User's Manual

RL78 Family

Renesas Flash Driver RL78 Type 11

User's Manual

RENESAS Microcontrollers

RL78 / L23

Renesas Electronics
www.renesas.com

Rev.1.01 Nov 2025

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.

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.

How to Use This Manual

- Readers

  This manual is intended for engineers who wish to develop application systems using the RL78/L23 microcontrollers.

- Purpose

  This manual is intended to give users an understanding of the methods for using the Renesas Flash Driver (RFD) RL78 Type 11 to reprogram the flash memory in the RL78/L23 microcontroller.

- Organization

  This manual is separated into the following sections.

  1. Overview
  2. System Configuration
  3. API Functions of RFD RL78 Type 11
  4. Flash Memory Sequencer Operation
  5. Sample Programs
  6. Creating a Sample Project for RFD RL78 Type 11

- How to Read this Manual

  It is assumed that the readers of this manual have general knowledge in the fields of electrical engineering, logic circuits, microcontrollers, C language, and assemblers.

  To understand the hardware functions of the RL78/L23:

  - Refer to the User's Manual of the target RL78/L23 devices.

- Conventions
  - Data significance: Higher digits on the left and lower digits on the right
  - Active low representations: $\overline{\times\times\times}$ (overscore over pin and signal name)
  - Note: Footnote for item marked with Note in the text
  - Caution: Information requiring particular attention
  - Remark: Supplementary information
  - Numeric representation:

    Binary: ×××× or ××××B

    Decimal: ××××

    Hexadecimal: ××××H or 0x××××
  - Prefixes indicating power of 2 (address space and memory capacity):

    K (kilo) 2^{10} = 1024

    M (mega) 2^{20} = 1024^2

- Related Documents

  The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

No	Document Title	Document Number
1	RL78/L23 User's Manual: Hardware	R01UH1082EJ
2	E1/E20/E2 Emulator, E2 Emulator Lite Additional Document for User's Manual (Notes on Connection of RL78)	R20UT1994EJ
3	Renesas Flash Driver and EEPROM Emulation Software Target MCU List for RL78	R20UT5631EJ

Table of Contents

Abbreviations

Abbreviation	Description
RFD	Renesas flash driver
API	Application program interface
BGO	Background operation Instructions in the code flash memory can be executed during reprogramming of the data flash memory.
FSW	Flash shield window This is a function for disabling programming and erasure of the specified window range or the flash areas outside the specified window range during self-programming.
RAM	Random access memory Randomly accessible volatile memory. It is memory for holding values that are to be changed during program execution.
ROM	Read-only memory Non-volatile memory. It is memory whose contents cannot be changed. The code flash memory may be called ROM.

Terminology

Terminology	Description
Code flash memory	Flash memory for storing application code and constant data. Note that this memory may be abbreviated as “CF” in this document.
Data flash memory	Flash memory for storing data. Note that this memory may be abbreviated as “DF” in this document.
Extra area	Generic name of the configuration setting area, security setting area, block protection area, and boot swapping setting area.
Flash memory sequencer	The RL78 microcontroller has a dedicated circuit for controlling the flash memory. This circuit is called the flash memory sequencer in this document. The flash memory sequencer consists of the code/data flash area sequencer, which reprograms the code flash area or data flash area, and the extra area sequencer, which reprograms the extra area.
Flash memory control mode	The flash memory sequencer has the following modes, which indicate the programming enabled or disabled state. - Code flash memory programming mode - Data flash memory programming mode - Non-programmable mode
Code flash memory programming mode	The code flash memory (and extra area) can be reprogrammed in this mode.
Data flash memory programming mode	The data flash memory can be reprogrammed in this mode.
Non-programmable mode	The flash memory (and extra area) cannot be reprogrammed in this mode.
Self-programming	A method of reprogramming the flash memory by executing a user program instead of using an external flash memory programming tool.
Serial programming	A method of reprogramming the flash memory using an external flash memory programming tool.
Boot area	A boot area is an area with the boot cluster size which begins from the address 00000H.
Boot clusters 0 and 1	A boot cluster is a block group for the boot swapping and either boot cluster 0 or 1 is allocated to the boot area. The flash memory programmer can set boot cluster size. Refer to the user's manual for the target device for the size setup of a boot cluster. Physical area name: [Example: Boot cluster size = 16KB, BTFLG = 1] Boot cluster 0: 00000H to 03FFFH Boot cluster 1: 04000H to 07FFFH
Boot swapping	Boot clusters 0 and 1 are swapped.
Bank programming function	The user area is divided into two bank areas and the user program in one bank area(rewrite bank) can be updated while execution of the program in the other bank area(startup bank) is in progress.
Bank programming mode	Bank programming mode indicates the mode which reprograms the flash memory for executing a bank programming function.
Startup bank	Startup bank is one of the user areas where the code flash memory was divided into two bank areas on “bank programming.” It is a first half bank area of the side which executes a user program.
Rewrite bank	Startup bank is one of the user areas where the code flash memory was divided into two bank areas on “bank programming.” It is a second half bank area of the side which a user program reprograms.

Bank swapping function	Swap the “Startup bank” and “Rewrite bank.”

1. Overview

1.1 Outline

Renesas Flash Driver RL78 Type 11 (hereafter called RFD RL78 Type 11) is software for reprogramming the flash memory in the RL78/L23.

1.1.1 Purpose

The purpose of this document is to give the information about RFD RL78 Type 11.

1.2 Contents

The API functions of RFD RL78 Type 11 are called from the user program to reprogram the code flash memory or data flash memory.

The RFD RL78 Type 11 package includes the following.

- This user's manual
- Source code files of RFD RL78 Type 11 for controlling the data flash memory and code flash memory incorporated in the RL78/L23.
- Sample programs for erasing and reprogramming the data flash memory, code flash memory, and extra area.

1.3 Features

RFD RL78 Type 11 reprograms the flash memory according to the specified flow of command processing for the flash memory control circuit. Each API function of RFD RL78 Type 11 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. Such a configuration is adopted so as to flexibly handle processing dependent on the user application, such as, timeout processing in which the timeout value varies with the conditions of user application program execution.

Figure 1-1 shows the flash memory control by the user application using the API functions of RFD RL78 Type 11.

RFD RL78 Type 11 provides sample programs of the processing that is implemented by combinations of two or more API functions and user programs. Refer to the sample programs when embedding the flash memory control processing in the user application.

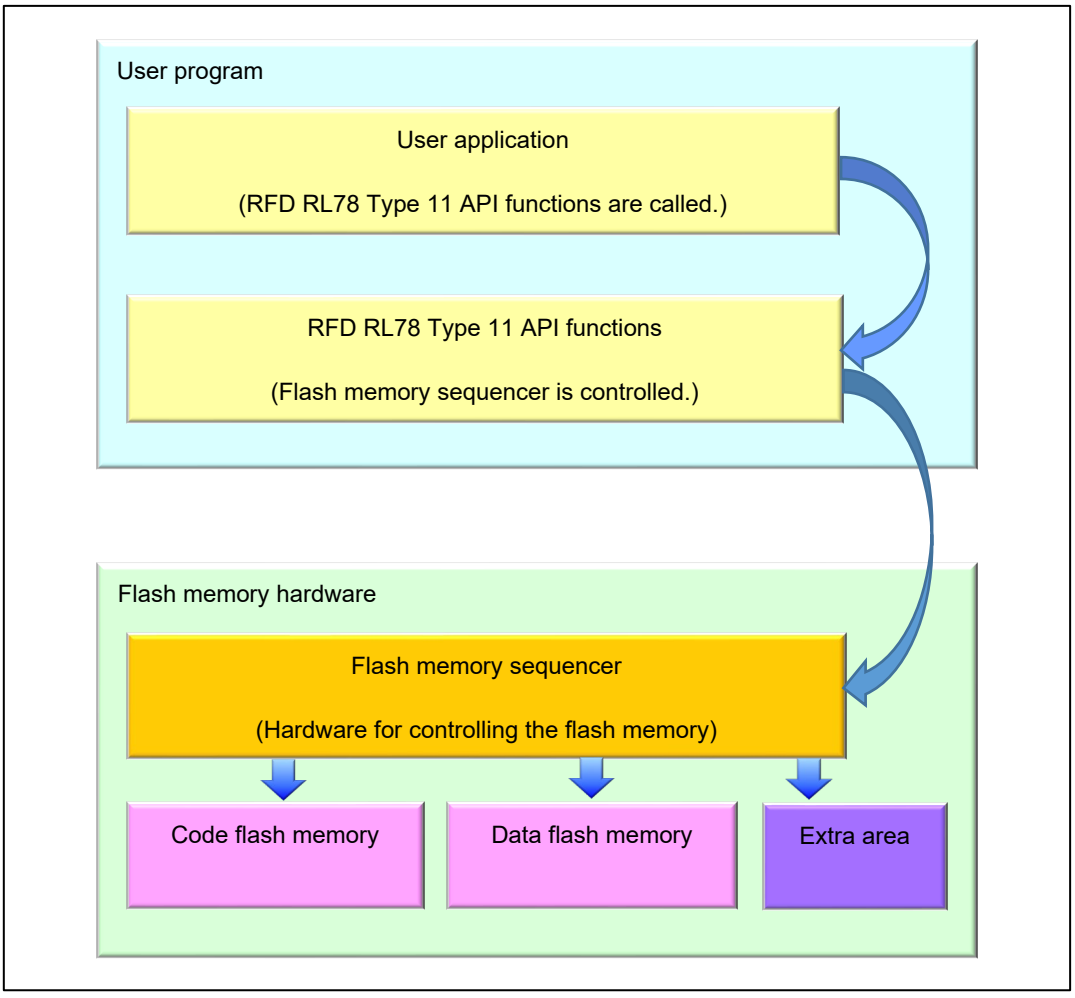


Figure 1-1 Flash Memory Control Using API Functions of RFD RL78 Type 11

1.4 Operating Environment

- Host Computer
  The operation of RFD RL78 Type 11 does not depend on the host computer but the appropriate environment for the C compiler package, debugger and emulator must be prepared. (RFD RL78 Type 11 was developed and tested on Windows10 Enterprise.)

- C Compiler Package
  Table 1-1 shows the target C compiler packages for RFD RL78 Type 11.

Table 1-1 The Target C Compiler Packages for RFD RL78 Type 11

Compiler	IDE（Integrated Development Environment）	Manufacturer	Version
CC-RL	CS+ or e^2 studio	Renesas Electronics	V1.15 or later
IAR	IAR Embedded Workbench® for Renesas RL78	IAR Systems®	V5.10.3 or later
LLVM	e^2 studio	(Open Source Software)	V17.0.1.202412 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 RFD RL78 Type 11 was confirmed.

Table 1-2 Emulator on which RFD RL78 Type 11 Operation was Confirmed

Emulator	Manufacturer
E2 emulator	Renesas Electronics
E2 emulator Lite	Renesas Electronics

- Target MCU
  RL78/L23

- Renesas Flash Driver(RFD) RL78 Type 11
  Table 1-3 shows the Renesas Flash Driver(RFD) RL78 Type 11 supported by this manual.

Table 1-3 The RFD RL78 Type 11 Supported by This Manual

Package	Manufacturer	Package Version
RFD RL78 Type 11	Renesas Electronics	V1.00

1.5 Points for Caution

(1) Reprogramming of the code flash memory or extra area

Place the reprogramming code in RAM when reprogramming the code flash memory or extra area.

However, place the reprogramming code in ROM when reprogramming the code flash memory on bank programming mode.

(2) Precondition for control of the data flash area

Be sure to set the DFLEN bit (bit 0) of the data flash control register (DFLCTL) to 1 (enable access to the data flash area) before controlling the data flash area.

(3) Program execution during reprogramming of the flash memory

Self-programming in the RL78/L23 uses the flash memory sequencer to control the reprogramming of the flash memory. In the following flash memory control modes in which the flash memory can be reprogrammed, the CPU cannot read data from the target flash memory. However, in cases where reprogramming is executed in bank programming mode, it is possible to refer to the startup bank.

- In the code flash memory programming mode, the CPU cannot read data from the code flash memory. The API functions of RFD RL78 Type 11 and the user program to be executed in the code flash memory programming mode should be copied from ROM to RAM in advance and executed and referenced in RAM.
- In the data flash memory programming mode, the CPU cannot read data from the data flash memory. The data to be read in the data flash memory programming mode should be copied from the data flash memory to RAM in advance and referenced in RAM.

(4) Clock setting under self-programming execution

- The high-speed on-chip oscillator should be kept operating during self-programming. When self-programming is executed, it is necessary to use the main system clock and to select “X1 oscillator or the high-speed on-chip oscillator.” Cannot execute self-programming, if the middle-speed on-chip oscillator or the subsystem clock is selected. Stop the middle-speed on-chip oscillator (MIOEN = 0) and select the high-speed on-chip oscillator (MCM1 = 0) as the main on-chip oscillator clock (fOCO).
  - * For the clock setting under self-programming execution, refer to the user's manual of the target RL78 microcontroller.

(5) The precautions in the case of debugging self-programming with an on-chip debugger

In the case which debugs self-programming with an on-chip debugger, because 128 bytes of area is used from the top address of RAM when a debugger is executed, it is necessary to vacate this area. Additionally, in case CS+ or e^2 studio is used as the development environment, the debugger settings need to be configured to use flash self-programming

- Example settings for CS+:

  On the project, select “Connect Settings” tab from “RL78 E2 [Lite] (Debug Tool)”, and set “Yes” to “Flash” - “Using the flash self programming”.
- Example settings for e^2 studio:

  On the project, select “Property” - “Run/Debug Settings”, and edit the target “HardwareDebug” setting. On the displayed screen, select “Debugger” tab - “Connection Settings” tab, and set “Yes” to “Flash” - “Program uses flash self programming”.

(6) The precautions in the case of executing the data copy from ROM to RAM, when using CC-RL compiler.

When using CC-RL compiler, the Sample_INITSCT_xxxx function is called from the main function of main.c file (xxxx = CodeFlash, DataFlash ,Extra or BankSwap). This function copies the data and the program for RFD RL78 Type 11 to RAM from ROM.

However, the following setting will be necessary if this processing is executed by the start-up routine in the cstart.asm file which is a CC-RL compiler function.

(CC-RL compiler function : “Initialization of RAM area sections by using an initialization table [V1.12 or later]”)

- Set “-ram_init_table_section” by linker.
- Set “__USE_RAM_INIT_TABLE” to the column which defines the macro of assemble options.

* For details, please refer to the user's manual of CC-RL compiler

Because “copy processing from ROM to RAM” of a Sample_INITSCT_xxxx function duplicates in this case,It is necessary to set same [Macro definition] as “Compiler Option”, and to cancel processing of a Sample_INITSCT_xxxx function.

- Set “__USE_RAM_INIT_TABLE” to the column which defines the macro of compiler options.

(7) The precautions in case where an on-chip debugger is connected after bank swap execution

The bank swap is executed and the swapping state of the bank 0 and the bank 1 has restriction in on-chip debugger connection. If “the option byte and on-chip debugging security ID” of the rewrite bank side in a chip are not programmed, it will become impossible to connect an on-chip debugger. If bank programming / bank swapping execution, and a debugger are cut, the setup is necessary to subsequent reconnection. It is setting up an Integrated Development Environment so that “the option byte and on-chip debugging security ID” of rewrite bank side may not be erased by program download of debugger reconnection.

- Example settings for CS+:

  On the project, select “Download File Settings” tab from “RL78 E2 [Lite] (Debug Tool)”, and select “Data priority” to “Download” - “Download Mode”.

  Example settings for e^2 studio (CC-RL) : On the project, select “Property” - “Run/Debug Settings”, and edit the target “HardwareDebug” setting. On the displayed screen, select “Debugger” tab - “Connection Settings” tab, and set “No” to “Flash” - “Erase Flash ROM When Starting”.
- Example settings for IAR (Embedded Workbench) : Remove the check mark to “Erase flash before next ID check” in an “E2 Lite Hardware Setup” window.
- Example settings for e^2 studio (LLVM) :

  Description which fills up a debugging monitor area with 0xff is added. (*. ld file, vects.c file)

  - *.ld file:

```
.debug_monitor 0xCE : AT(0xCE)
{
        KEEP(*(.debug_monitor))
} > OCDSTAD
```

  - vects.c file:

```
const unsigned char Debug_Monitor[]  __attribute__ ((section (".debug_monitor"))) = {
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff
};
```

  Caution: The countermeasure method when the option byte in rewrite bank side and on-chip debugging security ID are erased accidentally.

  If problems occur in debugger starting, it is necessary to execute erase (select “Erase Chip” as Erase Option) by the dedicated flash memory programmer. And be sure to restore a chip to an initial state and to redo from the beginning.

1.6 C Compiler Definitions

The definitions of the target compiler written in the header file (r_rfd_compiler.h) for RFD RL78 Type 11 are shown below.

The definitions differ between compilers. The “r_rfd_compiler.h” file is used to identify the current compiler and the definitions for the target compiler are used.

- Definition of CC-RL compiler :
  “__CCRL__” is defined.
  #define COMPILER_CC (1)
- Definition of 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_rfd_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_RFD_FAR_FUNC                                  __far
    #define R_RFD_NO_OPERATION                              __nop
    #define R_RFD_DISABLE_INTERRUPT                         __DI
    #define R_RFD_ENABLE_INTERRUPT                          __EI
    #define R_RFD_GET_PSW_IE_STATE                          __get_psw
    #define R_RFD_IS_PSW_IE_ENABLE(u08_psw_ie_state) (0u != ((u08_psw_ie_state) & 0x80u))
#elif (COMPILER_IAR == COMPILER)
    #define R_RFD_FAR_FUNC                                  __far_func
    #define R_RFD_NO_OPERATION                              __no_operation
    #define R_RFD_DISABLE_INTERRUPT                         __disable_interrupt
    #define R_RFD_ENABLE_INTERRUPT                          __enable_interrupt
    #define R_RFD_GET_PSW_IE_STATE                          __get_interrupt_state
    #define R_RFD_IS_PSW_IE_ENABLE(u08_psw_ie_state) (0u != ((u08_psw_ie_state) & 0x80u))
#elif (COMPILER_LLVM == COMPILER)
    #define R_RFD_FAR_FUNC                                  __far
    #define R_RFD_NO_OPERATION                              __nop
    #define R_RFD_DISABLE_INTERRUPT                         __DI
    #define R_RFD_ENABLE_INTERRUPT                          __EI
    #define R_RFD_GET_PSW_IE_STATE                          (uint8_t)__builtin_rl78_pswie
    #define R_RFD_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=S3 -g -g_line -lang=c99

  - [IAR (Embedded Workbench)]

    Major compile options:

    --core s3 --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 -Wmissing-declarations -Wconversion -Wpointer-arith -Wshadow -Waggregate-return -g -mcpu=s3

2. System Configuration

2.1 File Structure

2.1.1 Folder Structure

Figure 2-1 shows the folder structure of RFD RL78 Type 11.

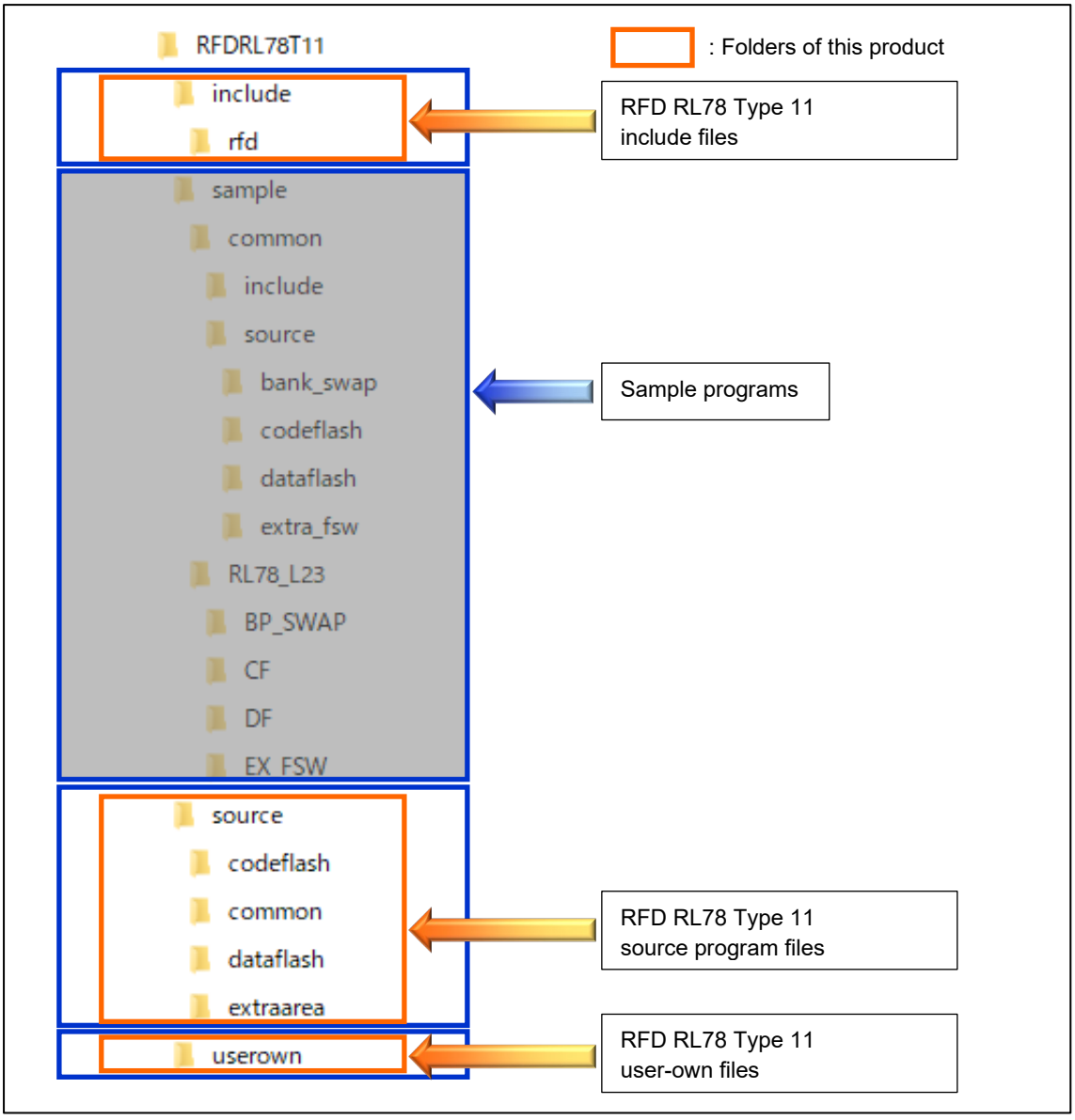


Figure 2-1 Folder Structure of RFD RL78 Type 11

Note: Figure 2-1 shows an example of using RL78/L23. Refer to “5.1.1 Folder Structure“ for the sample folder.

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_rfd_common_api.c	This file contains the API functions for settings used in common for flash memory control.
2	r_rfd_common_control_api.c	This file contains the API functions for command control used in common for flash memory control.
3	r_rfd_common_get_api.c	This file contains the API functions for information acquisition 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_rfd_code_flash_api.c	This file contains the API functions for code flash memory control.
2	r_rfd_code_flash_control_api.c	This file contains the API functions for command control of code flash memory operation.

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_rfd_data_flash_api.c	This file contains the API functions for data flash memory control.
2	r_rfd_data_flash_control_api.c	This file contains the API functions for command control of data flash memory operation.

Table 2-4 shows the program source files in the “source\extraarea\” folder.

Table 2-4 Program Source File in the “source\extraarea\” Folder

No.	Source File Name	Description
1	r_rfd_extra_area_api.c	This file contains the API functions for extra area control.
2	r_rfd_extra_area_security_api.c	This file contains the API functions for the security facilities for the extra area.
3	r_rfd_extra_area_control_api.c	This file contains the API functions for command control of extra area operation.
4	r_rfd_extra_extension_api.c	This file contains the API functions for extended facilities used in common for flash memory control.

Table 2-5 shows the program source file in the “userown\” folder.

Table 2-5 Program Source File in the “userown\” Folder

No.	Source File Name	Description
1	r_rfd_common_userown.c	This file contains the hook functions for user processing to be performed in RFD RL78 Type 11.
2	r_rfd_code_flash_userown.c	This file contains the hook functions for user processing to be performed in code flash memory reprogramming of RFD RL78 Type11.
3	r_rfd_data_flash_userown.c	This file contains the hook functions for user processing to be performed in data flash memory reprogramming of RFD RL78 Type11.
4	r_rfd_extra_area_userown.c	This file contains the hook functions for user processing to be performed in extra area reprogramming of RFD RL78 Type11.

2.1.2.2 Header File List of Header Files

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

Table 2-6 Program Header Files in the “include\rfd” Folder

No.	Header File Name	Description
1	r_rfd.h	Common header file. This file needs to be included when RFD RL78 Type 11 is used.
2	r_rfd_compiler.h	This file describes the definitions that differ between compilers used in RFD RL78 Type 11.
3	r_rfd_memmap.h	This file defines macros to describe sections used in RFD RL78 Type 11.
4	r_rfd_device.h	This file defines the hardware-specific macros used in RFD RL78 Type 11.
5	r_rfd_types.h	This file defines the types of variables used in RFD RL78 Type 11.
6	r_typedefs.h	This file defines the types of data used in RFD RL78 Type 11.

Table 2-7 shows the program header files in the “include\” folder.

Table 2-7 Program Header Files in the “include\” Folder

No.	Header File Name	Description
1	r_rfd_common_api.h	This file defines the prototype declarations of the API functions for setting used in common for flash memory control.
2	r_rfd_code_flash_api.h	This file defines the prototype declarations of the API functions for code flash memory control.
3	r_rfd_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_rfd_common_get_api.h	This file defines the prototype declarations of the API functions for information acquisition used in common for flash memory control.
5	r_rfd_extra_extension_api.h	This file defines the prototype declarations of the API functions for extended facilities used in common for flash memory control.
6	r_rfd_common_userown.h	This file defines the prototype declarations of the hook functions for user processing to be performed in RFD RL78 Type 11.
7	r_rfd_data_flash_api.h	This file defines the prototype declarations of the API functions for data flash memory control.
8	r_rfd_extra_area_api.h	This file defines the prototype declarations of the API functions for extra area control.
9	r_rfd_extra_area_security_api.h	This file defines the prototype declarations of the API functions for the security facilities for the extra area.
10	r_rfd_code_flash_control_api.h	This file defines the prototype declarations of the API functions for command control of code flash memory operation.
11	r_rfd_data_flash_control_api.h	This file defines the prototype declarations of the API functions for command control of data flash memory operation.
12	r_rfd_extra_area_control_api.h	This file defines the prototype declarations of the API functions for command control of extra area operation.
13	r_rfd_code_flash_userown.h	This file defines the prototype declarations of the hook functions for user processing to be performed in code flash memory reprogramming of RFD RL78 Type 11.
14	r_rfd_data_flash_userown.h	This file defines the prototype declarations of the hook functions for user processing to be performed in data flash memory reprogramming of RFD RL78 Type 11.
15	r_rfd_extra_area_userown.h	This file defines the prototype declarations of the hook functions for user processing to be performed in extra area reprogramming of RFD RL78 Type 11.

2.2 Resources of RL78/L23

2.2.1 Memory Map

Table 2-8 shows the memory map (code flash memory [CF : 1 block = 2Kbytes], data flash memory [DF : 1 block = 256 bytes], and RAM) of the RL78/L23.

Table 2-8 Memory Map (Code Flash Memory, Data Flash Memory, and RAM)

RL78	Device	Code Flash Memory: CF	RAM
L23	R7F100LxE (x=F,G,J,L)	64 Kbytes (00000H-0FFFFH)	16 Kbytes (FBF00H-FFEFFH)
	R7F100LxG (x=F,G,J,L,M,P)	128 Kbytes (00000H-1FFFFH)	16 Kbytes (FBF00H-FFEFFH)
	R7F100LxJ (x=F,G,J,L,M,P)	256 Kbytes (00000H-3FFFFH)	32 Kbytes (F7F00H-FFEFFH)
	R7F100LxL (x=F,G,J,L,M,P)	512 Kbytes (00000H-7FFFFH)	32 Kbytes (F7F00H-FFEFFH)
	Data Flash Memory: DF	8 Kbytes (F1000H-F2FFFH) All RL78/L23	

2.2.2 The Allocation of Blocks

Figure 2-2 and Figure 2-3 shows the allocation of blocks in code flash memory (CF) and data flash memory (DF) for RL78/L23.

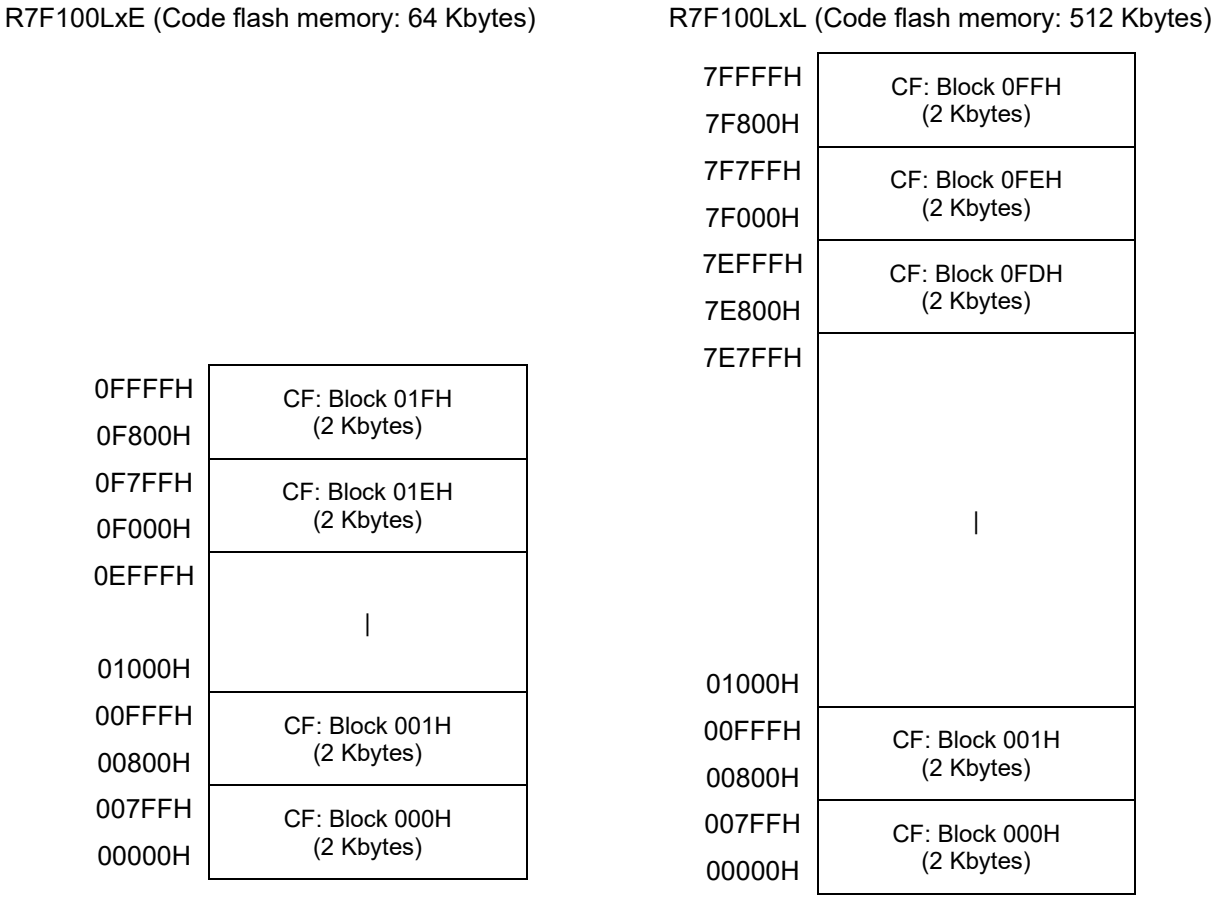


Figure 2-2 Blocks in the Code Flash Memory

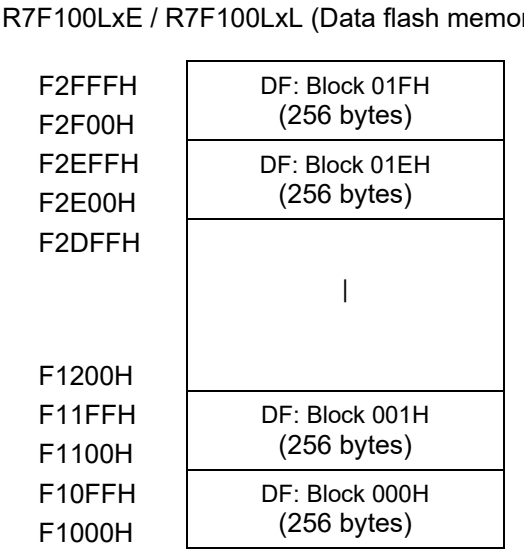


Figure 2-3 Blocks in the Data Flash Memory

2.2.3 List of Registers Related to Flash Memory Sequencer Control

Table 2-9 shows the registers in the RL78/L23 used by RFD RL78 Type 11.

Table 2-9 Registers in the RL78/L23 Used by RFD RL78 Type 11

Base Address	Offset	Register Name	Size	Function Name and Note
F0000H	90H	DFLCTL	1 byte	Data flash control register
	AAH	FLMODE	1 byte	Flash operating mode select register
	ABH	FLMWRP	1 byte	Flash operating mode protect register
	B0H	FLSEC	2 bytes	Flash security flag monitoring register
	B2H	FLFSWS	2 bytes	Flash FSW monitoring register S
	B4H	FLFSWE	2 bytes	Flash FSW monitoring register E
	B6H	FSSET	1 byte	Flash memory sequencer initial setting register
	B7H	FSSE	1 byte	Flash extra area sequencer control register
	C0H	PFCMD	1 byte	Flash protect command register
	C1H	PFS	1 byte	Flash status register
	FFH	VECTCTRL	1 byte	Interrupt vector jump enable register
F0200H	C0H	FLPMC	1 byte	Flash programming mode control register
	C1H	FLARS	1 byte	Flash area selection register
	C2H	FLAPL	2 bytes	Flash address pointer register L
	C4H	FLAPH	1 byte	Flash address pointer register H
	C5H	FSSQ	1 byte	Flash memory sequencer control register
	C6H	FLSEDL	2 bytes	Flash end address pointer register L
	C8H	FLSEDH	1 byte	Flash end address pointer register H
	C9H	FLRST	1 byte	Flash registers initialization register
	CAH	FSASTL	1 byte	Flash memory sequencer status register L
	CBH	FSASTH	1 byte	Flash memory sequencer status register H
	CCH	FLWL	2 bytes	Flash write buffer register L
	CEH	FLWH	2 bytes	Flash write buffer register H
F0400H	80H	FLSIVC0	2 bytes	Interrupt vector change register 0
	82H	FLSIVC1	2 bytes	Interrupt vector change register 1

2.2.4 Flash Operation Mode

(1) The range of operating frequency in each flash operation mode of RL78/L23

Table 2-10 shows the range of operating frequency in each flash operation mode of RL78/L23.

Table 2-10 Operating Frequency Ranges for Individual Flash Operation Modes and Power Supply Voltages

Power Supply Voltage (V_{DD})	Flash Operation Mode	Operating Frequency
1.8 V ≤ V_{DD} ≤ 5.5 V	HS (high-speed main) mode	1 MHz to 32 MHz
	LS (low-speed main) mode	1 MHz to 24 MHz
1.6 V ≤ V_{DD} < 1.8 V	HS (high-speed main) mode	1 MHz to 2 MHz
	LS (low-speed main) mode	1 MHz to 2 MHz

Note: The flash memory cannot be reprogrammed in the LP (low-power main) mode.

2.3 Resources Used in RFD RL78 Type 11

2.3.1 Sections Used in RFD RL78 Type 11

2.3.1.1 Sections Used for Reprogramming of the Code Flash Memory

The CPU cannot read from the code flash memory in the “code flash memory programming mode” used for reprogramming of the code flash memory. The sections allocated as program areas should be copied from ROM to RAM in advance and programs should be executed in RAM. The initial values for the initialized global variable section (RFD_DATA) allocated to RAM should be copied from ROM to RAM in advance according to the directions of the target compiler.

Table 2-11 shows the sections used for reprogramming of the code flash memory and allocations of the sections.

Table 2-11 Sections Used for Reprogramming of the Code Flash Memory

Section Name	Description	Allocation
RFD_CMN	Program section of API functions used in common for flash memory control	ROM
RFD_CF	Program section of API functions for code flash memory control	RAM
RFD_DATA	Data section for initialized global variables	RAM
SMP_CF	Program section of sample functions for code flash memory control	RAM

The case which reprograms a rewrite bank by the bank programming function locates the section of program areas to a startup bank (ROM). The initial values for the initialized global variable section (RFD_DATA) allocated to RAM should be copied from ROM to RAM in advance according to the directions of the target compiler.

Table 2-12 shows the sections used for reprogramming of the code flash memory and allocations of the sections. (The case of bank programming.)

Table 2-12 Sections Used for Reprogramming of the Code Flash Memory (The Case of Bank Programming)

Section Name	Description	Allocation
RFD_CMN	Program section of API functions used in common for flash memory control	ROM
RFD_CF	Program section of API functions for code flash memory control	ROM
RFD_DATA	Data section for initialized global variables	RAM
SMP_CF	Program section of sample functions for code flash memory control	ROM
SMP_BPS	Program code after bank swap execution	ROM

2.3.1.2 Sections Used for Reprogramming of the Data Flash Memory

The initial values for the initialized global variable section (RFD_DATA) allocated to RAM should be copied from ROM to RAM in advance according to the directions of the target compiler.

Table 2-13 shows the sections used for reprogramming of the data flash memory and allocations of the sections.

Table 2-13 Sections Used for Reprogramming of the Data Flash Memory

Section Name	Description	Allocation
RFD_CMN	Program section of API functions used in common for flash memory control	ROM
RFD_DF	Program section of API functions for data flash memory control	ROM
RFD_DATA	Data section for initialized global variables	RAM
SMP_DF	Program section of sample functions for data flash memory control	ROM

2.3.1.3 Sections Used for Reprogramming of the Extra Area

The CPU cannot read from the code flash memory in the “code flash memory programming mode” used for reprogramming of the extra flash memory. The sections allocated as program areas should be copied from ROM to RAM in advance and programs should be executed in RAM. The initial values for the initialized global variable section (RFD_DATA) allocated to RAM should be copied from ROM to RAM in advance according to the directions of the target compiler.

Table 2-14 shows the sections used for reprogramming of the extra area and allocations of the sections.

Table 2-14 Sections Used for Reprogramming of the Extra Area

Section Name	Description	Allocation
RFD_CMN	Program section of API functions used in common for flash memory control	ROM
RFD_EX	Program section of API functions for extra area control	RAM
RFD_DATA	Data section for initialized global variables	RAM
SMP_EX	Program section of sample functions for extra area control	RAM

2.3.2 Code Size and Stack Size which API Functions Use

Table 2-15 shows code size and stack size which API functions for RFD RL78 Type 11 use.

Table 2-15 Code Size and Stack Size which API Functions for RFD RL78 Type 11 Use

API Name	Code Size (Bytes)			Stack Size (Bytes)		
	CC-RL	IAR	LLVM	CC-RL	IAR	LLVM
R_RFD_Init	19	21	18	4	4	4
R_RFD_SetDataFlashAccessMode	15	18	17	4	4	4
R_RFD_ChangeInterruptVector	62	84	68	12	14	12
R_RFD_RestoreInterruptVector	49	66	51	10	10	10
R_RFD_ForceReset	2	2	2	4	4	4
R_RFD_GetSecurityAndBootFlags	6	6	6	4	4	4
R_RFD_GetFSW	46	77	49	10	12	8
R_RFD_EraseCodeFlashReq	34	43	36	4	4	4
R_RFD_WriteCodeFlashReq	28	58	44	4	6	6
R_RFD_BlankCheckCodeFlashReq	34	43	36	4	4	4
R_RFD_SetCFProgrammingMode	60	77	65	10	10	10
R_RFD_SetCFNonProgrammableMode	79	79	74	12	12	12
R_RFD_CheckCFProgrammingMode	10	13	12	4	4	4
R_RFD_CheckCFNonProgrammableMode	11	14	13	4	4	4
R_RFD_CheckCFSeqEndStep1	13	24	13	4	6	4
R_RFD_CheckCFSeqEndStep2	8	19	10	4	6	4
R_RFD_GetCFSeqErrorStatus	8	8	8	4	4	4
R_RFD_ClearCFSeqRegister	11	10	8	4	4	4
R_RFD_ForceStopCFSeq	5	6	5	4	4	4
r_rfd_cf_wait_count	19	19	19	6	6	6
R_RFD_EraseDataFlashReq	26	41	32	4	4	4
R_RFD_WriteDataFlashReq	20	27	23	4	6	6
R_RFD_BlankCheckDataFlashReq	34	76	38	6	12	6
R_RFD_SetDFProgrammingMode	60	77	65	10	10	10
R_RFD_SetDFNonProgrammableMode	79	79	74	12	12	12
R_RFD_CheckDFProgrammingMode	10	13	12	4	4	4
R_RFD_CheckDFNonProgrammableMode	11	14	13	4	4	4
R_RFD_CheckDFSeqEndStep1	13	24	13	4	6	4
R_RFD_CheckDFSeqEndStep2	8	19	10	4	6	4
R_RFD_GetDFSeqErrorStatus	8	8	8	4	4	4
R_RFD_ClearDFSeqRegister	11	10	8	4	4	4
R_RFD_ForceStopDFSeq	5	6	5	4	4	4
r_rfd_df_wait_count	19	19	19	6	6	6
R_RFD_SetExtraEraseProtectReq	26	31	26	4	4	4
R_RFD_SetExtraWriteProtectReq	26	31	26	4	4	4
R_RFD_SetExtraBootAreaProtectReq	26	31	26	4	4	4
R_RFD_SetExtraBootAreaReq	52	82	76	4	6	4
R_RFD_SetExtraFSWProtectReq	29	38	28	4	4	4

R_RFD_SetExtraFSWReq	39	43	43	6	4	8
R_RFD_SetExtraSoftwareReadProtectAreaReq	39	43	43	6	4	8
R_RFD_SetBootAreaImmediately	16	21	18	4	4	4
R_RFD_SetExtraProgrammingMode	60	77	65	10	10	10
R_RFD_SetExtraNonProgrammableMode	79	79	74	12	12	12
R_RFD_CheckExtraProgrammingMode	10	13	12	4	4	4
R_RFD_CheckExtraNonProgrammableMode	11	14	13	4	4	4
R_RFD_CheckExtraSeqEndStep1	13	24	13	4	6	4
R_RFD_CheckExtraSeqEndStep2	6	19	8	4	6	4
R_RFD_GetExtraSeqErrorStatus	8	8	8	4	4	4
R_RFD_ClearExtraSeqRegister	11	10	8	4	4	4
r_rfd_extra_wait_count	19	19	19	6	6	6
R_RFD_HOOK_EnterCriticalSection	9	9	11	4	4	4
R_RFD_HOOK_ExitCriticalSection	11	10	9	4	4	4
R_RFD_HOOK_EnterCFCriticalSection	9	9	11	4	4	4
R_RFD_HOOK_ExitCFCriticalSection	11	10	9	4	4	4
R_RFD_HOOK_EnterDFCriticalSection	9	9	11	4	4	4
R_RFD_HOOK_ExitDFCriticalSection	11	10	9	4	4	4
R_RFD_HOOK_EnterExtraCriticalSection	9	9	11	4	4	4
R_RFD_HOOK_ExitExtraCriticalSection	11	10	9	4	4	4

3. API Functions of RFD RL78 Type 11

3.1 List of API Functions of RFD RL78 Type 11

3.1.1 API Functions Used in Common for Flash Memory Control

Table 3-1 shows the API functions used in common for flash memory control in RFD RL78 Type 11.

Table 3-1 API Functions Used in Common for Flash Memory Control in RFD RL78 Type 11

	API Name	Overview
1	R_RFD_Init	Sets the frequency specified by the parameter in the flash memory sequencer and initializes RFD RL78 Type 11.
2	R_RFD_SetDataFlashAccessMode	Enables or disables access to the data flash memory according to the parameter setting.
3	R_RFD_ChangeInterruptVector	Changes the branch destination address for all interrupts to the RAM address specified by the parameter.
4	R_RFD_RestoreInterruptVector	Changes the branch destination address for interrupts that was changed to a RAM address back to the normal interrupt vector addresses.
5	R_RFD_ForceReset	Generates an internal reset of the CPU.
6	R_RFD_GetSecurityAndBootFlags	Acquires the information on the security flags (protection flags) and boot area switching flag.
7	R_RFD_GetFSW	Acquires the range of the flash shield window, the flash shield window area, and the protection flag value.

3.1.2 API Functions for Code Flash Memory Control

Table 3-2 shows the API functions for code flash memory control in RFD RL78 Type 11.

Table 3-2 API Functions for Code Flash Memory Control in RFD RL78 Type 11

	API Name	Overview
1	R_RFD_EraseCodeFlashReq	Activates the code/data flash area sequencer and begins the erasure of the code flash memory (one block).
2	R_RFD_WriteCodeFlashReq	Activates the code/data flash area sequencer and begins the programming of the code flash memory (4 bytes).
3	R_RFD_BlankCheckCodeFlashReq	Activates the code/data flash area sequencer and begins the blank check of the code flash memory (one block).
4	R_RFD_SetCFProgrammingMode	The mode of a flash memory sequencer transfers to code flash programming mode and then sets the specified CPU operating frequency in the flash memory sequencer.
5	R_RFD_SetCFNonProgrammableMode	The mode of a flash memory sequencer transfers from code flash programming mode to non-programmable mode.
6	R_RFD_CheckCFProgrammingMode	Check whether flash memory control mode is code flash programming mode.
7	R_RFD_CheckCFNonProgrammableMode	Check whether flash memory control mode is non-programmable mode.
8	R_RFD_CheckCFSeqEndStep1	Checks if the operation of the activated code/data flash area sequencer has been completed.
9	R_RFD_CheckCFSeqEndStep2	Checks if the command operation has been completed after the flash memory sequencer control register is cleared.
10	R_RFD_GetCFSeqErrorStatus	Acquires the information on errors that occurred during command execution in the code/data flash area sequencer.
11	R_RFD_ClearCFSeqRegister	Clears the registers for controlling the code/data flash area sequencer and extra area sequencer.
12	R_RFD_ForceStopCFSeq	Forcibly stops the operation of the code/data flash area sequencer.
13	r_rfd_cf_wait_count	Executes a software loop to wait for the time specified by the parameter (time count in units of 1 µs).

3.1.3 API Functions for Data Flash Memory Control

Table 3-3 shows the API functions for data flash memory control in RFD RL78 Type 11.

Table 3-3 API Functions for Data Flash Memory Control in RFD RL78 Type 11

	API Name	Overview
1	R_RFD_EraseDataFlashReq	Activates the code/data flash area sequencer and begins the erasure of the data flash memory (one block).
2	R_RFD_WriteDataFlashReq	Activates the code/data flash area sequencer and begins the programming of the data flash memory (1 byte).
3	R_RFD_BlankCheckDataFlashReq	Activates the code/data flash area sequencer and begins the blank check of the data flash memory (Specified number of bytes).
4	R_RFD_SetDFProgrammingMode	The mode of a flash memory sequencer transfers to data flash programming mode and then sets the specified CPU operating frequency in the flash memory sequencer.
5	R_RFD_SetDFNonProgrammableMode	The mode of a flash memory sequencer transfers from data flash programming mode to non-programmable mode.
6	R_RFD_CheckDFProgrammingMode	Check whether flash memory control mode is data flash programming mode.
7	R_RFD_CheckDFNonProgrammableMode	Check whether flash memory control mode is non-programmable mode.
8	R_RFD_CheckDFSeqEndStep1	Checks if the operation of the activated code/data flash area sequencer has been completed.
9	R_RFD_CheckDFSeqEndStep2	Checks if the command operation has been completed after the flash memory sequencer control register is cleared.
10	R_RFD_GetDFSeqErrorStatus	Acquires the information on errors that occurred during command execution in the code/data flash area sequencer.
11	R_RFD_ClearDFSeqRegister	Clears the registers for controlling the code/data flash area sequencer and extra area sequencer.
12	R_RFD_ForceStopDFSeq	Forcibly stops the operation of the code/data flash area sequencer.
13	r_rfd_df_wait_count	Executes a software loop to wait for the time specified by the parameter (time count in units of 1 µs).

3.1.4 API Functions for Extra Area Control

Table 3-4 shows the API functions for extra area control in RFD RL78 Type 11.

Table 3-4 API Functions for Extra Area Control in RFD RL78 Type 11

	API Name	Overview
1	R_RFD_SetExtraEraseProtectReq	Activates the extra area sequencer and begins the setting of the block erase-disabled flag.
2	R_RFD_SetExtraWriteProtectReq	Activates the extra area sequencer and begins the setting of the write-disabled flag.
3	R_RFD_SetExtraBootAreaProtectReq	Activates the extra area sequencer and begins the setting of the boot area rewrite-disabled flag.
4	R_RFD_SetExtraBootAreaReq	Activates the extra area sequencer and begins the setting of the boot area switching flag.
5	R_RFD_SetExtraFSWProtectReq	Activates the extra area sequencer and begins the setting of the flag for protection against flash shield window modification.
6	R_RFD_SetExtraFSWReq	Activates the extra area sequencer and begins the setting of the range and area control of the flash shield window specified by the parameters.
7	R_RFD_SetExtraSoftwareReadProtectAreaReq	Activates the extra area sequencer and begins the setting of the flash read protection.
8	R_RFD_SetBootAreaImmediately	[For use in boot swapping] Allocates the boot cluster specified by the parameter to the boot area immediately. [For use in bank swapping] Allocates the bank specified by the parameter to the startup bank immediately.
9	R_RFD_SetExtraProgrammingMode	The mode of a flash memory sequencer transfers to code flash programming mode and then sets the specified CPU operating frequency in the flash memory sequencer.
10	R_RFD_SetExtraNonProgrammableMode	The mode of a flash memory sequencer transfers from code flash programming mode to non-programmable mode.
11	R_RFD_CheckExtraProgrammingMode	Check whether flash memory control mode is code flash programming mode.
12	R_RFD_CheckExtraNonProgrammableMode	Check whether flash memory control mode is non-programmable mode.
13	R_RFD_CheckExtraSeqEndStep1	Checks if the operation of the activated extra area sequencer has been completed.
14	R_RFD_CheckExtraSeqEndStep2	Checks if the command operation has been completed after the flash extra area sequencer control register is cleared.
15	R_RFD_GetExtraSeqErrorStatus	Acquires the information on errors that occurred during command execution in the code/data flash area sequencer or extra area sequencer.
16	R_RFD_ClearExtraSeqRegister	Clears the registers for controlling the code/data flash area sequencer and extra area sequencer.
17	r_rfd_extra_wait_count	Executes a software loop to wait for the time specified by the parameter (time count in units of 1 µs).

3.1.5 Hook Functions

Table 3-5 shows the hook functions in RFD RL78 Type 11.

Table 3-5 Hook Functions in RFD RL78 Type 11

	API Name	Overview
1	R_RFD_HOOK_EnterCriticalSection	Save the state of an interrupt enable flag (IE), executes the instruction for disabling interrupts.
2	R_RFD_HOOK_ExitCriticalSection	Restore the state of the saved interrupt enable flag (IE).
3	R_RFD_HOOK_EnterCFCriticalSection	Save the state of an interrupt enable flag (IE), executes the instruction for disabling interrupts.
4	R_RFD_HOOK_ExitCFCriticalSection	Restore the state of the saved interrupt enable flag (IE).
5	R_RFD_HOOK_EnterDFCriticalSection	Save the state of an interrupt enable flag (IE), executes the instruction for disabling interrupts.
6	R_RFD_HOOK_ExitDFCriticalSection	Restore the state of the saved interrupt enable flag (IE).
7	R_RFD_HOOK_EnterExtraCriticalSection	Save the state of an interrupt enable flag (IE), executes the instruction for disabling interrupts.
8	R_RFD_HOOK_ExitExtraCriticalSection	Save the state of an interrupt enable flag (IE), executes the instruction for disabling interrupts.

3.2 Data Type Definitions

3.2.1 Data Types

Table 3-6 shows the data type definitions in RFD RL78 Type 11.

Table 3-6 Data Type Definitions in RFD RL78 Type 11

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
bool	unsigned char	Boolean value (false = 0, true = 1)

Remark: In the C language standard C 99 and later, these data types are defined in “stdint.h” and “stdbool.h”.

3.2.2 Global Variables

The following shows the global variables used in RFD RL78 Type 11.

(1) g_u08_change_interrupt_vector_flag

Type/Name	uint8_t g_u08_change_interrupt_vector_flag
Default value	0x00 (R_RFD_VALUE_U08_INIT_VARIABLE)
Description	Execution flag for the R_RFD_ChangeInterruptVector function — R_RFD_VALUE_U08_SET_FWEDIS_FLAG_ON: 0x55u — R_RFD_VALUE_U08_SET_FWEDIS_FLAG_OFF: 0x00u
Definition file	r_rfd_common_api.c

(2) g_u08_cpu_frequency

Type/Name	uint8_t g_u08_cpu_frequency
Default value	0x00 (R_RFD_VALUE_U08_INIT_VARIABLE)
Description	CPU operating frequency (RL78/L23: 1 MHz to 32 MHz) - Value of (CPU operating frequency – 1): 0x00u to 0x1Fu (0 to 31)
Definition file	r_rfd_common_api.c

(3) sg_u08_xx_psw_ie_state

Type/Name	static uint8_t sg_u08_psw_ie_state static uint8_t sg_u08_cf_psw_ie_state static uint8_t sg_u08_df_psw_ie_state static uint8_t sg_u08_extra_psw_ie_state
Default value	0x00 (R_RFD_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_rfd_common_userown.c r_rfd_code_flash_userown.c r_rfd_data_flash_userown.c r_rfd_extra_area_userown.c

Note: The user needs to implement the processing for copying the initial values to be assigned to the initialized global variables from the Data section in ROM to RAM.

3.2.3 Enumerations

- e_rfd_df_access (enumerated-type variable name: e_rfd_df_access_t)
  Data flash memory access control

Symbol Name	Value	Description
R_RFD_ENUM_DF_ACCESS_DISABLE	0x00	Access to the data flash memory is disabled.
R_RFD_ENUM_DF_ACCESS_ENABLE	0x01	Access to the data flash memory is enabled.

- e_rfd_boot_cluster (enumerated-type variable name: e_rfd_boot_cluster_t)
  Boot cluster number

Symbol Name	Value	Description
R_RFD_ENUM_BOOT_CLUSTER_1	0x00	Boot cluster 1
R_RFD_ENUM_BOOT_CLUSTER_0	0x01	Boot cluster 0

- e_rfd_fsw_mode (enumerated-type variable name: e_rfd_fsw_mode_t)
  Flash shield window area

Symbol Name	Value	Description
R_RFD_ENUM_FSW_MODE_INSIDE	0x00	Inside shield area
R_RFD_ENUM_FSW_MODE_OUTSIDE	0x01	Outside shield area

- e_rfd_protect (enumerated-type variable name: e_rfd_protect_t)
  Protection enable or disable

Symbol Name	Value	Description
R_RFD_ENUM_PROTECT_ON	0x00	Protection is enabled.
R_RFD_ENUM_PROTECT_OFF	0x01	Protection is disabled.

- e_rfd_ret (enumerated-type variable name: e_rfd_ret_t)
  Return values

Symbol Name	Value	Description
R_RFD_ENUM_RET_STS_OK	0x00	Normal end
R_RFD_ENUM_RET_STS_BUSY	0x01	Busy
R_RFD_ENUM_RET_ERR_PARAMETER	0x10	Parameter error
R_RFD_ENUM_RET_ERR_MODE_MISMATCHED	0x11	Mode mismatch error

3.2.4 Macro Definitions

3.2.4.1 Macro Definitions for Setting the Global Data of RFD

- 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_RFD_VALUE_U08_MASK1_8BIT	0xFFu	8-bit mask value
R_RFD_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_RFD_VALUE_U08_SHIFT_8BIT	8u	Value for 8-bit shifting
R_RFD_VALUE_U08_SHIFT_16BIT	16u	Value for 16-bit shifting

- Macro definitions for the g_u08_change_interrupt_vector_flag global data
  Whether the interrupt branch destination is specified by the vector table in ROM or the specified address in RAM is used is defined.

Symbol Name	Value	Description
R_RFD_VALUE_U08_SET_FWEDIS_FLAG_ON	0x55u	The R_RFD_ChangeInterruptVector function has been executed. Execution after an interrupt branch to the specified address in RAM.
R_RFD_VALUE_U08_SET_FWEDIS_FLAG_OFF	0x00u	The R_RFD_ChangeInterruptVector function has not been executed. Execution after an interrupt branch to the address specified by the vector table in ROM.

- Macro definitions for Initial value settings
  Defines the initial value of the global variable.

Symbol Name	Value	Description
R_RFD_VALUE_U08_INIT_VARIABLE	0x00u	Initial value of the global variable
R_RFD_VALUE_U08_INIT_FLAG	0x00u	Initial value of the flag

3.2.4.2 Macro Definitions for Setting the Registers and Extra Area in the RL78/L23

- Macro definitions for DFLCTL (data flash control register)
  Whether to enable or disable access to the data flash memory is specified.
  Target register definition: R_RFD_REG_U08_DFLCTL
  (Target bit [DFLEN]: R_RFD_REG_U01_DFLCTL_DFLEN)

Symbol Name	Value	Description
R_RFD_VALUE_U01_ DFLEN_DATA_FLASH_ACCESS_DISABLE	0u	Access to the data flash memory is disabled.
R_RFD_VALUE_U01_ DFLEN_DATA_FLASH_ACCESS_ENABLE	1u	Access to the data flash memory is enabled.

- Macro definitions for FLARS (flash area selection register)
  The target area of access is specified.

  Target register definition: R_RFD_REG_U08_FLARS

Symbol Name	Value	Description
R_RFD_VALUE_U08_FLARS_USER_AREA	0x00u	The user area is specified.
R_RFD_VALUE_U08_FLARS_EXTRA_AREA	0x01u	The extra area is specified.

- 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_RFD_REG_U08_FSSQ

Symbol Name	Value	Description
R_RFD_VALUE_U08_FSSQ_WRITE	0x81u	Write command for the flash memory
R_RFD_VALUE_U08_FSSQ_BLANKCHECK_CF	0x83u	Blank check command for the code flash memory
R_RFD_VALUE_U08_FSSQ_BLANKCHECK_DF	0x8Bu	Blank check command for the data flash memory
R_RFD_VALUE_U08_FSSQ_ERASE	0x84u	Erase command for the flash memory
R_RFD_VALUE_U08_FSSQ_CLEAR	0x00u	Value for clearing the settings for operation of the flash memory sequencer

- Macro definition 2 for FSSQ (flash memory sequencer control register)
  The value of the bit for forcibly stopping the flash memory sequencer is defined.
  [Bit 6] FSSTP: Bit for forcibly stopping the sequencer.
  The sequencer is forcibly stopped when FSSTP = 1.

  Target register definition: R_RFD_REG_U01_FSSQ_FSSTP

Symbol Name	Value	Description
R_RFD_VALUE_U01_FSSQ_FSSTP_ON	1u	Value for forcibly stopping the flash memory sequencer

- Macro definitions for FSSE (flash extra area sequencer control register)
  The commands to be executed in the activated extra area sequencer are defined.
  [Bit 7] ESQST: Bit for starting or stopping the sequencer.
  The sequencer starts operation when ESQST = 1.
  [Bits 3 to 0] ESQMD3 to ESQMD0: Command for the extra area sequencer

  Target register definition: R_RFD_REG_U08_FSSE

Symbol Name	Value	Description
R_RFD_VALUE_U08_FSSE_FSW	0x81u	Command for setting the flash shield window function
R_RFD_VALUE_U08_FSSE_SOFTWARE_READ	0x86u	Command for setting the flash read protection
R_RFD_VALUE_U08_FSSE_SECURITY_FLAG	0x87u	Command for setting the security flag
R_RFD_VALUE_U08_FSSE_CLEAR	0x00u	Value for clearing the settings for operation of the extra area sequencer

- Macro definition for PFCMD (flash protect command register)
  The fixed value to be written to the register that is used to write-protect specific registers is defined.

  Target register definition: R_RFD_REG_U08_PFCMD

Symbol Name	Value	Description
R_RFD_VALUE_U08_ PFCMD_SPECIFIC_SEQUENCE_WRITE	0xA5u	Value for releasing protection in the specific sequence for the flash memory sequencer

- Macro definition for PFS (flash states register)
  [Bit0] FPRERR: Error status of the specific sequence. FRPERR = 1 indicates a protection error.

  Target register definition: R_RFD_REG_U08_PFS

Symbol Name	Value	Description
R_RFD_VALUE_U08_MASK1_PFS_FPRERR	0x01u	Value for comparing the protection error which happened while executing the specific sequence.

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

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

  [Bit 4] EEEMD: Bit for specifying the data flash memory control mode.

  The data flash memory programming mode is entered when EEEMD = 1.

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

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

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

  Target register definition: R_RFD_REG_U08_FLPMC

Symbol Name	Value	Description
R_RFD_VALUE_U08_FLPMC_MODE_ NONPROGRAMMABLE_FWEDIS_ENABLE	0x00u	The flash memory sequencer is in the non-programmable mode. Execution after an interrupt branch to RAM. [The R_RFD_ChangeInterruptVector function has been executed.]
R_RFD_VALUE_U08_FLPMC_MODE_ NONPROGRAMMABLE_FWEDIS_DISABLE	0x08u	The flash memory sequencer is in the non-programmable mode. Execution after an interrupt branch to the address specified by the vector table in ROM. [The R_RFD_ChangeInterruptVector function has not been executed.]
R_RFD_VALUE_U08_FLPMC_MODE_ CODE_FLASH_PROGRAMMING	0x02u	Code flash memory programming mode
R_RFD_VALUE_U08_FLPMC_MODE_ DATA_FLASH_PROGRAMMING	0x10u	Data flash memory programming mode
R_RFD_VALUE_U08_MASK0_FLPMC_FWEDIS	0xF7u	Mask value for checking the FWEDIS bit

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

  The end state of the flash memory sequencer (extra area sequencer or code/data flash area sequencer) is defined.

  [Bit 7] ESQEND: End state of the extra area sequencer. ESQEND = 1 indicates that the sequencer has completed operation. This bit is cleared when the ESQST bit is cleared.

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

  Target register definition: R_RFD_REG_U08_FSASTH

Symbol Name	Value	Description
R_RFD_VALUE_U08_MASK1_FSASTH_SQEND	0x40u	Value to be compared with the end state of the code/data flash area sequencer
R_RFD_VALUE_U08_MASK1_FSASTH_ESQEND	0x80u	Value to be compared with the end state of the extra area 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 (extra area sequencer or code/data flash area sequencer) is finished is defined.

  [Bit 5] ESEQER: Error status of the extra area sequencer. ESEQER = 1 indicates a sequencer error.

  [Bit 4] SEQER: Error status of the code/data flash area sequencer. SEQER = 1 indicates a sequencer error.

  [Bit 3] BLER: Error status of the blank check command. BLER = 1 indicates a blank 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_RFD_REG_U08_FSASTL

Symbol Name	Value	Description
R_RFD_VALUE_U08_ MASK1_FSASTL_ERROR_FLAG	0x3Bu	Value of the error status mask when the operation of the flash memory sequencer (extra area sequencer or code/data flash area sequencer) is finished.

- Macro definitions 1 for FSSET (flash memory sequencer initial setting register)

  The boot swapping setting bit, temporary boot swapping setting bit, or other setting bits are masked by ANDing with 0.

  [Bit 7] TMSPMD: Boot swapping setting. When TMSPMD = 0, boot swapping is executed according to the information in the extra area. When TMSPMD = 1, boot swapping is executed according to the TMBTSEL bit setting.

  [Bit 6] TMBTSEL: Temporary boot swapping setting. When TMBTSEL = 0, boot cluster 0 is selected as the boot area. When TMBTSEL = 1, boot cluster 1 is selected as the boot area.

  Target register definition: R_RFD_REG_U08_FSSET

Symbol Name	Value	Description
R_RFD_VALUE_U08_MASK0_ FSSET_TMSPMD_AND_TMBTSEL	0x3Fu	The boot swapping setting and temporary boot swapping setting are masked.
R_RFD_VALUE_U08_MASK1_ FSSET_TMSPMD_AND_TMBTSEL	0xC0u	The bits other than the boot swapping setting or temporary boot swapping setting are masked.
R_RFD_VALUE_U08_MASK1_FSSET_TMSPMD	0x80u	The bits other than the boot swapping setting are masked.
R_RFD_VALUE_U08_ FSSET_BOOT_CLUSTER_0	0x80u	Value for specifying boot cluster 0 for temporary boot swapping.
R_RFD_VALUE_U08_ FSSET_BOOT_CLUSTER_1	0xC0u	Value for specifying boot cluster 1 for temporary boot swapping.

- Macro definitions 2 for FSSET (flash memory sequencer initial 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: Enter the CPU operating frequency converted for the FSSET register.

  Target register definition: R_RFD_REG_U08_FSSET

Symbol Name	Value	Description
R_RFD_VALUE_U08_FREQUENCY_LOWER_LIMIT	1u	Lowest allowable operating frequency (1 MHz)
R_RFD_VALUE_U08_FREQUENCY_UPPER_LIMIT	32u	Highest allowable operating frequency (32 MHz)
R_RFD_VALUE_U08_FREQUENCY_ADJUST	1u	Correction value (-1) for conversion of the FSSET register setting

- Macro definitions for VECTCTRL (Interrupt address control register)

  The register values for selecting whether to branch to the vector address in ROM or the specified address in RAM after the occurrence of an interrupt during self-programming are specified.

  Target register definition: R_RFD_REG_U08_VECTCTRL

Symbol Name	Value	Description
R_RFD_VALUE_U08_VECTCTRL_OFF	0x00u	Register value for branching to the vector address in ROM corresponding to each interrupt
R_RFD_VALUE_U08_VECTCTRL_ON	0x01u	Register value for branching to a user-specified single address in RAM after any interrupt

- Macro definitions for FLRST (flash registers initialization register)

  The values for specifying the initialization of the registers for the flash memory sequencer (extra area sequencer or code/data flash area sequencer) are defined.

  [Bit 0] FLRST: When FLRST = 1, the registers for the flash memory sequencer (extra area sequencer or code/data flash area sequencer) are initialized.

  Target register definition: R_RFD_REG_U08_FLRST

Symbol Name	Value	Description
R_RFD_VALUE_U08_FLRST_ON	0x01u	Value for initializing the sequencer registers
R_RFD_VALUE_U08_FLRST_OFF	0x00u	Value for not initializing the sequencer registers

- Macro definitions for FLFSWS and FLFSWE (flash FSW monitoring registers START and END)
  The mask values used to acquire or make the FSW settings are defined.
  FLFSWE [bit 15] FSWC: The target area of FSW is specified. FSWC = 0 specifies the inside of the specified range and FSWC = 1 specifies the outside of the specified range.
  FLFSWE [bits 8 to 0]: The end block number + 1 of FSW is specified.
  FLFSWS [bit 15] FSPR: Modification of the FSW settings is disabled. FSPR = 0 disables modification.
  FLFSWS [bits 8 to 0]: The FSW start block number is specified.
  Target register definitions: R_RFD_REG_U16_FLFSWE and R_RFD_REG_U16_FLFSWS

(1) Mask values for acquiring FSW settings

Symbol Name	Value	Description
R_RFD_VALUE_U16_MASK1_FLFSW_BLOCK_NUMBER	0x01FFu	Mask value for acquiring the block number setting
R_RFD_VALUE_U16_MASK1_FLFSWE_FSWC	0x8000u	Mask value for acquiring the FSW target area setting (FSWC)
R_RFD_VALUE_U16_MASK1_FLFSWS_FSPR	0x8000u	Mask value for acquiring the modification disabling setting (FSPR)

(2) Mask value for making FSW settings

Symbol Name	Value	Description
R_RFD_VALUE_U16_MASK0_FSW_PROTECT_FLAG	0x7FFFu	Mask value for setting the FSW protection
R_RFD_VALUE_U16_MASK0_FSW_CONTROL_FLAG	0x7FFFu	Mask value for setting the FSW area
R_RFD_VALUE_U16_MASK1_FSW_EXCEPT_BLOCK_INFO	0xFE00u	Mask value for setting the FSW block

- 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 = 256bytes) and blank check for the data flash memory are defined.

FLAPH [Bits 3 to 0]: FLAP19 to FLAP16 specify the upper bits of the start address of a data flash memory area. This value is fixed to 0x0F.

FLAPL [Bits 15 to 0]: FLAP15 to FLAP0 specify the lower bits of the start address of a data flash memory area.

FLSEDH [Bits 3 to 0]: EWA19 to EWA16 specify the upper bits of the end address of a data flash memory area. This value is fixed to 0x0F.

FLSEDL [Bits 15 to 0]: EWA15 to EWA0 specify the lower bits of the end address of a data flash memory area.

Target register definitions: R_RFD_REG_U08_FLAPH, R_RFD_REG_U16_FLAPL, R_RFD_REG_U08_FLSEDH, and R_RFD_REG_U16_FLSEDL

Symbol Name	Value	Description
R_RFD_VALUE_U16_ DATA_FLASH_ADDR_LOW	0x1000u	Value for the lower bits of the start address of a data flash area (16 bits)
R_RFD_VALUE_U08_ DATA_FLASH_ADDR_HIGH	0x0Fu	Value for the upper bits of the start address of a data flash area (8 bits)
R_RFD_VALUE_U16_ DATA_FLASH_BLOCK_ADDR_END	0x00FFu	Value for the lower bits of the end address of a data flash block (16 bits)
R_RFD_VALUE_U08_ DATA_FLASH_SHIFT_LOW_ADDR	8u	Value for shifting the lower address bits to calculate the offset of a data flash area from the block number

(2) The start and end addresses of erasure and blank check (1 block = 2-Kbyte) for the code flash memory are defined.

FLAPH [Bits 3 to 0]: FLAP19 to FLAP16 specify the upper bits of the start address of a code flash memory area.

FLAPL [Bits 15 to 0]: FLAP15 to FLAP0 specify the lower bits of the start address of a code flash memory area.

FLSEDH [Bits 3 to 0]: EWA19 to EWA16 specify the upper bits of the end address of a code flash memory area.

FLSEDL [Bits 15 to 0]: EWA15 to EWA0 specify the lower bits of the end address of a code flash memory area.

Target register definitions: R_RFD_REG_U08_FLAPH, R_RFD_REG_U16_FLAPL, R_RFD_REG_U08_FLSEDH, and R_RFD_REG_U16_FLSEDL

Symbol Name	Value	Description
R_RFD_VALUE_U16_ CODE_FLASH_BLOCK_ADDR_LOW	0x001Fu	Mask value for the lower bits of the start address of a code flash block (16 bits)
R_RFD_VALUE_U16_ CODE_FLASH_BLOCK_ADDR_HIGH	0x01E0u	Mask value for the upper bits of the start address of a code flash block (16 bits; only the lower 8 bits after shifting are used)
R_RFD_VALUE_U16_ CODE_FLASH_BLOCK_ADDR_END	0x07FFu	Lower address in 2-Kbyte units of the end of a code flash block (16 bits)
R_RFD_VALUE_U08_ CODE_FLASH_SHIFT_LOW_ADDR	11u	Value for shifting the lower address bits to calculate the offset of a code flash area from the block number
R_RFD_VALUE_U08_ CODE_FLASH_SHIFT_HIGH_ADDR	5u	Value for shifting the upper address bits to calculate the offset of a code flash area from the block number

Example: Block number = 107 -> 0x006B
R_RFD_VALUE_U16_CODE_FLASH_BLOCK_ADDR_LOW : 0x000B -> 0x5800 (shifted to the left by 11 bits)
R_RFD_VALUE_U16_CODE_FLASH_BLOCK_ADDR_HIGH : 0x0060 -> 0x0003 (shifted to the right by 5 bits)
Block start address = 0x0003_5800

- Macro definitions for FLSEC (Flash security flag monitoring register)
  The mask values for extra area settings and security monitoring are defined.
  [Bit 12] WRPR: Write-disabled flag. WRPR = 0 disables programming.
  [Bit 10] SEPR: Block erase-disabled flag. SEPR = 0 disables block erasure.
  [Bit 9] BTPR : Changing of the boot cluster size and bank swapping setting (BTBLS[3:0]) disabled flag
  Boot swapping setting : Block(Boot area) erase-disabled flag
  Writing of the boot area is disabled at BTPR=0.
  Bank swapping setting : Block(startup bank) erase-disabled flag
  Writing of the startup bank is disabled at BTPR=0.
  [Bit 8] BTFLG : Boot area switching flag / startup bank switching flag
  Boot swapping setting : BTFLG = 0, Boot cluster 1 as the boot area.
  BTFLG = 1, Boot cluster 0 as the boot area.
  Bank swapping setting : BTFLG = 0, Bank 1 as the startup bank.
  BTFLG = 1, Bank 0 as the startup bank.
  Target register definitions: R_RFD_REG_U16_FLWH, R_RFD_REG_U16_FLWL, and R_RFD_REG_U16_FLSEC

Symbol Name	Value	Description
R_RFD_VALUE_U16_MASK0_ERASE_PROTECT_FLAG	0xFBFFu	Mask value for setting the block erasure protection
R_RFD_VALUE_U16_MASK0_WRITE_PROTECT_FLAG	0xEFFFu	Mask value for setting the write protection
R_RFD_VALUE_U16_MASK0_ BOOT_CLUSTER_PROTECT_FLAG	0xFDFFu	Mask value for setting the protection against rewriting of the boot area
R_RFD_VALUE_U16_MASK0_BOOT_FLAG	0xFEFFu	Mask value for switching and monitoring the boot area flag
R_RFD_VALUE_U16_MASK1_BOOT_FLAG	0x0100u	Mask value for switching and monitoring the boot area flag

- Macro definitions for setting the flash read protection
  The mask values for extra area settings are defined.
  The extra area for setting flash read protection:
  [Bit 31] SWPR: Modification of the flash read protection settings is disabled. SWPR = 0 disables modification.
  [Bits 24 to 16]: End number of the blocks for the flash read protection.
  [Bits 8 to 0]: Start number of the blocks for the flash read protection.
  Target register definition: None (extra area settings only)

Symbol Name	Value	Description
R_RFD_VALUE_U16_MASK0_ SW_READ_PROTECT_FLAG	0x7FFFu	Mask value for setting flash read protection.
R_RFD_VALUE_U16_MASK1_ SW_READ_EXCEPT_BLOCK_INFO	0xFE00u	Mask value for setting the blocks for flash read protection.

3.3 Specifications of API Functions

This section describes the detailed specifications of the API functions of Renesas Flash Driver (RFD) RL78 Type 11.

There are some prerequisites for using the API functions of RFD RL78 Type 11 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_RFD_Init() function once before starting the use of RFD functions.
- The high-speed on-chip oscillator must be active while self-programming is in progress. Execute API functions of RFD RL78 Type 11 only while the high-speed on-chip oscillator is active.
- To control the data flash memory, execute API functions of RFD RL78 Type 11 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 RFD RL78 Type 11.

3.3.1.1 R_RFD_Init

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_Init(unit8_t i_u08_cpu_frequency);	
Reentrancy	Non-reentrant	
Parameters (IN)	unit8_t i_u08_cpu_frequency	CPU operating frequency [1 to 32 (MHz)]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end: The frequency is within the allowable range.]
		R_RFD_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 RFD RL78 Type 11.	
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 RFD functions.	

Details of Specifications:

- The execution flag (g_u08_change_interrupt_vector_flag) for R_RFD_ChangeInterruptVector() is initialized to 0x00 (not executed).
- 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”.

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.

  * RFD RL78 Type 11 does not activate or check the high-speed on-chip oscillator.
- For the parameter (i_u08_cpu_frequency), 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_frequency) 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_RFD_SetDataFlashAccessMode

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_SetDataFlashAccessMode (e_rfd_df_access_t i_e_df_access);	
Reentrancy	Non-reentrant	
Parameters (IN)	e_rfd_df_access_t i_e_df_access	Control of access to the data flash memory R_RFD_ENUM_DF_ACCESS_ENABLE: 0x01 [Access to the data flash memory is enabled.] R_RFD_ENUM_DF_ACCESS_DISABLE: 0x00 [Access to the data flash memory is disabled.]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Enables or disables access to the data flash memory according to the parameter setting.	
Preconditions	Execute this function in the non-programmable mode.	
Remarks	-	

Details of Specifications:

- When the parameter (i_e_df_access) is set to R_RFD_ENUM_DF_ACCESS_DISABLE, the DFLEN bit (bit 0 of DFLCTL) is set to 0 (R_RFD_VALUE_U01_DFLEN_DATA_FLASH_ACCESS_DISABLE) to disable access to the data flash memory.
- When the parameter (i_e_df_access) is set to R_RFD_ENUM_DF_ACCESS_ENABLE, the DFLEN bit (bit 0 of DFLCTL) is set to 1 (R_RFD_VALUE_U01_DFLEN_DATA_FLASH_ACCESS_ENABLE) to enable access to the data flash memory.
- Wait for the setup time (setup time: 250nsec). After the wait, the data flash memory can be accessed.

Notes:

- If the value specified by the parameter (i_e_df_access) is neither R_RFD_ENUM_DF_ACCESS_DISABLE nor R_RFD_ENUM_DF_ACCESS_ENABLE, the DFLEN bit (bit 0 of DFLCTL) is set to 0 (R_RFD_VALUE_U01_DFLEN_DATA_FLASH_ACCESS_DISABLE) to disable access to the data flash memory.
- If this function is executed while the sequencer is not in the non-programmable mode, the subsequent operation is indeterminate.

3.3.1.3 R_RFD_ChangeInterruptVector

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_ChangeInterruptVector (uint32_t i_u32_interrupt_vector_addr);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint32_t i_u32_interrupt_vector_addr	Destination address of interrupt branch [RAM address]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFD_ENUM_RET_ERR_MODE_MISMATCHED: 0x11 [Mode mismatch error]
Description	Changes the branch destination address for all interrupts to the RAM address specified by the parameter.	
Preconditions	Execute this function in the non-programmable mode.	
Remarks	—	

Details of Specifications:

- The hook function R_RFD_HOOK_EnterCriticalSection() is called to save the current interrupt disabled (DI) or enabled (EI) state and disable interrupts.
- The branch destination address for all interrupts is changed to the RAM address specified by the parameter (i_u32_interrupt_vector_addr).
  - The specific sequence is executed to set the FWEDIS bit (bit 3) of the FLPMC register to 0 (FLPMC = 0x00).
  - The value of the parameter (i_u32_interrupt_vector_addr) is set in the interrupt vector change registers (FLSIVC0 and FLSIVC1).
  - The interrupt address control register is appropriately set up so that execution branches to the specified RAM address (VECTCTRL = 0x01 [R_RFD_VALUE_U08_VECTCTRL_ON]).
- The hook function R_RFD_HOOK_ExitCriticalSection() is called to restore the interrupt disabled (DI) or enabled (EI) state.
- The execution flag (g_u08_change_interrupt_vector_flag) of this function is modified to indicate the executed state (R_RFD_VALUE_U08_SET_FWEDIS_FLAG_ON: 0x55).

Notes:

- If the value specified by the parameter (i_u32_interrupt_vector_addr) is not a RAM address, the subsequent operation is indeterminate.
- When this function is executed, interrupts need to be disabled in the period between the calls of the hook functions R_RFD_HOOK_EnterCriticalSection() and R_RFD_HOOK_ExitCriticalSection(). If interrupts are enabled and an interrupt occurs in this period, the subsequent operation is indeterminate.
- If this function is executed while the sequencer is not in the non-programmable mode, the subsequent operation is indeterminate.

- - Example of defining a interrupt function to be placed on the RAM
- The argument of the R_RFD_ChangeInterruptVector function is the address of the interrupt function placed on the RAM. Also, when the R_RFD_ChangeInterruptVector function is used, the destination of all interrupt function is changed to the specified address on the RAM. After the R_RFD_ChangeInterruptVector function is executed, all interrupts will branch to the RAM address specified by this function, instead of to the address specified in the interrupt vector table, even if an interrupt occurs. Therefore, if there are multiple interrupt sources and different processing is desired for each, it is necessary to identify the interrupt sources within the interrupt function.
- The interrupt factor can be determined by referring to the interrupt request flag when an interrupt occurs on the RAM. However, the interrupt request flag is not cleared automatically, so the interrupt request flag should be cleared (set to 0) after determination.
- Here are examples of prototype declarations, function definitions, and function calls for each compiler for interrupt function to be placed on the RAM.

- CC-RL compiler

  Prototype: #pragma interrupt Xxxxx;
  __far void Xxxxx(void);
  Function definition: __far void Xxxxx(void){}
  Function call: R_RFD_ChangeInterruptVector((uint32_t)((void (__far *)(void)) Xxxxx));

- IAR compiler

  Prototype: __interrupt void Xxxxx(void);
  Function definition: __interrupt void Xxxxx(void){}
  Function call (V4.21 or later): R_RFD_ChangeInterruptVector((uint32_t)((__far unsigned char *) &Xxxxx));
  (V5.10 or later): R_RFD_ChangeInterruptVector((uint32_t)((void (__far_func *)(void)) Xxxxx));

  Note: By placing interrupt function on the RAM, the following “warning” will be output, but it has been confirmed that there is no problem with these. In addition, the “warning” can be suppressed by selecting “C/C++Compiler-Extra Options” from the IAR Embedded Workbench project options and setting “--diag_suppress=Ta030,Be006” in the “Command line options” input field, but it is recommended that this be set when development is complete, as other “warnings” may not be output.

  Examples of “warning”:

  [Ta030]: Note that this function's segment ‘SMP_CF’ must be placed in near code memory.
  [Be006]: possible conflict for segment/section “SMP_CF”.

- LLVM compiler

  Prototype: __far void Xxxxx(void) __attribute__ ((interrupt));
  Function definition: __far void Xxxxx(void){}
  Function call: R_RFD_ChangeInterruptVector((uint32_t)((void (__far *)(void)) Xxxxx));

Note: “Xxxxx” is the name of the interrupt function.

3.3.1.4 R_RFD_RestoreInterruptVector

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_RestoreInterruptVector (void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFD_ENUM_RET_ERR_MODE_MISMATCHED: 0x11 [Mode mismatch error]
Description	Changes the branch destination address for interrupts that was changed to a RAM address back to the normal interrupt vector addresses.	
Preconditions	Execute this function in the non-programmable mode.	
Remarks	—	

Details of Specifications:

- The hook function R_RFD_HOOK_EnterCriticalSection() is called to save the current interrupt disabled (DI) or enabled (EI) state and disable interrupts.
- The branch destination address for interrupts that was changed to a RAM address by the R_RFD_ChangeInterruptVector() function is changed back to the original locations — that is, the addresses specified by the interrupt table in ROM.
  - The specific sequence is executed to set the FWEDIS bit (bit 3) of the FLPMC register to 1 (FLPMC = 0x08).
  - The interrupt address control register is appropriately set up so that execution branches to the addresses specified by the interrupt vector table in ROM (VECTCTRL = 0x00 [R_RFD_VALUE_U08_VECTCTRL_OFF]).
- The hook function R_RFD_HOOK_ExitCriticalSection() is called to restore the interrupt disabled (DI) or enabled (EI) state.
- The execution flag (g_u08_change_interrupt_vector_flag) of this function is modified to indicate the unexecuted state (R_RFD_VALUE_U08_SET_FWEDIS_FLAG_OFF: 0x00).

Notes:

- When this function is executed, interrupts need to be disabled in the period between the calls of the hook functions R_RFD_HOOK_EnterCriticalSection() and R_RFD_HOOK_ExitCriticalSection(). If interrupts are enabled and an interrupt occurs in this period, the subsequent operation is indeterminate.
- If this function is executed while the sequencer is not in the non-programmable mode, the subsequent operation is indeterminate.

3.3.1.5 R_RFD_ForceReset

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_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.

3.3.1.6 R_RFD_GetSecurityAndBootFlags

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_GetSecurityAndBootFlags (uint16_t __near * onp_u16_security_and_boot_flags);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	uint16_t __near * onp_u16_security_and_boot_flags	Pointer to the variable for storing the information on security flags (protection flags) and boot area switching flag
Return Value	N/A	
Description	Acquires the information on the security flags (protection flags) and boot area switching flag.	
Preconditions	Use this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	-	

Details of Specifications:

- The value of the FLSEC register (16 bits) that shows the information on the security flags (protection flags) and boot area switching flag is read and stored in the variable pointed to by the parameter (onp_u16_security_and_boot_flags).

Notes:

- Security flag and boot area switching flag information to be acquired (bits 15 to 0 of the FLSEC register):
  Bits 15 to 13: -
  Bit 12 (WRPR): Write-disabled flag
  Bit 11: -
  Bit 10 (SEPR): Block erase-disabled flag
  Bit 9 (BTPR): Boot area rewrite-disabled flag
  Bit 8 (BTFLG): Boot area switching flag
  Bits 7 to 4: -
  Bits 3 (SWPR): flash read protection flag
  Bits 2 to 0: -
- For the information on the BTFLG bit (bit 8) acquired by this function, note that a value of 0 indicates boot cluster 1 and a value of 1 indicates boot cluster 0.
- Correct values may not be acquired if this function is executed while command execution is in progress in the code/data flash area sequencer or extra area sequencer.

3.3.1.7 R_RFD_GetFSW

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_GetFSW (uint16_t __near * onp_u16_start_block_number, uint16_t __near * onp_u16_end_block_number, e_rfd_fsw_mode_t __near * onp_e_fsw_mode, e_rfd_protect_t __near * onp_e_protect_flag);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	uint16_t __near * onp_u16_start_block_number	Pointer to the variable for storing the start block number
	uint16_t __near * onp_u16_end_block_number	Pointer to the variable for storing the end block number +1
	e_rfd_fsw_mode_t * onp_e_fsw_mode	Pointer to the variable for storing the information on the flash shield window area
	e_rfd_protect_t * onp_e_protect_flag	Pointer to the variable for storing the information on the protection flag
Return Value	N/A	
Description	Acquires the range of the flash shield window, the flash shield window area, and the protection flag value.	
Preconditions	Use this function while command execution is not in progress in the code/data flash memory area sequencer or extra area sequencer.	
Remarks	—	

Details of Specifications:

- The values of the FLFSWS register (16 bits) and FLFSWE register (16 bits) that indicate the start block and end blocks + 1 of the flash shield window, flash shield window area, and protection flag are read and stored in the variables pointed to by the corresponding parameters.
  - — Values (output) of the variables pointed to by the parameters:

    *onp_u16_start_block_number: Start block
    (Setting in bits 8 to 0 of FLFSWS. Bits 15 to 9 are masked with 0.)

    *onp_u16_end_block_number: End block + 1
    (Setting in bits 8 to 0 of FLFSWE. Bits 15 to 9 are masked with 0.)

    *onp_e_fsw_mode: Output value indicating the flash shield window area
    (Bit 15 (FSWC) of FLFSWE)

    1: R_RFD_ENUM_FSW_MODE_OUTSIDE (Outside shield area)

    0: R_RFD_ENUM_FSW_MODE_INSIDE (Inside shield area)

    *onp_e_protect_flag: Output value indicating the protection flag
    (Bit 15 (FSPR) of FLFSWS)

    1: R_RFD_ENUM_PROTECT_OFF (Shield window protection is disabled.)

    0: R_RFD_ENUM_PROTECT_ON (Shield window protection is enabled.)

Notes:

- If this function is executed in the initial state of the device, onp_u16_start_block_number = 511 and onp_u16_end_block_number = 511 are acquired.
- Correct values may not be acquired if this function is executed while command execution is in progress in the code/data flash memory area sequencer or extra area sequencer.

3.3.2 Specifications of API Functions for Code Flash Memory Control

This section describes the API functions for code flash memory control in RFD RL78 Type 11.

3.3.2.1 R_RFD_EraseCodeFlashReq

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_EraseCodeFlashReq (uint16_t i_u16_block_number);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint16_t i_u16_block_number	Target block number for erasure [0 to 511] Example: For RL78/L23, 0 to 255 (512 Kbytes max.)
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Activates the code/data flash area sequencer and begins the erasure of the code flash memory (one block).	
Preconditions	Use this function in the code flash memory programming mode while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	Execute the R_RFD_CheckCFSeqEndStep1() function after this function.	

Details of Specifications:

- The code/data flash area is selected as the target area of reprogramming.
  FLARS register = R_RFD_VALUE_U08_FLARS_USER_AREA: 0x00 (EXA (bit 0) = 0)
- The code/data flash area sequencer is activated and the address of one block (2 Kbytes) to be erased in the code flash memory is set in the sequencer.
  - The start address and end address of the target block (2 Kbytes) in the code flash memory are calculated from the block number for erasure specified by the parameter (i_u16_block_number) and set in the FLAPL and FLAPH registers and the FLSEDL and FLSEDH registers, respectively.
- R_RFD_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 9 bits of the 16-bit parameter (i_u16_block_number) are used; the upper 7 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 code flash memory programming mode, the subsequent operation is indeterminate.
- If this function is executed while command execution is in progress in the code/data flash area sequencer or extra area sequencer, the subsequent operation is indeterminate.
- The size of the code flash memory is different for each device. About the maximum block number which can be used, refer to the user's manual of the target RL78 microcontroller.

3.3.2.2 R_RFD_WriteCodeFlashReq

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_WriteCodeFlashReq (uint32_t i_u32_start_addr, uint8_t __near * inp_u08_write_data);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint32_t i_u32_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 code/data flash area sequencer and begins the programming of the code flash memory (4 bytes).	
Preconditions	Use this function in the code flash memory programming mode while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	Execute the R_RFD_CheckCFSeqEndStep1() function after this function.	

Details of Specifications:

- The code/data flash area is selected as the target area of reprogramming.
  FLARS register = R_RFD_VALUE_U08_FLARS_USER_AREA: 0x00 (EXA (bit 0) = 0)
- The code/data flash area 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_u32_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 FLWL and FLWH registers.
- R_RFD_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 lower 24 bits of the 32-bit parameter i_u32_start_addr are used with the upper 8 bits masked with 0x00. 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 code flash memory programming mode, the subsequent operation is indeterminate.
- If this function is executed while command execution is in progress in the code/data flash area sequencer or extra area sequencer, the subsequent operation is indeterminate.

3.3.2.3 R_RFD_BlankCheckCodeFlashReq

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_BlankCheckCodeFlashReq (uint16_t i_u16_block_number);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint16_t i_u16_block_number	Target block number for blank check [0 to 511] Example: For RL78/L23, 0 to 255 (512 Kbytes max.)
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Activates the code/data flash area sequencer and begins the blank check of the code flash memory (one block).	
Preconditions	Use this function in the code flash memory programming mode while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	Execute the R_RFD_CheckCFSeqEndStep1() function after this function.	

Details of Specifications:

- The code/data flash area is selected as the target area of reprogramming.
  FLARS register = R_RFD_VALUE_U08_FLARS_USER_AREA: 0x00 (EXA (bit 0) = 0)
- The code/data flash area sequencer is activated and the address of one block (2 Kbytes) to be checked for blanks in the code flash memory is set in the sequencer.
  - The start address and end address of the target block (2 Kbytes) in the code flash memory are calculated from the block number for blank check specified by the parameter (i_u16_block_number) and set in the FLAPL and FLAPH registers and the FLSEDL and FLSEDH registers, respectively.
- R_RFD_VALUE_U08_FSSQ_BLANKCHECK_CF = 0x83 is set in the FSSQ register to start the blank check. (SQST (bit 7) = 1, MDCH (bit 3) = 0, SQMD (bits 2 to 0) = 3 (0b011), and the other bits are set to 0.)

Notes:

- The lower 9 bits of the 16-bit parameter (i_u16_block_number) are used; the upper 7 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 code flash memory programming mode, the subsequent operation is indeterminate.
- If this function is executed while command execution is in progress in the code/data flash area sequencer or extra area sequencer, the subsequent operation is indeterminate.
- The size of the code flash memory is different for each device. About the maximum block number which can be used, refer to the user's manual of the target RL78 microcontroller.

3.3.2.4 R_RFD_SetCFProgrammingMode

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_SetCFProgrammingMode(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFD_ENUM_RET_ERR_MODE_MISMATCHED: 0x11 [Mode mismatch error] (The flash memory sequencer is not placed in the specified mode.)
Description	The mode of a flash memory sequencer transfers to code flash programming mode and then sets the specified CPU operating frequency in the flash memory sequencer.	
Preconditions	Execute this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	-	

Details of Specifications:

- The hook function R_RFD_HOOK_EnterCFCriticalSection() is called to save the current interrupt disabled (DI) or enabled (EI) state and disable interrupts.
- R_RFD_VALUE_U08_FLPMC_MODE_CODE_FLASH_PROGRAMMING is set to the FLPMC register value (l_u08_set_flpmc_value). And the mode of a flash memory sequencer transfers to code flash programming mode.
- The hook function R_RFD_HOOK_ExitCFCriticalSection() is called to restore the interrupt disabled (DI) or enabled (EI) state.
- Set the “g_u08_fset_cpu_frequency” set in the R_RFD_Init function to the FSSET register.

Notes:

- When this function is executed, interrupts need to be disabled in the period between the calls of the hook functions R_RFD_HOOK_EnterCFCriticalSection() and R_RFD_HOOK_ExitCFCriticalSection(). If interrupts are enabled and an interrupt occurs in this period, the subsequent operation is indeterminate.
- If this function is executed before the R_RFD_Init function, the reprogrammed data are not guaranteed even after the reprogramming processing by the RFD is completed. To use RFD RL78 Type 11, be sure to execute the R_RFD_Init() function once before starting the use of RFD functions.
- When transitioning to code flash memory programming mode, please transition from non-programmable mode.

3.3.2.5 R_RFD_SetCFNonProgrammableMode

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_SetCFNonProgrammableMode(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFD_ENUM_RET_ERR_MODE_MISMATCHED: 0x11 [Mode mismatch error] (The flash memory sequencer is not placed in the specified mode.)
Description	The mode of a flash memory sequencer transfers from code flash programming mode to non-programmable mode.	
Preconditions	Execute this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	-	

Details of Specifications:

- The hook function R_RFD_HOOK_EnterCFCriticalSection() is called to save the current interrupt disabled (DI) or enabled (EI) state and disable interrupts.
- The value of Non-programmable mode is set to the FLPMC register value (l_u08_set_flpmc_value), the mode of a flash memory sequencer transfers from code flash programming mode to non-programmable mode.
- When the mode is shifted, 10 µs wait is performed.
- The hook function R_RFD_HOOK_ExitCFCriticalSection() is called to restore the interrupt disabled (DI) or enabled (EI) state.

Notes:

- When this function is executed, interrupts need to be disabled in the period between the calls of the hook functions R_RFD_HOOK_EnterCFCriticalSection() and R_RFD_HOOK_ExitCFCriticalSection(). If interrupts are enabled and an interrupt occurs in this period, the subsequent operation is indeterminate.
- FLPMC register is set up according to the value of the execution flag(g_u08_change_interrupt_vector_flag) of the R_RFD_ChangeInterruptVector() function as follows.
  - Execution flag of R_RFD_ChangeInterruptVector() = 0x00 (not executed)
    FLPMC = 0x08 (FWEDIS (bit 3 of FLPMC) = 1)
    (Execution after an interrupt branch according to the interrupt vector table in ROM.)
  - Execution flag of R_RFD_ChangeInterruptVector() = 0x55 (executed)
    FLPMC = 0x00 (FWEDIS (bit 3 of FLPMC) = 0)
    (Execution after an interrupt branch to the specified address in RAM.)
- If this function is executed before the R_RFD_Init function, the reprogrammed data are not guaranteed even after the reprogramming processing by the RFD is completed. To use RFD RL78 Type 11, be sure to execute the R_RFD_Init() function once before starting the use of RFD functions.

3.3.2.6 R_RFD_CheckCFProgrammingMode

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_CheckCFProgrammingMode(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFD_ENUM_RET_ERR_MODE_MISMATCHED: 0x11 [Mode mismatch error]
Description	Check whether flash memory control mode is code flash programming mode.	
Preconditions	Execute this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	-	

Details of Specifications:

- The value of the FLPMC register is read to check whether flash memory control mode is code flash programming mode.
  - Code flash memory programming mode: 0x02 (FLSPM (bit 1 of FLPMC) = 1)

Notes:

- If the control mode of the flash memory sequencer was specified by a function other than R_RFD_SetCFProgrammingMode(), this function may not be executed correctly.
- If this function is executed during command execution in the code/data flash area sequencer or the extra area sequencer, the subsequent operation is indeterminate.

3.3.2.7 R_RFD_CheckCFNonProgrammableMode

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_CheckCFNonProgrammableMode(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFD_ENUM_RET_ERR_MODE_MISMATCHED: 0x11 [Mode mismatch error]
Description	Check whether flash memory control mode is non-programmable mode.	
Preconditions	Execute this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	-	

Details of Specifications:

- The value of the FLPMC register is read to check whether flash memory control mode is non-programmable mode.
  - Non-programmable mode:
    FLPMC = 0x08 [FWEDIS (bit 3 of FLPMC) = 1] or 0x00 [FWEDIS (bit 3 of FLPMC) = 0]

Notes:

- If the control mode of the flash memory sequencer was specified by a function other than R_RFD_SetCFNonProgrammableMode(), this function may not be executed correctly.
- If this function is executed during command execution in the code/data flash area sequencer or the extra area sequencer, the subsequent operation is indeterminate.

3.3.2.8 R_RFD_CheckCFSeqEndStep1

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_CheckCFSeqEndStep1(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFD_ENUM_RET_STS_BUSY: 0x01 [Sequencer command execution is in progress.]
Description	Checks if the operation of the activated code/data flash area sequencer has been completed.	
Preconditions	Execute this command after starting the command for activating the code/data flash area sequencer.	
Remarks	Execute this function again if R_RFD_ENUM_RET_STS_BUSY is returned. After confirming that R_RFD_ENUM_RET_STS_OK has been returned from this function, execute the R_RFD_CheckCFSeqEndStep2() function.	

Details of Specifications:

- Whether the operation of the activated code/data flash area sequencer has been completed (SQEND (bit 6 of FSASTH) = 1) is checked.
- When the operation of the code/data flash memory area sequencer has been completed, 0x00 value is set in the flash memory sequencer control register (FSSQ), and SQST [bit7] is cleared. And R_RFD_ENUM_RET_STS_OK is returned. If the operation has not been completed, R_RFD_ENUM_RET_STS_BUSY is returned.

Notes:

- Execute this function again if R_RFD_ENUM_RET_STS_BUSY is returned.
- If execution of this function is attempted before the command for activating the code/data flash area sequencer is started, this function is not executed correctly.
- After confirming that R_RFD_ENUM_RET_STS_OK has been returned from this function, execute the R_RFD_CheckCFSeqEndStep2() function.

3.3.2.9 R_RFD_CheckCFSeqEndStep2

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_CheckCFSeqEndStep2(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end: Sequencer operation has been completed.]
		R_RFD_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_RFD_ENUM_RET_STS_OK has been returned from the R_RFD_CheckCFSeqEndStep1() function.	
Remarks	Execute this function again if R_RFD_ENUM_RET_STS_BUSY is returned.	

Details of Specifications:

- After 0x00 value is set in a flash memory sequencer control register (FSSQ), check whether all operations of the code/data flash memory area sequencer command have been finished [SQEND(bit6 of FSASTH) = 0].
- When the command execution in the code/data flash area sequencer has been completed, R_RFD_ENUM_RET_STS_OK is returned.
  If the operation has not been completed, R_RFD_ENUM_RET_STS_BUSY is returned.

Notes:

- Execute this function again if R_RFD_ENUM_RET_STS_BUSY is returned.
- If execution of this function is attempted before R_RFD_ENUM_RET_STS_OK has been confirmed by the R_RFD_CheckCFSeqEndStep1() function, this function is not executed correctly.

3.3.2.10 R_RFD_GetCFSeqErrorStatus

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_GetCFSeqErrorStatus (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 code/data flash area sequencer.	
Preconditions	Execute this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	-	

Details of Specifications:

- The FSASTL register (8 bits) is read and the value of bits 5 to 0 is stored in the variable pointed to by the parameter (onp_u08_error_status).
  Note: Bits 7, bit 6, and bit 2 set the fixed value 0.
  Error information to be acquired (five bits of the FSASTL register: bits 5 to 0):
  Bit 5: Extra area sequencer error
  Bit 4: Code/data flash area sequencer error
  Bit 3: Blank check command error
  Bit 1: Write command error
  Bit 0: Erase command error

Note:

- Correct values may not be acquired if this function is executed while command execution is in progress in the code/data flash area sequencer or extra area sequencer.

3.3.2.11 R_RFD_ClearCFSeqRegister

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_ClearCFSeqRegister(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Clears the registers for controlling the code/data flash area sequencer and extra area sequencer.	
Preconditions	Use this function in the code flash memory programming mode. Use this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	Execute this function after execution of the R_RFD_CheckCFSeqEndStep2() function.	

Details of Specifications:

- The flash registers initialization register (FLRST) is set to 0x01 and then cleared to 0x00 to clear the following registers.
  - Target registers for controlling the code/data flash area sequencer or extra area sequencer: FLAPH, FLAPL, FLSEDH, FLSEDL, FLWH, FLWL, FLARS, FSSQ, and FSSE

Notes:

- This function does not clear the information on errors generated during command execution in the flash memory sequencer (the information in the FSASTL register).
- If this function is executed while operation is in progress in the code/data flash area sequencer or extra area sequencer, the subsequent operation is indeterminate.
- If this function is executed while the sequencer is not in the code flash memory programming mode, the subsequent operation is indeterminate.

3.3.2.12 R_RFD_ForceStopCFSeq

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_ForceStopCFSeq(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Forcibly stops the operation of the code/data flash area sequencer.	
Preconditions	Use this function after starting the command for activating the code/data flash area sequencer (while command execution is in progress or the sequencer is operating). Use this function before the R_RFD_CheckCFSeqEndStep1() function returns R_RFD_ENUM_RET_STS_OK (before the sequencer operation is completed).	
Remarks	Execute the R_RFD_CheckCFSeqEndStep1() function after this function.	

Details of Specifications:

- While the code/data flash area sequencer is executing the blank check command or erase command, the FSSTP bit (bit 6) of the FSSQ register is set to 1 to forcibly stop the code/data flash area sequencer.

Notes:

- Use this function only when forced stop of command execution is necessary **in an emergency situation**.
- Execute this function only while the code/data flash area sequencer is executing the blank check command, or erase command.
- When this function is executed during execution of the erase command, the target area should be erased again.
- Do not execute this function while the code/data flash area sequencer is executing a command other than the blank check, or erase command or while the extra area sequencer is operating. Otherwise, the subsequent operation is indeterminate. (If this function is executed during the write command execution, undefined data are written.)
- This function cannot be used while the command execution state is undetermined.
- The command that has been forcibly stopped by this function may generate an error. In this case, do not refer to the error flags because the command execution may have not been completed.

3.3.2.13 r_rfd_cf_wait_count

Information:

Syntax	R_RFD_FAR_FUNC void r_rfd_cf_wait_count(uint8_t i_u08_count);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint8_t i_u08_count	Wait time (Time count in units of 1 µs: A value from 1 to 255 can be specified.)
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Executes a software loop to wait for the time specified by the parameter (time count in units of 1 µs).	
Preconditions	-	
Remarks	-	

Details of Specifications:

- A value of 1 is added to the g_u08_cpu_frequency value (CPU operating frequency – 1) to obtain the CPU operating frequency.
- The number of software loop repetitions for the specified wait time (time count in units of 1 µs) is calculated and the software loops are executed.

  Number of software loop repetitions for the specified wait time (time count in units of 1 µs)
  = ((frequency [MHz]) × (specified count [µs]) / (loop execution cycles: 8 [cycles])) + 1

  Example: Frequency value = 32 [MHz] and time count = 10 [µs]

  Number of software loop repetitions for the wait time (time count in units of 1 µs)
  = (32 [MHz] × 10 [µs] / 8 [cycles]) + 1

  (1 is added so that the result after rounding does not become smaller than the wait time.)

  = 41 [repetitions]

  Execution time of this function = 1/32 [MHz] × 8 [cycles] × 41 [repetitions] = 10.25 [µs]

Note:

- The range of wait time is from 1 µs to 255 µs, which does not include the overhead of the processing other than the loop processing.

3.3.3 Specifications of API Functions for Data Flash Memory Control

This section describes the API functions for data flash memory control in RFD RL78 Type 11.

3.3.3.1 R_RFD_EraseDataFlashReq

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_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 63] Example: For RL78/L23, 0 to 31 (8 Kbytes max.)
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Activates the code/data flash area sequencer and begins the erasure of the data flash memory (one block).	
Preconditions	Use this function while access to the data flash memory is enabled (DFLEN = 1). Use this function in the data flash memory programming mode while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	Execute the R_RFD_CheckDFSeqEndStep1() function after this function.	

Details of Specifications:

- The code/data flash area is selected as the target area of reprogramming.
  FLARS register = R_RFD_VALUE_U08_FLARS_USER_AREA: 0x00 (EXA (bit 0) = 0)
- The code/data flash area sequencer is activated and the address of one block (256 bytes) to be erased in the data flash memory is set in the sequencer.
  - The start address and end address of the target block (256 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_RFD_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 6 bits of the 8-bit parameter (i_u08_block_number) are used; the upper 2 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 access to the data flash memory is disabled, the subsequent operation is indeterminate.
- If this function is executed while the sequencer is not in the data flash memory programming mode, the subsequent operation is indeterminate.
- If this function is executed while command execution is in progress in the code/data flash area sequencer or extra area sequencer, the subsequent operation is indeterminate.
- The size of the code flash memory is different for each device. About the maximum block number which can be used, refer to the user's manual of the target RL78 microcontroller.

3.3.3.2 R_RFD_WriteDataFlashReq

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_WriteDataFlashReq (uint32_t i_u32_start_addr, uint8_t __near * inp_u08_write_data);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint32_t i_u32_start_addr	Target start address for programming [Address in the data 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 1 byte]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Activates the code/data flash area sequencer and begins the programming of the data flash memory (1 byte).	
Preconditions	Use this function while access to the data flash memory is enabled (DFLEN = 1). Use this function in the data flash memory programming mode while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	Execute the R_RFD_CheckDFSeqEndStep1() function after this function.	

Details of Specifications:

- The code/data flash area is selected as the target area of reprogramming.
  FLARS register = R_RFD_VALUE_U08_FLARS_USER_AREA: 0x00 (EXA (bit 0) = 0)
- The code/data flash area sequencer is activated, and the programming start address in the data flash memory and the write data (1 byte) are set in the sequencer.
  - The target start address in the data flash memory specified by the parameter i_u32_start_addr is set in the FLAPL and FLAPH registers.
  - The 1-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 lower 8 bits of the FLWL register.
- R_RFD_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 lower 24 bits of the 32-bit parameter i_u32_start_addr are used with the upper 8 bits masked with 0x00. The start address must be within the space of the data flash memory implemented in the device. If the specified address is outside the allowable space, the subsequent operation is indeterminate.
- If this function is executed while access to the data flash memory is disabled, the subsequent operation is indeterminate.
- If this function is executed while the sequencer is not in the data flash memory programming mode, the subsequent operation is indeterminate.
- If this function is executed while command execution is in progress in the code/data flash area sequencer or extra area sequencer, the subsequent operation is indeterminate.

3.3.3.3 R_RFD_BlankCheckDataFlashReq

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_BlankCheckDataFlashReq (uint32_t i_u32_start_addr, uint16_t i_u16_blankcheck_length);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint32_t i_u32_start_addr	Target start address for blank check [Address in the data flash area]
	uint16_t i_u16_blankcheck_length	Target data length for blank check
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Activates the code/data flash area sequencer and begins the blank check of the data flash memory (Specified number of bytes).	
Preconditions	Use this function while access to the data flash memory is enabled (DFLEN = 1). Use this function in the data flash memory programming mode while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	Execute the R_RFD_CheckDFSeqEndStep1() function after this function.	

Details of Specifications:

- The code/data flash area is selected as the target area of reprogramming.
  FLARS register = R_RFD_VALUE_U08_FLARS_USER_AREA: 0x00 (EXA (bit 0) = 0)
- The code/data flash area sequencer is activated and set the start and end addresses to be blank check for data flash memory.
  - The start address of the data flash memory for the blank check of an argument (i_u32_start_addr) is set in the FLAPL and FLAPH register.
  - The end address (start address + specified byte size) of the data flash memory is calculated, and it sets in the FLSEDL and FLSEDH register.
- R_RFD_VALUE_U08_FSSQ_BLANKCHECK_DF = 0x8B is set in the FSSQ register to start the blank check. (SQST (bit 7) = 1, MDCH (bit 3) = 1, SQMD (bits 2 to 0) = 3 (0b011), and the other bits are set to 0.)

Notes:

- It cannot be set to straddle blocks. Set within the range of 1 block.
- The lower 24 bits of the 32-bit parameter i_u32_start_addr are used with the upper 8 bits masked with 0x00. The start address must be within the space of the data flash memory implemented in the device. If the specified address is outside the allowable space, the subsequent operation is indeterminate.
- If this function is executed while access to the data flash memory is disabled, the subsequent operation is indeterminate.
- If this function is executed while the sequencer is not in the data flash memory programming mode, the subsequent operation is indeterminate.
- If this function is executed while command execution is in progress in the code/data flash area sequencer or extra area sequencer, the subsequent operation is indeterminate.

3.3.3.4 R_RFD_SetDFProgrammingMode

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_SetDFProgrammingMode(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFD_ENUM_RET_ERR_MODE_MISMATCHED: 0x11 [Mode mismatch error] (The flash memory sequencer is not placed in the specified mode.)
Description	The mode of a flash memory sequencer transfers to data flash programming mode and then sets the specified CPU operating frequency in the flash memory sequencer.	
Preconditions	Execute this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	-	

Details of Specifications:

- The hook function R_RFD_HOOK_EnterDFCriticalSection() is called to save the current interrupt disabled (DI) or enabled (EI) state and disable interrupts.
- R_RFD_VALUE_U08_FLPMC_MODE_DATA_FLASH_PROGRAMMING is set to the FLPMC register value (l_u08_set_flpmc_value). And the mode of a flash memory sequencer transfers to data flash programming mode.
- The hook function R_RFD_HOOK_ExitDFCriticalSection() is called to restore the interrupt disabled (DI) or enabled (EI) state.
- Set the “g_u08_fset_cpu_frequency” set in the R_RFD_Init function to the FSSET register.

Notes:

- When this function is executed, interrupts need to be disabled in the period between the calls of the hook functions R_RFD_HOOK_EnterDFCriticalSection() and R_RFD_HOOK_ExitDFCriticalSection(). If interrupts are enabled and an interrupt occurs in this period, the subsequent operation is indeterminate.
- If this function is executed before the R_RFD_Init function, the reprogrammed data are not guaranteed even after the reprogramming processing by the RFD is completed. To use RFD RL78 Type 11, be sure to execute the R_RFD_Init() function once before starting the use of RFD functions.
- When transitioning to data flash memory programming mode, please transition from non-programmable mode.

3.3.3.5 R_RFD_SetDFNonProgrammableMode

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_SetDFNonProgrammableMode(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFD_ENUM_RET_ERR_MODE_MISMATCHED: 0x11 [Mode mismatch error] (The flash memory sequencer is not placed in the specified mode.)
Description	The mode of a flash memory sequencer transfers from data flash programming mode to non-programmable mode.	
Preconditions	Execute this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	-	

Details of Specifications:

- The hook function R_RFD_HOOK_EnterCFCriticalSection() is called to save the current interrupt disabled (DI) or enabled (EI) state and disable interrupts.
- The value of Non-programmable mode is set to the FLPMC register value (l_u08_set_flpmc_value), the mode of a flash memory sequencer transfers from data flash programming mode to non-programmable mode.
- When the mode is shifted, 10 µs wait is performed.
- The hook function R_RFD_HOOK_ExitCFCriticalSection() is called to restore the interrupt disabled (DI) or enabled (EI) state.

Notes:

- When this function is executed, interrupts need to be disabled in the period between the calls of the hook functions R_RFD_HOOK_EnterCFCriticalSection() and R_RFD_HOOK_ExitCFCriticalSection(). If interrupts are enabled and an interrupt occurs in this period, the subsequent operation is indeterminate.
- FLPMC register is set up according to the value of the execution flag(g_u08_change_interrupt_vector_flag) of the R_RFD_ChangeInterruptVector() function as follows.
  - Execution flag of R_RFD_ChangeInterruptVector() = 0x00 (not executed)
    FLPMC = 0x08 (FWEDIS (bit 3 of FLPMC) = 1)
    (Execution after an interrupt branch according to the interrupt vector table in ROM.)
  - Execution flag of R_RFD_ChangeInterruptVector() = 0x55 (executed)
    FLPMC = 0x00 (FWEDIS (bit 3 of FLPMC) = 0)
    (Execution after an interrupt branch to the specified address in RAM.)
- If this function is executed before the R_RFD_Init function, the reprogrammed data are not guaranteed even after the reprogramming processing by the RFD is completed. To use RFD RL78 Type 11, be sure to execute the R_RFD_Init() function once before starting the use of RFD functions.

3.3.3.6 R_RFD_CheckDFProgrammingMode

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_CheckDFProgrammingMode(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFD_ENUM_RET_ERR_MODE_MISMATCHED: 0x11 [Mode mismatch error]
Description	Check whether flash memory control mode is data flash programming mode.	
Preconditions	Execute this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	-	

Details of Specifications:

- The value of the FLPMC register is read to check whether flash memory control mode is data flash programming mode.
  - Data flash memory programming mode: 0x10 (EEEMD (bit 4 of FLPMC) = 1)

Notes:

- If the control mode of the flash memory sequencer was specified by a function other than R_RFD_SetDFProgrammingMode(), this function may not be executed correctly.
- If this function is executed during command execution in the code/data flash area sequencer or the extra area sequencer, the subsequent operation is indeterminate.

3.3.3.7 R_RFD_CheckDFNonProgrammableMode

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_CheckDFNonProgrammableMode(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFD_ENUM_RET_ERR_MODE_MISMATCHED: 0x11 [Mode mismatch error]
Description	Check whether flash memory control mode is non-programmable mode.	
Preconditions	Execute this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	-	

Details of Specifications:

- The value of the FLPMC register is read to check whether flash memory control mode is non-programmable mode.
  - Non-programmable mode:
    FLPMC = 0x08 [FWEDIS (bit 3 of FLPMC) = 1] or 0x00 [FWEDIS (bit 3 of FLPMC) = 0]

Notes:

- If the control mode of the flash memory sequencer was specified by a function other than R_RFD_SetDFNonProgrammableMode(), this function may not be executed correctly.
- If this function is executed during command execution in the code/data flash area sequencer or the extra area sequencer, the subsequent operation is indeterminate.

3.3.3.8 R_RFD_CheckDFSeqEndStep1

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_CheckDFSeqEndStep1(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFD_ENUM_RET_STS_BUSY: 0x01 [Sequencer command execution is in progress.]
Description	Checks if the operation of the activated code/data flash area sequencer has been completed.	
Preconditions	Execute this command after starting the command for activating the code/data flash area sequencer.	
Remarks	Execute this function again if R_RFD_ENUM_RET_STS_BUSY is returned. After confirming that R_RFD_ENUM_RET_STS_OK has been returned from this function, execute the R_RFD_CheckDFSeqEndStep2() function.	

Details of Specifications:

- Whether the operation of the activated code/data flash area sequencer has been completed (SQEND (bit 6 of FSASTH) = 1) is checked.
- When the operation of the code/data flash memory area sequencer has been completed, 0x00 value is set in the flash memory sequencer control register (FSSQ), and SQST [bit7] is cleared. And R_RFD_ENUM_RET_STS_OK is returned. If the operation has not been completed, R_RFD_ENUM_RET_STS_BUSY is returned.

Notes:

- Execute this function again if R_RFD_ENUM_RET_STS_BUSY is returned.
- If execution of this function is attempted before the command for activating the code/data flash area sequencer is started, this function is not executed correctly.
- After confirming that R_RFD_ENUM_RET_STS_OK has been returned from this function, execute the R_RFD_CheckDFSeqEndStep2() function.

3.3.3.9 R_RFD_CheckDFSeqEndStep2

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_CheckDFSeqEndStep2(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end: Sequencer operation has been completed.]
		R_RFD_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_RFD_ENUM_RET_STS_OK has been returned from the R_RFD_CheckDFSeqEndStep1() function.	
Remarks	Execute this function again if R_RFD_ENUM_RET_STS_BUSY is returned.	

Details of Specifications:

- After 0x00 value is set in a flash memory sequencer control register (FSSQ), check whether all operations of the code/data flash memory area sequencer command have been finished [SQEND(bit6 of FSASTH) = 0].
- When the command execution in the code/data flash area sequencer has been completed, R_RFD_ENUM_RET_STS_OK is returned.
  If the operation has not been completed, R_RFD_ENUM_RET_STS_BUSY is returned.

Notes:

- Execute this function again if R_RFD_ENUM_RET_STS_BUSY is returned.
- If execution of this function is attempted before R_RFD_ENUM_RET_STS_OK has been confirmed by the R_RFD_CheckDFSeqEndStep1() function, this function is not executed correctly.

3.3.3.10 R_RFD_GetDFSeqErrorStatus

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_GetDFSeqErrorStatus (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 code/data flash area sequencer.	
Preconditions	Execute this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	-	

Details of Specifications:

- The FSASTL register (8 bits) is read and the value of bits 5 to 0 is stored in the variable pointed to by the parameter (onp_u08_error_status).
- Note: Bits 7, bit 6, and bit 2 set the fixed value 0.
  Error information to be acquired (five bits of the FSASTL register: bits 5 to 0):
  Bit 5: Extra area sequencer error
  Bit 4: Code/data flash area sequencer error
  Bit 3: Blank check command error
  Bit 1: Write command error
  Bit 0: Erase command error

Note:

- Correct values may not be acquired if this function is executed while command execution is in progress in the code/data flash area sequencer or extra area sequencer.

3.3.3.11 R_RFD_ClearDFSeqRegister

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_ClearDFSeqRegister(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Clears the registers for controlling the code/data flash area sequencer and extra area sequencer.	
Preconditions	Use this function in the data flash memory programming mode. Use this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	Execute this function after execution of the R_RFD_CheckDFSeqEndStep2() function.	

Details of Specifications:

- The flash registers initialization register (FLRST) is set to 0x01 and then cleared to 0x00 to clear the following registers.
  - Target registers for controlling the code/data flash area sequencer or extra area sequencer: FLAPH, FLAPL, FLSEDH, FLSEDL, FLWH, FLWL, FLARS, FSSQ, and FSSE

Notes:

- This function does not clear the information on errors generated during command execution in the flash memory sequencer (the information in the FSASTL register).
- If this function is executed while operation is in progress in the code/data flash area sequencer or extra area sequencer, the subsequent operation is indeterminate.
- If this function is executed while the sequencer is not in the data flash memory programming mode, the subsequent operation is indeterminate.

3.3.3.12 R_RFD_ForceStopDFSeq

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_ForceStopDFSeq(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Forcibly stops the operation of the code/data flash area sequencer.	
Preconditions	Use this function after starting the command for activating the code/data flash area sequencer (while command execution is in progress or the sequencer is operating). Use this function before the R_RFD_CheckDFSeqEndStep1() function returns R_RFD_ENUM_RET_STS_OK (before the sequencer operation is completed).	
Remarks	Execute the R_RFD_CheckDFSeqEndStep1() function after this function.	

Details of Specifications:

- While the code/data flash area sequencer is executing the blank check command or erase command, the FSSTP bit (bit 6) of the FSSQ register is set to 1 to forcibly stop the code/data flash area sequencer.

Notes:

- Use this function only when forced stop of command execution is necessary **in an emergency situation**.
- Execute this function only while the code/data flash area sequencer is executing the blank check command, or erase command.
- When this function is executed during execution of the erase command, the target area should be erased again.
- Do not execute this function while the code/data flash area sequencer is executing a command other than the blank check, or erase command or while the extra area sequencer is operating. Otherwise, the subsequent operation is indeterminate. (If this function is executed during the write command execution, undefined data are written.)
- This function cannot be used while the command execution state is undetermined.
- The command that has been forcibly stopped by this function may generate an error. In this case, do not refer to the error flags because the command execution may have not been completed.

3.3.3.13 r_rfd_df_wait_count

Information:

Syntax	R_RFD_FAR_FUNC void r_rfd_df_wait_count(uint8_t i_u08_count);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint8_t i_u08_count	Wait time (Time count in units of 1 µs: A value from 1 to 255 can be specified.)
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Executes a software loop to wait for the time specified by the parameter (time count in units of 1 µs).	
Preconditions	-	
Remarks	-	

Details of Specifications:

- A value of 1 is added to the g_u08_cpu_frequency value (CPU operating frequency – 1) to obtain the CPU operating frequency.
- The number of software loop repetitions for the specified wait time (time count in units of 1 µs) is calculated and the software loops are executed.

  Number of software loop repetitions for the specified wait time (time count in units of 1 µs)
  = ((frequency [MHz]) × (specified count [µs]) / (loop execution cycles: 8 [cycles])) + 1

  Example: Frequency value = 32 [MHz] and time count = 10 [µs]

  Number of software loop repetitions for the wait time (time count in units of 1 µs)
  = (32 [MHz] × 10 [µs] / 8 [cycles]) + 1

  (1 is added so that the result after rounding does not become smaller than the wait time.)

  = 41 [repetitions]

  Execution time of this function = 1/32 [MHz] × 8 [cycles] × 41 [repetitions] = 10.25 [µs]

Note:

- The range of wait time is from 1 µs to 255 µs, which does not include the overhead of the processing other than the loop processing.

3.3.4 Specifications of API Functions for Extra Area Control

This section describes the API functions for extra area control in RFD RL78 Type 11.

3.3.4.1 R_RFD_SetExtraEraseProtectReq

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_SetExtraEraseProtectReq(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Activates the extra area sequencer and begins the setting of the block erase-disabled flag.	
Preconditions	Use this function in the code flash memory programming mode while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	Execute the R_RFD_CheckExtraSeqEndStep1() function after this function.	

Details of Specifications:

- The extra area is selected as the target area of reprogramming.
  FLARS register = R_RFD_VALUE_U08_FLARS_EXTRA_AREA: 0x01 (EXA (bit 0) = 1)
- The extra area sequencer is activated and the setting of the block erase-disabled flag is started.
  - The FLSEC register is read and this value is set in the FLWL register with the current value of the BTFLG bit (bit 8) retained and the SEPR bit (bit 10) cleared to 0 (block erasure is disabled). 0xFFFF is set in the FLWH registers.
- R_RFD_VALUE_U08_FSSE_SECURITY_FLAG = 0x87 is set in the FSSE register to start the setting of the flag.
  (ESQST (bit 7) = 1, ESQMD (bits 2 to 0) = 7 (0b111), and the other bits are set to 0.)

Notes:

- If this function is executed while the sequencer is not in the code flash memory programming mode, the subsequent operation is indeterminate.
- If this function is executed while command execution is in progress in the code/data flash area sequencer or extra area sequencer, the subsequent operation is indeterminate.

3.3.4.2 R_RFD_SetExtraWriteProtectReq

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_SetExtraWriteProtectReq(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Activates the extra area sequencer and begins the setting of the write-disabled flag.	
Preconditions	Use this function in the code flash memory programming mode while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	Execute the R_RFD_CheckExtraSeqEndStep1() function after this function.	

Details of Specifications:

- The extra area is selected as the target area of reprogramming.
  FLARS register = R_RFD_VALUE_U08_FLARS_EXTRA_AREA: 0x01 (EXA (bit 0) = 1)
- The extra area sequencer is activated and the setting of the write-disabled flag is started.
  - The FLSEC register is read and this value is set in the FLWL register with the current value of the BTFLG bit (bit 8) retained and the WRPR bit (bit 12) cleared to 0 (programming is disabled). 0xFFFF is set in the FLWH registers.
- R_RFD_VALUE_U08_FSSE_SECURITY_FLAG = 0x87 is set in the FSSE register to start the setting of the flag.
  (ESQST (bit 7) = 1, ESQMD (bits 2 to 0) = 7 (0b111), and the other bits are set to 0.)

Notes:

- If this function is executed while the sequencer is not in the code flash memory programming mode, the subsequent operation is indeterminate.
- If this function is executed while command execution is in progress in the code/data flash area sequencer or extra area sequencer, the subsequent operation is indeterminate.

3.3.4.3 R_RFD_SetExtraBootAreaProtectReq

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_SetExtraBootAreaProtectReq(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Activates the extra area sequencer and begins the setting of the boot area rewrite-disabled flag.	
Preconditions	Use this function in the code flash memory programming mode while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	Execute the R_RFD_CheckExtraSeqEndStep1() function after this function.	

Details of Specifications:

- The extra area is selected as the target area of reprogramming.
  FLARS register = R_RFD_VALUE_U08_FLARS_EXTRA_AREA: 0x01 (EXA (bit 0) = 1)
- The extra area sequencer is activated and the setting of the boot area rewrite-disabled flag is started.
  - The FLSEC register is read and this value is set in the FLWL register with the current value of the BTFLG bit (bit 8) retained and the BTPR bit (bit 9) cleared to 0 (programming is disabled). 0xFFFF is set in the FLWH registers.
- R_RFD_VALUE_U08_FSSE_SECURITY_FLAG = 0x87 is set in the FSSE register to start the setting of the flag.
  (ESQST (bit 7) = 1, ESQMD (bits 2 to 0) = 7 (0b111), and the other bits are set to 0.)

Notes:

- If this function is executed while the sequencer is not in the code flash memory programming mode, the subsequent operation is indeterminate.
- If this function is executed while command execution is in progress in the code/data flash area sequencer or extra area sequencer, the subsequent operation is indeterminate.

3.3.4.4 R_RFD_SetExtraBootAreaReq

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_SetExtraBootAreaReq (e_rfd_boot_cluster_t i_e_boot_cluster);	
Reentrancy	Non-reentrant	
Parameters (IN)	e_rfd_boot_cluster_t i_e_boot_cluster	Boot cluster number (Bank number) R_RFD_ENUM_BOOT_CLUSTER_0: 0x01 [Boot cluster 0 (Bank 0)] R_RFD_ENUM_BOOT_CLUSTER_1: 0x00 [Boot cluster 1 (Bank 1)]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Activates the extra area sequencer and begins the setting of the boot area switching flag.	
Preconditions	Use this function in the code flash memory programming mode while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	Execute the R_RFD_CheckExtraSeqEndStep1() function after this function.	

Details of Specifications:

- This function specifies that the boot swapping is executed only after reset instead of immediately after the setting of the BTFLG.
  - The FSSET and FLSEC registers are read.

  [For use in boot swapping]
  - Only when the TMSPMD bit (bit 7) of the FSSET register is 0, the TMSPMD bit is set to 1 and the boot cluster selected by the BTFLG bit (bit 8) of the FLSEC register is reflected in the TMBTSEL bit (bit 6) of the FSSET register.

  TMSPMD = 0: Boot swapping is executed according to the information in the extra area (BTFLG).
  1: Boot swapping is executed according to the TMBTSEL setting.

  BTFLG = 0: Boot cluster 1 is used as the boot area.
  1: Boot cluster 0 is used as the boot area.

  TMBTSEL = 0: Boot cluster 0 is used as the boot area.
  1: Boot cluster 1 is used as the boot area.

  [For use in bank swapping]
  - Only when the TMSPMD bit (bit 7) of the FSSET register is 0, the TMSPMD bit is set to 1 and the startup bank selected by the BTFLG bit (bit 8) of the FLSEC register is reflected in the TMBTSEL bit (bit 6) of the FSSET register.

  TMSPMD = 0: Bank swapping is executed according to the information in the extra area (BTFLG).
  1: Bank swapping is executed according to the TMBTSEL setting.

  BTFLG = 0: Bank 1 is used as the startup bank.
  1: Bank 0 is used as the startup bank.

  TMBTSEL = 0: Bank 0 is used as the startup bank.
  1: Bank 1 is used as the startup bank.

- The extra area is selected as the target area of reprogramming.
  FLARS register = R_RFD_VALUE_U08_FLARS_EXTRA_AREA: 0x01 (EXA (bit 0) = 1)
- The extra area sequencer is activated and the setting of the boot area switching flag is started.
  The value shown below is set in the FLWL register, in which the boot cluster[bank] selected by the parameter (i_e_boot_cluster) is set in the bit that corresponds to the BTFLG bit (bit 8) of the FLSEC register, and R_RFD_VALUE_U08_MASK1_16BIT (0xFFFF) is set in the FLWH register.
  - When R_RFD_ENUM_BOOT_CLUSTER_1 is specified:
    R_RFD_VALUE_U16_MASK0_BOOT_FLAG (0xFEFF) is set in the FLWL register.
  - When R_RFD_ENUM_BOOT_CLUSTER_0 is specified:
    R_RFD_VALUE_U08_MASK1_16BIT (0xFFFF) is set in the FLWL register.
- R_RFD_VALUE_U08_FSSE_SECURITY_FLAG = 0x87 is set in the FSSE register to start the setting of the flag.
  (ESQST (bit 7) = 1, ESQMD (bits 2 to 0) = 7 (0b111), and the other bits are set to 0.)

Notes:

- The parameter (i_e_boot_cluster) must be a correct value (enumerated type: e_rfd_boot_cluster_t). If the value specified for this parameter is neither R_RFD_ENUM_BOOT_CLUSTER_0 nor R_RFD_ENUM_BOOT_CLUSTER_1, R_RFD_ENUM_BOOT_CLUSTER_0 is used.
  [For use in boot swapping]
  Example of “boot cluster =16-Kbyte”
  Boot cluster that is selected as the boot area:
  Allocated to addresses 00000H to 03FFFH (boot area).
  Boot cluster that is not selected as the boot area:
  Allocated to addresses 04000H to 07FFFH (the area immediately following the boot area).
  [For use in bank swapping]
  Example of “bank area” cluster =256-Kbyte”
  Bank that is selected as the startup bank:
  Allocated to addresses 00000H to 3FFFFH (startup bank).
  Bank that is not selected as the startup bank:
  Allocated to addresses 40000H to 7FFFFH (rewrite bank).
- If this function is executed while the sequencer is not in the code flash memory programming mode, the subsequent operation is indeterminate.
- If this function is executed while command execution is in progress in the code/data flash area sequencer or extra area sequencer, the subsequent operation is indeterminate.

3.3.4.5 R_RFD_SetExtraFSWProtectReq

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_SetExtraFSWProtectReq (void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Activates the extra area sequencer and begins the setting of the flag for protection against flash shield window modification.	
Preconditions	Use this function in the code flash memory programming mode while command execution is not in progress in the code/data flash memory area sequencer or extra area sequencer.	
Remarks	Execute the R_RFD_CheckExtraSeqEndStep1() function after this function.	

Details of Specifications:

- The extra area is selected as the target area of reprogramming.
  FLARS register = R_RFD_VALUE_U08_FLARS_EXTRA_AREA: 0x01 (EXA (bit 0) = 1)
- The extra area sequencer is activated and the setting of the flag for protection against flash shield window modification.
  - The FLFSWS register is read and this value is set in the FLWL register with the reserved bits (bits 14 to 9) set to 1 and the FSPR bit (bit 15) set to 0 (modification is disabled).
  - The FLFSWE register is read and this value is set in the FLWH register with the reserved bits (bits 14 to 9) set to 1.
- R_RFD_VALUE_U08_FSSE_FSW = 0x81 is set in the FSSE register to start the setting of the flag.
  (ESQST (bit 7) = 1, ESQMD (bits 2 to 0) = 1 (0b001), and the other bits are set to 0.)

Notes:

- If this function is executed while the sequencer is not in the code flash memory programming mode, the subsequent operation is indeterminate.
- If this function is executed while command execution is in progress in the code/data flash memory area sequencer or extra area sequencer, the subsequent operation is indeterminate.

3.3.4.6 R_RFD_SetExtraFSWReq

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_SetExtraFSWReq (uint16_t i_u16_start_block_number, uint16_t i_u16_end_block_number, e_rfd_fsw_mode_t i_e_fsw_mode);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint16_t i_u16_start_block_number	Start block number Example: For RL78/L23, 0 to 255 (512 Kbytes max.)
	uint16_t i_u16_end_block_number	End block number + 1 Example: For RL78/L23, 1 to 256 (512 Kbytes max.)
	e_rfd_fsw_mode_t i_e_fsw_mode	Flash shield window area R_RFD_ENUM_FSW_MODE_INSIDE: 0x00 [Inside shield area] R_RFD_ENUM_FSW_MODE_OUTSIDE: 0x01 [Outside shield area]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Activates the extra area sequencer and begins the setting of the range and area control of the flash shield window specified by the parameters.	
Preconditions	Use this function in the code flash memory programming mode while command execution is not in progress in the code/data flash memory area sequencer or extra area sequencer.	
Remarks	Execute the R_RFD_CheckExtraSeqEndStep1() function after this function.	

Details of Specifications:

- The extra area is selected as the target area of reprogramming.
  FLARS register = R_RFD_VALUE_U08_FLARS_EXTRA_AREA: 0x01 (EXA (bit 0) = 1)
- The extra area sequencer is activated, and the setting of the start block number and the end block number + 1 of the flash shield window and the flash shield window area is started.
  - The block number specified by the parameter i_u16_start_block_number, which corresponds to the FSWS (flash shield window start block address) register, is set in the FLWL register. Bits 15 to 9 (the bits other than the block address bits) are set to 1.
  - The block number specified by the parameter i_u16_end_block_number, which corresponds to the FSWE (flash shield window end block address) register, is set in the FLWH register. Bits 15 to 9 (the bits other than the block address bits) are set to 1. Only when R_RFD_ENUM_FSW_MODE_INSIDE (inside shield area: 0x00) is specified by the parameter i_e_fsw_mode, bit 15, which corresponds to the FSWC bit, is set to 0.
- R_RFD_VALUE_U08_FSSE_FSW = 0x81 is set in the FSSE register to start the setting.
  (ESQST (bit 7) = 1, ESQMD (bits 2 to 0) = 1 (0b001), and the other bits are set to 0.)

Notes:

- Bits 8 to 0 of a 16-bit parameter are used as the block number to be set (the maximum number is 511); bits 15 to 9 are not used.
- Specify the parameters so that the condition i_u16_start_block_number < i_u16_end_block_number is satisfied.
- For the parameter i_u16_end_block_number, specify the end block number + 1 of the desired window range.

  Examples:

  — To shield four blocks from block 12 to block 15 (inside shield area):
  i_u16_start_block_number = 12, i_u16_end_block_number = 16, and
  i_e_fsw_mode = R_RFD_ENUM_FSW_MODE_INSIDE (0x00)

  — To shield the areas outside the four blocks from block 12 to block 15 (outside shield area):
  i_u16_start_block_number = 12, i_u16_end_block_number = 16, and
  i_e_fsw_mode = R_RFD_ENUM_FSW_MODE_OUTSIDE (0x01)
- If the value specified for the parameter i_e_fsw_mode is neither R_RFD_ENUM_FSW_MODE_INSIDE nor R_RFD_ENUM_FSW_MODE_OUTSIDE, the outside shield area (R_RFD_ENUM_FSW_MODE_OUTSIDE) is used.
- Execute this function only while modification of the flash shield window is enabled by the protection flag (FSPR = 1). If this function is executed while modification is disabled by the flag (FSPR = 0), an extra area sequencer error (bit 0 of FSASTL) will occur and the flash shield window settings will not be changed to the values specified by the parameters.
- If this function is executed while the sequencer is not in the code flash memory programming mode, the subsequent operation is indeterminate.
- If this function is executed while command execution is in progress in the code/data flash memory area sequencer or extra area sequencer, the subsequent operation is indeterminate.
- The size of the code flash memory is different for each device. About the maximum block number which can be used, refer to the user's manual of the target RL78 microcontroller.

3.3.4.7 R_RFD_SetExtraSoftwareReadProtectAreaReq

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_SetExtraSoftwareReadProtectAreaReq (uint16_t i_u16_start_block_number, uint16_t i_u16_end_block_number, e_rfd_protect_t i_e_protect_flag);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint16_t i_u16_start_block_number	Start block number Example: For RL78/L23, 0 to 255 (512 Kbytes max.)
	uint16_t i_u16_end_block_number	End block number Example: For RL78/L23, 0 to 255 (512 Kbytes max.)
	e_rfd_protect_t i_e_protect_flag	Protection flag enable or disable R_RFD_ENUM_PROTECT_OFF: 0x01 [Modification is enabled.] R_RFD_ENUM_PROTECT_ON: 0x00 [Modification is disabled.]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Activates the extra area sequencer and begins the setting of the flash read protection.	
Preconditions	Use this function in the code flash memory programming mode while command execution is not in progress in the code/data flash memory area sequencer or extra area sequencer.	
Remarks	Execute the R_RFD_CheckExtraSeqEndStep1() function after this function.	

Details of Specifications:

- The extra area is selected as the target area of reprogramming.
  FLARS register = R_RFD_VALUE_U08_FLARS_EXTRA_AREA: 0x01 (EXA (bit 0) = 1)
- The extra area sequencer is activated and the setting of the start and end block numbers of the area to be protected and the enabled or disabled protection flag value is started.
  - The block number specified by the parameter i_u16_start_block_number, which corresponds to the LOWAddr (start block address for the flash read protection) register, is set in the FLWL register. Bits 15 to 9 (the bits other than the block address bits) are set to 1.
  - The block number specified by the parameter i_u16_end_block_number, which corresponds to the UPAddr (end block address for the flash read protection) register, is set in the FLWH register. Bits 15 to 9 (the bits other than the block address bits) are set to 1. Only when R_RFD_ENUM_PROTECT_ON (modification is disabled: 0x00) is specified by the parameter i_e_protect_flag, bit 15, which corresponds to the SWPR bit, is set to 0.
- R_RFD_VALUE_U08_FSSE_SOFTWARE_READ = 0x86 is set in the FSSE register to start the setting. (ESQST (bit 7) = 1, ESQMD (bits 2 to 0) = 6 (0b110), and the other bits are set to 0.)

Notes:

- Bits 8 to 0 of a 16-bit parameter are used as the block number to be set (the maximum number is 511); bits 15 to 9 are not used.
- Specify the parameters so that the condition i_u16_start_block_number ≤ i_u16_end_block_number is satisfied.
- For the parameter i_u16_end_block_number (end block number), specify the end block number of the flash read protection. (Unlike the flash shield window, this setting is not the end block number + 1.)
  Example:
  — To specify four blocks from block 12 to block 15:
  i_u16_start_block_number = 12 and i_u16_end_block_number = 15
- If the value specified for the parameter i_e_protect_flag is neither R_RFD_ENUM_PROTECT_OFF nor R_RFD_ENUM_PROTECT_ON, R_RFD_ENUM_PROTECT_OFF (modification is enabled) is used.
- Execute this function only while the reading by the flash protection flag is enabled (SWPR = 1). If this function is executed while the reading by the flash protection flag is disabled (SWPR = 0), an extra area sequencer error (bit 5 of FSASTL) will occur, and the protection settings will not be changed to the values specified by the parameters.
- If this function is executed while the sequencer is not in the code flash memory programming mode, the subsequent operation is indeterminate.
- If this function is executed while command execution is in progress in the code/data flash memory area sequencer or extra area sequencer, the subsequent operation is indeterminate.
- The size of the code flash memory is different for each device. About the maximum block number which can be used, refer to the user's manual of the target RL78 microcontroller.

3.3.4.8 R_RFD_SetBootAreaImmediately

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_SetBootAreaImmediately (e_rfd_boot_cluster_t i_e_boot_cluster);	
Reentrancy	Non-reentrant	
Parameters (IN)	e_rfd_boot_cluster_t i_e_boot_cluster	Boot cluster number (Bank number)
		R_RFD_ENUM_BOOT_CLUSTER_0: 0x01 [Boot cluster 0 (Bank 0)] R_RFD_ENUM_BOOT_CLUSTER_1: 0x00 [Boot cluster 1 (Bank 1)]
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	[For use in boot swapping] Allocates the boot cluster specified by the parameter to the boot area immediately. [For use in bank swapping] Allocates the bank specified by the parameter to the startup bank immediately.	
Preconditions	Use this function in the code flash memory programming mode. Use this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	-	

Details of Specifications:

[For use in boot swapping]

- The value indicating the boot cluster number specified through the parameter (i_e_boot_cluster) by the user is set in the TMBTSEL bit (bit 6) of the FSSET register and a value of 1 is set in the TMSPMD bit (bit 7); the specified boot cluster is immediately allocated to the boot area.
  - When R_RFD_ENUM_BOOT_CLUSTER_0 is specified by the parameter (i_e_boot_cluster):
    The value of “R_RFD_VALUE_U08_FSSET_BOOT_CLUSTER_0 (0x80u) | (g_u08_fset_cpu_frequency)” is set in the FSSET register.
  - When R_RFD_ENUM_BOOT_CLUSTER_1 is specified by the parameter (i_e_boot_cluster):
    The value of “R_RFD_VALUE_U08_FSSET_BOOT_CLUSTER_1 (0xC0u) | (g_u08_fset_cpu_frequency)” is set in the FSSET register.

[For use in bank swapping]

- The value indicating the bank number specified through the parameter (i_e_boot_cluster) by the user is set in the TMBTSEL bit (bit 6) of the FSSET register and a value of 1 is set in the TMSPMD bit (bit 7); the specified bank is immediately allocated to the startup bank.
  - When R_RFD_ENUM_BOOT_CLUSTER_0 is specified by the parameter (i_e_boot_cluster):
    The value of “R_RFD_VALUE_U08_FSSET_BOOT_CLUSTER_0 (0x80u) | (g_u08_fset_cpu_frequency)” is set in the FSSET register.
  - When R_RFD_ENUM_BOOT_CLUSTER_1 is specified by the parameter (i_e_boot_cluster):
    The value of “R_RFD_VALUE_U08_FSSET_BOOT_CLUSTER_1 (0xC0u) | (g_u08_fset_cpu_frequency)” is set in the FSSET register.

Notes:

[For use in boot swapping]

- If an unallowable value is specified by the parameter (i_e_boot_cluster), boot cluster 0 is allocated to the boot area (00000H to 03FFFH : the case of boot cluster size = 16KB).
- The boot cluster that is not selected as the boot area is allocated to the area (04000H to 07FFFH : the case of boot cluster size = 16KB) immediately following the boot area.
- If a CPU reset is applied, the cluster selected by the boot area switching flag (BTFLG: bit 0) of the FLSEC register is allocated to the boot area regardless of the setting by this function.
- If this function is executed while command execution is in progress in the code/data flash area sequencer or extra area sequencer, the subsequent operation is indeterminate.

[For use in bank swapping]

- If an unallowable value is specified by the parameter (i_e_boot_cluster), bank 0 is allocated to the startup bank(00000H to 3FFFFH : the case of bank size = 256KB).
- The bank that is not selected as the startup bank is allocated to the rewrite bank (40000H to 7FFFFH : the case of bank size = 256KB) immediately.
- If a CPU reset is applied, the bank selected by the boot area switching flag (BTFLG: bit 0) of the FLSEC register is allocated to the startup bank regardless of the setting by this function.
- If this function is executed while command execution is in progress in the code/data flash area sequencer or extra area sequencer, the subsequent operation is indeterminate.

3.3.4.9 R_RFD_SetExtraProgrammingMode

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_SetExtraProgrammingMode(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFD_ENUM_RET_ERR_MODE_MISMATCHED: 0x11 [Mode mismatch error] (The flash memory sequencer is not placed in the specified mode.)
Description	The mode of a flash memory sequencer transfers to code flash programming mode and then sets the specified CPU operating frequency in the flash memory sequencer.	
Preconditions	Execute this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	-	

Details of Specifications:

- The hook function R_RFD_HOOK_EnterExtraCriticalSection() is called to save the current interrupt disabled (DI) or enabled (EI) state and disable interrupts.
- R_RFD_VALUE_U08_FLPMC_MODE_CODE_FLASH_PROGRAMMING is set to the FLPMC register value (l_u08_set_flpmc_value). And the mode of a flash memory sequencer transfers to code flash programming mode.
- The hook function R_RFD_HOOK_ExitExtraCriticalSection() is called to restore the interrupt disabled (DI) or enabled (EI) state.
- Set the “g_u08_fset_cpu_frequency” set in the R_RFD_Init function to the FSSET register.

Notes:

- When this function is executed, interrupts need to be disabled in the period between the calls of the hook functions R_RFD_HOOK_EnterExtraCriticalSection() and R_RFD_HOOK_ExitExtraCriticalSection(). If interrupts are enabled and an interrupt occurs in this period, the subsequent operation is indeterminate.
- If this function is executed before the R_RFD_Init function, the reprogrammed data are not guaranteed even after the reprogramming processing by the RFD is completed. To use RFD RL78 Type 11, be sure to execute the R_RFD_Init() function once before starting the use of RFD functions.
- When transitioning to code flash memory programming mode, please transition from non-programmable mode.

3.3.4.10 R_RFD_SetExtraNonProgrammableMode

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_SetExtraNonProgrammableMode(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFD_ENUM_RET_ERR_MODE_MISMATCHED: 0x11 [Mode mismatch error] (The flash memory sequencer is not placed in the specified mode.)
Description	The mode of a flash memory sequencer transfers from code flash programming mode to non-programmable mode.	
Preconditions	Execute this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	-	

Details of Specifications:

- The hook function R_RFD_HOOK_EnterExtraCriticalSection() is called to save the current interrupt disabled (DI) or enabled (EI) state and disable interrupts.
- The value of Non-programmable mode is set to the FLPMC register value (l_u08_set_flpmc_value), the mode of a flash memory sequencer transfers from code flash programming mode to non-programmable mode.
- When the mode is shifted, 10 µs wait is performed.
- The hook function R_RFD_HOOK_ExitExtraCriticalSection() is called to restore the interrupt disabled (DI) or enabled (EI) state.

Notes:

- When this function is executed, interrupts need to be disabled in the period between the calls of the hook functions R_RFD_HOOK_EnterExtraCriticalSection() and R_RFD_HOOK_ExitExtraCriticalSection(). If interrupts are enabled and an interrupt occurs in this period, the subsequent operation is indeterminate.
- FLPMC register is set up according to the value of the execution flag(g_u08_change_interrupt_vector_flag) of the R_RFD_ChangeInterruptVector() function as follows.
  - Execution flag of R_RFD_ChangeInterruptVector() = 0x00 (not executed)
    FLPMC = 0x08 (FWEDIS (bit 3 of FLPMC) = 1)
    (Execution after an interrupt branch according to the interrupt vector table in ROM.)
  - Execution flag of R_RFD_ChangeInterruptVector() = 0x55 (executed)
    FLPMC = 0x00 (FWEDIS (bit 3 of FLPMC) = 0)
    (Execution after an interrupt branch to the specified address in RAM.)
- If this function is executed before the R_RFD_Init function, the reprogrammed data are not guaranteed even after the reprogramming processing by the RFD is completed. To use RFD RL78 Type 11, be sure to execute the R_RFD_Init() function once before starting the use of RFD functions.

3.3.4.11 R_RFD_CheckExtraProgrammingMode

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_CheckExtraProgrammingMode(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFD_ENUM_RET_ERR_MODE_MISMATCHED: 0x11 [Mode mismatch error]
Description	Check whether flash memory control mode is code flash programming mode.	
Preconditions	Execute this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	-	

Details of Specifications:

- The value of the FLPMC register is read to check whether flash memory control mode is code flash programming mode.
  - Code flash memory programming mode: 0x02 (FLSPM (bit 1 of FLPMC) = 1)

Notes:

- If the control mode of the flash memory sequencer was specified by a function other than R_RFD_SetExtraProgrammingMode(), this function may not be executed correctly.
- If this function is executed during command execution in the code/data flash area sequencer or the extra area sequencer, the subsequent operation is indeterminate.

3.3.4.12 R_RFD_CheckExtraNonProgrammableMode

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_CheckExtraNonProgrammableMode(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFD_ENUM_RET_ERR_MODE_MISMATCHED: 0x11 [Mode mismatch error]
Description	Check whether flash memory control mode is non-programmable mode.	
Preconditions	Execute this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	-	

Details of Specifications:

- The value of the FLPMC register is read to check whether flash memory control mode is non-programmable mode.
  - Non-programmable mode:
    FLPMC = 0x08 [FWEDIS (bit 3 of FLPMC) = 1] or 0x00 [FWEDIS (bit 3 of FLPMC) = 0]

Notes:

- If the control mode of the flash memory sequencer was specified by a function other than R_RFD_SetExtraNonProgrammableMode(), this function may not be executed correctly.
- If this function is executed during command execution in the code/data flash area sequencer or the extra area sequencer, the subsequent operation is indeterminate.

3.3.4.13 R_RFD_CheckExtraSeqEndStep1

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_CheckExtraSeqEndStep1(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end]
		R_RFD_ENUM_RET_STS_BUSY: 0x01 [Sequencer command execution is in progress.]
Description	Checks if the operation of the activated extra area sequencer has been completed.	
Preconditions	Execute this command after starting the command for activating the extra area sequencer.	
Remarks	Execute this function again if R_RFD_ENUM_RET_STS_BUSY is returned. After confirming that R_RFD_ENUM_RET_STS_OK has been returned from this function, execute the R_RFD_CheckExtraSeqEndStep2() function.	

Details of Specifications:

- Whether the operation of the activated extra area sequencer has been completed (ESQEND (bit 7 of FSASTH) = 1) is checked.
- When the operation of the extra area sequencer has been completed, 0x00 value is set in the flash extra area sequencer control register (FSSE), and ESQST [bit7] is cleared. And R_RFD_ENUM_RET_STS_OK is returned. If the operation has not been completed, R_RFD_ENUM_RET_STS_BUSY is returned.

Notes:

- Execute this function again if R_RFD_ENUM_RET_STS_BUSY is returned.
- If execution of this function is attempted before the command for activating the extra area sequencer is started, this function is not executed correctly.
- After confirming that R_RFD_ENUM_RET_STS_OK has been returned from this function, execute the R_RFD_CheckExtraSeqEndStep2() function.

3.3.4.14 R_RFD_CheckExtraSeqEndStep2

Information:

Syntax	R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_CheckExtraSeqEndStep2(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_rfd_ret_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end: Sequencer operation has been completed.]
		R_RFD_ENUM_RET_STS_BUSY: 0x01 [Sequencer operation is in progress.]
Description	Checks if the command operation has been completed after the flash extra area sequencer control register is cleared.	
Preconditions	Execute this function after checking that R_RFD_ENUM_RET_STS_OK has been returned from the R_RFD_CheckExtraSeqEndStep1() function.	
Remarks	Execute this function again if R_RFD_ENUM_RET_STS_BUSY is returned.	

Details of Specifications:

- After 0x00 value is set in a flash extra area sequencer control register (FSSE), check whether all operations of the extra area sequencer command have been finished [ESQEND(bit7 of FSASTH) = 0].
- When the command operation in the extra area sequencer has been completed, R_RFD_ENUM_RET_STS_OK is returned.
  If the operation has not been completed, R_RFD_ENUM_RET_STS_BUSY is returned.

Notes:

- Execute this function again if R_RFD_ENUM_RET_STS_BUSY is returned.
- If execution of this function is attempted before R_RFD_ENUM_RET_STS_OK has not been confirmed by the R_RFD_CheckExtraSeqEndStep1() function, this function is not executed correctly.

3.3.4.15 R_RFD_GetExtraSeqErrorStatus

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_GetExtraSeqErrorStatus (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 code/data flash area sequencer or extra area sequencer.	
Preconditions	Execute this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	-	

Details of Specifications:

- The FSASTL register (8 bits) is read and the value of bits 5 to 0 is stored in the variable pointed to by the parameter (onp_u08_error_status).
- Note: Bits 7, bit 6, and bit 2 set the fixed value 0.
  Error information to be acquired (five bits of the FSASTL register: bits 5 to 0):
  Bit 5: Extra area sequencer error
  Bit 4: Code/data flash area sequencer error
  Bit 3: Blank check command error
  Bit 1: Write command error
  Bit 0: Erase command error

Note:

- Correct values may not be acquired if this function is executed while command execution is in progress in the code/data flash area sequencer or extra area sequencer.

3.3.4.16 R_RFD_ClearExtraSeqRegister

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_ClearExtraSeqRegister(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Clears the registers for controlling the code/data flash area sequencer and extra area sequencer.	
Preconditions	Use this function in the code flash memory programming mode. Use this function while command execution is not in progress in the code/data flash area sequencer or extra area sequencer.	
Remarks	Execute this function after execution of the R_RFD_CheckExtraSeqEndStep2() function.	

Details of Specifications:

- The flash registers initialization register (FLRST) is set to 0x01 and then cleared to 0x00 to clear the following registers.
  - Target registers for controlling the code/data flash area sequencer or extra area sequencer:
    FLAPH, FLAPL, FLSEDH, FLSEDL, FLWH, FLWL, FLARS, FSSQ, and FSSE

Notes:

- This function does not clear the information on errors generated during command execution in the flash memory sequencer (the information in the FSASTL register).
- If this function is executed while operation is in progress in the code/data flash area sequencer or extra area sequencer, the subsequent operation is indeterminate.
- If this function is executed while the sequencer is not in the code flash memory programming mode, the subsequent operation is indeterminate.

3.3.4.17 r_rfd_extra_wait_count

Information:

Syntax	R_RFD_FAR_FUNC void r_rfd_extra_wait_count(uint8_t i_u08_count);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint8_t i_u08_count	Wait time (Time count in units of 1 µs: A value from 1 to 255 can be specified.)
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Executes a software loop to wait for the time specified by the parameter (time count in units of 1 µs).	
Preconditions	-	
Remarks	-	

Details of Specifications:

- A value of 1 is added to the g_u08_cpu_frequency value (CPU operating frequency – 1) to obtain the CPU operating frequency.
- The number of software loop repetitions for the specified wait time (time count in units of 1 µs) is calculated and the software loops are executed.

  Number of software loop repetitions for the specified wait time (time count in units of 1 µs)
  = ((frequency [MHz]) × (specified count [µs]) / (loop execution cycles: 8 [cycles])) + 1

  Example: Frequency value = 32 [MHz] and time count = 10 [µs]

  Number of software loop repetitions for the wait time (time count in units of 1 µs)
  = (32 [MHz] × 10 [µs] / 8 [cycles]) + 1

  (1 is added so that the result after rounding does not become smaller than the wait time.)

  = 41 [repetitions]

  Execution time of this function = 1/32 [MHz] × 8 [cycles] × 41 [repetitions] = 10.25 [µs]

Note:

- The range of wait time is from 1 µs to 255 µs, which does not include the overhead of the processing other than the loop processing.

3.3.5 Specifications of Hook Functions

This section describes the hook functions of RFD RL78 Type 11.

3.3.5.1 R_RFD_HOOK_EnterCriticalSection

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_HOOK_EnterCriticalSection(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Save the state of an interrupt enable flag (IE), 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_RFD_DISABLE_INTERRUPT) is executed.

Note:

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

3.3.5.2 R_RFD_HOOK_ExitCriticalSection

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_HOOK_ExitCriticalSection(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Restore the state of the saved interrupt enable flag (IE).	
Preconditions	After the processing is executed with interrupt disabled, a previous interrupt state is restored.	
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_RFD_ENABLE_INTERRUPT) is executed and the interrupt enabled state (EI) is restored.

Note:

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

3.3.5.3 R_RFD_HOOK_EnterCFCriticalSection

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_HOOK_EnterCFCriticalSection(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Save the state of an interrupt enable flag (IE), 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_cf_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_RFD_DISABLE_INTERRUPT) is executed.

Note:

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

3.3.5.4 R_RFD_HOOK_ExitCFCriticalSection

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_HOOK_ExitCFCriticalSection(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Restore the state of the saved interrupt enable flag (IE).	
Preconditions	After the processing is executed with interrupt disabled, a previous interrupt state is restored.	
Remarks	—	

Details of Specifications:

- According to the value of the variable sg_u08_cf_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_cf_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_RFD_ENABLE_INTERRUPT) is executed and the interrupt enabled state (EI) is restored.

Note:

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

3.3.5.5 R_RFD_HOOK_EnterDFCriticalSection

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_HOOK_EnterDFCriticalSection(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Save the state of an interrupt enable flag (IE), 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_df_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_RFD_DISABLE_INTERRUPT) is executed.

Note:

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

3.3.5.6 R_RFD_HOOK_ExitDFCriticalSection

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_HOOK_ExitDFCriticalSection(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Restore the state of the saved interrupt enable flag (IE).	
Preconditions	After the processing is executed with interrupt disabled, a previous interrupt state is restored.	
Remarks	—	

Details of Specifications:

- According to the value of the variable sg_u08_df_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_df_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_RFD_ENABLE_INTERRUPT) is executed and the interrupt enabled state (EI) is restored.

Note:

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

3.3.5.7 R_RFD_HOOK_EnterExtraCriticalSection

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_HOOK_EnterExtraCriticalSection(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Save the state of an interrupt enable flag (IE), 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_extra_ 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_RFD_DISABLE_INTERRUPT) is executed.

Note:

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

3.3.5.8 R_RFD_HOOK_ExitExtraCriticalSection

Information:

Syntax	R_RFD_FAR_FUNC void R_RFD_HOOK_ExitExtraCriticalSection(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Restore the state of the saved interrupt enable flag (IE).	
Preconditions	After the processing is executed with interrupt disabled, a previous interrupt state is restored.	
Remarks	—	

Details of Specifications:

- According to the value of the variable sg_u08_extra_ 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_extra_ 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_RFD_ENABLE_INTERRUPT) is executed and the interrupt enabled state (EI) is restored.

Note:

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

4. Flash Memory Sequencer Operation

4.1 Setting of Flash Memory Control Mode

The flash memory control mode can be changed to the code or data flash memory reprogrammable mode by executing the specific sequence of the flash memory sequencer.

- Code flash memory (and extra area) reprogrammable state:
  Code flash memory programming mode
- Data flash memory reprogrammable state:
  Data flash memory programming mode
- Flash memory (and extra area) non-programmable state:
  Non-programmable mode

Target function of this operation:R_RFD_SetCFProgrammingMode, R_RFD_SetCFNonProgrammableMode,
R_RFD_SetDFProgrammingMode,,R_RFD_CheckDFNonProgrammableMode
R_RFD_SetExtraProgrammingMode,R_RFD_CheckExtraNonProgrammableMode

- When transitioning to code flash memory programming mode or data flash memory programming mode, please transition from non-programmable mode.
- Do not transition directly from code flash memory programming mode to data flash memory programming mode.
- Do not transition directly from data flash memory programming mode to code flash memory programming mode.
- When transition non-programmable mode, follow the procedure corresponding to the current flash memory control mode.

Note: To control the data flash area, the DFLEN bit (bit 0) of the data flash control register (DFLCTL) must be set to 1 (access to the data flash memory must be enabled) in advance.

4.1.1 Procedure for Executing Specific Sequence

The flash programming mode control register (FLPMC) can only be written to by the following specific sequence and the flash memory sequencer can be placed in a desired mode.

Procedure	Specific Sequence (Program Processing)
Step 1	Write a specific value (= 0xA5) to the PFCMD register.
Step 2	Write the value for the desired mode setting to the FLPMC register.
Step 3	Write the inverted value of the desired mode setting to the FLPMC register.
Step 4	Write the value for the desired mode setting to the FLPMC register.

- The specific sequence can only be executed while the FLRST bit (bit 0) of the FLRST register is 0 and the flash memory sequencer is stopped.
- if “accesses to other memories/registers” or “interrupts” occur during Steps 1, 2, 3, and 4 in this specific sequence, the writing to a specific register (FLPMC register) will not be performed.In this case, a protection error occurs and the status flag (FPRERR (bit 0)) of the flash status register (PFS) is set to 1. The FPRERR bit is cleared when a reset is applied or the next time the specific sequence is started.

PFCMD register (After reset: Undefined value):

7	6	5	4	3	2	1	0
REG7	REG6	REG5	REG4	REG3	REG2	REG1	REG0
W	W	W	W	W	W	W	W

- The flash protect command register (PFCMD) is a write-only register and an undefined value is always read from this register.

FLPMC register (After reset: 0x08):

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

PFS register (After reset: 0x00):

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

4.1.2 Procedure for Transition from Non-programmable Mode to the Code Flash Memory Programming Mode

Step 1:	PFCMD register = 0xA5
Step 2:	FLPMC register = 0x02
Step 3:	FLPMC register = 0xFD
Step 4:	FLPMC register = 0x02

- Steps 2 and 4
  FLPMC register setting (0x02)
  EEEMD[bit4] = 0, FWEDIS[bit3] = 0,
  FLSPM[bit1] = 1
- Step 3
  Inverted value or FLPMC register setting (0xFD)

4.1.3 Procedure for Transition from Non-programmable Mode to the Data Flash Memory Programming Mode

Step 1:	PFCMD register = 0xA5
Step 2:	FLPMC register = 0x10
Step 3:	FLPMC register = 0xEF
Step 4:	FLPMC register = 0x10

- Steps 2 and 4
  FLPMC register setting (0x10)
  EEEMD[bit4] = 1, FWEDIS[bit3] = 0,
  FLSPM[bit1] = 0
- Step 3
  Inverted value or FLPMC register setting (0xEF)

4.1.4 Procedure for Transition to the Non-programmable Mode

Data can be read from the target flash memory after the certain amount of wait time has passed since the end of the procedure for a transition from the code flash memory programming mode or data flash memory programming mode to the non-programmable mode.

(1) When the interrupt vector addresses have not been changed to a RAM address

The following shows the transition procedure when the R_RFD_ChangeInterruptVector function has not been executed or when the R_RFD_RestoreInterruptVector function has been executed to change the interrupt branch destinations to the addresses indicated by the interrupt vector table in ROM (initial state).

Step 1:	PFCMD register = 0xA5
Step 2:	FLPMC register = 0x08
Step 3:	FLPMC register = 0xF7
Step 4:	FLPMC register = 0x08

- Steps 2 and 4
  FLPMC register setting (0x08)
  EEEMD[bit4] = 0, FWEDIS[bit3] = 1,
  FLSPM[bit1] = 0
- Step 3
  Inverted value or FLPMC register setting (0xF7)

Step 5: After the wait time (10μs) has passed, data can be read from the target flash memory.

(2) When the interrupt vector addresses have been changed to a RAM address

The following shows the transition procedure when the R_RFD_ChangeInterruptVector() has been executed to change the interrupt branch destinations to a specified address in RAM.

Step 1:	PFCMD register = 0xA5
Step 2:	FLPMC register = 0x00
Step 3:	FLPMC register = 0xFF
Step 4:	FLPMC register = 0x00

- Steps 2 and 4
  FLPMC register setting (0x00)
  EEEMD (bit 4) = 0, FWEDIS (bit 3) = 0,
  FLSPM (bit 1) = 0
- Step 3
  Inverted value or FLPMC register setting (0xFF)

Step 5: After the wait time (10μs) has passed, data can be read from the target flash memory.

4.2 Clearing the Registers for Flash Memory Sequencer Control

The registers shown below can be cleared by setting the FLRST bit of the flash registers initialization register (FLRST) to 1.

Target registers to be initialized: FLAPH, FLAPL, FLSEDH, FLSEDL, FLWH, FLWL, FLARS, FSSQ, and FSSE

Target function of this operation: R_RFD_ClearSeqRegister

Operation Procedure:

- Set the FLRST bit to 1. (Write 0x01 to the FLRST register.)
- Wait for at least one cycle (by using a NOP instruction, etc.).
- Clear the FLRST bit to 0. (Write 0x00 to the FLRST register.)

Note: The FLRST bit can only be modified while both the SQST bit of the FSSQ register and the ESQST bit of the FSSE register are 0 (the flash memory sequencer is stopped). With other settings, the FLRST bit cannot be modified (the writing to this bit is ignored).

FLRST register (After reset: 0x00):

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

4.3 Specifying the Operating Frequency of the Flash Memory Sequencer

Set to FSET [bit4-0] of the flash memory sequencer initial setting register (FSSET) the value (g_u08_fset_cpu_frequency) generated with the “R_RFD_Init function”.

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_RFD_Init and
R_RFD_SetCFProgrammingMode,R_RFD_SetCFNonProgrammableMode,
R_RFD_SetDFProgrammingMode,R_RFD_SetDFNonProgrammableMode
R_RFD_SetExtraProgrammingMode,R_RFD_SetExtraNonProgrammableMode

Operation Procedure:

- Change the flash memory control mode to the code flash memory programming mode or data flash memory programming mode. For the procedures for transitions between modes, see section 4.1.1, Procedure for Executing Specific Sequence, section 4.1.2, Procedure for transition from non-programmable mode to the Code Flash Memory Programming Mode, and section 4.1.3, Procedure for transition from non-programmable mode to the Data Flash Memory Programming Mode.
- Read the flash memory sequencer initial setting register (FSSET) and write the read value to the FSSET register with the values of the TMSPMD bit (bit 7) and TMBTSEL bit (bit 6) retained, bit 5 set to 0, and the bits corresponding to FSET (bits 4 to 0) set to the CPU operating frequency (1 MHz to 32 MHz).

Note: The FSET bits (bits 4 to 0) of the FSSET register can be written to in the code flash memory programming mode or data flash memory programming mode. In other modes, the FSET bits cannot be modified (the writing to the bits is ignored).
Before operating (such as reprogramming) the code flash memory, data flash memory, or extra area 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
TMSPMD	**TMBTSEL**	0	FSET4	FSET3	FSET2	FSET1	FSET0
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W

4.4 Flash Memory Sequencer Commands

4.4.1 Overview

The flash memory sequencer in the RL78/L23 consists of the code/data flash area sequencer, which reprograms the code flash area or data flash area, and the extra area sequencer, which reprograms the extra area. To reprogram individual areas, the commands for the respective sequencers need to be executed. Before using the flash memory sequencer commands, please read and understand the descriptions in (3) Program execution during reprogramming of the flash memory in section 1.5, Points for Caution.

4.4.1.1 Selection of the Area to be Reprogrammed

The area to be reprogrammed needs to be selected by the EXA bit (bit 0) of the flash area selection register (FLARS); select the user area to reprogram the code/data flash area or select the extra area to reprogram the extra area. The EXA bit cannot be modified while the FLRST bit (bit 0) of the FLRST register is 1.

FLARS register (After reset: 0x00):

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

EXA = 0 (after reset): User area is selected.
EXA = 1: Extra area is selected.

4.4.2 Code/Data Flash Area Sequencer Commands

Dedicated commands for the code/data flash area sequencer are used to reprogram the code flash area or data flash area. 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.

FSSQ register (After reset: 0x00):

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

Table 4-1 shows the dedicated commands for the code/data flash area sequencer.

Table 4-1 Dedicated Commands for the Code/Data Flash Area Sequencer

SQMD2 to SQMD0	MDCH Setting	Function of Dedicated Command
		Description
1H	CF: 0 DF: 0	Write
		The data specified in the FLWH and FLWL registers are written to the flash memory address specified by the FLAPH and FLAPL registers. • Code flash memory programming (1 word (4 bytes)): Specify data in the FLWH and FLWL registers. • Data flash memory programming (1 byte): Specify data in the FLW7 to FLW0 bits (bits 7 to 0) of the FLWL register.
3H	CF: 0 DF: 1	Blank check
		Blank check is performed in the area between the address specified by the FLAPH and FLAPL registers and the address specified by the FLSEDH and FLSEDL registers. The value to be set in the MDCH bit (bit 3) of the FSSQ register differs depending on the target flash memory to be checked. For the code flash memory, set the MDCH bit (bit 3) to 0. For the data flash memory, set to 1.
4H	CF: 0 DF: 0	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.
Others	-	Setting prohibited

Note: CF: Code flash memory access
DF: Data flash memory access

- FLAPH and FLAPL registers (flash address pointer registers)
  FLAPH register (After reset: 0x00):

7	6	5	4	3	2	1	0
0	0	0	0	FLAP 19	FLAP 18	FLAP 17	FLAP 16
R	R	R	R	R/W	R/W	R/W	R/W

  FLAPL register (After reset: 0x0000):

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

- FLWH and FLWL registers (flash write buffer registers)
  FLWH register (After reset: 0x0000):

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
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

  FLWL register (After reset: 0x0000):

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
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

Note that the bits used in the FLWH and FLWL registers differ depending on the command to be executed.

- FLSEDH and FLSEDL registers (flash end address pointer registers)
  FLSEDH register (After reset: 0x00):

7	6	5	4	3	2	1	0
0	0	0	0	EWA 19	EWA 18	EWA 17	EWA 16
R	R	R	R	R/W	R/W	R/W	R/W

  FLSEDL register (After reset: 0x0000):

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

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

4.4.2.1 Reprogramming the Code Flash Area

To reprogram the code flash area, change the flash memory control mode to the code flash memory programming mode and then execute commands for the code/data flash area sequencer. Before executing a command, the necessary address and data for the command should be specified in the respective registers.

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

- Block erase unit: **2 Kbytes**
- Write unit: **1 word (4 bytes)**

Target functions of this operation: R_RFD_EraseCodeFlashReq, R_RFD_WriteCodeFlashReq, and R_RFD_BlankCheckCodeFlashReq

Operation Procedure:

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

- Change the control mode to the **code flash memory programming mode**. For the mode transition procedure, see section 4.1.1, Procedure for Executing Specific Sequence, and section 4.1.2, Procedure for Transition to the Code Flash Memory Programming Mode.
- Set the FLARS register to 0x00 (EXA (bit 0) = 0): Select the **user area**.
- 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: 0x002000)
FLSEDH and FLSEDL registers: End block address of the code flash memory (Example: 0x0027FF)

(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 flash memory area (Example: 0x002000)
FLSEDH and FLSEDL registers: Set to all 0s or specify nothing (Example: 0x000000)
FLWH and FLWL registers: Specify the data to be written (1 word (4 bytes)).

(3) Blank check: 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 flash memory area (Example: 0x002000)
FLSEDH and FLSEDL registers: End address of the target flash memory area (Example: 0x0027FF)
Note: To perform blank check only in a 1-word (4-byte) area, set FLAPH = FLSEDH and FLAPL = FLSEDL.

- 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, Blank check: 0x83
- Wait until command execution is completed in the code/data flash area sequencer. For the procedure for waiting for the completion of command execution, see section 4.4.4.1, Procedure for Judging the End of Command Execution in the Code/Data Flash Area Sequencer.
- Processing after command execution
  To continue command processing:
  The same command or a different code flash area reprogramming command can be executed with the data in the registers modified while the sequencer is placed in the **code flash memory programming mode**.
  To complete command processing:
  Place the sequencer in the **non-programmable mode**. For the mode transition procedure, see section 4.1.1, Procedure for Executing Specific Sequence, and section 4.1.4, Procedure for transition to the Non-programmable Mode.

4.4.2.2 Reprogramming the Data Flash Area

To reprogram the data flash area, change the flash memory control mode to the data flash memory programming mode and then execute commands for the code/data flash area sequencer. Before executing a command, the necessary address and data for the command should be specified in the respective registers.

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

- Block erase unit: **256 bytes**
- Write unit: **1 byte**

Target functions of this operation: R_RFD_EraseDataFlashReq, R_RFD_WriteDataFlashReq, and R_RFD_BlankCheckDataFlashReq

Operation Procedure:

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

- Change the control mode to the **data flash memory programming mode**. For the mode transition procedure, see section 4.1.1, Procedure for Executing Specific Sequence, and section 4.1.3, Procedure for transition from non-programmable mode to the Data Flash Memory Programming Mode.
- Set the FLARS register to 0x00 (EXA (bit 0) = 0): Select the **user area**.
- 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: 0x0F1400)
FLSEDH and FLSEDL registers: End block address of the data flash memory (Example: 0x0F14FF)

(2) Write: 1 byte
FLAPH and FLAPL registers: Start address of the target flash memory area (Example: 0x0F1101)
FLSEDH and FLSEDL registers: Set to all 0s or specify nothing. (Example: 0x000000)
FLWH and FLWL registers: Specify the data to be written (0x00000000 to 0x000000FF).
Only the FLW7 to FLW0 bits (bits 7 to 0) are valid.

(3) Blank check:
FLAPH and FLAPL registers: Start address of the target flash memory area (Example: 0x0F1400)
FLSEDH and FLSEDL registers: End address of the target flash memory area (Example: 0x0F14FF)
Note: To perform blank check only in a 1-byte area, set FLAPH = FLSEDH and FLAPL = FLSEDL.

- 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, Blank check: 0x8B **(MDCH (bit 3) = 1: Only for DF)**
- Wait until command execution is completed in the code/data flash area sequencer. For the procedure for waiting for the completion of command execution, see section 4.4.4.1, Procedure for Judging the End of Command Execution in the Code/Data Flash Area Sequencer.
- Processing after command execution
  To continue command processing:
  The same command or a different data flash area reprogramming command can be executed with the data in the registers modified while the sequencer is placed in the **data flash memory programming mode**.
  To complete command processing:
  Place the sequencer in the **non-programmable mode**. For the mode transition procedure, see section 4.1.1, Procedure for Executing Specific Sequence, and section 4.1.4, Procedure for transition to the Non-programmable Mode.

4.4.3 Extra Area Sequencer Commands

Dedicated commands for the extra area sequencer are used to reprogram the extra area. To issue a command, specify the desired command number in the ESQMD3 to ESQMD0 bits (bits 3 to 0) of the flash extra area sequencer control register (FSSE) and set the ESQST bit (bit 7) to 1.

FSSE register (After reset: 0x00):

7	6	5	4	3	2	1	0
ESQST	0	0	0	ESQMD3	ESQMD2	ESQMD1	ESQMD0
R/W	R	R	R	R/W	R/W	R/W	R/W

Table 4-2 shows the dedicated commands for the extra area sequencer.

Table 4-2 Dedicated Commands for the Extra Area Sequencer

ESQMD3 to ESQMD0	Function of Dedicated Command Description
1H	Extra area write (programming of FSW-related data)
	The data specified in the FLWH and FLWL registers are written to the extra flash area. The FSW range, FSW area control, and FSW protection flag are set up. While the FSW protection flag is set (FSPR = 0), this command cannot be executed. If this command is attempted while the protection flag is set, a sequencer error will occur (ESEQER =1 in the FSASTL register).
6H	Extra area write (programming of the setting of the flash read protection)
	The data specified in the FLWH and FLWL registers are written to the extra flash area. The flash read protection are set up. While the protection flag is set (SWPR = 0), this command cannot be executed. If this command is attempted while the protection flag is set, a sequencer error will occur (ESEQER =1 in the FSASTL register).
7H	Extra area write (programming of the security flags and the boot area switching flag)
	The data specified in the FLWH and FLWL registers are written to the extra flash area. The security flags and the boot area switching flag are set up. For the security flags, only the disabling setting can be specified. While the boot area protection is specified (BTPR = 0), the boot area switching flag cannot be modified.
Others	Setting prohibited

4.4.3.1 Reprogramming the Extra Area

To reprogram the extra area, change the flash memory control mode to the code flash memory programming mode and then execute commands for the extra area sequencer. Before executing a command, the necessary data for the command should be specified in the respective registers.

Unit of writing for reprogramming of the extra area:

- Write unit: 1 word (4 bytes)

Note: The erase command is not provided and therefore the unit of erasing is not shown.

Target functions of this operation: R_RFD_SetExtraEraseProtectReq, R_RFD_SetExtraWriteProtectReq, R_RFD_SetExtraBootAreaProtectReq, R_RFD_SetExtraBootAreaReq, R_RFD_SetExtraFSWProtectReq, R_RFD_SetExtraFSWReq, and R_RFD_SetExtraSoftwareReadProtectAreaReq

Operation Procedure:

The data write command for the extra area can be used.

- Change the control mode to the **code flash memory programming mode**. For the mode transition procedure, see section 4.1.1, Procedure for Executing Specific Sequence, and section 4.1.2, Procedure for transition from non-programmable mode to the Code Flash Memory Programming Mode.
- Set the FLARS register to 0x01 (EXA (bit 0) = 1): Select the **extra area**.
- Specify 1-word (4-byte) data in the FLWH and FLWL registers before executing a command. The individual bits (FLW31 to FLW0) of the FLWH and FLWL registers correspond to EX bits 31 to 0 of the target extra area data.

FLWH register (After reset: 0x0000):

15	14	13	12	11	10	9	8
FLW 31	FLW 30	FLW 29	FLW 28	FLW 27	FLW 26	FLW 25	FLW 24
7	6	5	4	3	2	1	0
FLW 23	FLW 22	FLW 21	FLW 20	FLW 19	FLW 18	FLW 17	FLW 16

FLWL register (After reset: 0x0000):

15	14	13	12	11	10	9	8
FLW 15	FLW 14	FLW 13	FLW 12	FLW 11	FLW 10	FLW 9	FLW 8
7	6	5	4	3	2	1	0
FLW 7	FLW 6	FLW 5	FLW 4	FLW 3	FLW 2	FLW 1	FLW 0

Note that the bits used in the FLWH and FLWL registers differ depending on the command to be executed.

- Specify the area to be programmed through the command. Specify the desired command number in the SQMD3 to SQMD0 (bits 3 to 0) bits of the FSSE register and set the ESQST bit (bit 7) to 1.

(1) Programming of the FSW-related data: 0x81

(2) Programming of the setting of the flash read protection area and flag: 0x86

(3) Programming of the security flags and the boot area switching flag: 0x87

- Wait until command execution is completed in the extra area sequencer. For the procedure for waiting for the completion of command execution, see section 4.4.4.2, Procedure for Judging the End of Command Execution in the Extra Area Sequencer.
- Processing after command execution

  To continue command processing:

  The same command or a different extra area reprogramming command can be executed with the data in the registers modified while the sequencer is placed in the **code flash memory programming mode**.

  To complete command processing:

  Place the sequencer in the **non-programmable mode**. For the mode transition procedure, see section 4.1.1, Procedure for Executing Specific Sequence, and section 4.1.4, Procedure for Transition to the Non-programmable Mode.

4.4.3.2 Data Settings for Extra Area Sequencer Commands

The extra area is programmed in units of 1 word (4 bytes) including the data not to be modified. Specify the extra area data (EX bits 31 to 0) for the target command in the FLW31 to FLW0 bits of the FLWH and FLWL registers as shown below and then execute the command.

(1) Programming of the FSW-related data

Specify the following extra area data (EX bits 31 to 0) in the FLW31 to FLW0 bits of the FLWH and FLWL registers.

EX bit 31	EX bit 30	EX bit 29	EX bit 28	EX bit 27	EX bit 26	EX bit 25	EX bit 24
FSWC	—	—	—	—	—	—	FSWE8
EX bit 23	EX bit 22	EX bit 21	EX bit 20	EX bit 19	EX bit 18	EX bit 17	EX bit 16
FSWE7	FSWE6	FSWE5	FSWE4	FSWE3	FSWE2	FSWE1	FSWE0
EX bit 15	EX bit 14	EX bit 13	EX bit 12	EX bit 11	EX bit 10	EX bit 9	EX bit 8
FSPR	—	—	—	—	—	—	FSWS8
EX bit 7	EX bit 6	EX bit 5	EX bit 4	EX bit 3	EX bit 2	EX bit 1	EX bit 0
FSWS7	FSWS6	FSWS5	FSWS4	FSWS3	FSWS2	FSWS1	FSWS0

- FSWE8 to FSWE0 (bits 24 to 16): Specify the value of (end block) + 1 of the window range.
- FSWC (bit 31): Specify the FSW area control.

  FSWC = 0: The **inside** of the window range is shielded.
  1 (setting at shipment): The **outside** of the window range is shielded.
- FSWS8 to FSWS0 (bits 8 to 0): Specify the start block of the window range.
- FSPR (bit 15): Specify the FSW write protection.

  FSPR = 0: Reprogramming of the FSW settings is **disabled**.
  1 (setting at shipment): Reprogramming of the FSW settings is **enabled**.

(2) Programming of the setting of the flash read protection.

Specify the following extra area data (EX bits 31 to 0) in the FLW31 to FLW0 bits of the FLWH and FLWL registers.

EX bit 31	EX bit 30	EX bit 29	EX bit 28	EX bit 27	EX bit 26	EX bit 25	EX bit 24
SWPR	—	—	—	—	—	—	UPAddr8
EX bit 23	EX bit 22	EX bit 21	EX bit 20	EX bit 19	EX bit 18	EX bit 17	EX bit 16
UPAddr7	UPAddr6	UPAddr5	UPAddr4	UPAddr3	UPAddr2	UPAddr1	UPAddr0
EX bit 15	EX bit 14	EX bit 13	EX bit 12	EX bit 11	EX bit 10	EX bit 9	EX bit 8
—	—	—	—	—	—	—	LOWAddr8
EX bit 7	EX bit 6	EX bit 5	EX bit 4	EX bit 3	EX bit 2	EX bit 1	EX bit 0
LOWAddr7	LOWAddr6	LOWAddr5	LOWAddr4	LOWAddr3	LOWAddr2	LOWAddr1	LOWAddr0

- UPAddr8 to UPAddr0 (bits 24 to 16): Specify the end block of the flash read protection.
- LOWAddr8 to LOWAddr0 (bits 8 to 0): Specify the start block of the flash read protection.
- SWPR (bit 31): Specify the write protection for the setting of the flash read protection is disabled.

SWPR = 0: Modification of the read- prohibited area setting is **disabled**.
1 (setting at shipment): Modification of the read- prohibited area setting is **enabled**.

(3) Programming of the security flags and the boot area switching flag

Specify the following extra area data (EX bits 31 to 0) in the FLW31 to FLW0 bits of the FLWH and FLWL registers.

EX bit 31	EX bit 30	EX bit 29	EX bit 28	EX bit 27	EX bit 26	EX bit 25	EX bit 24
1	1	1	1	1	1	1	1
EX bit 23	EX bit 22	EX bit 21	EX bit 20	EX bit 19	EX bit 18	EX bit 17	EX bit 16
1	1	1	1	1	1	1	1
EX bit 15	EX bit 14	EX bit 13	EX bit 12	EX bit 11	EX bit 10	EX bit 9	EX bit 8
1	1	1	WRPR	1	SEPR	BTPR	BTFLG
EX bit 7	EX bit 6	EX bit 5	EX bit 4	EX bit 3	EX bit 2	EX bit 1	EX bit 0
1	1	1	1	1	1	1	1

- WRPR (bit 12): Specify the write protection in the serial programming mode.

  WRPR = 0: Programming in the serial programming mode is **disabled**.
  1 (setting at shipment): Programming in the serial programming mode is **enabled**.

- SEPR (bit 10): Specify the block erasure protection in the serial programming mode.

  SEPR = 0: Block erasure in the serial programming mode is **disabled**.
  1 (setting at shipment): Block erasure in the serial programming mode is **enabled**.

- BTPR (bit 9): Specify the protection against reprogramming of the boot area in the serial or self programming mode.

  BTPR = 0: Reprogramming of the boot area is **disabled**.
  1 (setting at shipment): Reprogramming of the boot area is **enabled**.

- BTFLG (bit 8): Control the boot cluster to be allocated to the boot area when TMSPMD = 0 (boot swapping is executed according to the setting of the boot area switching flag (BTFLG) in the extra area).

  BTFLG = 0: Boot cluster 1 is used as the boot area.
  1 (setting at shipment): Boot cluster 0 is used as the boot area.

Notes: 1. When modifying the BTFLG flag, set the other bits to 1.

2. When modifying a security flag other than the BTFLG flag to 0 (disabled), set the other bits to 1 except for the BTFLG flag (set to the read value).

3. After the WRPR flag is set to 0 (disabled), it can be set to 1 (enabled) only when the erase chip command is executed in the serial programming mode

*** While any of the following protections is set (operation is disabled), the erase chip command cannot be executed in the serial programming mode.**

- **SEPR = 0 (Protection against block erasure)**
- **BTPR = 0 (Protection against reprogramming of the boot area)**
- **IFPR = 0 (Protection against connection of a programmer or OCD)**

4.4.4 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/L23, a specific procedure for judging the end of command execution should be used.

Read the ESQEND bit (bit 7) or SQEND bit (bit 6) of the FSASTH register and confirm that it is set to 1 to judge the end of command execution in the code/data flash area sequencer or extra area sequencer. After this judgement, read the error bits (BLER (bit 3), 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 / 0x04):

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

FSASTL register (After reset: 0x00 / 0x80):

7	6	5	4	3	2	1	0
MBTSEL	MOPEN	ESEQER	SEQER	BLER	0	WRER	ERER
R	R	R	R	R	R	R	R

Note: The boot flag monitor bit (MBTSEL (bit 7)) holds the inverted value of the boot area switching flag (BTFLG (bit 8)) in the extra area.

4.4.4.1 Procedure for Judging the End of Command Execution in the Code/Data Flash Area Sequencer

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.

4.4.4.2 Procedure for Judging the End of Command Execution in the Extra Area Sequencer

Judgment Procedure:

(1) After starting the execution of a command in the extra area sequencer, wait until the ESQEND bit (bit 7) of the FSASTH register is automatically set.
(2) After confirming that the ESQEND bit (bit 7) has been set, clear the ESQST bit (bit 7) of the FSSE register.
(3) Wait until the ESQEND bit (bit 7) of the FSASTH register is automatically cleared; the procedure ends when the bit is cleared.

4.4.5 Procedure for Forcibly Terminating Command Execution in the Code/Data Flash Area Sequencer

Command execution in the code/data flash area sequencer can be forcibly terminated if an emergency stop is necessary.

Note: Command execution in the extra area sequencer cannot be forcibly terminated.

Procedure of Forced Termination:

(1) Set the FSSTP bit (bit 6) of the FSSQ register to 1 between the start of command execution (step (1) in section 4.4.4.1, Procedure for Judging the End of Command Execution in the Code/Data Flash Area Sequencer) and the clearing of the SQST bit (bit 7) of the FSSQ register (step (2)); the command execution started in the code/data flash area sequencer is forcibly stopped.

(2) Check that the SQEND bit (bit 6) of the FSASTH register has been set and then clear the SQST bit (bit 7) and FSSTP bit (bit 6) 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.

FSSQ register (After reset: 0x00):

7	6	5	4	3	2	1	0
SQST	FSSTP	0	0	MDCH	SQMD2	SQMD1	SQMD 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

4.5 Boot Swapping Function / Bank Swapping Function

4.5.1 Overview

If reprogramming fails due to a temporary power failure or a reset from an external source while the boot area, which stores the vector table data, basic functions of programs, and boot program for self-programming, is being reprogrammed, the data in the boot area are damaged; the user program cannot be restarted or reprogrammed even by a reset applied after that. Boot swapping function or bank swapping function is provided to avoid this situation.

4.5.2 Boot Swapping Function

The boot swapping function replaces boot cluster 0, which is the boot area, with boot cluster 1, which is the target area of boot swapping. Before starting the reprogramming processing, write a new boot program to boot cluster 1. Swap boot cluster 1 and boot cluster 0 so that boot cluster 1 is allocated to the boot area. Even if momentary power loss occurs during reprogramming of the area to boot, the next reset start is booted from the boot cluster 0 to which the original boot program was written, and can redo reprogramming.

Boot area	The logical area started from 00000H including the reset vector, and the size is different with values set to the BTBLS bit.
Boot clusters 0 and 1	A boot cluster is a block group of the size fixed by the set value of BTBLS bits. The one of the two is located to a boot area (boot cluster 0). The example of the size fixed by the set value of the BTBLS bits of RL78/L23 is shown below. - BTBLS = 00H : Boot cluster size : 2 Kbytes Boot cluster 0 [00000H to 007FFH] / Boot cluster 1 [00800H to 00FFFH] - BTBLS = 01H : Boot cluster size : 4 Kbytes Boot cluster 0 [00000H to 00FFFH] / Boot cluster 1 [01000H to 01FFFH] - BTBLS = 02H : Boot cluster size : 8 Kbytes Boot cluster 0 [00000H to 01FFFH] / Boot cluster 1 [02000H to 03FFFH] - BTBLS = 03H : Boot cluster size : 16 Kbytes Boot cluster 0 [00000H to 03FFFH] / Boot cluster 1 [04000H to 07FFFH] - BTBLS = 04H : Boot cluster size : 32 Kbytes Boot cluster 0 [00000H to 07FFFH] / Boot cluster 1 [08000H to 0FFFFH] - BTBLS = 05H : Boot cluster size : 64 Kbytes Boot cluster 0 [00000H to 0FFFFH] / Boot cluster 1 [10000H to 1FFFFH] - BTBLS = 06H : Boot cluster size : 128 Kbytes Boot cluster 0 [00000H to 1FFFFH] / Boot cluster 1 [20000H to 3FFFFH]

Note: Use the dedicated flash memory programmer to set the BTBLS bits. When BTBLS bits are set, it is necessary to set a BAPR bit to 0 at the same time.
The logical addresses of boot cluster 0 and boot cluster 1 are switched after boot swapping.
The TMSPMD bit (bit 7) and TMBTSEL bit (bit 6) of the FSSET register can only be modified while BTPR = 1 and the flash memory sequencer is in the code flash memory programming mode or data flash memory programming mode. In other cases, the TMSPMD and TMBTSEL bits cannot be manipulated (writing to these bits is ignored).

4.5.2.1 Operation of the Boot Swapping Function

The operation of the boot swapping function is controlled by the boot area switching flag (BTFLG) in the extra area or the TMBTSEL bit (bit 6) of the flash memory sequencer initial setting register (FSSET) depending on the setting of the TMSPMD bit (bit 7) of the FSSET register.

- When the TMSPMD bit (bit 7) is 0 (after reset), the boot area is determined according to the setting of the BTFLG in the extra area.
  BTFLG = 0: Boot cluster 1 is used as the boot area.
  1 (setting at shipment): Boot cluster 0 is used as the boot area.
- When the TMSPMD bit (bit 7) is 1, the boot area is determined according to the setting of the TMBTSEL bit (bit 6) in the FSSET register.
  TMBTSEL = 0 (after reset): Boot cluster 0 is used as the boot area.
  1: Boot cluster 1 is used as the boot area.

FSSET register (After reset: 0x00):

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

4.5.2.2 Execution of the Boot Swapping Function

The function can be executed in two ways: execution after reset and immediate execution.

Note: When writing to the FSSET register to manipulate the TMSPMD bit or TMBTSEL bit, do not modify the value of the FSET4 to FSET0 bits (CPU operating frequency) of the register. Before writing to the FSSET register, be sure to read the register, and then write to it without changing the value of the FSET4 to FSET0 bits.
If an incorrect CPU operating frequency is set in the FSSET register, the operation of the flash memory sequencer is indeterminate and the reprogrammed values in the flash memory are not guaranteed.

4.5.2.3 Boot Swapping Execution after Reset

Boot swapping is not executed immediately after writing BTFLG, but it is executed after reset.

Note: When BTPR = 0, neither the TMSPMD bit can be modified nor the BTFLG can be set by programming of the extra area. Therefore, boot swapping is not executed.

Target function of this operation: R_RFD_SetExtraBootAreaReq

Operation Procedures:

(1) When the TMSPMD bit = 0 (boot swapping according to BTFLG):

- Read the BTFLG bit of the FLSEC register.
  a) When the BTFLG bit = 0 in the FLSEC register:
  - Set the TMSPMD bit to 1 (boot swapping according to TMBTSEL) and the TMBTSEL bit to 1.
  - Write to the BTFLG bit in the **extra area**. (Specify the boot cluster to be used as the boot area. ESQMD = 7H in the FSSE register)
  - Boot swapping is executed after the reset operation and execution branches to the reset vector address in the specified boot cluster.

  b) When the BTFLG bit = 1 in the FLSEC register:
  - Set the TMSPMD bit to 1 (boot swapping according to TMBTSEL) and the TMBTSEL bit to 0.
  - Write to the BTFLG bit in the **extra area**. (Specify the boot cluster to be used as the boot area. ESQMD = 7H in the FSSE register)
  - Boot swapping is executed after the reset operation and execution branches to the reset vector address in the specified boot cluster.

(2) When the TMSPMD bit = 1 (boot swapping according to TMBTSEL):

- Write to the BTFLG bit in the **extra area**. (Specify the boot cluster to be used as the boot area. ESQMD = 7H in the FSSE register)
- Boot swapping is executed after the reset operation and execution branches to the reset vector address in the specified boot cluster.

4.5.2.4 Immediate Execution of Boot Swapping

The specified boot cluster is immediately allocated to the boot area (boot swapping is performed immediately).

Note: When BTPR = 0, the TMSPMD bit cannot be modified and boot swapping is not executed.

Target function of this operation: R_RFD_SetBootAreaImmediately

Operation Procedures:

(1) When the TMSPMD bit = 0 (boot swapping according to BTFLG):

- Read the MBTSEL bit of the FSAST register and set the value in the TMBTSEL bit of the FSSET register.
  a) When the TMBTSEL bit = 1:
     - Set the TMSPMD bit to 1 (boot swapping according to TMBTSEL) and the TMBTSEL bit to 0. Boot swapping is executed immediately.
  b) When the TMBTSEL bit = 0:
     - Set the TMSPMD bit to 1 (boot swapping according to TMBTSEL) and the TMBTSEL bit to 1. Boot swapping is executed immediately.

(2) When the TMSPMD bit = 1 (boot swapping according to TMBTSEL):

a) When the TMBTSEL bit = 1:
   - Set the TMBTSEL bit to 0. Boot swapping is executed immediately.
b) When the TMBTSEL bit = 0:
   - Set the TMBTSEL bit to 1. Boot swapping is executed immediately.

4.5.3 Bank swapping function

The bank swapping function replaces startup bank (bank 0) and rewrite bank (bank 1). Before starting the reprogramming processing, write a new boot program to bank 1. Swap bank 1 and bank 0 so that bank 1 is allocated at the startup bank side. Even if momentary power loss occurs during reprogramming of the rewrite bank, the next reset start is booted from the startup bank to which the original boot program was written, and can redo reprogramming.

Startup area	The logical area started from 00000H including the reset vector.
Bank 0 and 1	The value of the bank swapping setup (BTBLS bit) may change with devices. And, the ability to execute that bank swap depend on the device being used. Be sure to confirm with reference to the user's manual of a target device. The example of a bank swap setting of RL78/L23 (R7F100LxL) is shown below. - BTBLS = 07H : Bank size : 256 Kbytes (R7F100LxL) Bank 0 [0x00000 to 0x3FFFF] Bank 1 [0x40000 to 0x7FFFF]

Note: Use the dedicated flash memory programmer to set the BTBLS bits. When BTBLS bits are set, it is necessary to set a BAPR bit to 0 at the same time.
The logical addresses of bank 0 and bank 1 are switched after bank swapping.
The TMSPMD bit (bit 7) and TMBTSEL bit (bit 6) of the FSSET register can only be modified while BTPR = 1 and the flash memory sequencer is in the code flash memory programming mode or data flash memory programming mode. In other cases, the TMSPMD and TMBTSEL bits cannot be manipulated (writing to these bits is ignored).

4.5.3.1 Operation of the Bank Swapping Function

The operation of the bank swapping function is controlled by the startup bank switching flag (BTFLG) in the extra area or the TMBTSEL bit (bit 6) of the flash memory sequencer initial setting register (FSSET) depending on the setting of the TMSPMD bit (bit 7) of the FSSET register.

- When the TMSPMD bit (bit 7) is 0 (after reset), the startup bank is determined according to the setting of the BTFLG in the extra area.
  BTFLG = 0: Bank 1 is used as the startup bank.
  1 (setting at shipment): Bank 0 is used as the startup bank.
- When the TMSPMD bit (bit 7) is 1, the startup bank is determined according to the setting of the TMBTSEL bit (bit 6) in the FSSET register.
  TMBTSEL = 0 (after reset): Bank 0 is used as the startup bank.
  1: Bank 1 is used as the startup bank.

FSSET register (After reset: 0x00):

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

4.5.3.2 Execution of the Bank Swapping Function

The function can be executed in two ways : “execution after reset” or “active bank swapping execution” (immediate execution bank swapping).

Note: When writing to the FSSET register to manipulate the TMSPMD bit or TMBTSEL bit, do not modify the value of the FSET4 to FSET0 bits (CPU operating frequency) of the register. Before writing to the FSSET register, be sure to read the register, and then write to it without changing the value of the FSET4 to FSET0 bits.
If an incorrect CPU operating frequency is set in the FSSET register, the operation of the flash memory sequencer is indeterminate and the reprogrammed values in the flash memory are not guaranteed.

4.5.3.3 Bank Swapping Execution after Reset

Bank swapping is not executed immediately after the BTFLG is written to but executed after reset.

Note: When BTPR = 0, neither the TMSPMD bit can be modified nor the BTFLG can be set by programming of the extra area. Therefore, bank swapping is not executed.

Target function of this operation: R_RFD_SetExtraBootAreaReq

Operation Procedures:

(1) When the TMSPMD bit = 0 (bank swapping according to BTFLG):

- Read the BTFLG bit of the FLSEC register.
  a) When the BTFLG bit = 0 in the FLSEC register:
  - Set the TMSPMD bit to 1 (bank swapping according to TMBTSEL) and the TMBTSEL bit to 1.
  - Write to the BTFLG bit in the **extra area**. (Specify the bank to be used as the startup bank. ESQMD = 7H in the FSSE register)
  - Bank swapping is executed after the reset operation and execution branches to the reset vector address in the specified bank.

  b) When the BTFLG bit = 1 in the FLSEC register:
  - Set the TMSPMD bit to 1 (bank swapping according to TMBTSEL) and the TMBTSEL bit to 0.
  - Write to the BTFLG bit in the **extra area**. (Specify the bank to be used as the startup bank. ESQMD = 7H in the FSSE register)
  - Bank swapping is executed after the reset operation and execution branches to the reset vector address in the specified bank.

(2) When the TMSPMD bit = 1 (bank swapping according to TMBTSEL):

- Write to the BTFLG bit in the **extra area**. (Specify the bank to be used as the startup bank. ESQMD = 7H in the FSSE register)
- Bank swapping is executed after the reset operation and execution branches to the reset vector address in the specified bank.

4.5.3.4 Active Bank Swapping Execution (Immediate execution bank swapping)

The specified bank is immediately allocated to the startup bank (0x00000~) (bank swapping is performed immediately).

Note: When BTPR = 0, the TMSPMD bit cannot be modified, and bank swapping is not executed.

Target function of this operation: R_RFD_SetBootAreaImmediately

Operation Procedures:

(1) When the TMSPMD bit = 0 (bank swapping according to BTFLG):

- Read the MBTSEL bit of the FSAST register and set the value in the TMBTSEL bit of the FSSET register.
  a) When the TMBTSEL bit = 1:
  - Set the TMSPMD bit to 1 (bank swapping according to TMBTSEL) and the TMBTSEL bit to 0. Bank swapping is executed immediately.
  b) When the TMBTSEL bit = 0:
  - Set the TMSPMD bit to 1 (bank swapping according to TMBTSEL) and the TMBTSEL bit to 1. Bank swapping is executed immediately.

(2) When the TMSPMD bit = 1 (bank swapping according to TMBTSEL):

a) When the TMBTSEL bit = 1:
- Set the TMBTSEL bit to 0. Bank swapping is executed immediately.

b) When the TMBTSEL bit = 0:
- Set the TMBTSEL bit to 1. Bank swapping is executed immediately.

4.6 Flash Shield Window Function

4.6.1 Overview

The flash shield window (FSW) function is provided as one of the security functions for self-programming. It disables programming and erasure of areas other than the specified window range only during self-programming. The window range for the FSW function is specified by the start block and the end block +1. There are the FSWC bit which specifies the range of flash memory shield area, and the FSWC bit which specifies whether to enable or disable changing of the flash shield window setting area.

4.6.2 Operation of the Flash Shield Window Function

The operation of the FSW function is determined by the settings in the flash FSW monitoring registers (FLFSWE and FLFSWS), which reflect the FSW information written to the extra area. To modify the FSW settings, use the extra area sequencer to write the setting values to the extra area for FSW settings.

FLFSWE register (**the value in the corresponding extra area is reflected** in this register after reset or when the extra area is programmed):

15	14	13	12	11	10	9	8
FSWC	0	0	0	0	0	0	FSWE8
R	R	R	R	R	R	R	R
7	6	5	4	3	2	1	0
FSWE7	FSWE6	FSWE5	FSWE4	FSWE3	FSWE2	FSWE1	FSWE0
R	R	R	R	R	R	R	R

FLFSWS register (**the value in the corresponding extra area is reflected** in this register after reset or when the extra area is programmed):

15	14	13	12	11	10	9	8
FSPR	0	0	0	0	0	0	FSWS8
R	R	R	R	R	R	R	R
7	6	5	4	3	2	1	0
FSWS7	FSWS6	FSWS5	FSWS4	FSWS3	FSWS2	FSWS1	FSWS0
R	R	R	R	R	R	R	R

- FSWE (bits 8 to 0) of the FLFSWE register: Specify the end block number + 1 of the window range.
- FSWC (bit 15) of the FLFSWE register: Control the FSW area.

  FSWC = 0: The **inside** of the window range is specified as the shield area.

  1 (setting at shipment): The **outside** of the window range is specified as the shield area.
- FSWS (bits 8 to 0) of the FLFSWS register: Specify the start block number of the window range.
- FSPR (bit 15) of the FLFSWS register: Enable or disable modification of FSW settings.

  FSPR = 0: Modification of FSW settings is **disabled**.

  1 (setting at shipment): Modification of FSW settings is **enabled**.

4.6.3 Execution of the Flash Shield Window Function

4.6.3.1 Control of the Flash Shield Window Area

The flash shield window (FSW) area can be switched between the outside shield area (FSWC = 1) which shields the outside of the window range, and the inside shield area (FSWC = 0) which shields the inside of the window range.

Target function of this operation: R_RFD_SetExtraFSWReq

Operation Procedure:

- Write to the FSWE, FSWC, and FSWS bits in the **extra area**. (ESQMD = 1H in the FSSE register)
  FSWE: End block number +1 of the FSW window range.
  FSWC: Control of the FSW area
  FSWC = 1 (setting at shipment): Outside shield area
  0: Inside shield area
  FSWS: Start block of the FSW window range.

Note: Set the FSPR bit and reserved bits (bits 14 to 9) to 1. The FSW settings cannot be modified while FSPR = 0. When the FSWS and FSWE bits are set to the same value, reprogramming is enabled in the entire area of the code flash memory regardless of the FSWC setting.

FSPR: Protection against FSW modification

(1) Outside shield area (FSWC = 1)
Example: Target device = R7F100LPL
Specify “start block = 03H”, “end block+1 = 06H”.

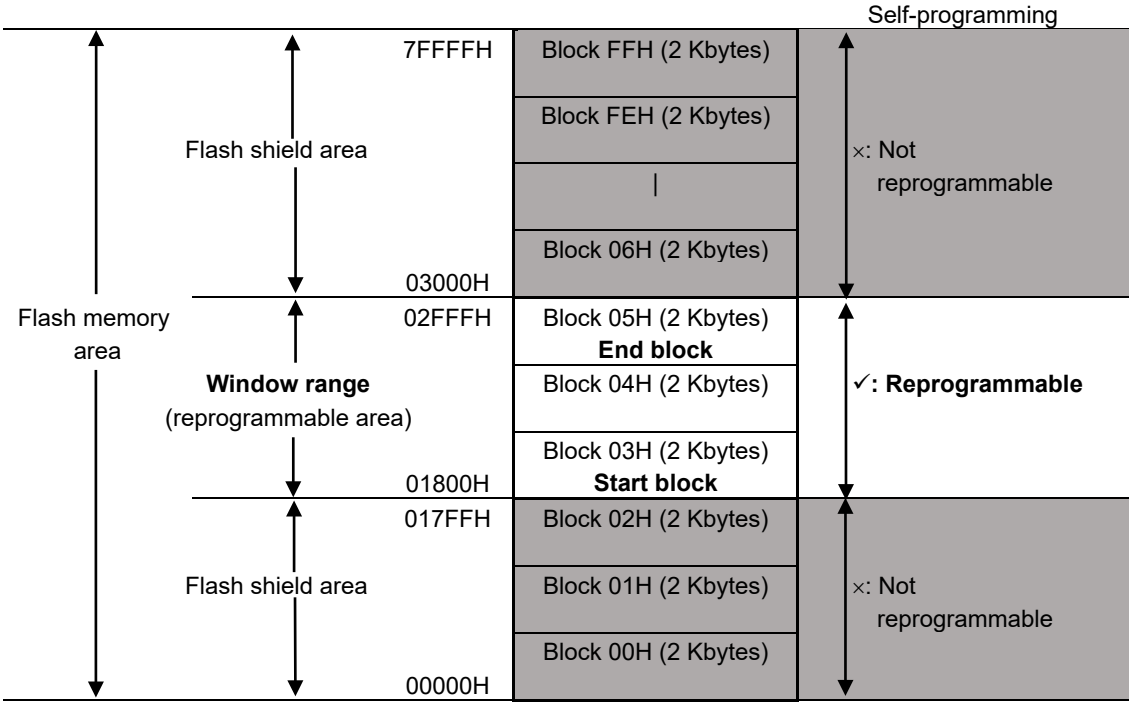


Figure 4-1 Example of FSW Settings (Outside Shield Area)

(2) Inside shield area (FSWC = 0)

Example: Target device = R7F100LPL

Specify “start block = 03H”, “end block+1 = 06H”.

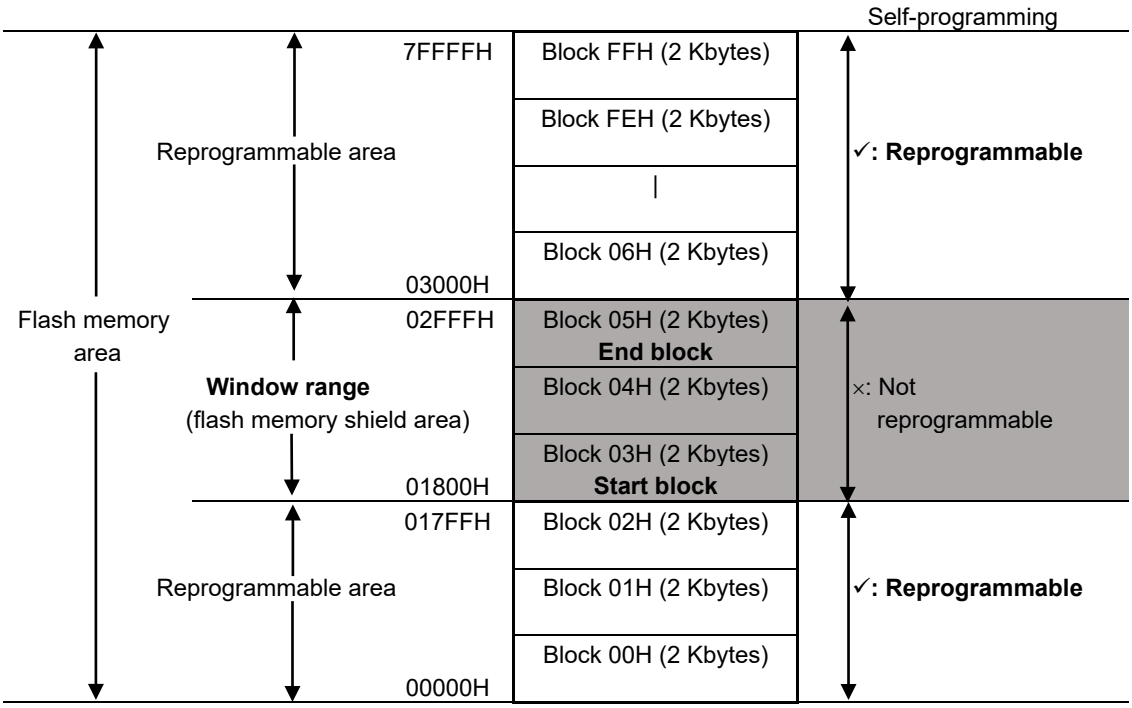


Figure 4-2 Example of FSW Settings (Inside Shield Area)

4.6.3.2 Protection against Flash Shield Window (FSW) Modification

Modification of the settings of the flash shield window range and FSW area can be prohibited (FSPR = 0).

Target function of this operation: R_RFD_SetExtraFSWProtectReq

Operation Procedure:

- Write 0 to the FSPR bit in the **extra area**. (ESQMD = 0x1 in the FSSE register)

Note: When writing to the FSPR bit, set the FSWS, FSWE, and FSWC bits of the **extra area** to the same value as those of the flash FSW monitoring registers (FLFSWE and FLFSWS) and set the reserved bits (bits 14 to 9) to 1.

How to Release the Protection against FSW Modification:

After the protection against FSW modification is set, it cannot be released during self-programming. It can only be released by the erase chip command in the serial programming mode.

Note: While any of the following protections is set, the erase chip command cannot be executed in the serial programming mode.

- SEPR = 0 (Protection against block erasure)
- BTPR = 0 (Protection against reprogramming of boot cluster 0)
- IFPR = 0 (Protection against connection of a programmer or OCD)

4.7 Interrupts in Code Flash Memory Programming Mode

4.7.1 Overview

When an interrupt occurs in the RL78, the interrupt vector table in ROM is referenced and execution branches to the interrupt processing code at the ROM address pointed to by the corresponding interrupt vector. As an interrupt vector is a 16-bit address, execution can branch within a maximum of 64-Kbyte ROM area. However, ROM cannot be referenced in the code flash memory programming mode, in which the code flash memory and extra area can be reprogrammed, and therefore interrupt processing cannot be executed in this mode.

In the RL78/L23, the branch destinations of all interrupts can be changed to the specified address in RAM. Even when ROM cannot be referenced, interrupt processing can be executed in RAM without using the interrupt vector table or interrupt processing code in ROM.

4.7.2 Operation when Interrupt Branch Destinations are Changed

The interrupt vector change registers (FLSIVC1 and FLSIVC0) and interrupt address control register (VECTCTRL) are used to change the branch destinations of all interrupts to an address in RAM. After these registers are set up, the interrupt processing in RAM can be executed without reference to the interrupt vector table in ROM if an interrupt occurs in the code flash memory programming mode.

The FLSIVC1 and FLSIVC0 registers specify the branch destination address of all interrupts generated during reprogramming of the code flash memory or extra area. Specify the lower 16 bits of the address in FLSIVC0 and the upper 4 bits in FLSIVC1.

FLPMC register (After reset: 0x08):

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

— While the FWEDIS bit (bit 3) of the FLPMC register is 0, set the VECTCTRL register to 0x01, and execution after any interrupt generated during self-programming branches to the user-specified RAM address stored in the FLSIVC1 and FLSIVC0 registers.

FLSIVC1 register (After reset: 0x000F (fixed value)):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

FLSIVC0 register (After reset: 0x0000): Specify the lower 16 bits of RAM address 0x000Fxxxx.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

VECTCTRL register (After reset: 0x00):

7	6	5	4	3	2	1	0
—	—	—	—	—	—	—	VECTCTRL
—	—	—	—	—	—	—	R/W

— VECTCTRL bit (bit 0) of the VECTCTRL register: Control whether to branch to RAM after an interrupt occurs during self-programming.

- When VECTCTRL (bit 0) = 0 (value after reset) or FWEDIS (bit 3) = 1 (value after reset) in the FLPMC register:

  Execution branches to the address pointed to by the vector table in ROM corresponding to the generated interrupt.
- When VECTCTRL (bit 0) = 1 (FWEDIS (bit 3) = 0 in the FLPMC register):

  Execution after any interrupt branches to the user-specified RAM address stored in the FLSIVC1 and FLSIVC0 registers.

Notes: 1. The user should check the interrupt flags to identify the source of the generated interrupt after the above registers are set up. Therefore, the interrupt flags are not automatically cleared; the user should clear them after identifying the interrupt source.

2. The interrupt branch destinations cannot be changed to a ROM address (can only be changed within the address range of 0FxxxxH).
3. The interrupt branch destination changed by the above registers is only valid during self-programming.
4. While manipulating these registers to change the interrupt branch destinations to RAM, be sure to disable interrupts.

4.7.3 Procedures for Changing the Interrupt Branch Destinations

To execute the interrupt processing in RAM, the FLSIVC1 and FLSIVC0 registers and bit 0 of the VECTCTRL register should be modified while the FWEDIS bit (bit 3) of the flash programming mode control register (FLPMC) is 0. Execute the specific sequence of the flash memory sequencer to manipulate the FWEDIS bit (bit 3) of the FLPMC register and set up the FLSIVC1 and FLSIVC0 registers and bit 0 of the VECTCTRL register so that the interrupt branch destinations are changed to a RAM address.

Note: The FLPMC register can only be written to by executing the specific sequence described in section 4.1.1, Procedure for Executing Specific Sequence.

(1) Changing the interrupt branch destinations to a RAM address

The following shows the procedure for changing the branch destinations of all interrupts to the specified RAM address.

Target function of this operation: R_RFD_ChangeInterruptVector

Operation Procedure:

- Save the current interrupt enabled or disabled setting and then disable interrupts.
- Execute the specific sequence to set the FWEDIS bit (bit 3) of the FLPMC register to 0.

Step 1:	PFCMD register = 0xA5
Step 2:	FLPMC register = 0x00
Step 3:	FLPMC register = 0xFF
Step 4:	FLPMC register = 0x00

- Steps 2 and 4
  FLPMC register setting (0x00)
  EEEMD (bit 4) = 0, FWEDIS (bit 3) = 0,
  FLSPM (bit 1) = 0
- Step 3
  Inverted value or FLPMC register setting (0xFF)

- Specify a RAM address in the FLSIVC1 and FLSIVC0 registers.
- Set the VECTCTRL register to 0x01 so that execution after an interrupt branch to the specified address in RAM.
- Restore the saved interrupt enabled or disabled setting.

Notes: 1. Keep FWEDIS (bit 3) = 0 while manipulating the registers to specify the interrupt processing in RAM.
2. Do not allocate the interrupt branch destination to the saddr space (FFE20H to FFEFFH).
3. When executing instructions in a RAM area and enabling generation of a RAM parity error reset (RPERDIS = 0), be sure to initialize the RAM area to be used + 10 bytes.

(2) Changing the interrupt branch destination from the RAM address back to the interrupt vector addresses in ROM

The following shows the procedure for changing the interrupt branch destination back to the addresses pointed to by the interrupt vector table in ROM (initial state).

Target function of this operation: R_RFD_RestoreInterruptVector

Operation Procedure:

- Save the current interrupt enabled or disabled setting and then disable interrupts.
- Execute the specific sequence to set the FWEDIS bit (bit 3) of the FLPMC register to 1.

Step 1:	PFCMD register = 0xA5
Step 2:	FLPMC register = 0x08
Step 3:	FLPMC register = 0xF7
Step 4:	FLPMC register = 0x08

- Steps 2 and 4
  FLPMC register setting (0x08)
  EEEMD (bit 4) = 0, FWEDIS (bit 3) = 1,
  FLSPM (bit 1) = 0
- Step 3
  Inverted value or FLPMC register setting (0xF7)

- Set the VECTCTRL register to 0x00 so that execution after interrupts branches to the addresses pointed to by the interrupt vector table in ROM.

Restore the saved interrupt enabled or disabled setting.

4.8 Examples of Command Execution for Reprogramming of Flash Areas

4.8.1 Example of Command Execution for Reprogramming of the Code Flash Area

Figure 4-3 shows a flowchart of command execution for reprogramming of the code flash area.

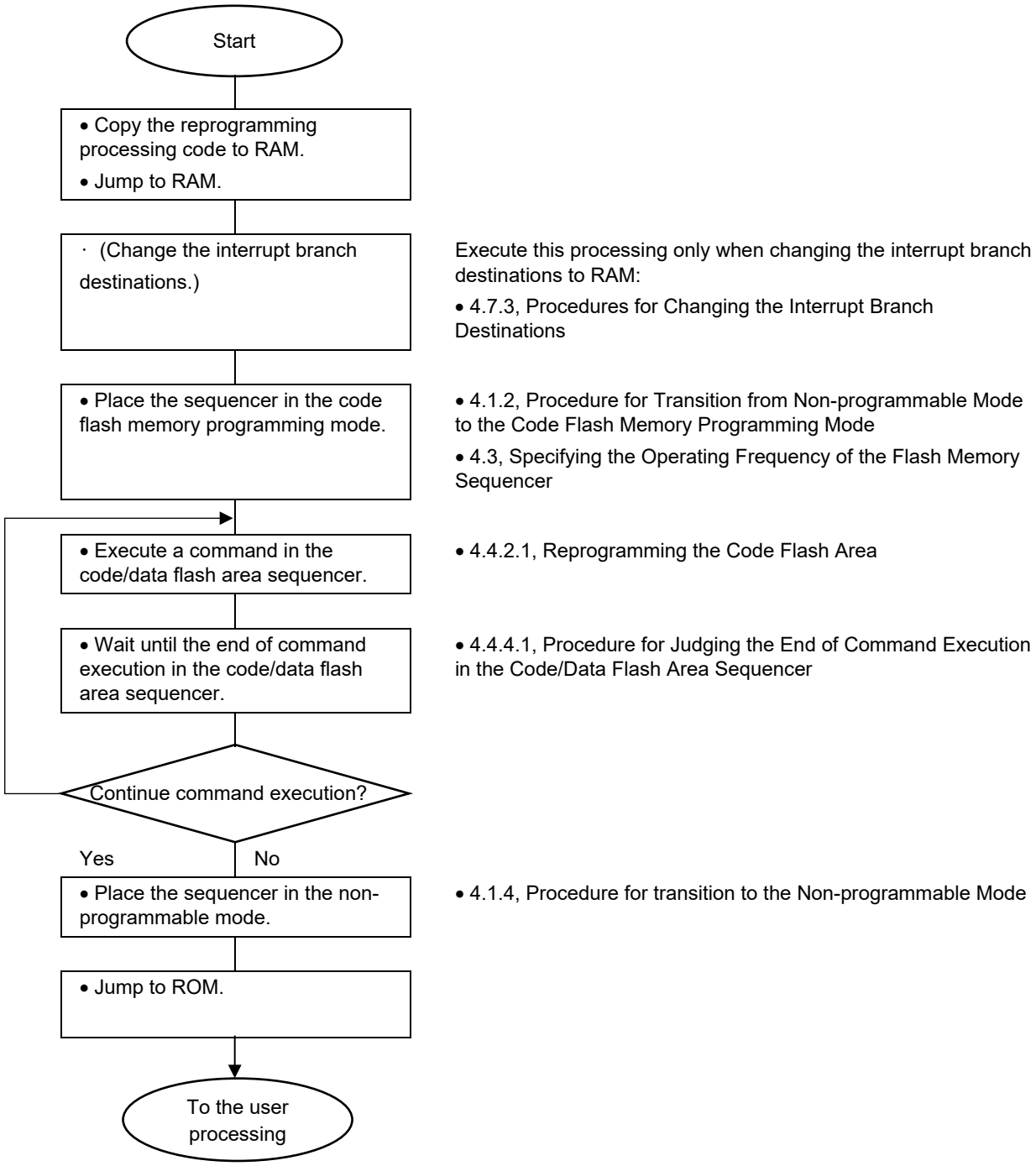


Figure 4-3 Flowchart of Command Execution for Reprogramming of the Code Flash Area

4.8.2 Example of Command Execution for Reprogramming of the Data Flash Area

Figure 4-4 shows a flowchart of command execution for reprogramming of the data flash area.

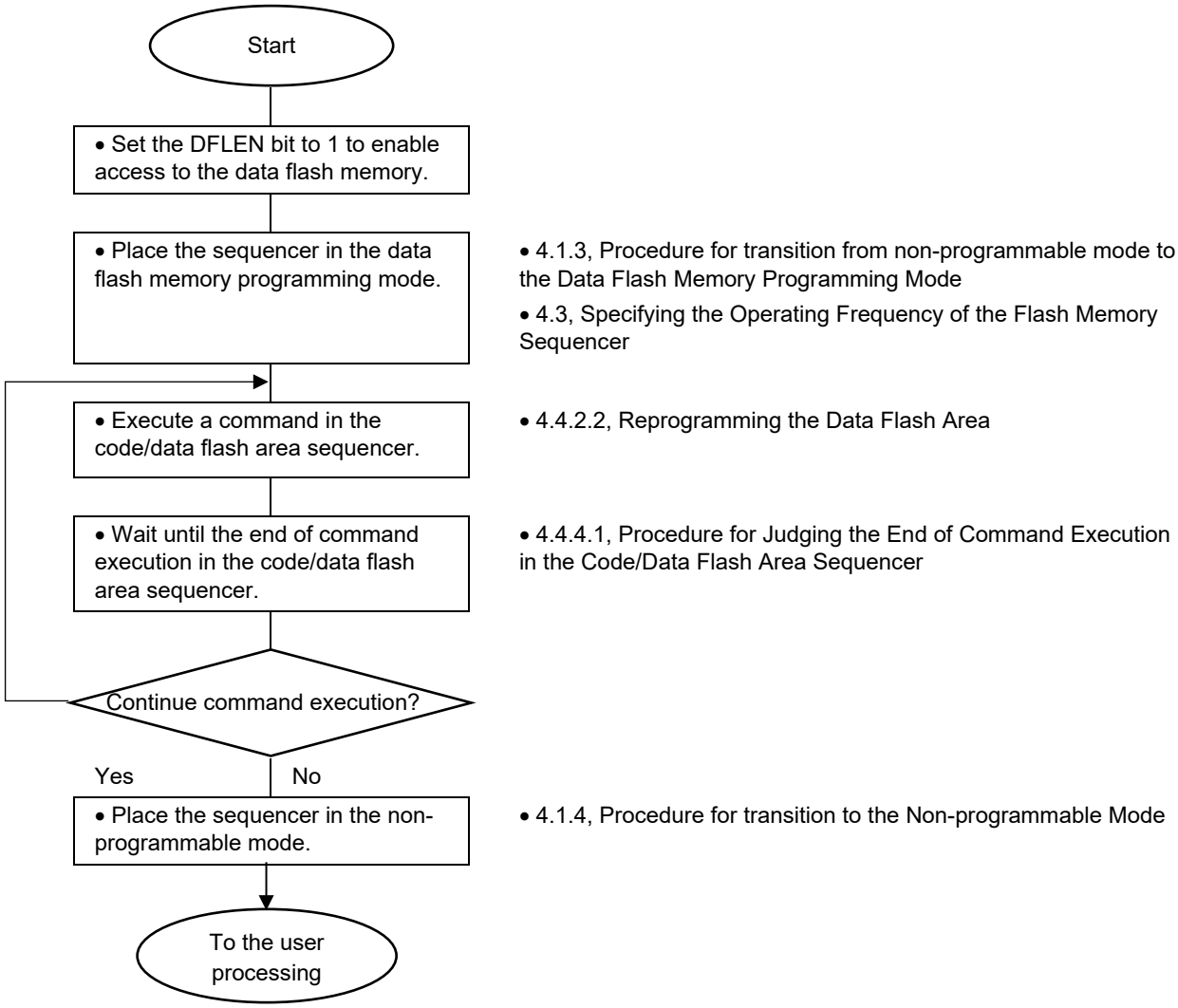


Figure 4-4 Flowchart of Command Execution for Reprogramming of the Data Flash Area

4.8.3 Example of Command Execution for Reprogramming of the Extra Area

Figure 4-5 shows a flowchart of command execution for reprogramming of the extra area.

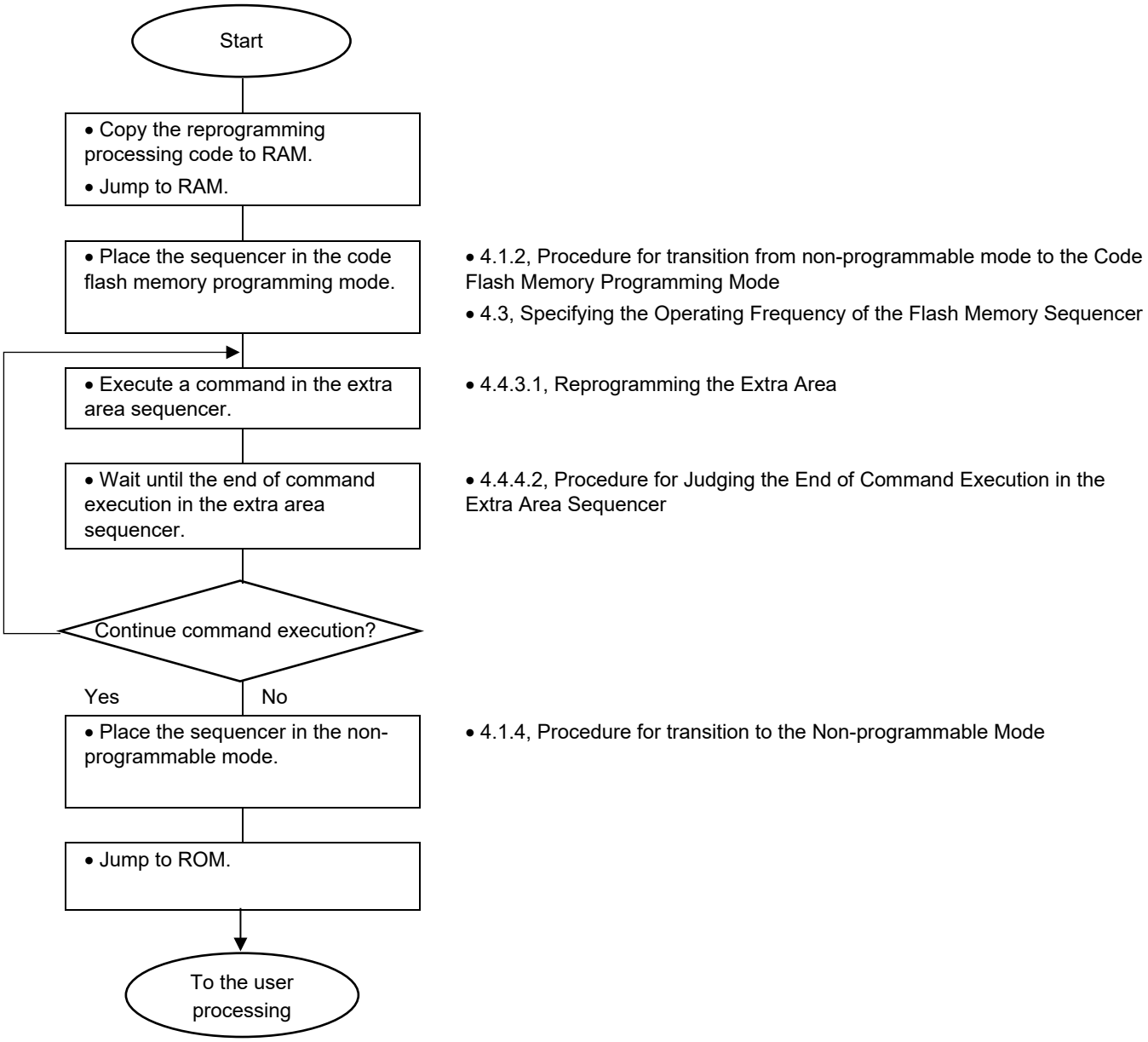


Figure 4-5 Flowchart of Command Execution for Reprogramming of the Extra Area

4.8.4 Example of Command Execution for Controlling Bank Programming

Figure 4-6 shows a flowchart of command execution for bank programming.

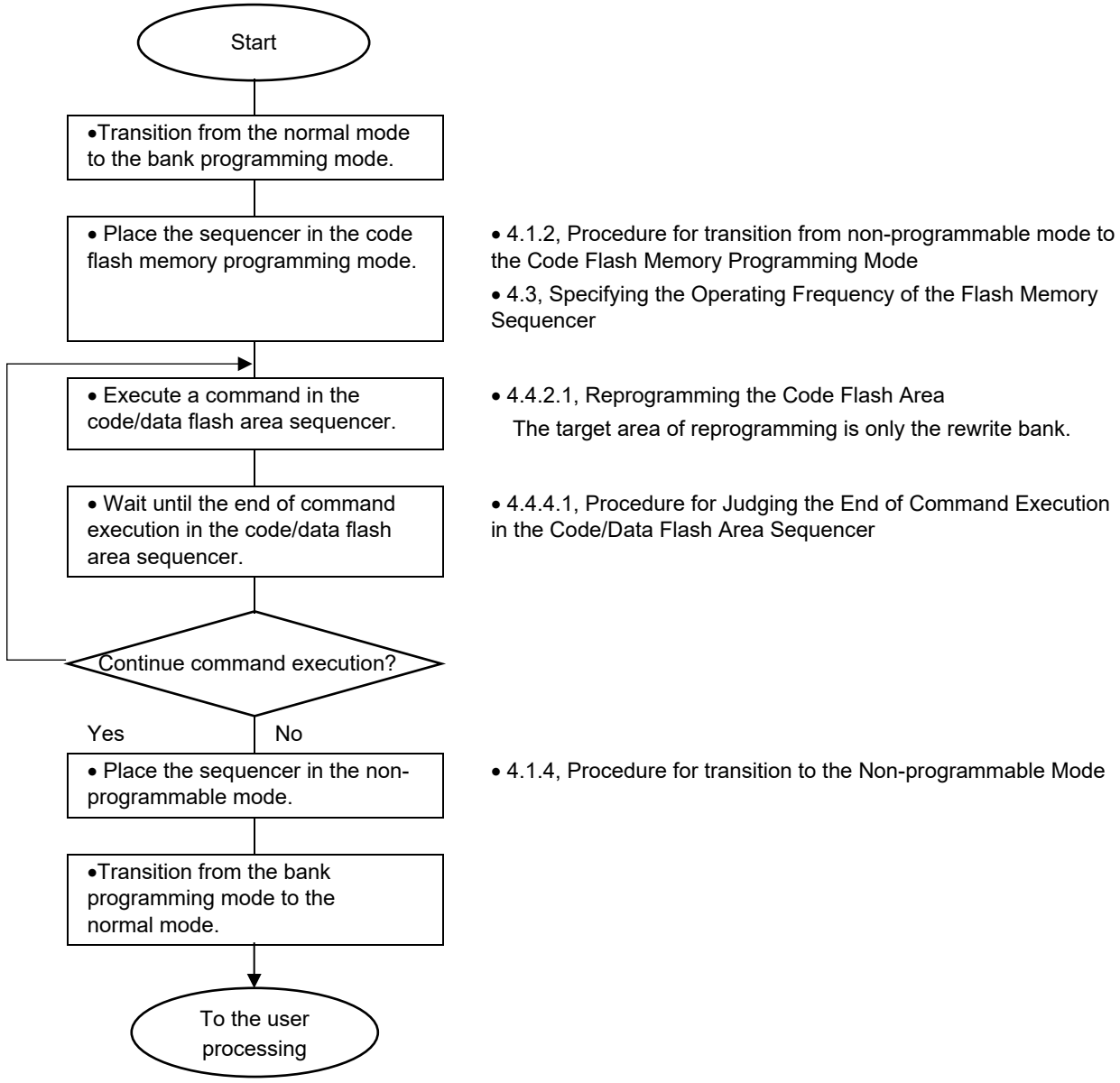


Figure 4-6 Flowchart of Command Execution for Bank Programming

5. Sample Programs

This section describes the sample programs provided together with RFD RL78 Type 11.

5.1 File Structure

5.1.1 Folder Structure

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

Figure 5-1 shows an example of using RL78/L23. The installed “sample” folder contains a folder for each device group (e.g. RL78_L23).

The “RL78_L23” folder is used when using RL78/L23.

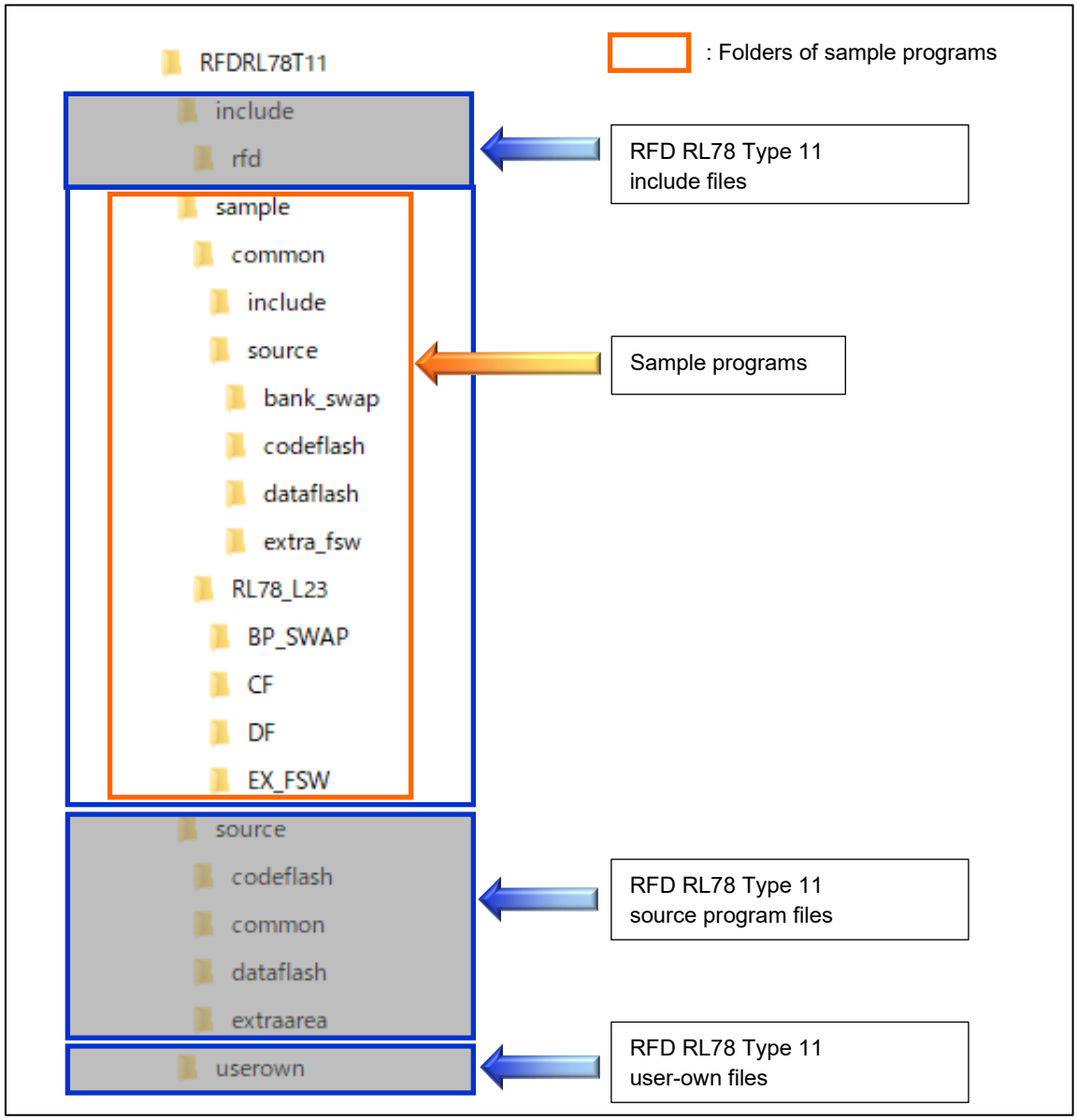


Figure 5-1 Structure of Sample Program Folders

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\dataflash\” folder.

Table 5-1 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 5-2 shows the program source file in the “sample\common\source\codeflash\” folder.

Table 5-2 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 5-3 shows the program source file in the “sample\common\source\extra_fsw\” folder.

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

No.	Source File Name	Description
1	sample_control_extra_fsw.c	This file contains the functions for controlling the FSW in the extra area.

Table 5-4 shows the program source file in the “sample\common\source\bank_swap\” folder.

Table 5-4 Program Source File in the “sample\common\source\bank_swap\” Folder

No.	Source File Name	Description
1	sample_control_bank_programming.c	This file contains the functions for controlling the bank programming.
2	sample_control_bank_swap.c	This file contains the functions for controlling the bank swapping.

Table 5-5 shows the program source files of the main processing for controlling the code flash memory (CF), data flash memory (DF), and FSW in the extra area (EX_FSW) in the “sample\RL78_L23” folder.

- Main processing for controlling the code flash memory (CF):
  “sample\RL78_L23\CF\[compiler name]\source\” folder
- Main processing for controlling the data flash memory (DF):
  “sample\RL78_L23\DF\[compiler name]\source\” folder
- Main processing for controlling the FSW in the extra area (EX_FSW):
  “sample\RL78_L23\EX_FSW\[compiler name]\source\” folder
- Main processing for controlling the bank programming / bank swapping (BP_SWAP):
  “sample\RL78_L23\BP_SWAP\[compiler name]\source\” folder

Table 5-5 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
3	main.c (for FSW control in extra area)	Sample file of the main processing functions for controlling the extra area (FSW function)
4	main.c (for bank programming and bank swapping control)	Sample file of the main processing functions for bank programming and bank swapping

5.1.2.2 List of Header Files

Table 5-6 shows the program header files in the “sample\common\include\” folder.

Table 5-6 Program Header Files in the “sample\common\include\” Folder

No.	Header File Name	Description
1	sample_control_data_flash.h	This file defines the prototype declarations of the sample functions for controlling the data flash memory.
2	sample_control_code_flash.h	This file defines the prototype declarations of the sample functions for controlling the code flash memory.
3	sample_control_extra_fsw.h	This file defines the prototype declarations of the sample functions for controlling the FSW in the extra area.
4	sample_control_bank_programming.h	This file defines the prototype declarations of the sample functions for controlling the bank programming.
5	sample_control_bank_swap.h	This file defines the prototype declarations of the sample functions for controlling the bank swapping.
6	sample_defines.h	This file defines the macros of the sample functions for controlling the flash memory.
7	sample_memmap.h	This file defines the macros that describes the sections used by the sample program that controls the flash memory.
8	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 5-7 shows the results (normal end or error) of execution in the flash memory sequencer and the status after execution.

Table 5-7 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	Code/data flash area sequencer error
SAMPLE_ENUM_RET_ERR_EXTRA_SEQUENCER	0x21u	Extra area 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_ACT_BLANKCHECK	0x24u	Blank check operation error
SAMPLE_ENUM_RET_ERR_CMD_ERASE	0x30u	Erase command error
SAMPLE_ENUM_RET_ERR_CMD_WRITE	0x31u	Write command error
SAMPLE_ENUM_RET_ERR_CMD_BLANKCHECK	0x32u	Blank check command error
SAMPLE_ENUM_RET_ERR_CMD_SET_EXTRA_AREA	0x33u	Extra area command setting error

5.2.2 Macro Definitions

5.2.2.1 User Definition Macro for Sample Program of RFD RL78 Type 11

- Macro for selecting “active bank swapping execution” (Immediate execution bank swapping)

Symbol Name	Description
SMP_BP_SWAP_IMMEDIATELY	This macro needs to be defined when “active bank swapping execution” (Immediate execution bank swapping) is selected.

Note: Be sure to define macro using this compile option. If this macro is not defined, “bank swapping execution after reset” is selected. Refer to “6.1.3.2 The Setting of User Definition Macro (CC-RL Compiler)“, “6.2.3.2 The Setting of User Definition Macro (IAR Compiler)“ or “6.3.3.2 The Setting of User Definition Macro (LLVM Compiler)“

5.3 Sample Program Functions

Table 5-8 shows the sample program functions.

Table 5-8 List of Sample Program Functions

	API Function Name	Outline
1	main (for code flash)	Executes the main processing of the sample program for controlling the reprogramming of the code flash memory.
2	Sample_CodeFlashControl	Executes the processing for reprogramming the code flash memory.
3	main (for data flash)	Executes the main processing of the sample program for controlling the reprogramming of the data flash memory.
4	Sample_DataFlashControl	Executes the processing for reprogramming the data flash memory.
5	main (for FSW control in extra area)	Executes the main processing of the sample program for controlling the reprogramming of the extra area (FSW function settings).
6	Sample_ExtraFSWControl	Executes the processing for reprogramming the extra area (FSW function settings).
7	main (for bank programming / bank swapping control)	Executes the main processing of the sample program for controlling the bank programming and bank swapping.
8	Sample_BankProgrammingControl	Executes the processing for reprogramming the rewrite bank area. - The blank check, erase, and write commands are executed in the code flash memory programming mode. - The written data are read in the non-programmable mode to check that the data have been written correctly
9	Sample_BankSwapControl	Executes the processing for reprogramming the extra area (boot area switching flag settings) and bank swapping. - The write command for the extra area (Programming of the security flags and the boot area switching flag command) is executed in the code flash memory programming mode. - A bank swap is executed after reset or immediately. The case of [Bank swapping executes after reset]: The sequencer is placed in the non-programmable mode and CPU reset will occur. After the bank swapping is executed, the exchanged program of the startup bank is executed. The case of [active bank swapping execution (Immediate execution bank swapping)]: After the sequencer shifts to non-programming mode, it shifts to code flash memory programming mode. After active bank swapping is executed, the sequencer shifts to non-programming mode. And user processing jumps to the specified address of exchanged startup bank.
10	Sample_CheckCFSeqEnd	Waits for the completion of command execution in the code/data flash area sequencer for the code flash reprogramming.
11	Sample_CheckDFSeqEnd	Waits for the completion of command execution in the code/data flash area sequencer for the data flash reprogramming.
12	Sample_CheckExtraSeqEnd	Waits for the completion of command execution in the extra area sequencer for the extra area reprogramming.

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 RFD RL78 Type 11 erases block 14 (0x00007000) in the code flash area and writes 64-word (256 bytes) data from the beginning of the block.

Note: In the code flash memory programming mode, cannot refer to the programs and data in the code flash memory. Copy the Sample_CodeFlashControl function and the processing to be executed and data to be referenced within the function to RAM in advance, and execute and reference them in RAM.
Refer to "5.3.4 Sample Program for Bank Programming / Bank Swapping Control" for the case where bank programming is executed.

Operating conditions (Example of a sample program for RL78/L23):

- CPU operating frequency: 32 MHz (The high-speed on-chip oscillator clock is used for the main system clock.)
- Code flash memory address for erasure and programming: 0x00007000
- Block number for erasure: 0x000E
- 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 RFD RL78 Type 11.

5.3.1.1 main Function

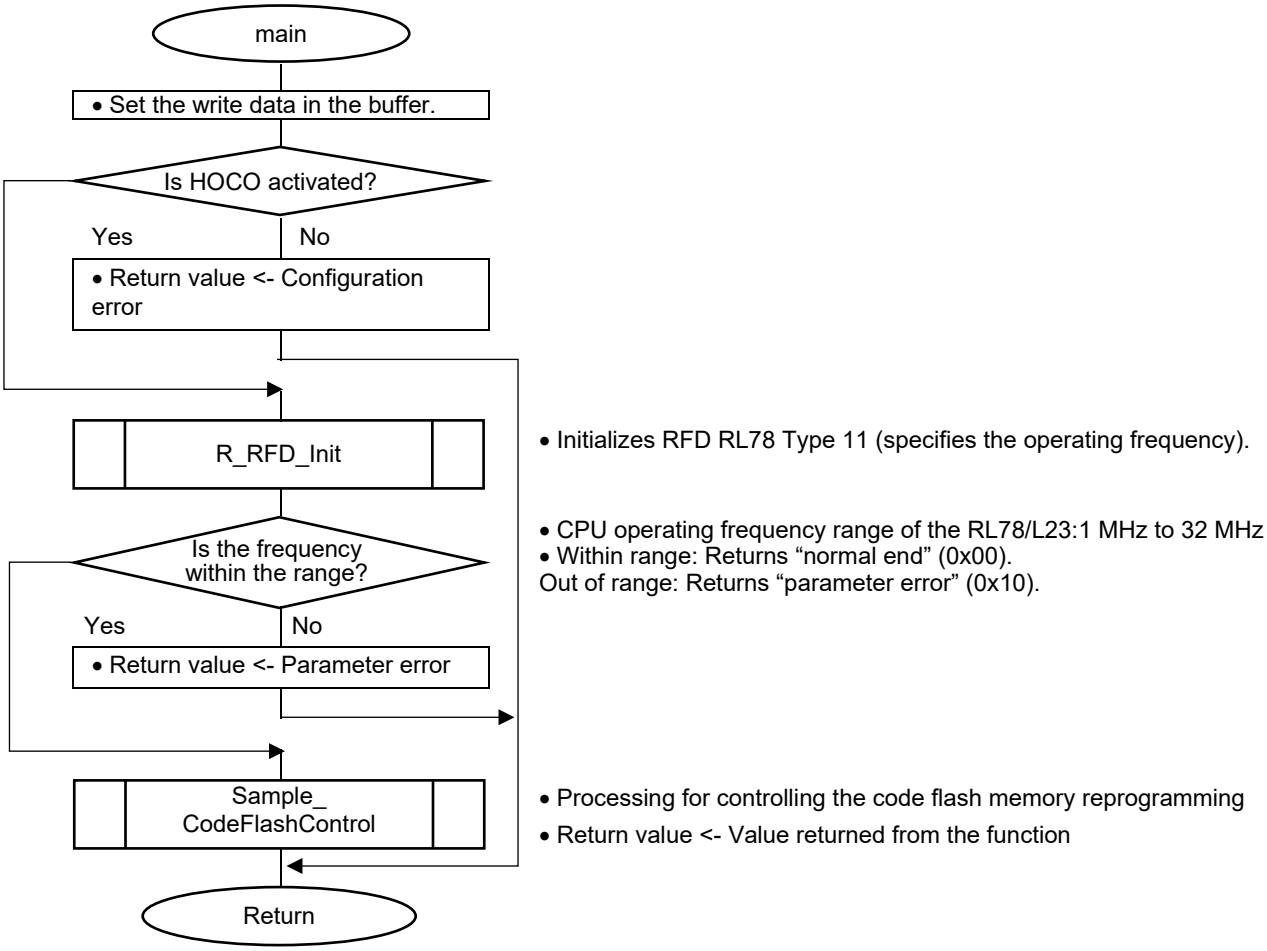


Figure 5-2 Flowchart of the Main Processing of the Sample Program for Controlling Code Flash Memory Reprogramming

5.3.1.2 Sample_CodeFlashControl Function

- The sequencer is placed in the code flash memory programming mode and the blank check and block erasure are executed.

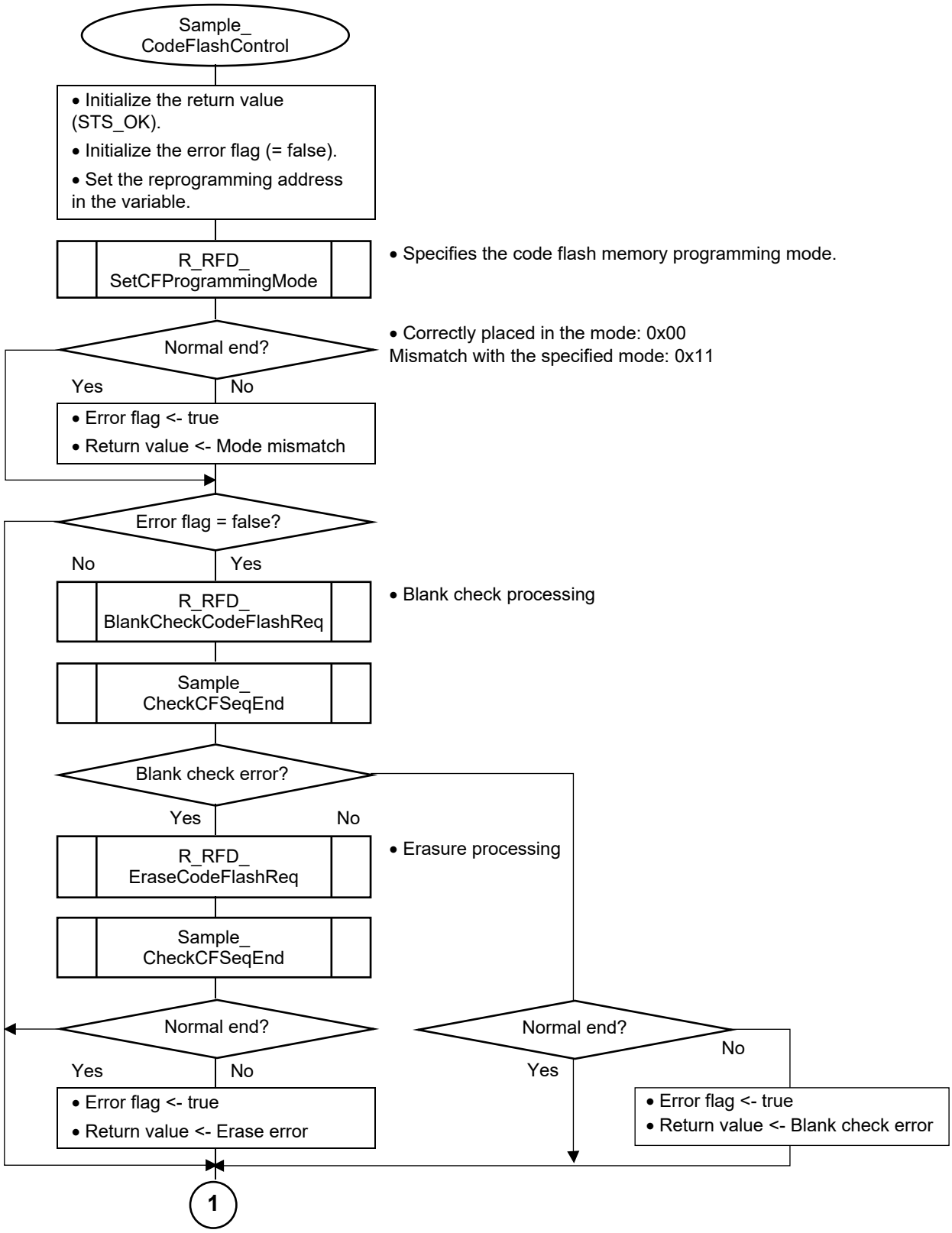


Figure 5-3 Flowchart of Sample Processing for Controlling Code Flash Memory Reprogramming (1/3)

- Programming is executed.

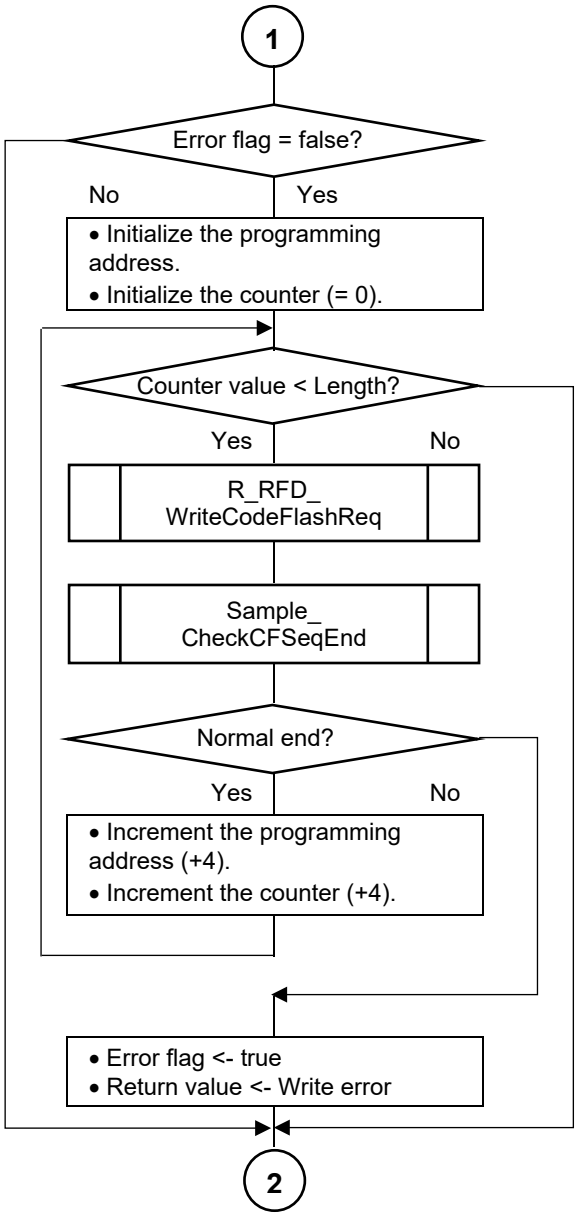
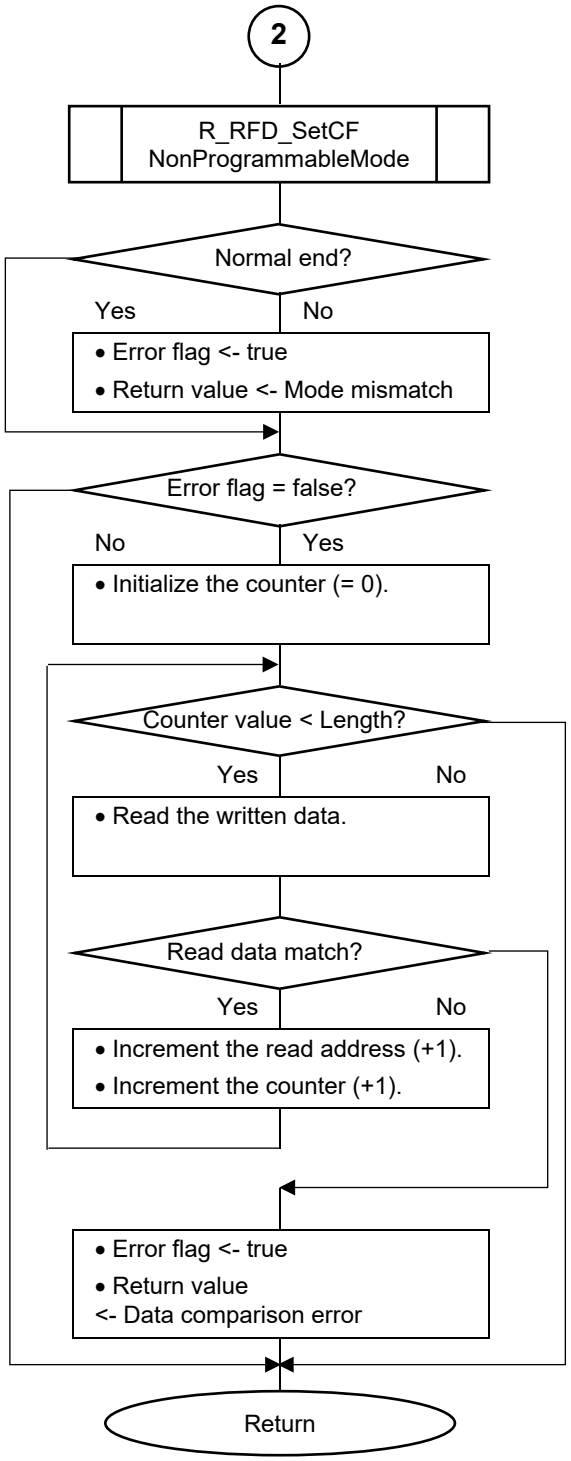


Figure 5-4 Flowchart of Sample Processing for Controlling Code Flash Memory Reprogramming (2/3)

- 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

• 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 RFD RL78 Type 11 erases block 0 (0x000F1000) in the data flash area and writes 64-byte data from the beginning of the block.

Note: In the data flash memory programming mode, the data in the data flash memory cannot be referenced. Copy the Sample_DataFlashControl function and the data to be referenced within the function to RAM in advance, and reference them in RAM.

Operating conditions (Example of a sample program for RL78/L23):

- CPU operating frequency: 32 MHz (The high-speed on-chip oscillator clock is used for the main system clock.)
- Data flash memory address for erasure and programming: 0x000F1000
- Block number for erasure: 0x0000
- Size of write data: 64 bytes

Figure 5-6 shows a flowchart of the main processing of the sample program for controlling the data flash memory reprogramming in RFD RL78 Type 11.

5.3.2.1 main Function

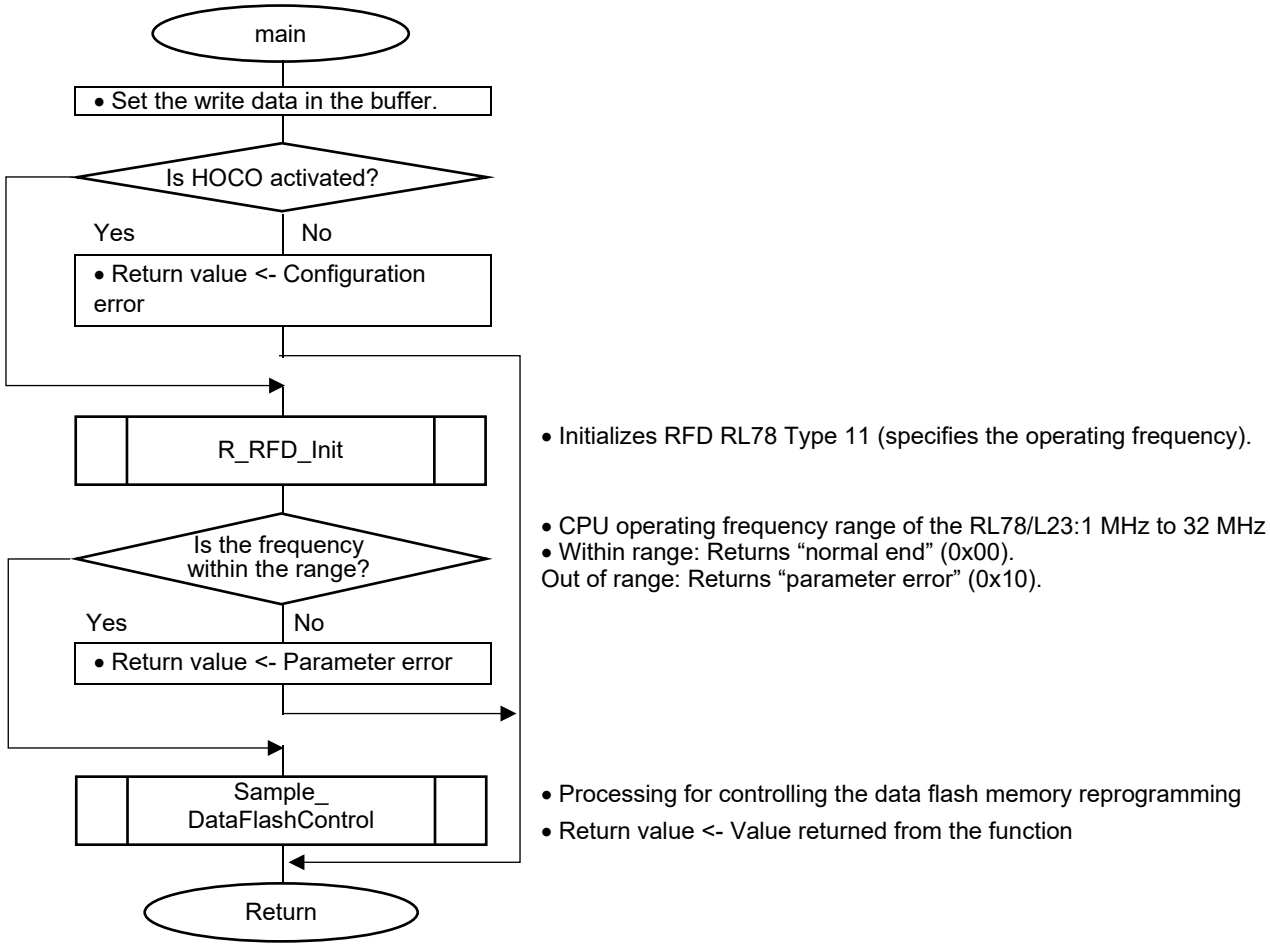


Figure 5-6 Flowchart of the Main Processing of the Sample Program for Controlling Data Flash Memory Reprogramming

5.3.2.2 Sample_DataFlashControl Function

- The sequencer is placed in the data flash memory programming mode and the blank check and block erasure are performed.

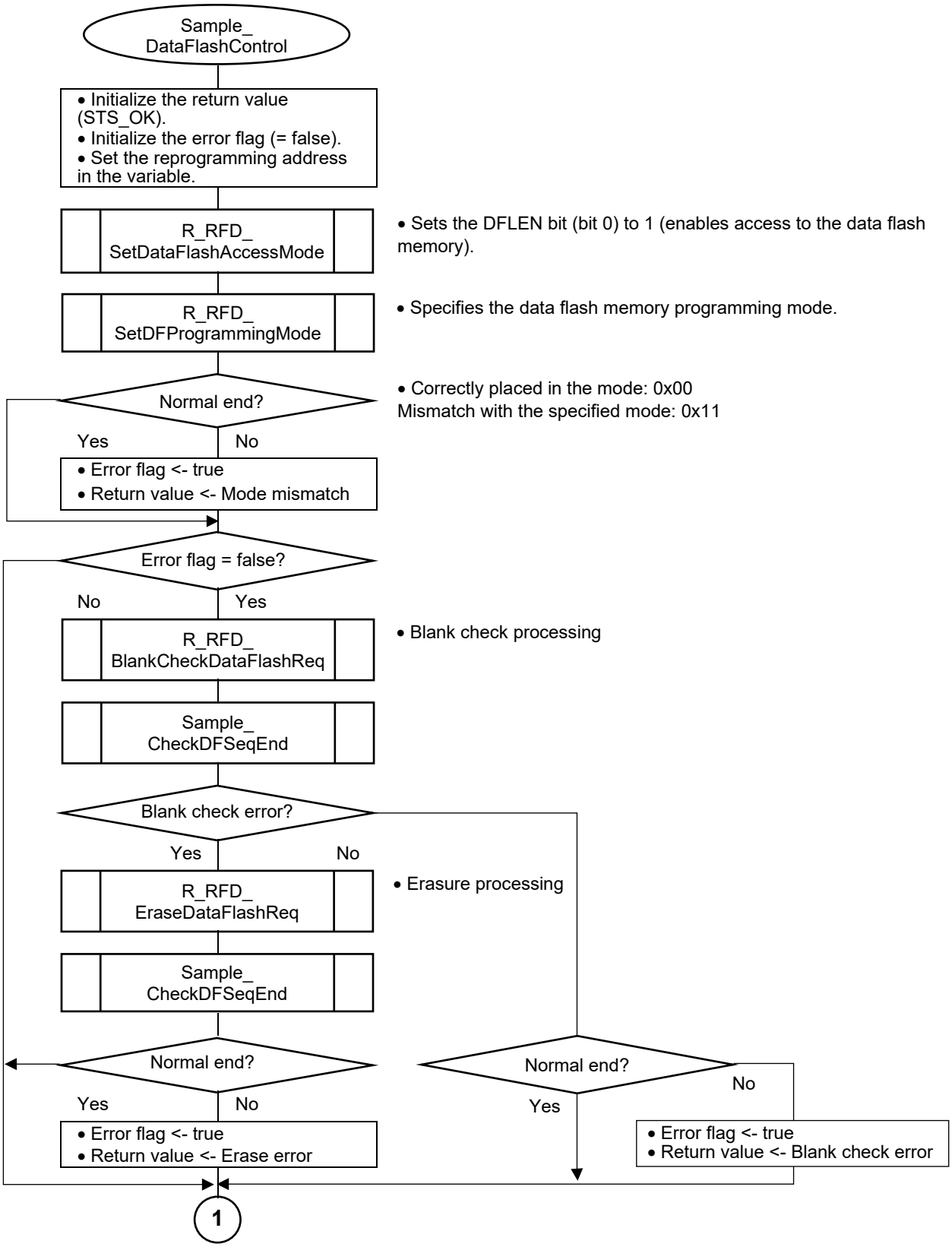


Figure 5-7 Flowchart of Sample Processing for Controlling Data Flash Memory Reprogramming (1/3)

- Programming is executed.

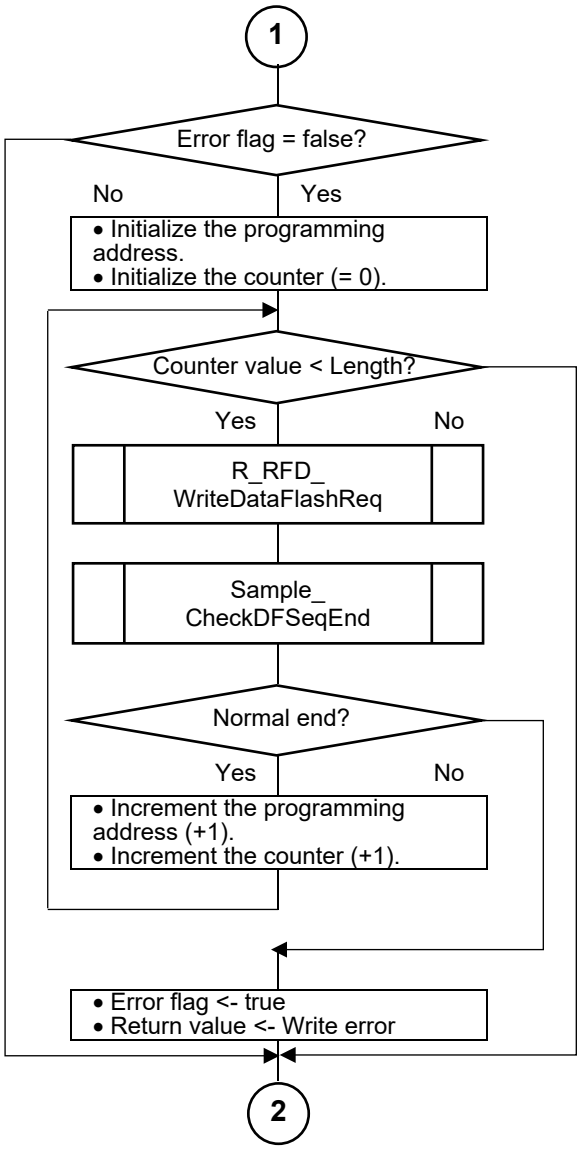


Figure 5-8 Flowchart of Sample Processing for Controlling Data Flash Memory Reprogramming (2/3)

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

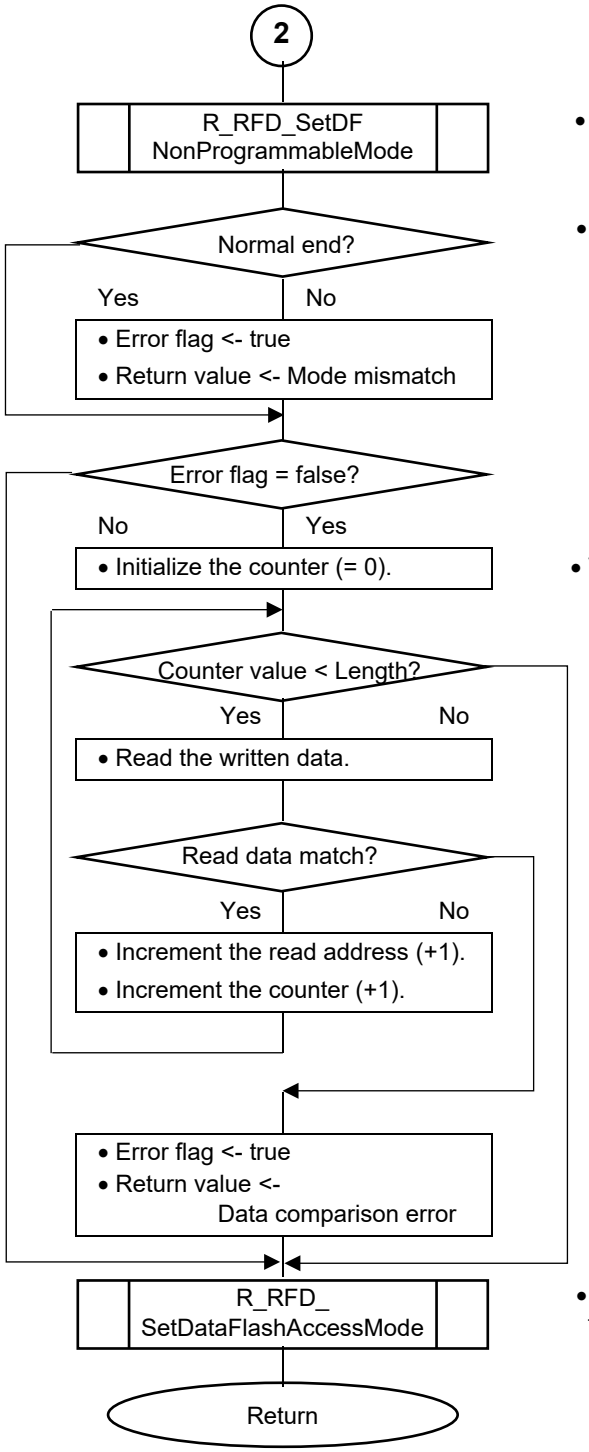


Figure 5-9 Flowchart of Sample Processing for Controlling Data Flash Memory Reprogramming (3/3)

5.3.3 Sample Program for Controlling the Reprogramming of the Extra Area

The sample program for controlling the reprogramming of the extra area in RFD RL78 Type 11 reprograms the 4-byte (32-bit) area used to control the flash shield window (FSW).

- FSWS (start block) = 0, FSWE (end block +1) = 256
  (Enables reprogramming of the entire area of the code flash memory.)
- FSWC (FSW area control) = 1 (outside shield area)

Note: In the code flash memory programming mode for reprogramming the extra area, cannot refer to the programs and data in the code flash memory. Copy the Sample_ExtraFSWControl function and the processing to be executed and data to be referenced within the function to RAM in advance, and execute and reference them in RAM.

Operating conditions (Example of a sample program for RL78/L23):

- CPU operating frequency: 32 MHz (The high-speed on-chip oscillator clock is used for the main system clock.)
- Area for programming: Extra area (FSW-related data)
- Size of write data: 4 bytes

Figure 5-10 shows a flowchart of the main processing of the sample program for controlling the extra area reprogramming in RFD RL78 Type 11.

5.3.3.1 main Function

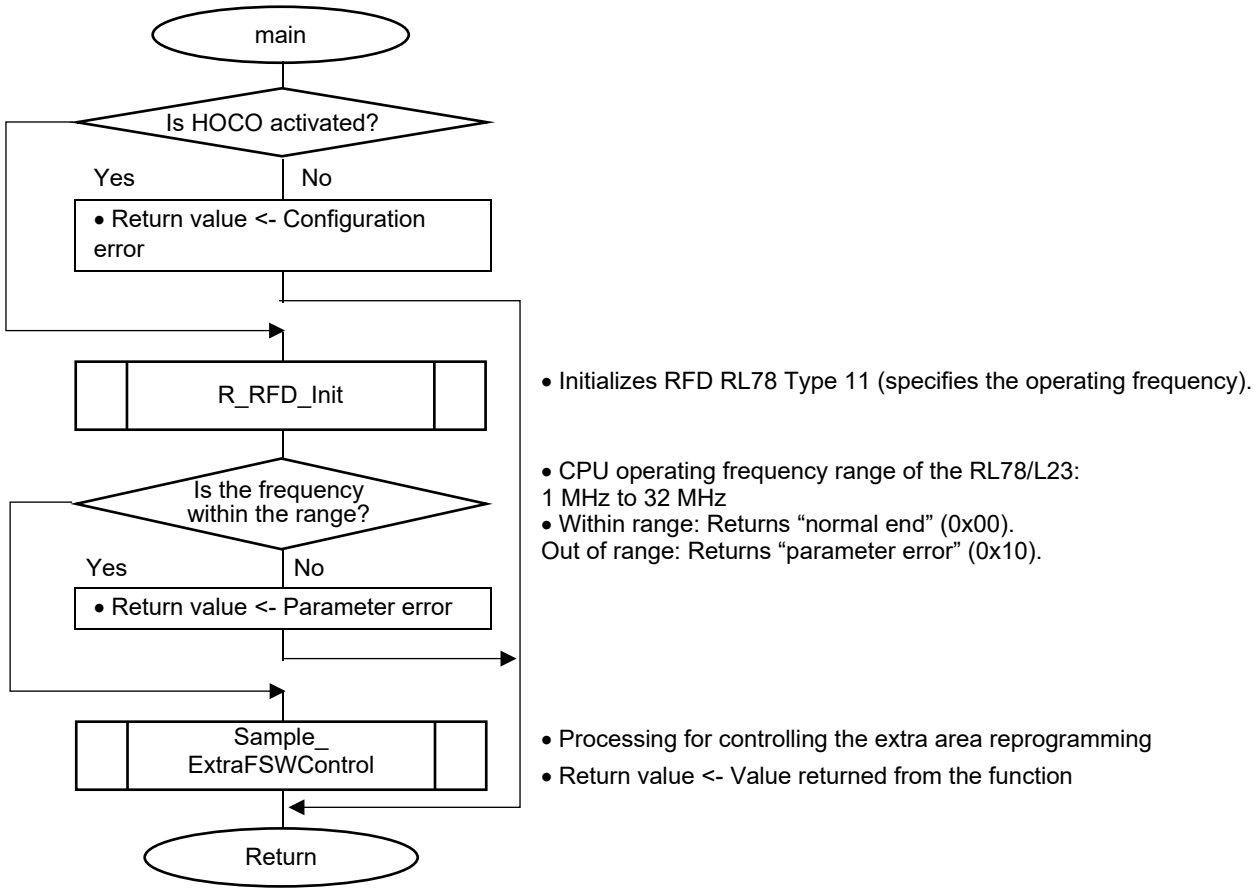


Figure 5-10 Flowchart of the Main Processing of the Sample Program for Controlling Extra Area (FSW) Reprogramming

5.3.3.2 Sample_ExtraFSWControl Function

- The sequencer is placed in the code flash memory programming mode and the FSW setting processing is performed.

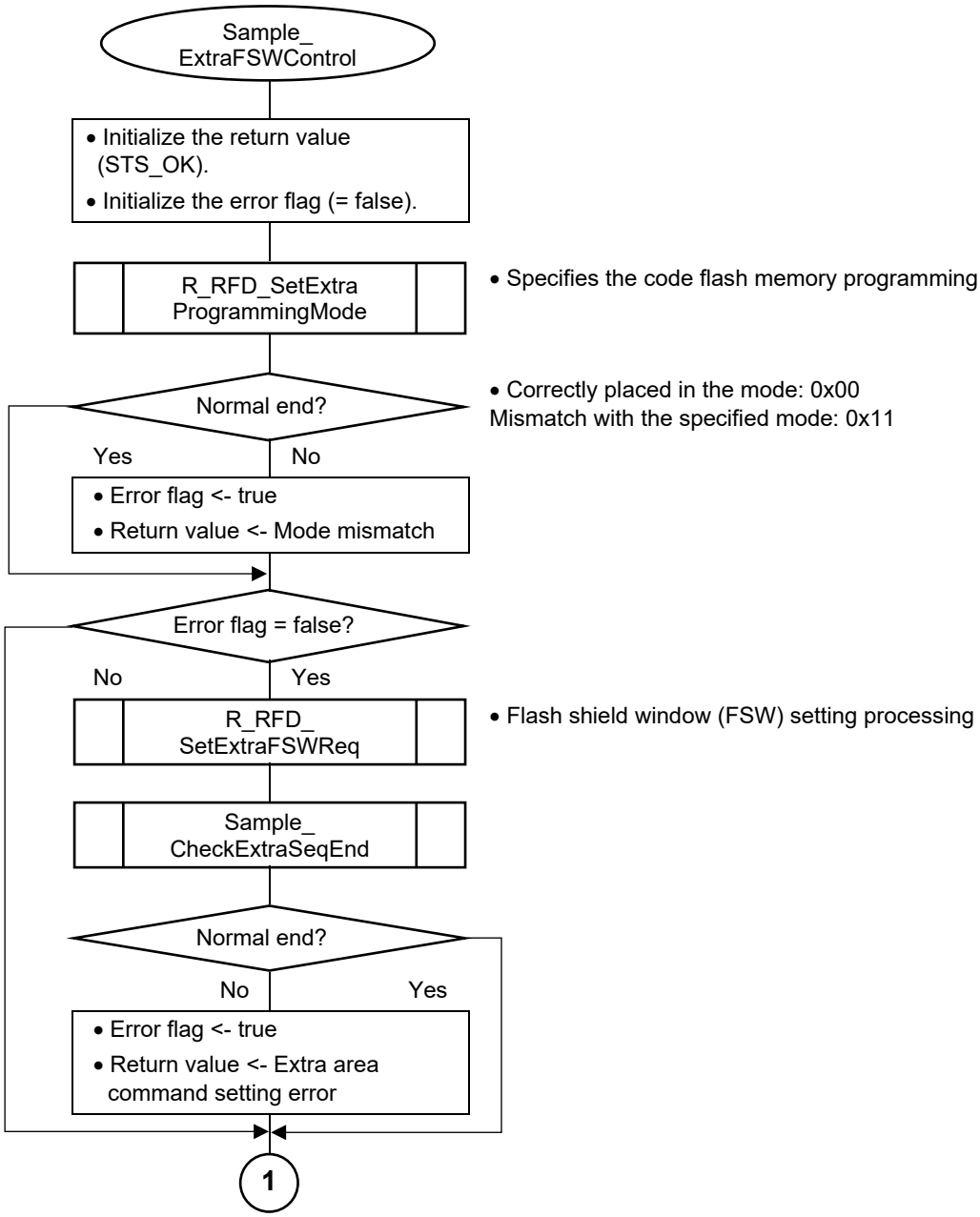
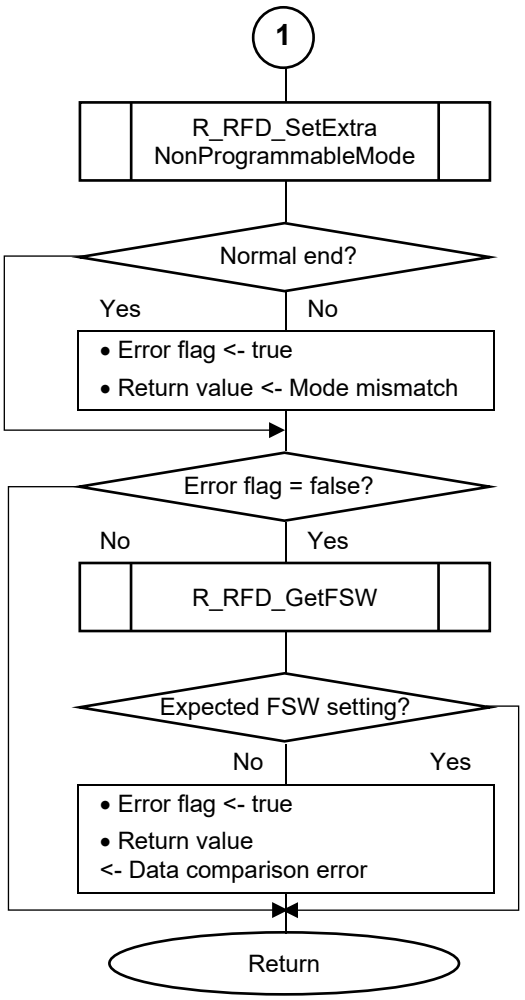


Figure 5-11 Flowchart of Sample Processing for Controlling Extra Area (FSW) Reprogramming (1/2)

- The sequencer is placed in the non-programmable mode and the FSW settings are read to check that the read settings match the expected values.



• Specifies the non-programmable mode.

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

• Read the start block and the end block+1 of the FSW, the FSW area, and the setting of protection against FSW modification.

Figure 5-12 Flowchart of Sample Processing for Controlling Extra Area (FSW) Reprogramming (2/2)

5.3.4 Sample Program for Bank Programming / Bank Swapping Control

This sample program executes bank programming control and bank swap control.
It is necessary to set the BTBLS bit to bank swapping by the dedicated flash memory programmer.

(1) Bank programming control

The mode shifts to bank programming mode and executes the user program which reprograms rewrite bank located to startup bank.

- The migration method to bank programming mode

The value of the FLMODE register can only be changed when the FLMWEN bit in the flash operating mode protect register (FLMWRP) is 1. After having changed the value of the FLMODE register, set the FLMWEN bit to 0. Set the FLMWEN bit to 1 before writing to the BANKPGEN bit.

FLMWRP register (After reset: 0x00):

7	6	5	4	3	2	1	0
—	—	—	—	—	—	—	FLMWEN
—	—	—	—	—	—	—	R/W

— The FLMWRP register is an 8-bit register for controlling access to the flash operating mode select register.
- When FLMWEN (bit 0) = 0 (value after reset) ： Rewriting the FLMODE register is disabled.
- When FLMWEN (bit 0) = 1 ： Rewriting the FLMODE register is enabled.

FLMODE register (After reset: 0x00/0x80/0xC0):

7	6	5	4	3	2	1	0
MODE1	MODE0	—	—	—	—	BANKPGEN	—
R/W	R/W	—	—	—	—	R/W	—

— The FLMODE register is an 8-bit register for controlling the flash operating modes.
- When BANKPGEN (bit 1) = 0 (value after reset) ： User mode (bank programing is disabled.)
- When BANKPGEN (bit 1) = 1 ： Bank programming mode (bank programming is enabled.)

(2) Bank swapping control

Bank swapping can be executed in two ways: “execution after reset” or “active bank swapping execution”. Even if unexpected momentary power failure or reset occurs after bank swapping is executed, the swapping state of this sample is maintained. As for the case which “active bank swapping execution” selects, reprogramming of the boot area switching flag of an extra area is executed before bank swapping. It is necessary to set user definition macro (“SMP_BP_SWAP_IMMEDIATELY”) in the case which selects “active bank swapping execution.”

Refer to “6.1.3.2 The Setting of User Definition Macro (CC-RL Compiler)“, “6.2.3.2 The Setting of User Definition Macro (IAR Compiler)“ or “6.3.3.2 The Setting of User Definition Macro (LLVM Compiler)“ for the setting method.

- Bank swapping executes after reset:

A boot area switching flag in the extra area is reprogrammed. And R_RFD_ForceReset function processing is executed, and startup bank and rewrite bank are replaced by causing internal reset of CPU. For how to execute, see section 4.5.3.3, Bank Swapping Execution after Reset.

- Active bank swapping execution (The case of the “SMP_BP_SWAP_IMMEDIATELY” macro definition set.)

The specified bank is immediately allocated to the startup bank (0x00000~) (bank swapping is performed immediately). For how to execute, see section 4.5.3.4, Active Bank Swapping Execution (Immediate execution bank swapping).

Note: In the code flash memory programming mode for reprogramming the extra area, cannot refer to the programs and data in the code flash memory. Copy the Sample_BankSwapControl function and the processing to be executed and data to be referenced within the function to RAM in advance, and execute and reference them in RAM.

Operating conditions (Example of a sample program for RL78/L23):

- CPU operating frequency: 32 MHz (The high-speed on-chip oscillator clock is used for the main system clock.)
- Reprogramming area 1 (Bank programming control execution):
  Rewrite bank top address (Reset Vector[+ 0x0000], Option Byte[+ 0x00C0], OCD Option Byte[+ 0x00C4])
  Rewrite bank top address + 0x5000 (The user program after bank swapping.)
- Reprogramming area 2 (Bank swapping control execution):
  Extra area (Boot area switching flag)

Figure 5-13 shows a flowchart of the main processing of the sample program for bank programming / bank swapping control in RFD RL78 Type 11.

5.3.4.1 main Function

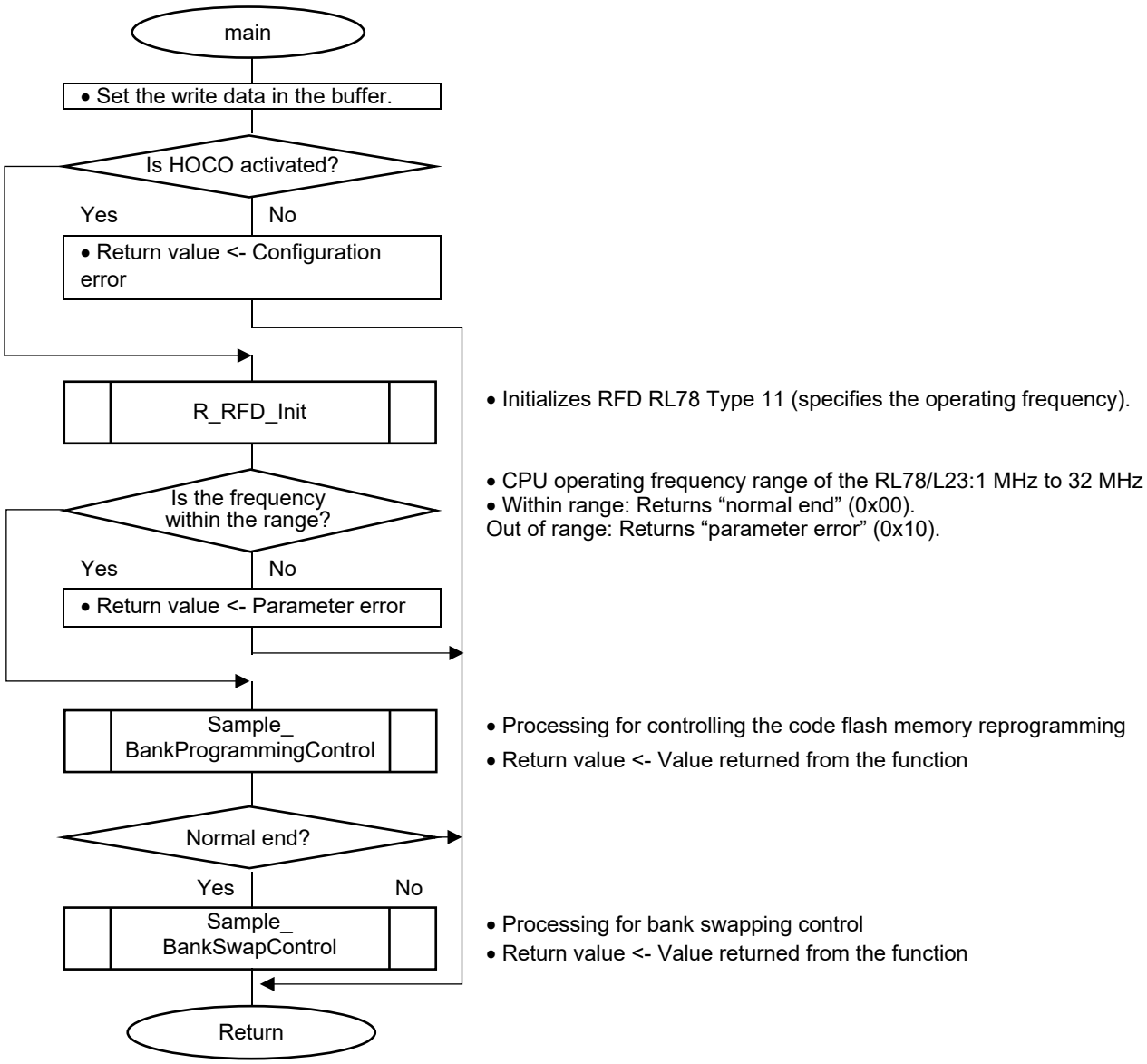


Figure 5-13 Flowchart of the Main Processing of the Sample Program for Bank Programming / Bank Swapping Control

5.3.4.2 Sample_BankProgrammingControl Function

- The sequencer is placed in the code flash memory programming mode and the blank check and block erasure are executed.

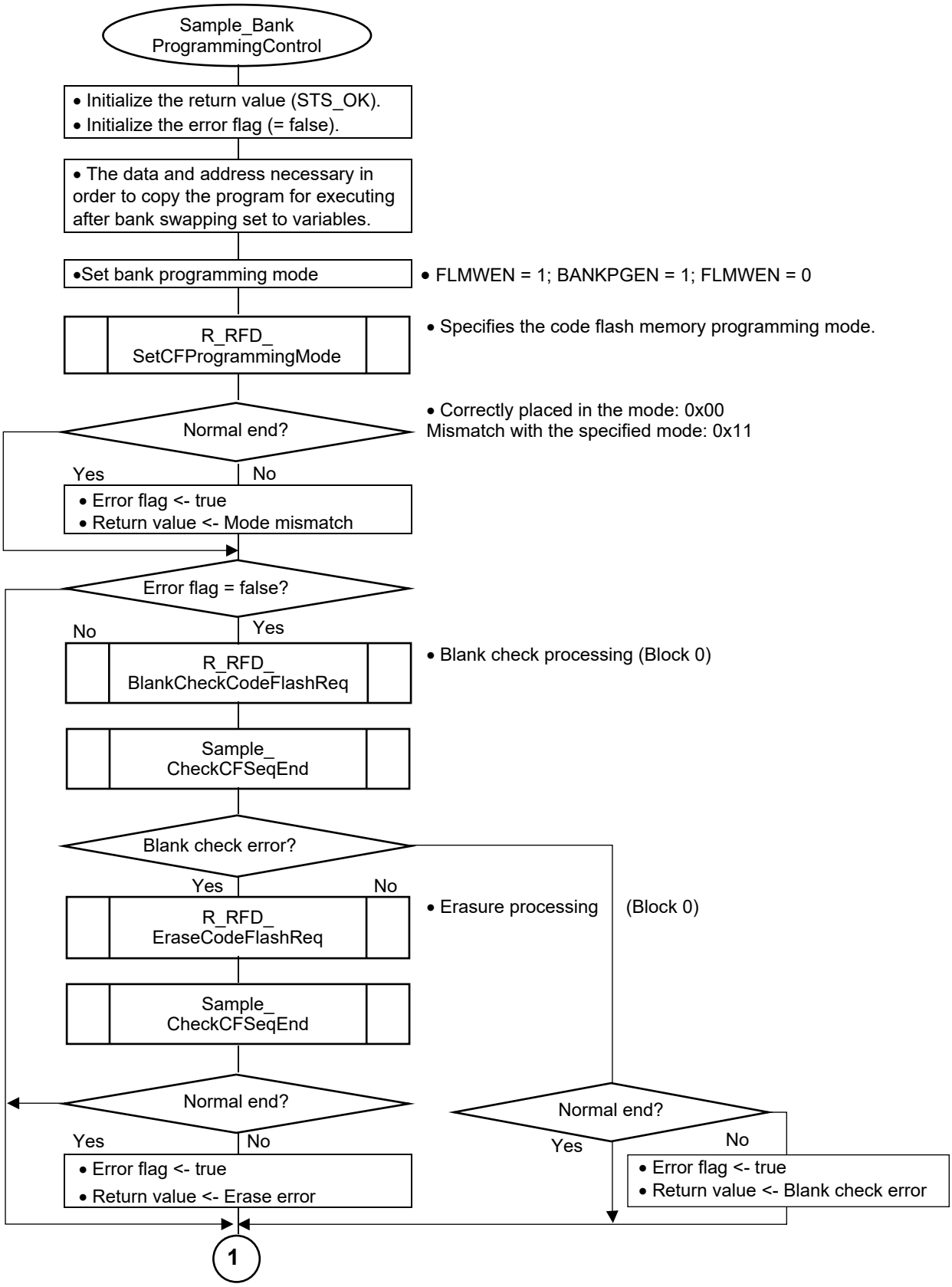


Figure 5-14 Flowchart of Sample Processing for Controlling Bank Programming (1/5)

- Programming of reset vector and option bytes is executed.

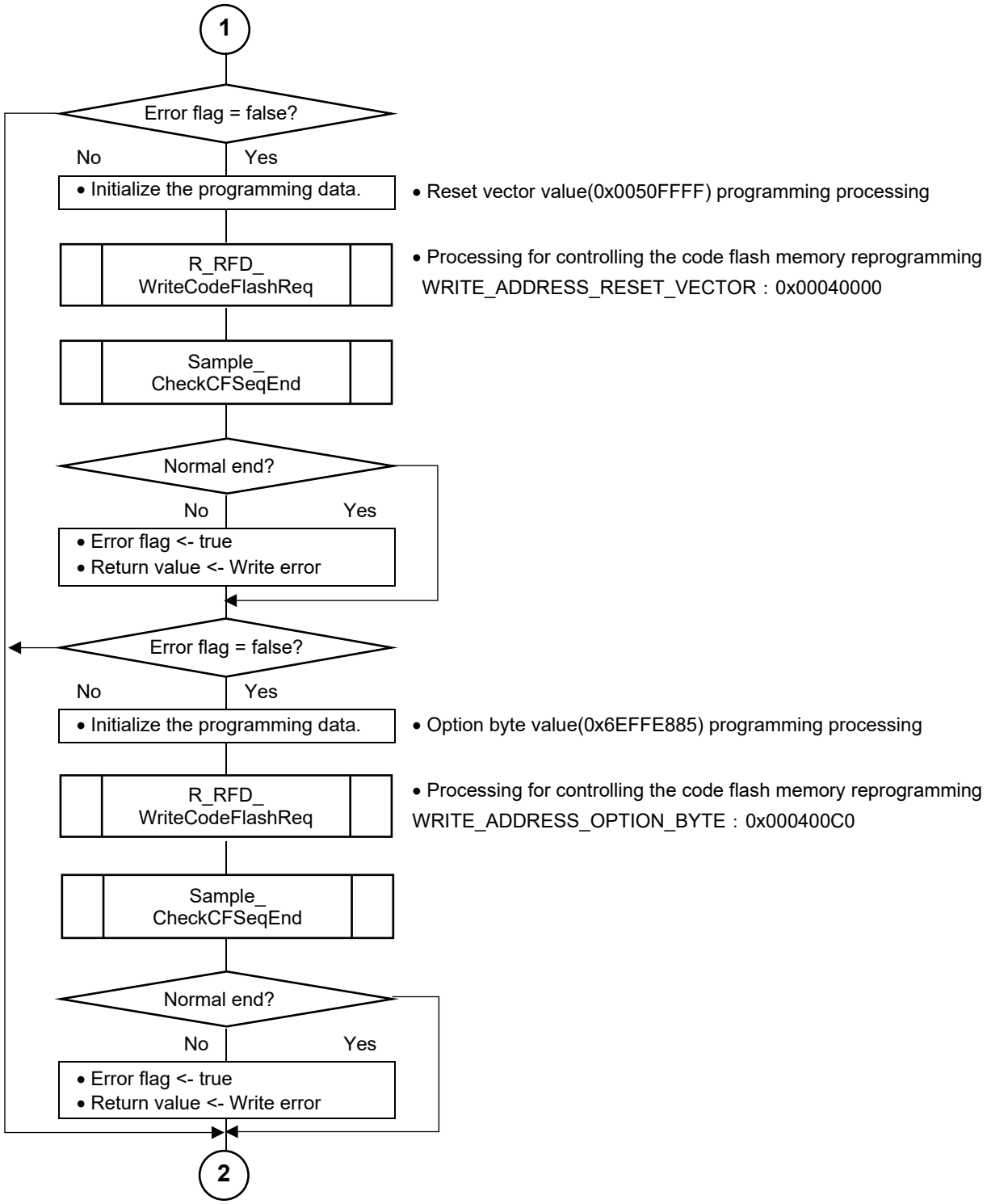


Figure 5-15 Flowchart of Sample Processing for Controlling Bank Programming (2/5)

- Programming of on-chip debug ID is executed.

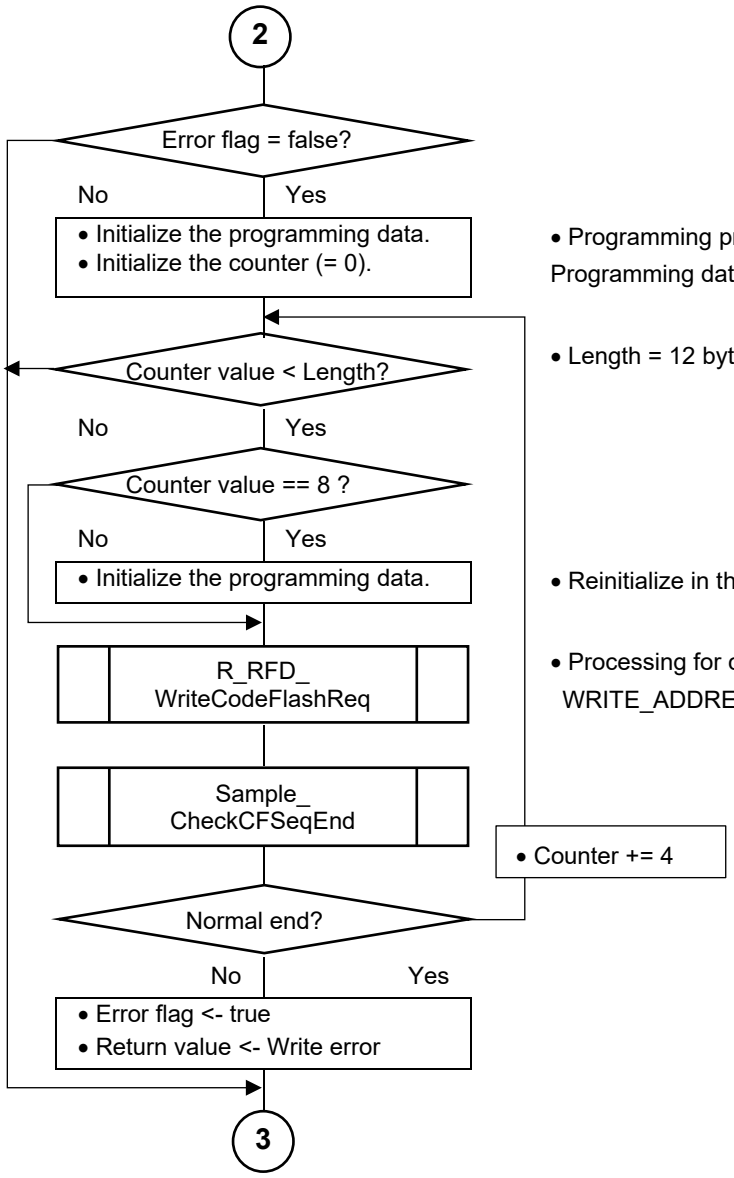


Figure 5-16 Flowchart of Sample Processing for Controlling Bank Programming (3/5)

- Reprogramming of the user program for Rewrite bank is executed.

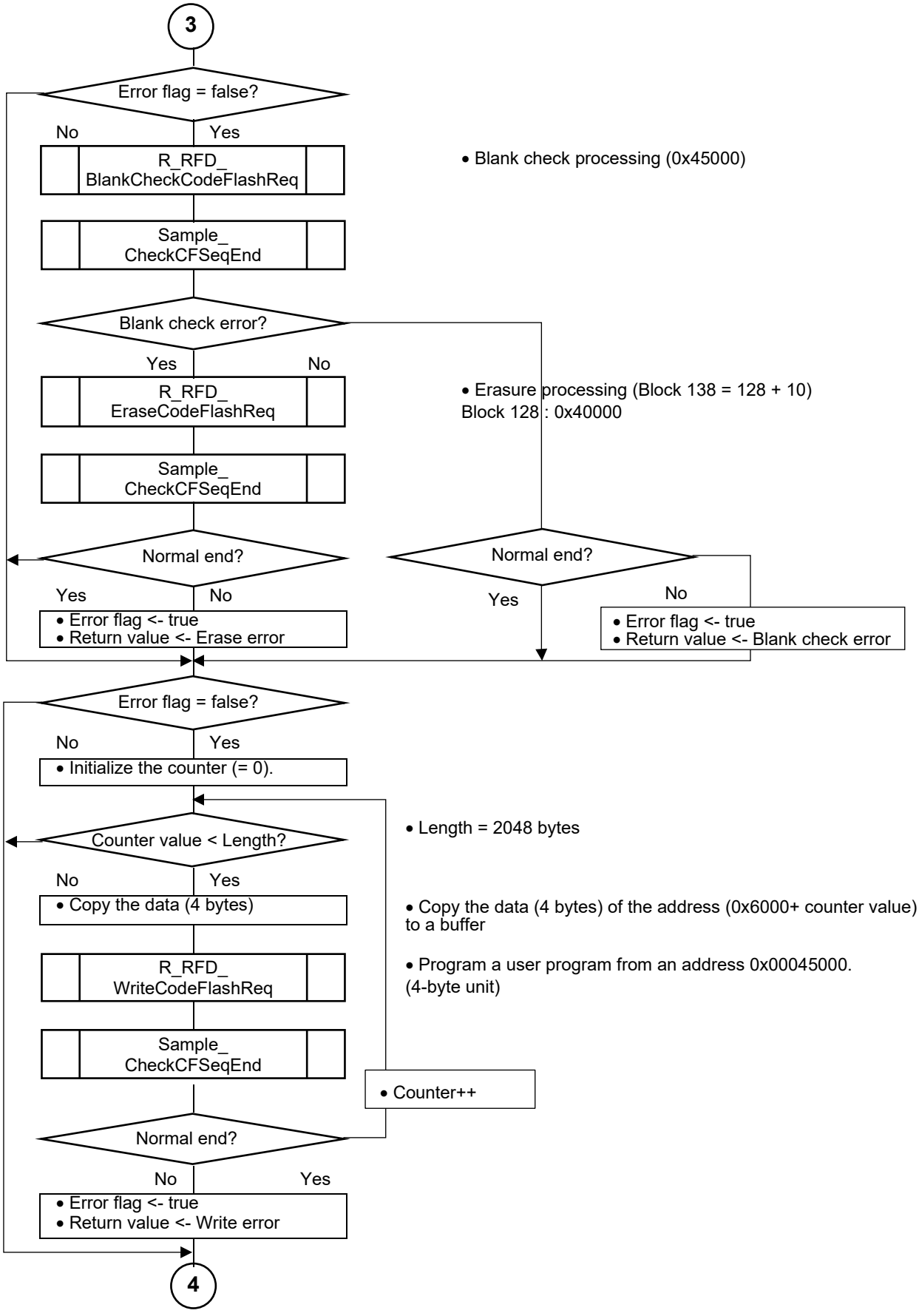


Figure 5-17 Flowchart of Sample Processing for Controlling Bank Programming (4/5)

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

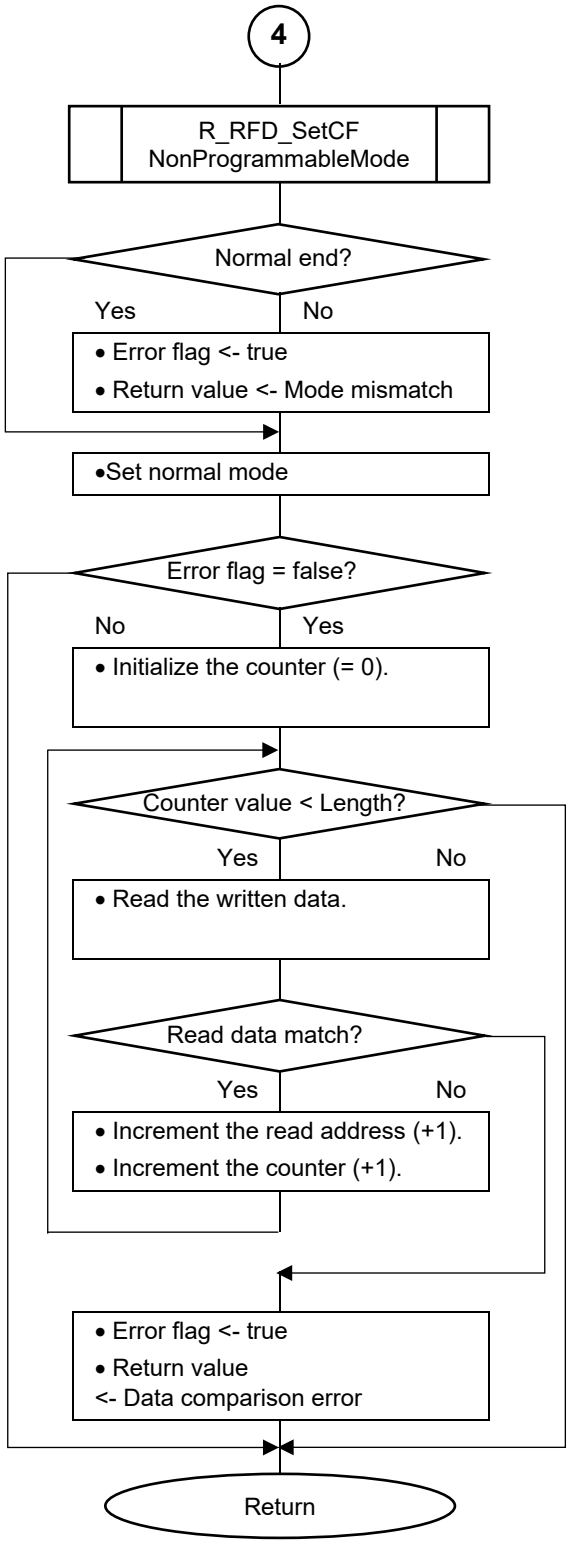


Figure 5-18 Flowchart of Sample Processing for Controlling Bank Programming (5/5)

5.3.4.3 Sample_BankSwappingControl Function

- The sequencer is placed in the code flash memory programming mode and the startup bank swapping setup (boot area switching flag) is executed.

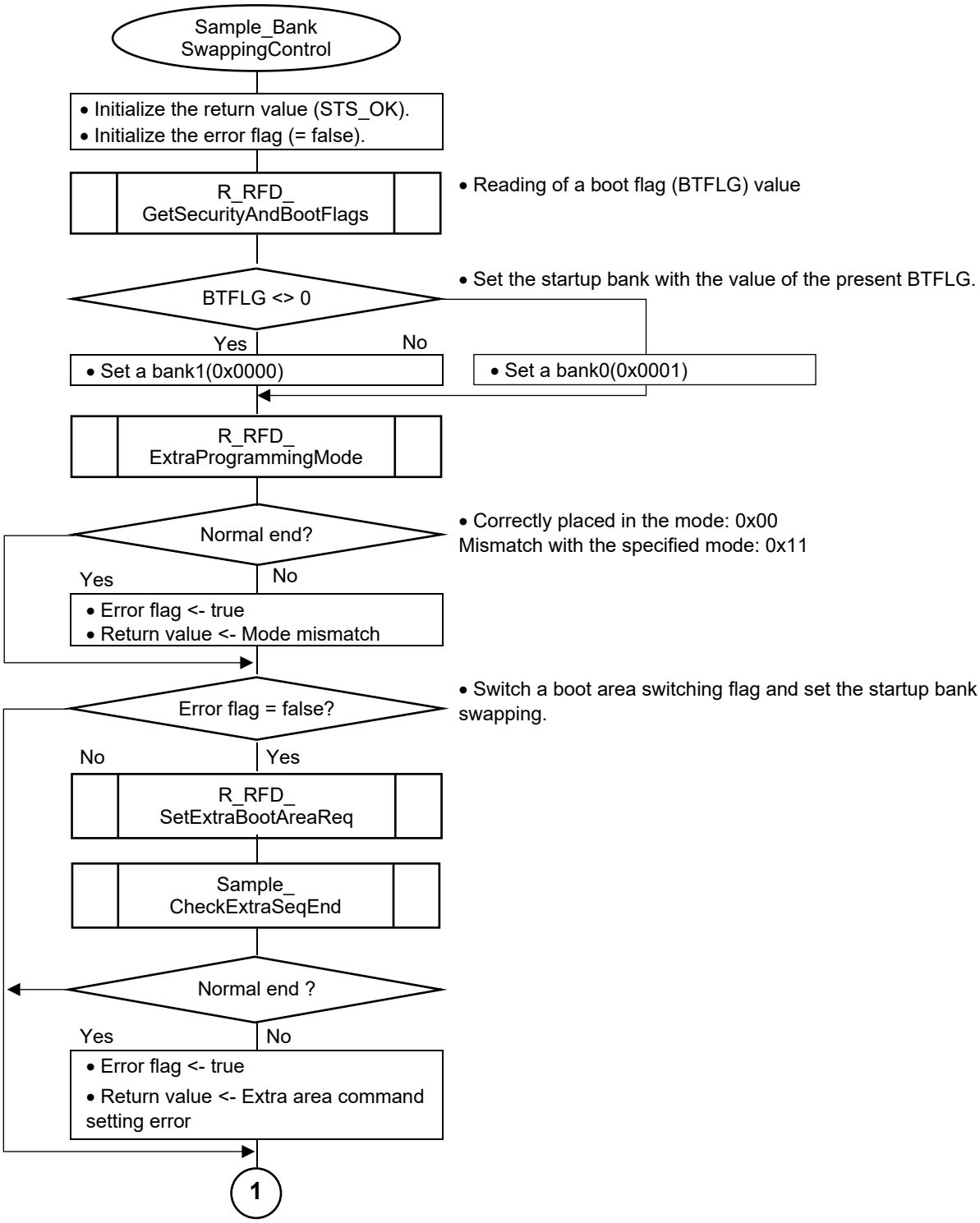


Figure 5-19 Flowchart of Sample Processing for Controlling Bank Swapping (1/2)

[Bank swapping executes after reset: Macro definition (“SMP_BP_SWAP_IMMEDIATELY”) is not set.]

- The sequencer is placed in the non-programmable mode and CPU reset will occur.

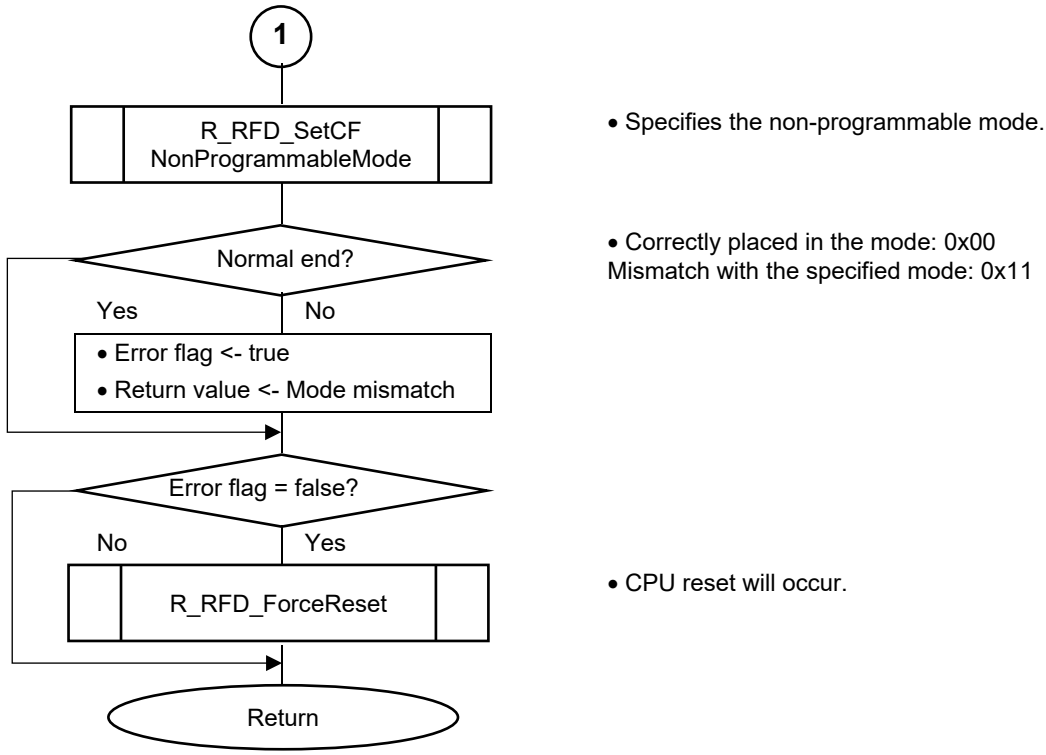
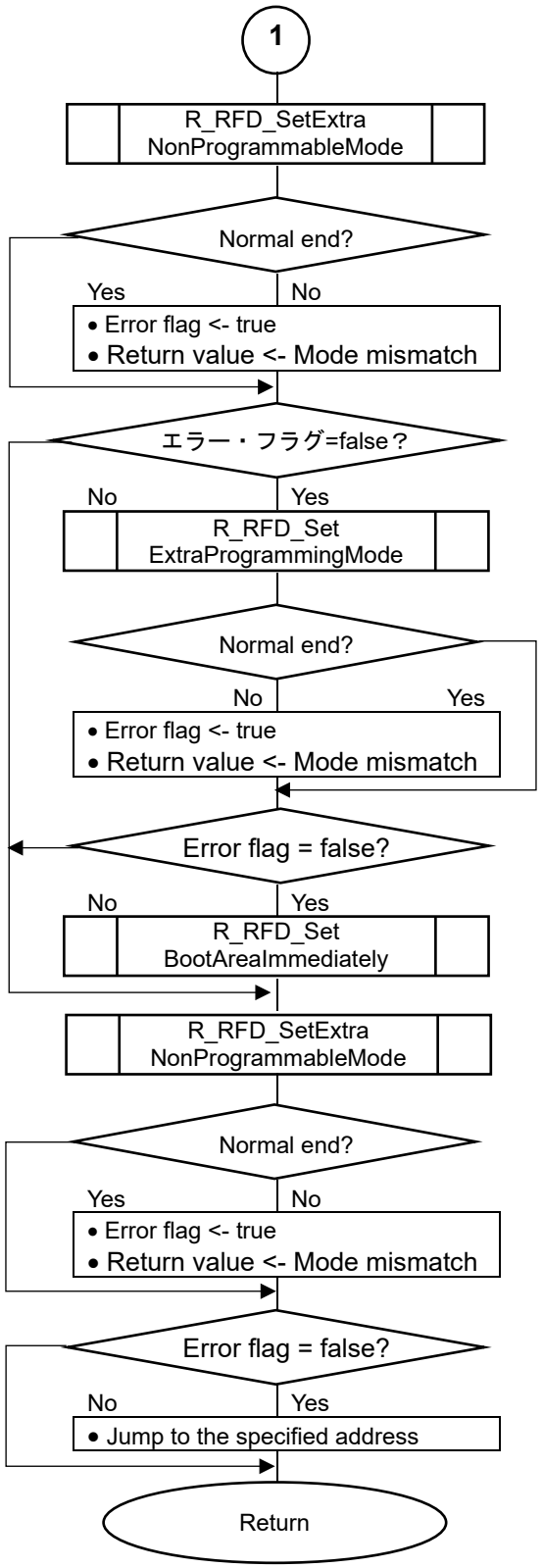


Figure 5-20 Flowchart of Sample Processing for Controlling Bank Swapping (2/2 a)

Note: Be sure not to execute a R_RFD_ForceReset function under debugger execution. The program may run out of control.

[Active bank swapping execution: Macro definition (“SMP_BP_SWAP_IMMEDIATELY”) is set.]

- The sequencer shifts to non-programming mode. Moreover, after it shifts to code flash memory programming mode, an active bank swap is executed.



• Specifies the non-programmable mode.

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

• Specifies the code flash memory programming mode.

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

• Execute active bank swapping.

• Specifies the non-programmable mode.

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

Figure 5-21 Flowchart of Sample Processing for Controlling Bank Swapping (2/2 b)

Note: Be sure not to execute a R_RFD_SetBootAreaImmediately function under debugger execution. The program may run out of control.

5.3.5 Sample Program Used in Common for Controlling the Flash Memory

5.3.5.1 Sample_CheckCFSeqEnd Function

- The end of the operation of the activated code/data flash area sequencer is confirmed and the execution result is returned.

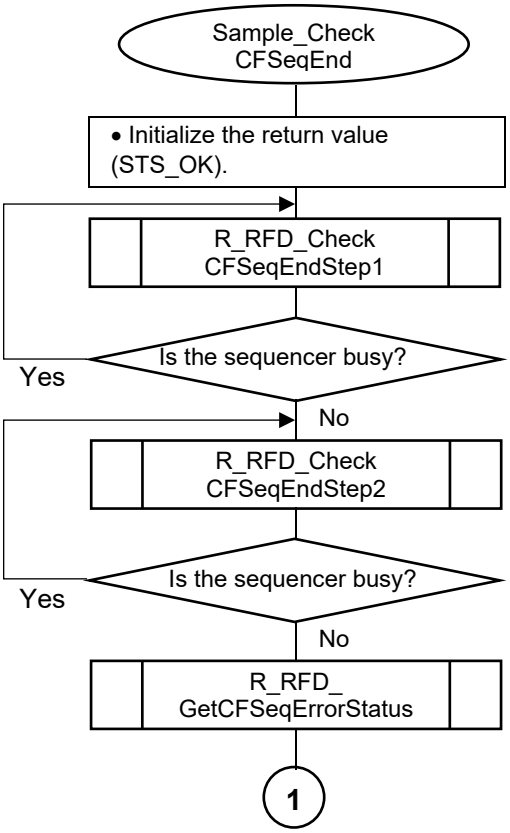


Figure 5-22 Flowchart of Sample_CheckCFSeqEnd Function (1/2)

- Returns the execution result.

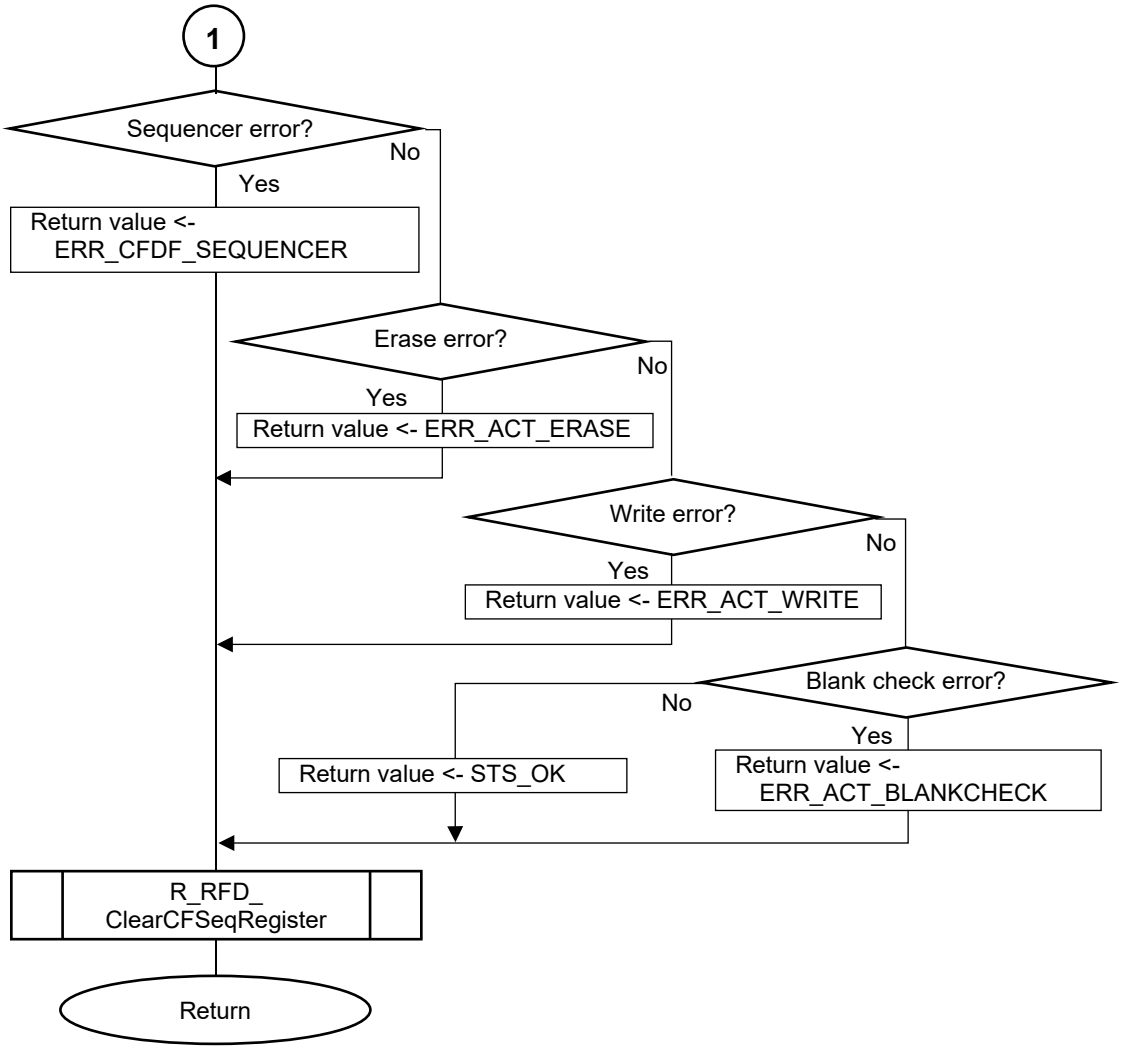


Figure 5-23 Flowchart of Sample_CheckCFSeqEnd Function (2/2)

5.3.5.2 Sample_CheckDFSeqEnd Function

- The end of the operation of the activated code/data flash area sequencer is confirmed and the execution result is returned.

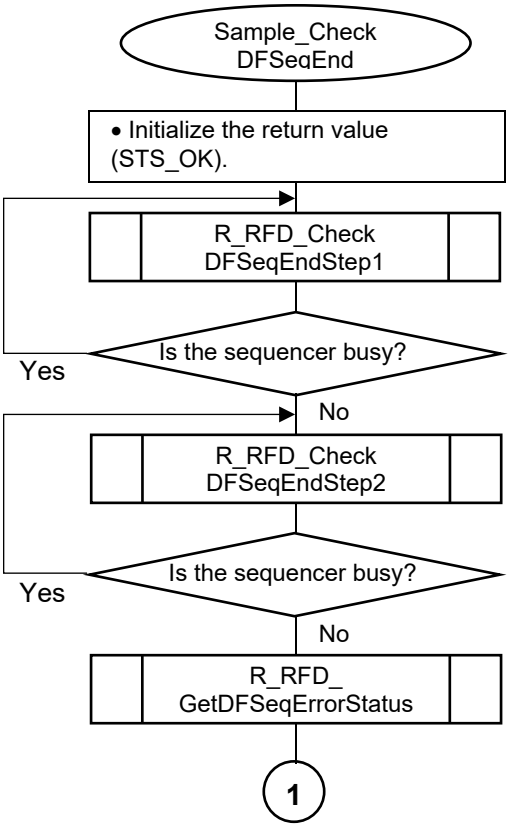


Figure 5-24 Flowchart of Sample_CheckDFSeqEnd Function (1/2)

- Returns the execution result.

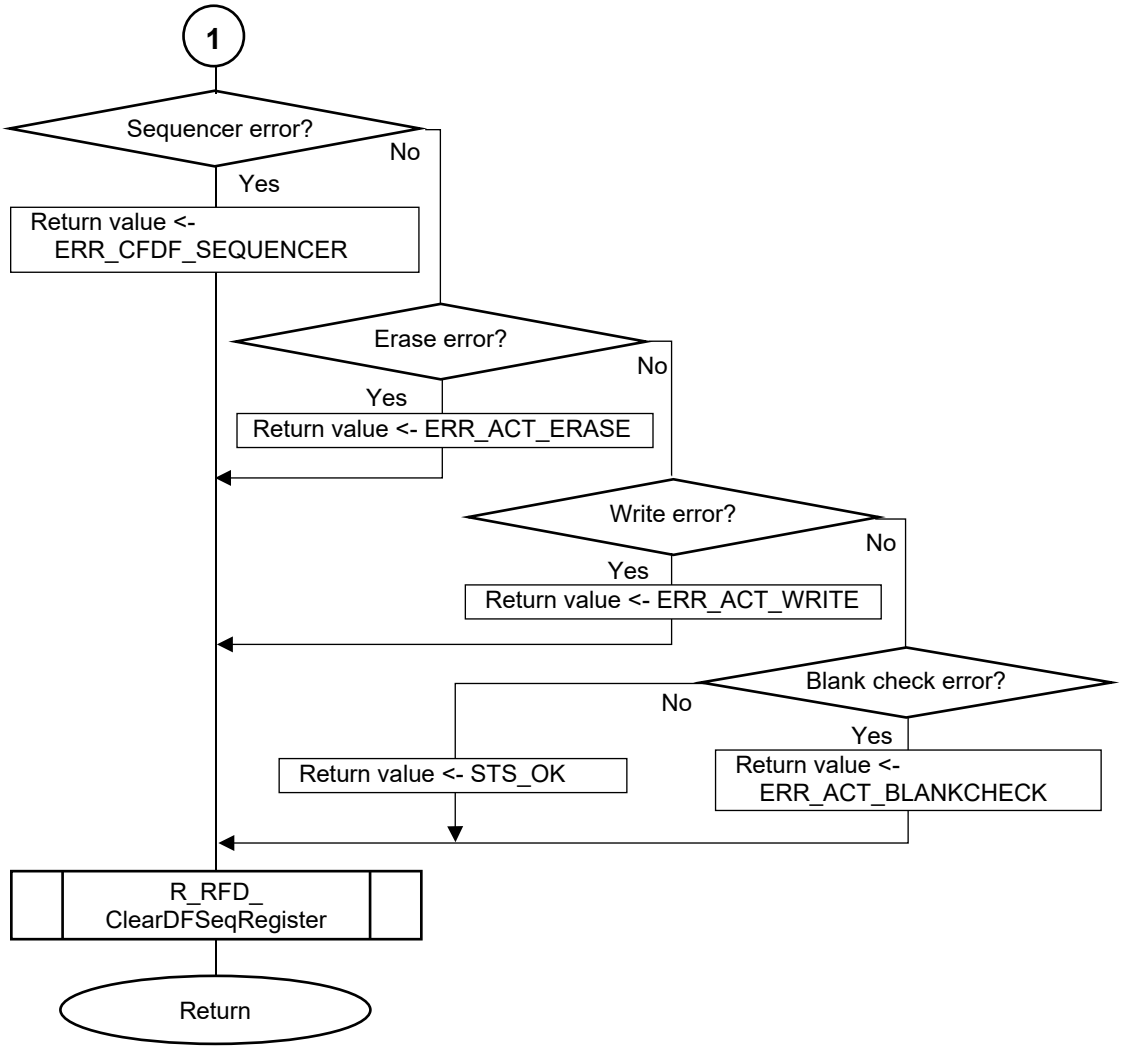


Figure 5-25 Flowchart of Sample_CheckDFSeqEnd Function (2/2)

5.3.5.3 Sample_CheckExtraSeqEnd Function

- The end of the operation of the activated extra area sequencer is confirmed and the execution result is returned.

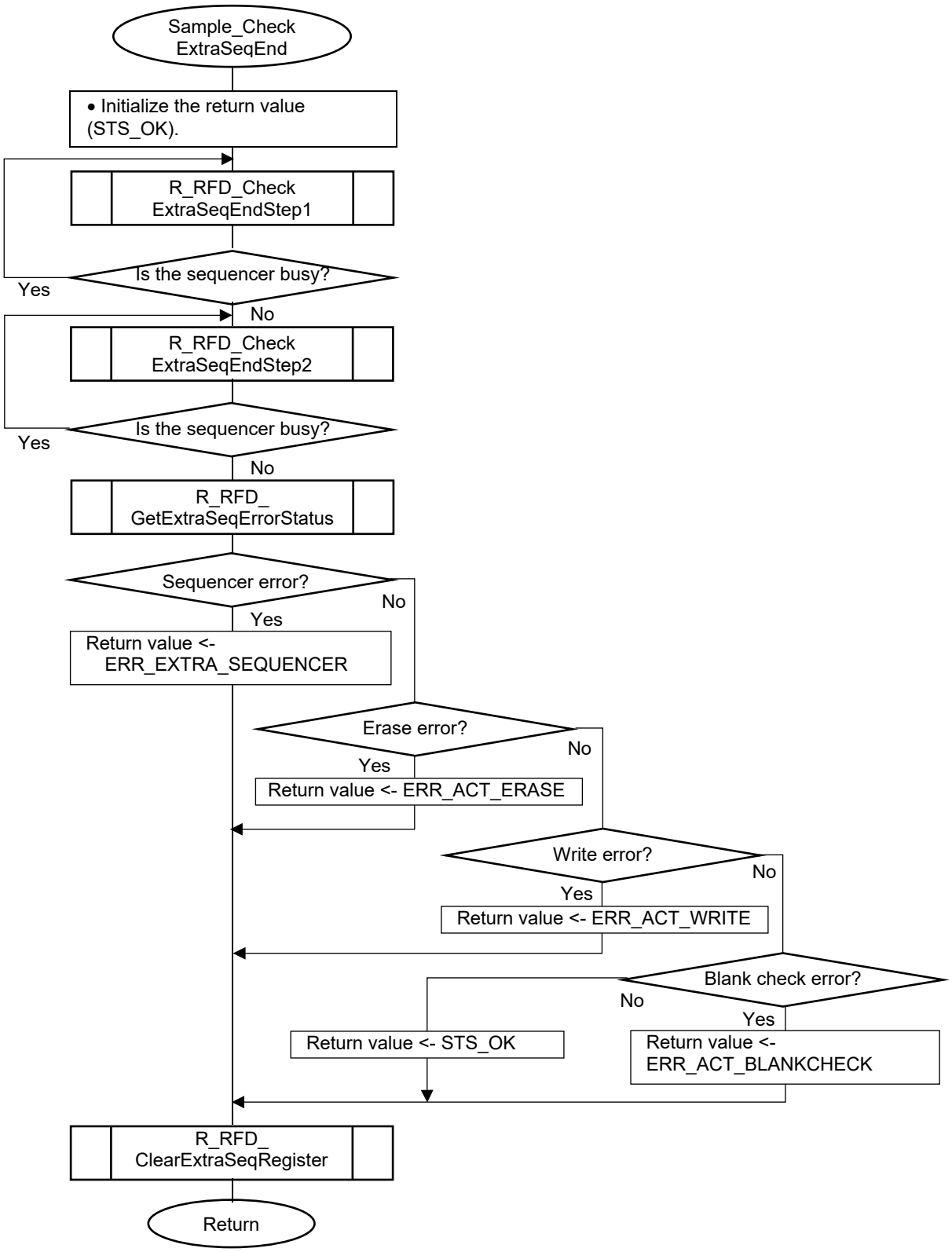


Figure 5-26 Flowchart of Sample_CheckExtraSeqEnd Function

5.4 Specifications of Sample Program Functions

This section describes the specifications of the functions in the sample programs for RFD RL78 Type 11.

The sample programs for RFD RL78 Type 11 are examples of basic processing for reprogramming the code flash area, data flash area, and extra 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	int (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_CHECK_WRITE_DATA: 0x13 [Written data comparison 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_CMD_BLANKCHECK: 0x32 [Blank check command 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_RFD_FAR_FUNC e_sample_ret_t Sample_CodeFlashControl (uint32_t i_u32_start_addrr, uint16_t i_u16_write_data_length, uint8_t __near * inp_u08_write_data);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint32_t i_u32_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_CHECK_WRITE_DATA: 0x13 [Written data comparison 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_CMD_BLANKCHECK: 0x32 [Blank check command error]
Description	Executes the processing for reprogramming the code flash memory. - The blank check, erase, and write commands are executed in the code flash memory programming mode. - 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	int (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_CHECK_WRITE_DATA: 0x13 [Written data comparison 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_CMD_BLANKCHECK: 0x32 [Blank check command 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_RFD_FAR_FUNC e_sample_ret_t Sample_DataFlashControl (uint32_t i_u32_start_addrr, uint16_t i_u16_write_data_length, uint8_t __near * inp_u08_write_data);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint32_t i_u32_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_CHECK_WRITE_DATA: 0x13 [Written data comparison 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_CMD_BLANKCHECK: 0x32 [Blank check command error]
Description	Executes the processing for reprogramming the data flash memory. - The blank check, erase, and write commands are executed in the data flash memory programming mode. - 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. Enable access to the data flash memory at the beginning of this function, and disable it after the reprogramming of the data flash memory is completed.	
Remarks	-	

5.4.3 Sample Program Functions for Controlling the Reprogramming of the Extra Area

5.4.3.1 main

Information:

Syntax	int main(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	int (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_CHECK_WRITE_DATA: 0x13 [Written data comparison error] SAMPLE_ENUM_RET_ERR_CMD_SET_EXTRA_AREA: 0x33 [Extra area command setting error]
Description	Executes the main processing of the sample program for controlling the reprogramming of the extra area (FSW function settings).	
Preconditions	Execute this function in the non-programmable mode while the high-speed on-chip oscillator is active.	
Remarks	-	

5.4.3.2 Sample_ExtraFSWControl

Information:

Syntax	R_RFD_FAR_FUNC e_sample_ret_t Sample_ExtraFSWControl (uint16_t i_u16_start_block_number, uint16_t i_u16_end_block_number, e_rfd_fsw_mode_t i_e_fsw_mode);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint16_t i_u16_start_block_numberr	Start block number Example: For RL78/L23, 0 to 255 (512 Kbytes max.)
	uint16_t i_u16_end_block_number	End block number +1 Example: For RL78/L23, 1 to 256 (512 Kbytes max.)
	e_rfd_fsw_mode_t i_e_fsw_mode	Flash shield window area
		R_RFD_ENUM_FSW_MODE_INSIDE: 0x00 [Inside shield area] R_RFD_ENUM_FSW_MODE_OUTSIDE: 0x01 [Outside shield area]
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_CHECK_WRITE_DATA: 0x13 [Written data comparison error] SAMPLE_ENUM_RET_ERR_CMD_SET_EXTRA_AREA: 0x33 [Extra area command setting error]
Description	Executes the processing for reprogramming the extra area (FSW function settings). - The write command for the extra area (FSW-related data programming command) is executed in the code flash memory programming mode. - The on-chip registers corresponding to 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	-	

5.4.4 Sample Program Functions for Controlling the Bank Programming / Bank Swapping

5.4.4.1 main

Information:

Syntax	int main(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	int (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_CHECK_WRITE_DATA: 0x13 [Written data comparison 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_CMD_BLANKCHECK: 0x32 [Blank check command error] SAMPLE_ENUM_RET_ERR_CMD_SET_EXTRA_AREA: 0x33 [Extra area command setting error]
Description	Executes the main processing of the sample program for controlling the bank programming and bank swapping.	
Preconditions	Execute this function in the non-programmable mode while the high-speed on-chip oscillator is active.	
Remarks	-	

5.4.4.2 Sample_BankProgrammingControl

Information:

Syntax	R_RFD_FAR_FUNC e_sample_ret_t Sample_BankProgrammingControl(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_MODE_MISMATCHED: 0x12 [Mode mismatch error] SAMPLE_ENUM_RET_ERR_CHECK_WRITE_DATA: 0x13 [Written data comparison 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_CMD_BLANKCHECK: 0x32 [Blank check command error]
Description	Executes the processing for reprogramming the rewrite bank area. - The blank check, erase, and write commands are executed in the code flash memory programming mode. - 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	-	

5.4.4.3 Sample_BankSwapControl

Information:

Syntax	R_RFD_FAR_FUNC e_sample_ret_t Sample_BankSwapControl(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_MODE_MISMATCHED: 0x12 [Mode mismatch error] SAMPLE_ENUM_RET_ERR_CMD_SET_EXTRA_AREA: 0x33 [Extra area command setting error]
Description	Executes the processing for reprogramming the extra area (boot area switching flag settings) and bank swapping. - The write command for the extra area (Programming of the security flags and the boot area switching flag command) is executed in the code flash memory programming mode. - A bank swap is executed after reset or immediately. The case of [Bank swapping executes after reset]: The sequencer is placed in the non-programmable mode and CPU reset will occur. After the bank swapping is executed, the exchanged program of the startup bank is executed. The case of [active bank swapping execution (Immediate execution bank swapping)]: After the sequencer shifts to non-programming mode, it shifts to code flash memory programming mode. After active bank swapping is executed, the sequencer shifts to non-programming mode. And user processing jumps to the specified address of exchanged startup bank.	
Preconditions	Execute this function in the non-programmable mode while the high-speed on-chip oscillator is active.	
Remarks	-	

5.4.5 Sample Program Functions Used in Common

5.4.5.1 Sample_CheckCFSeqEnd

Information:

Syntax	R_RFD_FAR_FUNC e_sample_ret_t Sample_CheckCFSeqEnd(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 [Code/data flash area sequencer error] SAMPLE_ENUM_RET_ERR_ACT_ERASE: 0x22 [Erase operation error] SAMPLE_ENUM_RET_ERR_ACT_WRITE: 0x23 [Write operation error] SAMPLE_ENUM_RET_ERR_ACT_BLANKCHECK: 0x24 [Blank check operation error]
Description	Waits for the completion of command execution in the code/data flash area sequencer for the code flash reprogramming.	
Preconditions	Use this function in the code flash memory programming mode while the high-speed on-chip oscillator is active.	
Remarks	-	

5.4.5.2 Sample_CheckDFSeqEnd

Information:

Syntax	R_RFD_FAR_FUNC e_sample_ret_t Sample_CheckDFSeqEnd(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 [Code/data flash area sequencer error] SAMPLE_ENUM_RET_ERR_ACT_ERASE: 0x22 [Erase operation error] SAMPLE_ENUM_RET_ERR_ACT_WRITE: 0x23 [Write operation error] SAMPLE_ENUM_RET_ERR_ACT_BLANKCHECK: 0x24 [Blank check operation error]
Description	Waits for the completion of command execution in the code/data flash area sequencer for the data flash reprogramming.	
Preconditions	Use this function in the data flash memory programming mode while the high-speed on-chip oscillator is active. When reprogramming the data flash memory, use this function while access to the data flash memory is enabled (DFLEN = 1).	
Remarks	-	

5.4.5.3 Sample_CheckExtraSeqEnd

Information:

Syntax	R_RFD_FAR_FUNC e_sample_ret_t Sample_CheckExtraSeqEnd(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_EXTRA_SEQUENCER: 0x21 [Extra area sequencer error] SAMPLE_ENUM_RET_ERR_ACT_ERASE: 0x22 [Erase operation error] SAMPLE_ENUM_RET_ERR_ACT_WRITE: 0x23 [Write operation error] SAMPLE_ENUM_RET_ERR_ACT_BLANKCHECK: 0x24 [Blank check operation error]
Description	Waits for the completion of command execution in the extra area sequencer for the extra area reprogramming.	
Preconditions	Execute this function in the code flash memory programming mode while the high-speed on-chip oscillator is active. Use this function while access to the data flash memory is enabled.	
Remarks	-	

6. Creating a Sample Project for RFD RL78 Type 11

RFD RL78Type 11 includes sample programs for a code flash memory area and a data flash memory area to program. The compilers which can be used by RFD RL78 Type 11 are a CC-RL compile, an IAR compiler and a LLVM compiler. Users can create a sample project using the Integrated Development Environment (IDE) corresponding to each compiler.
The example of the sample program for RL78/L23(R7F100LPL) is explained to this section. When using other than RL78/L23(R7F100LPL), 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/L23. Be sure to use them, after confirming that they support RL78/L23.

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

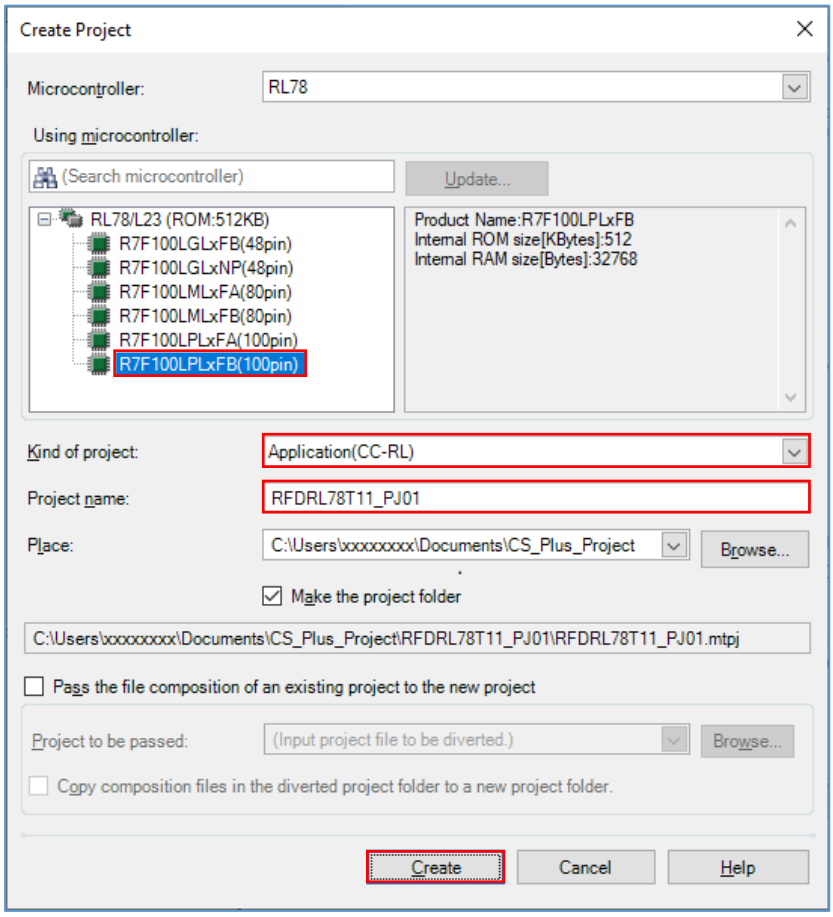
CS+ or e^2 studio can be used for a RENESAS CC-RL compiler as an IDE. RFD RL78 Type 11 is registered and built in the project created by the IDE. An example of creating a sample project in case each IDE is used is shown. Because to understand a CC-RL compiler and each IDE, it is necessary to refer to 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)

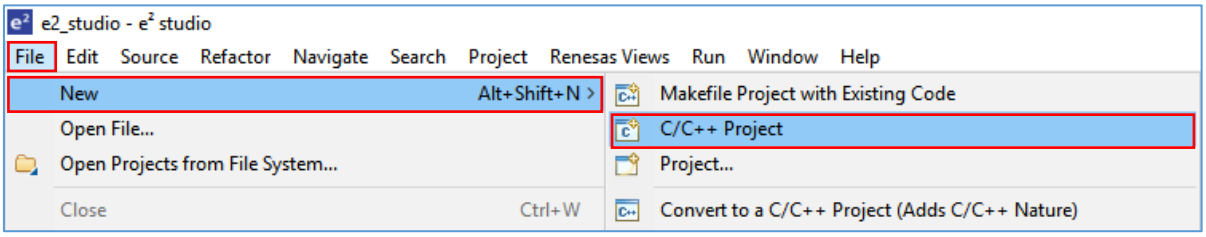
・The CS+ starts and from the [Project] menu, select [Create New Project...], the “Create Project” window will open.

- Select the product of “RL78/L23 (ROM: 512 Kbytes)” - “R7F100LPLxFB(100pin)” as [Using microcontroller].
- Select “Application (CC-RL)” as [Kind of project].
- [Project name] is temporarily set to “RFDRL78T11_PJ01”.
- When you click the [Create] button, the new project is created.

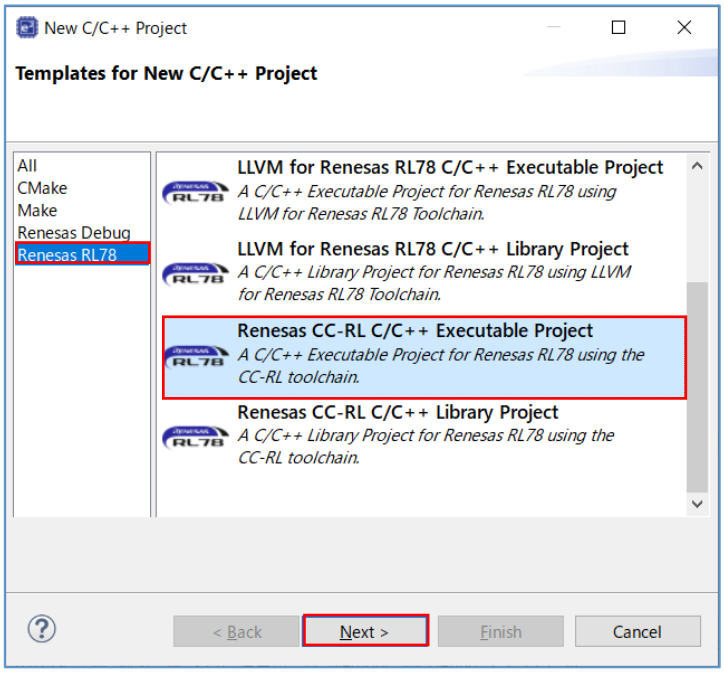


(2) An example of creating a sample project which used e² studio (IDE)

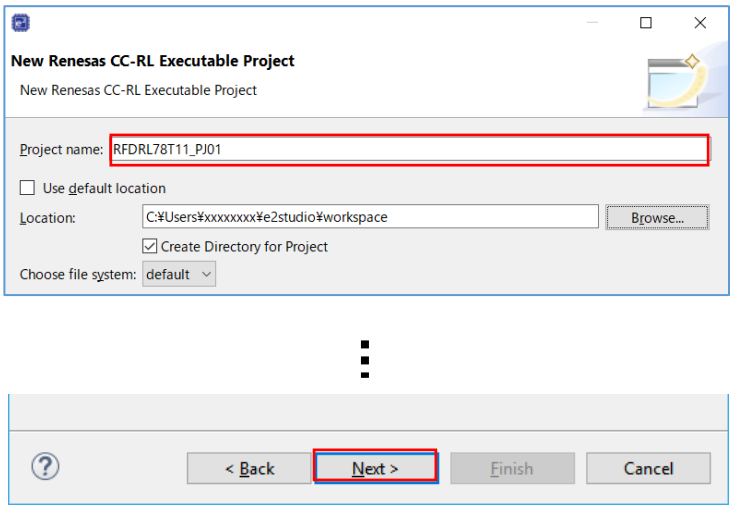
- The e² studio starts and from the [File] menu, select [New] – [C/C++ Project], the “Templates for New C/C++ Project” window will open.



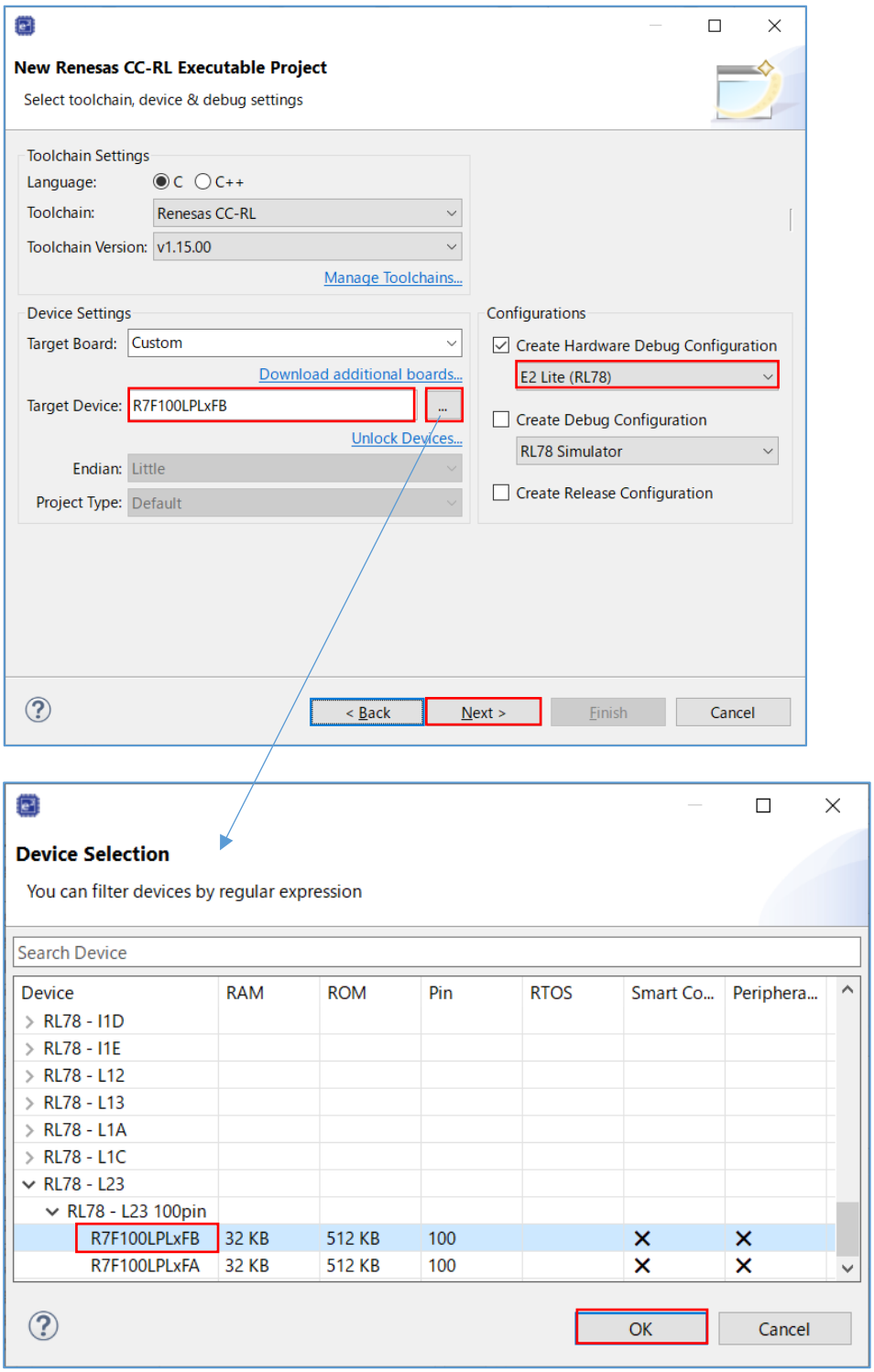
- Select [Renesas CC-RL C/C++ Executable Project] displayed after selection in [Renesas RL78], and press “next” button.



- Input “project name” on “New Renesas CC-RL Executable Project” window, and press “next” button. [Project name] is temporarily set to “RFDRL78T11_PJ01”.

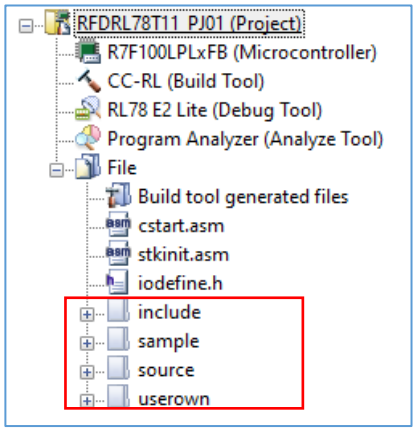


- Select the [Target Device] of [Device Settings], and select “RL78 - L23 100pin” - “R7F100LPLxFB”.
- It is a premise that E2 Lite is selected as a debugging tool and on-chip debugging is executed. Put a check mark to “Create Hardware Debug Configuration” by [Configurations]. And select “E2 Lite (RL78)”.
- When press the [Next] button, the “Select Coding Assistant settings” window will be displayed, so press the [Finish] button.

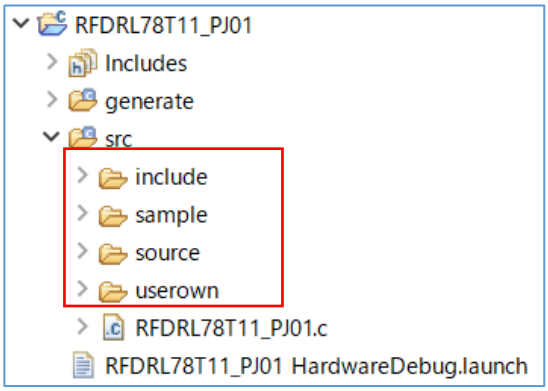


6.1.2 Example of Registration of Target Folders and Target Files

Using RFD RL78 Type 11, when programming each area [(1) code flash memory, (2) data flash memory, (3) extra area], and (4) bank programming / bank swapping control, the example which registers necessary files is shown. Each folder of the RFD RL78 Type 11 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 RFD (CS+)



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

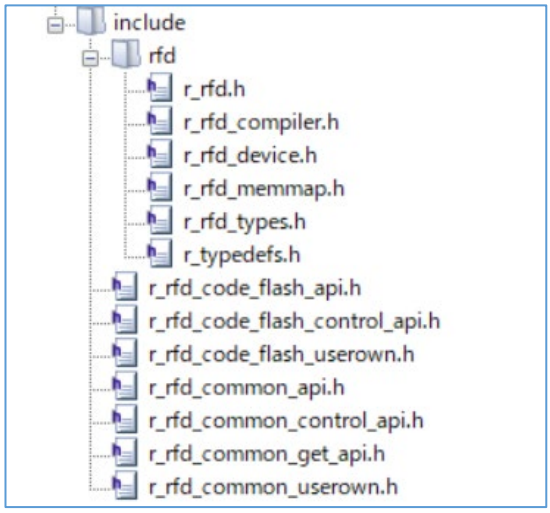
- Registration of the latest I/O header file(iodefine.h) outputted to target products by IDE
  “iodefine.h” uses the I/O header file which CS+ or e^2 studio outputs for target products.

  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
- 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 RFD RL78 Type 11. 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 “hdwinit.asm, and main.c,” 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 a hdwinit.asm in [project name] / generate folder and a [project name].c (“RFDRL78T11_PJ01.c”) in [project name] / src folder.

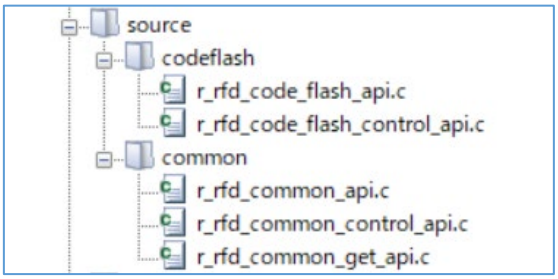
(1) Registration of the folders and files of the target in the case of reprogramming code flash memory

The folders (“include”, “source”, “userown”, “sample”) and source program file which are included in RFD RL78 Type 11 to register are shown below.

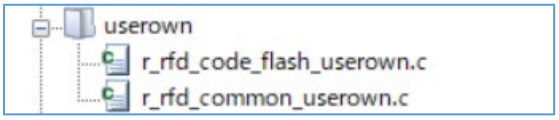
in the “include” folder



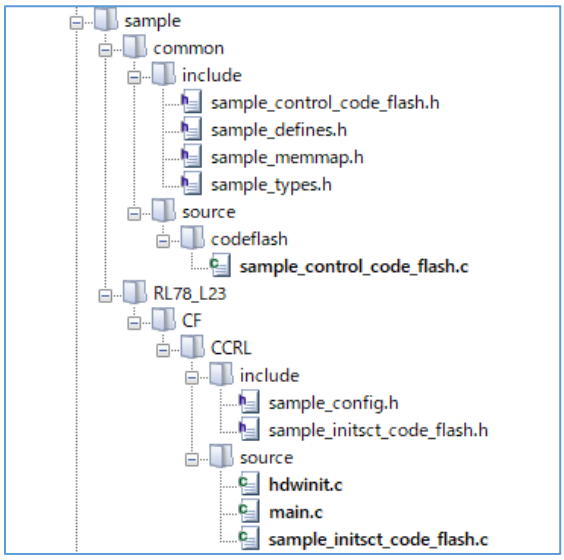
in the “source” folder



in the “userown” folder



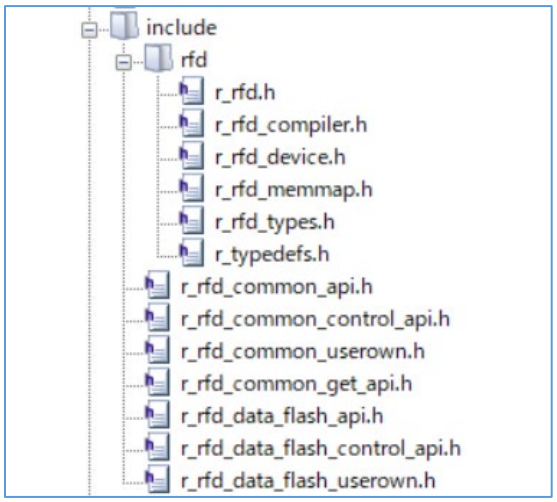
in the “sample” folder



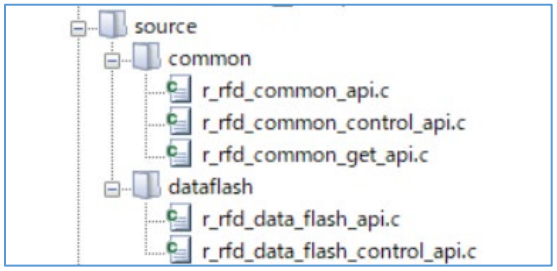
(2) Registration of the folders and files of the target in the case of reprogramming data flash memory

The folders (“include”, “source”, “userown”, “sample”) and source program file which are included in RFD RL78 Type 11 to register are shown below.

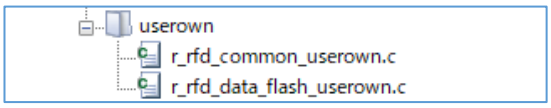
in the “include” folder



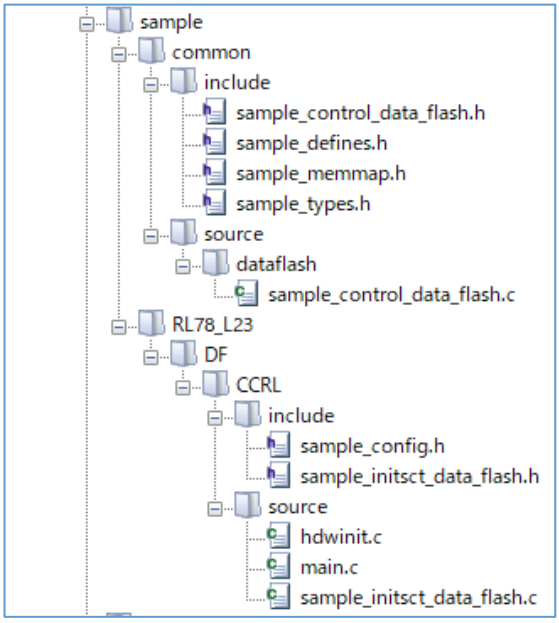
in the “source” folder



in the “userown” folder



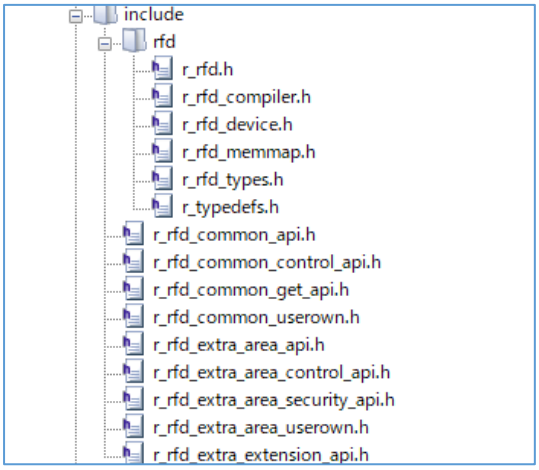
in the “sample” folder



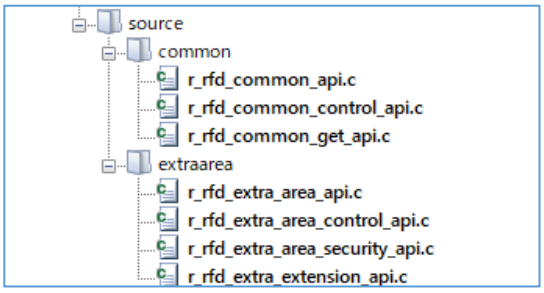
(3) Registration of the folders and files of the target in the case of reprogramming extra area (FSW setting)

The folders (“include”, “source”, “userown”, “sample”) and source program file which are included in RFD RL78 Type 11 to register are shown below.

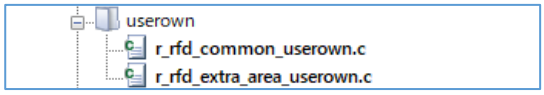
in the “include” folder



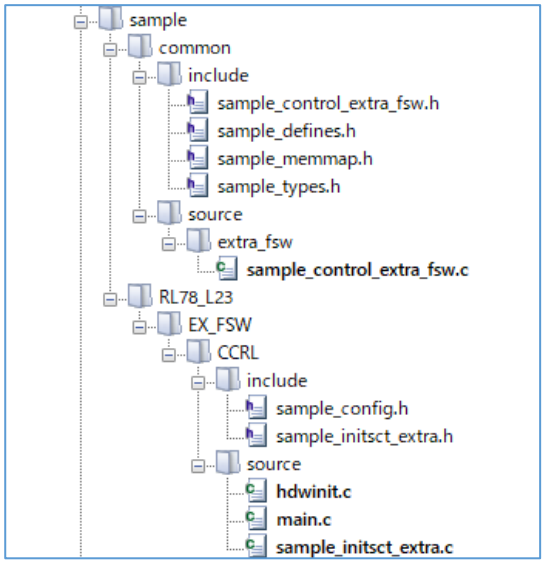
in the “source” folder



in the “userown” folder



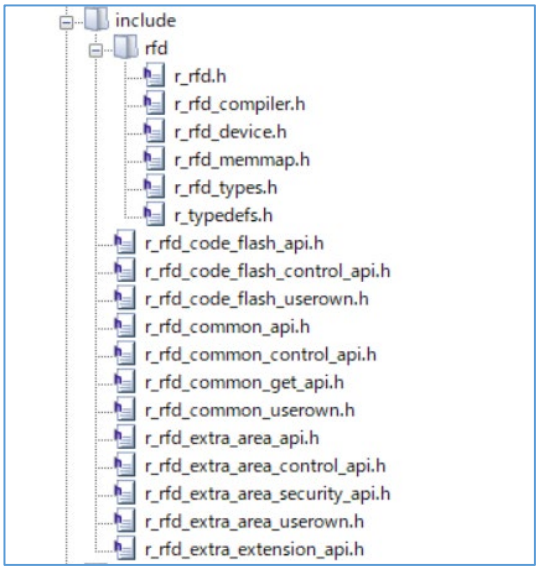
in the “sample” folder



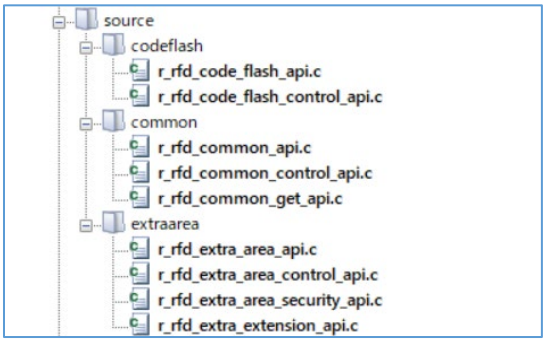
(4) Registration of the folders and files of the target in the case of bank programming / bank swapping control cexecution

The folders (“include”, “source”, “userown”, “sample”) and source program file which are included in RFD RL78 Type 11 to register are shown below.

in the “include” folder

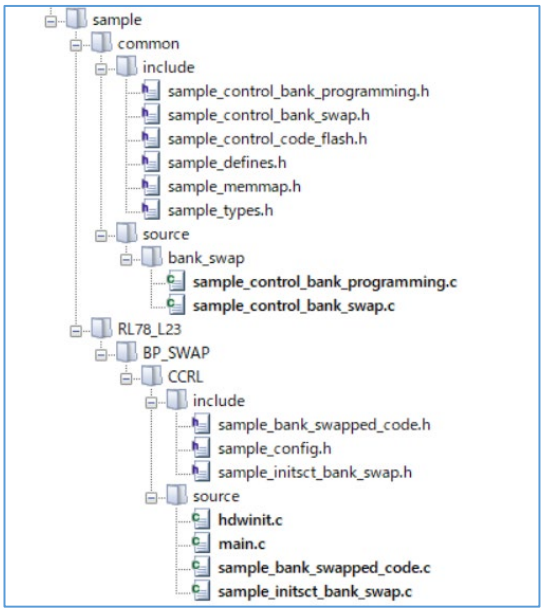


in the “source” folder



in the “userown” folder

in the “sample” folder



6.1.3 Build Tool Settings

Set IDE setting necessary in order to build RFD RL78 Type 11 using a CC-RL compiler.
CS+: Click the right mouse button for the “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 (“RFDRL78T11_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\rfd
include
sample\RL78_L23\CF\CCRL\include
sample\common\include
.
```

(2) Data flash memory reprogramming

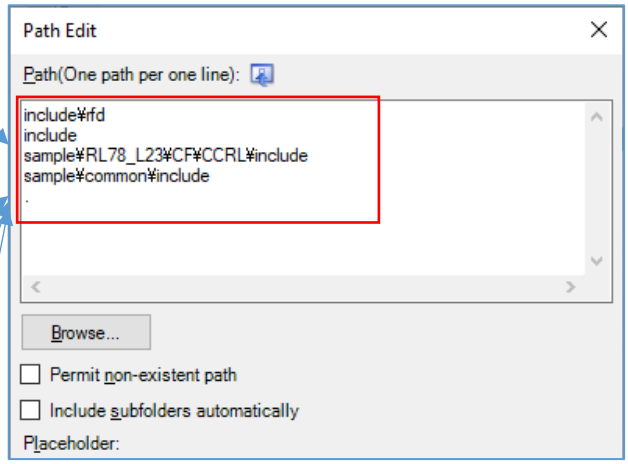
```
include\rfd
include
sample\RL78_L23\DF\CCRL\include
sample\common\include
.
```

(3) Extra area (FSW) reprogramming

```
include\rfd
include
sample\RL78_L23\EX_FSW\CCRL\include
sample\common\include
.
```

(4) Bank programming / bank swapping control

```
include\rfd
include
sample\RL78_L23\BP_SWAP\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}\generate
${ProjDirPath}\src\include\rfd
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_L23\CF\CCRL\include
${ProjDirPath}\src\sample\common\include
```

(2) Data flash memory reprogramming

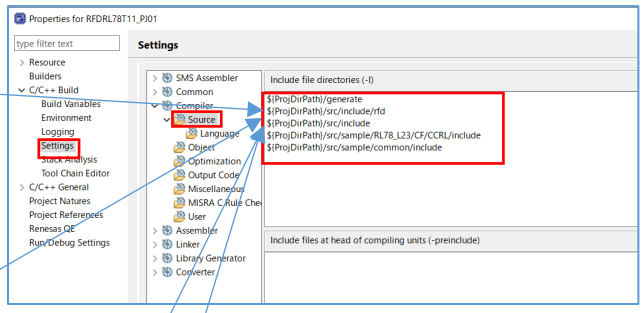
```
${ProjDirPath}\generate
${ProjDirPath}\src\include\rfd
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_L23\DF\CCRL\include
${ProjDirPath}\src\sample\common\include
```

(3) Extra area (FSW) reprogramming

```
${ProjDirPath}\generate
${ProjDirPath}\src\include\rfd
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_L23\EX_FSW\CCRL\include
${ProjDirPath}\src\sample\common\include
```

(4) Bank programming / bank swapping control

```
${ProjDirPath}\generate
${ProjDirPath}\src\include\rfd
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_L23\BP_SWAP\CCRL\include
${ProjDirPath}\src\sample\common\include
```



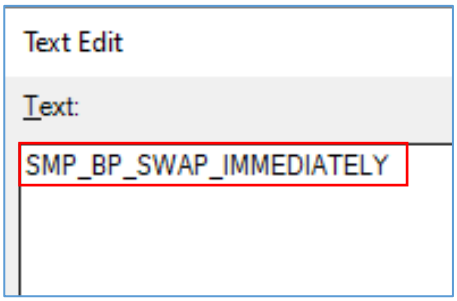
6.1.3.2 The Setting of User Definition Macro (CC-RL Compiler)

In the case which selects “active bank swapping execution” (immediate execution bank swapping) with the sample program function of bank swapping control, user definition macro (“SMP_BP_SWAP_IMMEDIATELY”) is necessary. In the case which selects “bank swapping execution after reset”, this definition is unnecessary.

- On CS+, the macro for selecting “active bank swapping execution (Immediate execution bank swapping)” is defined in “Common Options” tab.
  - Define the following macro in the “Text Edit” window displayed by selection of [Frequently Used Options(for Compile)] - [Macro definition]. Definition macro differs by each device to be used.

    Macro for selecting “active bank swapping execution” (Immediate execution bank swapping):

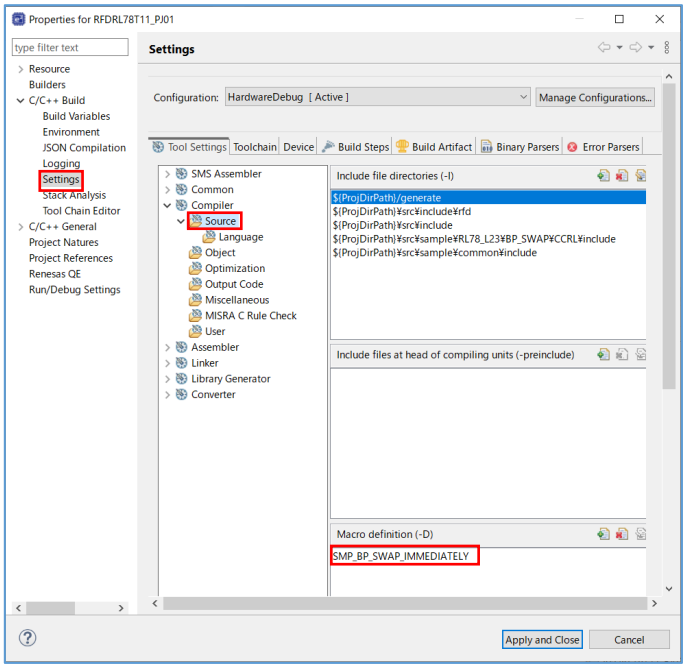
    SMP_BP_SWAP_IMMEDIATELY



- On e² studio, the macro for selecting “active bank swapping execution (Immediate execution bank swapping)” is defined in “Properties” window.
  - Define the following macro in the “Macro Definition (-D)” displayed by selection of “C/C++ Build” [Settings] - Compiler” [Source]. Definition macro differs by each device to be used.

    Macro for selecting “active bank swapping execution” (Immediate execution bank swapping):

    SMP_BP_SWAP_IMMEDIATELY



6.1.3.3 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.]

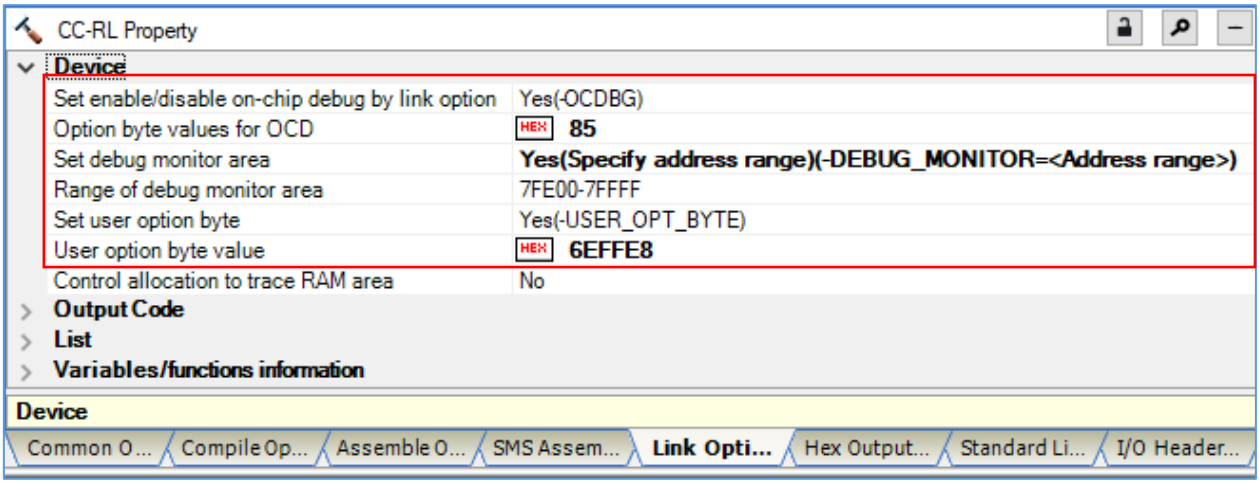
Note: Be sure to confirm the contents of “On-Chip Debug Option Byte” in “Option Byte” chapter on the user's manual of a target device. And describe the set value used with user application.

Select “Yes(Specify address range)(-OCDBG_MONITOR=<Address range>)” in [Set debug monitor area]. Set “**7FE00-7FFFF**” to [Range of debug monitor area]. [The example for RL78/L23]

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 “**6EFFE8**” to [User option byte value]. (WDT stop, LVD [reset mode], 32MHz [The example for RL78/L23])

Note: Be sure to confirm the contents of “User option byte” in “Option Byte” chapter 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 “**7FE00-7FFFF**” to [Memory area (-debug_monitor=<start address>-<end address>)]. [The example for RL78/L23]

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 “**6EFFE8**” to [User option byte value(-user_opt_byte=<value>)]. (WDT stop, LVD [reset mode], 32MHz [The example for RL78/L23])

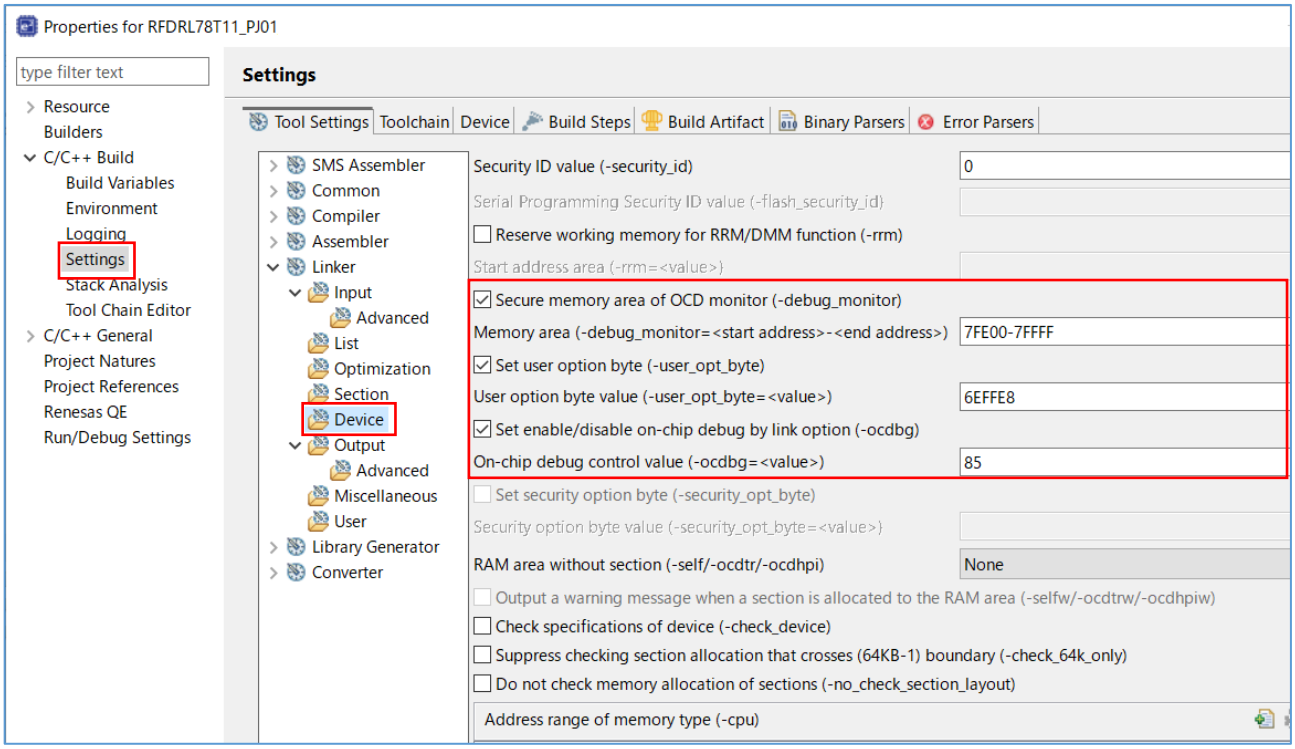
Note: Be sure to confirm the contents of “User option byte” in “Option Byte” chapter on the user's manual of a target device. And describe the set value used with user application.

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 “On-Chip Debug Option Byte” in “Option Byte” chapter on the user's manual of a target device. And describe the set value used with user application.

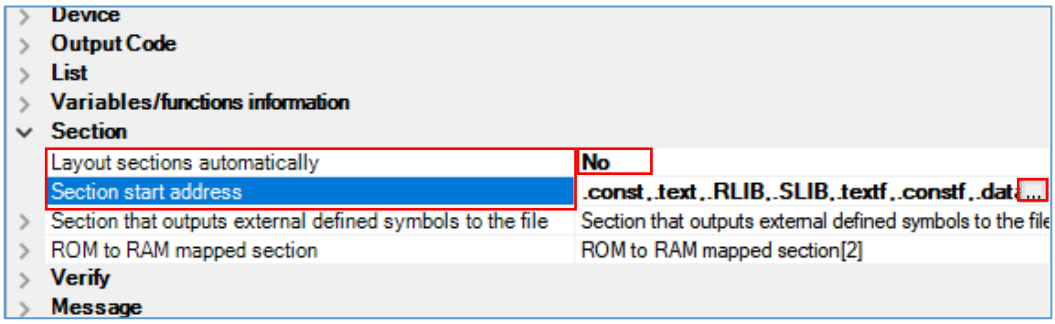


6.1.3.4 Section Item Settings

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

- Setting the [Section] items

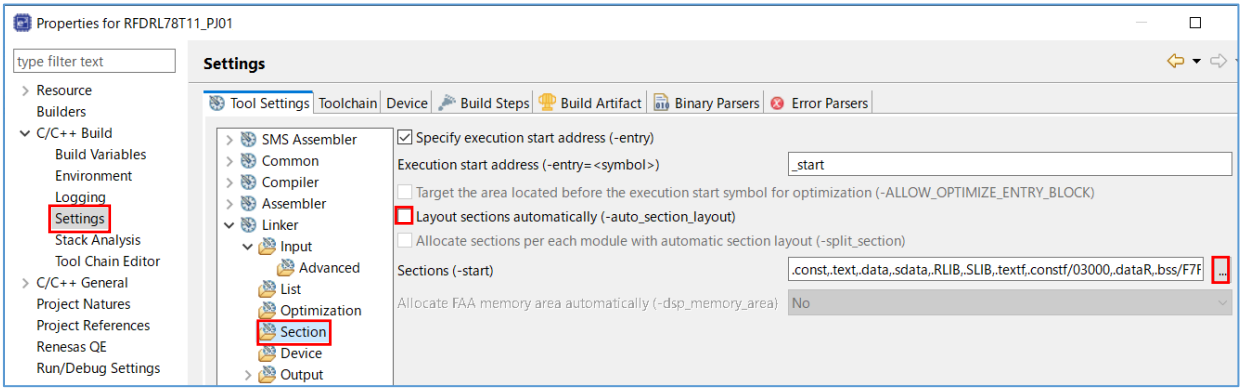
Set “No” to [Layout sections automatically]. And sections come to be displayed on [Section start address]. Press the “ ... “ button of the right-hand side which sections are displaying, and a “Section settings” screen is displayed.



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

- Select “C/C++ Build” [Setting] - “Linker” [Section]. And set section items on the displayed screen.

Remove a check mark to [Layout sections automatically(-auto_section_layout)]. Press the “ ... “ button of the right-hand side which sections are displaying, and a “Section viewer” screen is displayed.



・Section setting operation for CS+ and e^2 studio

Set “0x03000” to a top address.

Add the sections defined by “#pragma section” in RFD RL78 Type 11 to the program area (code flash memory) and the RAM area. Refer to “2.3.1 Sections in case of using RFD RL78 Type 11“ for the details of each section.

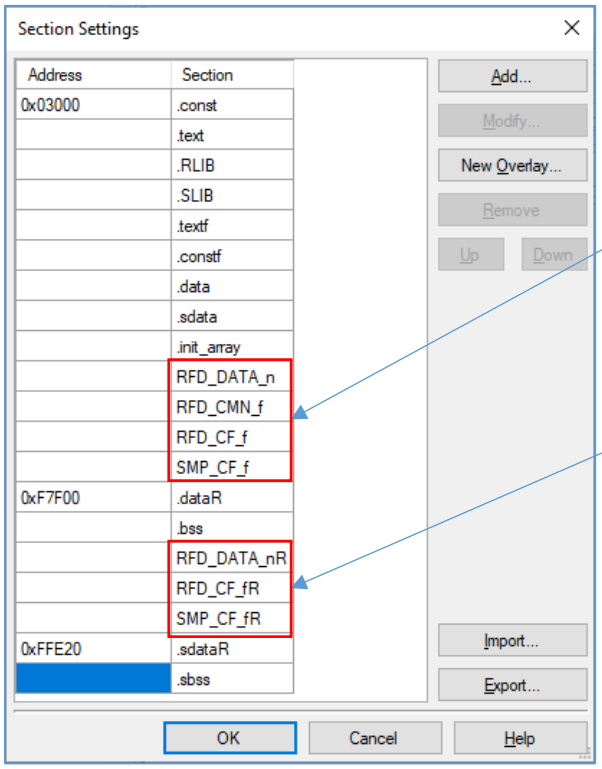
Note: In this description, it is a premise to select a medium model as Memory Model of Compile Options. (It is the same as the “auto select” in R7F100LPL) The section names of each program on “#pragma section” of CC-RL are set to “section name +_f” with a “__far” attribute. The section names copied to RAM from ROM are “section name +_fR” with a “__far” attribute. Refer to the user's manual of CC-RL for the section name of each program when a “small model” is selected.

(1) The addition of the sections for code flash memory reprogramming

・The addition of the sections for code flash memory reprogramming on CS+

Add sections necessary for code flash memory reprogramming on a “Section Settings” screen.

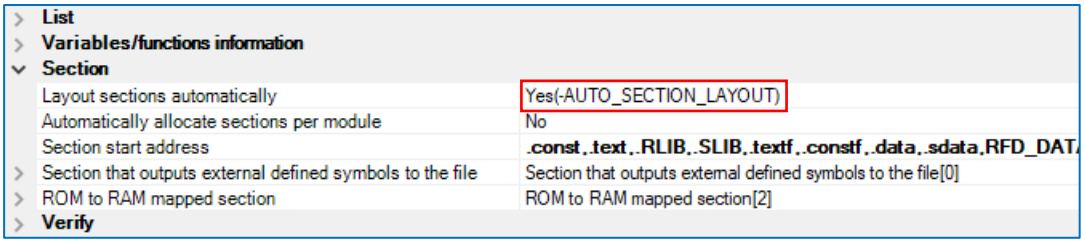
Add to the program area: RFD_DATA_n, RFD_CMN_f, RFD_CF_f, SMP_CF_f
Add to the RAM area: RFD_DATA_nR, RFD_CF_fR, SMP_CF_fR



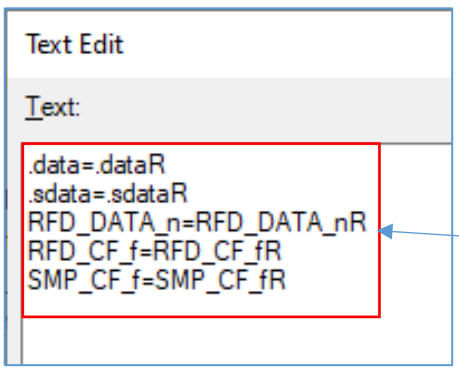
Additional sections
RFD_DATA_n
RFD_CMN_f
RFD_CF_f
SMP_CF_f

RFD_DATA_nR
RFD_CF_fR
SMP_CF_fR

Be sure to return [Layout sections automatically] to “Yes”, after pressing the “OK” button.



Press the right-hand side “ ... “ button by [ROM to RAM mapped section], display the “Text Edit” screen, and add the section for copying to RAM from ROM.

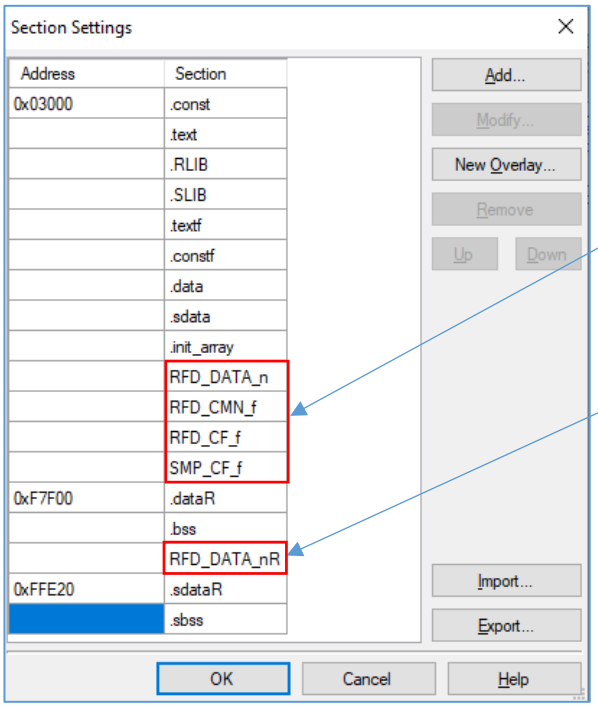


ROM to RAM mapped section
.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR
RFD_CF_f=RFD_CF_fR
SMP_CF_f=SMP_CF_fR

・The addition of the sections for code flash memory reprogramming on CS+ (Bank programming)

Add sections necessary for code flash memory reprogramming on a “Section Settings” screen.

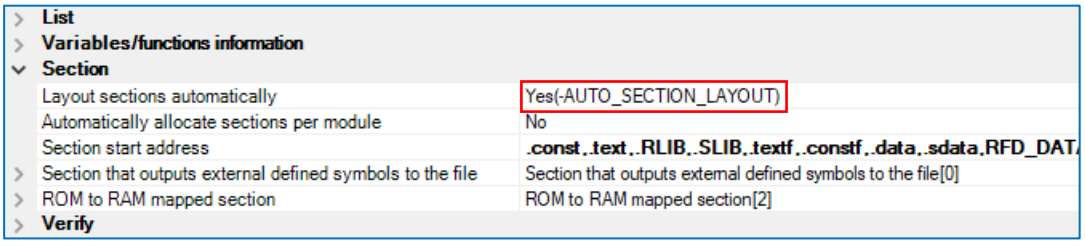
Add to the program area: RFD_DATA_n, RFD_CMN_f, RFD_CF_f, SMP_CF_f
Add to the RAM area: RFD_DATA_nR



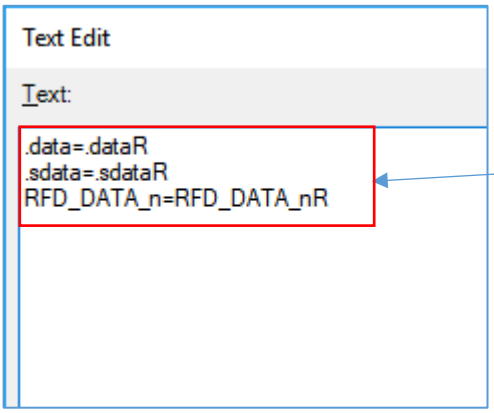
Additional sections
RFD_DATA_n
RFD_CMN_f
RFD_CF_f
SMP_CF_f

RFD_DATA_nR

Be sure to return [Layout sections automatically] to “Yes”, after pressing the “OK” button.



Press the right-hand side “ ... “ button by [ROM to RAM mapped section], display the “Text Edit” screen, and add the section for copying to RAM from ROM.

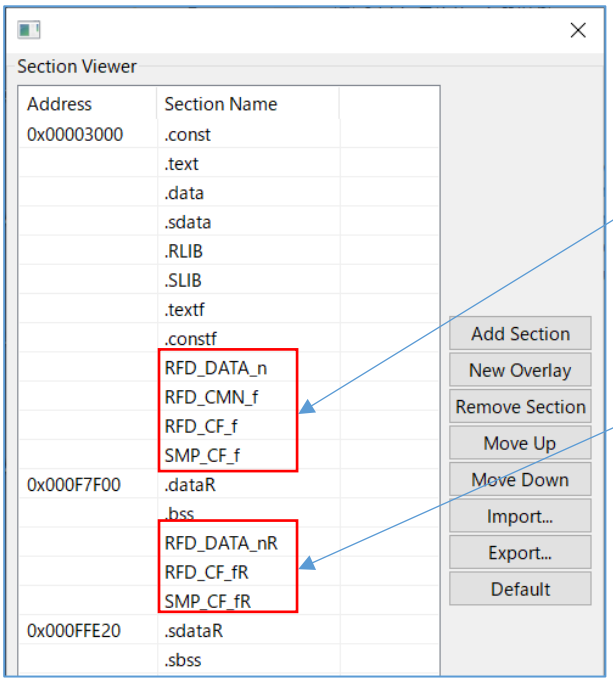


ROM to RAM mapped section
.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR

・The addition of the sections for code flash memory reprogramming on e^2 studio

Add sections necessary for code flash memory reprogramming on a “Section Viewer”.

Add to the program area: RFD_DATA_n, RFD_CMN_f, RFD_CF_f, SMP_CF_f
Add to the RAM area: RFD_DATA_nR, RFD_CF_fR, SMP_CF_fR

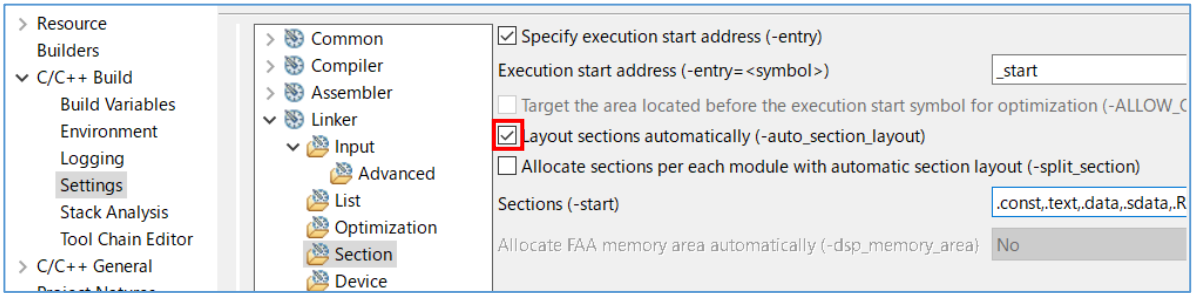


Additional sections
RFD_DATA_n
RFD_CMN_f
RFD_CF_f
SMP_CF_f

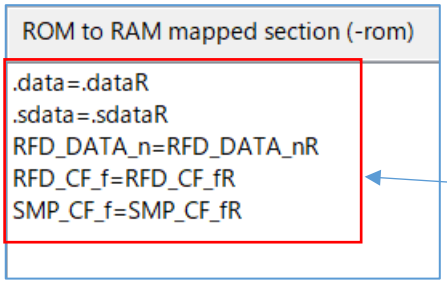
RFD_DATA_nR
RFD_CF_fR
SMP_CF_fR

-

Be sure to put a check mark to [Layout sections automatically (-auto_section_layout)], after pressing the “OK” button.



Select “C/C++ Build” [Settings] - “Linker” [Output], display the “ROM to RAM mapped section (-rom)” screen, and add the section for copying to RAM from ROM.

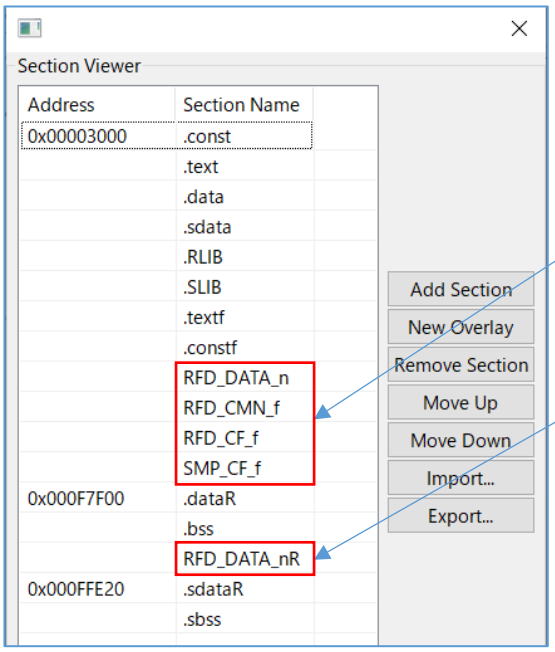


ROM to RAM mapped section
.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR
RFD_CF_f=RFD_CF_fR
SMP_CF_f=SMP_CF_fR

・The addition of the sections for code flash memory reprogramming on e^2 studio(Bank programming)

Add sections necessary for code flash memory reprogramming on a “Section Viewer”.

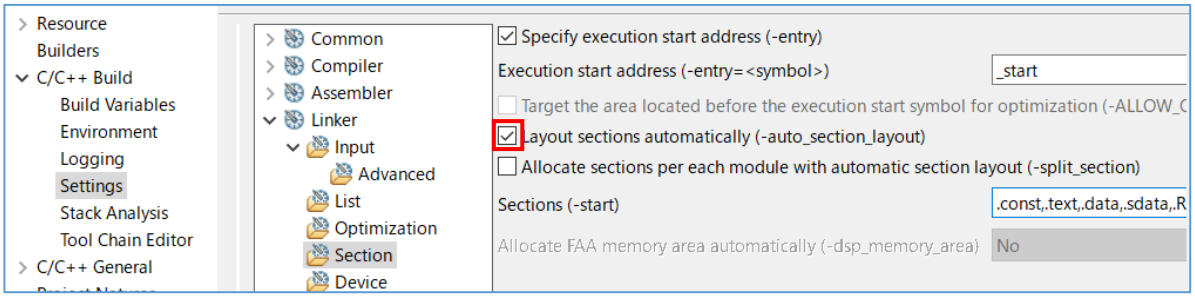
Add to the program area: RFD_DATA_n, RFD_CMN_f, RFD_CF_f, SMP_CF_f
Add to the RAM area: RFD_DATA_nR



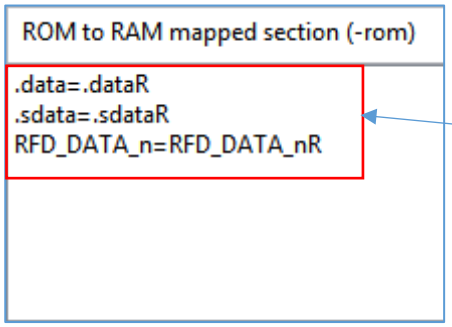
Additional sections
RFD_DATA_n
RFD_CMN_f
RFD_CF_f
SMP_CF_f

RFD_DATA_nR

Be sure to put a check mark to [Layout sections automatically (-auto_section_layout)], after pressing the “OK” button.



Select “C/C++ Build” [Settings] - “Linker” [Output], display the “ROM to RAM mapped section (-rom)” screen, and add the section for copying to RAM from ROM.



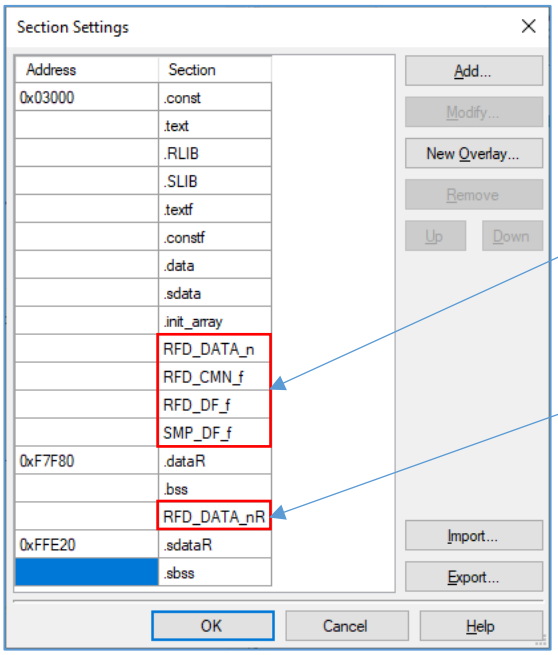
ROM to RAM mapped section
.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR

(2) The addition of the sections for data flash memory reprogramming

・The addition of the sections for data flash memory reprogramming on CS+

Add sections necessary for data flash memory reprogramming on a “section settings” screen.

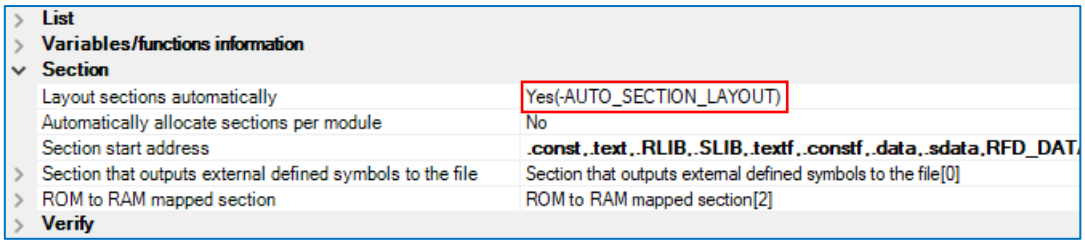
Add to the program area: RFD_DATA_n, RFD_CMN_f, RFD_DF_f, SMP_DF_f
Add to the RAM area: RFD_DATA_nR



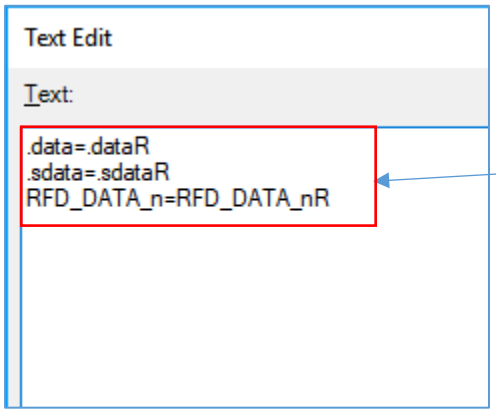
Additional sections
RFD_DATA_n
RFD_CMN_f
RFD_DF_f
SMP_DF_f

RFD_DATA_nR

Be sure to return [Layout sections automatically] to “Yes”, after pressing the “OK” button.



Press the right-hand side “ ... ” button by [ROM to RAM mapped section], display the “Text Edit” screen, and add the section for copying to RAM from ROM.

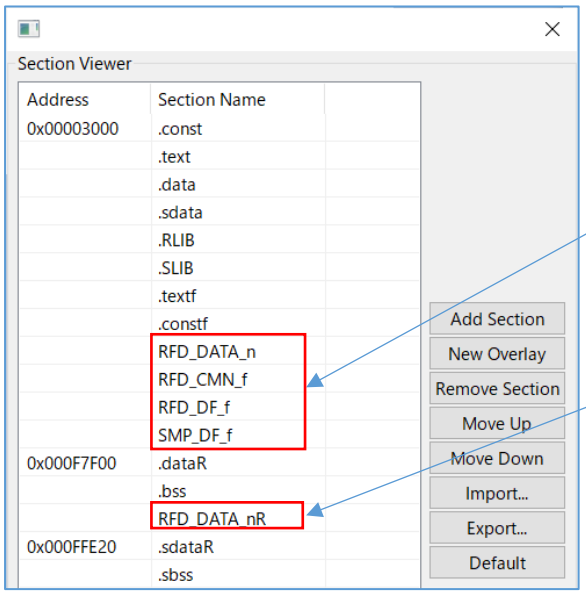


ROM to RAM mapped section
.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR

・The addition of the sections for data flash memory reprogramming on e^2 studio

Add sections necessary for data flash memory reprogramming on a “Section Viewer”.

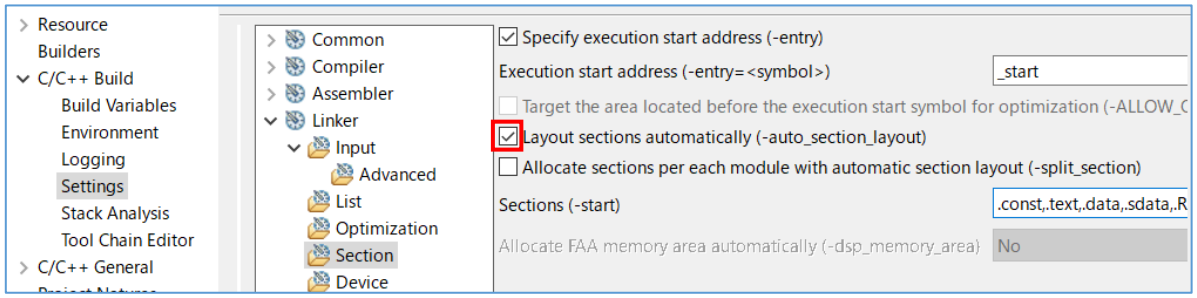
Add to the program area: RFD_DATA_n, RFD_CMN_f, RFD_DF_f, SMP_DF_f
Add to the RAM area: RFD_DATA_nR



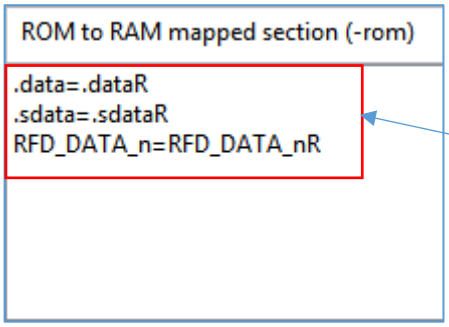
Additional sections
RFD_DATA_n
RFD_CMN_f
RFD_DF_f
SMP_DF_f

RFD_DATA_nR

Be sure to put a check mark to [Layout sections automatically (-auto_section_layout)], after pressing the “OK” button.



Select “C/C++ Build” [Settings] - “Linker” [Output], display the “ROM to RAM mapped section (-rom)” screen, and add the section for copying to RAM from ROM.



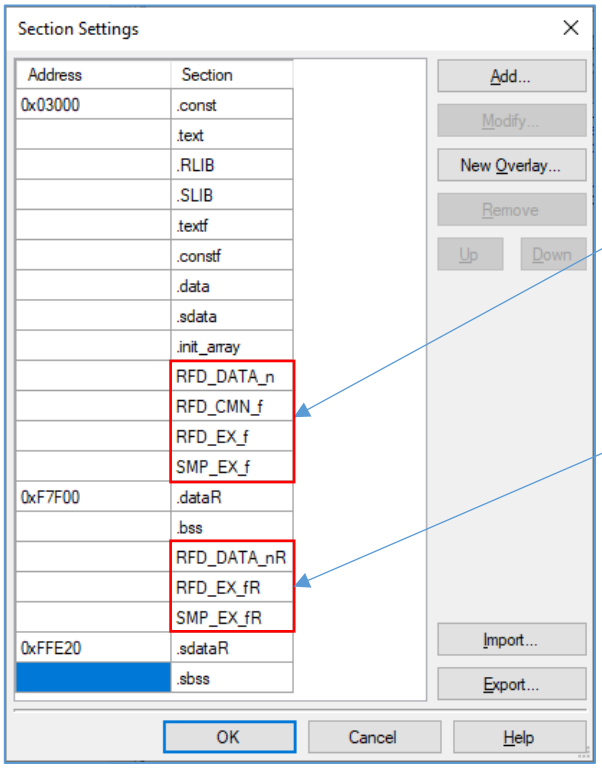
ROM to RAM mapped section
.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR

(3) The addition of the sections for extra area (FSW) reprogramming

・The addition of the sections for extra area (FSW) reprogramming on CS+

Add sections necessary for extra area (FSW) reprogramming on a “section settings” screen.

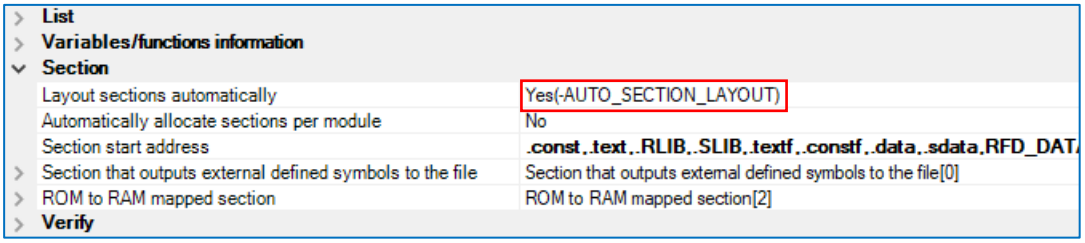
Add to the program area: RFD_DATA_n, RFD_CMN_f, RFD_EX_f, SMP_EX_f
Add to the RAM area: RFD_DATA_nR, RFD_EX_fR, SMP_EX_fR



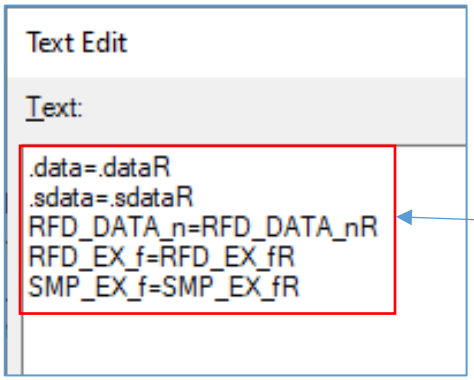
Additional sections
RFD_DATA_n
RFD_CMN_f
RFD_EX_f
SMP_EX_f

RFD_DATA_nR
RFD_EX_fR
SMP_EX_fR

Be sure to return [Layout sections automatically] to “Yes”, after pressing the “OK” button.



Press the right-hand side “ ... “ button by [ROM to RAM mapped section], display the “Text Edit” screen, and add the section for copying to RAM from ROM.

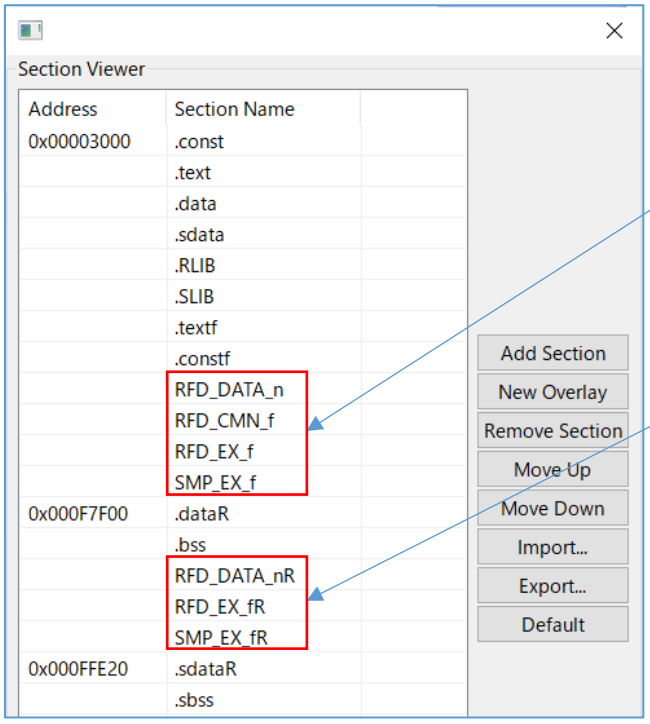


ROM to RAM mapped section
.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR
RFD_EX_f=RFD_EX_fR
SMP_EX_f=SMP_EX_fR

・The addition of the sections for extra area (FSW) reprogramming on e^2 studio

Add sections necessary for extra area (FSW) reprogramming on a “Section Viewer”.

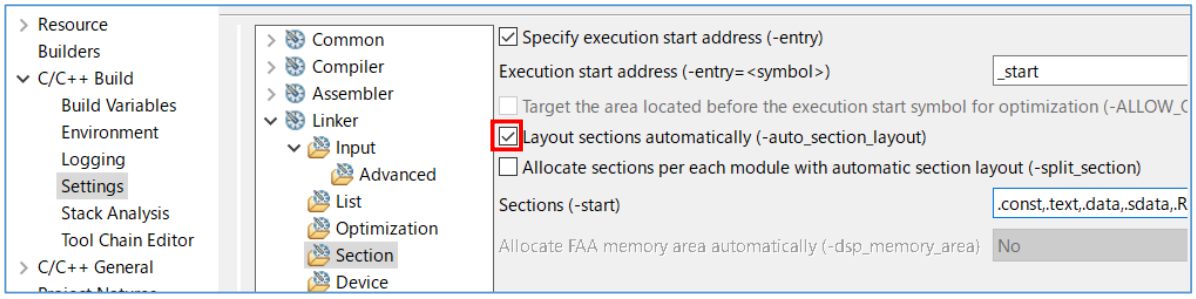
Add to the program area: RFD_DATA_n, RFD_CMN_f, RFD_EX_f, SMP_EX_f
Add to the RAM area: RFD_DATA_nR, RFD_EX_fR, SMP_EX_fR



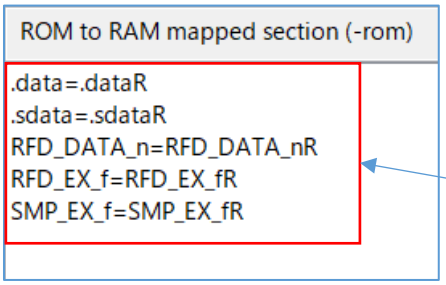
Additional sections
RFD_DATA_n
RFD_CMN_f
RFD_EX_f
SMP_EX_f

RFD_DATA_nR
RFD_EX_fR
SMP_EX_fR

Be sure to put a check mark to [Layout sections automatically (-auto_section_layout)], after pressing the “OK” button.



Select “C/C++ Build” [Settings] - “Linker” [Output], display the “ROM to RAM mapped section (-rom)” screen, and add the section for copying to RAM from ROM.



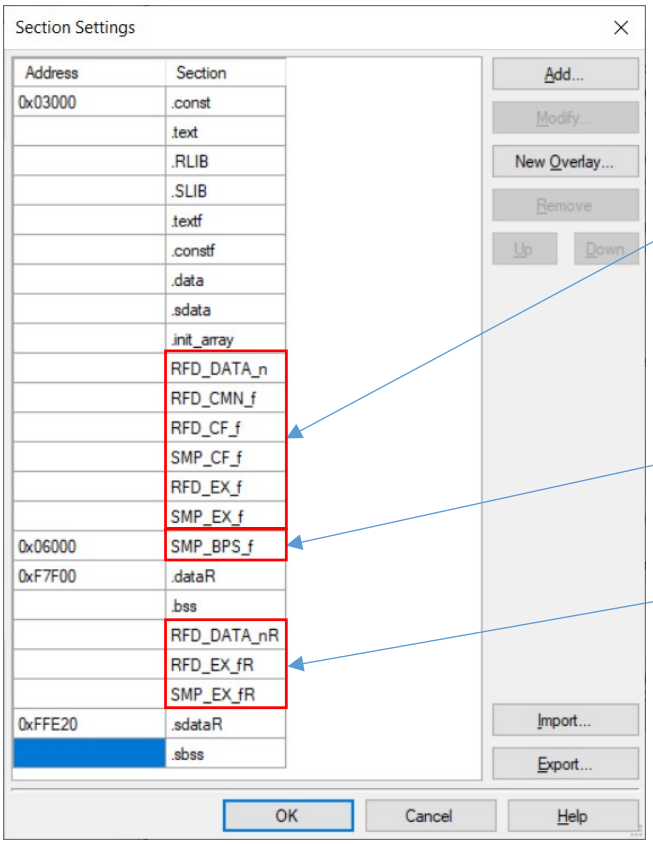
ROM to RAM mapped section
.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR
RFD_EX_f=RFD_EX_fR
SMP_EX_f=SMP_EX_fR

(4) The addition of the sections for bank programming / bank swapping control

・The addition of the sections for bank programming / bank swapping control on CS+

Add sections necessary for bank programming / bank swapping control on a “section settings” screen.

Add to the program area: RFD_DATA_n, RFD_CMN_f, RFD_CF_f, SMP_CF_f, RFD_EX_f, SMP_EX_f
Program code after bank swap execution: SMP_BPS_f (This Sample : locate code to 0x6000 on ROM)
Add to the RAM area: RFD_DATA_nR, RFD_EX_fR, SMP_EX_fR

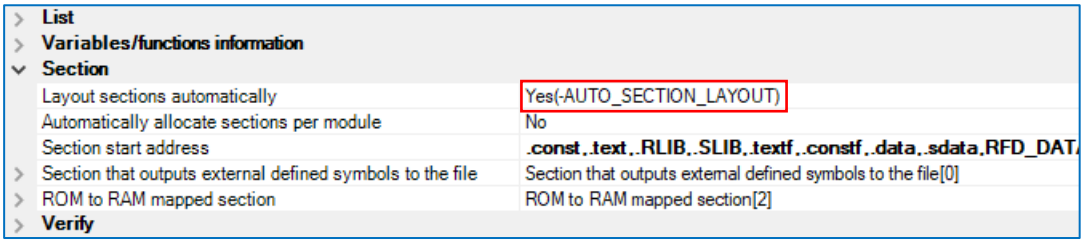


Additional sections
RFD_DATA_n
RFD_CMN_f
RFD_CF_f
SMP_CF_f
RFD_EX_f
SMP_EX_f

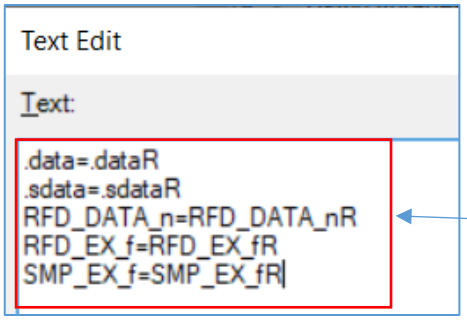
SMP_BSP_f

RFD_DATA_nR
RFD_EX_fR
SMP_EX_fR

Be sure to return [Layout sections automatically] to “Yes”, after pressing the “OK” button.



Press the right-hand side “ ... ” button by [ROM to RAM mapped section], display the “Text Edit” screen, and add the section for copying to RAM from ROM.

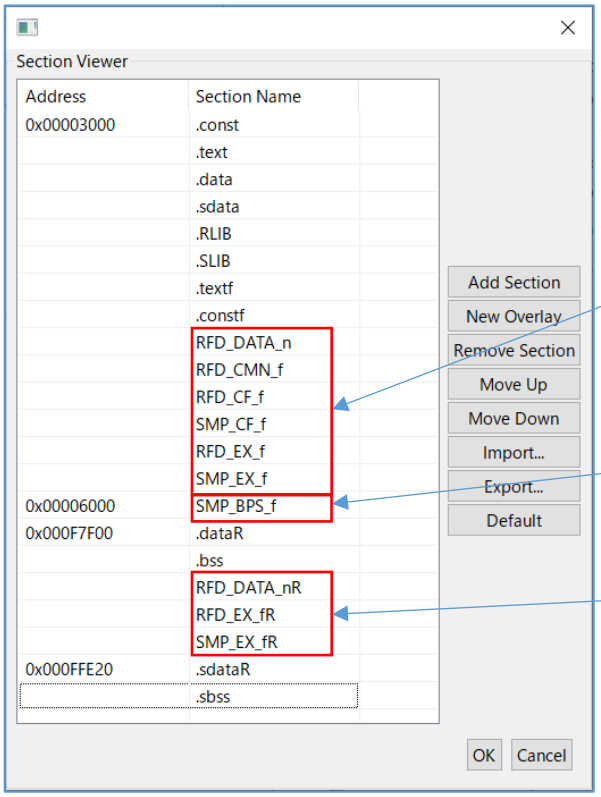


ROM to RAM mapped section
.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR
RFD_EX_f=RFD_EX_fR
SMP_EX_f=SMP_EX_fR

- The addition of the sections for bank programming / bank swapping control on e^2 studio

  Add sections necessary for bank programming / bank swapping control on a “Section Viewer”.

  Add to the program area: RFD_DATA_n, RFD_CMN_f, RFD_CF_f, SMP_CF_f, RFD_EX_f, SMP_EX_f
  Program code after bank swap execution: SMP_BPS_f (This Sample : locate code to 0x6000 on ROM)
  Add to the RAM area: RFD_DATA_nR, RFD_EX_fR, SMP_EX_fR

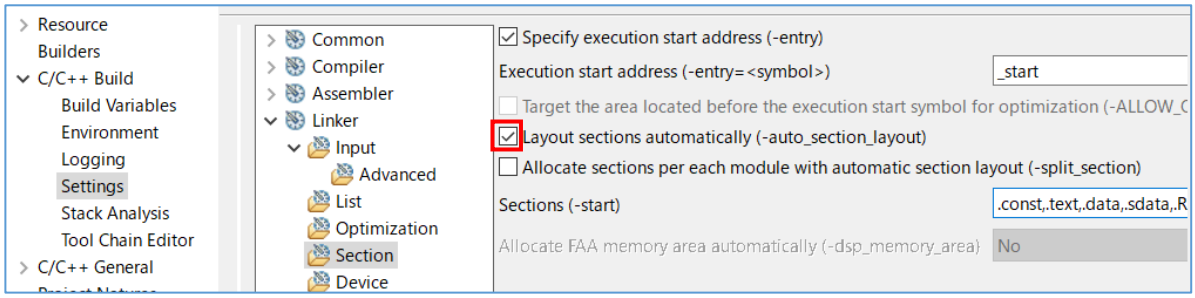


Additional sections
RFD_DATA_n
RFD_CMN_f
RFD_CF_f
SMP_CF_f
RFD_EX_f
SMP_EX_f

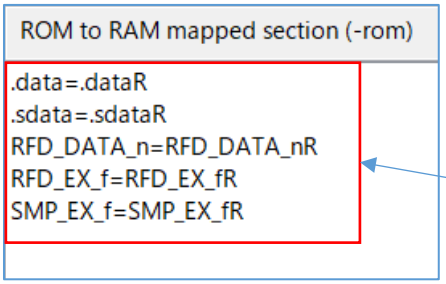
SMP_BSP_f

RFD_DATA_nR
RFD_EX_fR
SMP_EX_fR

Be sure to put a check mark to [Layout sections automatically (-auto_section_layout)], after pressing the “OK” button.



Select “C/C++ Build” [Settings] - “Linker” [Output], display the “ROM to RAM mapped section (-rom)” screen, and add the section for copying to RAM from ROM.



ROM to RAM mapped section
.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR
RFD_EX_f=RFD_EX_fR
SMP_EX_f=SMP_EX_fR

6.1.4 Debug Tool Settings

This section describes the contents of connection setting on a target board necessary in order 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. And a “RL78 E2 Lite Property” screen is 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 (“RFDRL78T11_PJ01 HardwareDebug”) of [Renesas GDB Hardware Debugging]. And the displayed “Debugger” tab performs debugging tool setting.

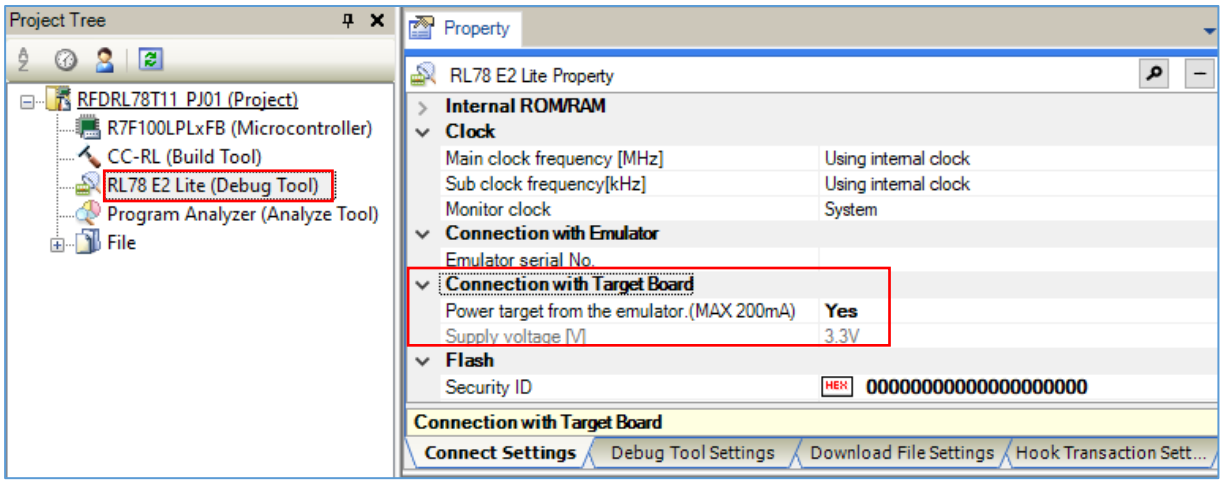
Note: The power is already supplied to the target board, or when power supply capacity is insufficient, the emulator including E2 Lite may be unable to supply power to a target board. Be sure to refer to “the user's manual and Additional Document for User's Manual (Notes on Connection of RL78)” for the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

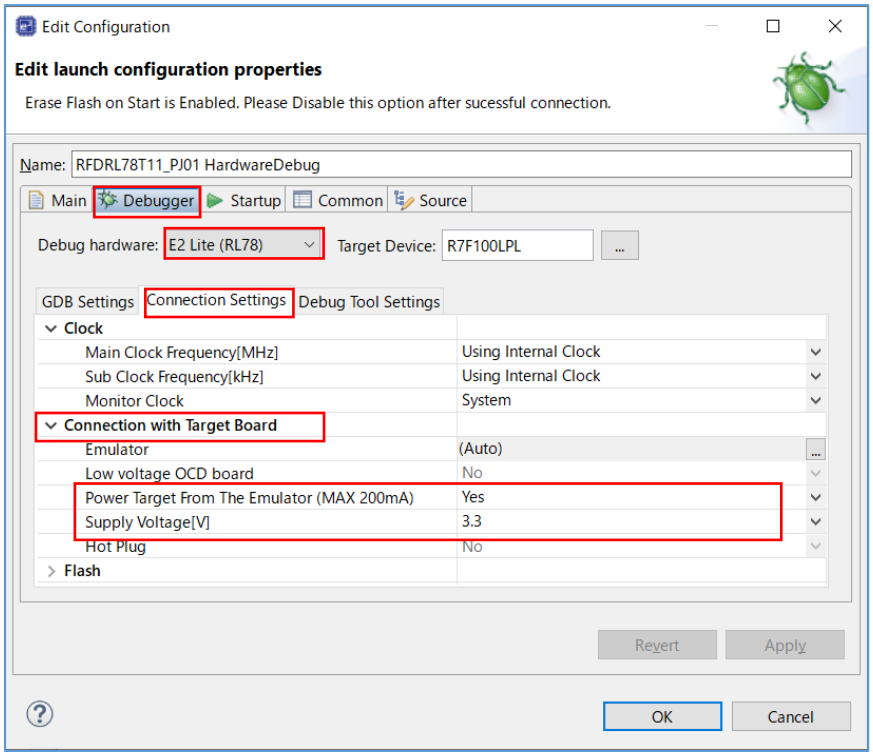
  - [Connection with Target Board] item

    In order 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

    In order 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

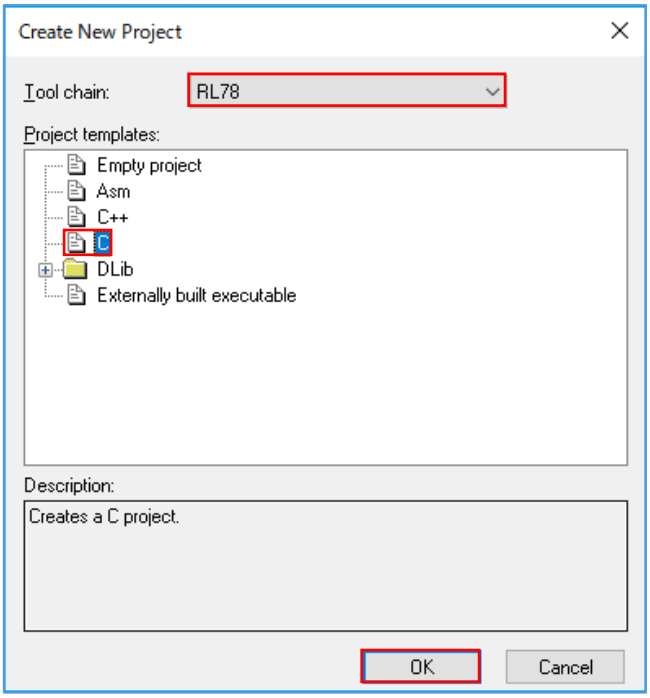
IAR Embedded Workbench can be used for an IAR compiler as an IDE. RFD RL78 Type 11 is registered and built in the project created by the IDE. An example of creating a sample project in case each IDE is used is shown. Because to understand an IAR compiler and each IDE, it is necessary to refer to the user's manual of each tool product.

6.2.1 Example of Creating a Sample Project

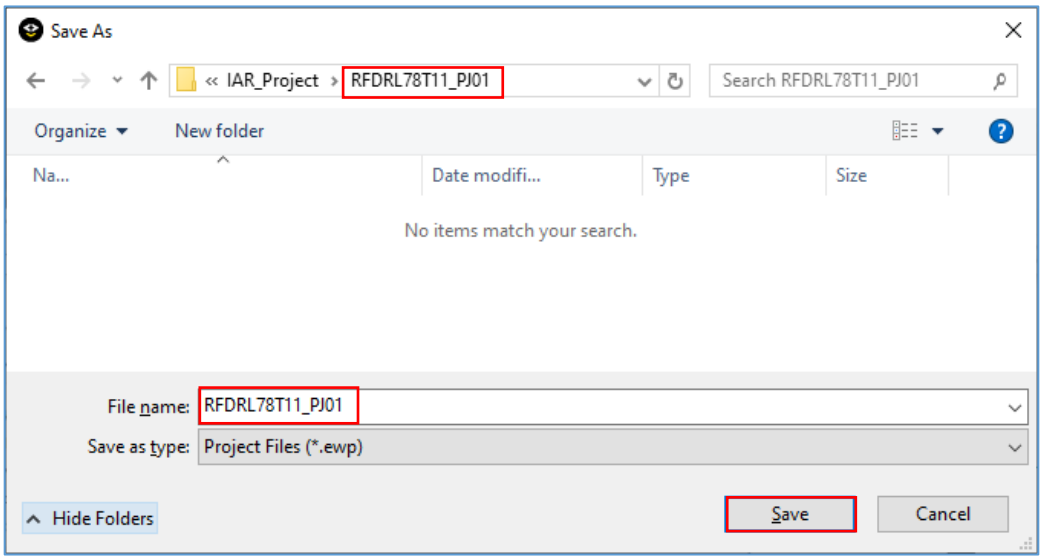
(1) An example of creating a sample project which used IAR Embedded Workbench (IDE)

・The 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.

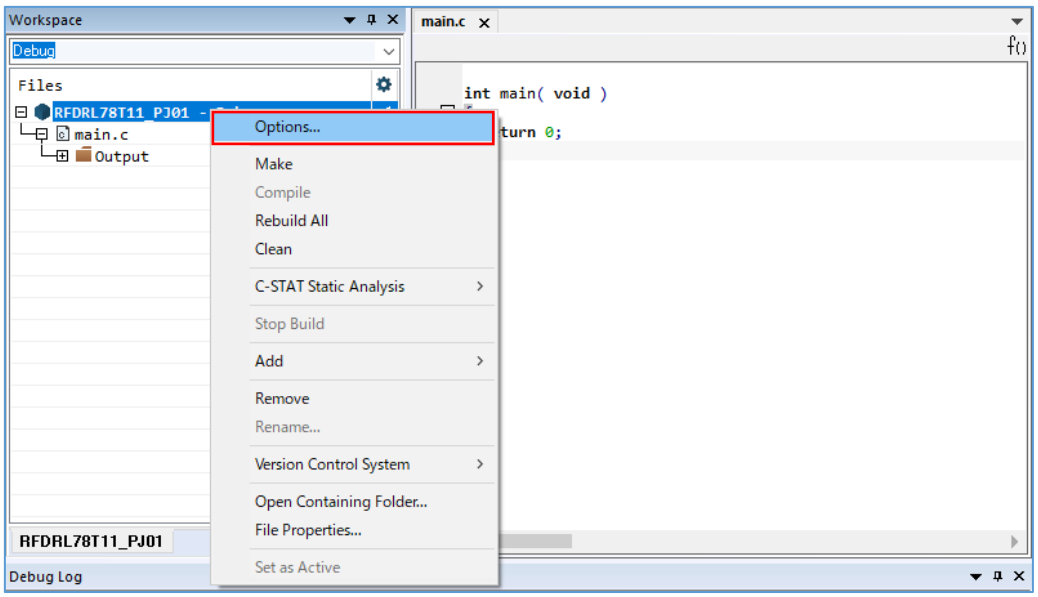


- Create "RFDRL78T11_PJ01" folder temporarily, and move into a folder.
- The Project File name is temporarily set to "RFDRL78T11_PJ01".

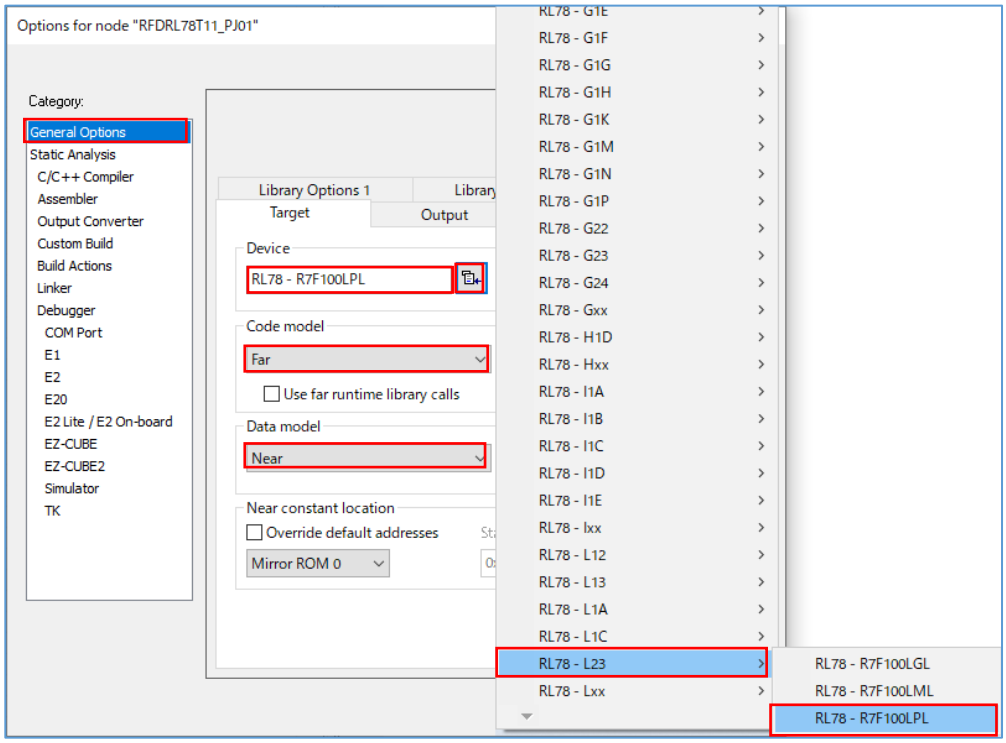


(2) Selection of a target device

・On IAR Embedded Workbench, I click the right mouse button of the project (“RFDRL78T11_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 - L23” - “RL78 - R7F100LPL”. Select “Far” as [code model] and select “Near” as [Data model].

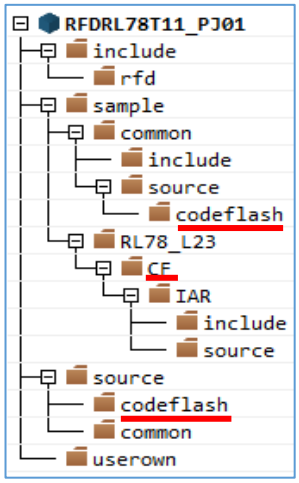


6.2.2 Example of Registration of Target Folders and Target Files

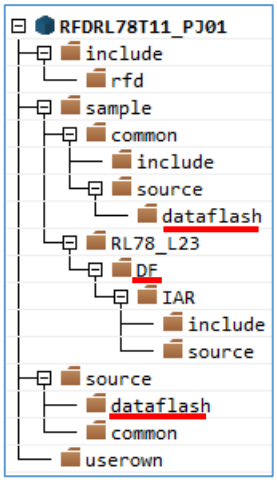
Using RFD RL78 Type 11, when programming each area [(1) code flash memory, (2) data flash memory, (3) extra area], and (4) bank programming / bank swapping control, the example which registers necessary files is shown. Each folder of the RFD RL78 Type 11 source-program file is “include”, “source”, “userown”, 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 RFD RL78 Type 11, and files are registered is shown. (Registering without making a group is also possible.)

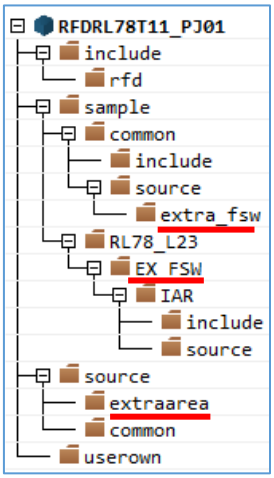
The example which added the group of each area [(1) Code flash memory, (2) Data flash memory, (3) Extra area, and (4) Bank Programming / Bank Swapping Control] is shown. (The group name which changes with areas is shown by “ ▬ “.)



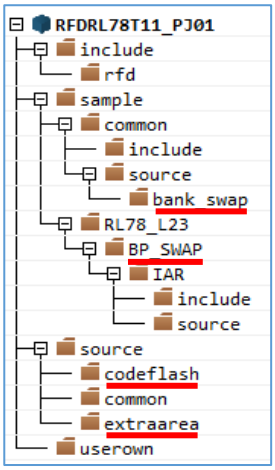
(1) Code flash memory



(2) Data flash memory



(3) Extra area



(4) Bank Programming / Bank Swapping Control

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

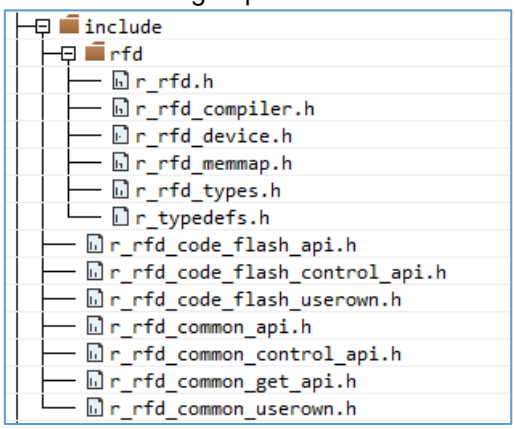
  There are files added automatically in the created project. The same file as these exists also in the “sample” folder of RFD RL78 Type 11. Therefore, using the function of IDE, select those files from tree and excludes from a project.

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

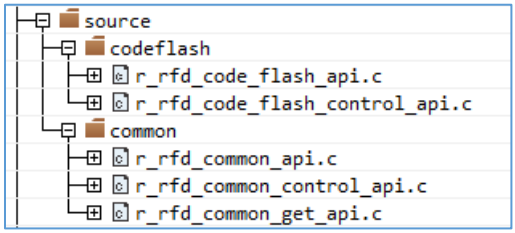
(1) Registration of the groups and files of the target in the case of reprogramming code flash memory

The groups (“include”, “source”, “userown”, “sample”) and source program file which are included in RFD RL78 Type 11 to register are shown below.

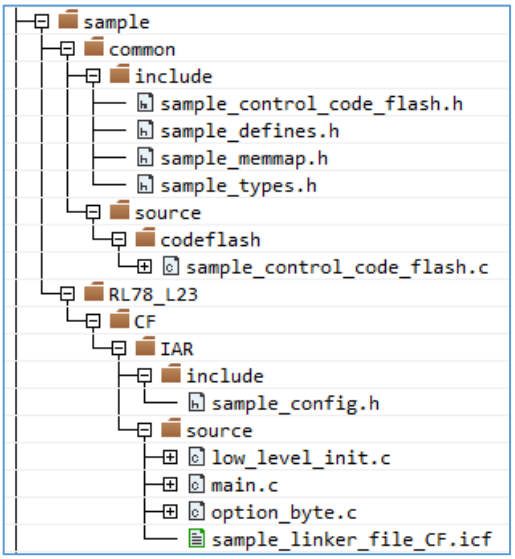
in the “include” group



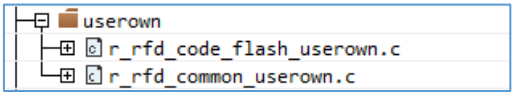
in the “source” group



in the “sample” group



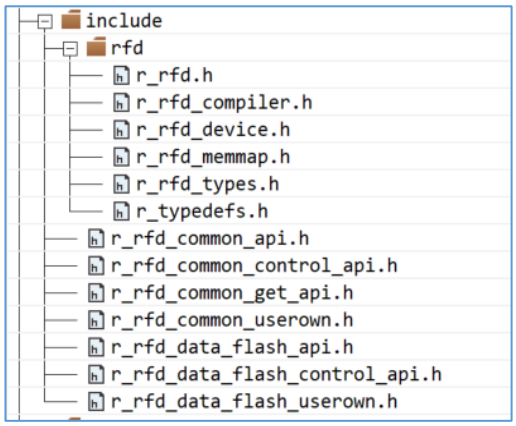
in the “userown” group



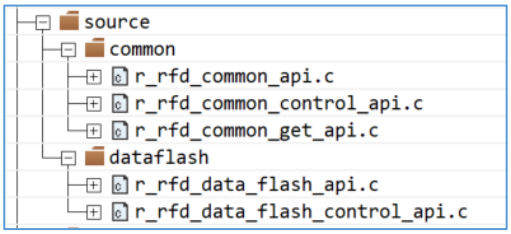
(2) Registration of the groups and files of the target in the case of reprogramming data flash memory

The groups (“include”, “source”, “userown”, “sample”) and source program file which are included in RFD RL78 Type 11 to register are shown below.

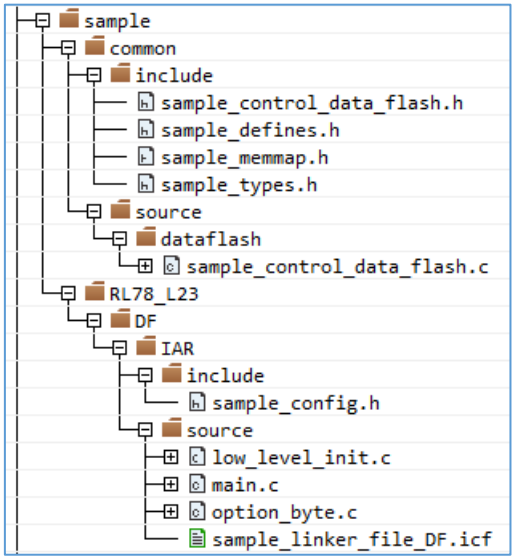
in the “include” group



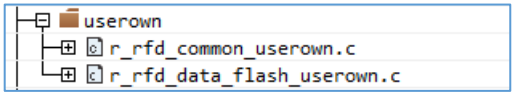
in the “source” group



in the “sample” group



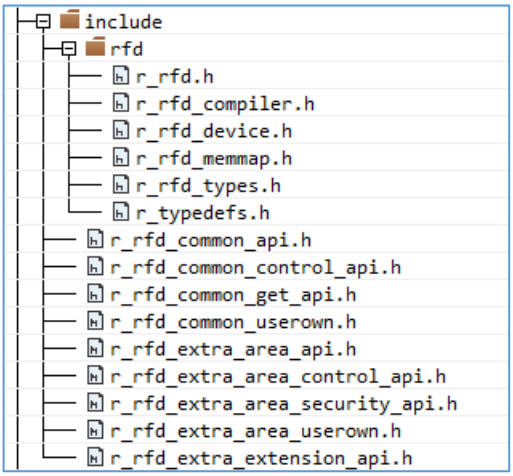
in the “userown” group



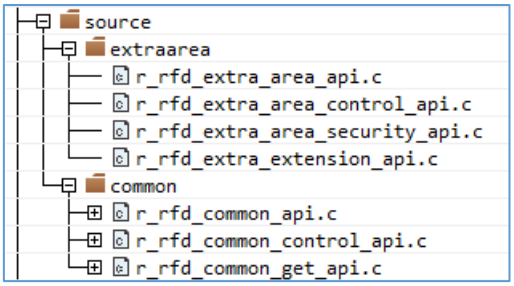
(3) Registration of the groups and files of the target in the case of reprogramming extra area (FSW setting)

The groups (“include”, “source”, “userown”, “sample”) and source program file which are included in RFD RL78 Type 11 to register are shown below.

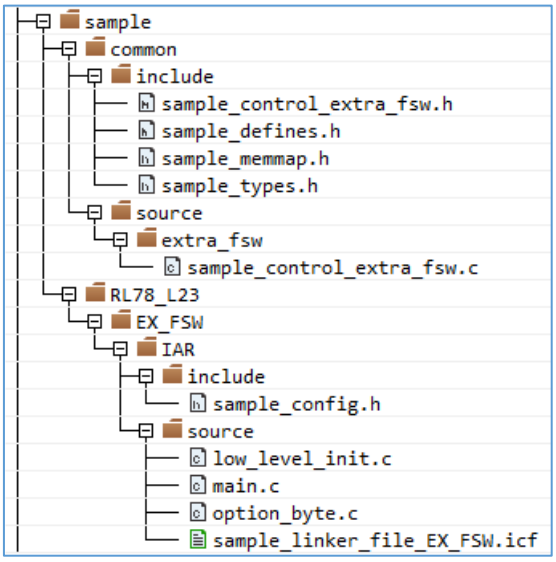
in the “include” group



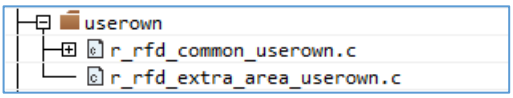
in the “source” group



in the “sample” group



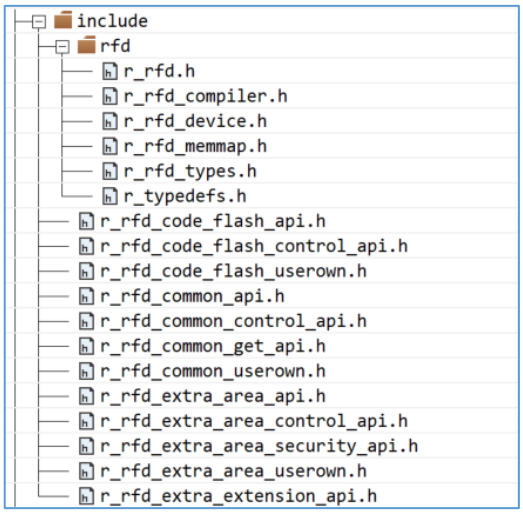
in the “userown” group



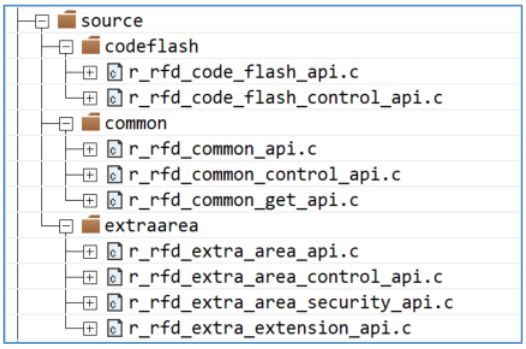
(4) Registration of the groups and files of the target in the case of bank programming / bank swapping control execution

The groups (“include”, “source”, “userown”, “sample”) and source program file which are included in RFD RL78 Type 11 to register are shown below.

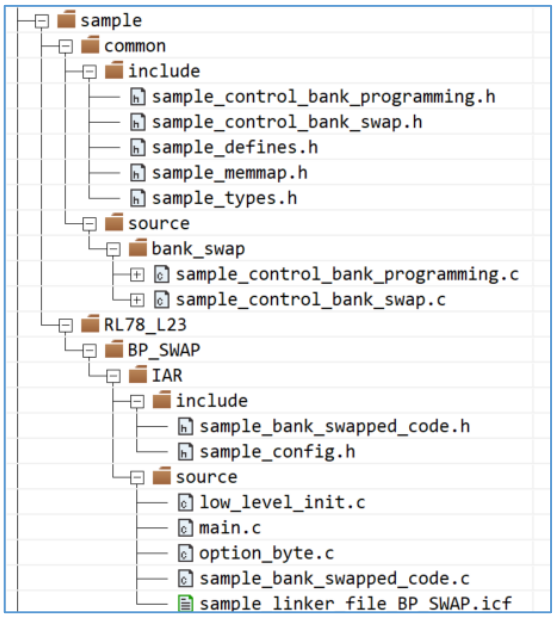
in the “include” group



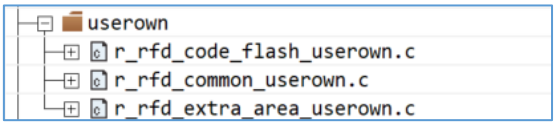
in the “source” group



in the “sample” group



in the “userown” group

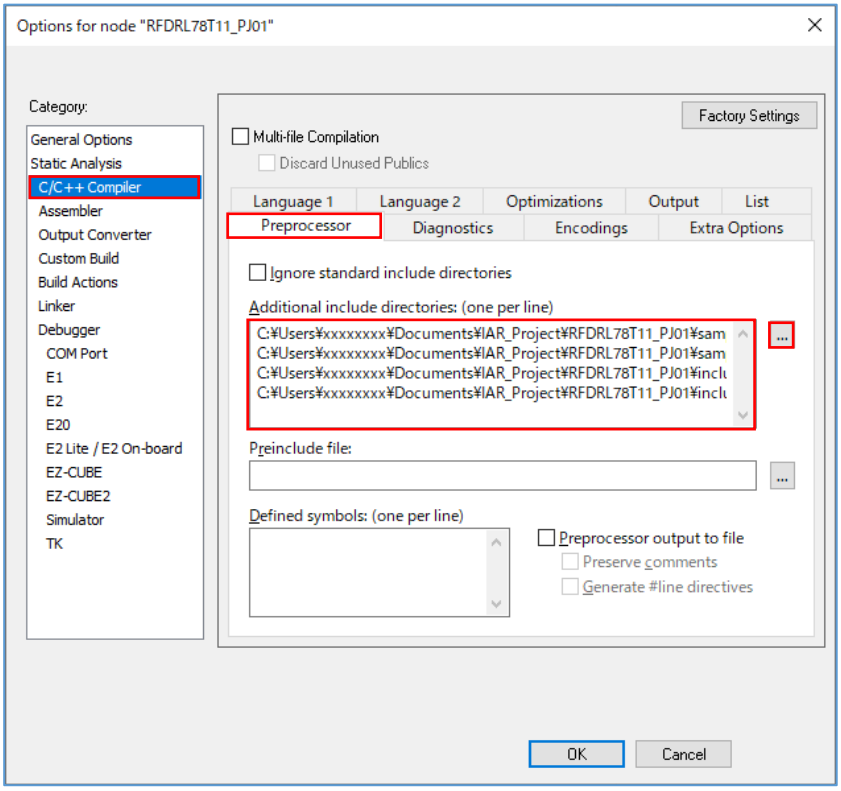


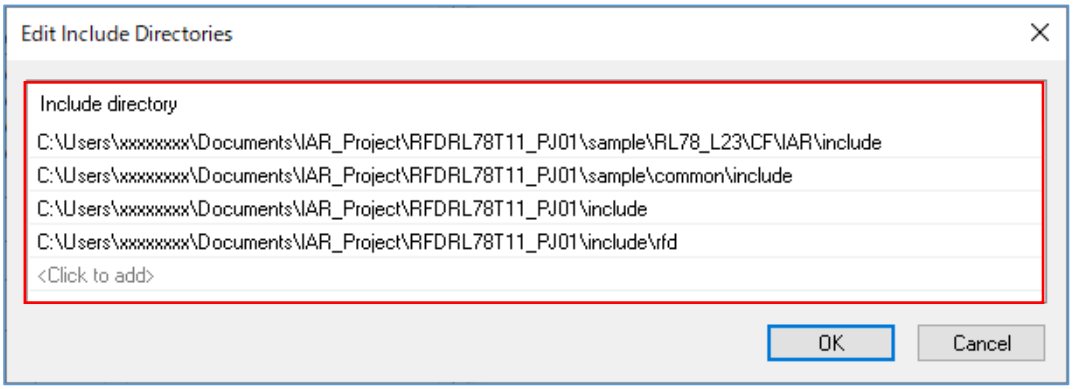
6.2.3 Integrated Development Environment (IDE) Settings

Set IDE setting necessary in order to build RFD RL78 Type 11 using an IAR compiler.
IAR Embedded Workbench: Click the right mouse button for the project (“RFDRL78T11_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 setting.

  It is the example which placed each folder (“include”, “source”, “userown”, “sample”) of the source program file of RFD RL78 Type 11 on “C:\Users\xxxxxxxx\Documents\IAR_Project\”.

(1) Code flash memory reprogramming

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T11_PJ01\sample\RL78_L23\CF\IAR\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T11_PJ01\sample\common\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T11_PJ01\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T11_PJ01\include\rfd

(2) Data flash memory reprogramming

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T11_PJ01\sample\RL78_L23\DF\IAR\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T11_PJ01\sample\common\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T11_PJ01\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T11_PJ01\include\rfd

(3) Extra area (FSW) reprogramming

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T11_PJ01\sample\RL78_L23\EX_FSW\IAR\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T11_PJ01\sample\common\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T11_PJ01\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T11_PJ01\include\rfd

(4) Bank programming / bank swapping control

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T11_PJ01\sample\RL78_L23\BP_SWAP\IAR\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T11_PJ01\sample\common\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T11_PJ01\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T11_PJ01\include\rfd

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.

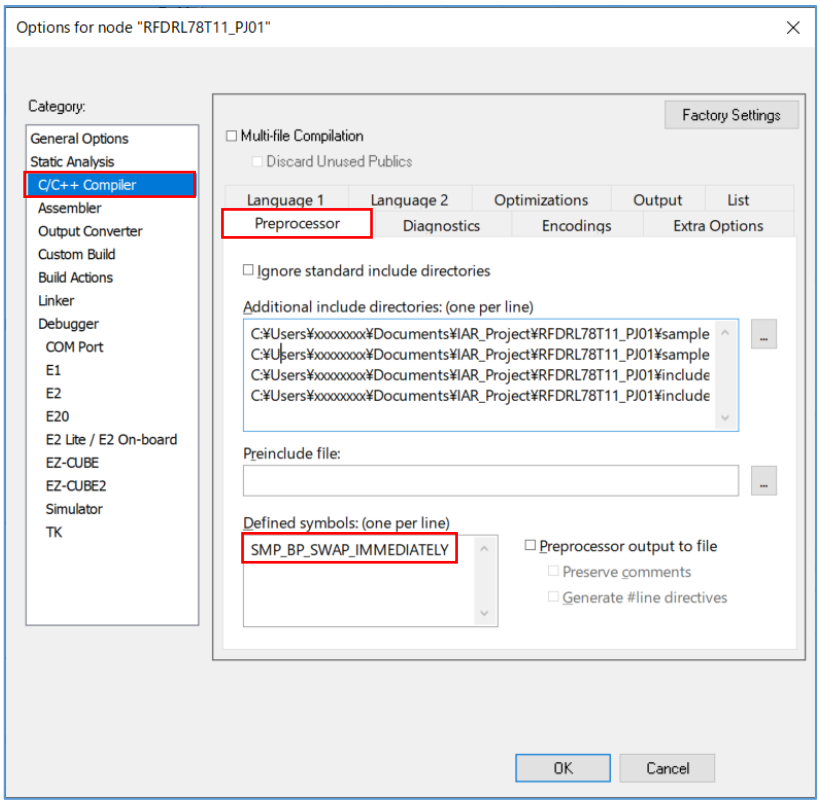
6.2.3.2 The Setting of User Definition Macro (IAR Compiler)

In the case which selects “active bank swapping execution” (immediate execution bank swapping) with the sample program function of bank swapping control, user definition macro (“SMP_BP_SWAP_IMMEDIATELY”) is necessary. In the case which selects “bank swapping execution after reset”, this definition is unnecessary.

- On IAR Embedded Workbench, the macro for selecting “active bank swapping execution (Immediate execution bank swapping)” is defined in “Preprocessor” tab.
  - Define the following macro in the column of [Defined symbols: (one per line)]. Definition macro differs by each device to be used.

    Macro for selecting “active bank swapping execution” (Immediate execution bank swapping):

    SMP_BP_SWAP_IMMEDIATELY



6.2.3.3 Stack Settings

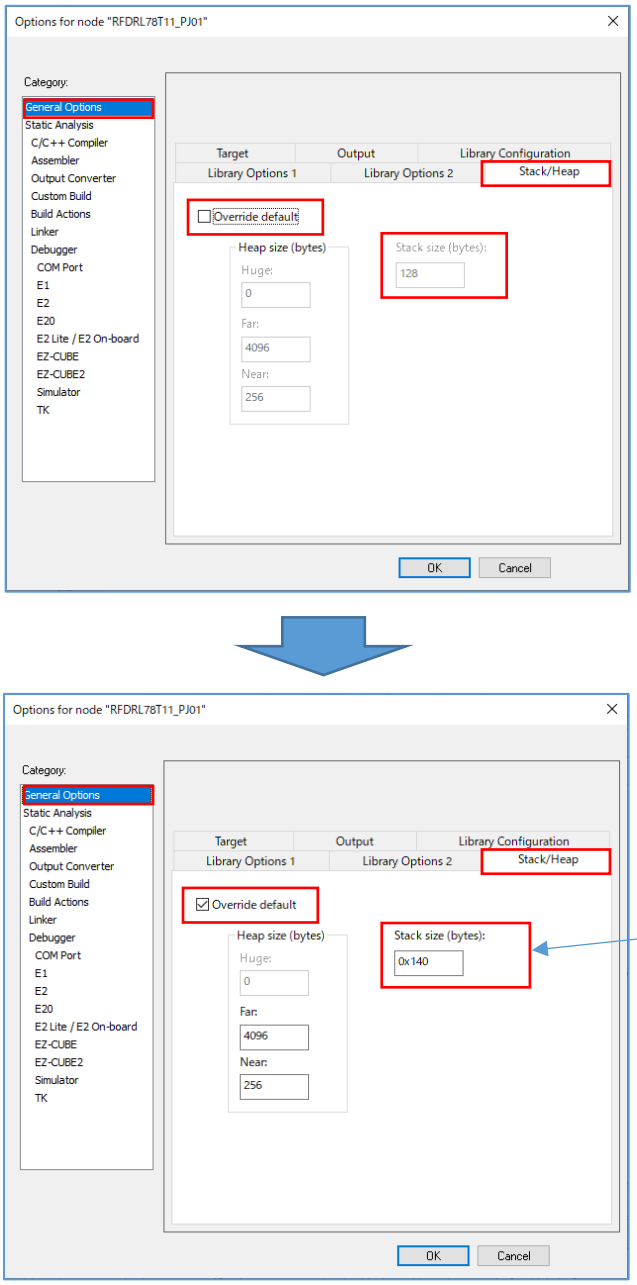
The initial value of the stack size of IAR Embedded Workbench for RL78 is 128 bytes. When the stack used by a user program and RFD RL78 Type 11 exceeds this size, it is necessary to modify use stack size.

The code flash reprogramming obtains the buffer for 0x40(64 bytes) data to a stack. Therefore, the stack size about 0x140(64+256)bytes is necessary.

The data flash reprogramming or extra area reprogramming takes adding user processing into consideration. Therefore, the stack size about 0x100(256)bytes is necessary.

The code flash reprogramming for bank programming / bank swapping control obtains the buffer for 1-block(2 KB) data to a stack. Therefore, the stack size about 0x900(2048+256)bytes is necessary.

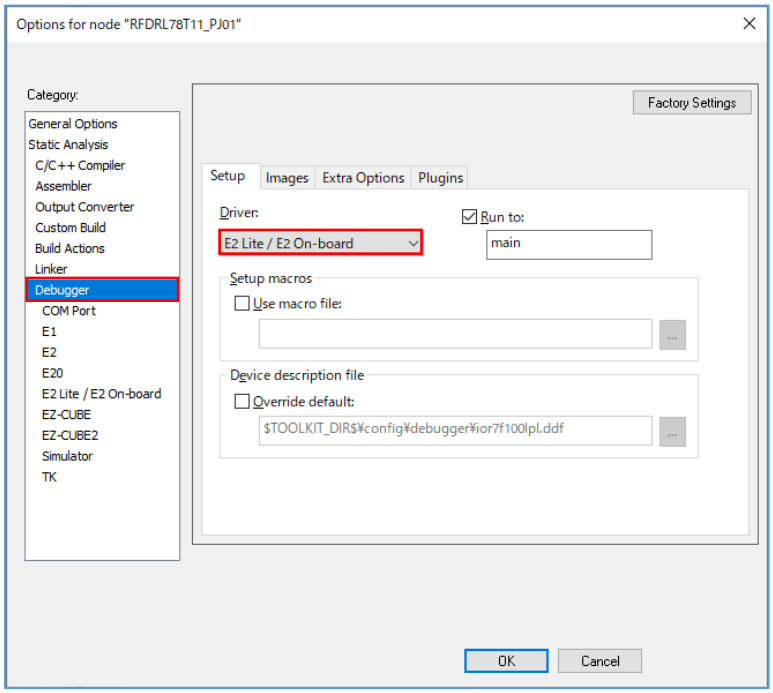
<<The example of a stack setting>>



Code flash reprogramming : 0x140 (64+256)bytes
Data flash reprogramming : 0x100 (256)bytes
Extra area reprogramming : 0x100 (256)bytes
Bank programming / bank swapping :
0x900(2048+256)bytes

6.2.3.4 Debugger Settings

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

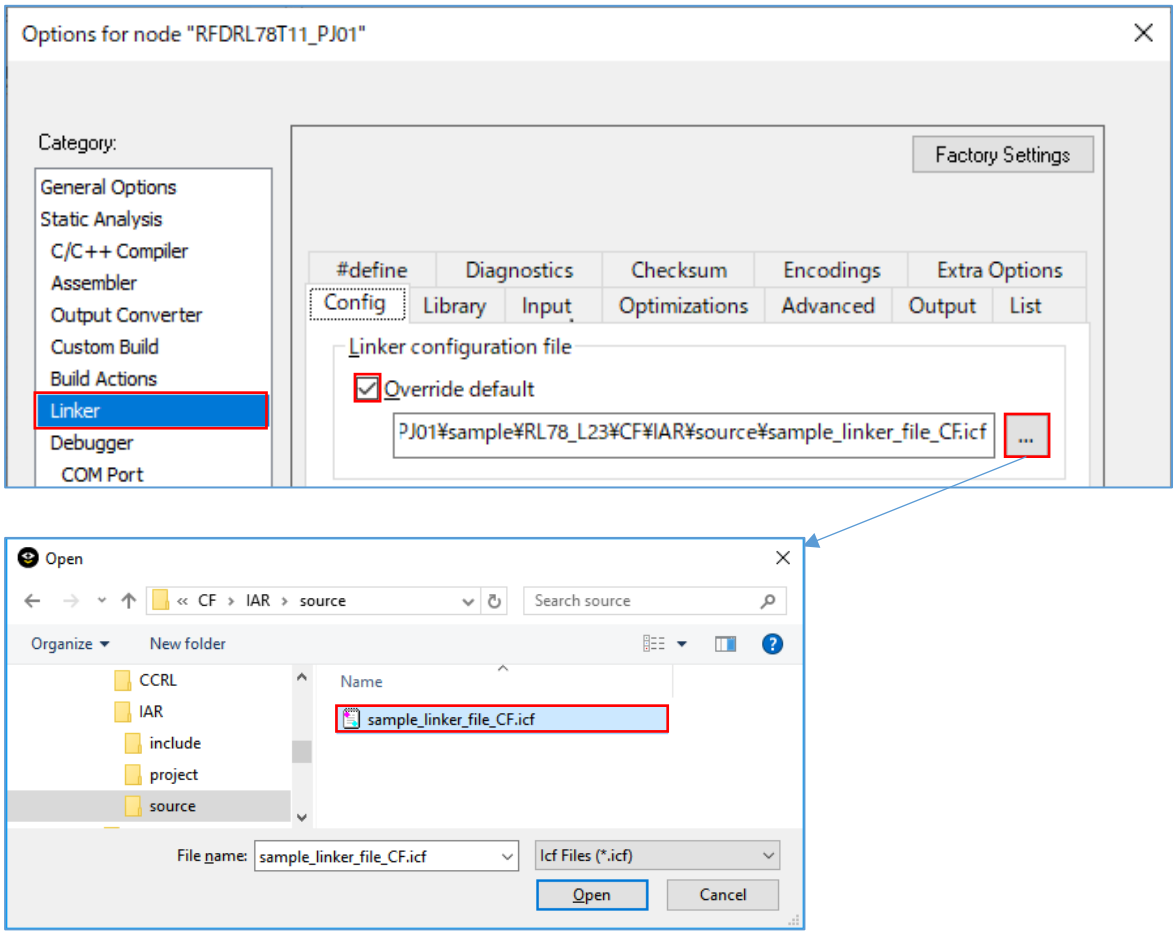


Note: Refer to each reference manual of IAR Embedded Workbench about the other items to be set.

6.2.4 Linker Configuration File(.icf) Settings

On IAR Embedded Workbench, Linker configuration file (*. icf) describes link setting executed by building. Select “Options” by the click right mouse buttan of project with tree. Select [Linker] by “Category” in the displayed window, And put a check mark to “Override default” of the [Config] tab. Select Linker configuration file (*. icf) in the “Open” window of “ ... ” button. Select the “sample_linker_file_(area name).icf” file prepared for RFD RL78 Type 11. Linker configuration file (*. icf) for every reprogramming area is as follows.

- For code flash memory reprogramming: sample_linker_file_CF.icf (\Sample\RL78_L23\CF\IAR\source\)
- For data flash memory reprogramming: sample_linker_file_DF.icf (\Sample\RL78_L23\DF\IAR\source\)
- For extra area (FSW): sample_linker_file_EX_FSW.icf (\Sample\RL78_L23\EX_FSW\IAR\source\)
- For bank programming / bank swapping:
  sample_linker_file_BP_SWAP.icf (\Sample\RL78_L23\BP_SWAP\IAR\source\)



Note: Refer to each reference manual of IAR Embedded Workbench about the descriptive content of Linker configuration file, and the details of the description method.

6.2.4.1 Section Settings

The outline of the section added to Linker configuration file (*. icf) currently prepared by RFD RL78 Type 11 is explained.

Note: Refer to each reference manual of IAR Embedded Workbench about the section setting method and the detail of functions for Linker configuration file.

・The setting outline of the section item described to Linker configuration file (*. icf) of RFD RL78 Type 11.

(1) The addition of the sections for code flash memory reprogramming

Add the initial value of each section of RFD_DATA, RFD_CF, and SMP_CF to ROM area (ROM_far).
Add RFD_CMN to ROM area (ROM_far).
The sections used in a RAM area (RAM_near, RAM_code) need to be copied to RAM.

- The additional section of the ROM_far area (Program, and data and program for copying to RAM area to be placed in ROM area) : RFD_DATA_init, RFD_CMN, RFD_CF_init, SMP_CF_init
- The additional section of RAM_near area (Data copied from ROM area) : RFD_DATA
- The additional section of RAM_code area (Program copied from ROM area) : RFD_CF, SMP_CF

(2) The addition of the sections for data flash memory reprogramming

Add the initial value of each section of RFD_DATA to ROM area (ROM_far).
Add RFD_CMN, RFD_DF, and SMP_DF to ROM area (ROM_far).
The sections used in a RAM area (RAM_near) need to be copied to RAM.

- The additional section of the ROM_far area (Program, and data for copying to RAM area to be placed in ROM area) : RFD_DATA_init, RFD_CMN, RFD_DF, SMP_DF
- The additional section of RAM_near area (Data copied from ROM area) : RFD_DATA

(3) The addition of the sections for extra area (FSW) reprogramming

Add the initial value of each section of RFD_DATA, RFD_EX, and SMP_EX to ROM area (ROM_far).
Add RFD_CMN to ROM area (ROM_far).
The sections used in a RAM area (RAM_near, RAM_code) need to be copied to RAM.

- The additional section of the ROM_far area (Program, and data and program for copying to RAM area to be placed in ROM area) : RFD_DATA_init, RFD_CMN, RFD_EX_init, SMP_EX_init
- The additional section of RAM_near area (Data copied from ROM area) : RFD_DATA
- The additional section of RAM_code area (program copied from ROM area) : RFD_EX, SMP_EX

(4) The case of bank programming / bank swapping control:

Add the initial value of each section of RFD_DATA, RFD_EX and SMP_EX to ROM area (ROM_far).
Add RFD_CMN, RFD_CF, and SMP_CF to ROM area (ROM_far).
The sections used in a RAM area (RAM_near, RAM_code) need to be copied to RAM. And SMP_BPS is the code programmed by bank programming, and needs to be located to 0x6000 as BPS_CODE block of the ROM area (ROM_far).

- The additional section of the ROM_far area (Program, and data and program for copying to RAM area to be placed in ROM area) : RFD_DATA_init, RFD_CMN, RFD_CF, SMP_CF, RFD_EX_init, SMP_EX_init
- The additional section to BPS_CODE block of the ROM_far area (Program) : SMP_BPS
- The additional section of RAM_near area (Data copied from ROM area) : RFD_DATA
- The additional section of RAM_code area (program copied from ROM area) : RFD_EX, SMP_EX

6.2.4.2 Option Bytes Settings

The Option bytes definition of RL78 is described in Linker configuration file (*. icf) of IAR Embedded Workbench attachment or the sample_linker_file_(area name).icf file prepared for RFD RL78 Type 11. The Option Bytes value for RFD RL78 Type 11 is described by 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 an Option Bytes definition of Linker configuration file for RFD RL78 Type 11 (*. icf).

```
define block OPT_BYTE with size = 4   { R_OPT_BYTE,
                                        ro section .option_byte,
                                        ro section OPTBYTE };
                                |
place at address mem:0x000C0            { block OPT_BYTE };
```

The example of description of the Option Bytes value in a “option_byte.c” file.

```
#pragma location = "OPTBYTE"
__root const unsigned char option_bytes[4] = {
    0x6E,   /* 01101110 */
            /* |||||||| */
            /* |||||||+-- Watchdog timer     */
            /* |||||||    operation stopped  */
            /* |||||||    in HALT/STOP mode  */
            /* ||||+++--- Watchdog timer     */
            /* ||||       overflow time is   */
            /* ||||       2^17 / fIL =       */
            /* ||||       3478.26 ms         */
            /* |||+------ Watchdog timer     */
            /* |||        operation disabled */
            /* |++------- 100% window open   */
            /* |          period             */
            /* +--------- Interval interrupt */
            /*            is not used        */
    0xFF,   /* 11111111 */
            /*        | */
            /*        +-- LVD reset mode */
    0xE8,   /* HS mode 32 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 “0x6EFFE8”.
  (WDT Stop, LVD [reset mode], 32MHz [The example for RL78/L23])

  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.

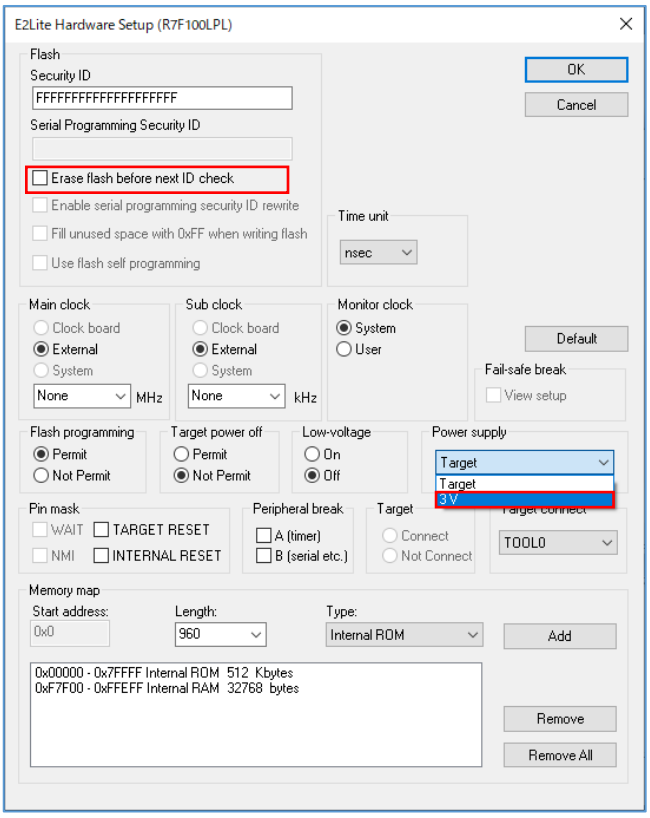
Note: In cases where a target cannot be connected by other causes, please confirm each reference manual from [Help] of IAR Embedded Workbench.

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” 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.



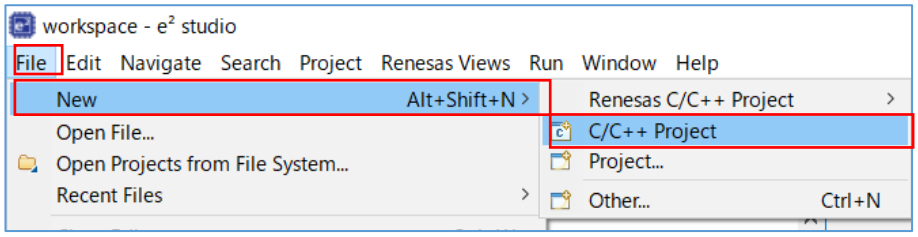
6.3 Creating a Project in the Case of Using LLVM Compiler

e² studio can be used for a LLVM compiler as an IDE. RFD RL78 Type 11 is registered and built in the project created by the IDE. An example of creating a sample project in case IDE is used is shown. Because to understand a LLVM compiler and IDE, it is necessary to refer to the user's manual of each tool product.

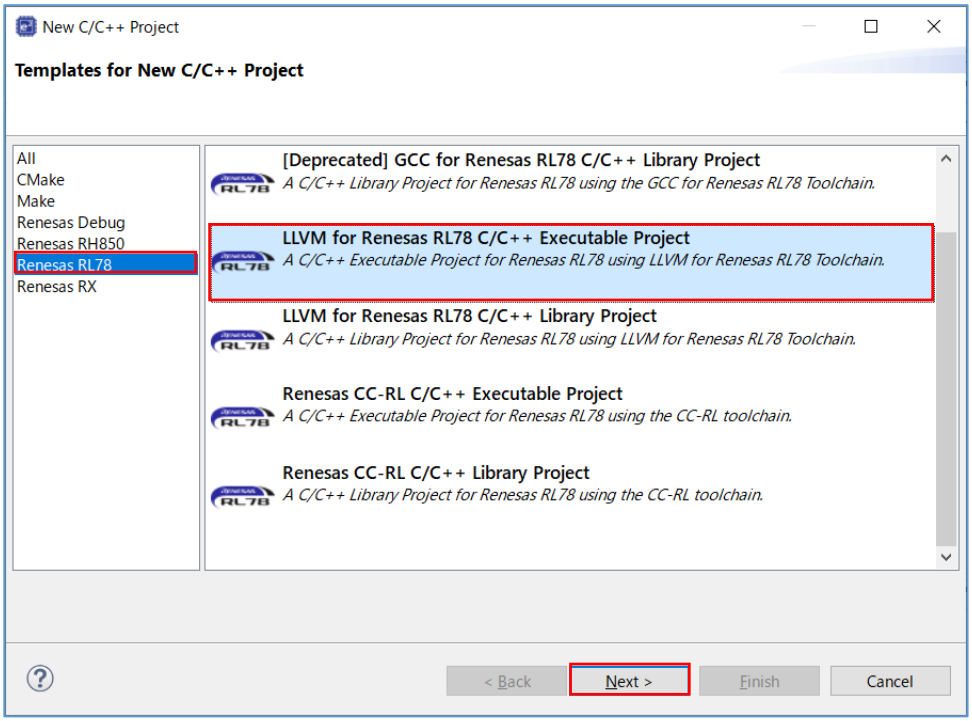
6.3.1 Example of Creating a Sample Project

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

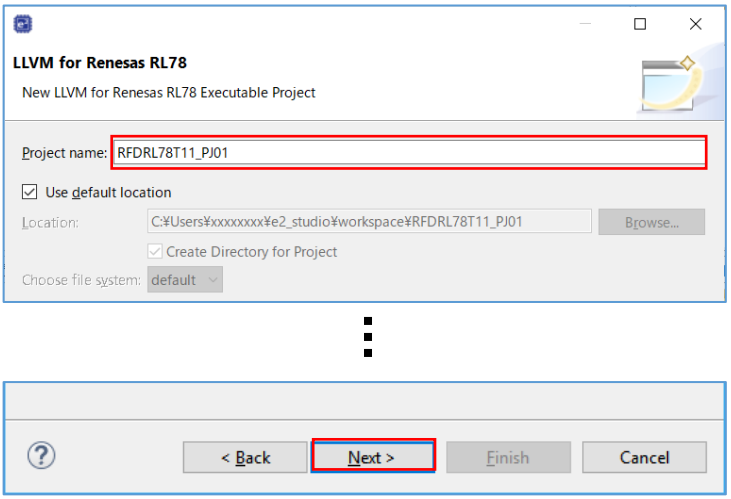
- The e² studio starts and from the [File] menu, select [New] – [C/C++ Project], the “Templates for New C/C++ Project” window will open.



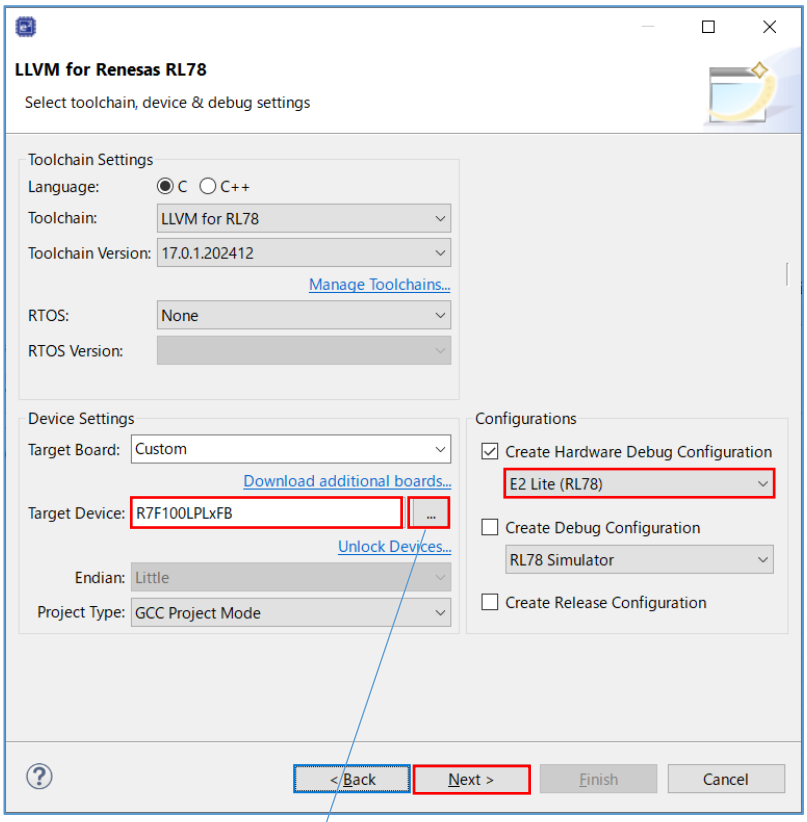
- Select [LLVM for Renesas RL78 C/C++ Executable Project] displayed after selection in [Renesas RL78], and press “Next” button.

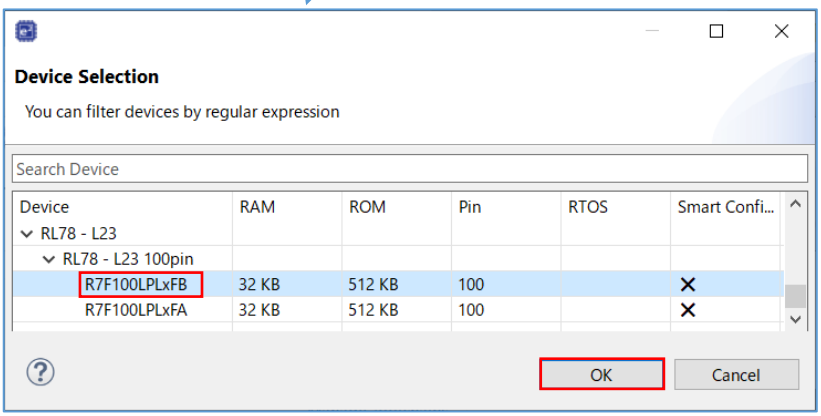


- Input “Project name” on “New LLVM for Renesas RL78 Executable Project” window, and press “Next” button. [Project name] is temporarily set to “RFDRL78T11_PJ01”.

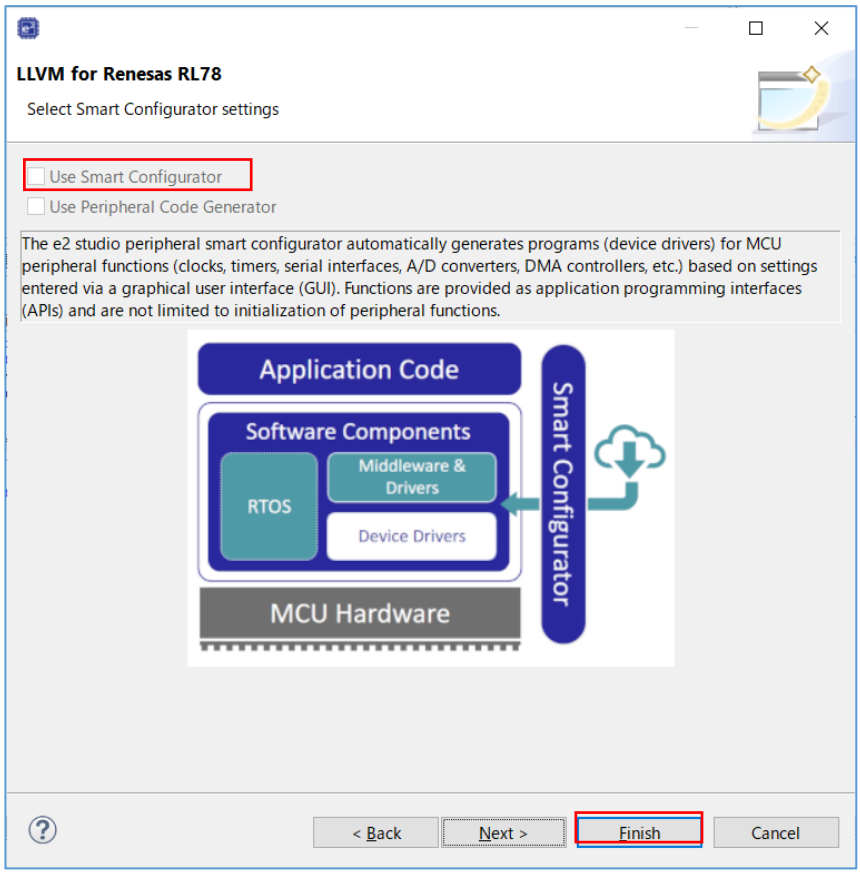


- Select the [Target Device] of [Device Settings], and select “RL78 – L23” – “RL78 – L23 100pin” – “R7F100LPLxFB”.
- It is a premise that E2 Lite is selected as a debugging tool and on-chip debugging is executed. Put a check mark to “Create Hardware Debug Configuration” by [Configurations]. And select “E2 Lite(RL78)”.
- Press “Next” button.



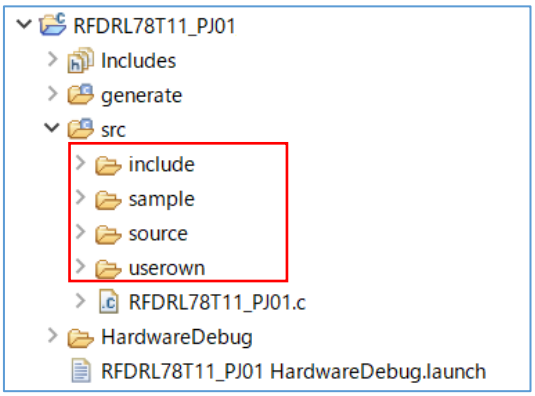


- Uncheck the “Use Smart Configurator”.
- Press [Finish] button.



6.3.2 Example of Registration of Target Folders and Target Files

Using RFD RL78 Type 11, when programming each area [(1) code flash memory, (2) data flash memory, (3) extra area], and (4) bank programming / bank swapping control, the example which registers necessary files is shown. Each folder of the RFD RL78 Type 11 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 RFD (e^2 studio)

Note : Register the “generate” folder output by e^2 studio as necessary.

- Registration of the latest I/O header file outputted to target products by e^2 studio
  “iodefine.h” and “iodefine_ext.h” are an I/O header file which e^2 studio outputs to target products. Replacing instead of “iodefine.h” and “iodefine_ext.h” included in RFD RL78 Type 11 is recommended. Registration of target folders and target files are implemented. Then, a user replaces “iodefine.h” and “iodefine_ext.h” which e^2 studio outputted with “iodefine.h” and “iodefine_ext.h” included in RFD RL78 Type 11.

- Registration of the vector table file outputted to target products by e^2 studio
  “interrupt_handlers.h”, “inthandler.c” and “vects.c” are files that contain vector tables that e^2 studio outputs for the target product. Since it depends on the product, please replace “interrupt_handlers.h”, “inthandler.c”, and “vects.c” included in RFD RL78 Type 11.
  When these are replaced, change the option byte values in the “vects.c” file. Refer to “6.3.4 Option Bytes 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 e^2 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]\src\sample\RL78_L23\CF\LLVM\include”
- The case of data flash programming : “\[Project name]\src\sample\RL78_L23\DF\LLVM\include”
- The case of extra area (FSW) programming :
  “\[Project name]\src\sample\RL78_L23\EX_FSW\LLVM\include”
- The case of bank programming / bank swapping control :
  “\[Project name]\sample\RL78_L23\BP_SWAP\LLVM\include”

The folder with which a user replaces the “inthandler.c” and “vects.c” files:

- The case of code flash programming : “\[Project name]\src\sample\RL78_L23\CF\LLVM\source”
- The case of data flash programming : “\[Project name]\src\sample\RL78_L23\DF\LLVM\source”
- The case of extra area (FSW) programming :
  “\[Project name]\src\sample\RL78_L23\EX_FSW\LLVM\source”
- The case of bank programming / bank swapping control :
  “\[Project name]\src\sample\RL78_L23\BP_SWAP\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 files as these exists also in the “sample” folder of RFD RL78 Type 11. Therefore, using the function of e^2 studio, Select those files from tree and excludes from a project.

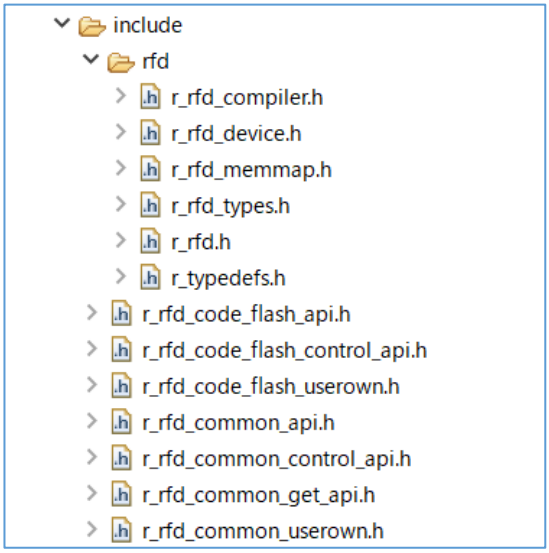
- e^2 studio : Clicks the right mouse button for the file of tree. And On the [Settings] screen displayed by the “property”, put a check mark to [Exclude resource from build] and exclude a target file (target folder). (Exclusion of a folder is also possible)

“hwinit.c”, “linker_script.ld”, and “start.S” in the [project name]/generate folder, and [project name].c (in this case “RFDRL78T11_PJ01.c”) in the [project name]/src folder are not used in RFD RL78 Type 11. Therefore, exclude those from the project.

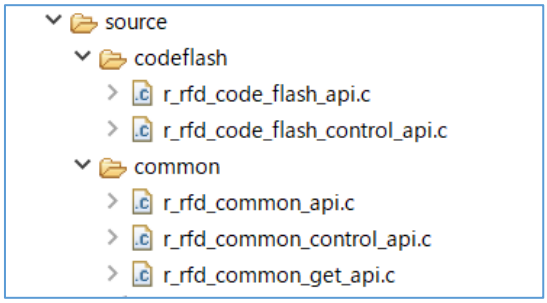
(1) Registration of the folders and files of the target in the case of reprogramming code flash memory

The folders (“include”, “source”, “userown”, “sample”) and source program file which are included in RFD RL78 Type 11 to register are shown below.

in the “include” folder



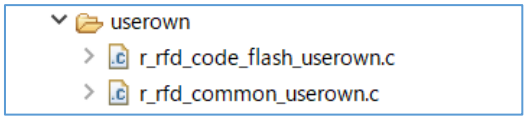
in the “source” folder



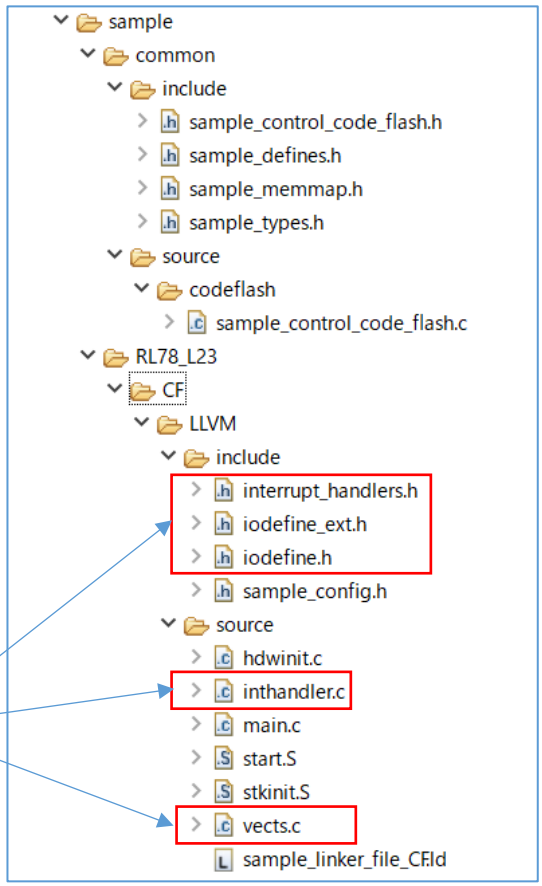
Transpose to “iodefine.h”, “iodefine_ext.h”, “interrupt_handlers.h”, “inthandler.c” and “vects.c” outputted by e^2 studio.

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

in the “userown” folder



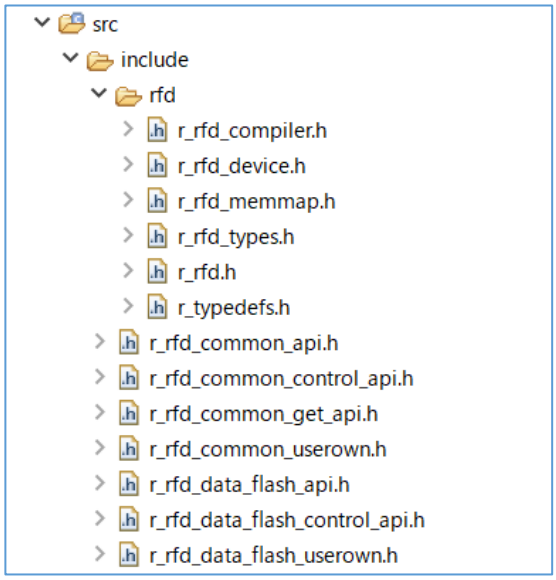
in the “sample” folder



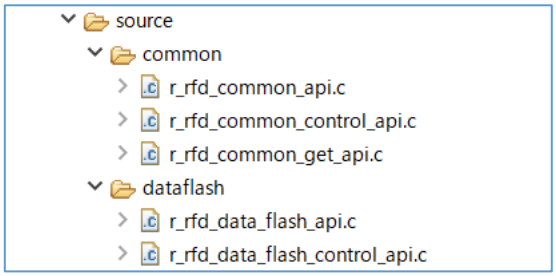
(2) Registration of the folders and files of the target in the case of reprogramming data flash memory

The folders (“include”, “source”, “userown”, “sample”) and source program file which are included in RFD RL78 Type 11 to register are shown below.

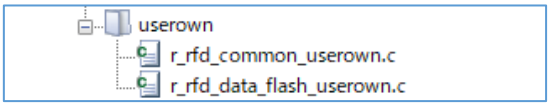
in the “include” folder



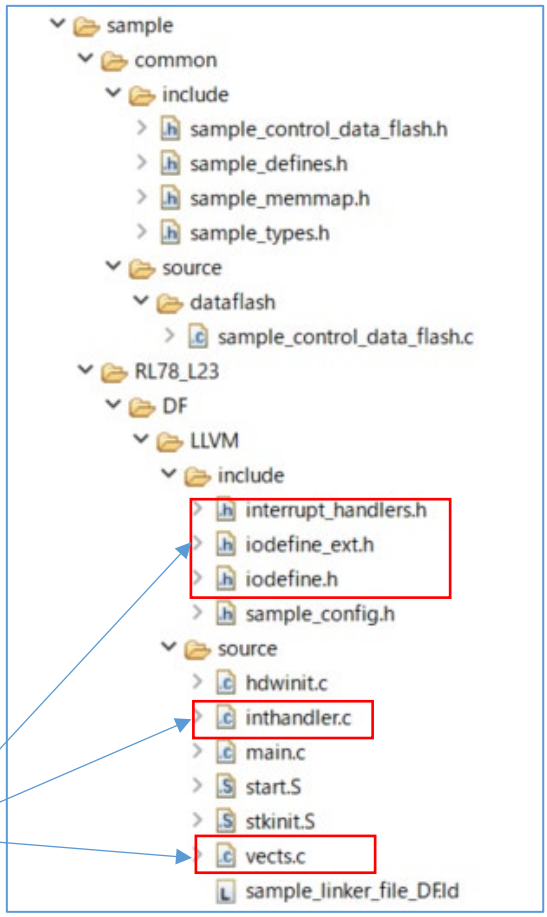
in the “source” folder



in the “userown” folder



in the “sample” folder



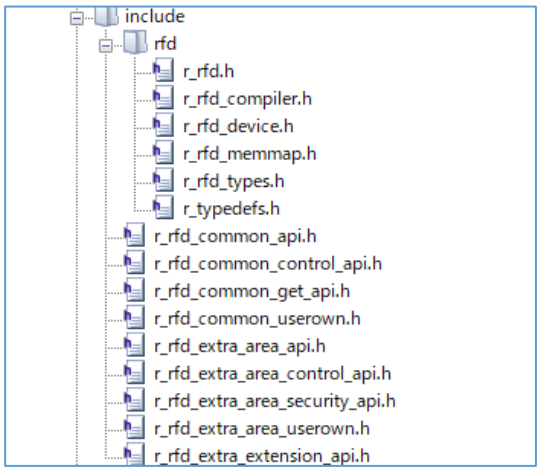
Transpose to “iodefine.h”, “iodefine_ext.h”, “interrupt_handlers.h”, “inthandler.c” and “vects.c” outputted by e^2 studio.

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

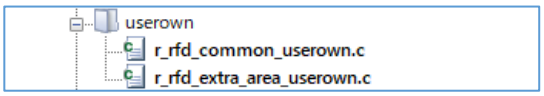
(3) Registration of the folders and files of the target in the case of reprogramming extra area (FSW setting)

The folders (“include”, “source”, “userown”, “sample”) and source program file which are included in RFD RL78 Type 11 to register are shown below.

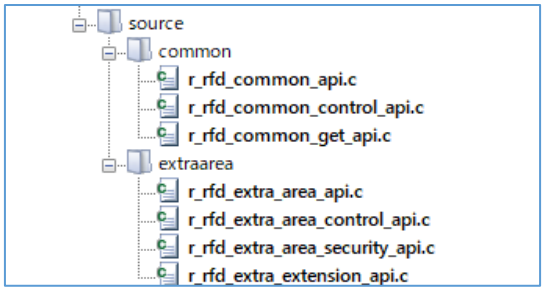
in the “include” folder



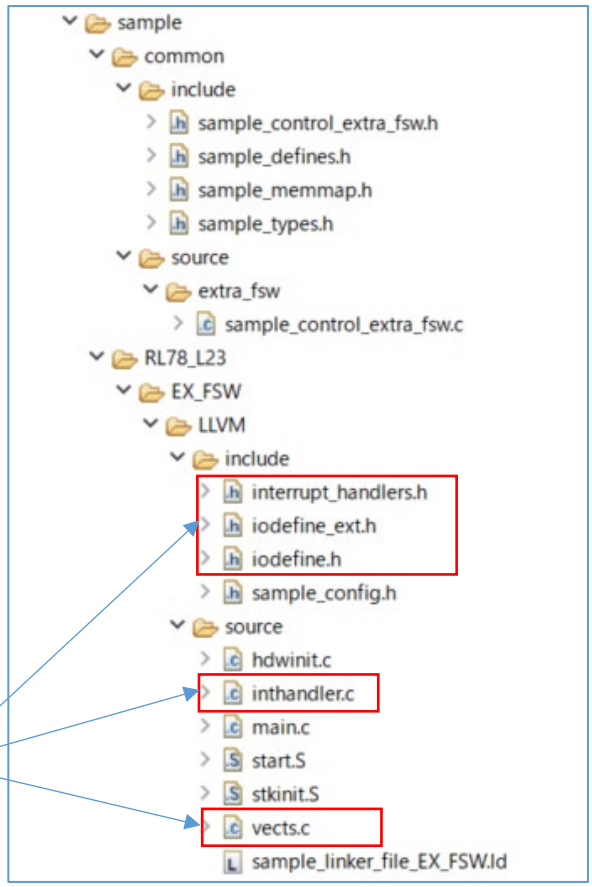
in the “userown” folder



in the “source” folder



in the “sample” folder



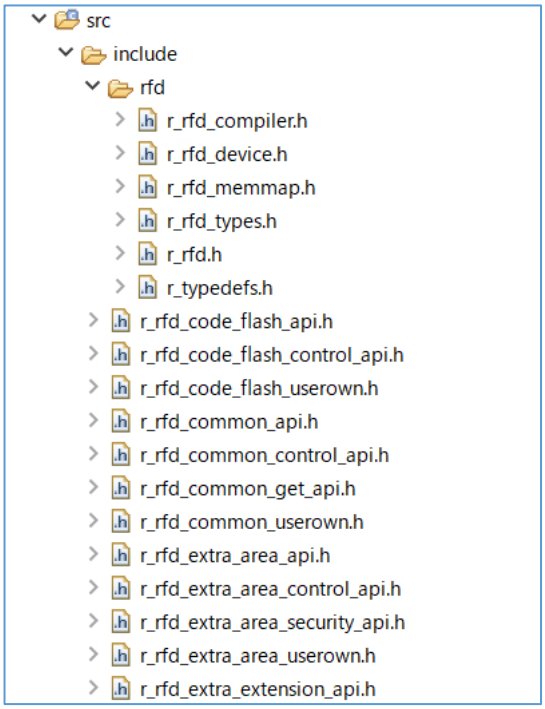
Transpose to “iodefine.h”, “iodefine_ext.h”, “interrupt_handlers.h”, “inthandler.c” and “vects.c” outputted by e^2 studio.

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

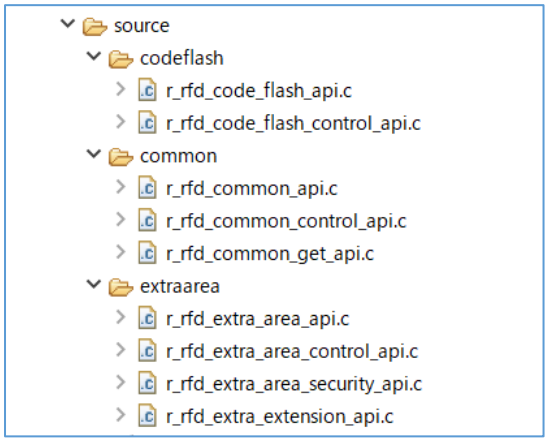
(4) Registration of the folders and files of the target in the case of bank programming / bank swapping control cexecution

The folders (“include”, “source”, “userown”, “sample”) and source program file which are included in RFD RL78 Type 11 to register are shown below.

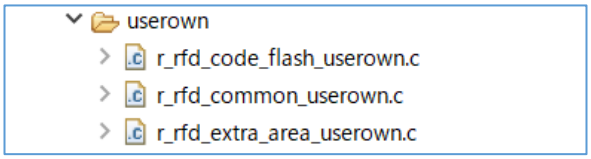
in the “include” folder



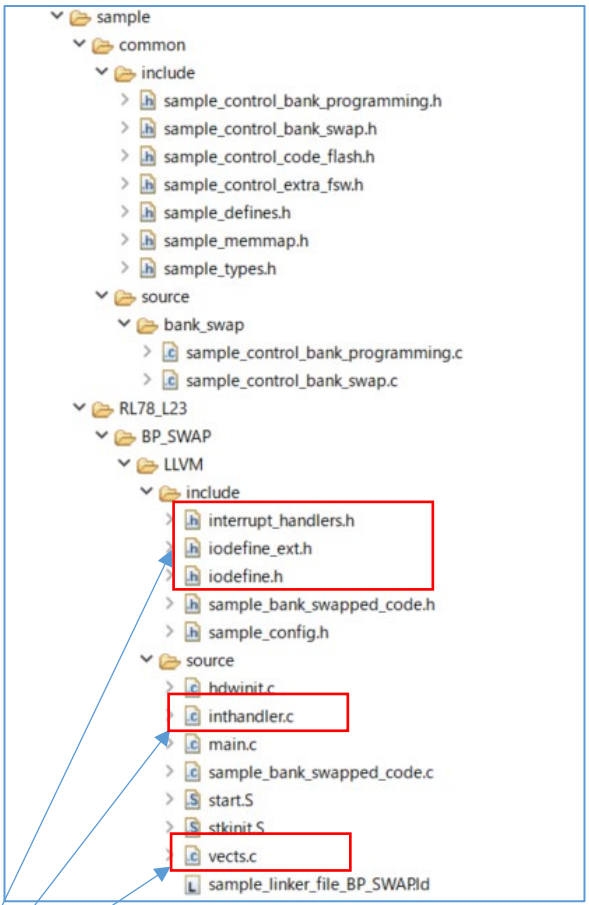
in the “source” folder



in the “userown” 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 e^2 studio setting necessary in order to build RFD RL78 Type 11 using a LLVM compiler.
Click the right mouse button for the project(“RFDRL78T11_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” [Settings] – “Compiler” [Includes].

(1) Code flash memory reprogramming

```
${ProjDirPath}\src\include\rfd
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_L23\CF\LLVM\include
${ProjDirPath}\src\sample\common\include
```

(2) Data flash memory reprogramming

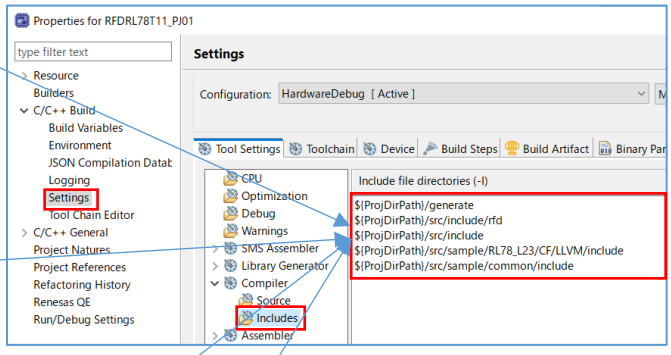
```
${ProjDirPath}\src\include\rfd
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_L23\DF\LLVM\include
${ProjDirPath}\src\sample\common\include
```

(3) Extra area(FSW) reprogramming

```
${ProjDirPath}\src\include\rfd
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_L23\EX_FSW\LLVM\include
${ProjDirPath}\src\sample\common\include
```

(4) Bank Programming / Bank Swapping Control

```
${ProjDirPath}\src\include\rfd
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_L23\BP_SWAP\LLVM\include
${ProjDirPath}\src\sample\common\include
```



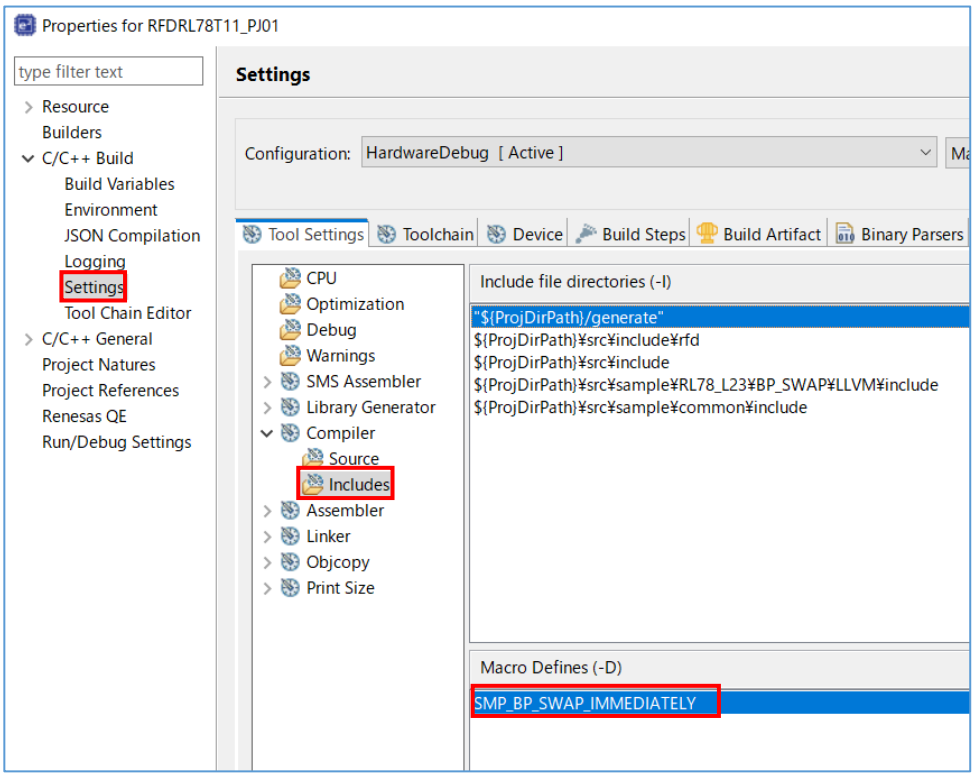
6.3.3.2 The Setting of User Definition Macro (LLVM Compiler)

In the case which selects “active bank swapping execution” (immediate execution bank swapping) with the sample program function of bank swapping control, user definition macro (“SMP_BP_SWAP_IMMEDIATELY”) is necessary. In the case which selects “bank swapping execution after reset”, this definition is unnecessary.

- On e^2 studio, the macro for selecting “active bank swapping execution (Immediate execution bank swapping)” is defined in “Properties” window.
  - Define the following macro in the “Macro Defines (-D)” displayed by selection of “C/C++ Build” [Settings] - “Compiler” [Includes]. Definition macro differs by each device to be used.

    Macro for selecting “active bank swapping execution” (Immediate execution bank swapping):

    SMP_BP_SWAP_IMMEDIATELY

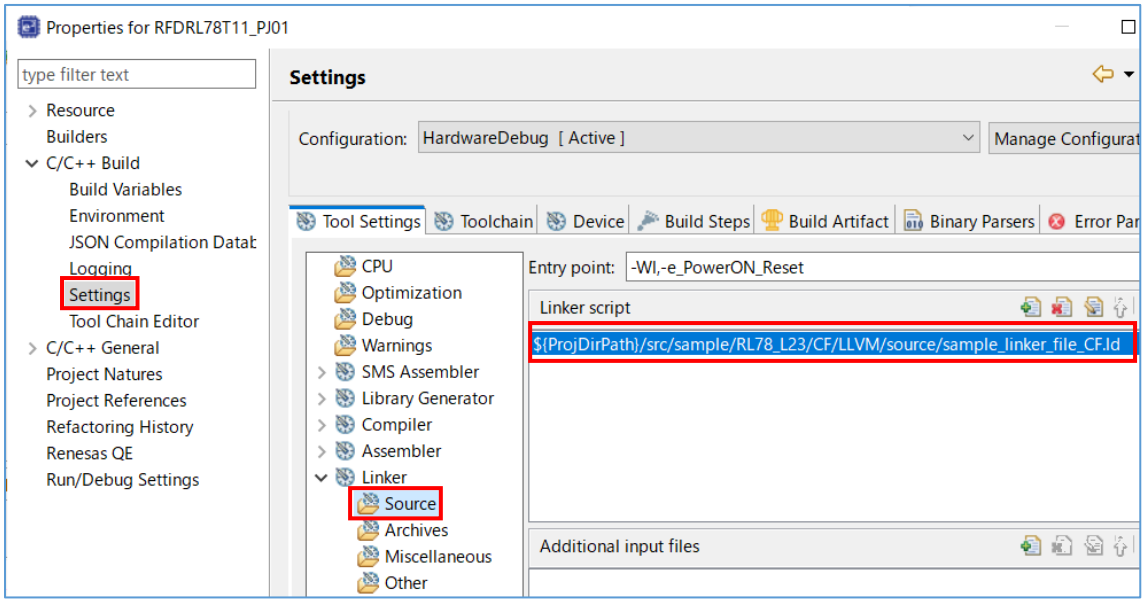


6.3.3.3 Linker Script File (.ld) Settings

On LLVM, linker script file (*.ld) describes link setting executed by building. Click the right mouse button for the project(“RFDRL78T11_PJ01”) in a tree, and select “Property”. And set each setting of the build tool in the displayed window. Input the Include linker script file path in the window displayed by selection of “C/C++ Build” [Settings] – “linker” [Source].
Input the path to the “sample_linker_file_(Area name).ld” file contained in the RFD RL78 Type 11 sample program.

The linker script file (*.ld) for RFD RL78 Type 11 is as follows:

- Code flash memory reprogramming: sample_linker_file_CF.ld (\sample\RL78_L23**CF**\LLVM\source\)
- Data flash memory reprogramming: sample_linker_file_DF.ld (\sample\RL78_L23**DF**\LLVM\source\)
- Extra area(FSW) reprogramming:
  sample_linker_file_EX_FSW.ld (\sample\RL78_L23**EX_FSW**\LLVM\source\)
- Bank programming / bank swapping control:
  sample_linker_file_BP_SWAP.ld (\sample\RL78_L23**BP_SWAP**\LLVM\source\)



Note : Refer to each reference manual of LLVM about the descriptive content of linker script file, and the details of the description method.

6.3.3.4 Section Settings

The setting outline of the section item described to linker script file (*.ld) of RFD RL78 Type 11.

(1) The sections for code flash memory reprogramming
- Section of code to be placed in the ROM area (RFD_ROM_CODE) : RFD_CMN
- Section of code that is copied from the ROM area to the RAM area (RFD_RAM_CODE) : RFD_CF, SMP_CF
- Section of data that is copied from the ROM area to the RAM area : RFD_DATA

(2) The sections for data flash memory reprogramming
- Section of code to be placed in the ROM area (RFD_ROM_CODE) : RFD_CMN, RFD_DF, SMP_DF
- Section of data that is copied from the ROM area to the RAM area : RFD_DATA

(3) The sections for extra area(FSW) reprogramming
- Section of code to be placed in the ROM area (RFD_ROM_CODE) : RFD_CMN
- Section of code that is copied from the ROM area to the RAM area (RFD_RAM_CODE) : RFD_EX, SMP_EX
- Section of data that is copied from the ROM area to the RAM area : RFD_DATA

(4) The sections for bank programming / bank swapping control
- Section of code to be placed in the ROM area (RFD_ROM_CODE) : RFD_CMN, RFD_CF, SMP_CF
  (RFD_ROM_BANK_CODE) : SMP_BPS

  *SMP_BPS is the code programmed by bank programming, and needs to be located to 0x6000.
- Section of code that is copied from the ROM area to the RAM area (RFD_RAM_CODE) : RFD_EX, SMP_EX
- Section of data that is copied from the ROM area to the RAM area : RFD_DATA

Note: When using the LLVM compiler, the compiler may automatically add subsections with different names when common processing is detected within the same section. Therefore, the following sections are added to the description in the sample_linker_file_XX.ld (“XX” = “CF”, “DF”, “EX_FSW” or “BP_SWAP”) file.

RFD_XXXX.* and SMP_XXXX.* (“XXXX” = “CF”, “DF”, “EX”, “BPS”, “DATA” or “CMN”)

Examples of subsections that could be added: RFD_CF.outlined-functions (etc.)

Refer to each reference manual of LLVM about the section setting method and the detail of functions for linker script file.

6.3.4 Option Bytes Settings

“Option Bytes” settings when using the LLVM compiler are set in the “vects.c” file.

Target file name: vects.c

- \[Project name]\src\sample\RL78_L23\[Area name]\LLVM\source\

Description of user option byte value:

In the “vects.c” file provided in the sample program, the option byte value and user option byte value are set in “Option_Bytes” as follows.

[The example for RL78/L23]
“0x6e, 0xff, 0xe8, 0x85” (WDT stop, LVD reset mode, HS mode /32MHz, Enable on-chip debug operation)

```
#include “interrupt_handlers.h”

extern void PowerON_Reset (void);

const unsigned char Option_Bytes[]    __attribute__ ((section (“.option_bytes”))) = {
        0x6e, 0xff, 0xe8, 0x85
};
```

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.

6.3.5 Setting of Connection with Target Board

This section describes the contents of connection setting on a target board necessary in order 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.

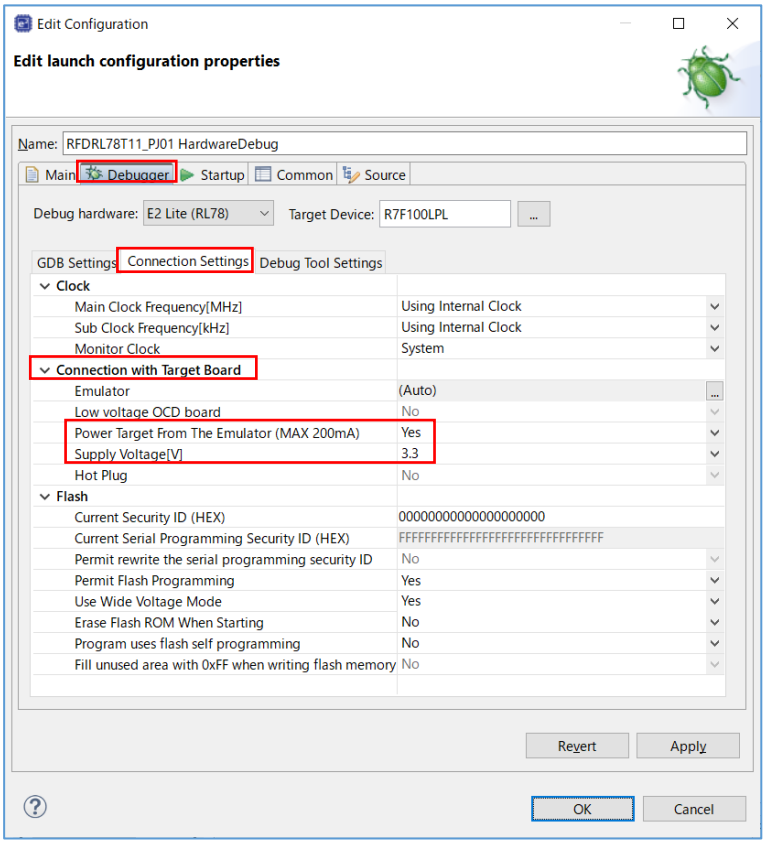
On e^2 studio, right-click a mouse in the target project of a tree. Selection of [Debug As] - [Debug Configurations…] will display the “Debug Configurations” screen. On the tree of a screen, select the target project (“RFDRL78T11_PJ01 HardwareDebug”) of [Renesas GDB Hardware Debugging]. And the displayed “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.

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

  - [Connection with Target Board] item

    In order 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)].



6.3.6 Caution

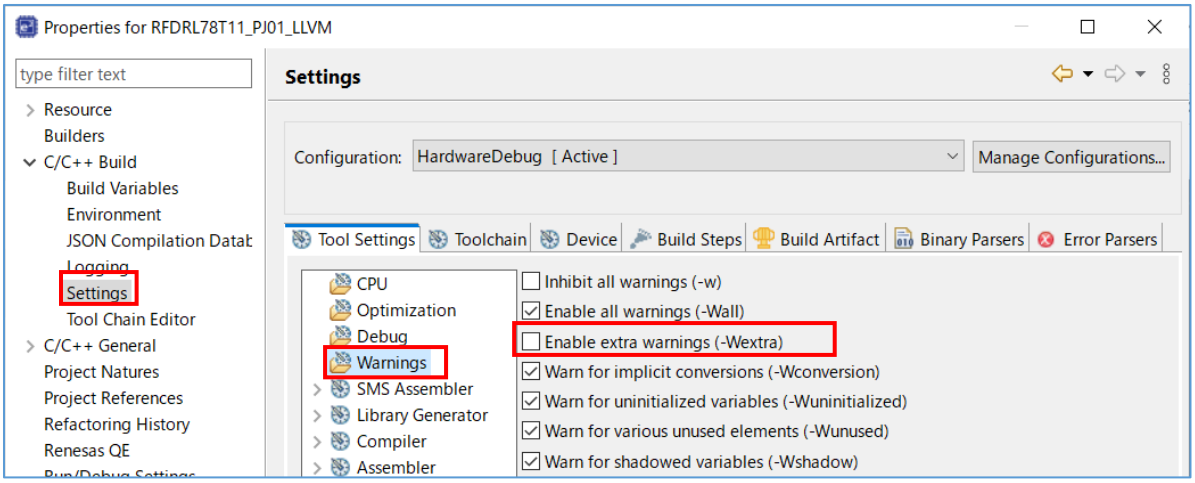
- About “warning” output when building is executed

The following “warning” may be output by “r_rfd_wait_count” or “r_rfd_xx_wait_count” function(xx_ = cf_ or df_ or extra_) when the build is executed. This means that the “i_u08_count” argument is not used in the function and is output.
The “r_rfd_wait_count” function or “r_rfd_xx_wait_count” functions are written in assembly language, and the argument “i_u08_count” is passed in the function as a general-purpose register.

Therefore, it is confirmed that there is no problem even if the variable name is not used.

The “warning” can be set to not be output by using the following property of e^2 studio, but it is recommended to set it after the development is completed because other warnings may not be output either.

- “C/C++ Build” [Settings] - “Warnings” and uncheck “Enable extra warnings (-Wextra)” as indicated.



6.4 Setting Related to Changing Devices

When using a device other than RL78/L23(R7F100LPL), the address settings in the section and some of the sample programs must be modified. This section describes the where to modify and procedure to modify.

To modify the setting values, refer to MCU List for RL78/L23 shown below and change the setting values according to the device you are using. An example of referencing the MCU List for RL78/L23 and an example of where to modify is shown below.

・MCU List for RL78/L23

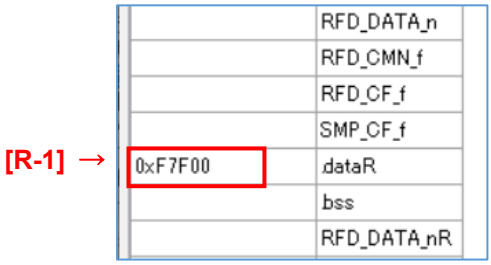
MCU Group	Code Flash memory		User RAM		Data Flash memory		Target MCU name
	Size (bytes)	Start/End Address	Size (bytes)	Start/End Address	Size (bytes)	Start/End Address	
RL78/L23	64K	0x00000 - 0x0FFFF	16K	0xFBF00 - 0xFFEFF	8K	0xF1000 - 0xF2FFF	R7F100LxE (x = F, G, J, L)
	128K	0x00000 - 0x1FFFF	16K	0xFBF00 - 0xFFEFF	8K	0xF1000 - 0xF2FFF	R7F100LxG (x = F, G, J, L, M, P)
	256K	0x00000 - 0x3FFFF	32K	0xF7F00 - 0xFFEFF	8K	0xF1000 - 0xF2FFF	R7F100LxJ (x = F, G, J, L, M, P)
	512K	0x00000 - 0x7FFFF	32K	0xF7F00 - 0xFFEFF	8K	0xF1000 - 0xF2FFF	R7F100LxL (x = F, G, J, L, M, P)

MCU Group	[R-1]	[R-2]	[R-3]	[R-4]	[R-5]	[R-6]	[R-7]	Target MCU name
	RAM Start Address	ROM End Address 1	ROM End Address 2	Data Flash End Address	OCD_ROM	Trace_RAM	END_BLOCK	
RL78/L23	0xFBF00	0x0FFFF	-	0xF2FFF	0xFE00	0xFC300	32	R7F100LxE (x = F, G, J, L)
	0xFBF00	0x0FFFF	0x1FFFF	0xF2FFF	0x1FE00	0xFC300	64	R7F100LxG (x = F, G, J, L, M, P)
	0xF7F00	0x0FFFF	0x3FFFF	0xF2FFF	0x3FE00	0xF8300	128	R7F100LxJ (x = F, G, J, L, M, P)
	0xF7F00	0x0FFFF	0x7FFFF	0xF2FFF	0x7FE00	0xF8300	256	R7F100LxL (x = F, G, J, L, M, P)

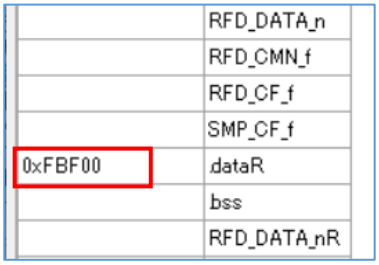
- Example of reference of the MCU List for RL78/L23

For example, when modifying the setting value indicated by [R-1] (the start address of RAM) as shown in the following figure. Here, refer to the setting value of the start address [R-1] (RAM Start Address) of RAM shown in the MCU List for RL78/L23 and set the value of RL78/L23 (R7F100LLG).

Example of where to modify the start address of RAM: RL78/L23(R7F100LPL RAM: 32 Kbytes).



Example of setting the start address value of RAM when using RL78/L23 (R7F100LLG RAM: 16 Kbytes).



The value to be set in [R-1] refers to the MCU List for RL78/L23 and sets the start address value of RAM of the target device.

In the column “Target MCU name” of the MCU List for RL78/L23, search for the row for R7F100LxG. Next, find the cell in the [R-1] column that intersects the row of R7F100LxG.

Since “0xFBF00” applies, the setting value of [R-1] is RL78/L23 (R7F100LxG) value “0xFBF00”.

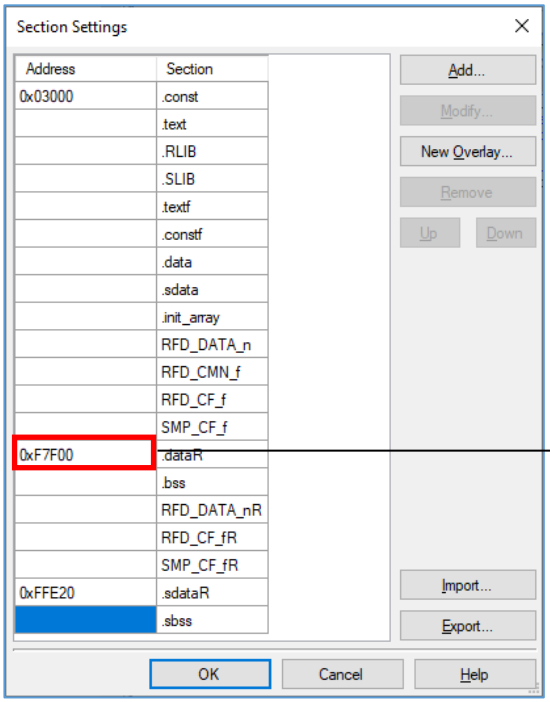
MCU Group	[R-1] RAM Start Address	[R-2] ROM End Address 1	[R-3] ROM End Address 2	[R-4] Data Flash End Address	[R-5] OCD_ROM	[R-6] Trace_RAM	[R-7] END_BLOCK	Target MCU name
RL78/L23	0xFBF00	0x0FFFF	-	0xF2FFF	0xFE00	0xFC300	32	R7F100LxE (x = F, G, J, L)
	0xFBF00	0x0FFFF	0x1FFFF	0xF2FFF	0x1FE00	0xFC300	64	R7F100LxG (x = F, G, J, L, M, P)
	0xF7F00	0x0FFFF	0x3FFFF	0xF2FFF	0x3FE00	0xF8300	128	R7F100LxJ (x = F, G, J, L, M, P)
	0xF7F00	0x0FFFF	0x7FFFF	0xF2FFF	0x7FE00	0xF8300	256	R7F100LxL (x = F, G, J, L, M, P)

- Example of where to modify

Points that need to be modified from the RL78/L23(R7F100LPL) settings are listed from “6.3.1“.
Points that need to be modified are indicated with “[R-x] →”. Refer to the MCU List for RL78/L23 to find the appropriate [R-x] setting for your device. Enter the searched value in [R-x]. (x = 1, 2, 3…)

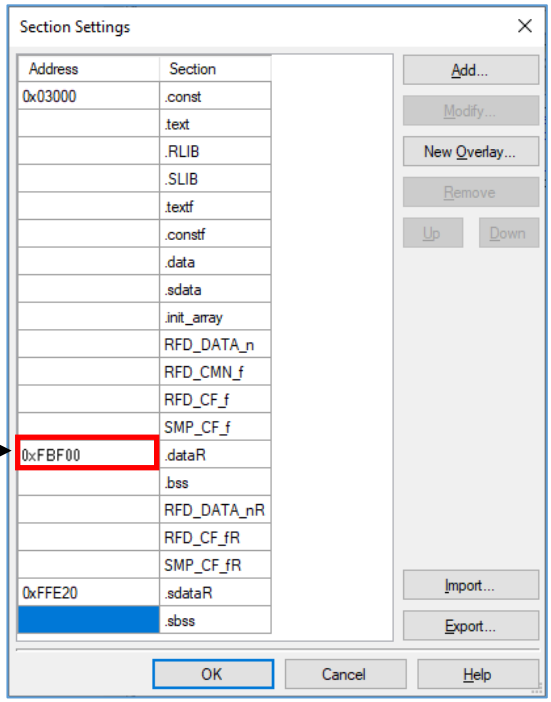
- Example of modification the section setting (start address of RAM) of code flash (CF) memory reprogramming (CS+: CC-RL compiler):

Setting for RL78/L23(RAM: 32 Kbytes)
Example: R7F100LPL



[R-1] →

Setting for RL78/L23(RAM: 16 Kbytes)
Example: R7F100LLG



6.4.1 CC-RL Compiler Environment Settings

Points of modifies and examples of modifies when using the CC-RL compiler environments (CS+ and e^2 studio) are described.

6.4.1.1 Section Settings

Modify the start address of the RAM area in the section settings.

This example shows the change from RL78/L23 (R7F100LPL) to RL78/L23 (R7F100LLG).

Since the RAM size is changed from 32 Kbytes to 16 Kbytes, modify the start address of the RAM from "0xF7F00" to "0xFBF00".

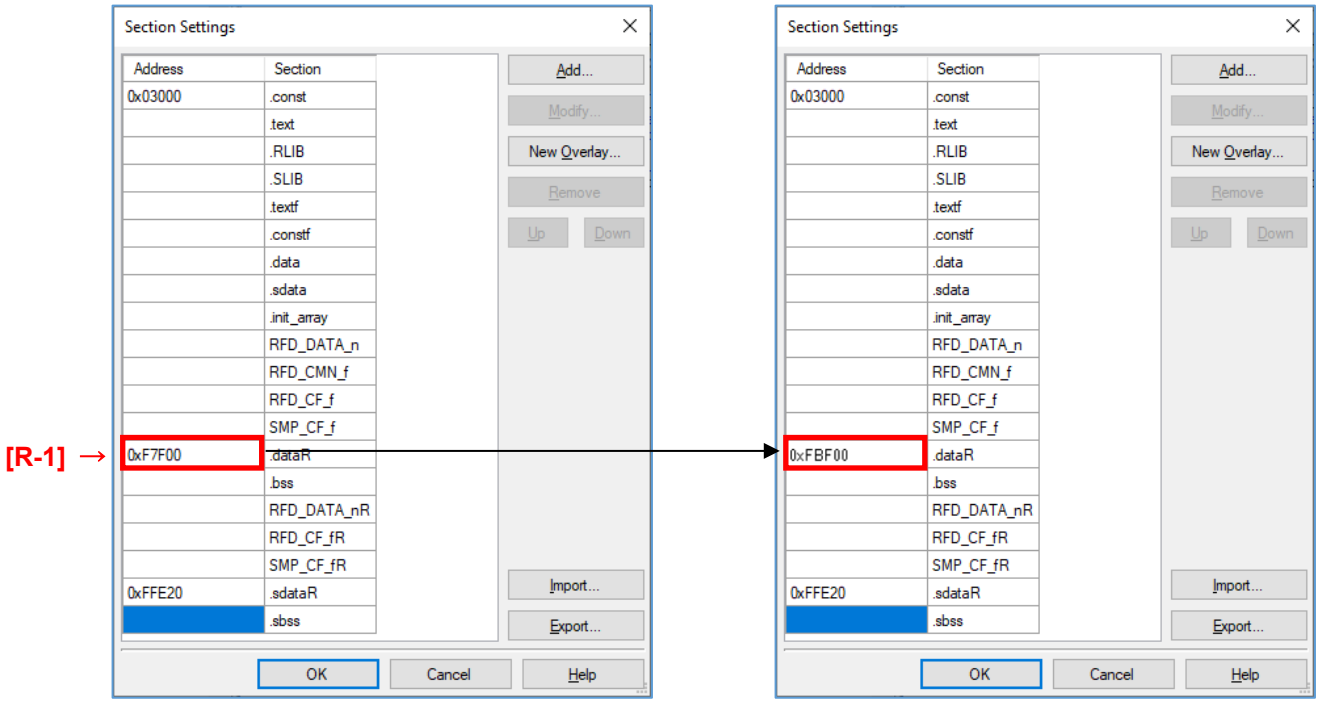
Note: For the start address of the RAM for each product, refer to "R-1" column in the MCU List for RL78/L23.

- Example of modifying section settings (start address of the RAM) in CS+:

The case of reprogramming the code flash memory.

Setting for RL78/L23(RAM: 32 Kbytes)
Example: R7F100LPL

Setting for RL78/L23(RAM: 16 Kbytes)
Example: R7F100LLG



Note: Other sample programs modify the top address of RAM into "0xFBF00" from "0xF7F00" similarly (reprogramming of a data flash area, reprogramming of an extra area, or bank programming / bank swapping control [Only the device in which the bank function was mounted can be used.]).

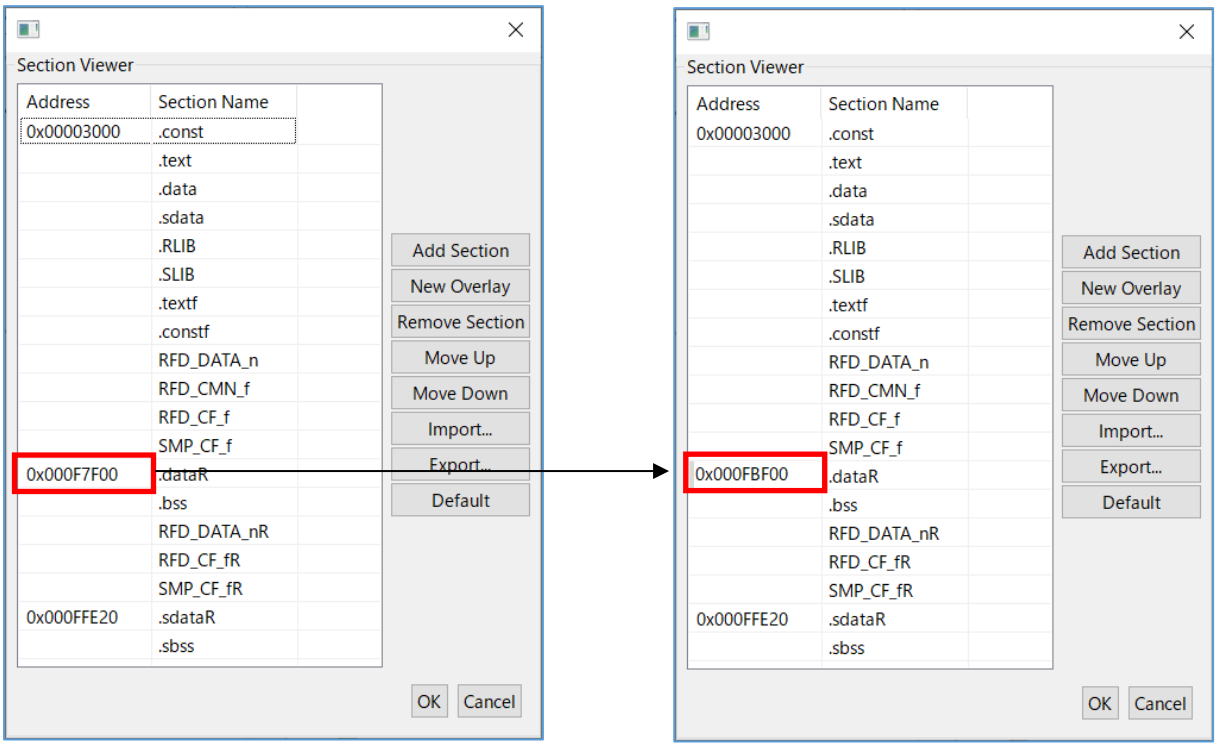
- Example of modifying section settings (start address of the RAM) in e^2 studio:

The case of reprogramming the code flash memory.

Setting for RL78/L23(RAM: 32 Kbytes)
Example: R7F100LPL

Setting for RL78/L23(RAM: 16 Kbytes)
Example: R7F100LLG

[R-1] →



Note: Other sample programs modify the top address of RAM into “0xFBF00” from “0xF7F00” similarly (reprogramming of a data flash area, reprogramming of an extra area, or bank programming / bank swapping control [Only the device in which the bank function was mounted can be used.]).

6.4.1.2 Debug Settings

When using the RL78/L23, the debug monitor area has a different range when using the debugger.

- The start of the “debug monitor area” address sets the address obtained by subtracting “511 bytes (0x1FF)” from the end address of the ROM area. If the end address is “0x7FFFF”, set it to “0x7FE00”.

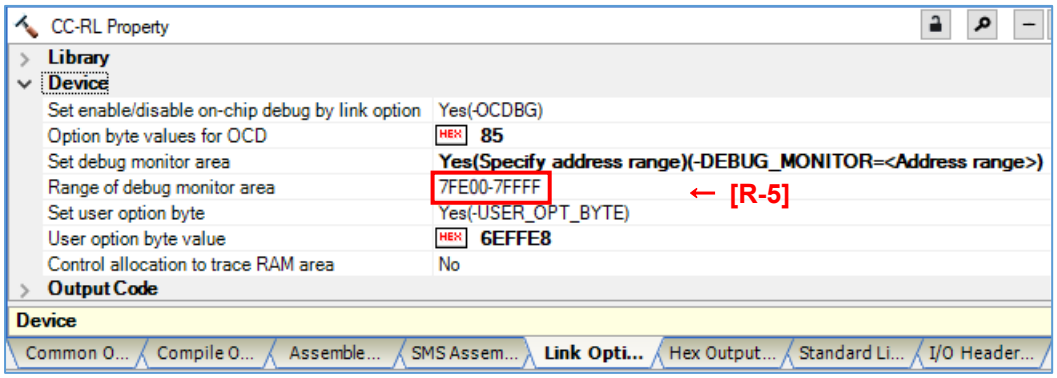
This example shows the modify from RL78/L23 (R7F100LPL) to RL78/L23 (R7F100LLG).

- Set the debug monitor area range to “0x1FE00 - 0x1FFFF” for the RL78/L23 (R7F100LLG).

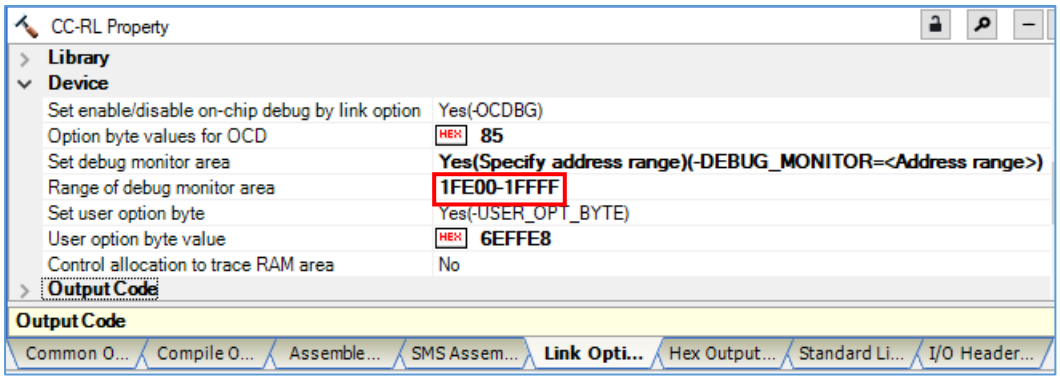
Note: For information on The start address of the “debug monitor area” for each product, refer to “[R-5]” column in the MCU List for RL78/L23.

- To set the debug monitor area in CS+, select the [Device] on the “Link Options” tab.

Setting for RL78/L23 (ROM: 512 Kbytes) Example: R7F100LPL

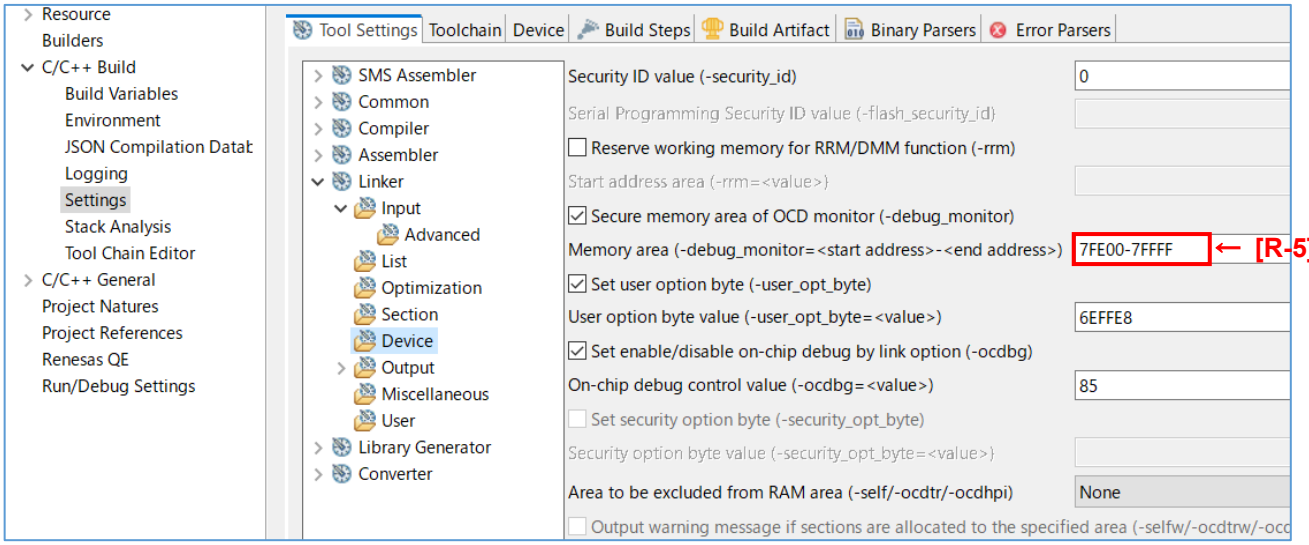


Setting for RL78/L23 (ROM: 128 Kbytes) Example: R7F100LLG

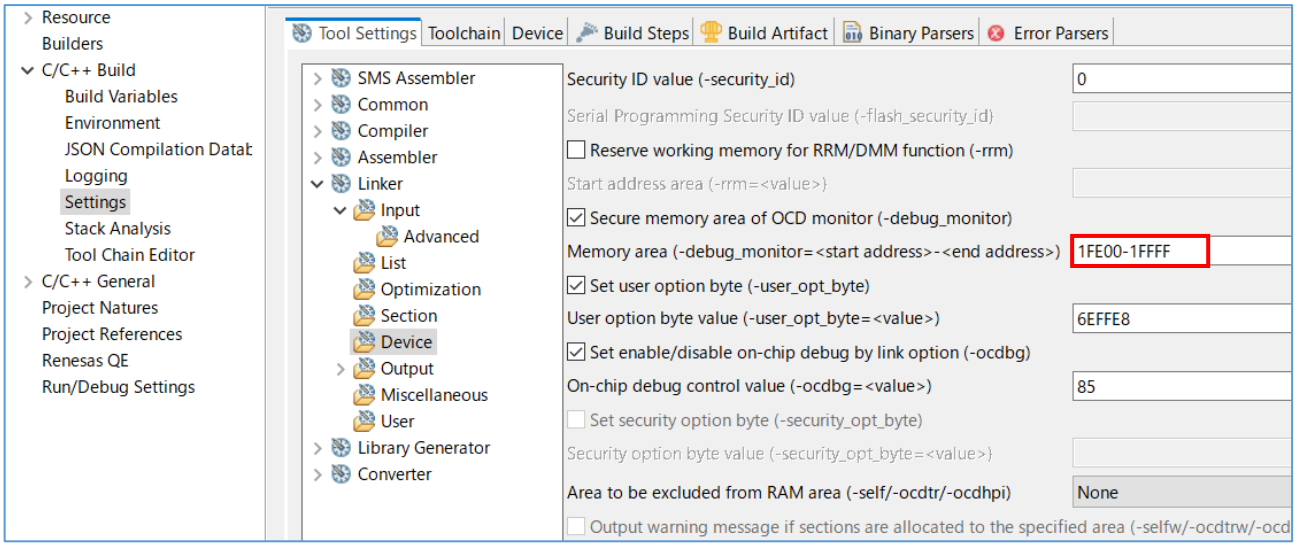


- To set the area of OCD monitor in e^2 studio, select the [Device] in the “Linker”.

Setting for RL78/L23 (ROM: 512 Kbytes) Example: R7F100LPL



Setting for RL78/L23 (ROM: 128 Kbytes) Example: R7F100LLG



6.4.2 IAR Compiler Environment Settings

Points of modifies and examples of modifies when using the IAR compiler environment (Embedded Workbench) is described.

6.4.2.1 Setting up Header files for Target Device

The “main.c” and “low_level_init.c” provided with RFD RL78 Type 11 includes the header files for the target device “RL78/L23: R7F100LPL”. When using other RL78/L23 products or RL78/L23 products, the included header file must be changed to the header file for the device used.
This section describes when RL78/L23 (R7F100LLG) is used.

Target files name: main.c, low_level_init.c

- For RL78/L23 (R7F100LPL):

< main.c >

```
#include “ior7f100lpl.h”
```

< low_level_init.c >

```
#include “ior7f100lpl.h”
#include “ior7f100lpl_ext.h”
```

- Example for RL78/L23 (R7F100LLG):

< main.c >

```
#include “ior7f100llg.h”
```

< low_level_init.c >

```
#include “ior7f100llg.h”
#include “ior7f100llg_ext.h”
```

Note: For the device type name of the product, refer to “Target MCU name” column in the MCU List for RL78/L23.

6.4.2.2 Linker Configuration File Settings

In the sample program provided by RFD RL78 Type 11, The sections (ROM, RAM, and Data flash range) for RL78/L23 (R7F100LPL) are set.
When using other RL78/L23 products, modify the contents of the linker configuration file “sample_linker_file_xx.icf (xx = CF, DF, EX_FSW or BP_SWAP)” provided for the RL78/L23 of RFD RL78 Type 11 because the section settings are different. The modifications are shown in red text below, so refer to the MCU List for RL78/L23 and change the setting values for the target device.

Target files name: sample_linker_file_xx.icf (xx = CF, DF, EX_FSW or BP_SWAP)

This example shows the modify from RL78/L23 (R7F100LPL) to RL78/L23 (R7F100LLG).

- Modify the ROM area to the range of 128 Kbytes [0x00000 - 0x1FFFF]
- Modify the start address to “0xFBF00” because the RAM area is 16 Kbytes [0x0FBF00 - 0x0FFEFF]
- Modify the end address to “0xF2FFF” because the data flash area is 8 Kbytes [0x0F1000 - 0x0F2FFF]

(1) Section Settings

For sample_linker_file_CF.icf or sample_linker_file_EX_FSW.icf or sample_linker_file_BP_SWAP.icf

Setting for RL78/L23 (ROM: 512 Kbytes, RAM: 32 Kbytes, Data Flash: 8 Kbytes) Example: R7F100LPL

```
define region ROM_near = mem:[from 0x000D8 to 0x0FFFF];   ← [R-2]
define region ROM_far = mem:[from 0x000D8 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF]
        | mem:[from 0x20000 to 0x2FFFF] | mem:[from 0x30000 to 0x3FFFF] | mem:[from 0x40000 to 0x4FFFF]
        | mem:[from 0x50000 to 0x5FFFF] | mem:[from 0x60000 to 0x6FFFF] | mem:[from 0x70000 to 0x7FFFF];
define region ROM_huge = mem:[from 0x000D8 to 0x7FFFF];   ← [R-2] or [R-3] Notes 2
                                                          ↑ [R-2], [R-3] Notes 1
define region SADDR = mem:[from 0xFFE20 to 0xFFEDF];
define region RAM_near = mem:[from 0xF7F00 to 0xFFE1F];   ← [R-1]
define region RAM_far = mem:[from 0xF7F00 to 0xFFE1F];    ← [R-1]
define region RAM_code = mem:[from 0xF7F00 to 0xFFE1F];   ← [R-1]
define region RAM_huge = mem:[from 0xF7F00 to 0xFFE1F];   ← [R-1]
define region VECTOR = mem:[from 0x00000 to 0x0007F];
define region CALLT = mem:[from 0x00080 to 0x000BF];
define region EEPROM = mem:[from 0xF1000 to 0xF2FFF];     ← [R-4]
```

Notes 1 When the ROM size is larger than 64 Kbytes, the description must change as the ROM size increases. For details of the description.

2 Sets the value [R-3] when there is an address value in [R-3] on the MCU List for RL78/L23. In the case of “-”, set the value of [R-2].

Setting for RL78/L23 (ROM: 128 Kbytes, RAM: 16 Kbytes, Data Flash: 8 Kbytes) Example: R7F100LLG

```
define region ROM_near = mem:[from 0x000D8 to 0x0FFFF];
define region ROM_far = mem:[from 0x000D8 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF ];
define region ROM_huge = mem:[from 0x000D8 to 0x1FFFF ];
define region SADDR = mem:[from 0xFFE20 to 0xFFEDF];
define region RAM_near = mem:[from 0xFBF00 to 0xFFE1F];
define region RAM_far = mem:[from 0xFBF00 to 0xFFE1F];
define region RAM_code = mem:[from 0xFBF00 to 0xFFE1F];
define region RAM_huge = mem:[from 0xFBF00 to 0xFFE1F];
define region VECTOR = mem:[from 0x00000 to 0x0007F];
define region CALLT = mem:[from 0x00080 to 0x000BF];
define region EEPROM = mem:[from 0xF1000 to 0xF2FFF ];
```

The case of sample_linker_file_DF.icf

Setting for RL78/L23 (ROM: 512 Kbytes, RAM: 32 Kbytes, Data Flash: 8 Kbytes) Example: R7F100LPL

```
define region ROM_near = mem:[from 0x000D8 to 0x0FFFF];   ← [R-2]
define region ROM_far = mem:[from 0x000D8 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF]
        | mem:[from 0x20000 to 0x2FFFF] | mem:[from 0x30000 to 0x3FFFF] | mem:[from 0x40000 to 0x4FFFF]
        | mem:[from 0x50000 to 0x5FFFF] | mem:[from 0x60000 to 0x6FFFF] | mem:[from 0x70000 to 0x7FFFF];
define region ROM_huge = mem:[from 0x000D8 to 0x7FFFF];   ← [R-2] or [R-3] Notes 2   ↑ [R-2], [R-3] Notes 1
define region SADDR = mem:[from 0xFFE20 to 0xFFEDF];
define region RAM_near = mem:[from 0xF7F00 to 0xFFE1F];   ← [R-1]
define region RAM_far = mem:[from 0xF7F00 to 0xFFE1F];    ← [R-1]
define region RAM_huge = mem:[from 0xF7F00 to 0xFFE1F];   ← [R-1]
define region VECTOR = mem:[from 0x00000 to 0x0007F];
define region CALLT = mem:[from 0x00080 to 0x000BF];
define region EEPROM = mem:[from 0xF1000 to 0xF2FFF];     ← [R-4]
```

Notes 1 When the ROM size is larger than 64 Kbytes, the description must change as the ROM size increases. For details of the description.

2 Sets the value [R-3] when there is an address value in [R-3]on the MCU List for RL78/L23. In the case of “-”, set the value of [R-2].

Setting for RL78/L23 (ROM: 128 KB, RAM: 16 KB, Data Flash: 8KB) Example: R7F100LLG

```
define region ROM_near = mem:[from 0x000D8 to 0x0FFFF];
define region ROM_far = mem:[from 0x000D8 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF ];
define region ROM_huge = mem:[from 0x000D8 to 0x1FFFF ];
define region SADDR = mem:[from 0xFFE20 to 0xFFEDF];
define region RAM_near = mem:[from 0xFBF00 to 0xFFE1F];
define region RAM_far = mem:[from 0xFBF00 to 0xFFE1F];
define region RAM_huge = mem:[from 0xFBF00 to 0xFFE1F];
define region VECTOR = mem:[from 0x00000 to 0x0007F];
define region CALLT = mem:[from 0x00080 to 0x000BF];
define region EEPROM = mem:[from 0xF1000 to 0xF2FFF ];
```

- Examples of ROM_far

The following is an example of entries in ROM_far for each ROM size. Refer to the row with the same ROM size as the target device. Colored areas indicate values for [R-2] or [R-3].

When ROM size is 64 KB or less ([R-3] is “-”).

ROM	[R-2] Value	mem:[from 0x000D8 to [R-2]];
64KB	0x0FFFF	mem:[from 0x000D8 to 0x0FFFF];

When ROM size exceeds 64KB ([R-3] is not “-”).

ROM	[R-3] Value	mem:[from 0x000D8 to [R-2]] | mem:[from 0x10000 to 0x1FFFF] | …Omitted… | mem:[from 0xX0000 to [R-3]];
128KB	0x1FFFF	mem:[from 0x000D8 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF];
256KB	0x3FFFF	mem:[from 0x000D8 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF] | mem:[from 0x20000 to 0x2FFFF] | mem:[from 0x30000 to 0x3FFFF];
512KB	0x7FFFF	mem:[from 0x000D8 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF] | mem:[from 0x20000 to 0x2FFFF] | mem:[from 0x30000 to 0x3FFFF] | mem:[from 0x40000 to 0x4FFFF] | mem:[from 0x50000 to 0x5FFFF] | mem:[from 0x60000 to 0x6FFFF] | mem:[from 0x70000 to 0x7FFFF];

(2) Debug Settings

- The start address of the debug monitor area is set by subtracting “511 bytes(0x1FF)” from the end address of the ROM area. If the end address is “0x7FFFF”, set “0x7FE00”.

- The start address of the TraceRAM area is set by adding “1 Kbyte(0x400))” to the start address of the RAM area. If the start address is “0xF7F00”, set “0xF8300”.

- When debugging self-programming with an on-chip debugger, 128 bytes of area is used from the start address of RAM. Therefore, it is necessary to set the start address of a RAM area, and the address adding “127 bytes (0x7F).” If the start address is “0xF7F00”, set “0xF7F00” and “0xF7F7F.”

As an example, modifying from RL78/L23 (R7F100LPL) to RL78/L23 (R7F100LLG) is shown.

- Set the debug monitor area range to [from 0x1FE00 size 0x0200].
- Set the TraceRAM area range to [from 0xFC300 size 0x0400].
- Set the area range required to debug the self-programming to [from 0xFBF00 to 0xFBF7F].

Modifies to the TraceRAM area, the debug monitor area when using the debugger, and the area range required to debug the self-programming.

Setting for RL78/L23 (ROM: 512 Kbytes, RAM: 32 Kbytes) Example: R7F100LPL

```
if (isdefinedsymbol(__RESERVE_OCD_ROM))
{
        if (__RESERVE_OCD_ROM == 1)
        {
        reserve region “OCD ROM area” = mem:[from 0x7FE00 size 0x0200];   ← [R-5]
        }
}
      |
[Omitted]
      |
if (isdefinedsymbol(__RESERVE_OCD_TRACE_RAM))
{
        if (__RESERVE_OCD_TRACE_RAM == 1)
        {
        reserve region “OCD Trace RAM” = mem:[from 0xF8300 size 0x0400];   ← [R-6]
        }
}
      |
[Omitted]
      |
if (isdefinedsymbol(__RESERVE_FLASH_SELF_PROGRAMMING_RAM))
{
   if (__RESERVE_FLASH_SELF_PROGRAMMING_RAM == 1)
   {
      reserve region “RESERVED_FLASH_SELF_PROGRAMMING_RAM” = mem:[from 0xF7F00 to 0xF7F7F];
   }                                                                    ↑ [R-1]
}
```

Setting for RL78/L23 (ROM: 128 Kbytes, RAM: 16 Kbytes) Example: R7F100LLG

```
if (isdefinedsymbol(__RESERVE_OCD_ROM))
{
        if (__RESERVE_OCD_ROM == 1)
        {
        reserve region “OCD ROM area” = mem:[from 0x1FE00 size 0x0200];
        }
}
      |
[Omitted]
      |
if (isdefinedsymbol(__RESERVE_OCD_TRACE_RAM))
{
        if (__RESERVE_OCD_TRACE_RAM == 1)
        {
        reserve region “OCD Trace RAM” = mem:[from 0xFC300 size 0x0400];
        }
}
      |
[Omitted]
      |
if (isdefinedsymbol(__RESERVE_FLASH_SELF_PROGRAMMING_RAM))
{
   if (__RESERVE_FLASH_SELF_PROGRAMMING_RAM == 1)
   {
      reserve region “RESERVED_FLASH_SELF_PROGRAMMING_RAM” = mem:[from 0xFBF00 to 0xFBF7F];
   }
}
```

6.4.3 LLVM Compiler Environment Settings

Points of modifies and examples of modifies when using the LLVM compiler environment (e² studio) is described.

6.4.3.1 Linker Script File Settings

In the sample program “RL78_L23” folder provided by RFD RL78 Type 11, The sections (ROM, RAM, and Data flash range) for RL78/L23 (R7F100LPL) are set.
When using other RL78/L23 products, modify the contents of the sample linker script file “sample_linker_file_xxx.ld” (xxx = CF, DF, EX_FSW or BP_SWAP) provided for the RL78/L23 of RFD RL78 Type 11, because the range of the section settings, “TraceRAM area” and “debug monitor area (OCDROM)” when using the debugger are different.

The following shows the modified part in red text. Refer to the “MCU List for RL78/L23” and modify the setting values for the target device.

Target file name: sample_linker_file_xxx.ld (xxx = CF, DF, EX_FSW or BP_SWAP)

This example shows the modify from RL78/L23 (R7F100LPL) to RL78/L23 (R7F100LLG).

- The start address of the OCDROM (debug monitor area) is set to the address obtained by subtracting “511 bytes (0x1FF)” from the end address of the ROM area; if the end address of the ROM area is “0x1FFFF”, set the ORIGIN of the OCDROM to “0x1FE00” [R-5].

- The size of the ROM area is the area from “0xD8” to the start address of the OCDROM. If the OCDROM start address is “0x1FE00”, set the ROM LENGTH to “130344”, which is the decimal value obtained by subtracting “0xD8” from the OCDROM start address “0x1FE00”.

- The start address ([R-4]+1) and size of “MIRROR (mirror area)” differs depending on the device. For RL78/L23 (R7F100LLG), set “0xF3000”, the start address of the mirror area, to the ORIGIN of the MIRROR. For the LENGTH, set “36608”, the decimal value from the start address “0xF3000” to the end address “0xFBEFF” ([R-1]-1) of the mirror area.
  For more information about the “Mirror area”, please refer to the hardware manual of the device.

- Set the start address of the RAM area “0xFBF00” [R-1] to ORIGIN in the RAM area, and set the LENGTH to “16384”, which is 16 KB in decimal.

- “TRACERAM” area uses an area of 1024 bytes from the address obtained by adding 1024 bytes to the start address of RAM, so set the ORIGIN to “0xFC300” [R-6]. Also, since the trace function may not be used or may not be available for some devices, please refer to the hardware manual of the device for details on the TRACERAM area.

(1) MEMORY settings (common to CF, DF, EX_FSW and BP_SWAP)

Setting for RL78/L23 (ROM: 512 Kbytes, RAM: 32 Kbytes, DF: 8 Kbytes) Example: R7F100LPL

```
MEMORY
{
 VEC : ORIGIN = 0x0, LENGTH = 4
 IVEC : ORIGIN = 0x4, LENGTH = 188
 CALLT0 : ORIGIN = 0x80, LENGTH = 0x40
 OPT : ORIGIN = 0xC0, LENGTH = 4
 SEC_ID : ORIGIN = 0xC4, LENGTH = 10
 OCDSTAD : ORIGIN = 0xCE, LENGTH = 10
 OCDROM : ORIGIN = 0x7FE00, LENGTH = 512          ← [R-5]
 ROM : ORIGIN = 0xD8, LENGTH = 523560
 MIRROR : ORIGIN = 0xF3000, LENGTH = 20224        ← [R-4] + 1
 SADDR : ORIGIN = 0xffe20, LENGTH = 0x000a0
 RAM : ORIGIN = 0xF7F00, LENGTH = 32768           ← [R-1]
 TRACERAM   : ORIGIN = 0xF8300, LENGTH = 1024     ← [R-6]
}
```

Setting for RL78/L23 (ROM: 128 Kbytes, RAM: 16 Kbytes, DF: 8 Kbytes) Example: R7F100LLG

```
MEMORY
{
 VEC : ORIGIN = 0x0, LENGTH = 4
 IVEC : ORIGIN = 0x4, LENGTH = 188
 CALLT0 : ORIGIN = 0x80, LENGTH = 0x40
 OPT : ORIGIN = 0xC0, LENGTH = 4
 SEC_ID : ORIGIN = 0xC4, LENGTH = 10
 OCDSTAD : ORIGIN = 0xCE, LENGTH = 10
 OCDROM : ORIGIN = 0x1FE00, LENGTH = 512
 ROM : ORIGIN = 0xD8, LENGTH = 130344
 MIRROR : ORIGIN = 0xF3000, LENGTH = 36608
 SADDR : ORIGIN = 0xffe20, LENGTH = 0x000a0
 RAM : ORIGIN = 0xFBF00, LENGTH = 16384
 TRACERAM   : ORIGIN = 0xFC300, LENGTH = 1024
}
```

(2) Set the start address of the RAM area

Setting for RL78/L23 (ROM: 512 Kbytes, RAM: 32 Kbytes, DF: 8 Kbytes) Example: R7F100LPL

```
  .data 0xF7F00 : AT(__mdata)                    ← [R-1]
  {
      . = ALIGN(2);
      PROVIDE (__datastart = .);
      __data = .;
      *(.data)
      *(.data.*)
      . = ALIGN(2);
      /*INPUT_SECTION_FLAGS(!SHF_EXECINSTR,SHF_WRITE,SHF_ALLOC)
*(*_n)*/
      __edata = .;
  } >RAM
```

Setting for RL78/L23 (ROM: 128 Kbytes, RAM: 16 Kbytes, DF: 8 Kbytes) Example: R7F100LLG

```
  .data 0xFBF00 : AT(__mdata)
  {
      . = ALIGN(2);
      PROVIDE (__datastart = .);
      __data = .;
      *(.data)
      *(.data.*)
      . = ALIGN(2);
      /*INPUT_SECTION_FLAGS(!SHF_EXECINSTR,SHF_WRITE,SHF_ALLOC)
*(*_n)*/
      __edata = .;
  } >RAM
```

6.4.4 Modifies in the Sample Program (Common to Compilers)

6.4.4.1 Extra Area [FSW Range] Modifying the Reprogramming Sample Program

The number of the maximum blocks of the code flash of RL78/L23(R7F100LPL) differs from RL78/L23(R7F100LLG). Therefore, modify the setting value of FSW range end block macro [END_BLOCK] in “sample_config.h” for Extra area.

[END_BLOCK] indicates the “End block number + 1 ([R-7])” of the FSW range. Also, the comment contains the value of “End block number ([R-7]) of the FSW range” and “End block number + 1 ([R-7]) of the FSW range”.

Example) For RL78/L23 (R7F100LPL), [R-7] is 256 and “[R-7] -1” is 255.
As an example, modifying from RL78/L23 (R7F100LPL) to RL78/L23 (R7F100LLG) is shown.

File Name: sample_config.h
File Path: \sample\RL78_L23\EX_FSW\IAR\include

Setting for RL78/L23 (ROM: 512 Kbytes) Example: R7F100LPL

```
/***********************************************************************************************************************
   User configurable parameters
***********************************************************************************************************************/

/**** CPU frequency (MHz) ****/
/* It must be rounded up digits after the decimal point to form an integer (MHz). */
#define CPU_FREQUENCY (32u)

/**** Block numbers for FSW ****/
/* Start block number for FSW */
#define START_BLOCK     (0u)
/* End block number for FSW */
/* It must be the block number points one block past the end of range for FSW. */
/* If the block number 255 is the end of range for FSW, please specify 256. */   ← [R-7]-1, [R-7]
#define END_BLOCK       (256u)    ← [R-7]
```

Setting for RL78/L23 (ROM: 128 Kbytes) Example: R7F100LLG

```
/***********************************************************************************************************************
   User configurable parameters
***********************************************************************************************************************/

/**** CPU frequency (MHz) ****/
/* It must be rounded up digits after the decimal point to form an integer (MHz). */
#define CPU_FREQUENCY (32u)

/**** Block numbers for FSW ****/
/* Start block number for FSW */
#define START_BLOCK     (0u)
/* End block number for FSW */
/* It must be the block number points one block past the end of range for FSW. */
/* If the block number 63 is the end of range for FSW, please specify 64. */
#define END_BLOCK       (64u)
```

6.4.4.2 Modification of bank programming / bank swapping control sample program

When using the sample program of bank programming / bank swapping control of RFD RL78 Type11,R7F100LxL (CF:512KB) and R7F100LxJ (CF: 256 KB) which are the products corresponding to the bank programming mode of RL78/L23 differ in the bank size of the divided code flash memory. Therefore, it is necessary to modify the value of the top address (“START_ADDRESS_BANK_PROGRAMMING”) for bank programming in a “sample_config.h” file.

“START_ADDRESS_BANK_PROGRAMMING” is calculated by “([R-3]+1) / 2.”

Example) For RL78/L23 (R7F100LPL), [R-3] is 0x7FFFF and “[R-3] +1” is 0x80000 and “([R-3] +1) / 2” is 0x40000.

File Name: sample_config.h

File Path: \sample\RL78_L23\BP_SWAP\CCRL\include

\sample\RL78_L23\BP_SWAP\IAR\include

\sample\RL78_L23\BP_SWAP\LLVM\include

Setting for RL78/L23 (ROM: 512 Kbytes) Example: R7F100LxL

```
/***********************************************************************************************************************
   User configurable parameters
***********************************************************************************************************************/

                |

/**** Option byte data for swapped bank ****/
#define OPTION_BYTE_OCD_SWAPPED_BANK            (0x85u)

/**** Start address of bank programming area ****/
/* It should be the head address of the bank programming area. */
#define START_ADDRESS_BANK_PROGRAMMING          (0x00040000uL)    ← ([R-3]+1) / 2

/**** Address to write reset vector data for swapped bank ****/
#define WRITE_ADDRESS_RESET_VECTOR              (START_ADDRESS_BANK_PROGRAMMING + 0x0000u)
```

Setting for RL78/L23 (ROM: 256 Kbytes) Example: R7F100LxJ

```
/***********************************************************************************************************************
   User configurable parameters
***********************************************************************************************************************/

                |

/**** Option byte data for swapped bank ****/
#define OPTION_BYTE_OCD_SWAPPED_BANK            (0x85u)

/**** Start address of bank programming area ****/
/* It should be the head address of the bank programming area. */
#define START_ADDRESS_BANK_PROGRAMMING          (0x00020000uL)

/**** Address to write reset vector data for swapped bank ****/
#define WRITE_ADDRESS_RESET_VECTOR              (START_ADDRESS_BANK_PROGRAMMING + 0x0000u)
```

7. Revision History

7.1 Major Modifications in this Revision

Rev.	Date	Description	
		Page	Summary
1.00	Apr.25.25	-	Newly created.
1.01	Nov.10.25	-	Description has been improved. Some names were modified and errors in writing were modified.

Renesas Flash Driver RL78 Type 11 User's Manual

Publication Date: Rev.1.01 Nov.10.25

Published by: Renesas Electronics Corporation

Renesas Flash Driver
RL78 Type 11

Renesas Electronics Corporation

R20UT5539EJ0101