RL78/G15 Group and RL78/G16 Group

Sample program for flash memory reprogramming

Renesas Flash Sample Program Type 01

R20AN0652EJ0120
Rev.1.20
Sep.25.2023

Introduction

This application note is intended to give users an understanding of the methods for using the Renesas Flash Sample Program Type 01(RFSP Type 01) for RL78/G15 group and RL78/G16 group. And this application note is intended for engineers who develop application systems using the RFSP Type 01.

Target devices

RL78/G15 Group
RL78/G16 Group

Table of contents

1 Overview

Renesas Flash Sample Program Type 01 (hereinafter referred to as "RFSP Type 01" in this document.) is the sample program for reprogramming the data in flash memory of RL78/G15 and RL78/G16.

1.1 Outline

The functions of RFSP Type 01 are called from the user program to reprogram the code flash memory or data flash memory.
In addition, please use this user's manual together with the user's manual of a target device.

1.1.1 Purpose

The purpose of this document is to describe information about RFSP Type 01.

1.2 Contents

The API functions of RFSP Type 01 are called from the user program to reprogram the code flash memory or data flash memory.

The RFSP Type 01 package includes the following:

- This application notes.
- Sample program files of RFSP Type 01 for controlling for code flash memory incorporated in the RL78/G15 and RL78/G16.
- Sample program files of RFSP Type 01 for controlling for data flash memory incorporated in the RL78/G15 and RL78/G16.

1.3 Features

The RFSP Type 01 reprograms the flash memory according to the specified flow of command processing for the flash memory control circuit. Each sample API function of RFSP Type 01 consists of a single sub-function or two or more sub-functions, and the necessary processing is implemented by combinations of individual sub-functions and user processing.

Figure 1-1 shows the flash memory control by the user application using the sample API functions of RFSP Type 01.

RFSP Type 01 provides sample programs of the processing that is implemented by combinations of two or more sample API functions and user programs. Refer to the sample programs when embedding the flash memory control processing in the user application.

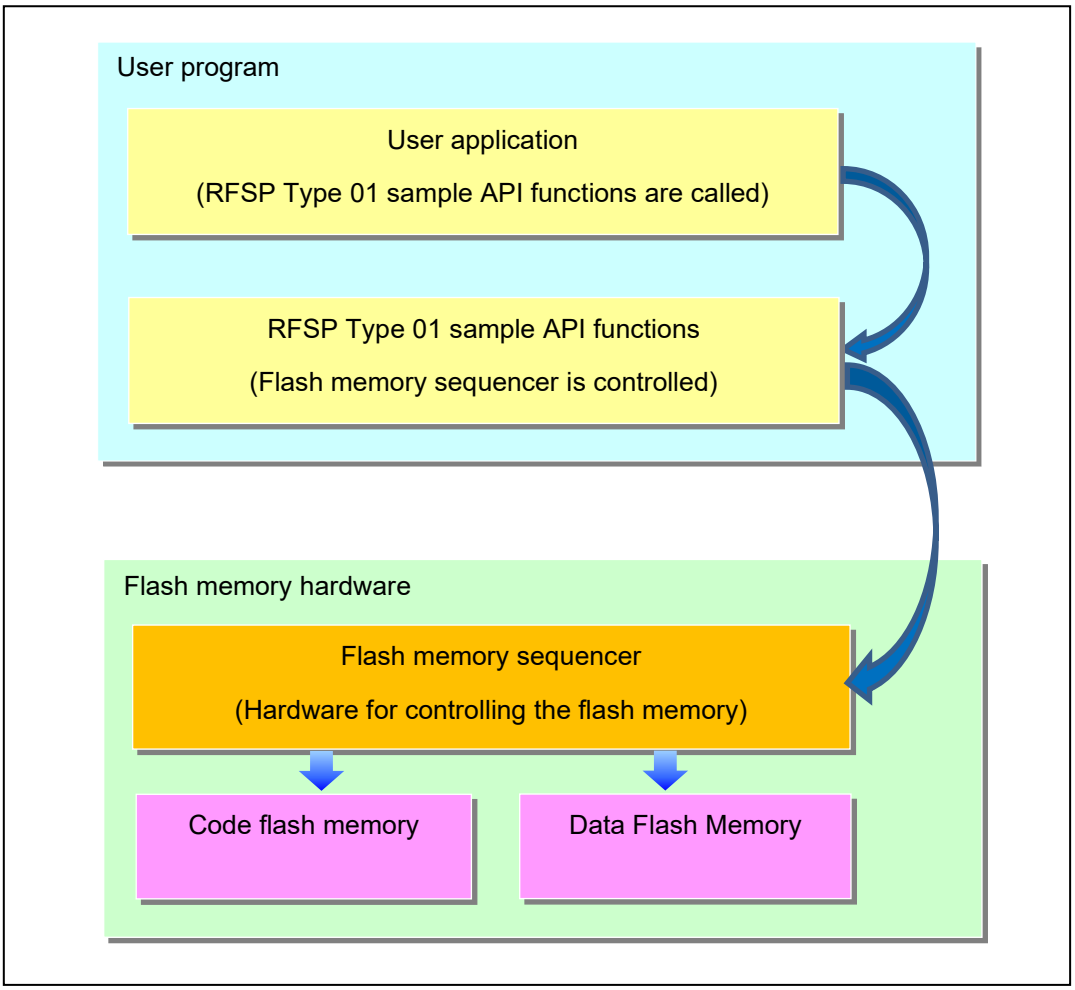


Figure 1-1 Flash Memory Control Using Sample API Functions of RFSP Type 01

1.4 Operating environment

- Host Computer
  The operation of RFSP Type 01 does not depend on the host computer but the appropriate environment for the C compiler package, debugger and emulator must be prepared. (RFSP Type 01 was developed and tested on Windows10 Pro.)

- C compiler package
  Table 1-1 shows the target C compiler packages for RFSP Type 01.

Table 1-1 the target C Compiler Packages for RFSP Type 01

Package	IDE (Integrated Development Environment)	Manufacturer	Version
CC-RL compiler	CS+, e^2 studio	Renesas Electronics	V1.10 or later
IAR compiler	IAR Embedded Workbench® for Renesas RL78	IAR Systems®	V4.21 or later
LLVM compiler	e^2 studio	Open Source Software	V10.0.0.202306 or later

Note. Integrated development environment and compiler must support the target device.

- Emulator
  Table 1-2 shows the emulator on which the operation of RFSP Type 01 was confirmed.

Table 1-2 Emulator on which RFSP Type 01 Operation was Confirmed

Emulator	Manufacturer
E2 Emulator Lite	Renesas Electronics

- Target MCU
  RL78/G15
  RL78/G16

1.5 Points for Caution

(1) Allocation of the user program for flash memory reprogramming operation

Allocate the user program for programming the code/data flash area to the code flash area. Self-programming by fetching from the RAM is prohibited. Additionally, reprogramming the boot area and the block for storing the user program for executing self-programming is prohibited.

(2) Prohibit the interrupts in self-programming mode

Prohibit an interrupt before setting the self-programming mode. To prohibit an interrupt, clear (0) the IE flag by the DI instruction in the same way as in the normal operation mode.

(3) Setting the CPU operating frequency for the flash memory sequencer

When using the flash memory sequencer to reprogram the code/data flash memory, set the value corresponding to the CPU operating frequency in the FSET4-0 bits of the FSSET register before proceeding. Note that if reprogramming is attempted while the value corresponding to the CPU operating frequency is not correct, operation is undefined and written data are not guaranteed. Even if the values in the flash memory are as expected immediately after reprogramming, retaining the values for any specified period is not guaranteed.

(4) Operation setting of high-speed on-chip oscillator

The high-speed on-chip oscillator should be kept operating before executing self-programming. If it is stopped, it should be made to operate again (HIOSTOP = 0), and the flash self-programming code should be re-executed after 30 us have elapsed.

(5) Restriction of execution of other operations during self-programming

Do not execute other settings or instructions which are not related to the self-programming procedure during the self-programming execution flow

(6) User program operation during flash memory reprogramming operation

The CPU is stopped during reprogramming through self-programming. The code flash or data flash memory cannot be accessed while it is being reprogrammed.

1.6 C Compiler Definitions

The definitions of the target compiler written in the header file (r_rfsp_compiler.h) for RFSP Type 01 are shown below.

The definitions differ between compilers. The "r_rfsp_compiler.h" file is used to identify the current compiler and the definitions for the target compiler are used.

- Definition of C compiler

  - Definition of CC-RL compiler:

    "__CCRL__" is defined

    #define COMPILER_CC (1)

  - IAR compiler:

    ” __IAR_SYSTEMS_ICC__” is defined

    #define COMPILER_IAR (2)

  - Definition of LLVM compiler:

    "__llvm__" is defined

    #define COMPILER_LLVM (3)

<Descriptions in the r_rfsp_compiler.h file>

```
/* Compiler definition */
#define COMPILER_CC                 (1)
#define COMPILER_IAR                (2)
#define COMPILER_LLVM               (3)

#if defined (__llvm__)
   #define COMPILER COMPILER_LLVM
#elif defined (__CCRL__)
   #define COMPILER COMPILER_CC
#elif defined (__IAR_SYSTEMS_ICC__)
   #define COMPILER COMPILER_IAR
#else
   /* Unknown compiler error */
   #error  "Non-supported compiler."
#endif

/* Compiler dependent definition */
#if (COMPILER_CC == COMPILER)
   #define R_RFSP_FAR_FUNC                          __far
   #define R_RFSP_NO_OPERATION                      __nop
   #define R_RFSP_DISABLE_INTERRUPT                 __DI
   #define R_RFSP_ENABLE_INTERRUPT                  __EI
   #define R_RFSP_GET_PSW_IE_STATE                  __get_psw
   #define R_RFSP_IS_PSW_IE_ENABLE(u08_psw_ie_state) (0u != ((u08_psw_ie_state) & 0x80u))
#elif (COMPILER_IAR == COMPILER)
   #define R_RFSP_FAR_FUNC                          __far_func
   #define R_RFSP_NO_OPERATION                      __no_operation
   #define R_RFSP_DISABLE_INTERRUPT                 __disable_interrupt
   #define R_RFSP_ENABLE_INTERRUPT                  __enable_interrupt
   #define R_RFSP_GET_PSW_IE_STATE                  __get_interrupt_state
   #define R_RFSP_IS_PSW_IE_ENABLE(u08_psw_ie_state) (0u != ((u08_psw_ie_state) & 0x80u))
```

```
#elif (COMPILER_LLVM == COMPILER)
   #define R_RFSP_FAR_FUNC                          __far
   #define R_RFSP_NO_OPERATION                      __nop
   #define R_RFSP_DISABLE_INTERRUPT                 __DI
   #define R_RFSP_ENABLE_INTERRUPT                  __EI
   #define R_RFSP_GET_PSW_IE_STATE                 (uint8_t)__builtin_rl78_pswie
   #define R_RFSP_IS_PSW_IE_ENABLE(u08_psw_ie_state) (0u != (u08_psw_ie_state))
#else
   /* Unknown compiler error */
   #error  "Non-supported compiler."
#endif
```

- C compiler options

  The contents of the C compiler option setup which normal operation can be checking are shown below.

  - [CC-RL(CS+)]

    Major compile options:

    -cpu=S2 -g -g_line -lang=c99

  - [IAR(IAR Embedded Workbench)]

    Major compile options:

    --core s2 --calling_convention v2 --code_model far --data_model near -e -Ol --no_cse --no_unroll --no_inline --no_code_motion --no_tbaa --no_cross_call --no_scheduling --no_clustering --debug

  - [LLVM(e^2 studio)]

    Major compile options:

    -Og -ffunction-sections -fdata-sections -fdiagnostics-parseable-fixits -Wunused -Wuninitialized -Wall -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wshadow -Waggregate-return -g -mcpu=s2 -mdisable-mda

2 System configuration

2.1 File Structure

2.1.1 Folder Structure

This section is explained in the sample program example for RL78/G15. When using a device other than RL78/G15, read G15 to the target device.

- Please read and change the folder name ("RL78_G15") of the sample of RL78/G15 into the folder name of a target device.
  The folder name in the case of using RL78/G16: "RL78_G16"

Figure 2-1 shows the folder structure of RFSP type 01.

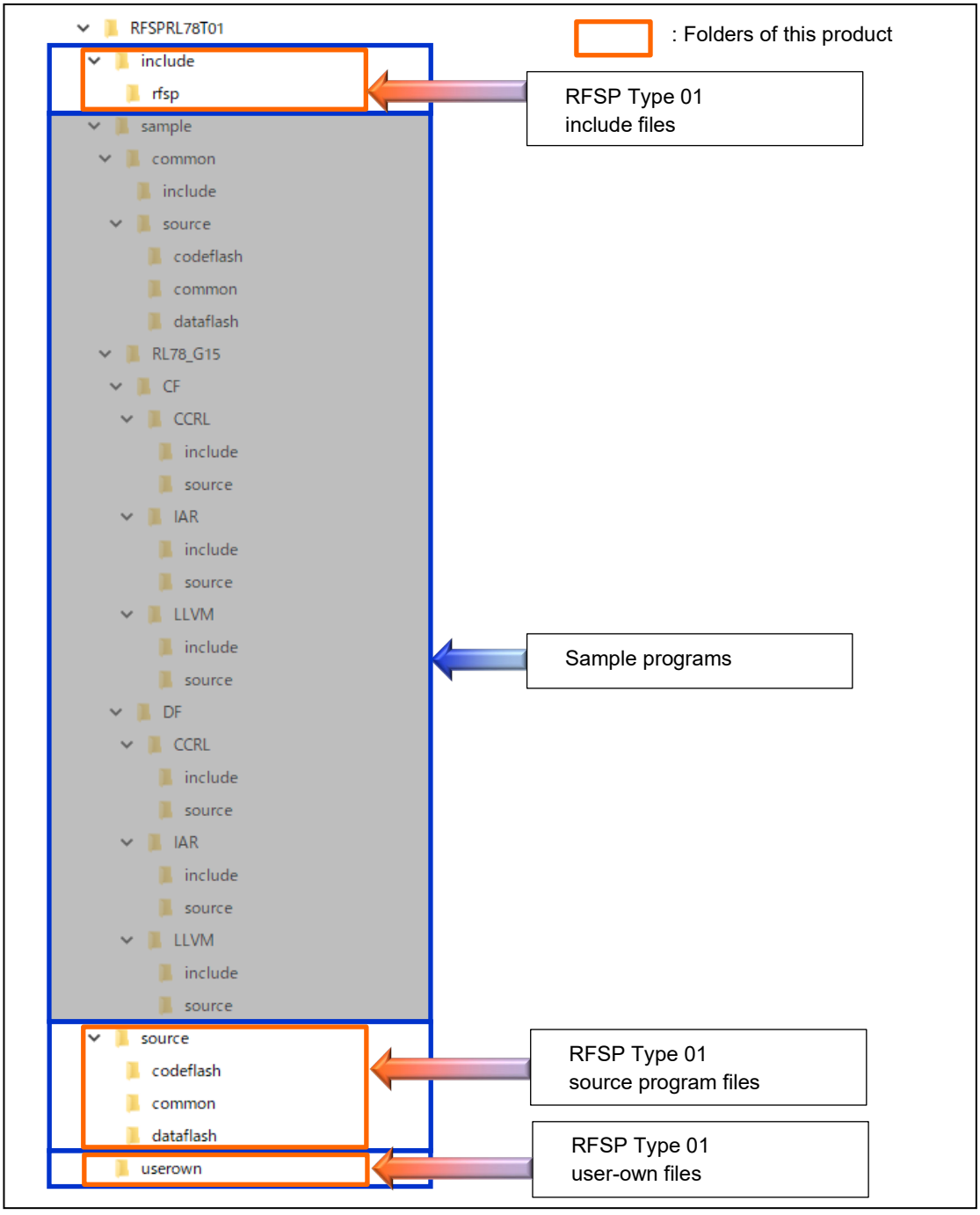


Figure 2-1 Folder Structure of RFSP Type 01

2.1.2 List of Files

2.1.2.1 List of Source files

Table 2-1 shows the program source files in the “source\common\” folder.

Table 2-1 Program Source Files in the "source\common\" Folder

No	Source File Name	Description
1	r_rfsp_common_api.c	This file contains the API functions for settings used in common for flash memory control.
2	r_rfsp_common_control_api.c	This file contains the API functions for command control used in common for flash memory control.

Table 2-2 shows the program source file in the “source\codeflash\” folder.

Table 2-2 Program Source File in the "source\codeflash\" Folder

No	Source File Name	Description
1	r_rfsp_code_flash_api.c	This file contains the API functions for code flash memory control.

Table 2-3 shows the program source file in the “source\dataflash\” folder.

Table 2-3 Program Source File in the "source\dataflash\" Folder

No	Source File Name	Description
1	r_rfsp_data_flash_api.c	This file contains the API functions for data flash memory control.

Table 2-4 shows the program source file in the “userown\” folder.

Table 2-4 Program Source File in the "userown\" Folder

No	Source File Name	Description
1	r_rfsp_common_userown.c	This file contains the hook functions for user processing to be performed in RFSP Type 01

2.1.2.2 List of Header File

Table 2-5 shows the program header files in the “include\rfsp\” folder.

Table 2-5 Program Header Files in the "include\rfsp\" Folder

No	Header File Name	Description
1	r_rfsp.h	Common header file. This file needs to be included when RFSP Type 01 is used.
2	r_rfsp_compiler.h	This file describes the definitions that differ between compilers used in RFSP Type 01
3	r_rfsp_device.h	This file defines the hardware-specific macros used in RFSP Type 01
4	r_rfsp_types.h	This file defines the types of variables used in RFSP Type 01
5	r_typedefs.h	This file defines the types of data used in RFSP Type 01

Table 2-6 shows the program header files in the “include\” folder.

Table 2-6 Program Header Files in the "include\" Folder

No	Header File Name	Description
1	r_rfsp_code_flash_api.h	This file defines the prototype declarations of the API functions for code flash memory control.
2	r_rfsp_common_api.h	This file defines the prototype declarations of the API functions for setting used in common for flash memory control.
3	r_rfsp_common_control_api.h	This file defines the prototype declarations of the API functions for command control used in common for flash memory control.
4	r_rfsp_common_userown.h	This file defines the prototype declarations of the hook functions for user processing to be performed in RFSP Type 01
5	r_rfsp_data_flash_api.h	This file defines the prototype declarations of the API functions for data flash memory control.

2.2 Resources of RL78/G15, RL78/G16

2.2.1 Memory Map

Table 2-7 shows the memory map (code flash memory (CF), data flash memory (DF), and RAM) of the RL78/G15

Table 2-7 Memory Map for RL78/G15 (ROM, Data Flash, and RAM)

RL78/G15	Code Flash: CF (1 block = 1Kbyte)	RAM
R5F120x8 (x=0,1,4,6)	8KB (0000H-1FFFH)	1KB (FFB00H-FFEFFH)
R5F120x7 (x=0,1,4,6)	4KB (0000H-0FFFH)	1KB (FFB00H-FFEFFH)
All RL78/G15 devices	**Data Flash Memory : DF (1 block = 512byte)**	
	1 Kbytes (9000H-93FFH)	

Table 2-8 shows the memory map (code flash memory (CF), data flash memory (DF), and RAM) of the RL78/G16

Table 2-8 Memory Map for RL78/G16 (ROM, Data Flash, and RAM)

RL78/G16	Code Flash: CF (1 block = 1Kbyte)	RAM
R5F121xC (x=1,4,6,7,B)	32KB (0000H-7FFFH)	2KB (FF700H-FFEFFH)
R5F121xA (x=1,4,6,7,B)	16KB (0000H-3FFFH)	2KB (FF700H-FFEFFH)
All RL78/G16 devices	**Data Flash Memory : DF (1 block = 512byte)**	
	1 Kbytes (9000H-93FFH)	

2.2.2 **Block Images**

Figure 2-2 and Figure 2-3 shows the allocation of blocks in code flash memory (CF) and data flash memory (DF) for RL78/G15. Refer to the user's manual of a target device for allocation of blocks for other devices.

R5F120x8 (Code flash memory: 8Kbytes)

1FFFH	CF: Block 07H
1C00H	(1Kbyte)
1BFFH	CF: Block 06H
1800H	(1Kbyte)
17FFH	CF: Block 05H
1400H	(1Kbyte)
13FFH	CF: Block 04H
1000H	(1Kbyte)
0FFFH	CF: Block 03H
0C00H	(1Kbyte)
0BFFH	CF: Block 02H
0800H	(1Kbyte)
07FFH	CF: Block 01H
0400H	(1Kbyte)
03FFH	CF: Block 00H
0000H	(1Kbyte)

R5F120x7 (Code flash memory: 4Kbytes)

0FFFH	CF: Block 03H
0C00H	(1Kbyte)
0BFFH	CF: Block 02H
0800H	(1Kbyte)
07FFH	CF: Block 01H
0400H	(1Kbyte)
03FFH	CF: Block 00H
0000H	(1Kbyte)

Figure 2-2 Blocks in the Code Flash Memory

93FFH	DF: Block 01H
9200H	(512bytes)
91FFH	DF: Block 00H
9000H	(512bytes)

Figure 2-3 Blocks in the Data Flash Memory

2.2.3 List of Registers Related to Flash Memory Sequencer Control

Table 2-9 shows the registers in the RL78/G15 and RL78/G16 used by RFSP Type 01.

Table 2-9 Registers in the RL78/G15 and RL78/G16 Used by RFSP Type 01

Base Address	Offset	Register Name	Size	Function name / Note
F0000H	BEH	FSSET	1byte	Flash memory sequencer frequency setting register
	C0H	FLPMC	1byte	Flash programming mode control register
	C1H	FSSQ	1byte	Flash memory sequencer control register
	C2H	FLAPL	1byte	Flash address pointer register L
	C3H	FLAPH	1byte	Flash address pointer register H
	C4H	FLSEDL	1byte	Flash end address pointer L
	C5H	FLSEDH	1byte	Flash end address pointer H
	C6H	FSASTL	1byte	Flash sequencer status register L
	C7H	FSASTH	1byte	Flash sequencer status register H
	C8H	FLWLL	1byte	Flash write buffer register LL
	C9H	FLWLH	1byte	Flash write buffer register LH
	CAH	FLWHL	1byte	Flash write buffer register HL
	CBH	FLWHH	1byte	Flash write buffer register HH

2.3 Resources Used in RFSP Type 01

2.3.1 Code Size and Stack Size which API Functions Use

Table 2-10 shows code size and stack size which API functions for RFSP Type 01 use.

Table 2-10 Code Size and Stack Sizes which API Functions for RFSP Type 01 Use

API Name	Code Size (Bytes)			Stack Size (Bytes)		
	CC-RL	IAR	LLVM	CC-RL	IAR	LLVM
R_RFSP_Init	16	21	16	4	4	4
R_RFSP_SetFlashMemoryMode	21	29	31	4	8	8
R_RFSP_CheckCFDFSeqEndStep1	13	24	16	4	6	4
R_RFSP_CheckCFDFSeqEndStep2	8	19	11	4	6	4
R_RFSP_GetSeqErrorStatus	8	8	11	4	4	6
R_RFSP_ForceReset	2	2	2	4	4	4
R_RFSP_EraseCodeFlashReq	23	33	33	4	4	4
R_RFSP_WriteCodeFlashReq	33	40	49	8	6	4
R_RFSP_EraseDataFlashReq	22	38	33	4	6	4
R_RFSP_WriteDataFlashReq	33	40	49	8	6	6
R_RFSP_HOOK_EnterCriticalSection	9	9	11	4	4	4
R_RFSP_HOOK_ExitCriticalSection	11	10	9	4	4	4

3 API Functions of RFSP Type 01

3.1 List of API Functions of RFSP Type 01

3.1.1 API Functions Used in Common Flash Memory Control

Table 3-1 shows the API functions used in common for flash memory control in RFSP Type 01.

Table 3-1 API Functions Used in Common for Flash Memory Control in RFSP Type 01

	API function name	summary
1	R_RFSP_Init	Sets the frequency specified by the parameter in the flash memory sequencer and initializes the RFSP Type 01.
2	R_RFSP_SetFlashMemoryMode	Sets the flash self-programming mode specified in the argument to the flash memory sequencer.
3	R_RFSP_CheckCFDFSeqEndStep1	Checks if the operation of the activated the flash memory sequencer has been completed.
4	R_RFSP_CheckCFDFSeqEndStep2	Checks if the command operation has been completed after the flash memory sequencer control register is cleared.
5	R_RFSP_GetSeqErrorStatus	Acquires the information on errors that occurred during command execution in the flash memory sequencer.
6	R_RFSP_ForceReset	Generates an internal reset of the CPU.

3.1.2 API Functions for Code Flash Memory Control

Table 3-2 show the API functions for code flash memory control in RFSP Type 01.

Table 3-2 API Functions for Code Flash Memory Control in RFSP Type 01

	API function name	summary
1	R_RFSP_EraseCodeFlashReq	Activates the flash memory sequencer and begins the erasure of the code flash memory (one block).
2	R_RFSP_WriteCodeFlashReq	Activates the flash memory sequencer and begins the programming of the code flash memory (4 bytes).

3.1.3 API Functions for Data Flash Memory Control

Table 3-3 show the API functions for data flash memory control in RFSP Type 01

Table 3-3 API Functions for Data Flash Memory Control in RFSP Type 01

	API function name	summary
1	R_RFSP_EraseDataFlashReq	Activates the flash memory sequencer and begins the erasure of the data flash memory (one block).
2	R_RFSP_WriteDataFlashReq	Activates the flash memory sequencer and begins the programming of the data flash memory (4 bytes).

3.1.4 **Hook function**

Table 3-4 show the hook functions in RFSP Type 01.

Table 3-4 Hook Functions in RFSP Type 01

	API function name	summary
1	R_RFSP_HOOK_EnterCriticalSection	Executes the instruction for disabling interrupts.
2	R_RFSP_HOOK_ExitCriticalSection	Executes the instruction for enabling interrupts.

3.2 Data type definition

3.2.1 data type

Table 3-5 shows the data type definitions in RFSP Type 01.

Table 3-5 Data Type Definition in RFSP Type 01

Macro value	Type	Description
int8_t	signed char	1-byte signed integer
uint8_t	unsigned char	1-byte unsigned integer
int16_t	signed short	2-byte signed integer
uint16_t	unsigned short	2-byte unsigned integer
int32_t	signed long	4-byte signed integer
uint32_t	unsigned long	4-byte unsigned integer
rBool_t	unsigned char	Boolean value (false = 0, true = 1)

3.2.2 Global variables

The following shows the global variables used in RFSP Type 01:

sg_u08_psw_ie_state

Type/Name	static uint8_t sg_u08_psw_ie_state
Default value	0x00 (R_RFSP_VALUE_U08_INIT_VARIABLE)
Description	Variable for saving or restoring the state of the interrupt enable flag (IE) in PSW — Interrupts are disabled: 0x00u — interrupts are enabled: 0x80u
Definition file	r_rfsp_common_userown.c

3.2.3 **Enumerations**

- e_rfsp_flash_memory_mode (enumerated-type variable name: e_rfsp_flash_memory_mode_t)
  Flash self-programming mode

Symbol Name	Value	Description
R_RFSP_ENUM_FLASH_MODE_NONPROGRAMMABLE	0x08	Non-programmable mode
R_RFSP_ENUM_FLASH_MODE_CODE_PROGRAMMING	0x02	Code flash memory programming setting
R_RFSP_ENUM_FLASH_MODE_DATA_PROGRAMMING	0x22	Data flash memory programming setting

- e_rfsp_ret (enumerated-type variable name: e_rfsp_ret_t)
  Return value

Symbol Name	Value	Description
R_RFSP_ENUM_RET_STS_OK	0x00	Normal end
R_RFSP_ENUM_RET_STS_BUSY	0x01	Busy
R_RFSP_ENUM_RET_ERR_PARAMETER	0x10	Parameter error
R_RFSP_ENUM_RET_ERR_MODE_MISMATCHED	0x11	Mode mismatch error

3.2.4 **Macro definition**

3.2.4.1 **Macro Definitions for Setting the Global Data of RFSP**

- Macro definitions for masking to obtain 16-bit and 8-bit data
  The data bits exceeding the specified size are masked by ANDing with 0.

Symbol Name	Value	Description
R_RFSP_VALUE_U08_MASK1_8BIT	0xFFu	8-bit mask value
R_RFSP_VALUE_U16_MASK1_16BIT	0xFFFFu	16-bit mask value

- Macro definitions for shifting data by 16 bits and 8 bits
  A 32-bit value is shifted by 16 bits or 8 bits, and a 16-bit value is shifted by 8 bits.

Symbol Name	Value	Description
R_RFSP_VALUE_U08_SHIFT_8BIT	8u	Value for 8-bit shifting
R_RFSP_VALUE_U08_SHIFT_16BIT	16u	Value for 16-bit shifting

- Macro definition for shifting an initial value
  The initial value for a global variable is defined.

Symbol Name	Value	Description
R_RFSP_VALUE_U08_INIT_VARIABLE	0x00u	The initial value for a global variable

3.2.4.2 **Macro Definitions for Setting the Registers in the RL78/G15 and RL78/G16.**

- Macro definitions 1 for FSSQ (flash memory sequencer control register)
  The commands to be executed in the activated flash memory sequencer are defined.
  [Bit 7] SQST: Bit for starting or stopping the sequencer.
  The sequencer starts operation when SQST = 1.
  [Bits 2 to 0] SQMD2 to SQMD0: Command for the flash memory sequencer
  Target register definition: R_RFSP_REG_U08_FSSQ

Symbol Name	Value	Description
R_RFSP_VALUE_U08_FSSQ_WRITE	0x81u	Write command for the flash memory
R_RFSP_VALUE_U08_FSSQ_ERASE	0x84u	Erase command for the flash memory
R_RFSP_VALUE_U08_FSSQ_CLEAR	0x00u	Value for clearing the settings for operation of the flash memory sequencer

- Macro definitions for FLPMC (flash programming mode control register)

  The values used to control the transition between the self-programming mode and the non-programmable mode are defined.

  [Bit 5] SELDFL: Bit which selects the target area of flash programming. A code flash area is selected by SELDFL=0, and a data flash area is selected by SELDFL=1.

  [Bit 3] FWEDIS: Bit for enabling or disabling the erasure or programming of the flash memory by software. FWEDIS should be set to 0 to erase or program the flash memory.

  [Bit 1] FLSPM: Bit for specifying the flash memory control mode.

  The flash memory programming mode is entered when FLSPM = 1.

  Target register definition: R_RFSP_REG_U08_FLPMC

Symbol Name	Value	Description
R_RFSP_VALUE_U08_FLPMC_MODE_NONPROGRAMMABLE	0x08u	Flash memory sequencer not executed
R_RFSP_VALUE_U08_FLPMC_MODE_CODE_FLASH_PROGRAMMING	0x02u	Self-programming mode (code flash area selection)
R_RFSP_VALUE_U08_FLPMC_MODE_DATA_FLASH_PROGRAMMING	0x22u	Self-programming mode (data flash area selection)

- Macro definition for FSASTH (flash memory sequencer status register: upper 8 bits)

  The end state of the flash memory sequencer is defined.

  [Bit 6] SQEND: End state of the flash memory sequencer. SQEND = 1 indicates that the sequencer has completed operation. This bit is cleared when the SQST bit is cleared.

  Target register definition: R_RFSP_REG_U08_FSASTH

Symbol Name	Value	Description
R_RFSP_VALUE_U08_MASK1_FSASTH_SQEND	0x40u	Value to be compared with the end state of the flash memory sequencer

- Macro definition for FSASTL (flash memory sequencer status register: lower 8 bits)

  The value of the error status mask when the operation of the flash memory sequencer is finished is defined.

  [Bit 4] SEQER: Error status of the flash memory sequencer. SEQER =1 indicates a sequencer error.

  [Bit 1] WRER: Error status of the write command. WRER = 1 indicates a write error.

  [Bit 0] ERER: Error status of the block erase command. ERER = 1 indicates an erasure error.

  Target register definition: R_RFSP_REG_U08_FSASTL

Symbol Name	Value	Description
R_RFSP_VALUE_U08_ MASK1_FSASTL_ERROR_FLAG	0x13u	Error status mask value at the end of the flash memory sequencer

- Macro definitions for FSSET (flash memory sequencer frequency setting register)

  The range of operating frequencies of the flash memory sequencer and the correction value (-1) for conversion of the FSSET register setting are defined.

  [Bits 4 to 0] FSET4 to FSET0: The value of (operating frequency – 1) should be specified in these bits.
  (Example: For 16 MHz, specify 16 – 1 = 15 (01111b).)

  Target register definition: R_RFSP_REG_U08_FSSET

Symbol Name	Value	Description
R_RFSP_VALUE_U08_FREQUENCY_LOWER_LIMIT	1u	Lowest allowable operating frequency (1 MHz)
R_RFSP_VALUE_U08_FREQUENCY_UPPER_LIMIT	16u	Highest allowable operating frequency (16 MHz)
R_RFSP_VALUE_U08_FREQUENCY_ADJUST	1u	Correction value (-1) for conversion of the FSSET register setting

- Macro definitions for FLAPH, FLAPL, FLSEDH, and FLSEDL (flash address pointer registers HIGH and LOW)

(1) The start and end addresses of erasure (1 block = 1 Kbyte) for the code flash memory are defined.

FLAPH [Bits 4 to 0]: FLAP12 to FLAP8 specify the upper bits of the start address of a code flash memory area.

FLAPL [Bits 7 to 0]: FLAP7 to FLAP0 specify the lower bits of the start address of a code flash memory area.

FLSEDH [Bits 4 to 0]: EWA12 to EWA8 specify the upper bits of the end address of a code flash memory area.

FLSEDL [Bits 7 to 2]: EWA7 to EWA2 specify the lower bits of the end address of a code flash memory area.

Target register definitions: R_RFSP_REG_U08_FLAPH, R_RFSP_REG_U08_FLAPL, R_RFSP_REG_U08_FLSEDH, and R_RFSP_REG_U08_FLSEDL

Symbol Name	Value	Description
R_RFSP_VALUE_U08_ CODE_FLASH_BLOCK_ADDR_LOW	0x00u	Mask value for the lower bits of the start address of a code flash block (8bit)
R_RFSP_VALUE_U08_ CODE_FLASH_BLOCK_ADDR_HIGH	0x1Fu	Mask value for the upper bits of the start address of a code flash block (8bit)
R_RFSP_VALUE_U08_ CODE_FLASH_BLOCK_ADDR_END_LOW	0xFCu	Lower address of the end of a code flash block (8bit)
R_RFSP_VALUE_U08_ CODE_FLASH_BLOCK_ADDR_END_HIGH	0x03u	Upper address of the end of a code flash block (8bit)
R_RFSP_VALUE_U08_ CODE_FLASH_SHIFT_HIGH_ADDR	2u	Value for shifting the upper address bits to calculate the offset of a code flash area from the block number

(2) The start and end addresses of erasure and blank check (1 block = 256 bytes) for the data flash memory are defined.

FLAPH [Bits 4 to 0]: FLAP12 to FLAP8 specify the upper bits of the start address of a data flash memory area.

FLAPL [Bits 7 to 0]: FLAP7 to FLAP0 specify the lower bits of the start address of a data flash memory area.

FLSEDH [Bits 4 to 0]: EWA12 to EWA8 specify the upper bits of the end address of a data flash memory area.

FLSEDL [Bits 7 to 0]: EWA7 to EWA2 specify the lower bits of the end address of a data flash memory area.

Target register definitions: R_RFSP_REG_U08_FLAPH, R_RFSP_REG_U08_FLAPL, R_RFSP_REG_U08_FLSEDH, and R_RFSP_REG_U08_FLSEDL

Symbol Name	Value	Description
R_RFSP_VALUE_U08_ DATA_FLASH_BLOCK_ADDR_LOW	0x00u	Mask value for the lower bits of the start address of a data flash block (8bit)
R_RFSP_VALUE_U08_ DATA_FLASH_BLOCK_ADDR_HIGH	0x01u	Mask value for the upper bits of the start address of a data flash block (8bit)
R_RFSP_VALUE_U08_ DATA_FLASH_BLOCK_ADDR_END_LOW	0xFCu	Lower address of the end of a data flash block (8bit)
R_RFSP_VALUE_U08_ DATA_FLASH_BLOCK_ADDR_END_HIGH	0x01u	Upper address of the end of a data flash block (8bit)
R_RFSP_VALUE_U08_ DATA_FLASH_SHIFT_HIGH_ADDR	1u	Value for shifting the upper address bits to calculate the offset of a data flash area from the block number

3.3 Specifications of API Functions

This section describes the detailed specifications of the API functions of Renesas Flash Sample Program (RFSP) Type 01.

There are some prerequisites for using the API functions of RFSP Type 01 to reprogram the flash memory. If the prerequisites are not satisfied, execution of the API functions may result in indeterminate operation.

Prerequisites:

- Execute the R_ RFSP _Init() function once before starting the use of RFSP functions.
- The high-speed on-chip oscillator must be active while self-programming is in progress. Execute API functions of RFSP Type 01 only while the high-speed on-chip oscillator is active.
- To control the data flash memory, execute API functions of RFSP Type 01 while access to the data flash memory is enabled. For the method of enabling access to the data flash memory, refer to the user's manual of the target RL78 microcontroller.

The following shows the format for describing the specifications of API functions.

Description format:

Information:

Syntax	Syntax for calling this function from a C-language program	
Reentrancy	Reentrant or Non-reentrant	
Parameters (IN)	Input parameters for this function	Parameter [Value, range, meaning of the parameter, etc.]
Parameters (IN/OUT)	Input/output parameters for this function	Parameter [Value, range, meaning of the parameter, etc.]
Parameters (OUT)	Output parameters for this function	Parameter [Value, range, meaning of the parameter, etc.]
Return Value	Type of the return value from this function (Enumerated type, pointer type, etc.)	Enumerator (constant value) of the return value: Value [Meaning of the constant: Detailed description]
		Enumerator (constant value) of the return value: Value [Meaning of the constant: Detailed description]
Description	Overview of function	
Preconditions	Overview of preconditions	
Remarks	Special notes on this function	

Details of Specifications:

The operation of this function is described.

Note:

Conditions of usage or restrictions on this function are described.

3.3.1 Specifications of API Functions Used in Common for Flash Memory Control

This section describes the API functions used in common for flash memory control in RFSP Type 01.

3.3.1.1 R_RFSP_Init

Information:

Syntax	R_RFSP_FAR_FUNC e_rfsp_ret_t R_RFSP_Init (unit8_t i_u08_cpu_freqency);	
Reentrancy	Non-reentrant	
Parameters (IN)	unit8_t i_u08_cpu_freqency	CPU operating frequency [1 to 16 (MHz)]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfsp_ret_t	R_RFSP_ENUM_RET_STS_OK : 0x00 [Normal end: The frequency is within the allowable range.]
		R_RFSP_ENUM_RET_ERR_PARAMETER : 0x10 [Parameter error: The frequency is outside the allowable range.]
Description	Sets the frequency specified by the parameter in the flash memory sequencer and initializes the RFSP Type 01.	
Preconditions	Execute this function in the non-programmable mode while the high-speed on-chip oscillator is active.	
Remarks	Execute this function once before starting the use of RFSP functions.	

Details of specifications:

- Whether the value of the parameter (CPU operating frequency) is within the range from 1 MHz to 32 MHz is checked. When the value is within the range, the value of (specified CPU operating frequency – 1) is set in the variable g_u08_cpu_frequency and a FSSET register.

Notes:

- The high-speed on-chip oscillator needs to be kept active while self-programming is in progress. Execute this function while the high-speed on-chip oscillator is active.
  * RFSP Type 01 does not activate or check the high-speed on-chip oscillator.
- For the parameter (i_u08_cpu_freqency), specify the integer obtained by rounding up the fraction of the CPU operating frequency that is actually used in the microcontroller.
- (Example: When the CPU operates at 4.5 MHz, specify 5 in this initialization function.)
  When the CPU operates at a frequency lower than 4 MHz, a value of 1 MHz, 2 MHz, or 3 MHz can be used but a non-integer value such as 1.5 MHz cannot be used.
  The frequency specified in the parameter (i_u08_cpu_freqency) should be the actual frequency at which the CPU operates during flash memory reprogramming; it is not necessarily that the frequency of the high-speed on-chip oscillator should be specified.
  — If the specified frequency differs from the actual CPU operating frequency, the subsequent operation is indeterminate. In this case, even if flash memory reprogramming is completed, the written data value and data retention period may not be guaranteed.
  * For the range of the CPU operating frequency, refer to the user's manual of the target RL78 microcontroller.
- If this function is executed while the sequencer is not in the non-programmable mode, the subsequent operation is indeterminate.

3.3.1.2 R_RFSP_SetFlashMemoryMode

Information:

Syntax	R_RFSP_FAR_FUNC e_rfsp_ret_t R_RFSP_SetFlashMemoryMode (e_rfsp_flash_memory_mode_t i_e_pe_mode);	
Reentrancy	Non-reentrant	
Parameters (IN)	e_rfsp_flash_memory_mode_t i_e_pe_mode	Flash self-programming mode
		R_RFSP_ENUM_FLASH_MODE_NONPROGRAMMABLE : 0x08 [Non-programable Mode] R_RFSP_ENUM_FLASH_MODE_CODE_PROGRAMMING : 0x02 [Self-programming mode: Code Flash area selection] R_RFSP_ENUM_FLASH_MODE_DATA_PROGRAMMING : 0x22 [Self-programming mode: Data Flash area selection]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfsp_ret_t	R_RFSP_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFSP_ENUM_RET_ERR_MODE_MISMATCHED : 0x11 [Mode mismatch error] (The flash memory sequencer is not placed in the specified mode.)specified mode)
Description	Sets the flash self-programming mode specified in the argument to the flash memory sequencer.	
Preconditions	Execute this function while command execution is not in progress in the flash memory sequencer.	
Remarks	-	

Details of Specifications:

- The FLPMC register is set up according to the value of the parameter (i_e_pe_mode) to place the flash memory sequencer in the specified flash memory control mode.

Notes:

- If the value specified by the parameter is not a flash memory control mode value, the operation is same as that for the non-programmable mode.
- If this function is executed before the R_RFSP_Init function, the reprogrammed data are not guaranteed even after the reprogramming processing by the RFSP is completed. To use RFSP Type 01, be sure to execute the R_RFSP_Init() function once before starting the use of RFSP functions.

3.3.1.3 R_RFSP_CheckCFDFSeqEndStep1

Information:

Syntax	R_RFSP_FAR_FUNC e_rfsp_ret_t R_RFSP_CheckCFDFSeqEndStep1 (void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfsp_ret_t	R_RFSP_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFSP_ENUM_RET_STS_BUSY: 0x01 [Sequencer command execution is in progress.]
Description	Checks if the operation of the activated the flash memory sequencer has been completed.	
Preconditions	Execute this function after the flash memory sequencer has completed operation.	
Remarks	Execute this function again if R_RFSP_STS_BUSY is returned. After confirming that R_RFSP_ENUM_RET_STS_OK has been returned from this function, execute the R_RFSP_CheckCFDFSeqEndStep2() function.	

Details of specifications:

- Whether the operation of the activated flash memory sequencer has been completed (SQEND (bit 6 of FSASTH) = 1) is checked.
- When the operation of the flash memory sequencer has been completed, the flash memory sequencer control register is cleared (FSSQ = 0x00) and R_RFSP_ENUM_RET_STS_OK is returned.
  If the operation has not been completed, R_RFSP_ENUM_RET_STS_BUSY is returned.

Notes:

- Execute this function again if R_RFSP_STS_BUSY is returned.
- If execution of this function is attempted before the command for activating the flash memory sequencer is started, this function is not executed correctly.
- After confirming that R_RFSP_ENUM_RET_STS_OK has been returned from this function, execute the R_RFSP_CheckCFDFSeqEndStep2() function.

3.3.1.4 R_RFSP_CheckCFDFSeqEndStep2

Information:

Syntax	R_RFSP_FAR_FUNC e_rfsp_ret_t R_RFSP_CheckCFDFSeqEndStep2 (void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfsp_ret_t	R_RFSP_ENUM_RET_STS_OK: 0x00 [Normal end: Sequencer operation has been completed.]
		R_RFSP_ENUM_RET_STS_BUSY : 0x01 [Sequencer operation is in progress.]
Description	Checks if the command operation has been completed after the flash memory sequencer control register is cleared.	
Preconditions	Execute this function after confirming that R_RFSP_ENUM_RET_STS_OK has been returned from the R_RFSP_CheckCFDFSeqEndStep1() function.	
Remarks	Execute this function again if R_RFSP_STS_BUSY is returned	

Details of specifications:

- Whether the command operation in the flash memory sequencer has been completed (SQEND (bit 6 of FSASTH) = 0) is checked after the flash memory sequencer control register is cleared (FSSQ = 0x00).
- When the command execution in the flash memory sequencer has been completed, R_RFSP_ENUM_RET_STS_OK is returned.
  If the operation has not been completed, R_RFSP_ENUM_RET_STS_BUSY is returned.

Notes:

- Execute this function again if R_RFSP_STS_BUSY is returned.
- If execution of this function is attempted before R_RFSP_ENUM_RET_STS_OK has been confirmed by the R_RFSP_CheckCFDFSeqEndStep1() function, this function is not executed correctly.

3.3.1.5 R_RFSP_GetSeqErrorStatus

Information:

Syntax	R_RFSP_FAR_FUNC void R_RFSP_GetSeqErrorStatus (uint8_t __near * onp_u08_error_status);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	uint8_t __near *onp_u08_error_status	Pointer to the variable for storing the information on errors
Return Value	N/A	
Description	Acquires the information on errors that occurred during command execution in the flash memory sequencer.	
Preconditions	Execute this function after the flash memory sequencer has completed operation.	
Remarks	-	

Details of specifications:

- The FSASTL register (8 bits) is read and the value is stored in the variable pointed to by the parameter (onp_u08_error_status).

  Error information to be acquired (three bits of the FSASTL register: bits 4, 1, and 0):
  Bit 7: (0) Reserved
  Bit 6: (0) Reserved
  Bit 5: (0) Reserved
  Bit 4: Flash memory sequencer error
  Bit 3: (0) Reserved
  Bit 2: (0) Reserved
  Bit 1: Write command error
  Bit 0: Erase command error

3.3.1.6 **R_RFSP_ForceReset**

Information:

Syntax	R_RFSP_FAR_FUNC void R_RFSP_ForceReset(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Generates an internal reset of the CPU.	
Preconditions	-	
Remarks	-	

Details of specifications:

- The illegal instruction code (0xFF) is intentionally executed to generate an internal reset of the CPU.

Notes:

- As an internal reset is generated in the CPU, the code after this function is not executed.
- For the internal reset by the instruction code 0xFF (illegal instruction), refer to the user's manual of the target RL78 microcontroller.
- A reset is not generated by this function during emulation by an on-chip debugging emulator.

3.3.2 **Specifications of API Functions for Code Flash Memory Control**

This section describes the API functions for code flash memory control in RFSP Type 01.

3.3.2.1 **R_RFSP_EraseCodeFlashReq**

Information:

Syntax	R_RFSP_FAR_FUNC void R_RFSP_EraseCodeFlashReq (uint8_t i_u08_block_number);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint8_t i_u08_block_number	Target block number for erasure [0 to 31] Example: For RL78/G15, 0 to 7 (8 Kbytes max.) For RL78/G16, 0 to 31 (32 Kbytes max.)
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Activates the flash memory sequencer and begins the erasure of the code flash memory (one block).	
Preconditions	Use this function in the self-programming mode (code flash area selection) while command execution is not in progress in the flash memory sequencer.	
Remarks	Execute the R_RFSP_CheckCFDFSeqEndStep1() function after this function.	

Details of specifications:

- The flash memory sequencer is activated and the address of one block (1 Kbyte) to be erased in the code flash memory is set in the sequencer.
  - The start address and end address of the target block (1 Kbyte) in the code flash memory are calculated from the block number for erasure specified by the parameter (i_u08_block_number) and set in the FLAPL and FLAPH registers and the FLSEDL and FLSEDH registers, respectively.
- R_RFSP_VALUE_U08_FSSQ_ERASE = 0x84 is set in the FSSQ register to start the erasure. (SQST (bit 7) = 1, SQMD (bits 2 to 0) = 4 (0b100), and the other bits are set to 0.)

Notes:

- The lower 5 bits of the 8-bit parameter (i_u08_block_number) are used; the upper 3 bits are not used. The target block number must not exceed the number of blocks in the code flash memory implemented in the device. If the specified number is outside the allowable range, the subsequent operation is indeterminate.
- If this function is executed while the sequencer is not in the self-programming mode (code flash area selection), the subsequent operation is indeterminate.
- If this function is executed while command execution is in progress in the flash memory sequencer, the subsequent operation is indeterminate.

3.3.2.2 R_RFSP_WriteCodeFlashReq

Information:

Syntax	R_RFSP_FAR_FUNC void R_RFSP_WriteCodeFlashReq (uint16_t i_u16_start_addr, uint8_t __near * inp_u08_write_data);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint16_t i_u16_start_addr	Target start address for programming (4-byte boundary) [Address in the code flash area]
	uint8_t __near * inp_u08_write_data	Pointer to the variable that stores write data [Size of the write data pointed to is 4 bytes]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Activates the flash memory sequencer and begins the programming of the code flash memory (4 bytes).	
Preconditions	Use this function in the self-programming mode (code flash area selection) while command execution is not in progress in the flash memory sequencer.	
Remarks	Execute the R_RFSP_CheckCFDFSeqEndStep1() function after this function.	

Details of specifications:

- The flash memory sequencer is activated, and the programming start address in the code flash memory and the write data (4 bytes) are set in the sequencer.
  - The target start address in the code flash memory specified by the parameter i_u16_start_addr is set in the FLAPL and FLAPH registers.
  - The 4-byte value in the variable (data to be written to the code flash memory) pointed to by the parameter inp_u08_write_data is set in the FLWLL, FLWLH, FLWHL and FLWHH registers.
- R_RFSP_VALUE_U08_FSSQ_WRITE = 0x81 is set in the FSSQ register to start programming. (SQST (bit 7) = 1, SQMD (bits 2 to 0) = 1 (0b001), and the other bits are set to 0.)

Notes:

- The 13 bits of the 16-bit parameter i_u16_start_addr are used with the upper 1 bit and lower 2 bits masked with ‘0’. The start address must be a 4-byte boundary address within the space of the code flash memory implemented in the device. If the specified address is outside the allowable space or is not a 4-byte boundary address, the subsequent operation is indeterminate.
- The parameter inp_u08_write_data is a pointer to the 8-bit input data. To repeat the function processing with this pointer updated, note that the pointer needs to be updated in units of 4 bytes (in units of programming of the code flash memory).
- If this function is executed while the sequencer is not in the self-programming mode, the subsequent operation is indeterminate.
- If this function is executed while command execution is in progress in the flash memory sequencer, the subsequent operation is indeterminate.

3.3.3 Specifications of API Functions for Data Flash Memory Control

This section describes the API functions for code flash memory control in RFSP Type 01.

3.3.3.1 R_RFSP_EraseDataFlashReq

Information:

Syntax	R_RFSP_FAR_FUNC void R_RFSP_EraseDataFlashReq (uint8_t i_u08_block_number);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint8_t i_u08_block_number	Target block number for erasure [0 to 1] Example: For RL78/G15, RL78/G16, 0 to 1 (1 Kbyte max.)
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Activates the flash memory sequencer and begins the erasure of the data flash memory (one block).	
Preconditions	Use this function in the self-programming mode (data flash area selection) while command execution is not in progress in the flash memory sequencer.	
Remarks	Execute the R_RFSP_CheckCFDFSeqEndStep1() function after this function.	

Details of specifications:

- The flash memory sequencer is activated and the address of one block (512 bytes) to be erased in the data flash memory is set in the sequencer.
  - The start address and end address of the target block (512 bytes) in the data flash memory are calculated from the block number for erasure specified by the parameter (i_u08_block_number) and set in the FLAPL and FLAPH registers and the FLSEDL and FLSEDH registers, respectively.
- R_RFSP_VALUE_U08_FSSQ_ERASE = 0x84 is set in the FSSQ register to start the erasure. (SQST (bit 7) = 1, SQMD (bits 2 to 0) = 4 (0b100), and the other bits are set to 0.)

Notes:

- The lower 1 bit of the 8-bit parameter (i_u08_block_number) are used; the upper 7 bits are not used. The target block number must not exceed the number of blocks in the data flash memory implemented in the device. If the specified number is outside the allowable range, the subsequent operation is indeterminate.
- If this function is executed while the sequencer is not in the self-programming mode (data flash area selection), the subsequent operation is indeterminate.
- If this function is executed while command execution is in progress in the flash memory sequencer, the subsequent operation is indeterminate.

3.3.3.2 R_RFSP_WriteDataFlashReq

Information:

Syntax	R_RFSP_FAR_FUNC void R_RFSP_WriteDataFlashReq (uint16_t i_u16_start_addr, uint8_t __near * inp_u08_write_data);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint16_t i_u16_start_addr	Target start address for programming (4-byte boundary) which considers that the programming address 0x9000 is base address "0x0000". [Address in the data flash area: 0x0000-0x03FC]
	uint8_t __near * inp_u08_write_data	Pointer to the variable that stores write data [Size of the write data pointed to is 4 bytes]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Activates the flash memory sequencer and begins the programming of the data flash memory (4 bytes).	
Preconditions	Use this function in the self-programming mode (data flash area selection) while command execution is not in progress in the flash memory sequencer.	
Remarks	Execute the R_RFSP_CheckCFDFSeqEndStep1() function after this function.	

Details of specifications:

- The flash memory sequencer is activated, and the programming start address in the data flash memory and the write data (4 bytes) are set in the sequencer.
  - The target start address in the code flash memory specified by the parameter i_u16_start_addr is set in the FLAPL and FLAPH registers.
  - The 4-byte value in the variable (data to be written to the data flash memory) pointed to by the parameter inp_u08_write_data is set in the FLWLL, FLWLH, FLWHL and FLWHH registers.
- R_RFSP_VALUE_U08_FSSQ_WRITE = 0x81 is set in the FSSQ register to start programming. (SQST (bit 7) = 1, SQMD (bits 2 to 0) = 1 (0b001), and the other bits are set to 0.)

Notes:

- The 8 bits of the 16-bit parameter i_u16_start_addr are used with the upper 6 bit and lower 2 bits masked with ‘0’. The start address must be a 4-byte boundary address within the space of the data flash memory implemented in the device. If the specified address is outside the allowable space or is not a 4-byte boundary address, the subsequent operation is indeterminate.
- The parameter inp_u08_write_data is a pointer to the 8-bit input data. To repeat the function processing with this pointer updated, note that the pointer needs to be updated in units of 4 bytes (in units of programming of the data flash memory).
- If this function is executed while the sequencer is not in the self-programming mode, the subsequent operation is indeterminate.
- If this function is executed while command execution is in progress in the flash memory sequencer, the subsequent operation is indeterminate.

3.3.4 Specifications of Hook Functions

This section describes the hook functions of RFSP Type 01.

3.3.4.1 R_RFSP_HOOK_EnterCriticalSection

Information:

Syntax	R_RFSP_FAR_FUNC void R_RFSP_HOOK_EnterCriticalSection(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Executes the instruction for disabling interrupts.	
Preconditions	Execute this function before the processing that should be executed with interrupts disabled.	
Remarks	-	

Details of specifications:

- The interrupt disabled or enabled state is acquired and saved in the variable sg_u08_psw_ie_state that is prepared to store the value of the interrupt enable flag (IE) of the PSW.
- The macro instruction for disabling interrupts (R_RFSP_DISABLE_INTERRUPT) is executed.

Note:

- Execute this function before the processing that should be executed with interrupts disabled (critical section), and execute the R_RFSP_HOOK_ExitCriticalSection function after the critical section ends.

3.3.4.2 R_RFSP_HOOK_ExitCriticalSection

Information:

Syntax	R_RFSP_FAR_FUNC void R_RFSP_HOOK_ExitCriticalSection(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Executes the instruction for enabling interrupts.	
Preconditions	Execute this function to enable interrupts after the processing executed with interrupts disabled.	
Remarks	-	

Details of specifications:

- According to the value of the variable sg_u08_psw_ie_state, which saves the interrupt enable flag (IE) of the PSW, the macro instruction for enabling interrupts is executed.
  Value of sg_u08_psw_ie_state:
  - 0x00 (bit 7 = 0: interrupts are disabled): Nothing is done.
  - 0x80 (bit 7 = 1: interrupts are enabled): The macro instruction for enabling interrupts (R_RFSP_ENABLE_INTERRUPT) is executed and the interrupt enabled state (EI) is restored.

Note:

- Execute this function after the R_RFSP_HOOK_EnterCriticalSection is executed and the processing executed with interrupts disabled (critical section) ends.

4 Flash Memory Sequencer Operation

Before operating the flash memory sequencer of the RL78/G15 or RL78/G16, the operating frequency of the CPU must be set to the flash memory sequencer frequency setting register.

4.1 Initial Setting of Operating Frequency

The value (g_u08_cpu_frequency: integer value – 1) of the CPU operating frequency (1 MHz to 32 MHz) specified by the arguments of R_RFSP_Init function is set in the FSET bits (bits 4 o 0) of the flash memory sequencer frequency setting register (FSSET).

Specify the integer value obtained by rounding up the fraction part of the CPU operating frequency. (Example: When the CPU operating frequency is 4.5 MHz, specify 5 in the initialization function.)

When the CPU operating frequency is lower than 4 MHz, a frequency of 1 MHz, 2 MHz, or 3 MHz can be specified. A non-integer frequency such as 1.5 MHz cannot be used.

Target functions of this operation: R_RFSP_Init, R_RFSP_SetFlashMemoryMode

Operation Procedure:

- To write to the flash memory sequencer frequency setting register (FSSET), set '0' to [bit7-5] and the CPU operating frequency [1 to 16 (MHz)] value [integer value -1] to [bit4-0] corresponding to FSET.

Note: Before operating (such as reprogramming) the code flash memory or data flash memory by using the flash memory sequencer, specify the CPU operating frequency in the FSET bits of the FSSET register.
Note that the reprogramming operation is indeterminate and written data are not guaranteed if reprogramming is attempted before the CPU operating frequency is specified correctly. (Even if expected data are read from the flash memory immediately after reprogramming, the data retention period cannot be guaranteed.)

FSSET register (After reset : 0x00):

7	6	5	4	3	2	1	0
0	0	0	FSET4	FSET3	FSET2	FSET1	FSET0
R	R	R	R/W	R/W	R/W	R/W	R/W

4.2 Self-Programming Mode and Target Area Setting

By setting RL78/G15 and RL78/G16 to self-programming mode, programming to each flash area (a code flash memory or a data flash memory) is possible.

4.2.1 Flash Memory Self-Programming Mode Setting

The self-programming mode of flash memory is set by the flash programming mode control register (FLPMC register).

FLPMC register (After reset : 0x08):

7	6	5	4	3	2	1	0
0	0	SELDFL	0	FWEDIS	0	FLSPM	0
R	R	R/W	R	R/W	R	R/W	R

- SELDFL [bit5] Selects the flash programming area.

  SELDFL = 0 (after a reset) / 1 : Selects the cod e flash area / Selects the data flash area

- FWEDIS [bit3] Controls over enabling or disabling erasure and programming of the flash memory.

  FWEDIS = 0 / 1 (after a reset) : [Write/Erase] Enable / [Write/Erase] Disable

- FLSPM [bit1] Selects flash programming mode.

  FLSPM = 0 (after a reset) / 1 : Read mode (normal mode) / Flash self-programming mode

Operation Procedure:

- Self-programming mode (code flash area selection) state setting

  Set the FLPMC register to 0x02

- Self-programming mode (data flash area selection) state setting

  Set the FLPMC register to 0x22

- Flash memory non-programable state setting

  Set the FLPMC register to 0x08

4.3 Flash Memory Sequencer

4.3.1 Outline

In order to reprogram the code flash area or data flash area for RL78/G15 or RL78/G16, the flash memory sequencer dedicated command needs to be executed.

4.3.2 Flash Memory Sequencer Commands

In order to reprogram the code flash area or data flash area of RL78/G15 or RL78/G16, the 1-word [4-byte] writing command and the block erasure command are prepared. To issue a command, specify the desired command number in the SQMD2 to SQMD0 bits (bits 2 to 0) of the flash memory sequencer control register (FSSQ) and set the SQST bit (bit 7) to 1. Be sure to execute the command after understanding it by referring to 1.5 Points for Caution

FSSQ register (After reset : 0x00):

7	6	5	4	3	2	1	0
SQST	0	0	0	0	SQMD2	SQMD1	SQMD0
R/W	R	R	R	R	R/W	R/W	R/W

Table 4-1 shows the dedicated commands for flash memory sequencer.

Table **4-1** Dedicated Commands for Flash Memory Sequencer

SQMD2-0	Function of Dedicated Command
	Description
0h	Initial value (command unselected)
1h	write
	The data specified in the FLWHH, FLWHL, FLWLH, and FLWLL registers are written to the flash memory address specified by the FLAPH and FLAPL registers. - Programming unit (code flash area): 1 word (4byte) [When SELDFL bit is set to 0] - Programming unit (data flash area): 1 word (4byte) [When setting SELDFL bit to 1]
4h	Block erase
	Data are erased from the blocks between the start address specified by the FLAPH and FLAPL registers and the end address specified by the FLSEDH and FLSEDL registers.
Other values	Setting prohibited

Note) An address in case the data flash area is specified by the writing command or the block erasure command specifies the relative address which set the top address (0x9000) to 0x0000.

FLAPH/FLAPL register (flash address pointer registers)

- Target MCU:RL78/G15

FLAPH register (After reset: 0x00):

7	6	5	4	3	2	1	0
0	0	0	FLAP 12	FLAP 11	FLAP 10	FLAP 9	FLAP 8
R	R	R	R/W	R/W	R/W	R/W	R/W

FLAPL register (After reset: 0x00):

7	6	5	4	3	2	1	0
FLAP 7	FLAP 6	FLAP 5	FLAP 4	FLAP 3	FLAP 2	FLAP 1	FLAP 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

- Target MCU:RL78/G16

FLAPH register (After reset: 0x00):

7	6	5	4	3	2	1	0
0	FLAP 14	FLAP 13	FLAP 12	FLAP 11	FLAP 10	FLAP 9	FLAP 8
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W

FLAPL register (After reset: 0x00):

7	6	5	4	3	2	1	0
FLAP 7	FLAP 6	FLAP 5	FLAP 4	FLAP 3	FLAP 2	FLAP 1	FLAP 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

FLSEDH/FLSEDL registers (flash end address pointer registers)

- Target MCU:RL78/G15

FLSEDH register (After reset: 0x00):

7	6	5	4	3	2	1	0
0	0	0	EWA 12	EWA 11	EWA 10	EWA 9	EWA 8
R	R	R	R/W	R/W	R/W	R/W	R/W

FLSEDL register (After reset: 0x0000):

7	6	5	4	3	2	1	0
EWA 7	EWA 6	EWA 5	EWA 4	EWA 3	EWA 2	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R	R

- Target MCU:RL78/G16

FLSEDH register (After reset: 0x00):

7	6	5	4	3	2	1	0
0	EWA14	EWA13	EWA 12	EWA 11	EWA 10	EWA 9	EWA 8
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W

FLSEDL register (After reset: 0x0000):

7	6	5	4	3	2	1	0
EWA 7	EWA 6	EWA 5	EWA 4	EWA 3	EWA 2	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R	R

FLWHH/ FLWHL/FLWLH/FLWLL registers (flash write buffer registers)

FLWHH register (After reset: 0x00):

15	14	13	12	11	10	9	8
FLW 31	FLW 30	FLW 29	FLW 28	FLW 27	FLW 26	FLW 25	FLW 24
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

FLWHL register (After reset: 0x00):

7	6	5	4	3	2	1	0
FLW 23	FLW 22	FLW 21	FLW 20	FLW 19	FLW 18	FLW 17	FLW 16
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

FLWLH register (After reset: 0x00):

15	14	13	12	11	10	9	8
FLW 15	FLW 14	FLW 13	FLW 12	FLW 11	FLW 10	FLW 9	FLW 8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

FLWLL register (After reset: 0x00):

7	6	5	4	3	2	1	0
FLW 7	FLW 6	FLW 5	FLW 4	FLW 3	FLW 2	FLW 1	FLW 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

4.3.2.1 Reprogramming the Code Flash Area

Code flash area reprogramming sets the flash programming mode to self-programming mode (code flash area selection) and then executes flash memory sequencer commands. The specified address and data required for each command execution must be set to the appropriate register in advance, and then the command must be started.

Units of erasure and writing for reprogramming of the code flash area:

- Block erase unit: **1 Kbytes**
- Write unit: **1 word (4 bytes)**

Target functions of this operation: R_RFSP_EraseCodeFlashReq, R_RFSP_WriteCodeFlashReq

Operation Procedure:

Block erase and write commands for the code flash memory can be used.

- Set the target area to code flash memory and shift to self-programming mode. For the mode transition procedure, see section 4.2.1 Flash Memory Self-Programming Mode Setting.
  Set the FLPMC register to 0x02
- Specify the necessary data in the respective registers before executing a command.

(1) Block erase
FLAPH and FLAPL registers: Start block address of the code flash memory (Example: 0x0800)
FLSEDH and FLSEDL registers: End block address of the code flash memory(Example: 0x0BFF)

(2) Write: This command is executed in units of one word (4 bytes); specify a multiple of 4 as an address — that is, set bits 1 and 0 to 0.
FLAPH and FLAPL registers: Start address of the target code flash memory area (Example: 0x0800)
FLSEDH and FLSEDL registers: Set to all 0s or specify nothing. (Example: 0x000000)
FLWHH, FLWHL, FLWLH and FLWLL registers: Specify the data to be written (1 word (4 bytes)).

- Specify the desired command number in the SQMD2 to SQMD0 bits (bits 2 to 0) of the FSSQ register and set the SQST bit (bit 7) to 1.
  Block erase: 0x84 Write: 0x81
  Note) Because CPU stops while reprogramming the code flash memory by self-programming, the user program is not executed. Therefore, be careful.
- Wait until command execution is completed in the flash memory sequencer. For the procedure for waiting for the completion of command execution, see section 4.3.3 Procedures for Judging the End of Command Execution in the Flash Memory Sequencer.
- Processing after command execution
  To continue command processing:
  Programming command or block erase command for code flash area reprogramming can be executed with the data in the registers modified while the sequencer is set in the self-programming mode(code flash area selection).
  To complete command processing:
  Transitions to a flash memory non-programable state. For the mode transition procedure, see section 4.2.1 Flash Memory Self-Programming Mode Setting.

4.3.2.2 Reprogramming the Data Flash Area

Data flash area reprogramming sets the flash programming mode to self-programming mode (data flash area selection) and then executes flash memory sequencer commands. The specified address and data required for each command execution must be set to the appropriate register in advance, and then the command must be started.

Units of erasure and writing for reprogramming of the data flash area:

- Block erase unit: **512 bytes**
- Write unit: **1 word (4 bytes)**

Target functions of this operation: R_RFSP_EraseDataFlashReq , R_RFSP_WriteDataFlashReq

Operation Procedure:

Block erase and write commands for the data flash memory can be used.

- Set the target area to data flash memory and shift to self-programming mode. For the mode transition procedure, see section 4.2.1 Flash Memory Self-Programming Mode Setting.
  Set the FLPMC register to 0x22
- Specify the necessary data in the respective registers before executing a command.

(1) Block erase
FLAPH and FLAPL registers: Start block address of the data flash memory
(Example: In the case where 0x9000 is specified, because 0x9000 is a base address, set the relative value 0x0000.)
FLSEDH and FLSEDL registers: End block address of the data flash memory

- (Example: In the case where 0x91FF is specified, because 0x9000 is a base address, set the relative value 0x01FF.)

(2) Write: This command is executed in units of one word (4 bytes); specify a multiple of 4 as an address — that is, set bits 1 and 0 to 0.
FLAPH and FLAPL registers: Start address of the target data flash memory
(Example: In the case where 0x9000 is specified, because 0x9000 is a base address, set the relative value 0x0000.)
FLSEDH and FLSEDL registers: Set to all 0s or specify nothing. (Example: 0x000000)
FLWHH, FLWHL, FLWLH and FLWLL registers: Specify the data to be written (1 word (4 bytes))

- Specify the desired command number in the SQMD2 to SQMD0 bits (bits 2 to 0) of the FSSQ register and set the SQST bit (bit 7) to 1.
  Block erase: 0x84 Write: 0x81
  Note) Because CPU stops while reprogramming the data flash memory by self-programming, the user program is not executed. Therefore, be careful.
- Wait until command execution is completed in the flash memory sequencer. For the procedure for waiting for the completion of command execution, see section 4.3.3 Procedures for Judging the End of Command Execution in the Flash Memory Sequencer.
- Processing after command execution
  To continue command processing:
  Programming command or block erase command for data flash area reprogramming can be executed with the data in the registers modified while the sequencer is set in the self-programming mode(data flash area selection).
  To complete command processing:
  Transitions to a flash memory non-programable state. For the mode transition procedure, see section 4.2.1 Flash Memory Self-Programming Mode Setting.

4.3.3 Procedures for Judging the End of Command Execution in the Flash Memory Sequencer

To terminate command execution in the flash memory sequencer started in the RL78/G15 or RL78/G16, a specific procedure for judging the end of command execution should be used.

Read the SQEND bit (bit 6) of the FSASTH register and confirm that it is set to 1 to judge the end of command execution in flash memory sequencer. After this judgement, read the error bits (WRER (bit 1), and ERER (bit 0)) of the FSASTL register to check whether an error has occurred in the execution of the respective commands.

FSASTH register (After reset: 0x00):

7	6	5	4	3	2	1	0
0	SQEND	0	0	0	0	0	0
R	R	R	R	R	R	R	R

FSASTL register (After reset: 0x00):

7	6	5	4	3	2	1	0
0	0	0	SEQER	0	0	WRER	ERER
R	R	R	R	R	R	R	R

Judgment Procedure:

(1) After starting the execution of a command in the code/data flash area, wait until the SQEND bit (bit 6) of the FSASTH register is automatically set.
(2) After confirming that the SQEND bit (bit 6) has been set, clear the SQST bit (bit 7) of the FSSQ register.
(3) Wait until the SQEND bit (bit 6) of the FSASTH register is automatically cleared; the procedure ends when the bit is cleared.

Figure 4-1 shows the flow of judging the end of command execution in the flash memory sequencer.

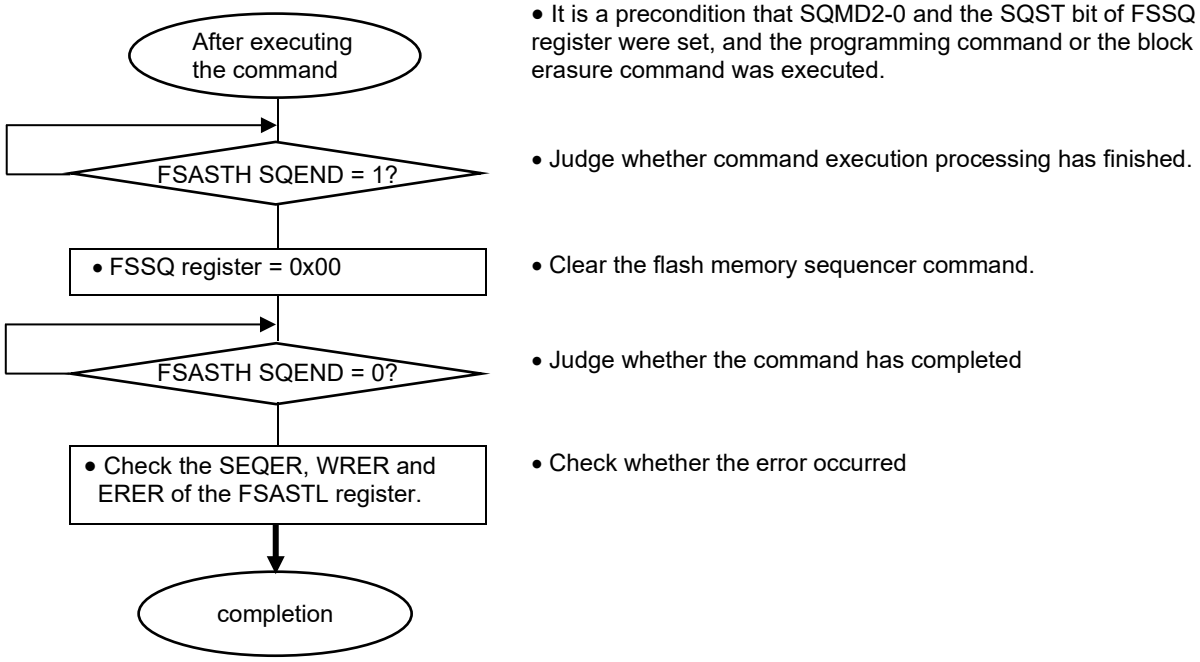


Figure 4-1 Flow of Judging the End of Command Execution in the Flash Memory Sequencer

4.4 Example of Command Execution for Reprogramming of the Flash Memory Areas

4.4.1 Example of Command Execution for Reprogramming of the Code/Data Flash memory Areas

Figure 4-2 shows the flow of command execution for reprogramming of code/data flash memory areas.

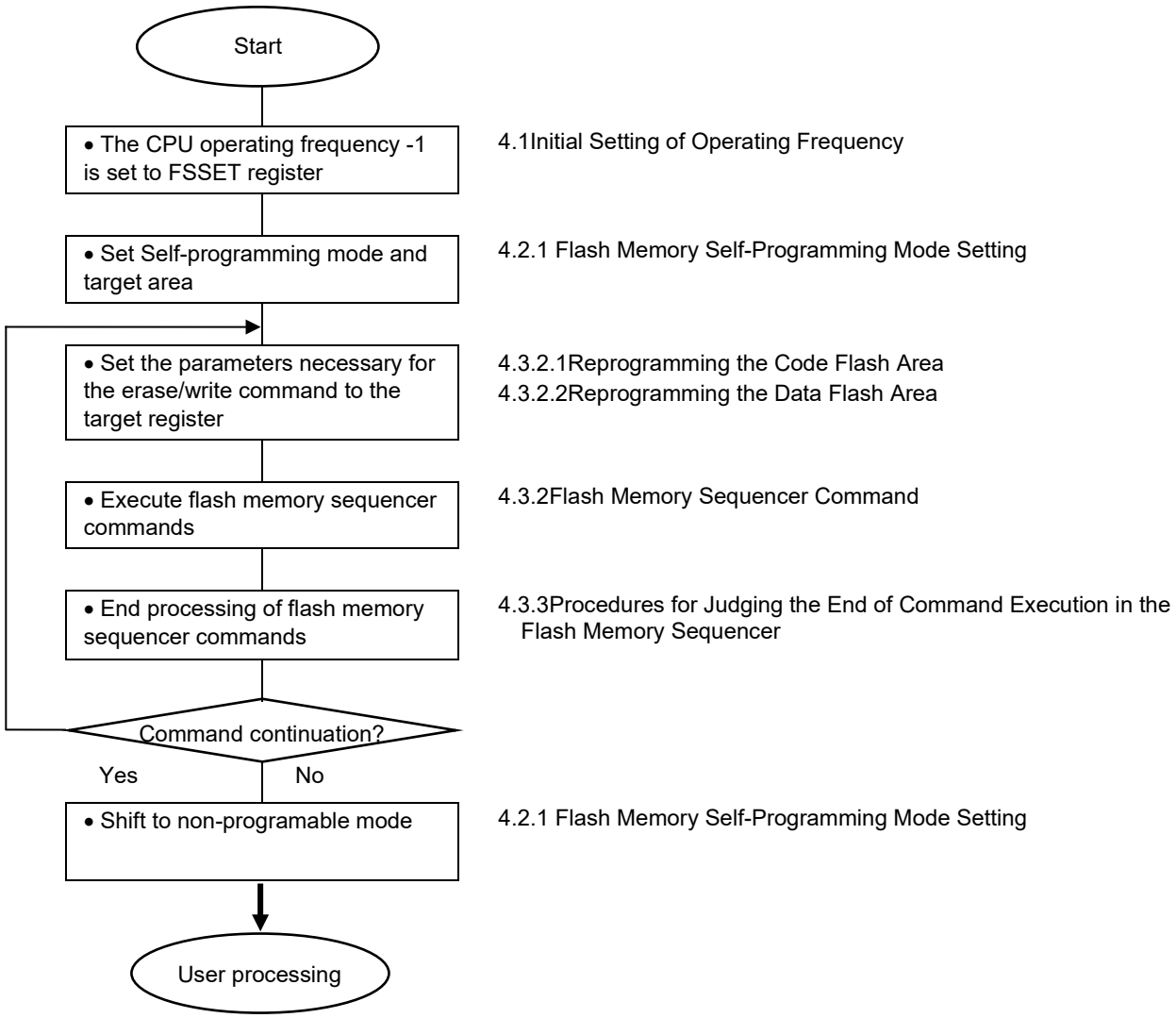


Figure 4-2 Flow of Command Execution for Reprogramming of Code/Data Flash Memory Areas

5 Sample Programs

5.1 File Structure

5.1.1 Folder configuration

This section describes the sample programs provided together with RFSP Type 01. This section explains the sample program for RL78/G15 as an example. When using other products of RL78/G15, replace "RL78/G15" with the name of target product.

- Replace the folder name for the RL78/G15 sample ("RL78_G15") with the folder name for the target product.
  The folder name in the case of using RL78/G16: "RL78_G16"

Figure 5-1 shows the structure of sample program folders.

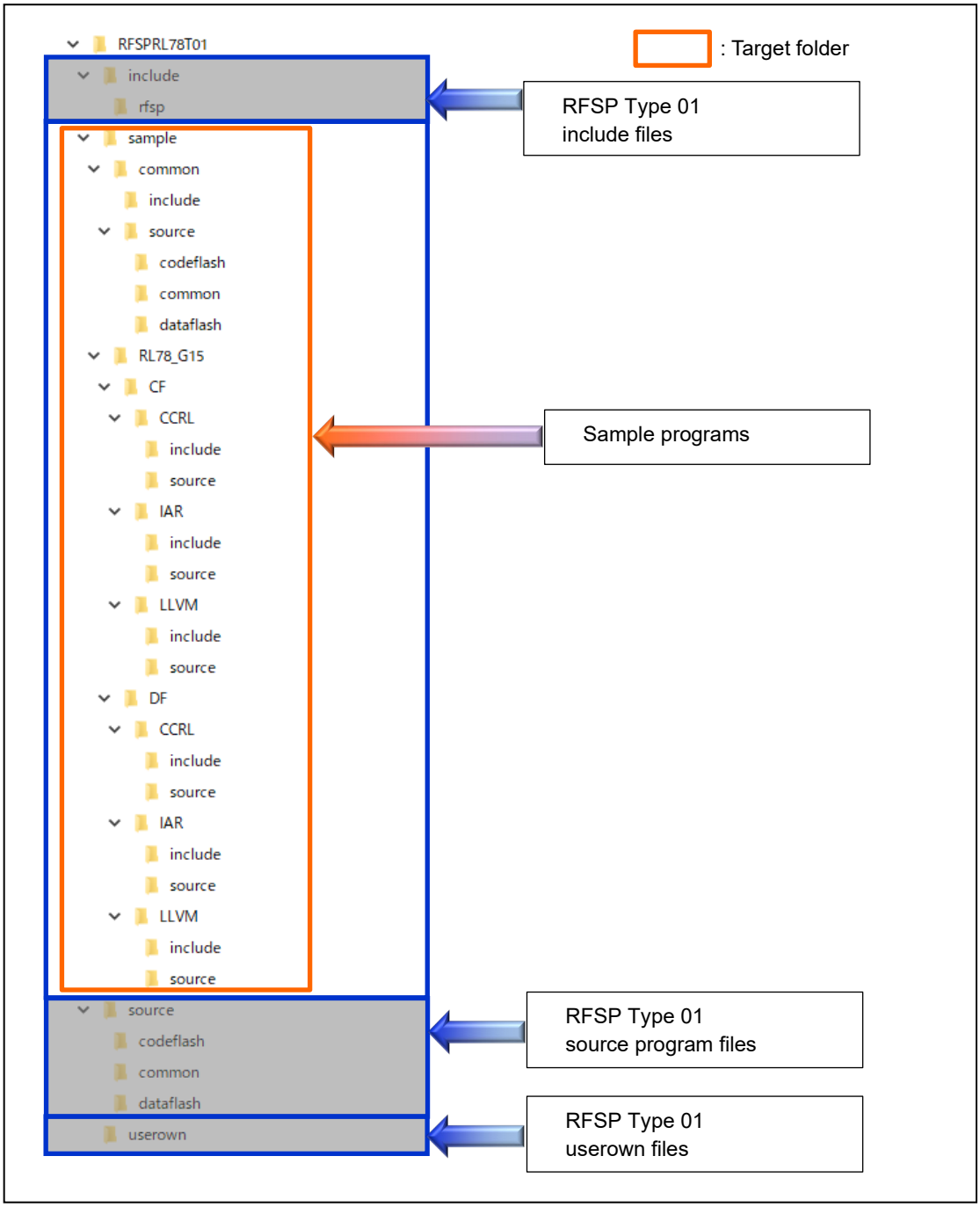


Figure 5-1 Structure of Sample Program Folders

5.1.2 List of Files

5.1.2.1 List of Source Files

Table 5-1 shows the program source file in the “sample\common\source\common\” folder.

Table 5-1 Program Source File in the “sample\common\source\common\” Folder

No.	Source File Name	Description
1	sample_control_common.c	This file contains the functions used in common for controlling the flash memory.

Table 5-2 shows the program source file in the “sample\common\source\dataflash\” folder.

Table 5-2 Program Source File in the “sample\common\source\dataflash\” Folder

No.	Source File Name	Description
1	sample_control_data_flash.c	This file contains the functions for controlling the data flash memory.

Table 5-3 shows the program source file in the “sample\common\source\codeflash\” folder.

Table 5-3 Program Source File in the “sample\common\source\codeflash\” Folder

No.	Source File Name	Description
1	sample_control_code_flash.c	This file contains the functions for controlling the code flash memory.

Table 5-4 shows the program source files of the main processing for controlling the code flash memory (CF), data flash memory (DF) in the “sample\RL78_G15\” folder.

- Main processing for controlling the code flash memory (CF) :

  "sample\RL78_G15\CF\[compiler name]\source\" folder

- main processing for controlling the data flash memory (DF) :

  "sample\RL78_G15\DF\[compiler name]\source\" folder

Table 5-4 Program Source Files of the Main Processing

No.	Source File Name	Description
1	main.c (for code flash)	Sample file of the main processing functions for controlling the code flash memory
2	main.c (for data flash)	Sample file of the main processing functions for controlling the data flash memory

5.1.2.2 List of Header File

Table 5-5 shows the program header files in the "sample\common\include\" folder.

Table 5-5 Program Header Files in the "sample\common\include\" Folder

No	Header file name	Summary
1	sample_control_common.h	This file defines the prototype declarations of the sample functions used in common for controlling the flash memory.
2	sample_control_data_flash.h	This file defines the prototype declarations of the sample functions for controlling the data flash memory.
3	sample_control_code_flash.h	This file defines the prototype declarations of the sample functions for controlling the code flash memory.
4	sample_defines.h	This file defines the macros of the sample functions for controlling the flash memory.
5	sample_types.h	This file defines the enumerated-type return values for the sample programs.

5.2 Data Type Definitions

5.2.1 Enumerations

— e_sample_ret (enumerated-type variable name: e_sample_ret_t)
Table 5-6 shows the results (normal end or error) of execution in the flash memory sequencer and the status after execution.

Table 5-6 Results (Normal End or Error) of Execution in the Flash Memory Sequencer and Status after Execution

Symbol Name	Value	Description
SAMPLE_ENUM_RET_STS_OK	0x00u	Status (Normal end)
SAMPLE_ENUM_RET_ERR_PARAMETER	0x10u	Parameter error
SAMPLE_ENUM_RET_ERR_CONFIGURATION	0x11u	Configuration error
SAMPLE_ENUM_RET_ERR_MODE_MISMATCHED	0x12u	Mode mismatch error
SAMPLE_ENUM_RET_ERR_CHECK_WRITE_DATA	0x13u	Written data comparison error
SAMPLE_ENUM_RET_ERR_CFDF_SEQUENCER	0x20u	Flash memory sequencer error
SAMPLE_ENUM_RET_ERR_ACT_ERASE	0x22u	Erase operation error
SAMPLE_ENUM_RET_ERR_ACT_WRITE	0x23u	Write operation error
SAMPLE_ENUM_RET_ERR_CMD_ERASE	0x30u	Erase command error
SAMPLE_ENUM_RET_ERR_CMD_WRITE	0x31u	Write command error

5.3 Sample Program Functions

Table 5-7 Shows the list of sample program functions.

Table 5-7 List of Sample Program Functions

	API Name	Outline
1	main Function (for code flash)	Executes the main processing of the sample program for controlling the reprogramming of the code flash memory.
2	Sample_CodeFlashControl Function	Executes the processing for reprogramming the code flash memory.
3	main Function (for data flash)	Executes the main processing of the sample program for controlling the reprogramming of the data flash memory.
4	Sample_DataFlashControl Function	Executes the processing for reprogramming the data flash memory.
5	Sample_CheckCFDFSeqEnd Function	Waits for the completion of command execution in the flash memory sequencer.

5.3.1 Sample Program for Controlling the Reprogramming of the Code Flash Memory

The sample program for controlling the reprogramming of the code flash memory in RFSP Type 01 erases block 3 (0x0C00) in the code flash area and writes 16 words (64 bytes) data from the beginning of the block.

Operating conditions:

- CPU operating frequency: 16 MHz (The high-speed on-chip oscillator clock is used for the main system clock.)
- Code flash memory address for erasure and programming: 0x0C00
- Block number for erasure: 0x03
- Size of write data: 16 words (64 bytes)

Figure 5-2 shows a flowchart of the main processing of the sample program for controlling the code flash memory reprogramming in RFSP Type 01.

5.3.1.1 main Function

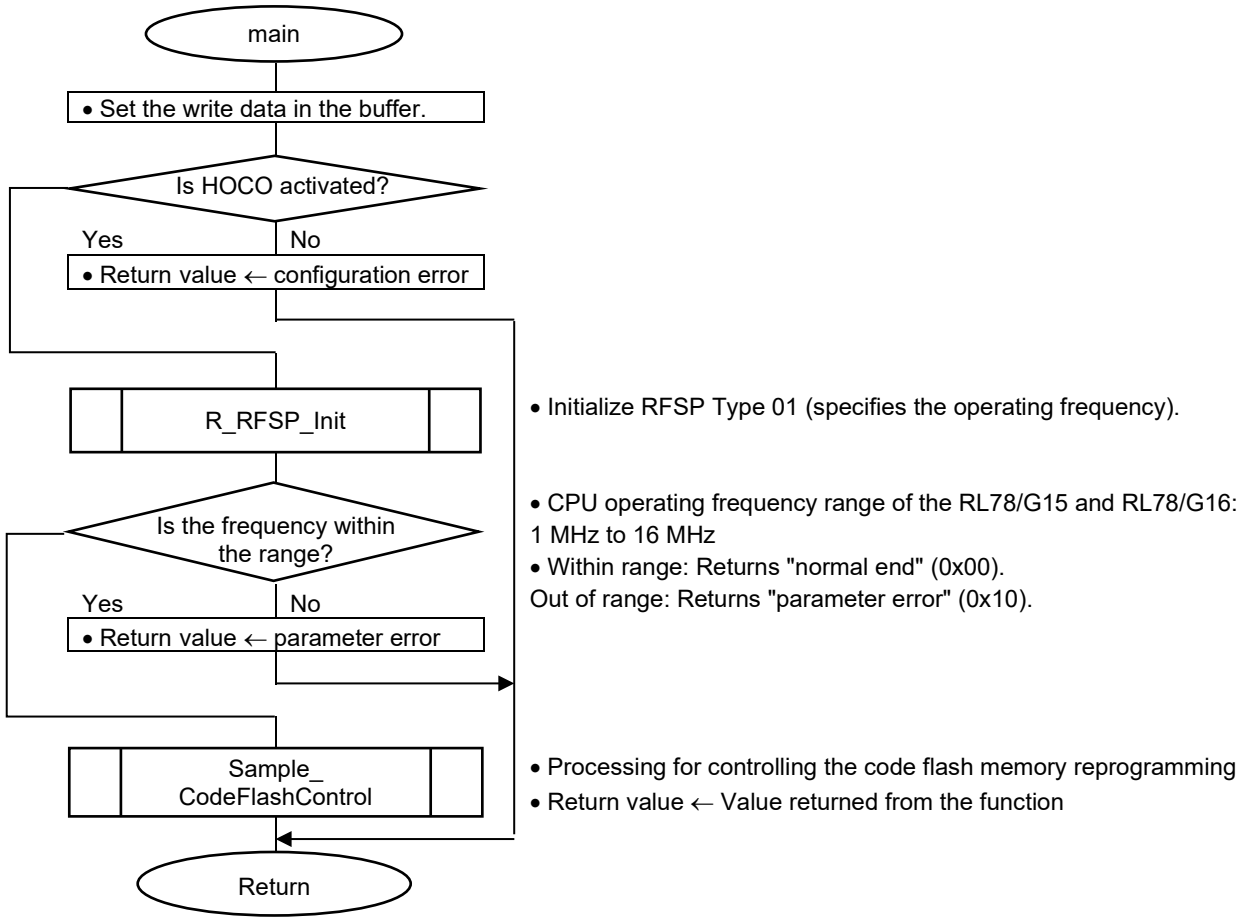


Figure 5-2 Flowchart of the Main Processing of the Sample Program for Controlling the Code Flash Memory Reprogramming

5.3.1.2 Sample_CodeFlashControl Function

・The sequencer is set self-programming mode (code flash area selection) and execute block erasure.

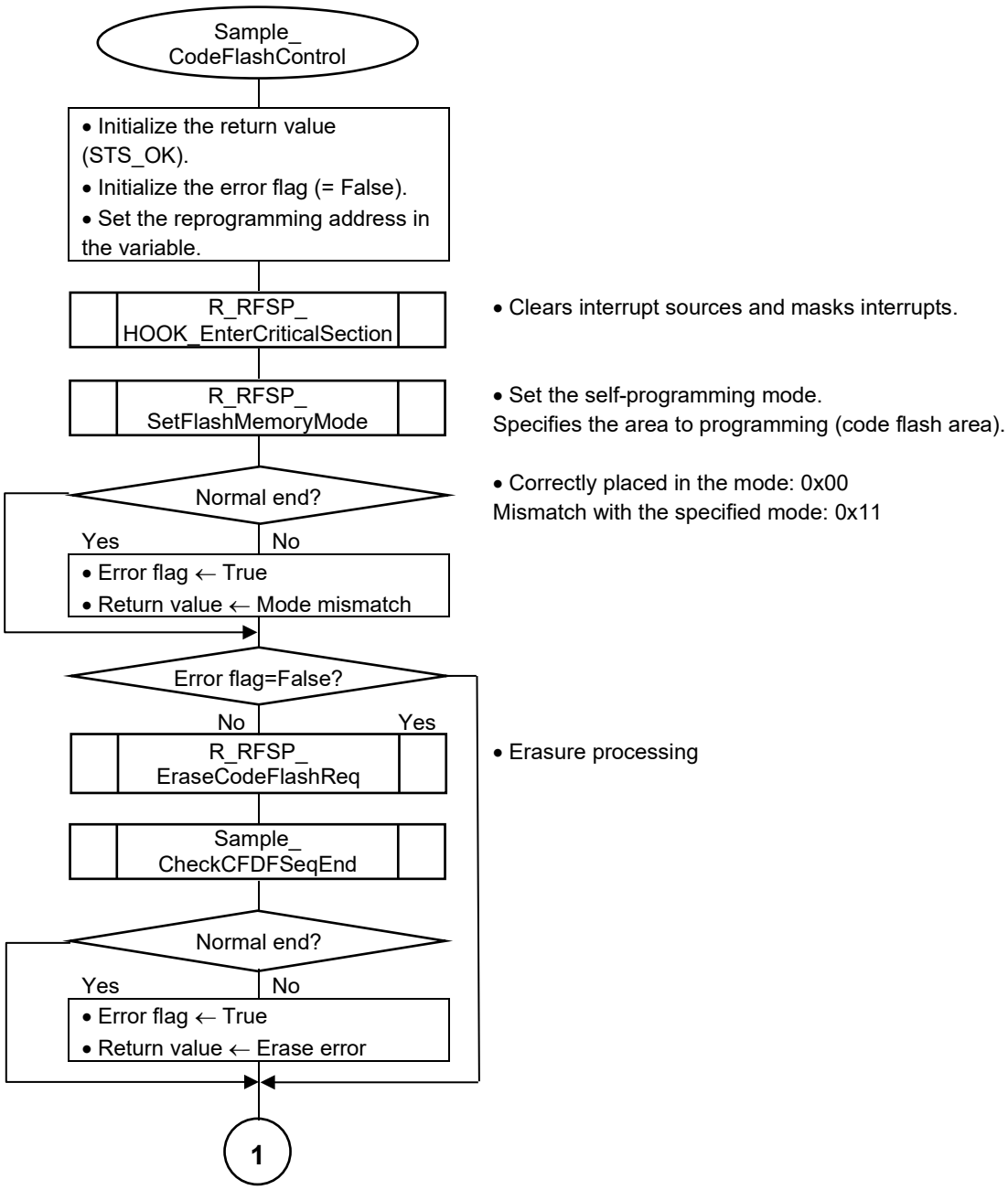


Figure 5-3 Flowchart of Sample Processing for Controlling Code Flash Memory Reprogramming (1/3)

- Programming is executed.

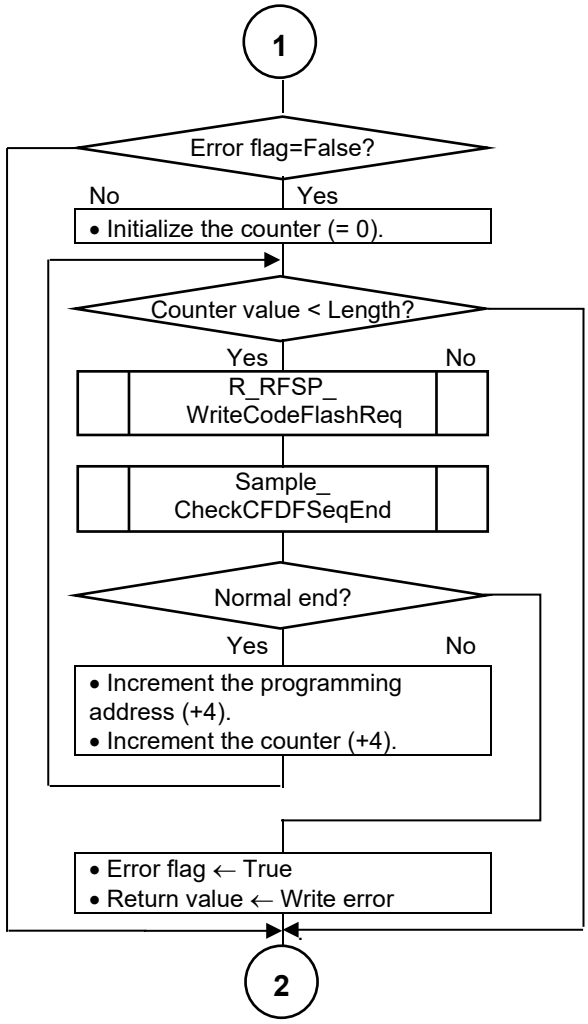
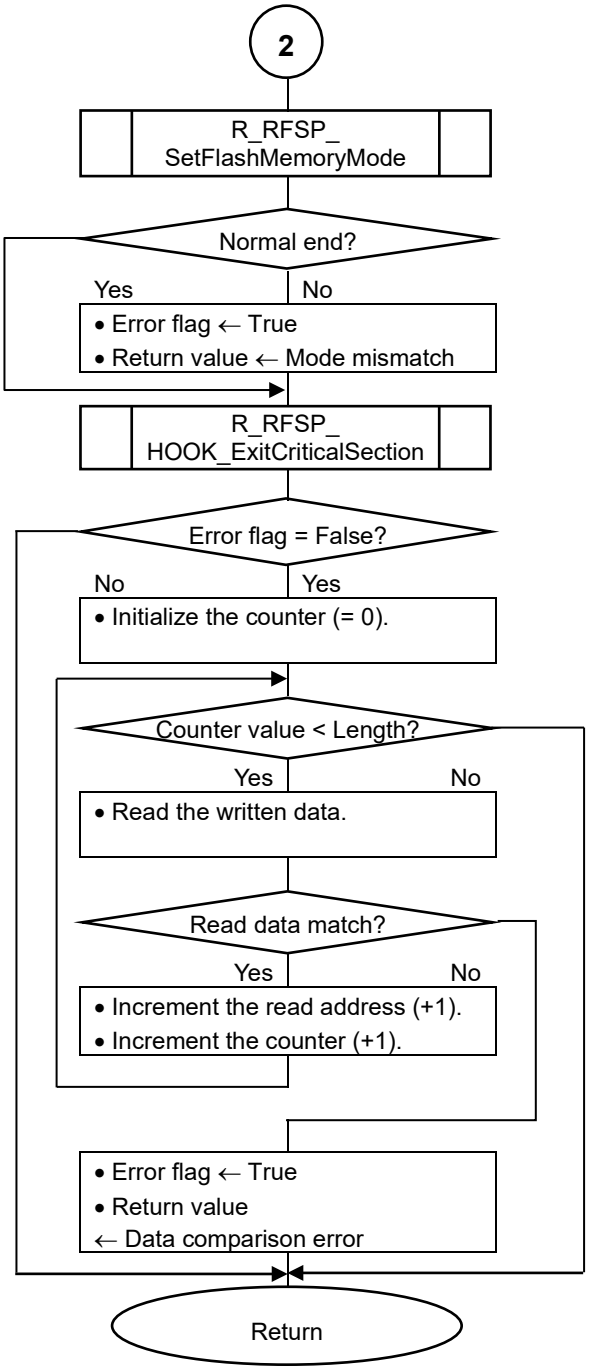


Figure 5-4 Flowchart of Sample Processing for Controlling Code Flash Memory Reprogramming (2/3)

- The sequencer in placed in the non-programmable mode and the verification check is executed through reading by the CPU.



• Specifies the non-programmable mode.

• Correctly placed in the mode: 0x00
Mismatch with the specified mode: 0x11

• Release the interrupt mask.

• Verification check through reading by the CPU

Figure 5-5 Flowchart of Sample Processing for Controlling Code Flash Memory Reprogramming (3/3)

5.3.2 Sample Program for Controlling the Reprogramming of the Data Flash Memory

The sample program for controlling the reprogramming of the data flash memory in RFSP Type 01 erases block 0 (0x9000) in the code flash area and writes 16 words (64 bytes) data from the beginning of the block.

Operating conditions:

- CPU operating frequency: 16 MHz (The high-speed on-chip oscillator clock is used for the main system clock.)
- Data flash memory address for erasure and programming: 0x9000
- Block number for erasure: 0x00
- Size of write data: 16 words (64 bytes)

Figure 5-6 shows a flowchart of the main processing of the sample program for controlling the data flash memory reprogramming in RFSP Type 01.

5.3.2.1 main Function

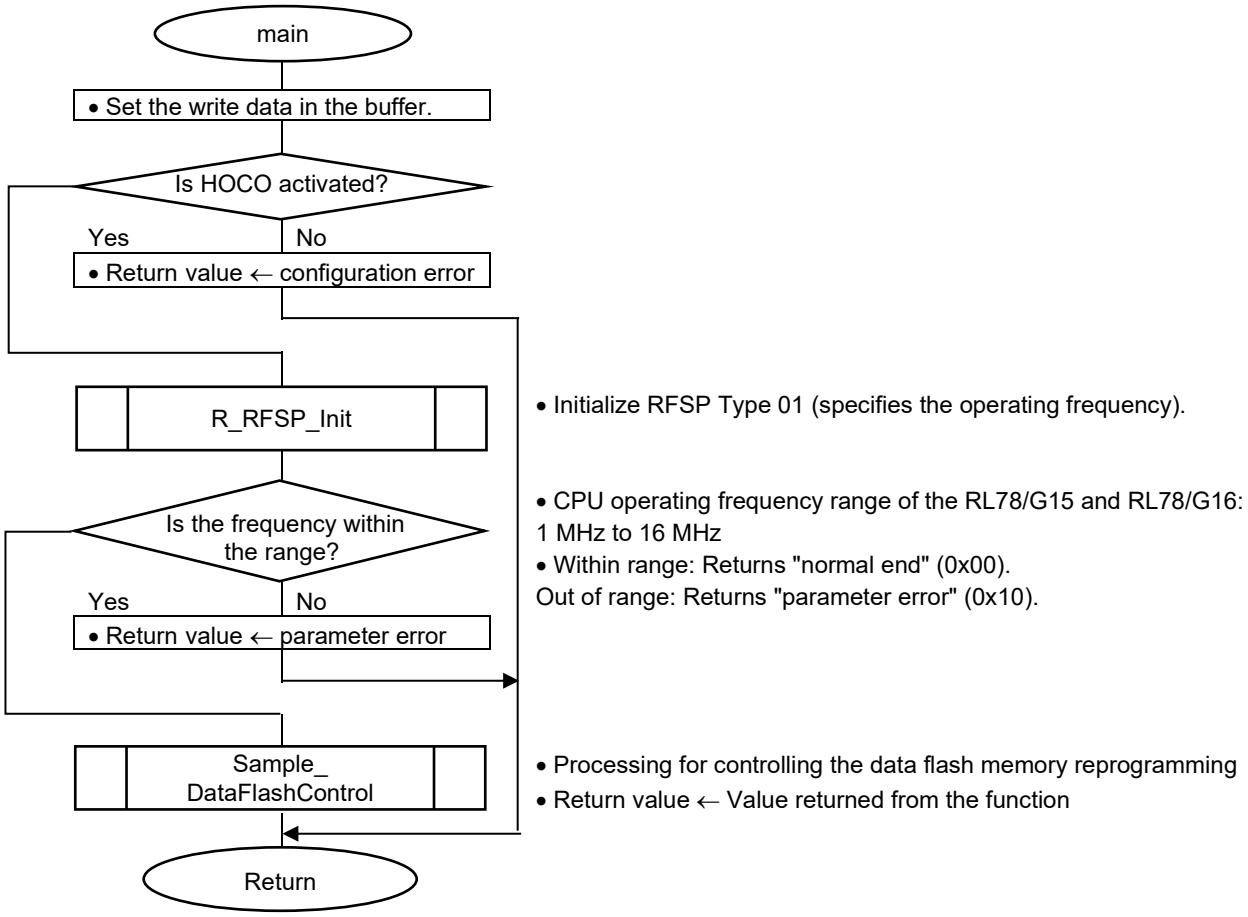


Figure 5-6 Main Processing Execution Flow of Data Flash Program Control Sample

5.3.2.2 Sample_DataFlashControl Function

・The sequencer is set self-programming mode (data flash area selection) and execute block erasure.

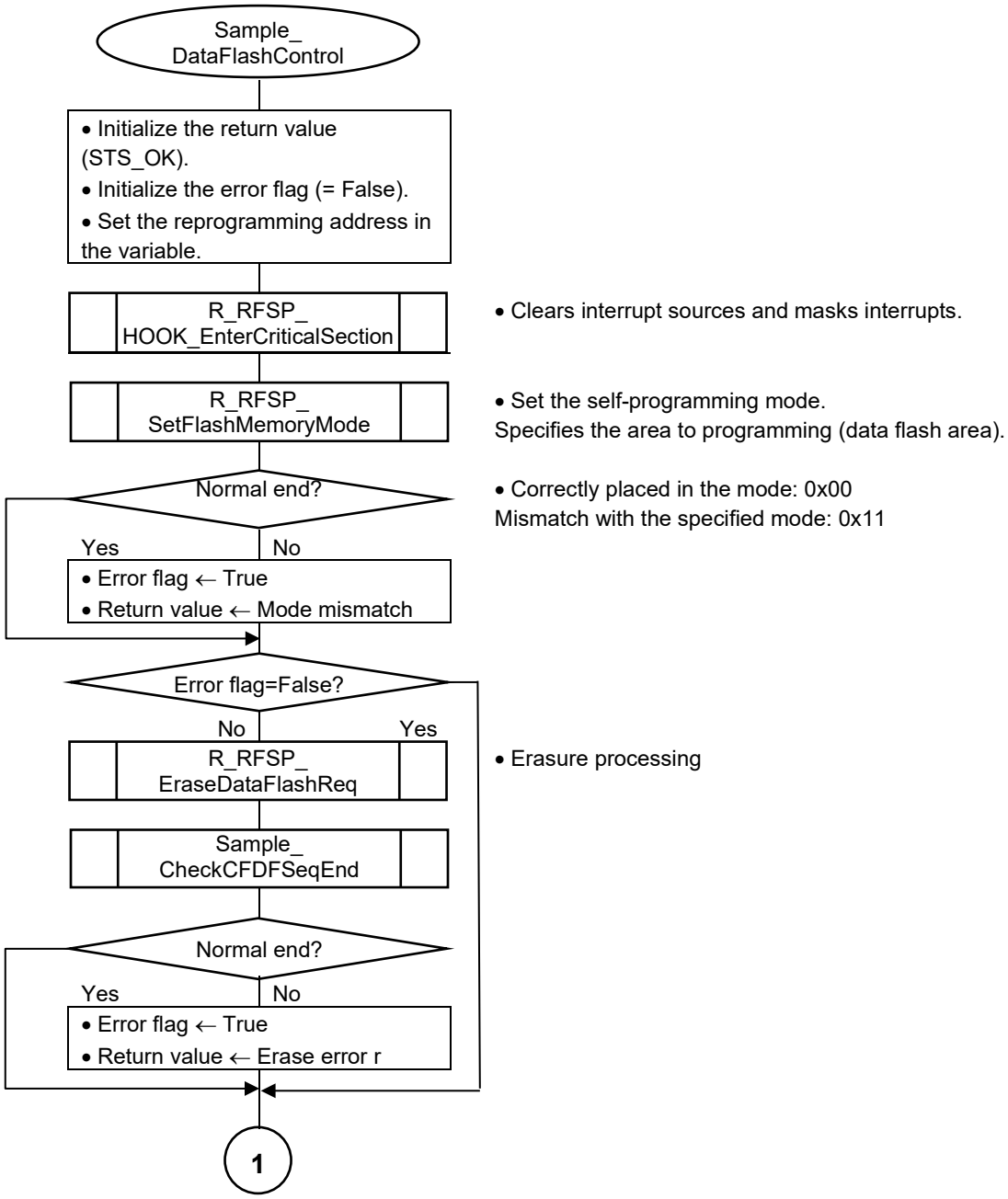


Figure 5-7 Flowchart of Sample Processing for Controlling Data Flash Memory Reprogramming (1/3)

- Programming is executed.

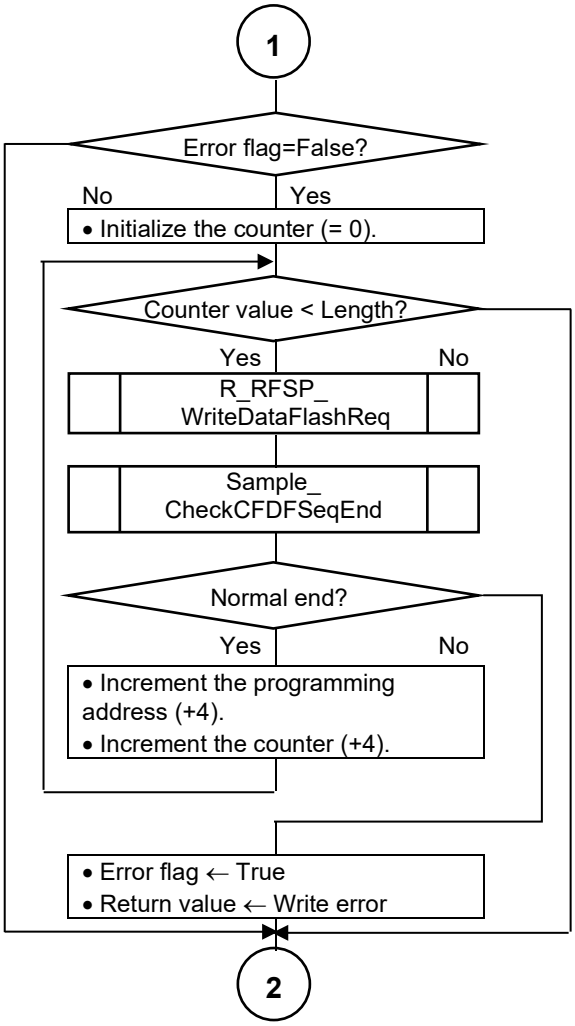
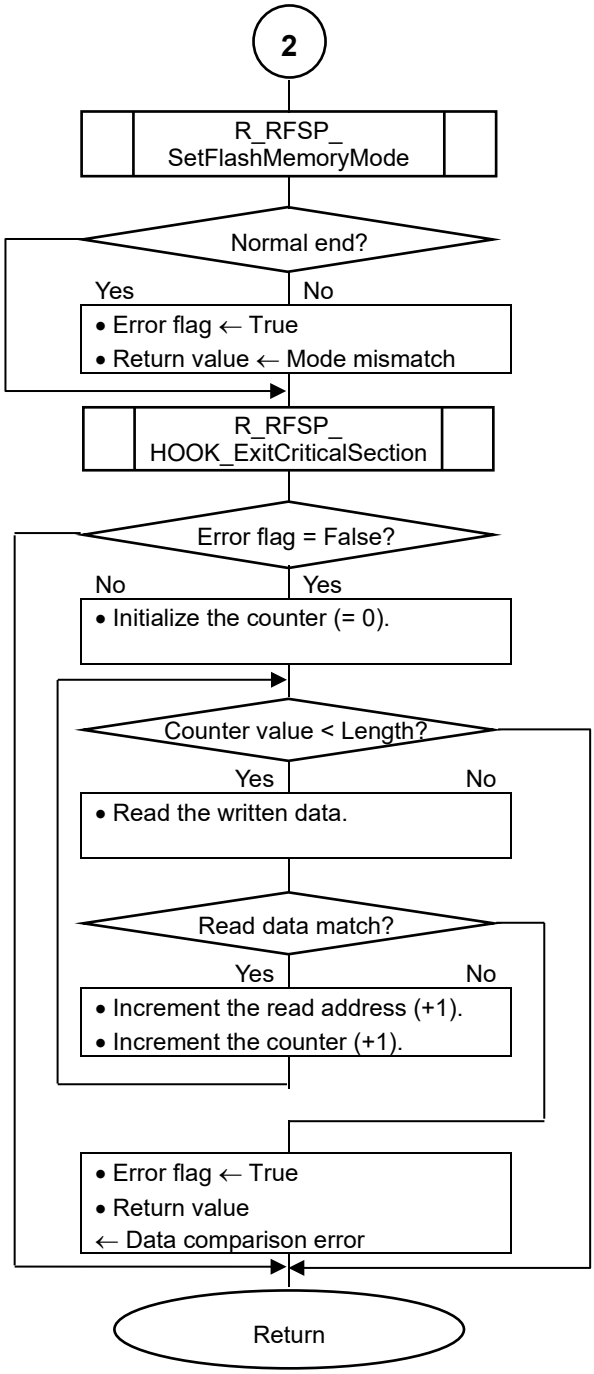


Figure 5-8 Flowchart of Sample Processing for Controlling Data Flash Memory Reprogramming (2/3)

- The sequencer in placed in the non-programmable mode and the verification check is executed through reading by the CPU.



• Specifies the non-programmable mode.

• Correctly placed in the mode: 0x00
Mismatch with the specified mode: 0x11

• Release the interrupt mask.

• Verification check through reading by the CPU

Figure 5-9 Flowchart of Sample Processing for Controlling Data Flash Memory Reprogramming (3/3)

5.3.3 Sample Program Used in Common for Controlling the Flash Memory

5.3.3.1 Sample_CheckCFDFSeqEnd Function

- The end of the operation of the activated flash memory sequencer is confirmed and the execution result is returned.

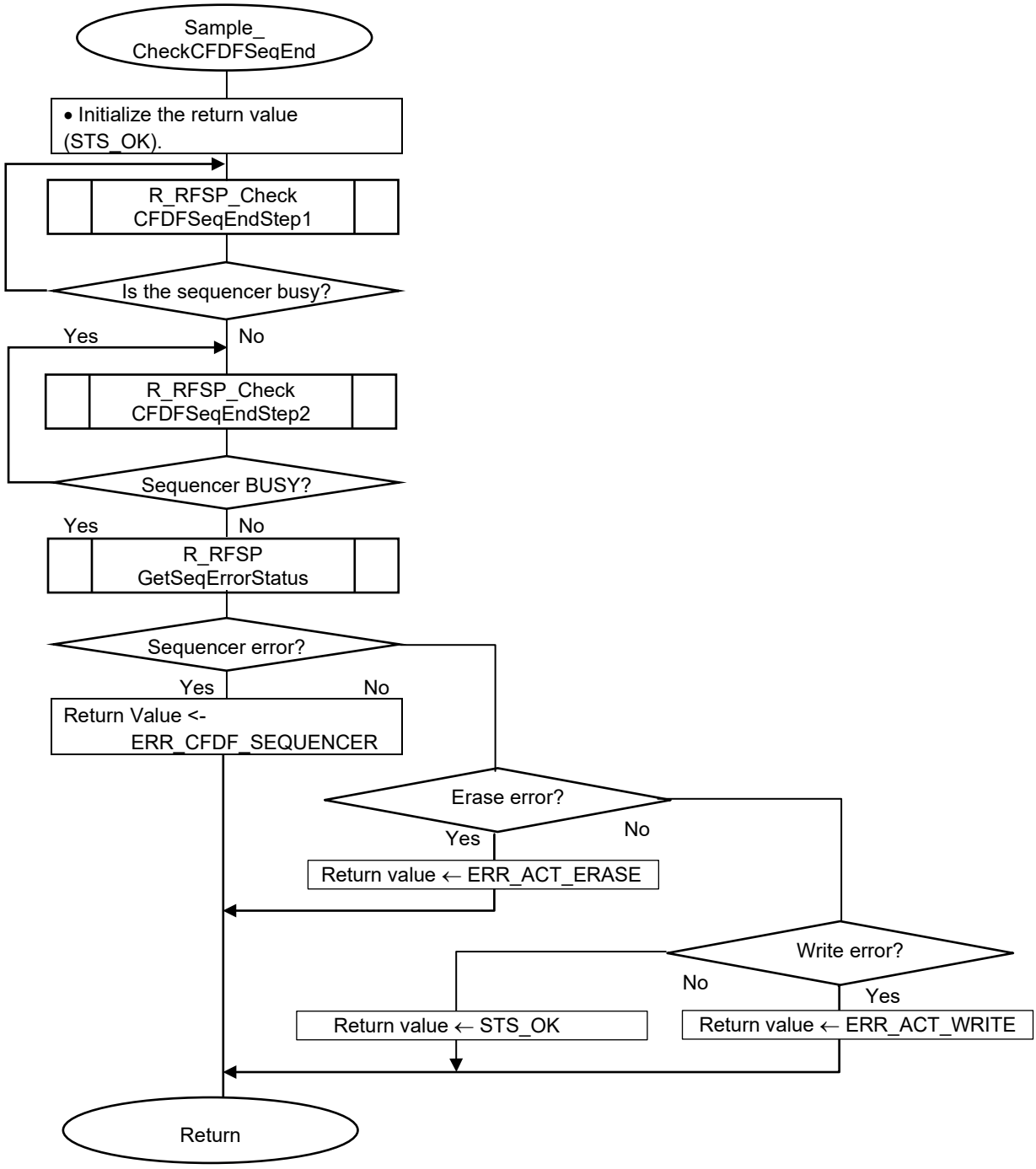


Figure 5-10 Flowchart of Sample_CheckCFDFSeqEnd Function

5.4 Specifications of Sample Program Functions

This section describes the specifications of the functions in the sample programs for RFSP Type 01.

The sample programs for RFSP Type 01 are examples of basic processing for reprogramming the code flash area, and data flash area. The functions in the sample programs can be used as reference for developing an application program that reprograms these areas.

Please be sure to thoroughly check the operation of the developed application program.

5.4.1 Sample Program Functions for Controlling the Reprogramming of the Code Flash Memory

5.4.1.1 main

Information:

Syntax	Int main(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_sample_ret_t	SAMPLE_ENUM_RET_STS_OK : 0x00 [Normal end] SAMPLE_ENUM_RET_ERR_PARAMETER : 0x10 [Parameter error] SAMPLE_ENUM_RET_ERR_CONFIGURATION : 0x11 [Configuration error] SAMPLE_ENUM_RET_ERR_MODE_MISMATCHED : 0x12 [Mode mismatch error] SAMPLE_ENUM_RET_ERR_CMD_ERASE : 0x30 [Erase command error] SAMPLE_ENUM_RET_ERR_CMD_WRITE : 0x31 [Write command error] SAMPLE_ENUM_RET_ERR_CHECK_WRITE_DATA : 0x13 [Written data comparison error]
Description	Executes the main processing of the sample program for controlling the reprogramming of the code flash memory.	
Preconditions	Execute this function in the non-programmable mode while the high-speed on-chip oscillator is active.	
Remarks	-	

5.4.1.2 **Sample_CodeFlashControl**

Information:

Syntax	R_RFSP_FAR_FUNC e_sample_ret_t Sample_CodeFlashControl (uint16_t i_u16_start_addrr, uint16_t i_u16_write_data_length, uint8_t __near * inp_u08_write_data);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint16_t i_u16_start_addr	Start address of the area to be reprogrammed
	uint16_t i_u16_write_data_length	Size of the reprogram data
	uint8_t __near * inp_u08_write_data	Pointer to the reprogram data buffer
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_sample_ret_t	SAMPLE_ENUM_RET_STS_OK : 0x00 [Normal end] SAMPLE_ENUM_RET_ERR_MODE_MISMATCHED : 0x12 [Mode mismatch error] SAMPLE_ENUM_RET_ERR_CMD_ERASE : 0x30 [Erase command error] SAMPLE_ENUM_RET_ERR_CMD_WRITE : 0x31 [Write command error] SAMPLE_ENUM_RET_ERR_CHECK_WRITE_DATA : 0x13 [Written data comparison error]
Description	Executes the processing for reprogramming the code flash memory. — The erase and write commands are executed in the self-programming mode (code flash area selection). — The written data are read in the non-programmable mode to check that the data have been written correctly.	
Preconditions	Execute this function in the non-programmable mode while the high-speed on-chip oscillator is active.	
Remarks	-	

5.4.2 Sample Program Functions for Controlling the Reprogramming of the Data Flash Memory

5.4.2.1 main

Information:

Syntax	Int main(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_sample_ret_t	SAMPLE_ENUM_RET_STS_OK : 0x00 [Normal end] SAMPLE_ENUM_RET_ERR_PARAMETER : 0x10 [Parameter error] SAMPLE_ENUM_RET_ERR_CONFIGURATION : 0x11 [Configuration error] SAMPLE_ENUM_RET_ERR_MODE_MISMATCHED : 0x12 [Mode mismatch error] SAMPLE_ENUM_RET_ERR_CMD_ERASE : 0x30 [Erase command error] SAMPLE_ENUM_RET_ERR_CMD_WRITE : 0x31 [Write command error] SAMPLE_ENUM_RET_ERR_CHECK_WRITE_DATA : 0x13 [Written data comparison error]
Description	Executes the main processing of the sample program for controlling the reprogramming of the data flash memory.	
Preconditions	Execute this function in the non-programmable mode while the high-speed on-chip oscillator is active.	
Remarks	-	

5.4.2.2 **Sample_DataFlashControl**

Information:

Syntax	R_RFSP_FAR_FUNC e_sample_ret_t Sample_DataFlashControl (uint16_t i_u16_start_addrr, uint16_t i_u16_write_data_length, uint8_t __near * inp_u08_write_data);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint16_t i_u16_start_addr	Start address of the area to be reprogrammed
	uint16_t i_u16_write_data_length	Size of the reprogram data
	uint8_t __near * inp_u08_write_data	Pointer to the reprogram data buffer
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_sample_ret_t	SAMPLE_ENUM_RET_STS_OK : 0x00 [Normal end] SAMPLE_ENUM_RET_ERR_MODE_MISMATCHED : 0x12 [Mode mismatch error] SAMPLE_ENUM_RET_ERR_CMD_ERASE : 0x30 [Erase command error] SAMPLE_ENUM_RET_ERR_CMD_WRITE : 0x31 [Write command error] SAMPLE_ENUM_RET_ERR_CHECK_WRITE_DATA : 0x13 [Written data comparison error]
Description	Executes the processing for reprogramming the data flash memory. — The erase and write commands are executed in the self-programming mode (data flash area selection). — The written data are read in the non-programmable mode to check that the data have been written correctly.	
Preconditions	Execute this function in the non-programmable mode while the high-speed on-chip oscillator is active.	
Remarks	-	

5.4.3 **Sample Program Functions Used in Common**

5.4.3.1 **Sample_CheckCFDFSeqEnd**

Information:

Syntax	R_RFSP_FAR_FUNC e_sample_ret_t Sample_CheckCFDFSeqEnd(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_sample_ret_t	SAMPLE_ENUM_RET_STS_OK : 0x00 [Normal end] SAMPLE_ENUM_RET_ERR_CFDF_SEQUENCER : 0x20 [Flash memory sequencer error] SAMPLE_ENUM_RET_ERR_ACT_ERASE : 0x22 [Erase operation error] SAMPLE_ENUM_RET_ERR_ACT_WRITE : 0x23 [Write operation error]
Description	Waits for the completion of command execution in the flash memory sequencer.	
Preconditions	Execute this function in the self-programming mode (code flash area selection) or self-programming mode (data flash area selection) while the high-speed on-chip oscillator is active.	
Remarks	-	

6 Creating a Sample Project for RFSP Type 01

RFSP Type 01 includes sample programs for a code flash memory area and a data flash memory area to program. The compilers which can be used by RFSP Type 01 are CC-RL compiler, IAR compiler and LLVM compiler. Users can create a sample project using the Integrated Development Environment(IDE) corresponding to each compiler.

This section is explained in the sample program example for RL78/G15. When using a device other than RL78/G15, read G15 to the target device. Section address settings must be changed by referring to the user's manual for the target device.

Note : The target Integrated Development Environment(IDE) and the compiler are premised on using the version for RL78/G15 and RL78/G16. Be sure to use them, after confirming that RL78/G15 or RL78/G16 are target products.

6.1 Creating a Project in the case of Using CC-RL Compiler

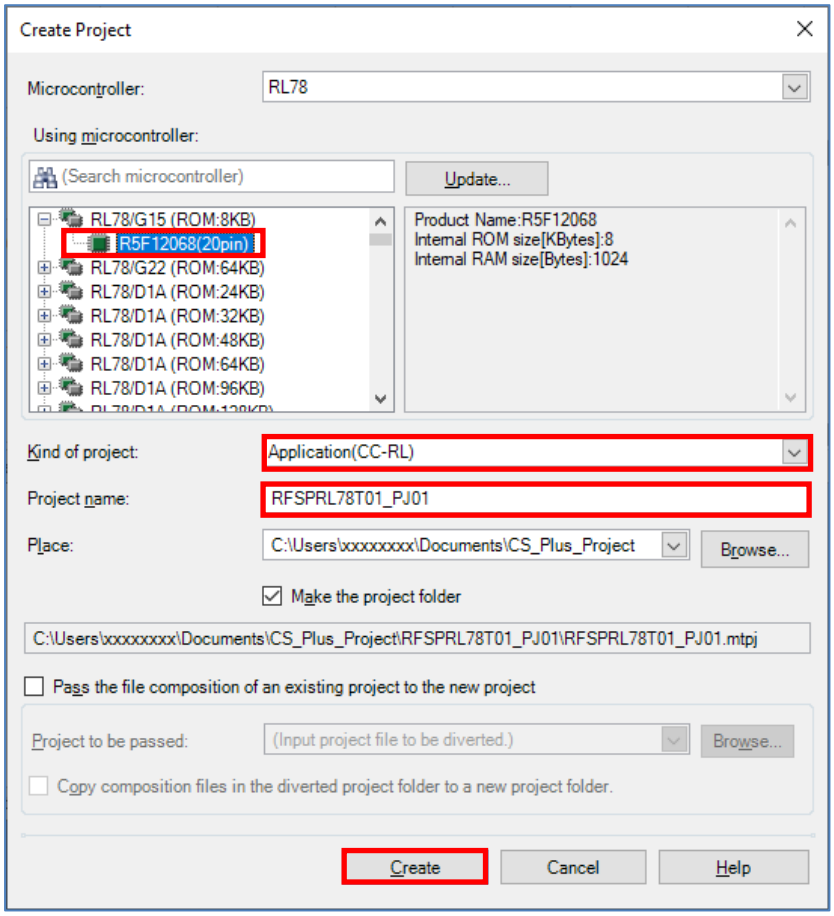
CS+ or e^2 studio can be used for RENESAS CC-RL compiler as an IDE. RFSP Type 01 is registered and built in the project created by the IDE. An example of creating a sample project in case each IDE is used is shown. Because to understand CC-RL compiler and each IDE, it is necessary to refer to the user's manual of each tool product.

6.1.1 Example of Creating a Sample Project

(1) An example of creating a sample project which used CS+ (IDE)

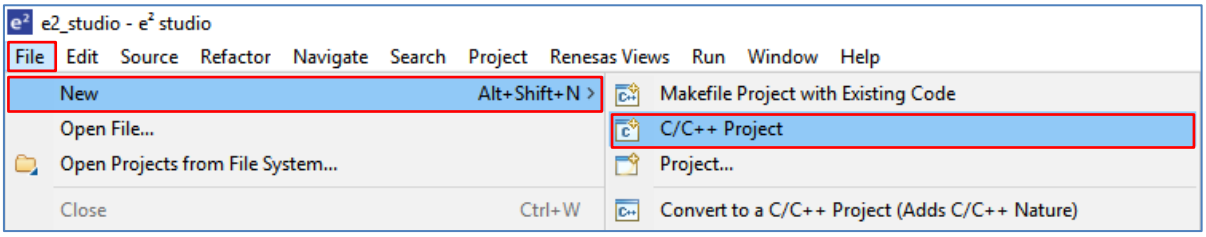
・The CS+ starts and from the [Project] menu, select [Create New Project...], the “Create Project” window will open.

- Select the product of ”RL78/G15 (ROM: 8KB)” - “R5F12068(20pin)” as [Using microcontroller].
- Select "Application(CC-RL)" as [Kind of project].
- [Project name] is temporarily set to "RFSPRL78T01_PJ01".
- When you click the [Create] button, the new project is created.

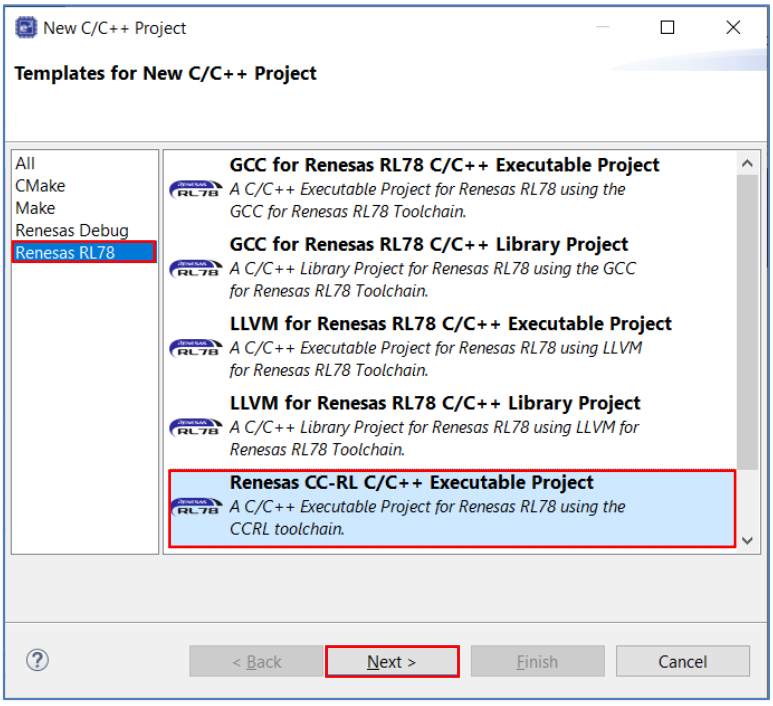


(2) An example of creating a sample project which used e² studio (IDE)

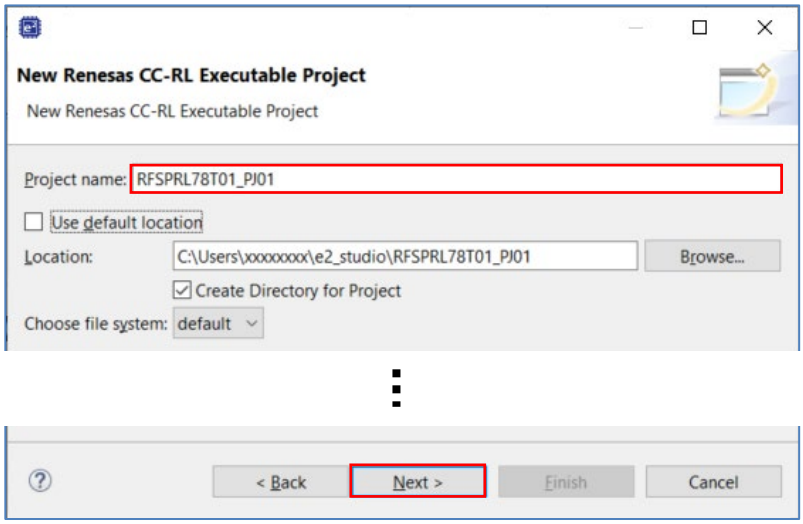
・The e² studio starts and from the [File] menu, select [New] – [C/C++ Project], the “Templates for New C/C++ Project” window will open.



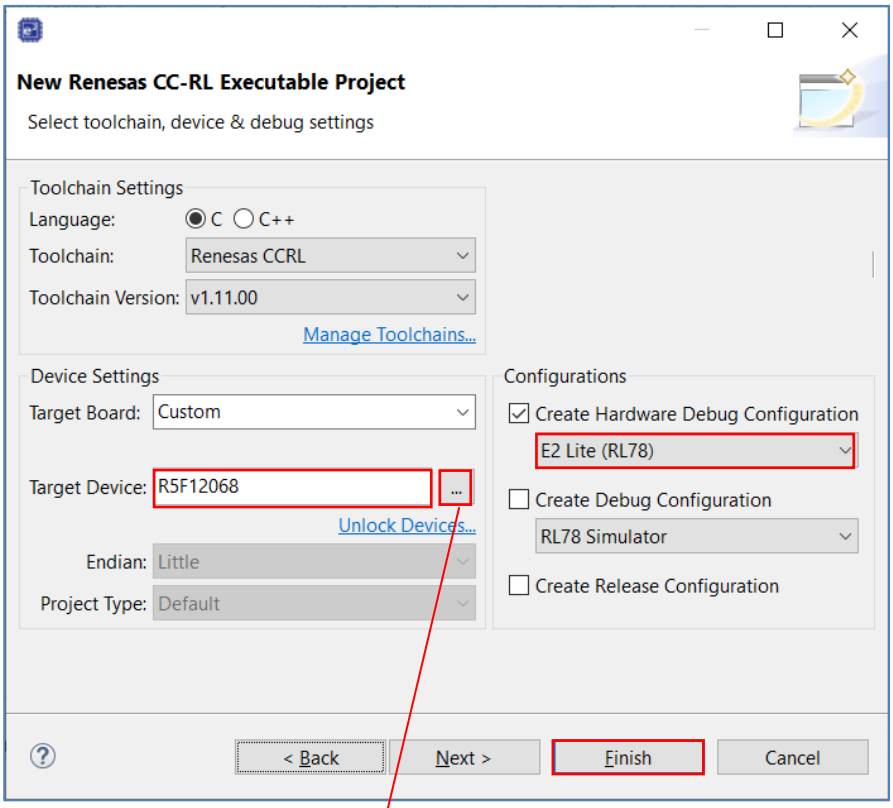
・Select [Renesas CC-RL C Executable Project] displayed after selection in [Renesas RL78], and press "Next" button.

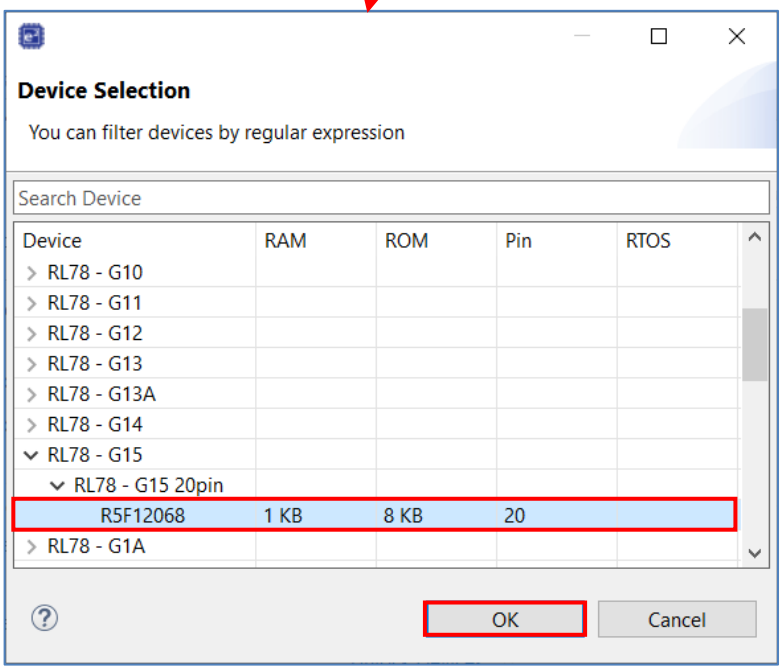


・Input "project name" on “New Renesas CC-RL Executable Project" window, and press "next" button. [Project name] is temporarily set to "RFSPRL78T01_PJ01".



- Select the [Target Device] of [Device Settings], and select ”RL78 - G15 20pin - R5F12068”.
- It is a premise that E2 Lite is selected as a debugging tool and on-chip debugging is executed. Put a check mark to "Create Hardware Debug Configuration" by [Configurations]. And select "E2 Lite(RL78)”.
- Press "Finish" button.

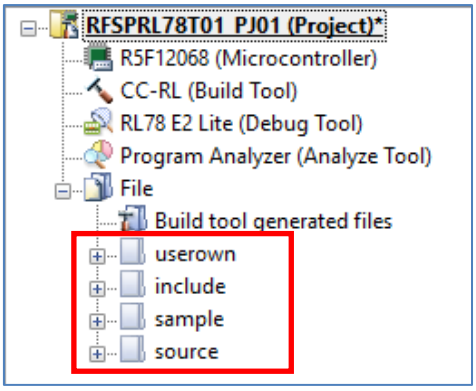




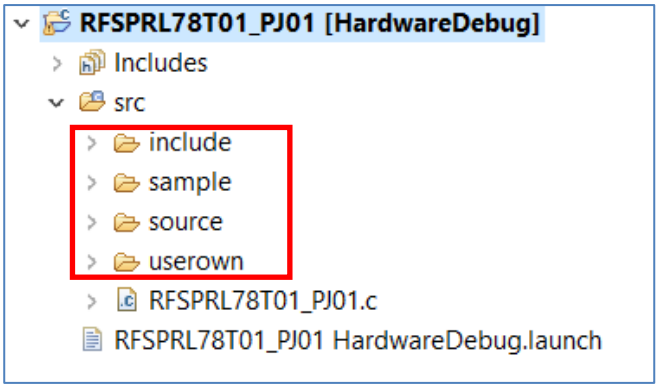
6.1.2 Example of Registration of Target Folders and the Target Files

Using RFSP Type 01, when programming each area [(1) code flash memory, (2) data flash memory], the example which registers necessary files is shown. Each folder of RFSP Type 01 source-program file is "include", "source", "userown", and "sample". The target file in each folder is selected and registered by the area programmed.

As other registration methods, after all the folders of "include", "source", "userown", and "sample" are registered, unnecessary files and folders can be removed using the function of "Remove from Project"(CS+) or [Resource Configuration] – [Exclude from Build] (e^2 studio).



The registration tree screen of RFSP (CS+)



The registration tree screen of RFSP (e^2 studio)

- Registration of the latest I/O header file(iodefine.h) outputted to target products by IDE

  "iodefine.h" is an I/O header file which CS+ or e^2 studio outputs to target products. Replacing instead of "iodefine.h" included in RFSP Type 01 is recommended. Registration of target folders and target files is implemented. Then, a user replaces "iodefine.h" which IDE outputted with "iodefine.h" included in RFSP Type 01.

  The folder to which an I/O header file (iodefine.h) is outputted by IDE :
  - CS+ : [Project name] Folder
  - e^2 studio : [Project name]/generate Folder

  The folder with which a user replaces the "iodefine.h" file :
  - The case of code flash programming : ”\[Project name]\sample\RL78_G15\CF\CCRL\include”
  - The case of data flash programming : ”\[Project name]\sample\RL78_G15\DF\CCRL\include”

- Exclusion of the file automatically added by the function of IDE.

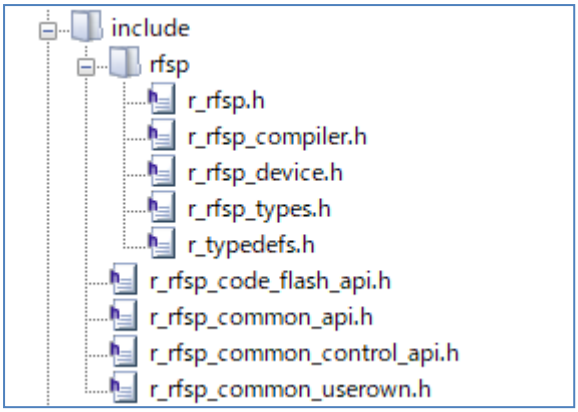
  There are files added automatically in the created project. The same file as these exists also in the "sample" folder of RFSP Type 01. Therefore, using the function of IDE, select those files from tree and excludes from a project.
  - CS+ : Click the right mouse button for the file of tree. And exclude target file using "Remove from Project" function. Targets are "cstart.asm, hdwinit.asm, stkinit.asm, main.c, and iodefine.h" in [project name] folder.
  - e^2 studio : Clicks the right mouse button for the file of tree. And, on the [Settings] screen displayed by the "property", put a check mark to [Exclude resource from build] and exclude a target file (target folder). (Exclusion of a folder is also possible)

Target files are cstart.asm, hdwinit.asm, iodefine.h, and stkinit.asm in a [project name] / generate folder. And [project name] .c ("RFSPRL78T01_PJ01.c") in a [project name] / src folder is a target.

(1) Registration of the folders and files of the target in the case of reprogramming code flash memory

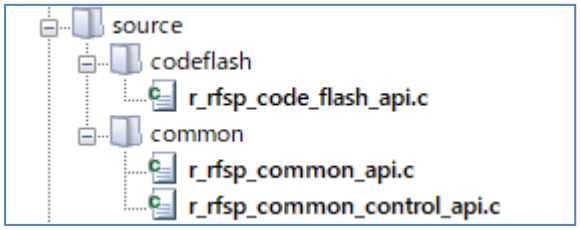
The folders ("include", "source", "userown", "sample") and source program file which are included in RFSP Type 01 to register are shown below.

in the “include” folder



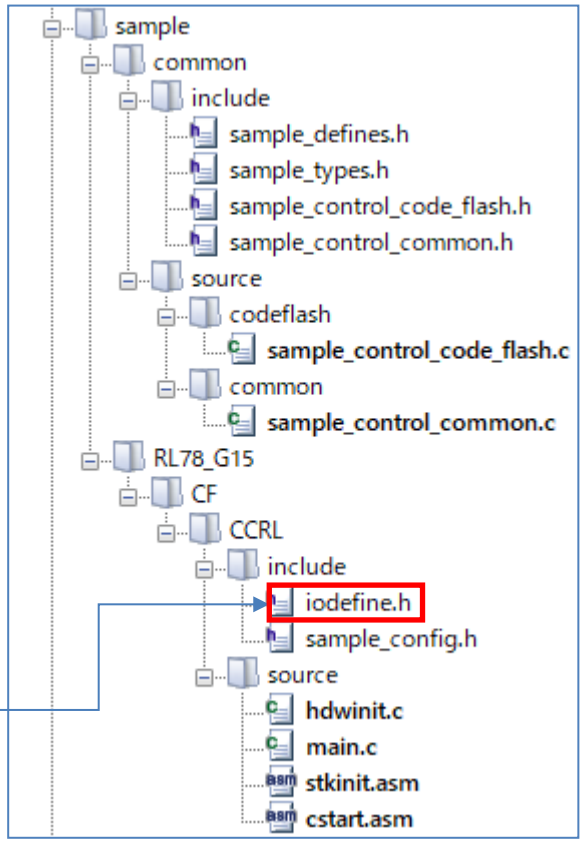
in the “userown” folder

In the “source” folder



Transpose to "iodefine.h" outputted by CS+ or e^2 studio.

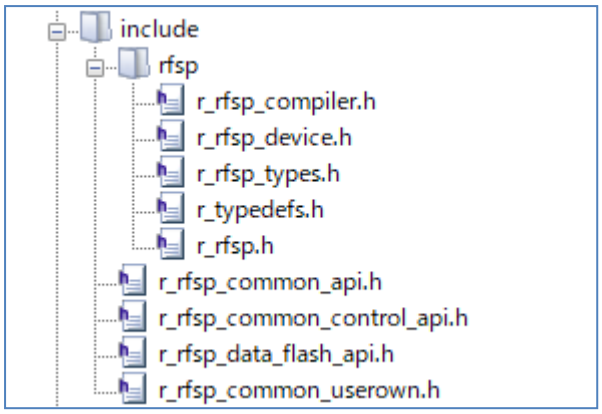
in the “sample” folder



(2) Registration of the folders and files of the target in the case of reprogramming data flash memory

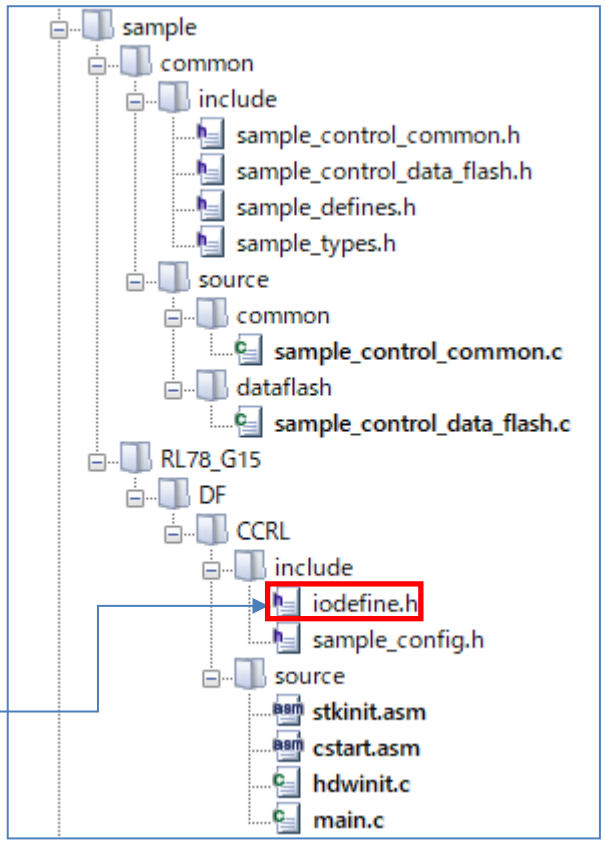
The folders ("include", "source", "userown", "sample") and source program file which are included in RFSP Type 01 to register are shown below.

in the “include” folder

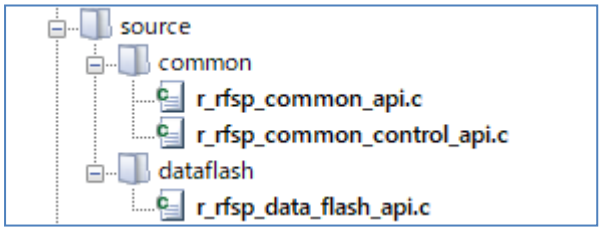


in the “userown” folder

In the “sample” folder



In the “source” folder



Transpose to "iodefine.h" outputted by CS+ or e^2 studio.

6.1.3 **Build Tool Settings**

Set IDE setting necessary to build RFSP Type 01 using CC-RL compiler.

CS+ : Click the right mouse button for “CC-RL(Build tool)” in a tree, and select "Property". And set each setting of the build tool in the displayed window.

e^2 studio : Click the right mouse button for the project(”RFSPRL78T01_PJ01“) in a tree, and select "Property". And set each setting of the build tool in the displayed window.

6.1.3.1 **Include Path Settings**

・Setting of the include path on CS+ inputs path in “Common Options” tab. (Change by a target area)

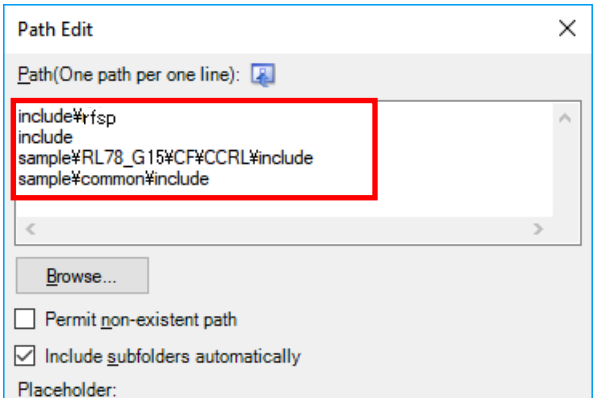
- Input the Include directory path in the ” Path Edit” window displayed by selection of [Frequently Used Options(for Compile)] - [Additional include paths].

(1) Code flash memory reprogramming

include\rfsp
include
sample\RL78_G15\CF\CCRL\include
sample\common\include

(2) Data flash memory reprogramming

include\rfsp
include
sample\RL78_G15\DF\CCRL\include
sample\common\include



・Setting of the include path on e^2 studio inputs path in “Properties” window. (Change by a target area)

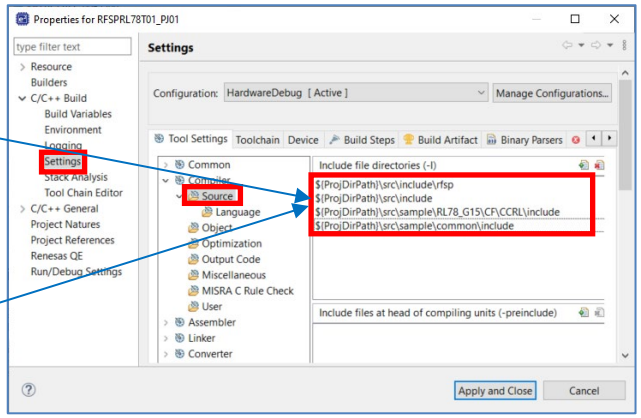
- Input the Include directory path in the window displayed by selection of ”C/C++” build [Setting] - “Compiler” [Source].

(1) Code flash memory reprogramming

${ProjDirPath}\src\include\rfsp
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_G15\CF\CCRL\include
${ProjDirPath}\src\sample\common\include

(2) Data flash memory reprogramming

${ProjDirPath}\src\include\rfsp
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_G15\DF\CCRL\include
${ProjDirPath}\src\sample\common\include



6.1.3.2 Device Item Settings

・Setting of the device Items on CS+ inputs in the “Link Options” tab. (Common in each area)

- Setting the [Device] items

Select "Yes (-OCDBG)" in [Set enable/disable on-chip debug by link option].

Note : The example of a setting on condition of on-chip debugging execution.

Input the "85" into [Option byte values for OCD]. (Example of permission of operation for on-chip debugging)

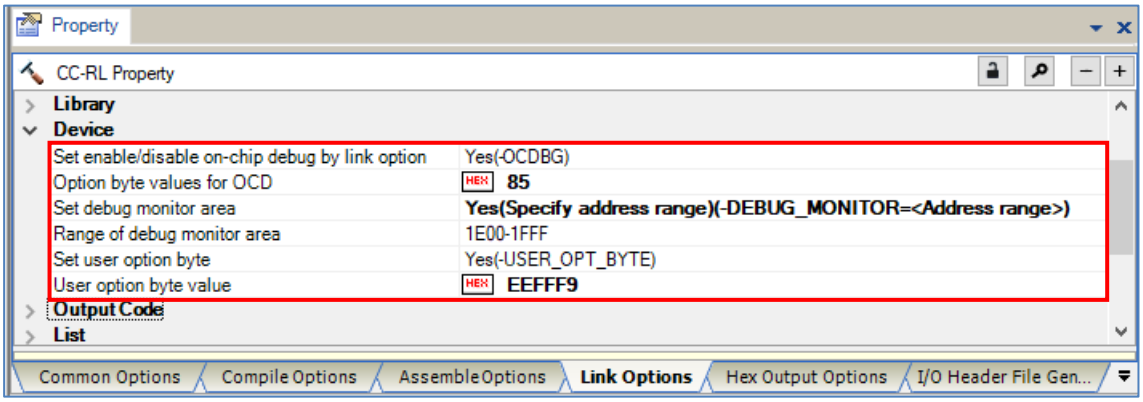
Note : Be sure to confirm the contents of "User option byte" of the chapter of "Option Bytes", and "On-chip debug option byte" on the user's manual of a target device. And describe the set value used with user application.

Select " Yes(Specify address range)(-OCDBG_MONITOR=<Address range>)" in [Set debug monitor area]. Set "1E00-1FFF" to [Range of debug monitor area].

Note : The user needs to input the range of the area which the debugger uses with reference to description of the user's manual for a target device. And please refer to “Memory Spaces Allocated for Use by the Monitor Program for Debugging” in “Allocation of Memory Spaces to User Resources” on a user's manual.

Select " Yes(-USER_OPT_BYTE)" in [Set user option byte].
Set ”EEFFF9” to [User option byte value]. (WDT stop, P125:RESET input, SPOR detection voltage:2.16V/2.11V, 16MHz [The example for RL78/G15,RL78/G16])

Note : Be sure to confirm the contents of "User option byte" of the chapter of "Option Bytes", and "User option bytes" on the user's manual of a target device. And describe the set value used with user application.



・Setting of the device Items on e^2 studio inputs in the “Properties” window. (Common in each area)

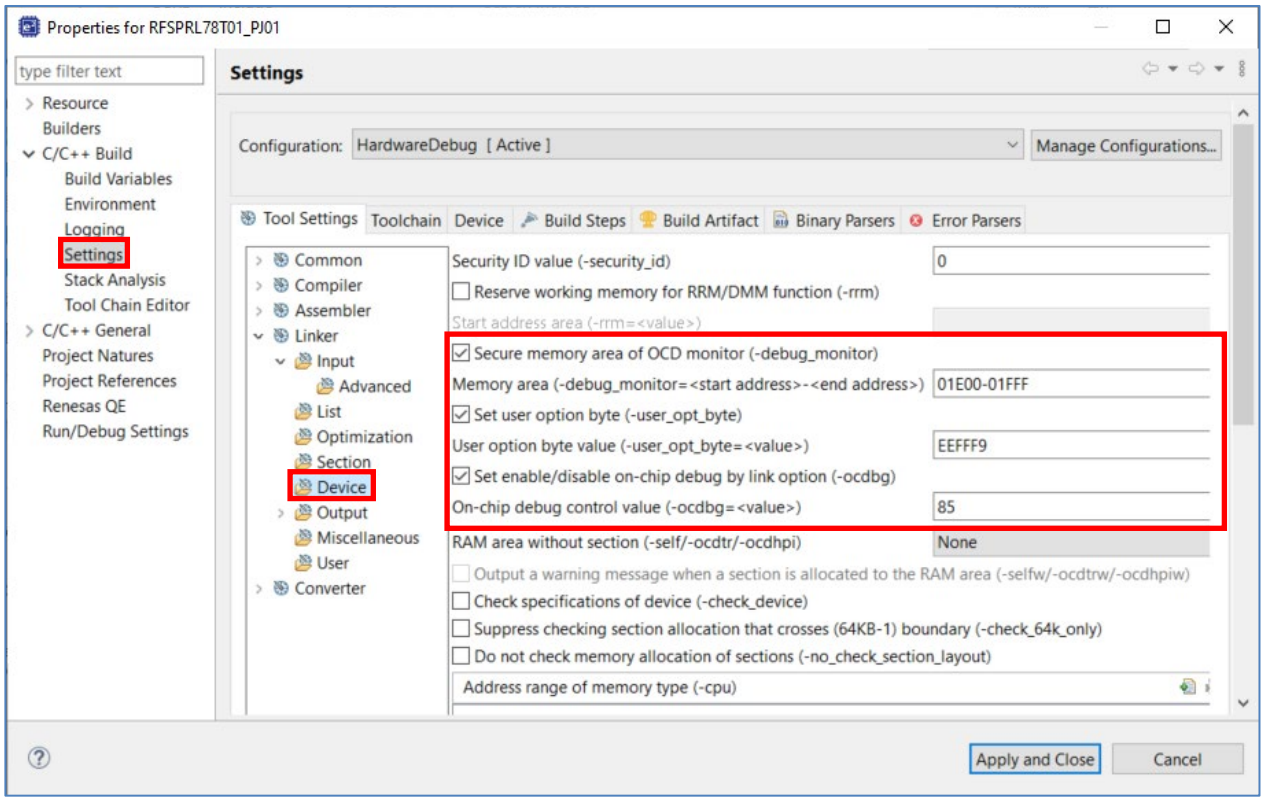
- Select ”C/C++ Build” [Setting] - “Linker” [Device]. And set device items on the displayed screen.
Put in a check mark to [Secure memory area of OCD monitor(-debug_monitor)] in the screen.
Note : The example of a setting on condition of on-chip debugging execution.

Set "1E00-1FFF" to [Memory area(-debug_monitor=<start address>-<end address>)].
Note : The user needs to input the range of the area which the debugger uses with reference to description of the user's manual for a target device. And please refer to “Memory Spaces Allocated for Use by the Monitor Program for Debugging” in “Allocation of Memory Spaces to User Resources” on a user's manual.

Put a check mark to [Set user option byte(-user_opt_byte)].
Set ”EEFFF9” to [User option byte value(-user_opt_byte=<value>)]. (WDT stop, P125:RESET input, SPOR detection voltage:2.16V/2.11V, 16MHz [The example for RL78/G15,RL78/G16])
Note : Be sure to confirm the contents of "User option byte" of the chapter of "Option Bytes", and "User option bytes" on the user's manual of a target device. And describe the set value used with user application.

Put a check mark to [Set enable /disable on-chip debug by link option(-ocdbg)].
Note : The example of a setting on condition of on-chip debugging execution.

Input the "85" into [On-chip debug control value(-ocdbg=<value>)]. (Example of permission of operation for on-chip debugging)
Note : Be sure to confirm the contents of "User option byte" of the chapter of "Option Bytes", and "On-chip debug option byte" on the user's manual of a target device. And describe the set value used with user application.



6.1.4 Debug Tool Settings

This section describes the contents of connection setting on a target board necessary to execute on-chip debugging. As a debugging tool, it is a premise that E2 Lite is selected. Refer to the user's manual for each IDE for the details of other debugging tool setting.

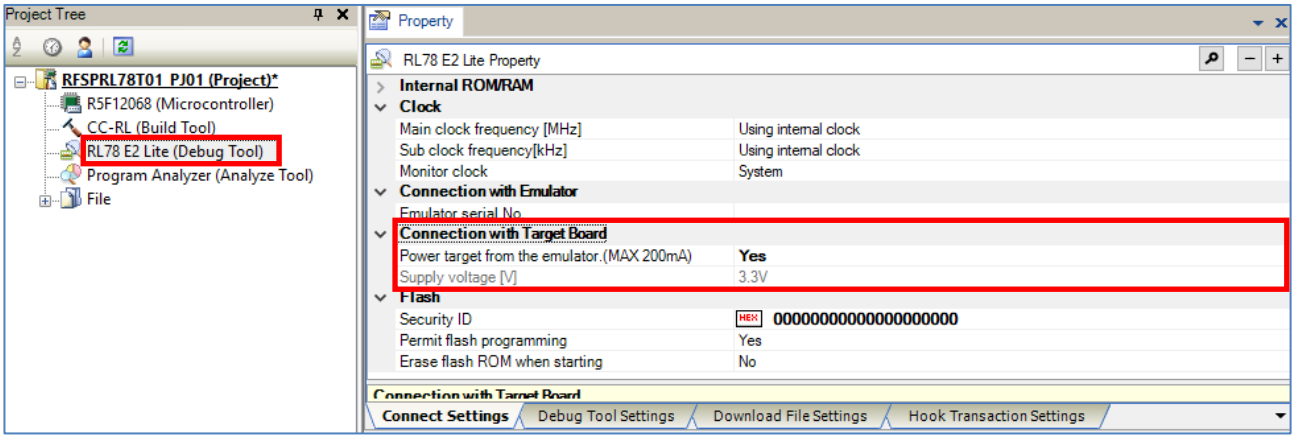
On CS+, right-click a mouse by "RL78 simulator (Debug Tool)" [initial setting] of a tree. And select the "RL78 E2 Lite" by "Using Debug Tool" displayed there. After that, right-click a mouse again, select "Property" and the "RL78 E2 Lite Property" screen will be displayed. And select each tab, and perform debugging tool setting.

On e^2 studio, right-click a mouse in the target project of a tree. Selection of [Debug As] - [Debug Configurations…] will display the "Debug Configurations" screen. On the tree of a screen, select the target project ("RFSPRL78T01_PJ01 HardwareDebug") of [Renesas GDB Hardware Debugging]. And the displayed "Debugger" tab performs debugging tool setting.

Note: The power is already supplied to the target board, or when power supply capacity is insufficient, the emulator including E2 Lite may be unable to supply power to a target board. Be sure to refer to "the user's manual and Additional Document for User's Manual (Notes on Connection of RL78)" for the emulator for target devices, and use an emulator.

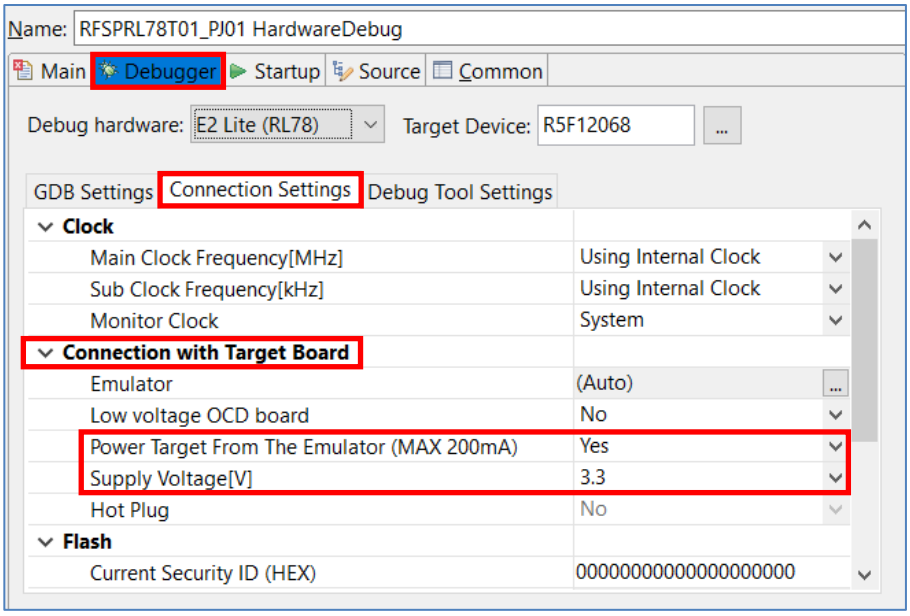
6.1.4.1 Setting of Connection with Target Board

- On CS+, set up the connection with target board(via E2 Lite) with "Connect Settings" tab. (Common in each area)
  - [Connection with Target Board] item
    To let power supply(Supply voltage : 3.3V) from E2 Lite to a target board, it is necessary to set "Yes" to [Power target from the emulator (MAX 200mA)].



・On e^2 studio, set up the connection with target board(via E2 Lite) with "Connect Settings" tab. (Common in each area)

- [Connection with Target Board] item
  To let power supply(Supply Voltage : 3.3V) from E2 Lite to a target board, it is necessary to set "Yes" to [Power Target From The Emulator (MAX 200mA)].



6.2 Creating a Project in the case of Using IAR Compiler

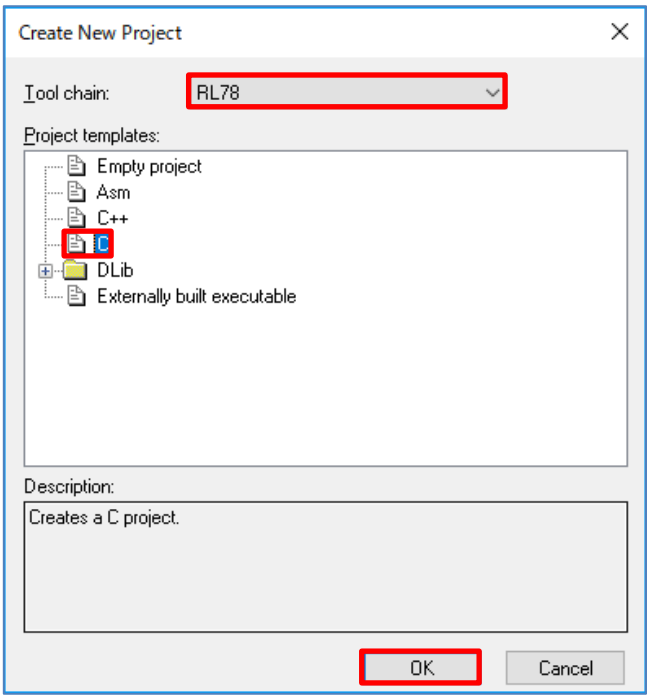
IAR Embedded Workbench can be used for IAR compiler as an IDE. RFSP Type 01 is registered and built in the project created by the IDE. An example of creating a sample project in case each IDE is used is shown. Because to understand IAR compiler and each IDE, it is necessary to refer to the user's manual of each tool product.

6.2.1 Example of Creating a Sample Project

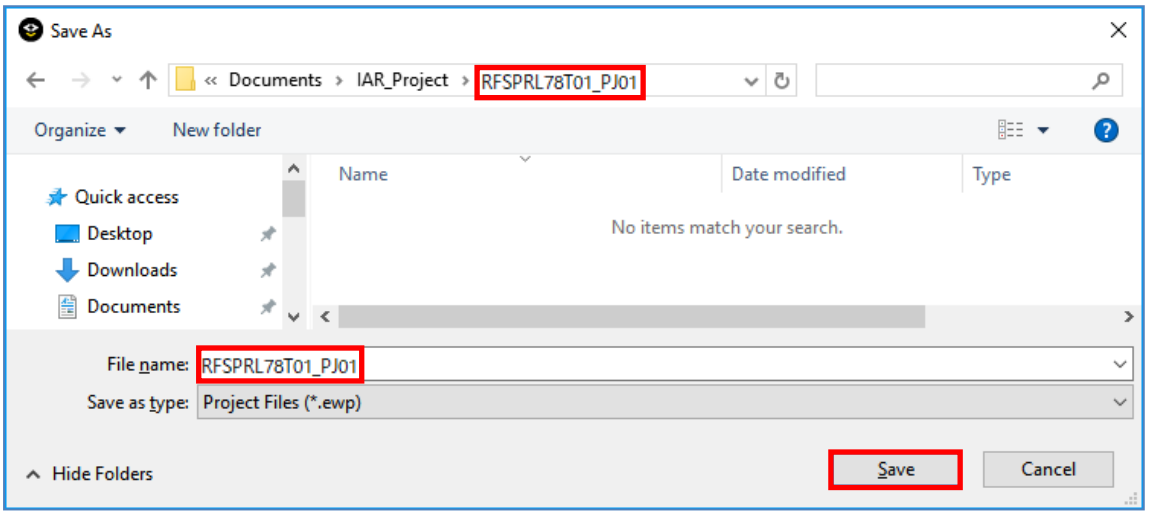
(1) An example of creating a sample project which used IAR Embedded Workbench (IDE)

・IAR Embedded Workbench starts and from the [Project] menu, select [Create New Project...], the “Create Project” window will open.

- Select the "C" as [project template].
- When you click the [OK] button, the “Save As” window will open.

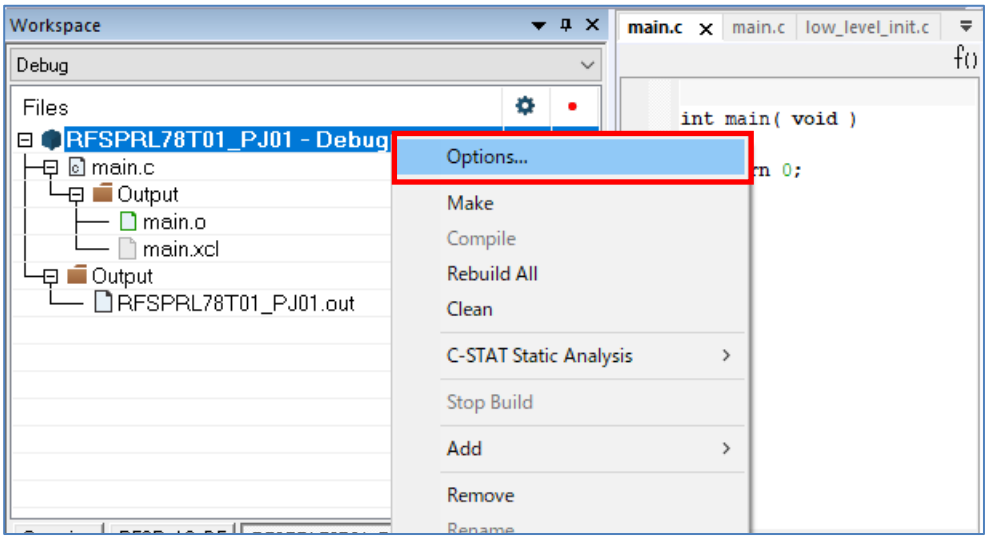


- Create "RFSPRL78T01_PJ01" folder temporarily, and move into a folder.
- The Project File name is temporarily set to "RFSPRL78T01_PJ01".

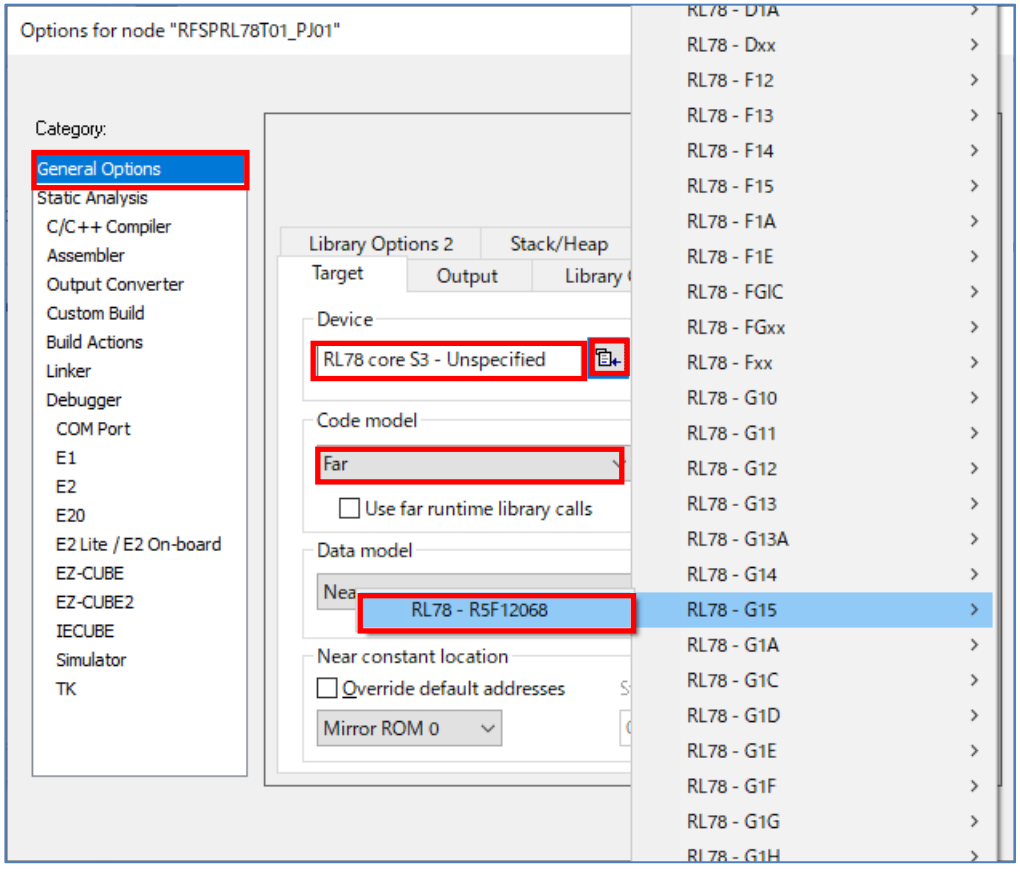


(2) Selection of a target device

On IAR Embedded Workbench, click the right mouse button for the project ("RFSPRL78T01_PJ01 - Debug") in a tree. When an "option" is selected, the "Options for node [Project name]" window is displayed.



- Input setting in the [General Option] - [Target] tab of "Option for node [Project name]" window.
- Press "" "button of [Device]. And select "RL78 – G15" - "RL78 – R5F12068". Select "Far" as [code model] and select "Near" as [data model].

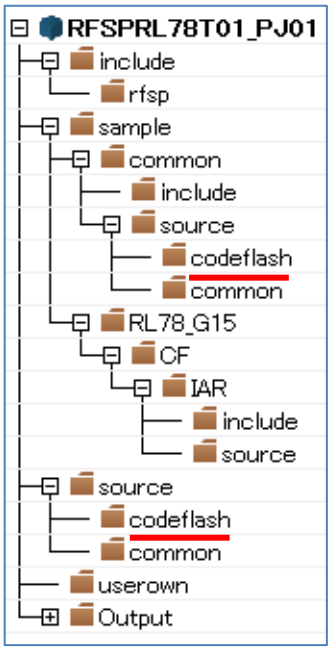


6.2.2 **Example of Registration of Target Folders and Target Files**

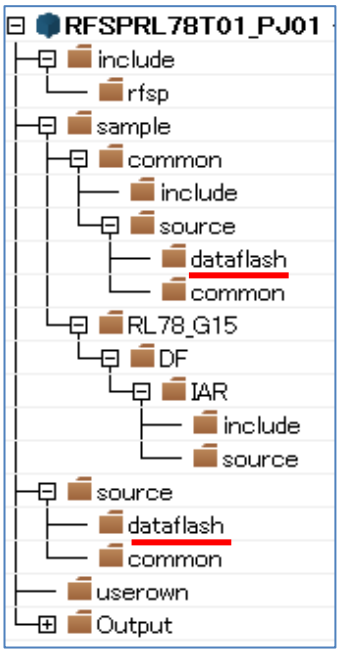
Using RFSP Type 01, when programming each area [(1) code flash memory, (2) data flash memory], the example which registers necessary files is shown. Each folder of a RFSP Type 01 source-program file is "include", "source", "userown", and "sample". The target file in each folder is selected and registered by the area programmed.

Instead of registering a folder by IAR Embedded Workbench, select [Add Group] of the [Project] menu, and add a group. The example into which I add the group of the same structure as the folder for RFSP Type 01, and files are registered is shown. (Registering without making a group is also possible.)

The example which added the group of each area [(1)Code flash memory, (2)Data flash memory] is shown. (The group name which changes with areas is shown by "—".)



(1)Code flash memory



(2)Data flash memory

・Exclusion of the file automatically added by the function of IDE.

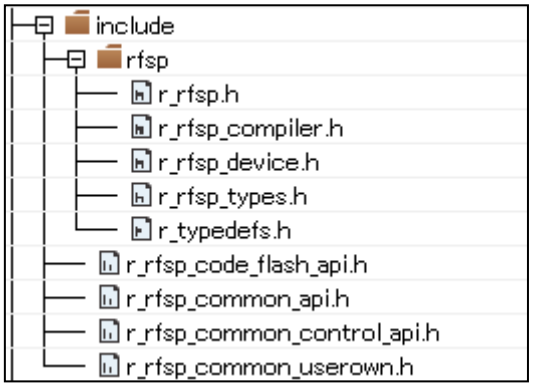
There are files added automatically in the created project. The same file as these exists also in the "sample" folder of RFSP Type 01. Therefore, using the function of IDE, Select those files from tree and excludes from a project.

- IAR Embedded Workbench : Clicks the right mouse button for the file of tree. And exclude the target "main.c" file by "Remove" function.

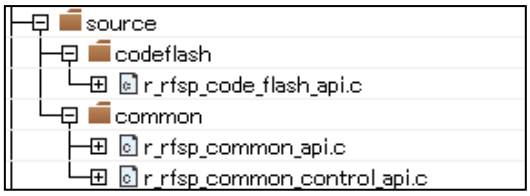
(1) Registration of the groups and files of the target in the case of reprogramming code flash memory

The groups ("include", "source", "userown", "sample") and source program file which are included in RFSP Type 01 to register are shown below.

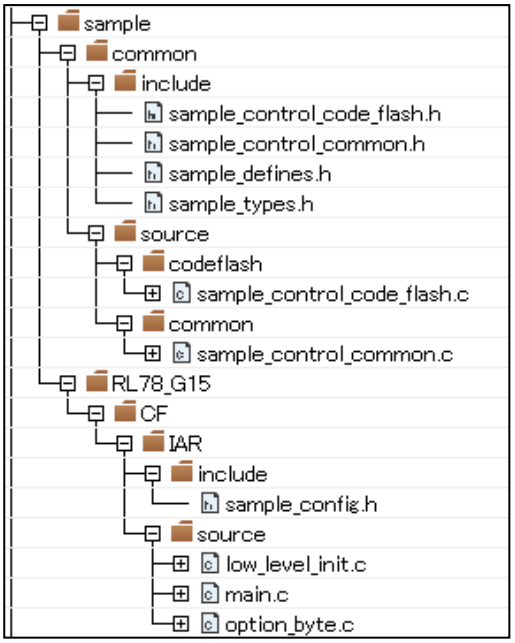
in the “include” group



in the “source” group



in the “sample” group

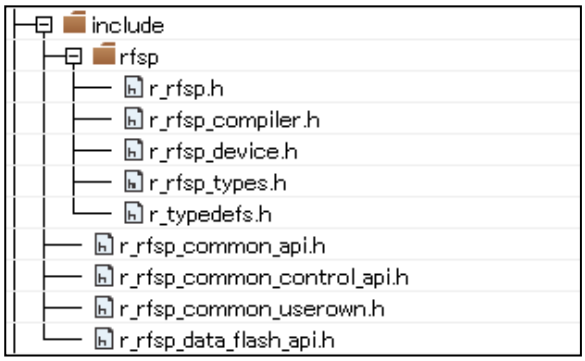


in the “userown” group

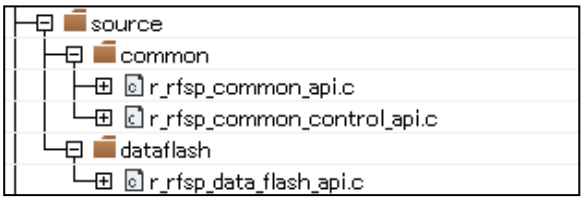
(2) Registration of the groups and files of the target in the case of reprogramming data flash memory

The groups ("include", "source", "userown", "sample") and source program file which are included in RFSP Type 01 to register are shown below.

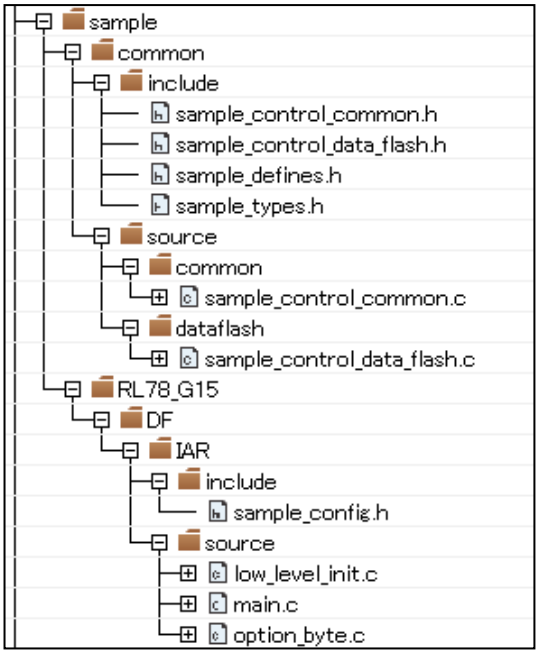
in the “include” group



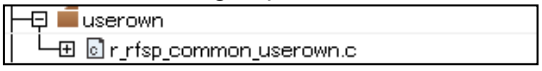
in the “source” group



in the “sample” group



in the “userown” group

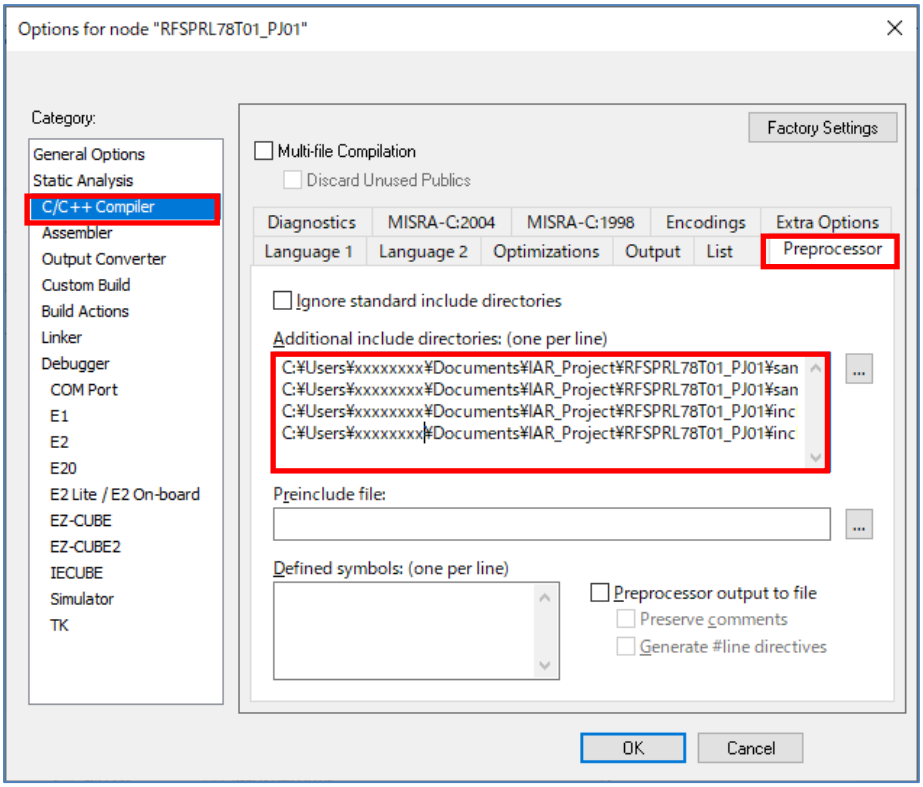


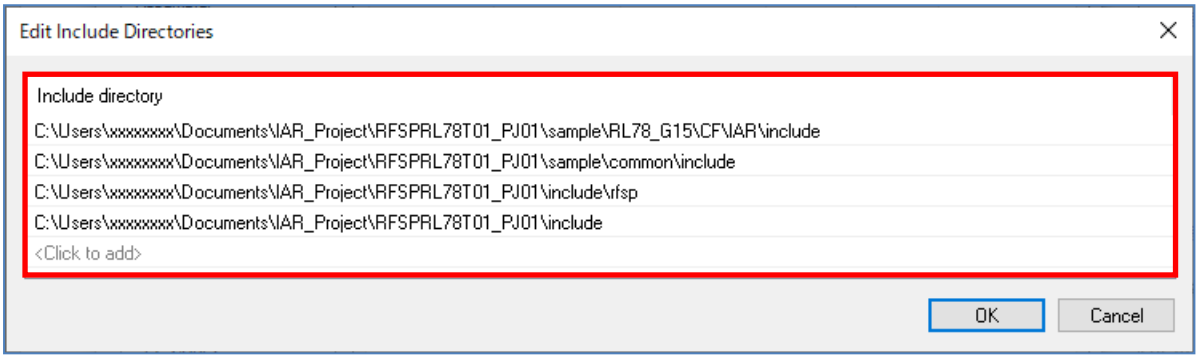
6.2.3 **Integrated Development Environment(IDE) Settings**

Set IDE setting necessary in order to build RFSP Type 01 using IAR compiler.
IAR Embedded Workbench : Click the right mouse button for the project(”RFSPRL78T01_PJ01“) in a tree, and select "Options". And set each setting of the “Category” in the displayed window.

6.2.3.1 **Include Path Settings**

・Setting of the include path on IAR Embedded Workbench selects "C/C++ Compiler" of "Category", and inputs path in "Preprocessor" tab. (Change by a target area)
- Input the Include directory path in the ”Edit include Directories” window displayed by selection of [Additional include directories: (one per line)].





- The example of folder path settings
  It is the example which placed each folder(”include”, ”source”, ”userown”, ”sample”) of the source program file of RFSP Type 01 on “C:\Users\xxxxxxxx\Documents\IAR_Project\".

  (1) Code flash memory reprogramming

  C:\Users\xxxxxxxx\Documents\IAR_Project\RFSPRL78T01_PJ01\sample\RL78_G15\CF\IAR\include

  C:\Users\xxxxxxxx\Documents\IAR_Project\RFSPRL78T01_PJ01\sample\common\include

  C:\Users\xxxxxxxx\Documents\IAR_Project\RFSPRL78T01_PJ01\include

  C:\Users\xxxxxxxx\Documents\IAR_Project\RFSPRL78T01_PJ01\include\rfsp

  (2) Data flash memory reprogramming

  C:\Users\xxxxxxxx\Documents\IAR_Project\RFSPRL78T01_PJ01\sample\RL78_G15\DF\IAR\include

  C:\Users\xxxxxxxx\Documents\IAR_Project\RFSPRL78T01_PJ01\sample\common\include

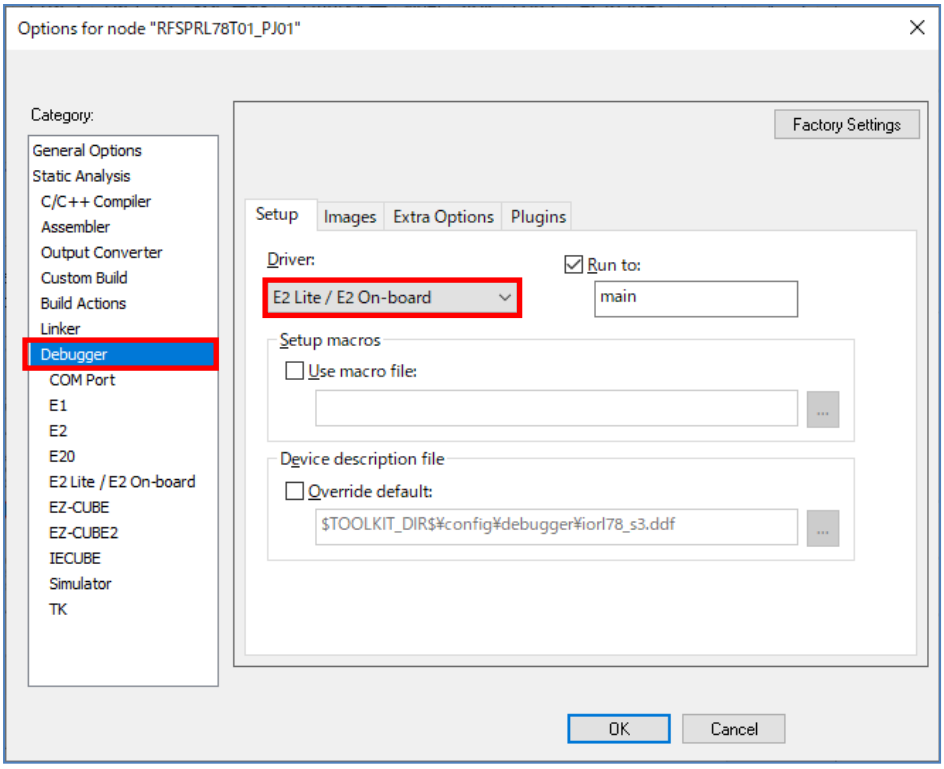
  C:\Users\xxxxxxxx\Documents\IAR_Project\RFSPRL78T01_PJ01\include

  C:\Users\xxxxxxxx\Documents\IAR_Project\RFSPRL78T01_PJ01\include\rfsp

Note : About the path setting of include directories.
When the project is copied in the case appointed by the absolute path, the setup is needed again. It is possible to appoint a relative path ($PROJ_DIR$) so that it can be used, even if it copies the project.
Refer to each reference manual of IAR Embedded Workbench about how to appoint the relative path.

6.2.3.2 Debugger settings

・Select ”E2 Lite/E2 On-Board” from [Driver] of [Debugger] – [Setup] tab on the assumption that on-chip debugging is implemented.



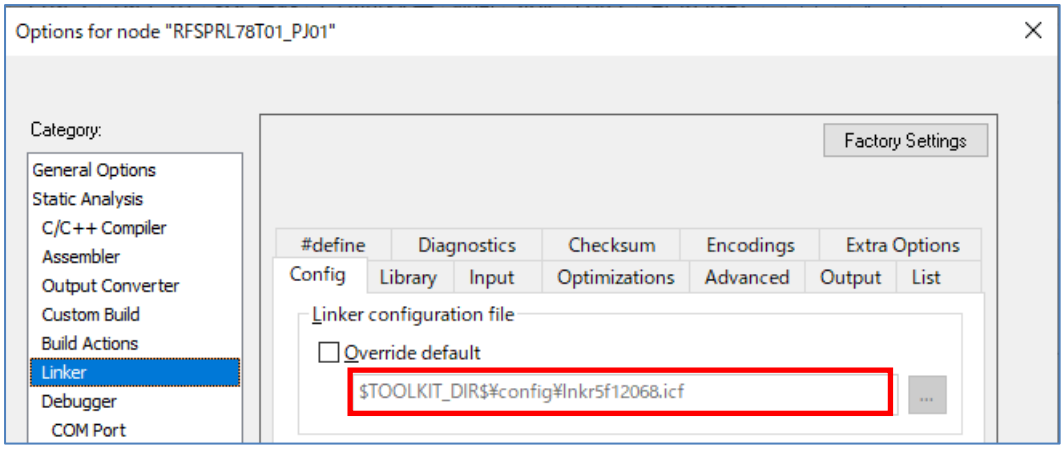
Note : Refer to each reference manual of IAR Embedded Workbench about the other items to be set.

6.2.4 **Linker Configuration File(.icf) Settings**

RFSP Type01 uses a link configuration file (*.icf) for the target device when building with IAR Embedded Workbench.

To specify the link configuration file (*.icf) for the target device, select "Options" by right-clicking the mouse of [Project] in the tree, and specify "$TOOLKIT_DIR$\config\lnkrxxxxxxx.icf" set in [Linker] - [Config] in the displayed screen.

Example：R5F12068 (RL78/G15) specifies the "lnkr5f12068.icf" file.



Note) For details on what is described in the linker configuration file and how to write it, please refer to each reference manual from [Help] in IAR Embedded Workbench.

6.2.4.1 Option Bytes Settings

The option bytes definition for RL78 is described in the linker configuration file (*. icf) attached to IAR Embedded Workbench. (Ex.R5F12068 [RL78/G15]:“lnkr5f12068.icf”) The option byte value in RFSP Type01 is described in the "option_byte.c" file.

Note : Refer to each reference manual of IAR Embedded Workbench about the option bytes setting method for Linker configuration file.

The example of defining the option bytes for a linker configuration file (*.icf) attached to IAR Embedded Workbench.

```
define block OPT_BYTE with size = 4  { R_OPT_BYTE,
                                ro section .option_byte,
                                ro section OPTBYTE };
                                 |
```

The example of description of the option bytes value in "option_byte.c" file.

```
#pragma location = "OPTBYTE"
__root const unsigned char option_bytes[4] = {
    0xEE,   /* 11101110 */
            /*     |||| */
            /*     |||+-- Watchdog timer     */
            /*     ||||    operation stopped  */
            /*     ||||    in HALT/STOP mode  */
            /*     +++--- Watchdog timer     */
            /*     |       overflow time is   */
            /*     |       2^16 / fIL =       */
            /*     |       3799 ms         */
            /*     +------ Watchdog timer     */
            /*             operation disabled */

    0xFF,   /* 11111111 */
            /*     ||   */
            /*     |++-- SPOR 2.16V/2.11V */
            /*     +---- P125 input reset */
    0xF9,   /* HS mode 16 MHz */
    0x85    /* OCD: enables on-chip debugging function */
};
```

- Description of user option byte value:
  The value of User option byte (000C0H-000C2H) in "option_byte.c" file is "0xEEFFF9".
  (WDT stop, P125:RESET input, SPOR detection voltage:2.16V/2.11V, 16MHz [The example for RL78/G15,RL78/G16])

  The value of on-chip debug option byte(000C3H) in "option_byte.c" file is ”0x85”.
  (The example of enable on-chip debug operation)

Note : Be sure to confirm the contents of "User option byte" of the chapter of "Option Bytes", and "On-chip debug option byte" by the user's manual of a target device. And describe the set value used with user application.

6.2.5 On-chip Debug Settings

After executing building of a target project, connect E2 Lite, select [Download and Debug] from [Project] menu, and start debugging.

6.2.5.1 Example of How to deal with Connection Errors

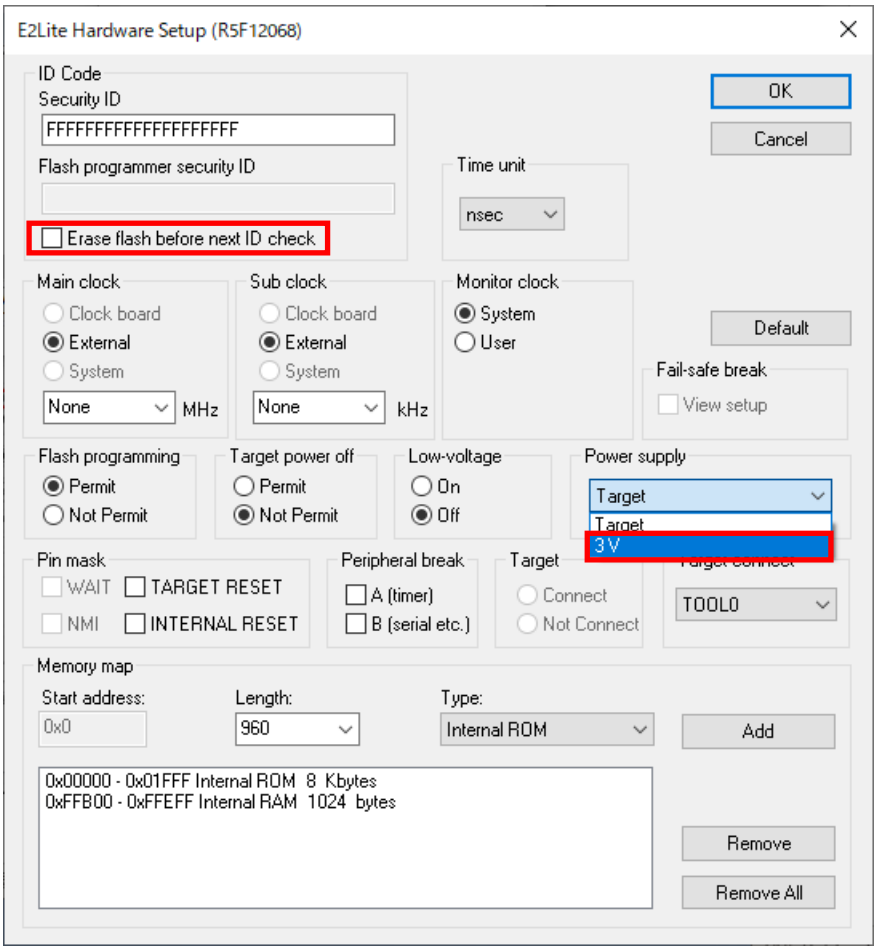
Explain the common examples of how to deal with an error which happened by connection in on-chip run debug. This is the case when an ID code mismatch or power failure occurs.

Note : In cases where a target cannot be connected by other causes, please confirm each reference manual from [Help] of IAR Embedded Workbench.

When selecting [Download and Debug] and starting debugging, an "E2 Lite Hardware setting" screen may be displayed. The cause may be ID code mismatch or power setting error.

- In the case of the ID code mismatch:
  "Cannot verify the ID code." etc. may be displayed as a message. In this case, put a check mark to "Erase flash before next ID check" of the [ID code] in an "E2 Lite Hardware Setup" window, and continue. And the flash memory is erased and debugger may be connected.
- In the case of power setting error:
  Initial setting of "Power supply" is "Target". When supplying power supply from E2 Lite, select "3V" by the pull-down menu for "Power supply".

Caution: Be sure not to set "3V"(supply power from E2 Lite) , when the power is supplied to the target.



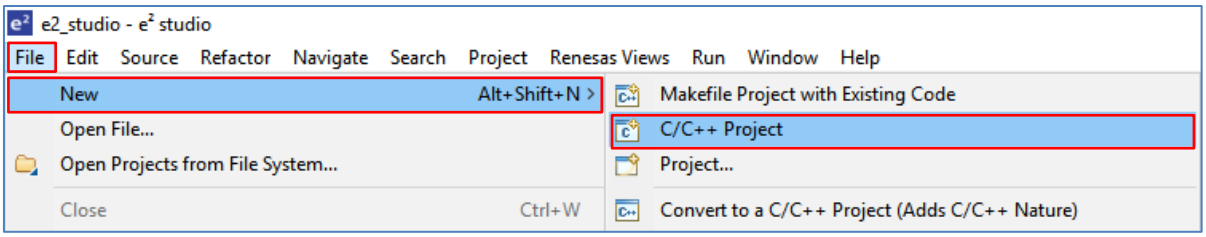
6.3 Creating a Project in the case of Using LLVM Compiler

e² studio can be used for LLVM for RL78 compiler as an IDE. RFSP Type 01 is registered and built in the project created by the IDE. An example of creating a sample project in case each IDE is used is shown. Because to understand LLVM compiler and each IDE, it is necessary to refer to the user's manual of each tool product.

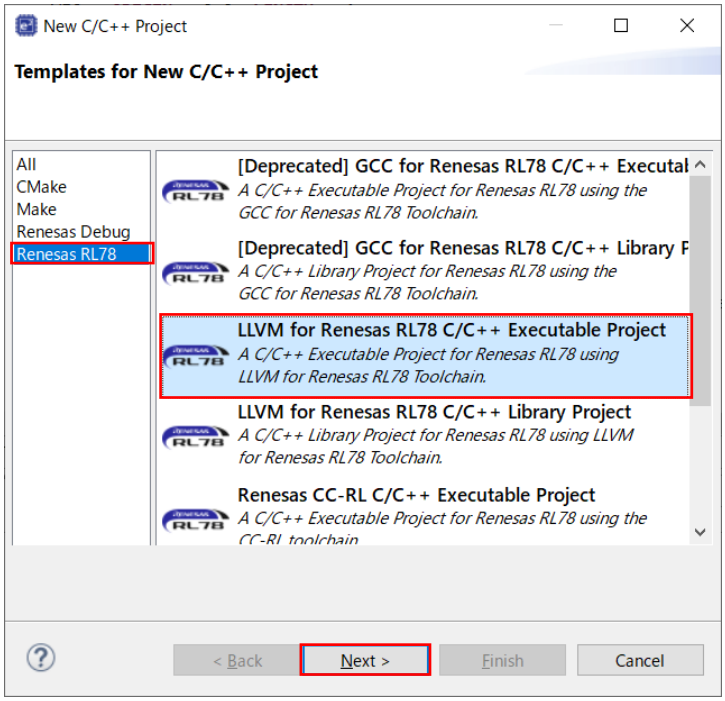
6.3.1 Example of Creating a Sample Project

(1) An example of creating a sample project which used e² studio (IDE)

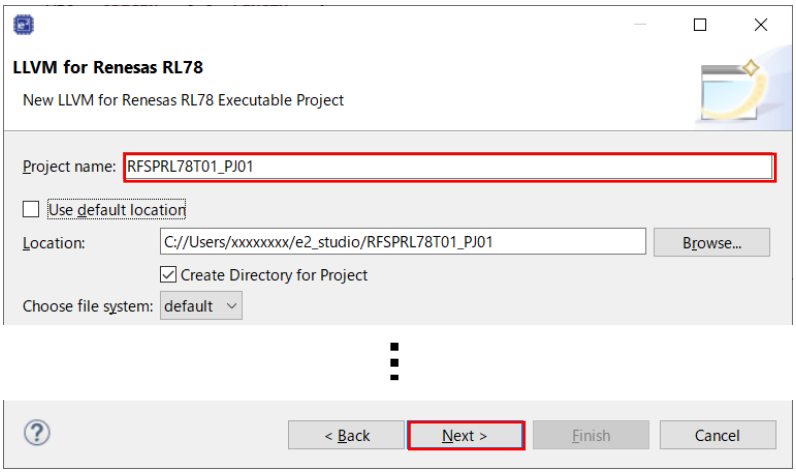
- The e² studio starts and from the [File] menu, select [New] – [C/C++ Project], the “Templates for New C/C++ Project” window will open.



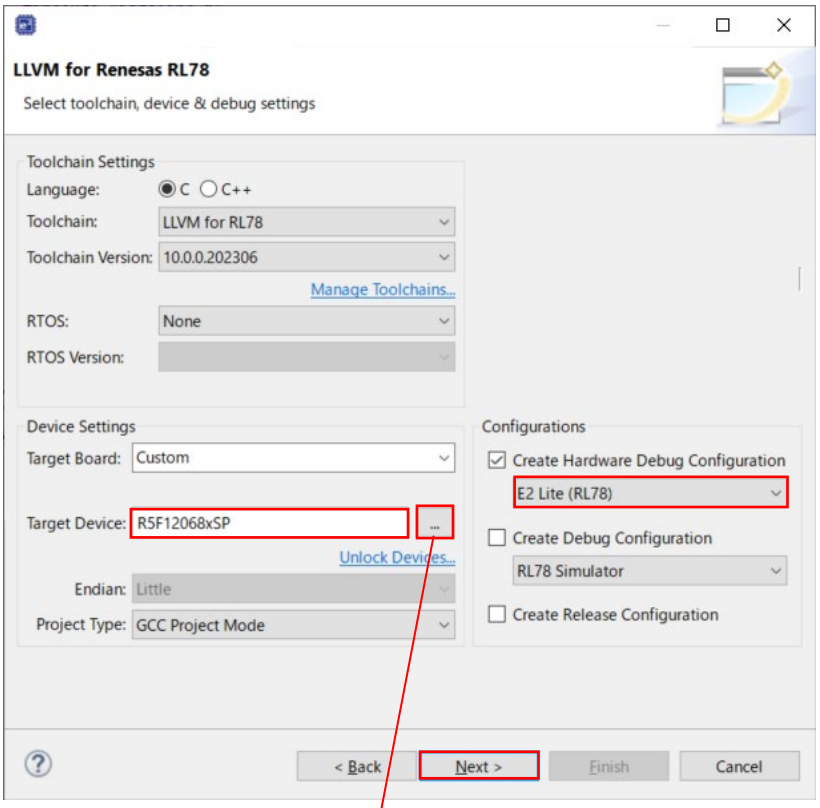
- Select [LLVM for Renesas RL78 C Executable Project] displayed after selection in [Renesas RL78], and press "Next" button.

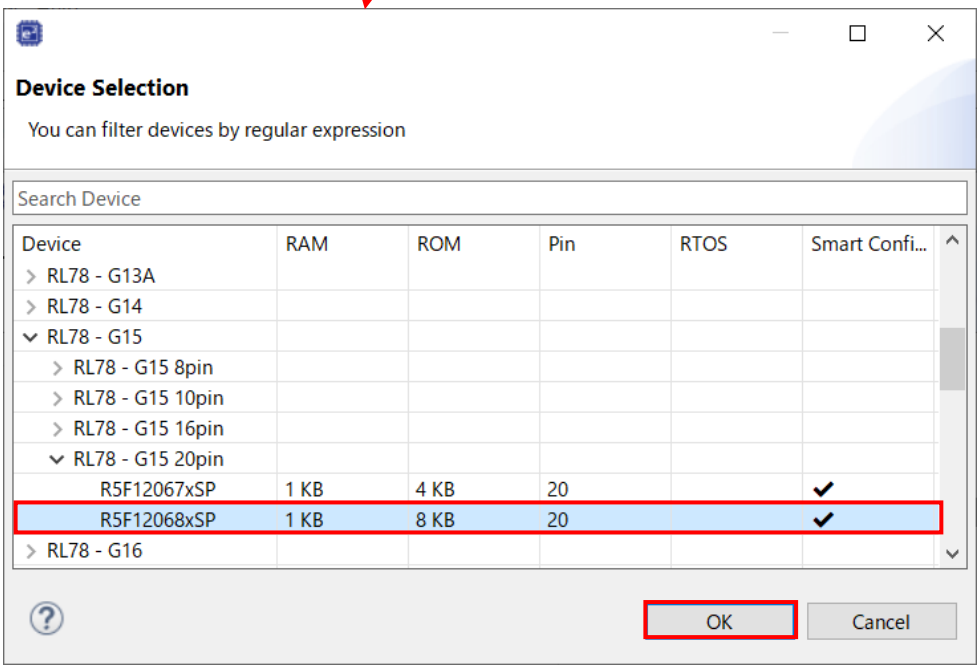


- Input "project name" on "New LLVM for Renesas RL78 Executable Project" window, and press "next" button. [Project name] is temporarily set to "RFSPRL78T01_PJ01".



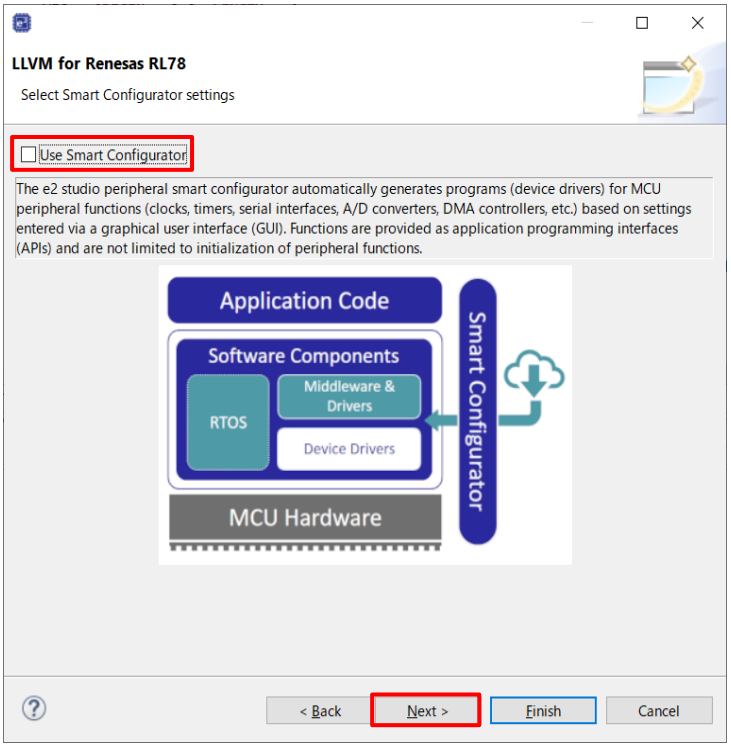
- Select the [Target Device] of [Device Settings], and select "RL78 - G15 20pin - R5F12068xSP".
- It is a premise that E2 Lite is selected as a debugging tool and on-chip debugging is executed. Put a check mark to "Create Hardware Debug Configuration" by [Configurations]. And select "E2 Lite(RL78)".
- Press "Next" button.





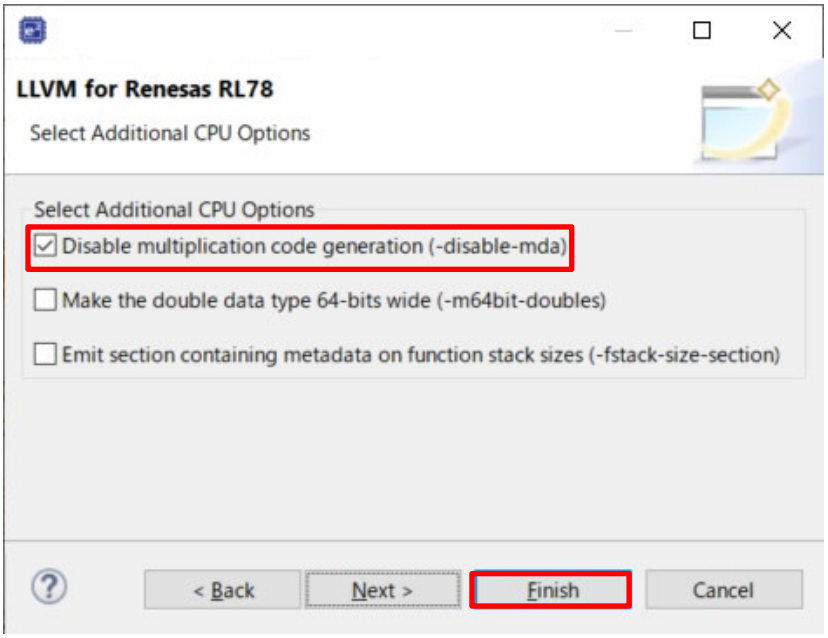
・Put off a check mark from "Use Smart Configurator".

・Press "Next" button.



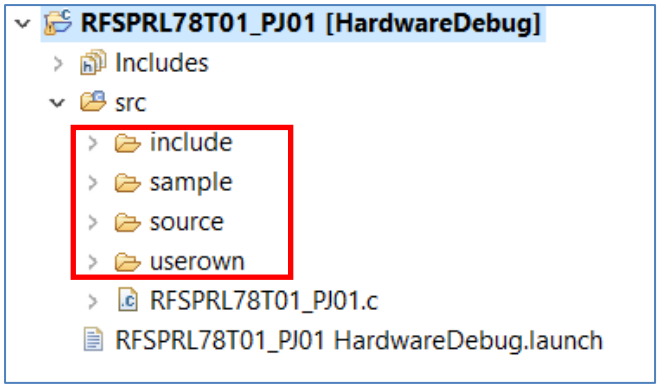
・Select "Disable multiplication code generation (-disable-mda)".

・Press "Finish" button.



6.3.2 Example of Registration of Target Folders and the Target Files

Using RFSP Type 01, when programming each area [(1) code flash memory, (2) data flash memory], the example which registers necessary files is shown. Each folder of RFSP Type 01 source-program file is "include", "source", "userown", and "sample". The target file in each folder is selected and registered by the area programmed.
As other registration methods, after all the folders of "include", "source", "userown", and "sample" are registered, unnecessary files and folders can be removed using the function of [Resource Configuration] – [Exclude from Build].



The registration tree screen of RFSP (e^2 studio)

- Registration of the latest I/O header files outputted to target products by e2 studio
  "iodefine.h" and "iodefine_ext.h" are I/O header files which e^2 studio outputs to target products. Replacing instead of "iodefine.h" and "iodefine_ext.h" included in RFSP Type 01 is recommended. Registration of target folders and target files are implemented. Then, a user replaces "iodefine.h" and "iodefine_ext.h" which e2 studio outputted with "iodefine.h" and "iodefine_ext.h" included in RFSP Type 01.

- Registration of the vector table file outputted to target products by e2 studio
  “interrupt_handlers.h”, “inthandler.c” and “vects.c” are files that contain vector tables that e2 studio outputs for the target product. Since it depends on the product, please replace “interrupt_handlers.h”, “inthandler.c”, and “vects.c” included in RFSP Type 01.
  When these are replaced, change the option byte values in the “vects.c” file. Refer to “6.3.3.2 Device Item Settings” for details on setting option byte values.

  The folder to which “iodefine.h, iodefine_ext.h, “interrupt_handlers.h”, “inthandler.c” and “vects.c” files are outputted by e2 studio :
  - [Project name]/generate Folder

  The folder with which a user replaces “iodefine.h”, “iodefine_ext.h” and “interrupt_handlers.h” files:
  - The case of code flash programming : ”\[Project name]\sample\RL78_G15\CF\LLVM\include”
  - The case of data flash programming : ”\[Project name]\sample\RL78_G15\DF\LLVM\include”

  The folder with which a user replaces the “inthandler.c” and “vects.c” files:
  - The case of code flash programming : “\[Project name]\sample\RL78_G15\CF\LLVM\source”
  - The case of data flash programming : “\[Project name]\sample\RL78_G15\DF\LLVM\source”

・Exclusion of the file automatically added by the function of e^2 studio.

There are files added automatically in the created project. The same file as these exists also in the "sample" folder of RFSP Type 01. Therefore, using the function of e^2 studio, select those files from tree and excludes from a project.

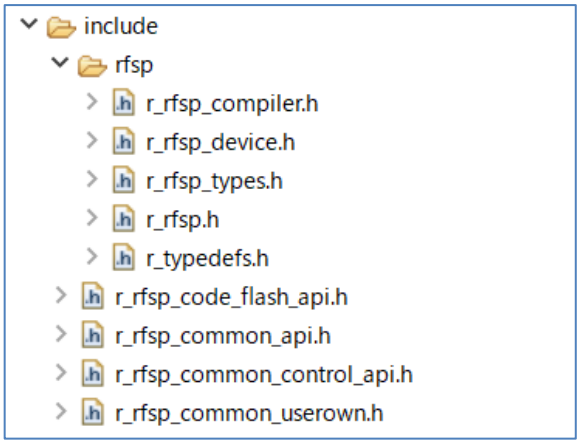
Clicks the right mouse button for the file of tree. And, on the [Settings] screen displayed by the "property", put a check mark to [Exclude resource from build] and exclude a target file (target folder). (Exclusion of a folder is also possible)

- Exclude(or delete) all the files in a [project name] / generate folder:
  hwinit.c, interrupt_handlers.h, inthandler.c, iodefine_ext.h, iodefine.h, linker_script.ld, start.S, typedefine.h, vects.c
- Exclude(or delete) a [project name].c files in a [project name] / src folder:
  RFSPRL78T01_PJ01.c is a target.

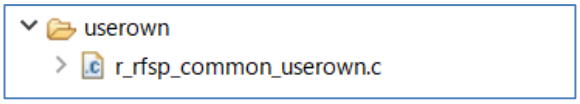
(1) Registration of the folders and files of the target in the case of reprogramming code flash memory

The folders ("include", "source", "userown", "sample") and source program file which are included in RFSP Type 01 to register are shown below.

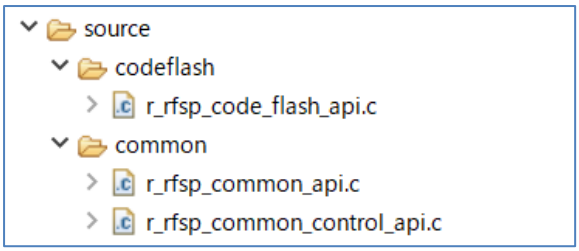
in the “include” folder



in the “userown” folder



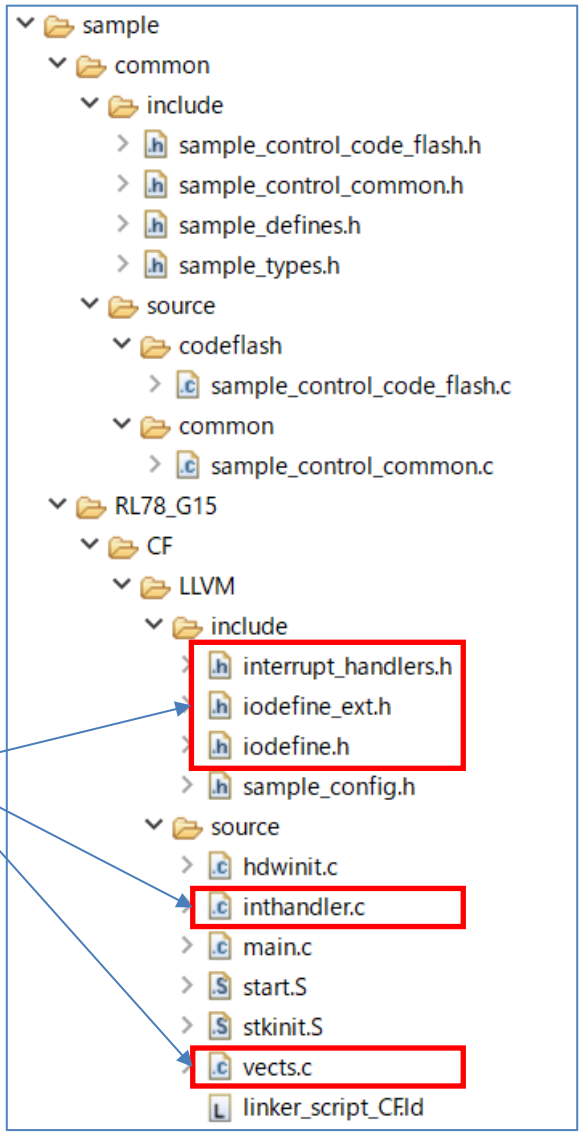
In the “source” folder



Transpose to “iodefine.h”, “iodefine_ext.h”, “interrupt_handlers.h”, “inthandler.c” and “vects.c” outputted by e^2 studio.

* “vects.c” should change the option byte value.

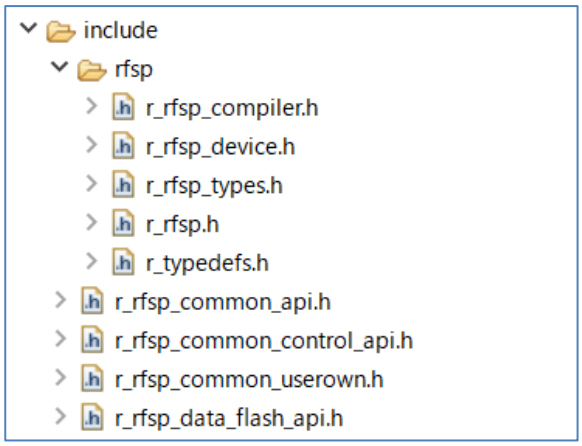
in the “sample” folder



(2) Registration of the folders and files of the target in the case of reprogramming data flash memory

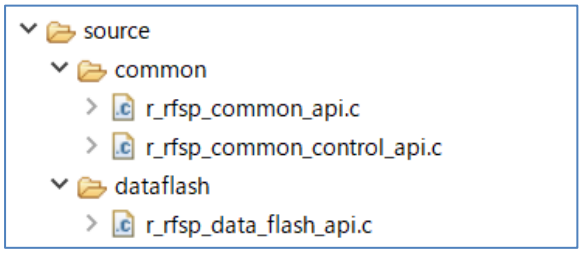
The folders ("include", "source", "userown", "sample") and source program file which are included in RFSP Type 01 to register are shown below.

in the “include” folder

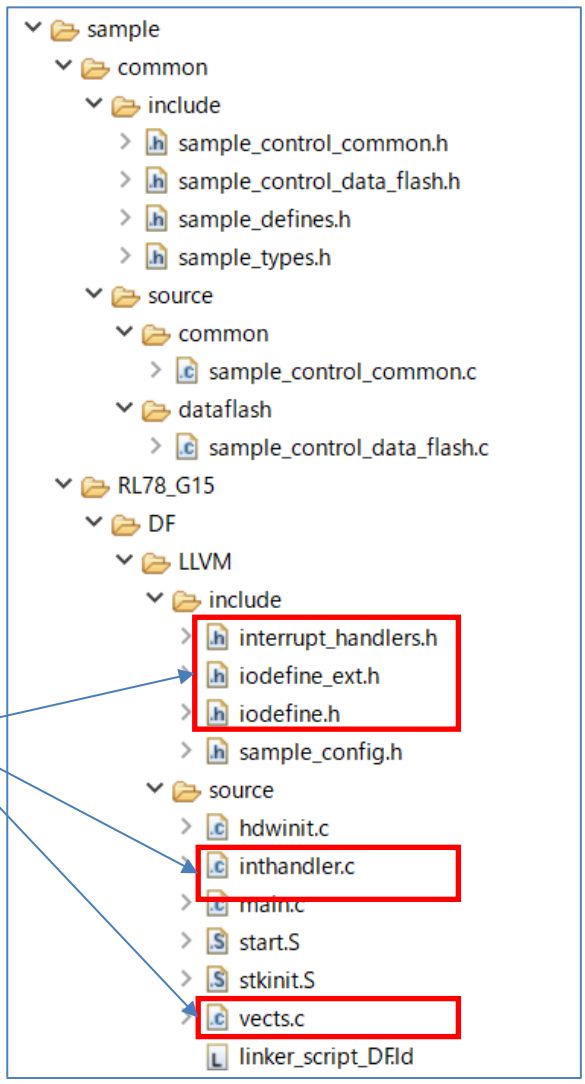


in the “userown” folder

In the “source” folder



In the “sample” folder



Transpose to “iodefine.h”, “iodefine_ext.h”, “interrupt_handlers.h”, “inthandler.c” and “vects.c” outputted by e^2 studio.

* “vects.c” should change the option byte value.

6.3.3 **Build Tool Settings**

Set IDE setting necessary to build RFSP Type 01 using LLVM compiler.

e^2 studio : Click the right mouse button for the project(”RFSPRL78T01_PJ01“) in a tree, and select "Property". And set each setting of the build tool in the displayed window.

6.3.3.1 **Include Path Settings**

・Setting of the include path on e^2 studio inputs path in “Properties” window. (Change by a target area)

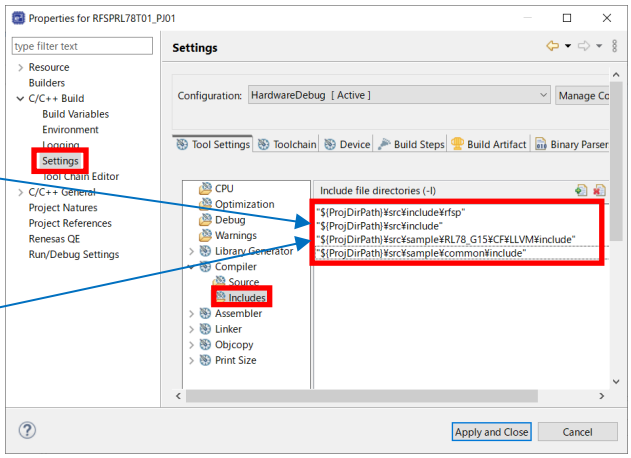
- Input the Include directory path in the window displayed by selection of "C/C++" build [Setting] - "Compiler" [Includes].

(1) Code flash memory reprogramming

```
${ProjDirPath}\src\include\rfsp
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_G15\CF\LLVM\include
${ProjDirPath}\src\sample\common\include
```

(2) Data flash memory reprogramming

```
${ProjDirPath}\src\include\rfsp
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_G15\DF\LLVM\include
${ProjDirPath}\src\sample\common\include
```



6.3.3.2 Device Item Settings

The option byte value in RFSP Type01 is described in the "vects.c" file.

The example of description of the option bytes value in "vects.c" file.

```
const unsigned char Option_Bytes[]  __attribute__ ((section (".option_bytes"))) = {
    0xee, 0xff, 0xf9, 0x85
};
```

- Description of user option byte value:
The value of User option byte (000C0H-000C2H) in "vects.c" file is "0xEEFFF9".
(WDT stop, P125:RESET input, SPOR detection voltage:2.16V/2.11V, 16MHz [The example for RL78/G15,RL78/G16])

The value of on-chip debug option byte(000C3H) in "vects.c" file is "0x85".
(The example of enable on-chip debug operation)

Note : Be sure to confirm the contents of "User option byte" of the chapter of "Option Bytes", and "On-chip debug option byte" by the user's manual of a target device. And describe the set value used with user application.

6.3.3.3 Linker Script Path Settings

Setting of the linker script path on e^2 studio inputs path in "Properties" window. (Change by a target area)

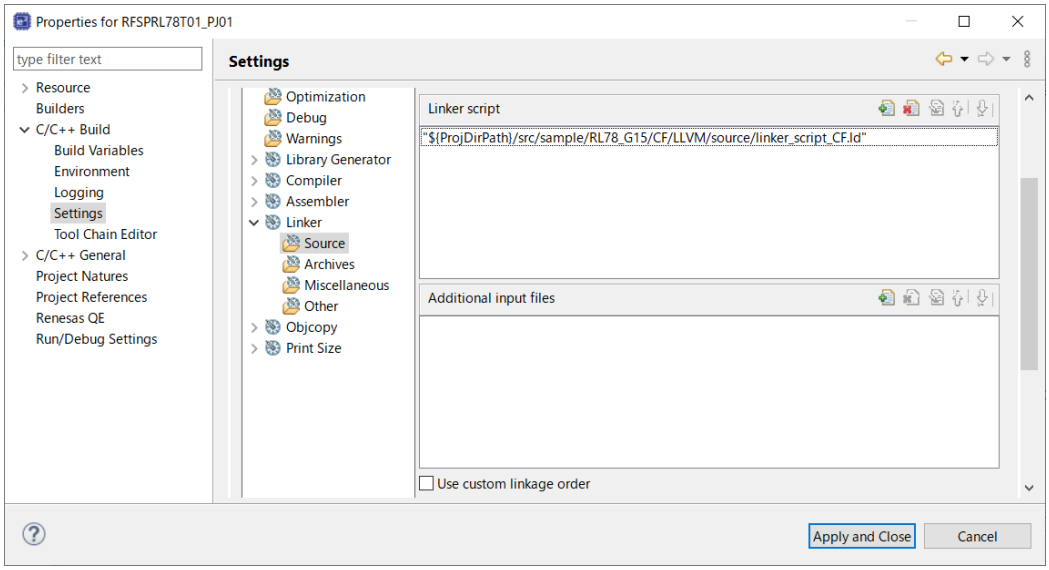
- Input the linker script path in the window displayed by selection of "C/C++" build [Setting] - "Linker" [Source].

(1) Code flash memory reprogramming

${ProjDirPath}/src/sample/RL78_G15/CF/LLVM/source/linker_script_CF.ld

(2) Data flash memory reprogramming

${ProjDirPath}/src/sample/RL78_G15/DF/LLVM/source/linker_script_DF.ld



6.3.4 Debug Tool Settings

This section describes the contents of connection setting on a target board necessary to execute on-chip debugging. As a debugging tool, it is a premise that E2 Lite is selected. Refer to the user's manual for each IDE for the details of other debugging tool setting.

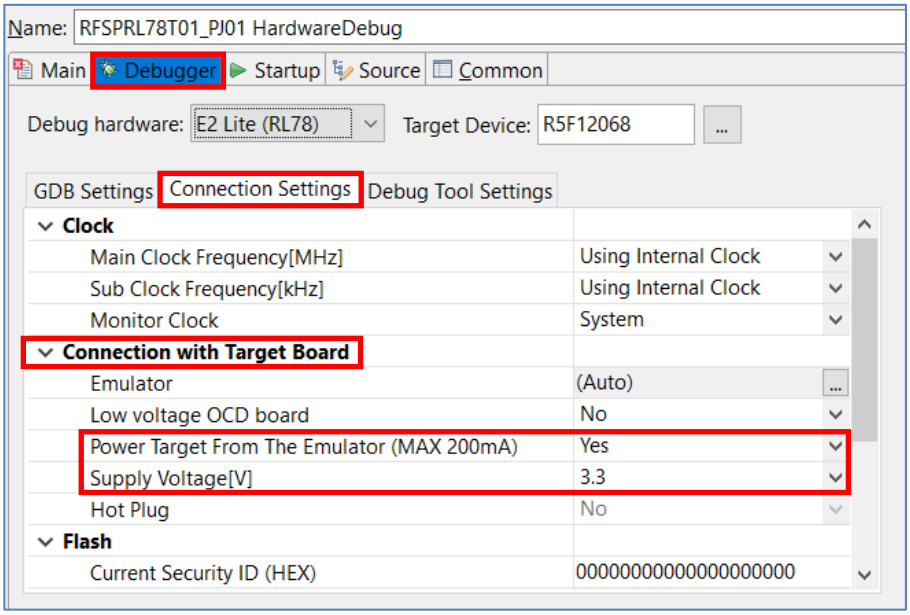
On e^2 studio, right-click a mouse in the target project of a tree. Selection of [Debug As] - [Debug Configurations…] will display the "Debug Configurations" screen. On the tree of a screen, select the target project ("RFSPRL78T01_PJ01 HardwareDebug") of [Renesas GDB Hardware Debugging]. And the displayed "Debugger" tab performs debugging tool setting.

Note: The power is already supplied to the target board, or when power supply capacity is insufficient, the emulator including E2 Lite may be unable to supply power to a target board. Be sure to refer to "the user's manual and Additional Document for User's Manual (Notes on Connection of RL78)" for the emulator for target devices, and use an emulator.

6.3.4.1 Setting of Connection with Target Board

- On e^2 studio, set up the connection with target board(via E2 Lite) with "Connect Settings" tab. (Common in each area)

  - [Connection with Target Board] item
    To let power supply(Supply Voltage : 3.3V) from E2 Lite to a target board, it is necessary to set "Yes" to [Power Target From The Emulator (MAX 200mA)].



7 Revision History

7.1 Major Modifications in this Revision

Rev.	Date	Description	
		Page	summary
1.00	May.30.22	-	Newly created.
1.10	Apr.28.23	-	RL78/G16 was added.
1.20	Sep.25.23	-	LLVM compiler was added.

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

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