user code for global. Do not edit comment generated here */
	static GpioIrqHandler *gp_smc_intc_radio_dio1_handle_cb;
	#ifdef APP_TEST_LOWPOWER
	static GpioIrqHandler *gp_smc_intc_board_sw_handle_cb;
	#endif // APP_TEST_LOWPOWER
	/* End user code. Do not edit comment generated here */
	
	void R_SMC_INTC_INTP_RadioDio1_SetHandler(GpioIrqHandler *p_irqHandler)
	{
	gp_smc_intc_radio_dio1_handle_cb = p_irqHandler;
	}

Target Application	All.

This function is to register the callback function to be called when INTP*x* interrupt for SX126x event is generated.

`GpioIrqHandler` is defined in `(package top)\samples\project\src\system\gpio.h`. It is a function pointer type declaration, defined as follow:

```
typedef void( GpioIrqHandler )( void );
```

Note that if `p_irqHandler` is set to NULL, nothing will be done even if corresponding INTP*x* interrupt is generated.

File	<ProjectDir>\src\smc_gen\SMC_INTC\SMC_INTC_user.c
Added Functions	void R_SMC_INTC_INTP_RadioDio1_Start(void) void R_SMC_INTC_INTP_RadioDio1_Stop(void)

Code	void R_SMC_INTC_INTP_RadioDio1_Start(void)
	{
	R_SMC_INTC_INTP11_Start();
	}
	
	void R_SMC_INTC_INTP_RadioDio1_Stop(void)
	{
	R_SMC_INTC_INTP11_Stop();
	}
	* In case RL78/G23

Target Application	All.

These functions are used to enable / disable INTPx interrupt for SX126x event.
These functions call the API function generated by the Smart Configurator. When RL78/G23 is used, API functions for INTP11 are used. When RL78/G22 is used, API functions for INTP6 is used. When RL78/L23 is used, API functions for INTP7 is used. Please also refer to `R_{Config_INTC}_Intpn_Start` function and `R_{Config_INTC}_Intpn_Stop` function in [3].

File	<ProjectDir>\src\smc_gen\SMC_INTC\SMC_INTC_user.c
Function	r_SMC_INTC_intp*x*_interrupt() * *x* = 6,7,11 * Refer to `r_{Config_INTC}_intp*n*_interrupt` function in [3].

Code	`static void __near r_SMC_INTC_intp11_interrupt(void)`
	`{`
	`/* Start user code for r_SMC_INTC_intp11_interrupt. Do not edit comment generated here */`
	`if( gp_smc_intc_radio_dio1_handle_cb != NULL )`
	`{`
	`(*gp_smc_intc_radio_dio1_handle_cb)();`
	`}`
	`/* End user code. Do not edit comment generated here */`
	`}`
	* In case RL78/G23

Target Application	All.

The code to call registered callback function is added to the function described above.

■ *Added API Functions and Code to Disable User SW on RL78 Fast Prototyping Board*

File	<ProjectDir>\src\smc_gen\SMC_INTC\SMC_INTC_user.c
Added Functions	void R_SMC_INTC_INTP_BoardSW_SetHandler(GpioIrqHandler *p_irqHandler)

Code	`/*****...*****`
	`Global variables and functions`
	`*****...*****`
	`/* Start user code for global. Do not edit comment generated here */`
	`static GpioIrqHandler *gp_smc_intc_radio_dio1_handle_cb;`
	`#ifdef APP_TEST_LOWPOWER`
	`static GpioIrqHandler *gp_smc_intc_board_sw_handle_cb;`
	`#endif // APP_TEST_LOWPOWER`
	`/* End user code. Do not edit comment generated here */`
	`void R_SMC_INTC_INTP_BoardSW_SetHandler( GpioIrqHandler *p_irqHandler )`
	`{`
	`gp_smc_intc_board_sw_handle_cb = p_irqHandler;`
	`}`

Target Application	LoRaSample, LoRaFuotaSample, LoRaFuotaSample_BankSwap, LoRaSensorSample, and LoRaWanPrivateLoRaComboSample.

This function is to register the callback function to call when INTP0 interrupt for user SW is generated.

`GpioIrqHandler` is defined in `(package top)\samples\project\src\system\gpio.h`. It is a function pointer type declaration, defined as follow:

```
typedef void( GpioIrqHandler )( void );
```

Note that if `p_irqHandler` is set to NULL, nothing will be done even if INTP0 interrupt is generated.

File	<ProjectDir>\src\smc_gen\SMC_INTC\SMC_INTC_user.c
Added Functions	void R_SMC_INTC_INTP_BoardSW_Start(void) void R_SMC_INTC_INTP_BoardSW_Stop(void)

Code	`void R_SMC_INTC_INTP_BoardSW_Start( void )`
	`{`
	`R_SMC_INTC_INTP0_Start();`
	`}`
	`void R_SMC_INTC_INTP_BoardSW_Stop( void )`
	`{`
	`R_SMC_INTC_INTP0_Stop();`
	`}`

Target Application	LoRaSample, LoRaFuotaSample, LoRaFuotaSample_BankSwap, LoRaSensorSample, and LoRaWanPrivateLoRaComboSample.

These functions are used to enable / disable INTP0 interrupt for SX126x event. These functions call the API function generated by the Smart Configurator. Please also refer to `R_{Config_INTC}_Intp*n*_Start` function and `R_{Config_INTC}_Intp*n*_Stop` function in [3].

File	<ProjectDir>\src\smc_gen\SMC_INTC\SMC_INTC_user.c
Functions	r_SMC_INTC_intp0_interrupt() * Refer to `r_{Config_INTC}_intpn_interrupt` function in [3].

Code	`static void __near r_SMC_INTC_intp0_interrupt(void)`
	`{`
	`    /* Start user code for r_SMC_INTC_intp0_interrupt. Do not edit comment generated here */`
	`#ifdef APP_TEST_LOWPOWER`
	`    if( gp_smc_intc_board_sw_handle_cb != NULL )`
	`    {`
	`        (*gp_smc_intc_board_sw_handle_cb)();`
	`    }`
	`#endif`
	`    /* End user code. Do not edit comment generated here */`
	`}`

Target Application	LoRaSample, LoRaFuotaSample, LoRaFuotaSample_BankSwap, LoRaSensorSample, and LoRaWanPrivateLoRaComboSample.

The code to call registered callback function is added to the function described above.

User SW on RL78 Fast Prototyping Board is used for debugging purposes (wake up MCU to receive AT command). When this debugging feature is not used, user SW is disabled (see section 3.4.3(2)). So, the code for INTP0 is not necessary.

- Disabled added API functions; `R_SMC_INTC_INTP_BoardSW_SetHandler()`, `R_SMC_INTC_INTP_BoardSW_Start()`, and `R_SMC_INTC_INTP_BoardSW_Stop()` (see above).
- Do nothing even if INTP0 interrupt is generated.

3.4.5 UART Communication Component (SMC_UART)

(1) Setting of the Smart Configurator

The LoRa®-based software uses UART mainly to communicate with a host PC which operates it with the AT command sets.
This component can be set by clicking [SMC_UART] item in the tree view. UART setting is shown in Table 13 (UART0 transmission for RL78/G2x),

Table 14 (UART0 reception for RL78/G2x) and Table 15 (UARTA0 for RL78/L23).

Table 13 UART0 Setting - [Transmission] Page (For other than RL78/G2x)

Item	Setting	Note
UART0 clock setting	-	
Operation clock	CK00	
Clock source	fCLK	
Transfer mode setting	Continuous transfer mode	
Data length setting	8 bits	
Transfer direction setting	LSB	
Parity setting	None	
Stop bit length setting	1 bit	
Transfer data level setting	Non-reverse	
Transfer rate setting	-	
Transfer rate setting	115200 (bps)	Specify this value.
Interrupt setting	-	
Transmit end interrupt priority (INTST0)	Level 0 (high)	
Callback function setting	-	
Transmission end	ON	

Table 14 UART0 Setting - [Reception] Page (For other than RL78/G2x)

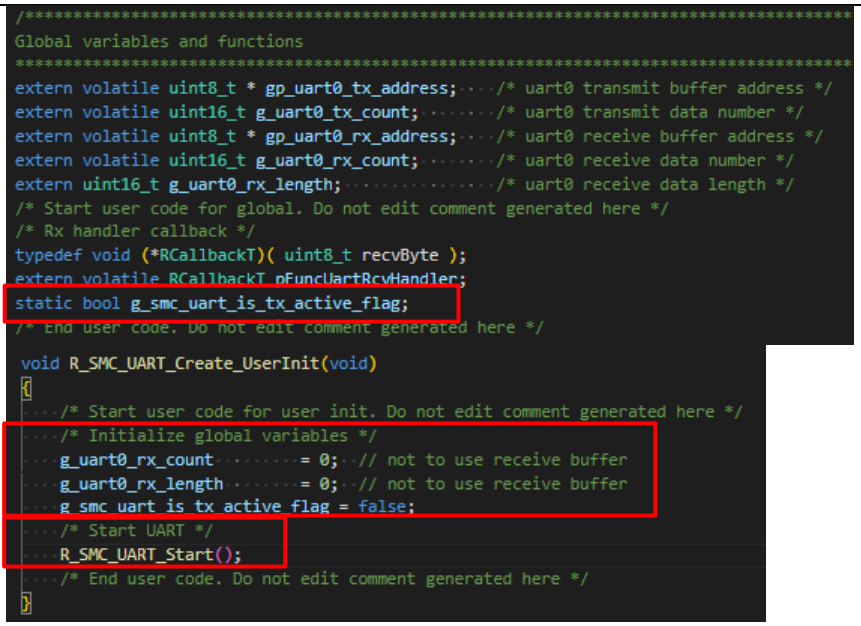
Item	Setting	Note
UART0 clock setting	-	
Operation clock	CK00	
Clock source	fCLK	
Data length setting	8 bits	
Transfer direction setting	LSB	
Parity setting	None	
Stop bit length setting	1 bit	
Receive data level setting	Non-reverse	
Transfer rate setting	-	
Transfer rate setting	115200 (bps)	Specify this value.
Interrupt setting	-	
Reception end interrupt priority (INTSR0)	Level 0 (high)	
Reception error interrupt priority (INTSRE0)	ON - Level 3 (low)	
Callback function setting	-	
Reception end	OFF	
Reception error	OFF	

Table 15 UARTA0 Setting (For RL78/L23)

Item	Setting	Note
UARTA0 clock setting	-	
Operation clock	fSEL/2	
	fSEL clock select fIHP	
ELCL clock	-	
UARTA0 clock output signal setting	Disable	
CLKA0 pin output setting	-	
TxDA0 pin output setting	Enable	
Data length setting	8 bits	
Transfer direction setting	LSB	
Parity setting	None	
Stop bit length setting	1 bit	
Transfer data level setting	Non-reverse	
Transmit mode setting	Continuous transmit by interrupt	
Receive error occurs setting	INTURE interrupt occurs	
Transfer rate setting	-	
Transfer rate setting	115200 (bps)	
Interrupt setting	-	
Transmit end interrupt priority (INTUT0)	Level 0 (high)	
Reception end interrupt priority (INTUR0)	Level 0 (high)	
Reception error interrupt priority (INTURE0)	Level 3 (low)	
Callback function setting	-	
Transmission end	ON	
Reception end	OFF	
Reception error	OFF	

(2) **Modification of the Generated Code**

■ ***Added Code to Initialize Variables and Start UART***

File	<ProjectDir>\src\smc_gen\SMC_UART\SMC_UART_user.c
Function	R_SMC_UART_Create_UserInit() * Refer to `R_{Config_UARTq}_Create_UserInit` function in [3].
Code	
Target Application	All.

```
/***********************************************************************************
Global variables and functions
***********************************************************************************/
extern volatile uint8_t * gp_uart0_tx_address;      /* uart0 transmit buffer address */
extern volatile uint16_t g_uart0_tx_count;          /* uart0 transmit data number */
extern volatile uint8_t * gp_uart0_rx_address;      /* uart0 receive buffer address */
extern volatile uint16_t g_uart0_rx_count;          /* uart0 receive data number */
extern uint16_t g_uart0_rx_length;                  /* uart0 receive data length */
/* Start user code for global. Do not edit comment generated here */
/* Rx handler callback */
typedef void (*RCallbackT)( uint8_t recvByte );
extern volatile RCallbackT pFuncUartRcvHandler;
static bool g_smc_uart_is_tx_active_flag;
/* End user code. Do not edit comment generated here */

void R_SMC_UART_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    /* Initialize global variables */
    g_uart0_rx_count        = 0;  // not to use receive buffer
    g_uart0_rx_length       = 0;  // not to use receive buffer
    g_smc_uart_is_tx_active_flag = false;
    /* Start UART */
    R_SMC_UART_Start();
    /* End user code. Do not edit comment generated here */
}
```

The code to initialize global variables is added to the function described above. `g_uart0_rx_count` and `g_uart0_rx_length` are generated variables by the Smart Configurator.
`g_smc_uart_is_tx_active_flag` is added variable to monitor whether UART transmission is ended (see description below).
And the code to start UART by calling `R_SMC_UART_Start()` function which is in `SMC_UART.c` (*) is added to the function described above.

(*) Refer to `R_{Config_UARTq}_Start` function in [3].

■ ***Added Code to Call Callback Function for UART Reception***

File	<ProjectDir>\src\smc_gen\SMC_UART\SMC_UART_user.c
Function	r_SMC_UART_interrupt_receive() * Refer to `r_{Config_UARTq}_interrupt_receive` function in [3].
Code	(For other than RL78/L23) (For RL78/L23)
Target Application	All except ping-pong.

```
/***********************************************************************************************
Global variables and functions
***********************************************************************************************/
extern volatile uint8_t * gp_uart0_tx_address;      /* uart0 transmit buffer address */
extern volatile uint16_t g_uart0_tx_count;          /* uart0 transmit data number */
extern volatile uint8_t * gp_uart0_rx_address;      /* uart0 receive buffer address */
extern volatile uint16_t g_uart0_rx_count;          /* uart0 receive data number */
extern uint16_t g_uart0_rx_length;                  /* uart0 receive data length */
/* Start user code for global. Do not edit comment generated here */
/* Rx handler callback */
typedef void (*RCallbackT)( uint8_t recvByte );
extern volatile RCallbackT pFuncUartRcvHandler;
static bool g_smc_uart_is_tx_active_flag;
/* End user code. Do not edit comment generated here */
```

(For other than RL78/L23)

```
static void __near r_SMC_UART_interrupt_receive(void)
{
    uint8_t rx_data;

    rx_data = RXD0;

    if (g_uart0_rx_length > g_uart0_rx_count)
    {
        *gp_uart0_rx_address = rx_data;
        gp_uart0_rx_address++;
        g_uart0_rx_count++;
    }
    /* Start user code */
    if( pFuncUartRcvHandler != NULL )
    {
        (*pFuncUartRcvHandler)( rx_data );
    }
    /* End user code */
}
```

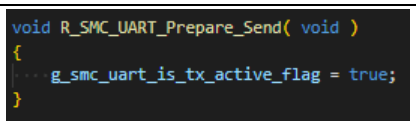
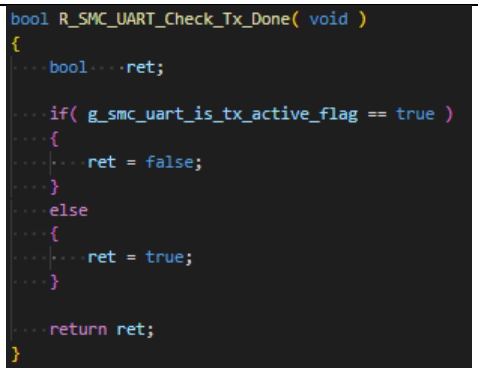
(For RL78/L23)

```
static void __near r_SMC_UART_interrupt_receive(void)
{
    /* Start user code */
    volatile uint8_t rx_data;
#if 0
    /* End user code */
    uint16_t temp;

    temp = g_uarta0_rx_total_num;

    if (temp > g_uarta0_rx_num)
    {
        *gp_uarta0_rx_address = RXBA0;
        gp_uarta0_rx_address++;
        g_uarta0_rx_num++;
    }
    /* Start user code */
#endif
    rx_data = RXBA0;
    if( pFuncUartRcvHandler != NULL )
    {
        (*pFuncUartRcvHandler)( rx_data );
    }
    /* End user code */
}
```

Disable original code

The code to call callback function for analyzing the receive data is added to the function described above.
The callback function is set at the application initialization.

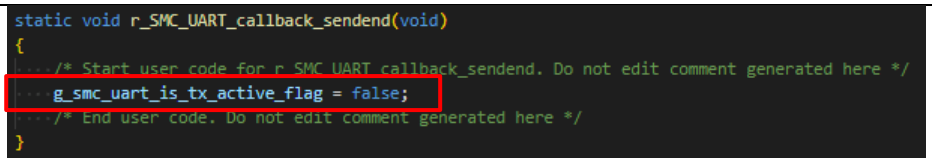
■ ***Added API Functions and Code to Check UART Transmission End***

File	<ProjectDir>\src\smc_gen\SMC_UART\SMC_UART_user.c
Added Functions	void R_SMC_UART_Prepare_Send(void) bool R_SMC_UART_Check_Tx_Done(void)
Code	 
Target Application	All.

When the requester (the LoRa®-based software) requests UART transmission by calling `R_SMC_UARTSend()` function which is in `SMC_UART.c` (*), the requester has no way of knowing when UART transmission is ended. So, the requester monitors whether UART transmission is ended by using added functions.

Note that:
- Call `R_SMC_UART_Prepare_Send()` before requesting UART transmission.
- `R_SMC_UART_Check_Tx_Done()` returns true if UART transmission is ended, false otherwise.

(*) Refer to `R_{Config_UARTq}_Send` function in [3].

File	<ProjectDir>\src\smc_gen\SMC_UART\SMC_UART_user.c
Function	r_SMC_UART_callback_sendend() * Refer to `r_{Config_UARTq}_callback_sendend` function in [3].
Code	
Target Application	All.

The code to monitor whether UART transmission is ended is added to the function described above.

This function is generated when [callback function setting] setting in UART transmission setting is enabled.

3.4.6 SPI (CSI) Communication Component (SMC_CSI)

(1) Setting of the Smart Configurator

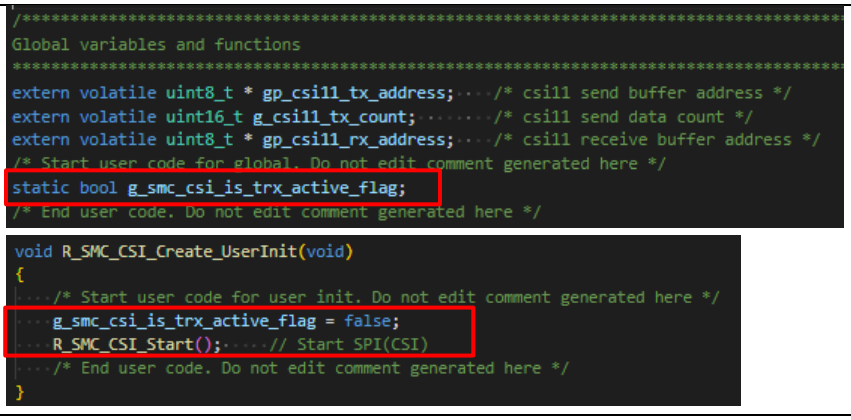
The LoRa®-based software uses SPI (CSI) to communicate with SX126x.
This component can be set by clicking [SMC_CSI] item in the tree view. SPI (CSI) setting is shown in Table 16.

Table 16 SPI (CSI) Setting

Item	Setting	Note
Transfer clock setting	-	
Transfer clock mode	Internal clock (master)	
Operation clock	CK01 (RL78/G23) CK10 (The other)	
Clock source	fCLK	
Transfer mode setting	Single transfer mode	
Data length setting	8 bits	
Transfer direction setting	MSB	
Specification of data timing	Type 4	
Transfer rate setting	-	
Baudrate	2000000 (bps)	Specify this value.
Interrupt setting		
Transfer interrupt priority (INTCSI***xx***) * ***xx*** = 11 (RL78/G23), 20 (RL78/G22), 31 (RL78/L23)	Level 0 (high)	
Callback function setting	-	
Transmission end	OFF	
Reception end	ON	
Reception error	ON	

(2) Modification of the Generated Code

■ ***Added Code to Initialize Variable and Start SPI***

File	<ProjectDir>\src\smc_gen\SMC_CSI\SMC_CSI_user.c
Function	R_SMC_CSI_Create_UserInit() * Refer to `r_{Config_CSIp}_interrupt` function in [3].
Code	
Target Application	All.

```
/***********************************************************************************
Global variables and functions
***********************************************************************************/
extern volatile uint8_t * gp_csi11_tx_address;      /* csi11 send buffer address */
extern volatile uint16_t g_csi11_tx_count;          /* csi11 send data count */
extern volatile uint8_t * gp_csi11_rx_address;      /* csi11 receive buffer address */
/* Start user code for global. Do not edit comment generated here */
static bool g_smc_csi_is_trx_active_flag;
/* End user code. Do not edit comment generated here */
```

```
void R_SMC_CSI_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    g_smc_csi_is_trx_active_flag = false;
    R_SMC_CSI_Start();      // Start SPI(CSI)
    /* End user code. Do not edit comment generated here */
}
```

The code to initialize global variables is added to the function described above.
A variable `g_smc_csi_is_trx_active_flag` is added to monitor whether SPI communication is ended (see description below).
And the code to start SPI by calling `R_SMC_CSI_Start()` function which is in `SMC_CSI.c` (*) is added to the function described above.

(*) Refer to `R_{Config_CSIp}_Start` function in [3].

■ ***Added API Functions and Code to Check SPI Communication End***

File	<ProjectDir>\src\smc_gen\SMC_CSI\SMC_CSI_user.c
Added Functions	void R_SMC_CSI_Prepare_Send_Receive(void) bool R_SMC_CSI_Check_TRx_Done(void)

Code	`void R_SMC_CSI_Prepare_Send_Receive( void )`
	`{`
	`    g_smc_csi_is_trx_active_flag = true;`
	`}`
	
	`bool R_SMC_CSI_Check_TRx_Done( void )`
	`{`
	`    bool    ret;`
	`    if( g_smc_csi_is_trx_active_flag == true )`
	`    {`
	`        ret = false;`
	`    }`
	`    else`
	`    {`
	`        ret = true;`
	`    }`
	`    return ret;`
	`}`

Target Application	All.

When the requester (the LoRa®-based software) requests SPI communication by calling `R_SMC_CSI_Send_Receive()` function which is in `SMC_CSI.c` (*), the requester has no way of knowing when SPI communication is ended. So, the requester monitors whether SPI communication is ended by using added functions.

Note that:
- Call `R_SMC_CSI_Prepare_Send_Receive()` before requesting SPI communication.
- `R_SMC_CSI_Check_TRx_Done()` returns true if SPI communication is ended, false otherwise.

(*) Refer to `R_{Config_CSIp}_Send_Receive` function in [3].

File	<ProjectDir>\src\smc_gen\SMC_CSI\SMC_CSI_user.c
Functions	r_SMC_CSI_callback_receiveend() r_SMC_CSI_callback_error() * Refer to `r_{Config_CSIp}_callback_receiveend` and `r_{Config_CSIp}_callback_error` functions in [3].

Code	`static void r_SMC_CSI_callback_receiveend(void)`
	`{`
	`    /* Start user code for r_SMC_CSI_callback_receiveend. Do not edit comment generated here */`
	`    g_smc_csi_is_trx_active_flag = false;`
	`    /* End user code. Do not edit comment generated here */`
	`}`
	
	`static void r_SMC_CSI_callback_error(uint8_t err_type)`
	`{`
	`    /* Start user code for r_SMC_CSI_callback_error. Do not edit comment generated here */`
	`    (void)err_type;  // unused argument`
	`    g_smc_csi_is_trx_active_flag = false;`
	`    /* End user code. Do not edit comment generated here */`
	`}`

Target Application	All.

The code to monitor whether SPI communication is ended is added to the function described above.

These functions are generated when [callback function setting] in SPI (CSI) setting is enabled.

3.4.7 IIC Communication Component (Master mode) (SMC_IIC)

(1) Setting of the Smart Configurator

The LoRa®-based software uses IIC to communicate with HS3001 (the sensor module).
This component can be set by clicking [SMC_IIC] item in the tree view. When using RL78/G23, serial interface IICA is used. IICA setting is shown in Table 17. When using RL78/G22 or RL78/L23, simplified IIC (serial array unit) is used. IIC setting is shown in Table 18.

Table 17 IIC (Serial Interface IICA) Master Setting for RL78/G23

Item	Setting	Note
Clock mode setting	-	
Clock mode setting	fCLK	
Local address setting	-	
Address	16	
Operation mode setting	Standard	
Transfer clock (fSCL)	100000 (bps)	
tR and tF setting	-	
Set tR and tF manually	OFF	
tR	-	
tF	-	
Interrupt setting	-	
Communication end interrupt priority (INTIICA1)	Level 3 (low)	
Callback function setting	-	
Master transmission end	ON	
Master reception end	ON	
Master error	ON	
Callback function enhanced feature setting	-	
Generate stop condition in master transmission/reception end callback function	ON	

Table 18 IIC (Simplified IIC Communication) Master Setting for RL78/G22 and RL78/L23

Item	Setting	Note
IIC21 clock setting (RL78/G22) IIC10 clock setting (RL78/L23)	-	
Operation clock	CK11 (RL78/G22) CK00 (RL78/L23)	
Clock source	fCLK	
Transfer rate setting	-	
Transfer rate	100000 (bps)	
Interrupt setting	-	
Communication end interrupt priority (INTIIC21)	Level 3 (low)	
Callback function setting	-	
Master transmission end	ON	
Master reception end	ON	
Master error	ON	

(2) **Modification of the Generated Code**

■ ***Added Code to Call Callback Routine from IIC Interrupts***

File	<ProjectDir>\src\smc_gen\SMC_IIC\SMC_IIC_user.c
Functions	r_SMC_IIC_callback_master_sendend() r_SMC_IIC_callback_master_receiveend() r_SMC_IIC_callback_master_error() * Refer to the following functions in [3] : (RL78/G23) r_{Config_IICAn}_callback_master_sendend, r_{Config_IICAn}_callback_master_receiveend, and r_{Config_IICAn}_callback_master_error (RL78/G22, RL78/L23) r_{Config_IICr}_callback_master_sendend, r_{Config_IICr}_callback_master_receiveend, and r_{Config_IICr}_callback_master_error

Code	`static void r_SMC_IIC_callback_master_sendend(void)`
	`{`
	`    SPT1 = 1U;`
	`/* Start user code for r_SMC_IIC_callback_master_sendend. Do not edit comment generated here */`
	`    app_iic_tx_done_callback();`
	`/* End user code. Do not edit comment generated here */`
	`}`
	
	`static void r_SMC_IIC_callback_master_receiveend(void)`
	`{`
	`    SPT1 = 1U;`
	`/* Start user code for r_SMC_IIC_callback_master_receiveend. Do not edit comment generated here */`
	`    app_iic_rx_done_callback();`
	`/* End user code. Do not edit comment generated here */`
	`}`
	
	`static void r_SMC_IIC_callback_master_error(MD_STATUS flag)`
	`{`
	`    /* Start user code for r_SMC_IIC_callback_master_error. Do not edit comment generated here */`
	`    app_iic_error_callback();`
	`    /* End user code. Do not edit comment generated here */`
	`}`

Target Application	LoRaSensorSample and RadioEvalApp (in case using sensor HS3001).

The code to call callback routine corresponding to each interrupt is added to the functions described above.

These functions are generated when [callback function setting] in IIC master setting is enabled.

3.4.8 Interval Timer Component (SMC_Timer16bitCapture, SMC_Timer16bitInterval)

(1) Setting of the Smart Configurator

The LoRa®-based software uses two 16-bit interval timers, one for 16-bit capture mode and one for 16-bit counter mode. 16-bit capture mode can be set by clicking [SMC_Timer16bitCapture] item in the tree view, and 16-bit counter mode can be set by clicking [SMC_Timer16bitInterval] item in the tree view. The settings for each mode are shown in
Table 19 and Table 20.

Table 19 Interval Timer Setting (ITL000_ITL001 for 16-bit Capture Mode; SMC_Timer16bitCapture)

Item	Setting	Note
Clock setting	-	
Operation clock (fITL0)	fSXP	
Clock source	fIT0/8	
Capture trigger setting	-	
Capture trigger	fSXP (rising edge)	
16-bit counter (ITL000 + ITL001) clear after capture is complete	OFF	
Interrupt setting	-	
Detection of compare match/capture completion (INTITL)	ON	
Priority	Level 1	Do not change it.

Table 20 Interval Timer Setting (ITL012_ITL013 for 16-bit Counter Mode; SMC_Timer16bitInterval)

Item	Setting	Note
Clock setting	-	
Operation clock (fITL0)	fSXP	
Clock source	fIT0/8	
Interval timer setting	-	
Interval value	100 ms	This setting is not used.
Interrupt setting	-	
Detection of compare match/capture completion (INTITL)	ON	
Priority	Level 1	Do not change it.

(2) Modification of the Generated Code

■ ***Changed Generated Code to Change the Interrupt Source***

File	(1) <ProjectDir>\src\smc_gen\SMC_Timer16bitCapture\SMC_Timer16bitCapture.c (2) <ProjectDir>\src\smc_gen\general\r_cg_itl_common_user.c
Functions	(1) R_SMC_Timer16bitCapture_Start() R_SMC_Timer16bitCapture_Stop() (2) r_itl_interrupt() * Refer to the following functions in [3] : `R_{Config_ITLn_ITLm}_Start`, `R_{Config_ITLn_ITLm}_Stop`, and `r_itl_interrupt`

Code

(1)

```
void R_SMC_Timer16bitCapture_Start(void)
{
    /* Start user code */
    ITLS0 &= (uint8_t)~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;
    ITLMKF0 &= (uint8_t)~_01_ITL_CHANNEL0_COUNT_MATCH_MASK;
#if 0   // not use capture complete
    /* End user code */
    ITLS0 &= (uint8_t)~_10_ITL_CAPTURE_COMPLETE_DETECTE;
    ITLMKF0 &= (uint8_t)~_10_ITL_CAPTURE_COMPLETE_MASK;
    /* Start user code */
#endif  // not use capture complete
    /* End user code */
    ITLEN00 = 1U;
}
```

Disable original code

```
void R_SMC_Timer16bitCapture_Stop(void)
{
    /* Start user code */
    ITLMKF0 |= _01_ITL_CHANNEL0_COUNT_MATCH_MASK;
    ITLS0 &= (uint8_t)~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;
#if 0   // not use capture complete
    /* End user code */
    ITLMKF0 |= _10_ITL_CAPTURE_COMPLETE_MASK;
    ITLS0 &= (uint8_t)~_10_ITL_CAPTURE_COMPLETE_DETECTE;
    /* Start user code */
#endif  // not use capture complete
    /* End user code */
    ITLEN00 = 0U;
}
```

Disable original code

(2)

```
static void __near r_itl_interrupt(void)
{
    /* Start user code */
    SetLowPowerPost();

#if RP_IT_INTITL_INTLEVEL != 0U
    BoardEnableMultipleInterrupt();
#endif
    /* End user code */
    if (_04_ITL_CHANNEL2_COUNT_MATCH_DETECTE == (ITLS0 & _04_ITL_CHANNEL2_COUNT_MATCH_DETECTE))
    {
        ITLS0 &= (uint8_t)~_04_ITL_CHANNEL2_COUNT_MATCH_DETECTE;
        R_SMC_Timer16bitInterval_Callback_Shared_Interrupt();
    }
    /* Start user code */
    if (_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE == (ITLS0 & _01_ITL_CHANNEL0_COUNT_MATCH_DETECTE))
    {
        ITLS0 &= (uint8_t)~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;
        R_SMC_Timer16bitCapture_Callback_Shared_Interrupt();
    }
#if 0
    /* End user code */
    if (_10_ITL_CAPTURE_COMPLETE_DETECTE == (ITLS0 & _10_ITL_CAPTURE_COMPLETE_DETECTE))
    {
        ITLS0 &= (uint8_t)~_10_ITL_CAPTURE_COMPLETE_DETECTE;
        R_SMC_Timer16bitCapture_Callback_Shared_Interrupt();
    }
    ITLS0 &= (uint8_t)~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;
    /* Start user code */
#endif
    /* End user code */
}
```

Disable original code

Target Application	All.

For the 16-bit capture mode (channel 0, 1; SMC_Timer16bitCapture), when the Smart Configurator generates, capture detection flag (`ITF0C`) is used for the interrupt source and compare match detection flag for channel 0 (`ITF00`) is not used.

But the LoRa®-based software needs to use `ITF00` for the interrupt source instead of `ITF0C`, so changed the generated code to do so.

■ ***Added Code for Interval Timer Interrupt***

File	<ProjectDir>\src\smc_gen\SMC_Timer16bitInterval\SMC_Timer16bitInterval_user.c
Function	R_SMC_Timer16bitInterval_Create_UserInit() * Refer to `R_{Config_ITLn_ITLm}_Create_UserInit` function in [3].

Code	`void R_SMC_Timer16bitInterval_Create_UserInit(void)`
	`{`
	`····/* Start user code for user init. Do not edit comment generated here */`
	`····ITLMKF0 |= _08_ITL_CHANNEL3_COUNT_MATCH_MASK;··// mask unused flag`
	`····/* Start interval timer interrupt */`
	`····R_ITL_Start_Interrupt();`
	`····/* End user code. Do not edit comment generated here */`
	`}`

Target Application	All.

The 16-bit counter mode (channel 2,3; SMC_Timer16bitInterval) is initialized after the16-bit capture mode (channel 0,1; SMC_Timer16bitCapture). Interval timer interrupt needs to be enabled at this time. The code to do so is added to the function described above.

File	<ProjectDir>\src\smc_gen\SMC_Timer16bitCapture\SMC_Timer16bitCapture_user.c
Function	R_SMC_Timer16bitCapture_Callback_Shared_Interrupt() * Refer to `r_{Config_ITLn_ITLm}_Callback_Shared_interrupt` function in [3].

Code	`void R_SMC_Timer16bitCapture_Callback_Shared_Interrupt(void)`
	`{`
	`····g_itlcap_value = ITLCAP00;`
	`····/* Start user code for R_SMC_Timer16bitCapture_Callback_Shared_Interrupt. Do not edit comment generated here */`
	`····RpMcu16bitIntervalTimerIntHandlerCbCapture();`
	`····/* End user code. Do not edit comment generated here */`
	`}`

Target Application	All.

The code to call callback routine when the interrupt is generated by detecting compare match (`ITF00`=1) is added to the function described above.

This function is generated when [Detection of compare match/capture completion (INTITL)] in Interval Timer Setting (16-bit capture mode) is enabled.

File	<ProjectDir>\src\smc_gen\SMC_Timer16bitInterval\SMC_Timer16bitInterval_user.c
Function	R_SMC_Timer16bitInterval_Callback_Shared_Interrupt() * Refer to `r_{Config_ITLn_ITLm}_Callback_Shared_interrupt` function in [3].

Code	`void R_SMC_Timer16bitCapture_Callback_Shared_Interrupt(void)`
	`{`
	`····g_itlcap_value = ITLCAP00;`
	`····/* Start user code for R_SMC_Timer16bitCapture_Callback_Shared_Interrupt. Do not edit comment generated here */`
	`····RpMcu16bitIntervalTimerIntHandlerCbCapture();`
	`····/* End user code. Do not edit comment generated here */`
	`}`

Target Application	All.

The code to call callback routine when the interrupt is generated by detecting compare match (`ITF02`=1) is added to the function described above.

This function is generated when [Detection of compare match/capture completion (INTITL)] in Interval Timer Setting (16-bit capture mode) is enabled.

■ ***Added Code to Set Interval Timer Count Clock for Capturing (fITL1)***

File	<ProjectDir>\src\smc_gen\SMC_Timer16bitCapture\SMC_Timer16bitCapture_user.c
Function	R_SMC_Timer16bitCapture_Create_UserInit() * Refer to `R_{Config_ITLn_ITLm}_Create_UserInit` function in [3].

Code	`void R_SMC_Timer16bitCapture_Create_UserInit(void)`
	`{`
	`    /* Start user code for user init. Do not edit comment generated here */`
	`    ITLMKF0 |= _02_ITL_CHANNEL1_COUNT_MATCH_MASK; // mask unused flag (also capture complete is not used)`
	`    ITLMKF0 |= _01_ITL_CHANNEL0_COUNT_MATCH_MASK; // use compare match (mask it at this time)`
	`    // Set interval timer count clock for capturing (fITL1)`
	`    ITLCSEL0 &= _8F_ITL_CLOCK_FITL1_CLEAR;`
	`    ITLCSEL0 |= _40_ITL_CLOCK_FITL1_FSXP;`
	`    /* End user code. Do not edit comment generated here */`
	`}`

Target Application	All.

The code to set interval timer count clock for capturing (fITL1) using the 16-bit capture mode with channel 2 and 3 is added to the function described above.

3.4.9 Interval Timer Component (SMC_TimerRadioCca)

(1) Setting of the Smart Configurator

The LoRa®-based software uses timer array unit (TAU) for carrier sense (CCA).
This component can be set by clicking [SMC_TimerRadioCca] item in the tree view. TAU setting is sown in Table 21.

Table 21 Interval Timer Setting (TAU0_2 for 16-bit Counter Mode; SMC_TimerRadioCca)

Item		Setting	Note
Clock setting		-	
	Operation clock	CK01	
	Clock source	fCLK/2^8	
Interval timer setting		-	
	Interval value	65536 count	Do not change it.
	Generates INTTM02 when counting is started	OFF	
Interrupt setting		-	
	End of timer channel 2 count, generate an interrupt (INTTM02)	OFF	
	Priority	-	

(2) Modification of the Generated Code

■ ***Added Code to Change TAU Trigger Setting***

File	<ProjectDir>\src\smc_gen\SMC_TimerRadioCca\SMC_TimerRadioCca_user.c
Function	R_SMC_TimerRadioCca_Create_UserInit() * Refer to `R_{Config_TAUm_n}_Create_UserInit` function in [3].

Code	`void R_SMC_TimerRadioCca_Create_UserInit(void)`
	`{`
	`    /* Start user code for user init. Do not edit comment generated here */`
	`    R_TAU0_Set_PowerOff();`
	`    /* End user code. Do not edit comment generated here */`
	`}`

Target Application	All.

The LoRa®-based software uses TAU for carrier sense (CCA). TAU does not need to be started at initialization. So, the code to stop the TAU (clock supply) is added to the function described above.

■ ***Added API Functions***

File	<ProjectDir>\src\smc_gen\SMC_TimerRadioCca\SMC_TimerRadioCca_user.c
Added Functions	void R_SMC_TimerRadioCca_Set_PowerOn(void) void R_SMC_TimerRadioCca_Set_PowerOff(void)

Code	`void R_SMC_TimerRadioCca_Set_PowerOn( void )`
	`{`
	`  R_TAU0_Set_PowerOn();`
	`}`
	
	`void R_SMC_TimerRadioCca_Set_PowerOff( void )`
	`{`
	`  R_TAU0_Set_PowerOff();`
	`}`

Target Application	All.

These functions are to start and stop TAU. These are called from the LoRa®-based software when it starts or stops carrier sense.

File	<ProjectDir>\src\smc_gen\SMC_TimerRadioCca\SMC_TimerRadioCca_user.c
Added Function	bool R_SMC_TimerRadioCca_Is_Running(void)

Code	`bool R_SMC_TimerRadioCca_Is_Running( void )`
	`{`
	`  bool  ret;`
	
	`  if( TAU0EN == 1 )`
	`  {`
	`  ret = true;`
	`  }`
	`  else`
	`  {`
	`  ret = false;`
	`  }`
	
	`  return ret;`
	`}`

Target Application	All.

This function is to check whether TAU is running. It returns true when TAU is running, false otherwise.
It is called from the LoRa®-based software.

File	<ProjectDir>\src\smc_gen\SMC_TimerRadioCca\SMC_TimerRadioCca_user.c
Added Function	uint16_t R_SMC_TimerRadioCca_GetCounterValue(void)

Code	`uint16_t R_SMC_TimerRadioCca_GetCounterValue( void )`
	`{`
	`  uint16_t  ret;`
	
	`  ret = _FFFF_TAU_TDR02_VALUE - TCR02;`
	
	`  return ret;`
	`}`

Target Application	All.

This function returns TAU counter value.

In case of the interval timer mode, count mode of the timer counter register is countdown. But this function calculates the counter value to return it as an up-count.

3.5 Generating Source Files

Output source files for the configuration details by clicking on the [Generate Code] button in the Smart Configurator view.
The Smart Configurator generates a source file in `<ProjectDir>\src\smc_gen` and updates the source file list in the project tree.

Please refer to [1] and/or [2] for the detail of generating source files.

3.5.1 Option Bytes Setting for CS+ / e2 studio

The Smart Configurator asks whether to change the option byte values in the link option setting of CS+/e2 studio according to the configurations of the Smart Configurator. Click [OK] button to accept the changes.

The option values can be confirmed as follows:

(CS+)

[CC-RL Priority] → [Link Options] page → [Device].

(e2 studio)

Right click project in [Project Explorer] → [Properties] → [C/C++ Build] → [Settings] → [Linker] → [Device].

Option bytes setting is shown in Table 22.

Table 22 Option Bytes Setting

MCU	**Option Byte Values**		
	On-chip debug option byte value (-OCDBG)	**Debug monitor area** (-DEBUG_MONITOR)	**User option byte value** (-USER_OPT_BYTE)
RL78/G23-128p	84	BFF00-BFFFF	EF3AAA
RL78/G23-64p		1FF00-1FFFF	
RL78/G22		0FF00-0FFFF	
RL78/L23		7FF00-7FFFF	

(-OCDBG, -DEBUG_MONITOR, and -USER_OPT_BYTE are the options of the rlink command.)

4. How to Change Peripheral Resource

Some peripheral resources can be changed using the Smart Configurator. This section describes how to change resources and points to note.

Note that additional user codes can be protected by inserting between `/* Start user code */` and `/* End user code */`. Please refer to [1] and/or [2] for the detail of the user code protection feature

4.1 Changing Main System Clock to 32MHz

To change the frequency of the high-speed on-chip oscillator from 8MHz to 32MHz, change the clock setting (Table 23).

Table 23 Clock Settings to Change Frequency of the High-Speed On-Chip Oscillator to 32MHz

Item	Setting	Note
Operation mode	High-speed main mode 1.8(V) ~ 5.5(V)	
EVDD setting	1.8V ≤ EVDD0 ≤ 5.5V	RL78/G23 only
High-speed on-chip oscillator	ON	
Frequency	32 (MHz)	
fHOCO start setting	Normal	

(Note) Other than the above settings are not necessary to change.

When the main system clock is changed to 32MHz, the CPU / peripheral hardware clock frequency (fCLK) is also changed to 32MHz. So, the following component settings which use the fCLK as clock source also need to be changed (Table 24).

Table 24 Component Settings to be Changed When Frequency of the High-Speed On-Chip Oscillator is 32MHz

Component	Config name	Item	Setting
UART Communication (RL78/G2x only)	SMC_UART	Clock source (in [Transmission] page)	fCLK/2
		Clock source (in [Reception] page)	fCLK/2
IIC Communication (Master mode)	SMC_IIC	Clock mode setting (RL78/G23) Clock source (RL78/G22, L23)	fCLK/2
Interval Timer	SMC_TimerRadioCca	Clock source	fCLK/2^10

Note

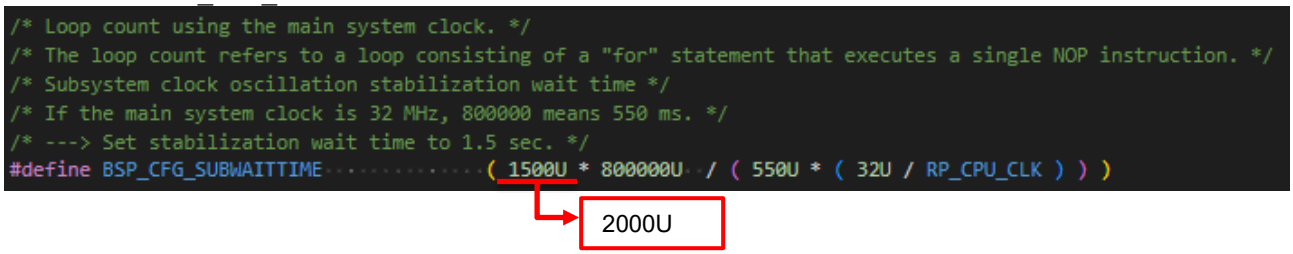
When frequency of the high-speed on-chip oscillator is set to 32MHz, please change the macro `RP_CPU_CLK` from 8 to 32. It can be changed on the CC-RL compile option setting in CS+/e2 studio.

When [Generate Code] button is clicked, user option byte value is changed to EF3AE8.

4.2 Changing Subsystem Clock Oscillation Stabilization time

This setting can be changed by changing the value of `BSP_CFG_SUBWAITTIME` macro. It is defined in `<ProjectDir>\src\smc_gen\r_config\r_bsp_config.h`.

For example, when the subsystem clock oscillation stabilization wait time is changed to 2.0 seconds, change the value of `BSP_CFG_SUBWAITTIME` macro as follow:



4.3 Changing Port and INTP assignment

Port assignments can be changed according to the user hardware system in the Smart Configurator. When the assignment of the port which has the symbolic name (Table 25) will be changed, move the symbolic name to the changed port.
Note that the symbolic names are referred by the LoRa®-based software. So, do not change the symbolic names.

Table 25 Ports Which Have Symbolic Name

Symbolic Name	I/O	Port Setting			
		RL78/G23-128p FPB (*2)	RL78/G23-64p FPB (*2)	RL78/G22 FPB (*2)	RL78/L23 FPB (*2)
SMC_PIN_ANT_SWICH_POWER	O	P42	P73	P17	P81
SMC_PIN_RADIO_BUSY	I	P106	P42	P31	P01
SMC_PIN_RADIO_DEVICE_SEL	I	P117	P24	P26	P25
SMC_PIN_RADIO_DIO_1 (*1)	I	P77 / INTP11	P77 / INTP11	P140 / INTP6	P02 / INTP7
SMC_PIN_RADIO_NSS	O	P46	P76	P146	P80
SMC_PIN_RADIO_RESET	O	P115	P22	P147	P27
SMC_PIN_RADIO_XTAL_SEL	I	P147	P25	P25	P24
(SMC_PIN_BOARD_SW) (*1)	I	P137 / INTP0	P137 / INTP0	P137 / INTP0	P137 / INTP0

(*1) Need to assign the INTP port.
(*2) FPB stands for Fast Prototyping Board.

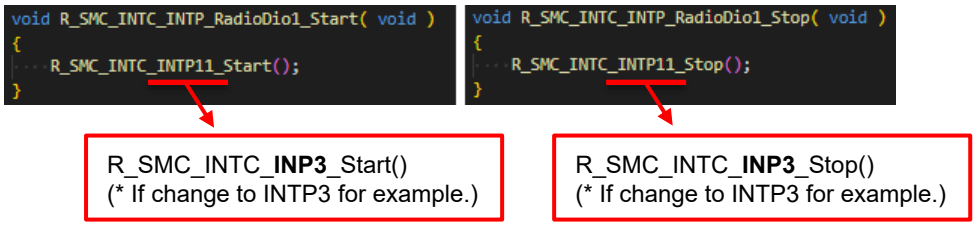
If you change the port to which INTP is assigned, you also need to change the setting of the interrupt controller component to reassign the INTP on the interrupt controller component view in the Smart Configurator.

If the INTP assignment is changed from current, please change the following code:

■ ***SMC_PIN_RADIO_DIO_1 (INTP11, INTP6, INTP7)***

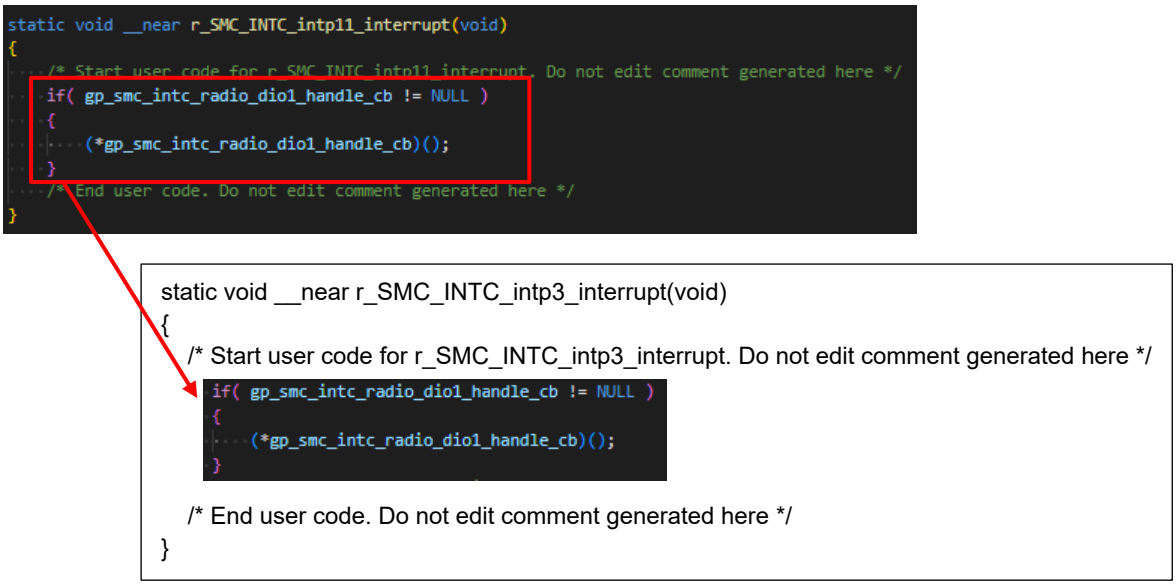
File	<ProjectDir>\src\smc_gen\SMC_INTC\SMC_INTC_user.c
Added Functions	void R_SMC_INTC_INTP_RadioDio1_Start(void) void R_SMC_INTC_INTP_RadioDio1_Stop(void)
Target Application	All.

Please replace with the start / stop API functions corresponding to the changed INTP channel number.



File	<ProjectDir>\src\smc_gen\SMC_INTC\SMC_INTC_user.c
Functions	r_SMC_INTC_intp*x*_interrupt() → r_SMC_INTC_intp*y*_interrupt() * *x* = 6, 7 or 11, y = changed INTP channel number * Refer to `r_{Config_INTC}_intpn_interrupt` function in [3].
Target Application	All.

Please move the function contents into the new interrupt function corresponding to the INTP channel number.

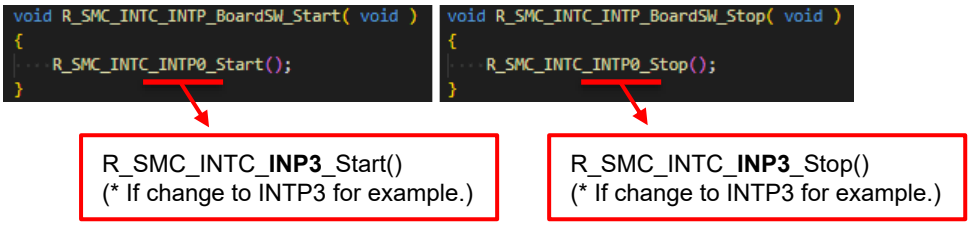


(* If change to INTP3 for example.)

■ ***SMC_PIN_BOARD_SW (INTP0)***

File	<ProjectDir>\src\smc_gen\SMC_INTC\SMC_INTC_user.c
Added Functions	void R_SMC_INTC_INTP_BoardSW_Start(void) void R_SMC_INTC_INTP_BoardSW_Stop(void)
Target Application	LoRaSample, LoRaFuotaSample, LoRaFuotaSample_BankSwap, LoRaSensorSample, and LoRaWanPrivateLoRaComboSample.

Please replace with the start / stop API functions corresponding to the changed INTP channel number.



File	<ProjectDir>\src\smc_gen\SMC_INTC\SMC_INTC_user.c
Functions	r_SMC_INTC_intp0_interrupt() → r_SMC_INTC_intp*y*_interrupt() * *y* = changed INTP channel number. * Refer to `r_{Config_INTC}_intp`*n*`_interrupt` function in [3].
Target Application	LoRaSample, LoRaFuotaSample, LoRaFuotaSample_BankSwap, LoRaSensorSample, and LoRaWanPrivateLoRaComboSample.

Please move the function contents into the new interrupt function corresponding to the changed INTP channel number.

(* If change to INTP3 for example.)

4.4 Changing UART Channel

UART (serial array unit) channel can be changed according to the user hardware system in the Smart Configurator by:

Right click the [SMC_UART] in the tree view → Select [Change resource] → Select Resource (Figure 5)

Note that changing from UART (serial array unit) to UARTA (serial interface UARTA) and UARTA to UART are not described in this section.

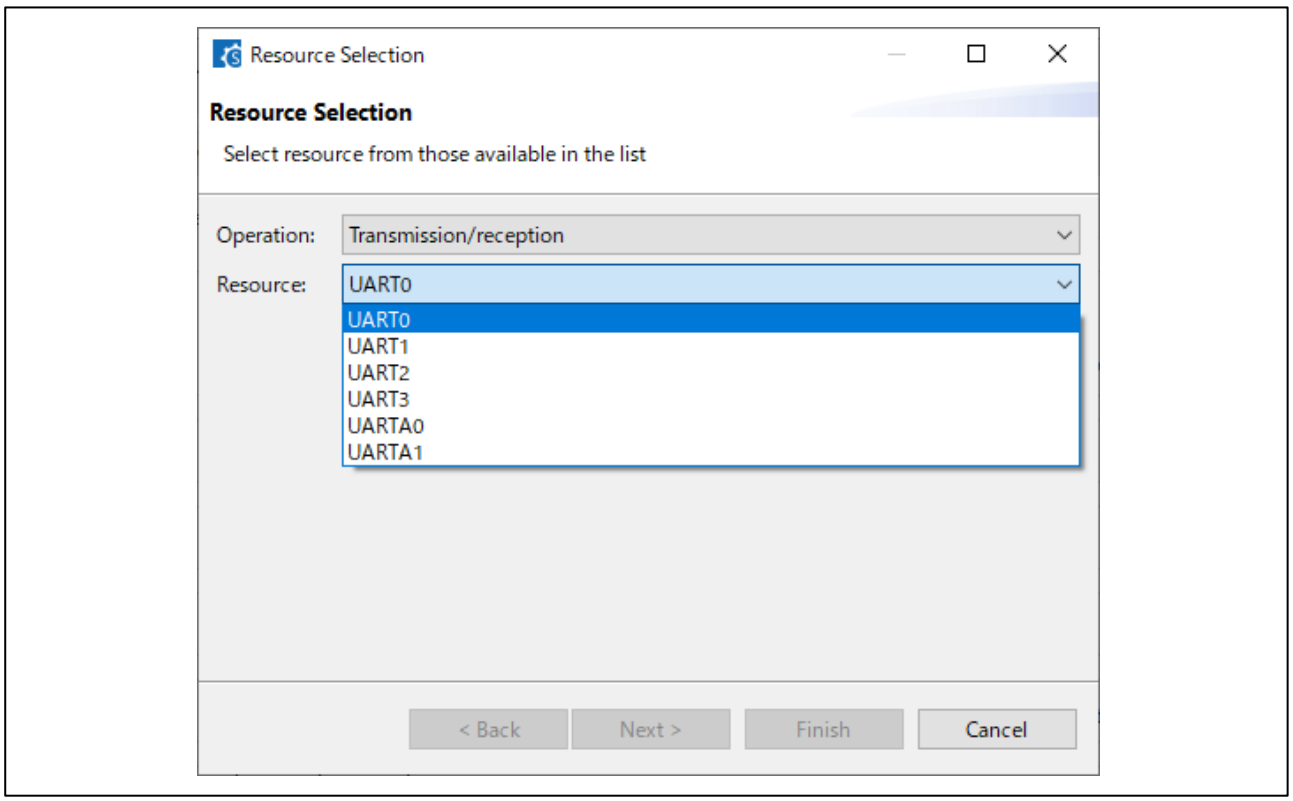


Figure 5 Change the UART Channel

When UART (serial array unit) channel is changed, please also confirm the setting of ports which selected UART channel uses.

When [Generate Code] button is clicked, modified code (see section 0) is protected. So, no change to generated code required.

4.5 Changing SPI (CSI) Channel

SPI (CSI) channel can be changed according to the user hardware system in the Smart Configurator by:
Right click the [SMC_CSI] in the tree view → Select [Change resource] → Select Resource (Figure 6)

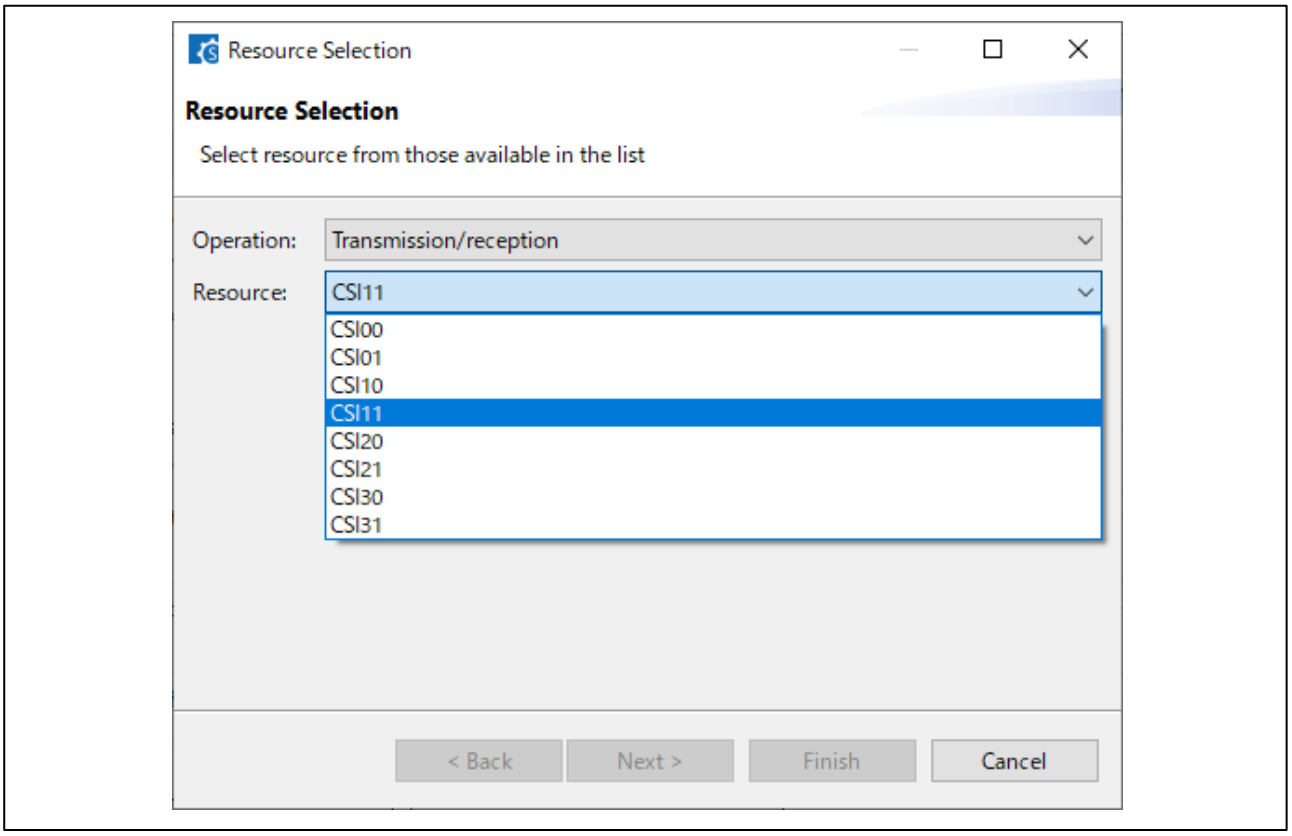


Figure 6 Change the SPI (CSI) Channel

When SPI (CSI) channel is changed, please also confirm the setting of ports which selected SPI (CSI) channel uses.

When [Generate Code] button is clicked, modified code (see section 3.4.6(2)) is protected. So, no change to generated code required.

4.6 Changing IIC Channel

When using RL78/G23, IICA (serial interface IICA) channel can be changed according to the user hardware system in the Smart Configurator by:

Right click the [SMC_IIC] in the tree view → Select [Change resource] → Select Resource (Figure 7)

When [Generate Code] button is clicked, modified code (see section 0) is protected. So, no change to generated code required.

Note that changing to IIC (serial array unit; simplified IIC) is not supported.

When using RL78/G22 and RL78/L23, IIC (serial array unit; simplified IIC) channel can be changed according to the user hardware system in the Smart Configurator by:

Right click the [SMC_IIC] in the tree view → Select [Change resource] → Select Resource (Figure 7)

When [Generate Code] button is clicked, modified code (see section 0) is protected. So, no change to generated code required.

Note that changing to IICA (serial interface IICA) is not supported.

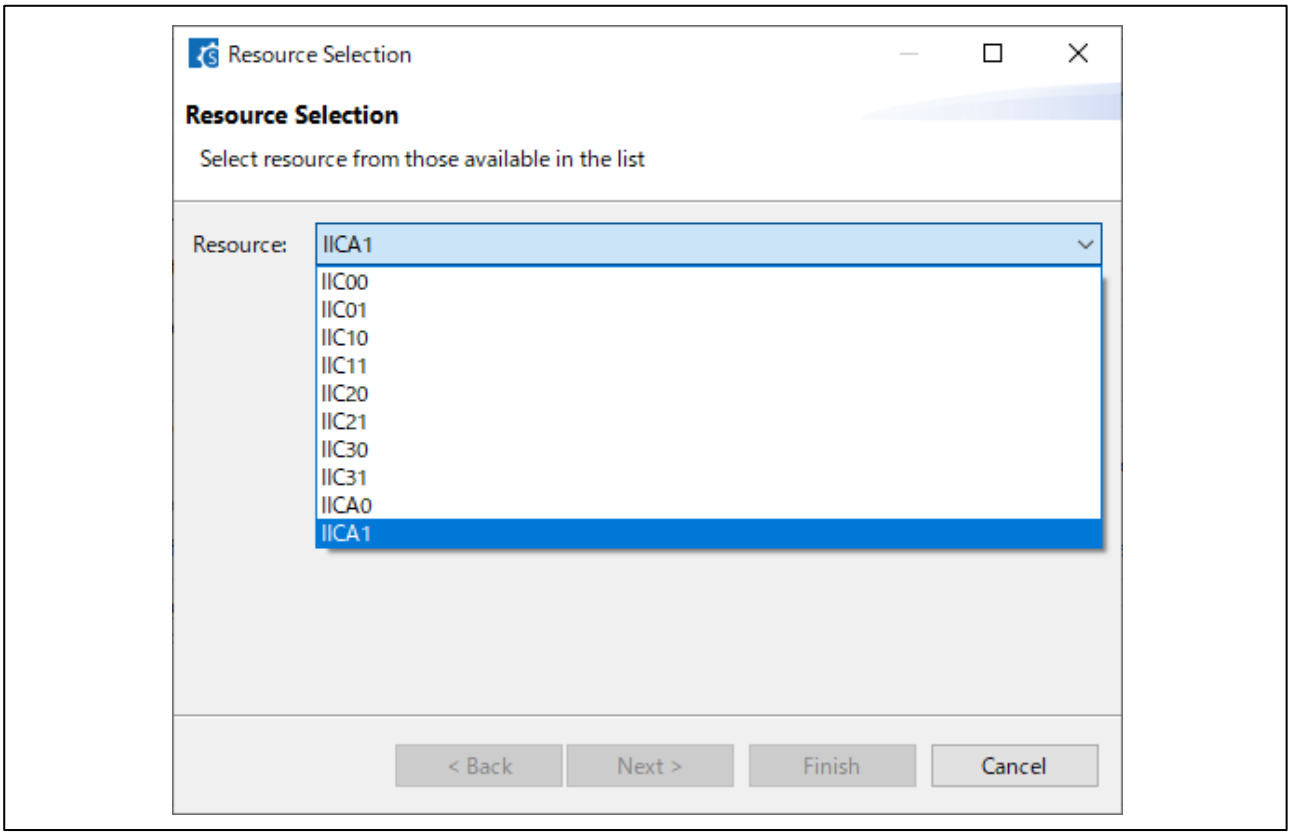


Figure 7 Change the IIC Channel

4.7 Changing TAU Unit and Channel for Radio CCA

The unit and channel of TAU (timer array unit) can be changed in the Smart Configurator by:
Right click the [SMC_RadioCca] in the tree view → Select [Change resource]
→ Select Resource (Figure 8)

When [Generate Code] button is clicked, modified code (see section3.4.9(2)) is protected. But some code changes are required.

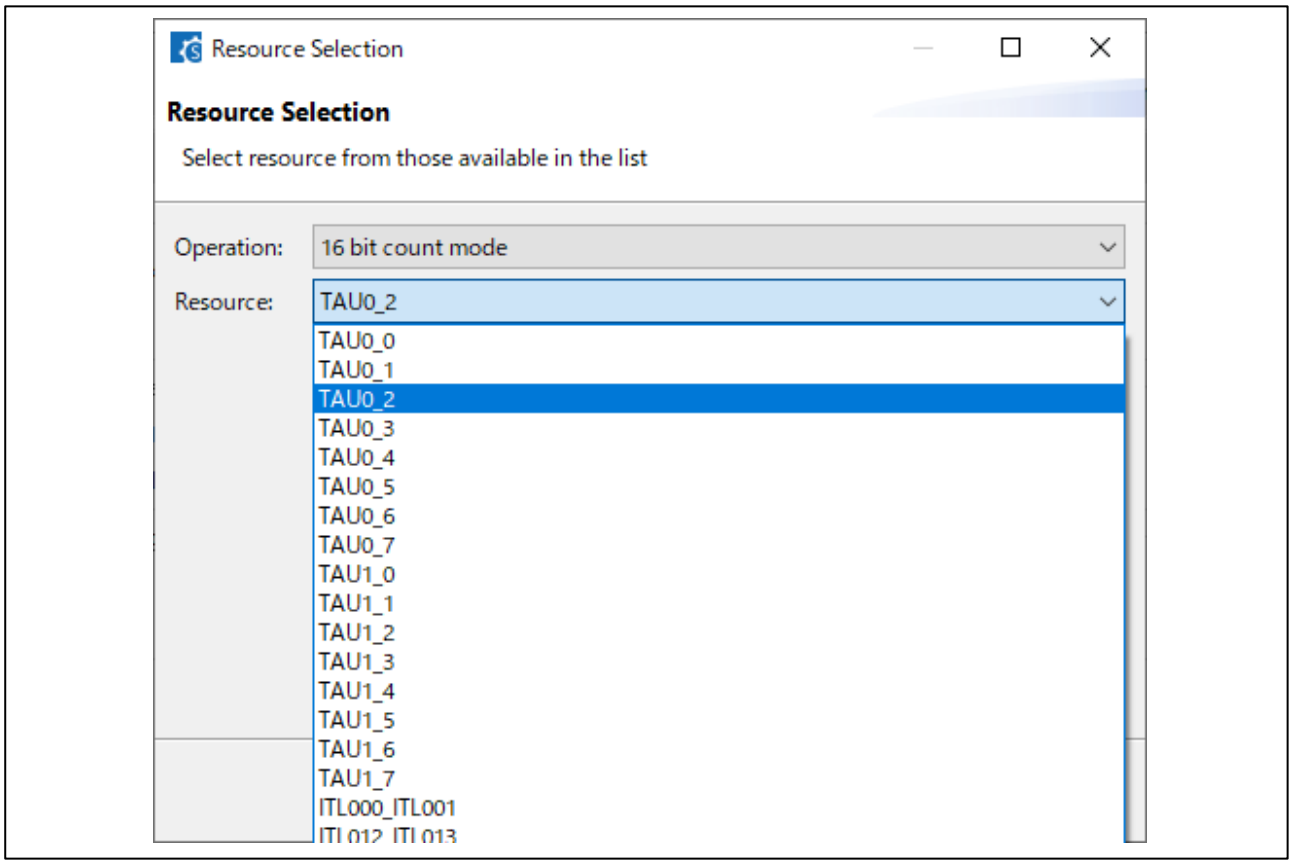


Figure 8 Change the TAU Unit and Channel

File	<ProjectDir>\src\smc_gen\SMC_TimerRadioCca\SMC_TimerRadioCca_user.c
Function	R_SMC_TimerRadioCca_Create_UserInit() * Refer to `R_{Config_TAUm_n}_Create_UserInit` function in [3].
Target Application	All.

File	<ProjectDir>\src\smc_gen\SMC_TimerRadioCca\SMC_TimerRadioCca_user.c
Added Functions	void R_SMC_TimerRadioCca_Set_PowerOn(void) void R_SMC_TimerRadioCca_Set_PowerOff(void)
Target Application	All.

`R_TAU0_Set_PowerOn()` and `R_TAU0_Set_PowerOff()` functions are called from the above functions. When the unit number is changed to 1, please change these function name to `R_TAU1_Set_PowerOn()` and `R_TAU1_Set_PowerOff()`.

File	<ProjectDir>\src\smc_gen\SMC_TimerRadioCca\SMC_TimerRadioCca_user.c
Added Function	bool R_SMC_TimerRadioCca_Is_Running(void)
Target Application	All.

This function refers to the TAU0EN register bit. When the unit number is changed to 1, please change it to TAU1EN.

File	<ProjectDir>\src\smc_gen\SMC_TimerRadioCca\SMC_TimerRadioCca_user.c
Added Function	uint16_t R_SMC_TimerRadioCca_GetCounterValue(void)
Target Application	All.

This function refers the timer count register "TCR02" and the macro "_FFFF_TAU_TDR02_VALUE". When the unit number and/or channel number are changed, please change the following items:

- Timer count register: TCR02 ---> TCR*mn*
- Macro name: _FFFF_TAU_TDR02_VALUE ---> _FFFF_TAU_TDR*mn*_VALUE

(*m* is unit number, and *n* is channel number)

5. Changing of Peripheral Modules for Function Expansion

5.1 [RL78/L23] UART Reception in the STOP Mode Using the UARTA

Operation of UARTA is possible even while MCU is in the STOP mode if fSXP (32.768kHz sub clock) is selected as the operation clock of UARTA. When a byte is received, MCU will wake up by UARTA receive interrupt. Note that, in this case, the maximum baud rate is 2400bps.

This feature is supported for all the sample applications except for ping-pong.

To do this, change the UARTA0 setting (Table 26).

Table 26 UARTA0 Setting to Receive in the STOP Mode (RL78/L23)

Item		Setting	Note
UARTA0 clock setting		-	
	Operation clock	fSXP	
	ELCL clock	-	
Transfer rate setting		-	
	Transfer rate setting	2400 (bps)	

(Note) Other than the above settings are not necessary to change.

To use this feature, the macro “ENABLE_UARTRX_IN_SLEEP_SUBCLK” needs to be specified in the project build option (undefined by default).

Note: The macro “ENABLE_UARTRX_IN_SLEEP_SUBCLK” can be specified for RL78/L23 only.

Revision History

Rev.	Date	Description	
		Section	Summary
04.60	Sep.27.24	-	Initial Release
04.80	Aug.21.25	All	Supports RL78/L23.
			Updates tool version.
		1.1	Updates related documentation [7].
		5	Added chapter to describe the setting for function expansion.
04.90	Nov.28.25	1	Updates tool version.

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

2. Processing at power-on

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

3. Input of signal during power-off state

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

4. Handling of unused pins

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

5. Clock signals

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

6. Voltage application waveform at input pin

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

7. Prohibition of access to reserved addresses

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

8. Differences between products

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

Notice

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

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. 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

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/.

Trademarks

Arm® and Cortex® are registered trademarks of Arm Limited. Semtech, the Semtech logo, LoRa, LoRaWAN and LoRa Alliance are registered trademarks or service marks, or trademarks or service marks, of Semtech Corporation and/or its affiliates. Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.