User's Manual

# RL78 Family

## Renesas Flash Driver RL78 Type 02

## User's Manual

RENESAS Microcontrollers

RL78/F24

RL78/F23

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev.1.10 Dec 2022

# 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/F23 and RL78/F24 microcontroller.

- Purpose

  This manual is intended to give users an understanding of the methods for using the Renesas Flash Driver (RFD) RL78 Type 02 to reprogram the flash memory in the RL78/F23 and RL78/F24 microcontroller.

- Organization

  This manual is separated into the following sections.

  1. Overview
  2. System Configuration
  3. API Functions of RFD RL78 Type 02
  4. Flash Memory Sequencer Operation
  5. Sample Programs
  6. Creating a Sample Project for RFD RL78 Type 02

- 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/F23 and RL78/F24:

  - Refer to the User's Manual of the target RL78/F23 and RL78/F24 device.

- 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/F23, F24 User’s Manual: Hardware | R01UH0944EJ |
| 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 - Automotive | R20UT5229EJ |

# Table of Contents

## Abbreviations

| Abbreviation | Description |
|---|---|
| RFD | Renesas flash driver |
| API | Application program interface |
| BGO | Background operation<br>Instructions in the code flash memory can be executed during reprogramming of the data flash memory. |
| FSW | Flash shield window<br>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<br>Randomly accessible volatile memory. It is memory for holding values that are to be changed during program execution. |
| ROM | Read-only memory<br>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.<br>Note that this memory may be abbreviated as “CF” in this document. |
| Data flash memory | Flash memory for storing data.<br>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 swap 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.<br>- Code flash memory programming mode<br>- Data flash memory programming mode<br>- 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 | Logical area from 00000H to 03FFFH (16 Kbytes) including the reset vectors |
| Boot clusters 0 and 1 | A boot cluster is a 16-Kbyte group of blocks and either boot cluster 0 or 1 is allocated to the boot area.<br>Physical area name:<br>Boot cluster 0: 00000H to 03FFFH (logical addresses at shipment)<br>Boot cluster 1: 04000H to 07FFFH (logical addresses at shipment) |
| Boot swap | Boot clusters 0 and 1 are swapped. |

# 1. Overview

## 1.1 Outline

Renesas Flash Driver RL78 Type 02 (hereafter called RFD RL78 Type 02) is software for reprogramming the flash memory in the RL78/F23 and RL78/F24.

### 1.1.1 Purpose

The purpose of this document is to give the information about RFD RL78 Type 02.

## 1.2 Contents

The API functions of RFD RL78 Type 02 are called from the user program to reprogram the code flash memory or data flash memory.

The RFD RL78 Type 02 package includes the following.

- This user's manual
- Source code files of RFD RL78 Type 02 for controlling the data flash memory and code flash memory incorporated in the RL78/F23 and RL78/F24
- Sample programs for erasing and reprogramming the data flash memory, code flash memory, and extra area

## 1.3 Features

RFD RL78 Type 02 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 02 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 02.

RFD RL78 Type 02 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.

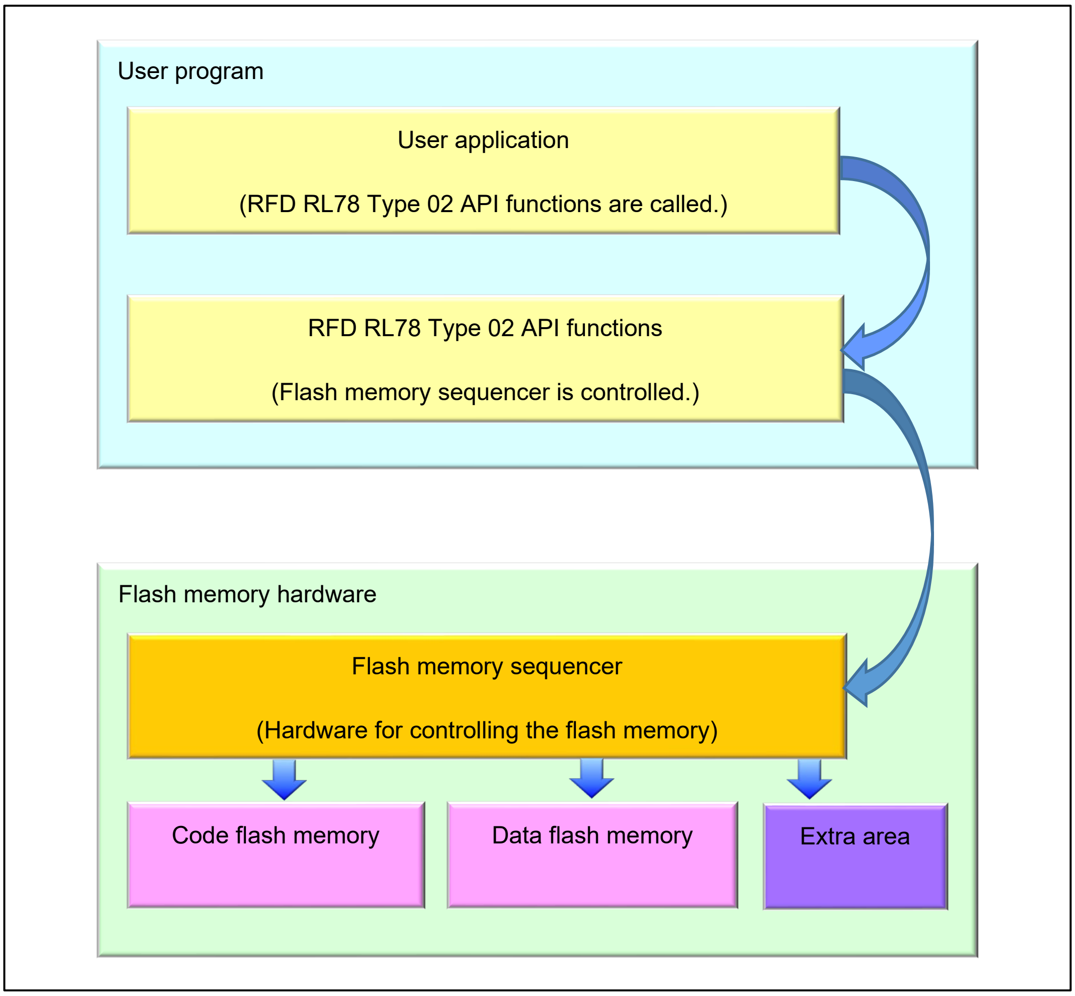

User program
User application
(RFD RL78 Type 02 API functions are called.)
RFD RL78 Type 02 API functions
(Flash memory sequencer is controlled.)
Flash memory hardware
Flash memory sequencer
(Hardware for controlling the flash memory)
Code flash memory
Data flash memory
Extra area

**Figure 1-1 Flash Memory Control Using API Functions of RFD RL78 Type 02**

## 1.4 Operating Environment

- Host Computer

  The operation of RFD RL78 Type 02 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 02 was developed and tested on Windows10 Enterprise.)

- C Compiler Package

  Table 1-1 shows the target C compiler packages for RFD RL78 Type 02.

**Table 1-1 the target C Compiler Packages for RFD RL78 Type 02**

| Package | Manufacturer | Version |
|---|---|---|
| CC-RL (for CS+ or e$^2$studio) | Renesas Electronics | V1.11 or later |
| IAR (Embedded Workbench) | IAR Systems | V4.21 or later |

- Emulator

  Table 1-2 shows the emulator on which the operation of RFD RL78 Type 02 was confirmed.

**Table 1-2 Emulator on which RFD RL78 Type 02 Operation was Confirmed**

| Emulator | Manufacturer |
|---|---|
| E2 emulator | Renesas Electronics |
| E2 emulator Lite | Renesas Electronics |

- Target MCU

  RL78/F24

  RL78/F23

## 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.

(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/F23 and RL78/F24 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.

- In the code flash memory programming mode, the CPU cannot read data from the code flash memory. The API functions of RFD RL78 Type 02 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) Points to note when using the Internal verify command

Execute the internal verify command only once for the target area immediately after writing.

(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”.

## 1.6 C Compiler Definitions

The definitions of the target compiler written in the header file (r_rfd_compiler.h) for RFD RL78 Type 02 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 V4 :
  “__IAR_SYSTEMS_ICC__” is defined
  #define COMPILER_IAR (2)

<Descriptions in the r_rfd_compiler.h file>

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

#if 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))
#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
  ```

# 2. System Configuration

## 2.1 File Structure

### 2.1.1 Folder Structure

Figure 2-1 shows the folder structure of RFD RL78 Type 02.

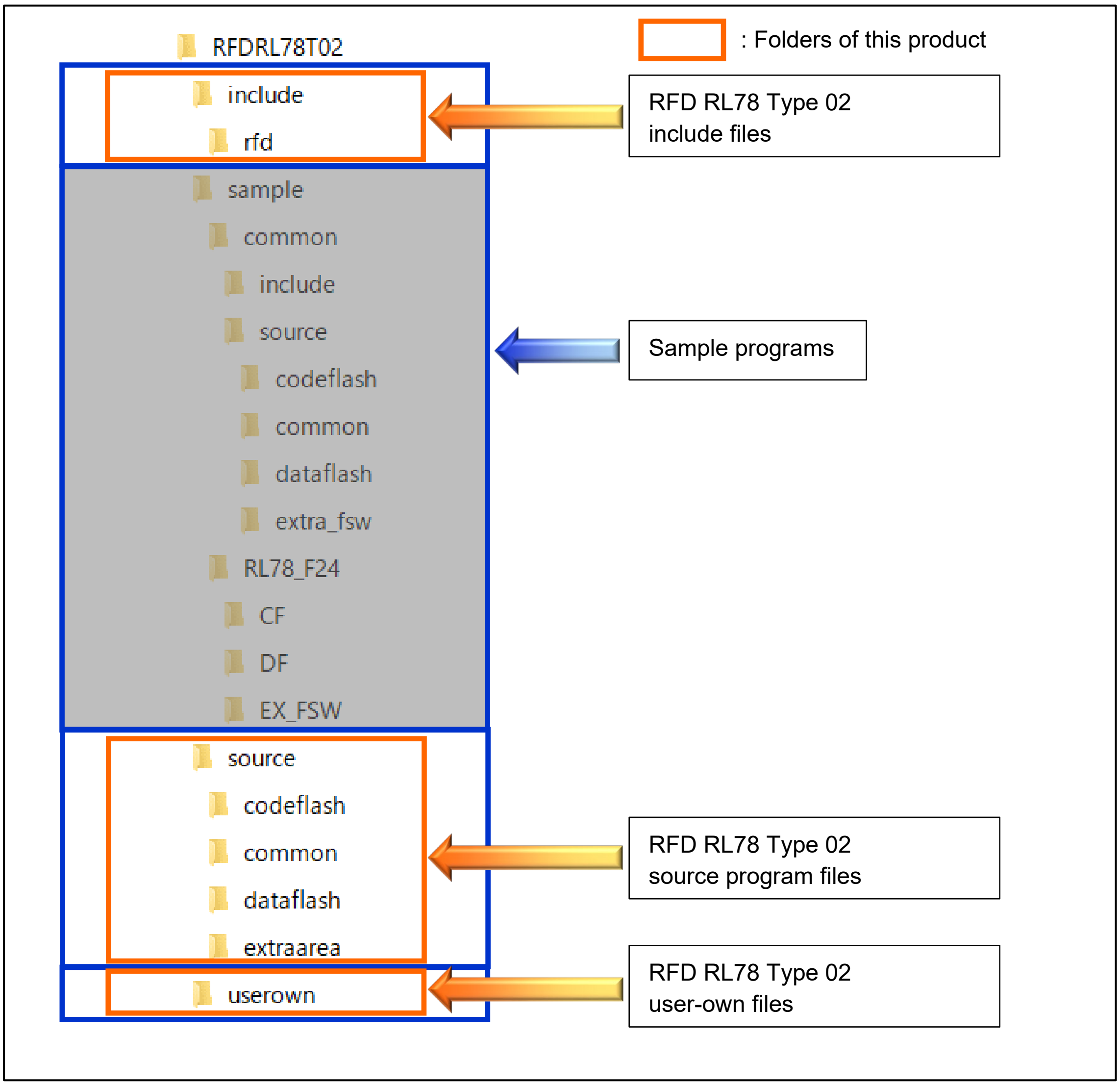


**Figure 2-1 Folder Structure of RFD RL78 Type 02**

Note: Figure 2-1 shows an example of using RL78/F24. 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. |
| 4 | r_rfd_common_extension_api.c | This file contains the API functions for extended facilities 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. |

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. |

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. |

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 02. |

#### 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.<br>This file needs to be included when RFD RL78 Type 02 is used. |
| 2 | r_rfd_compiler.h | This file describes the definitions that differ between compilers used in RFD RL78 Type 02. |
| 3 | r_rfd_memmap.h | This file defines macros to describe sections used in RFD RL78 Type 02. |
| 4 | r_rfd_device.h | This file defines the hardware-specific macros used in RFD RL78 Type 02. |
| 5 | r_rfd_types.h | This file defines the types of variables used in RFD RL78 Type 02. |
| 6 | r_typedefs.h | This file defines the types of data used in RFD RL78 Type 02. |

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_common_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 02. |
| 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. |

## 2.2 Resources of RL78/F23 and RL78/F24

### 2.2.1 Memory Map

Table 2-8 shows the memory map (code flash memory (CF), data flash memory (DF), and RAM) of the RL78/F23 and RL78/F24.

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

| RL78 | Device | Code Flash Memory: CF | RAM |
|---|---|---|---|
| F23 | R7F123FxG (x = B, G, L, M) | 128 Kbytes (00000H-1FFFFH) | 12 Kbytes (FCF00H-FFEFFH) |
| | **Data Flash Memory: DF** | 8 Kbytes (F1000H-F2FFFH) All RL78/F23 | |
| F24 | R7F124FxJ (x = B, G, L, M, P) | 256 Kbytes (00000H-3FFFFH) | 24 Kbytes (F9F00H-FFEFFH) |
| | **Data Flash Memory: DF** | 16 Kbytes (F1000H-F4FFFH) All RL78/F24 | |

### 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/F24.

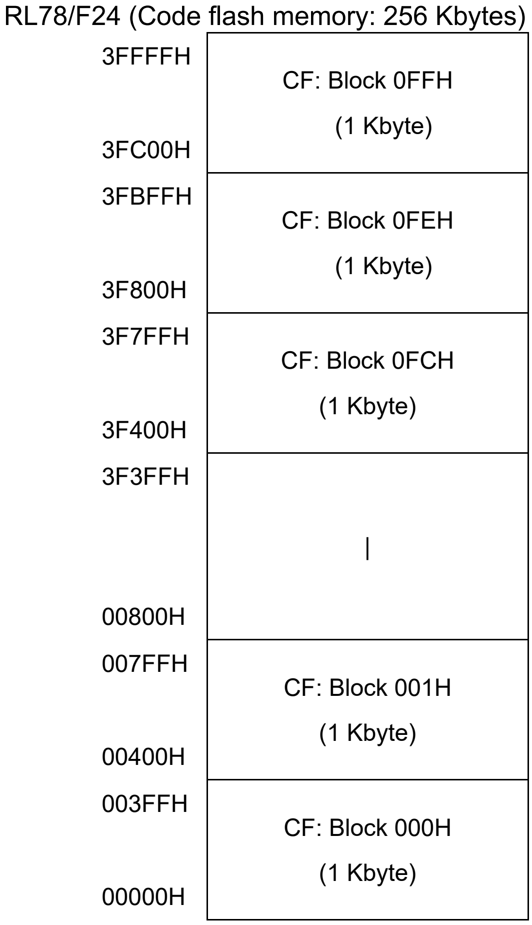


**Figure 2-2 Blocks in the Code Flash Memory**

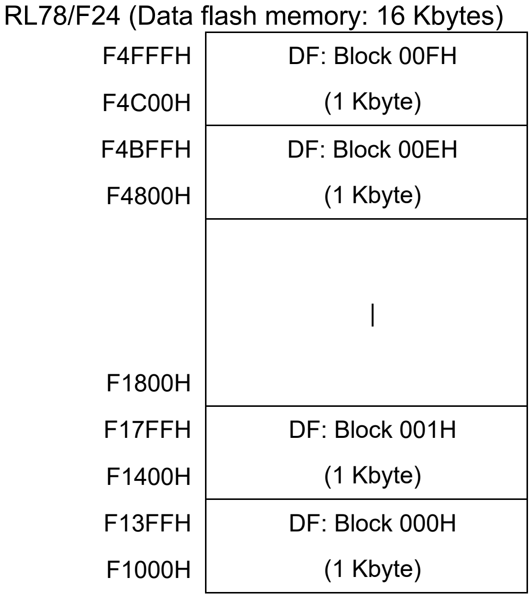


**Figure 2-3 Blocks in the Data Flash Memory**

### 2.2.3 List of Registers Related to Flash Memory Sequencer Control

Table 2-9 shows the registers in the RL78/F23 and RL78/F24 used by RFD RL78 Type 02.

**Table 2-9 Registers in the RL78/F23 and RL78/F24 Used by RFD RL78 Type 02**

| Base Address | Offset | Register Name | Size | Function Name and Note |
|---|---|---|---|---|
| F0000H | 90H | DFLCTL | 1 byte | Data flash control register |
| | C0H | FLPMC | 1 byte | Flash programming mode control register |
| | C1H | FLARS | 1 byte | Flash area select 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 |
| FFF00H | B0H | FLSEC | 2 bytes | Flash security flag monitor 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 |
| | C6H | FLWE | 1 byte | Flash ECC write buffer register |

## 2.3 Resources Used in RFD RL78 Type 02

### 2.3.1 Sections Used in RFD RL78 Type 02

#### 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 2-10 shows the sections used for reprogramming of the code flash memory and allocations of the sections.

**Table 2-10 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 | RAM |
| RFD_CF | Program section of API functions for code flash memory control | RAM |
| RFD_DATA | Data section for initialized global variables | RAM |
| SMP_CMN | Program section of sample functions used in common for flash memory control | RAM |
| SMP_CF | Program section of sample functions for code flash memory control | RAM |

#### 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 2-11 shows the sections used for reprogramming of the data flash memory and allocations of the sections.

**Table 2-11 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_CMN | Program section of sample functions used in common for flash memory control | ROM |
| 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 2-12 shows the sections used for reprogramming of the extra area and allocations of the sections.

**Table 2-12 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 | RAM |
| RFD_EX | Program section of API functions for extra area control | RAM |
| RFD_DATA | Data section for initialized global variables | RAM |
| SMP_CMN | Program section of sample functions used in common for flash memory control | 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 2-13 shows code size and stack size which API functions for RFD RL78 Type 02 use.

**Table 2-13 Code Size and Stack Size which API Functions for RFD RL78 Type 02 Use**

| **API Name** | **Code Size** (Bytes) | | **Stack Size** (Bytes) | |
|---|---|---|---|---|
| | CC-RL | IAR | CC-RL | IAR |
| R_RFD_Init | 37 | 44 | 4 | 4 |
| R_RFD_SetDataFlashAccessMode | 36 | 20 | 10 | 10 |
| R_RFD_SetFlashMemoryMode | 264 | 284 | 14 | 16 |
| R_RFD_CheckFlashMemoryMode | 26 | 36 | 4 | 4 |
| R_RFD_CheckCFDFSeqEndStep1 | 13 | 24 | 4 | 6 |
| R_RFD_CheckExtraSeqEndStep1 | 13 | 23 | 4 | 6 |
| R_RFD_CheckCFDFSeqEndStep2 | 8 | 19 | 4 | 6 |
| R_RFD_CheckExtraSeqEndStep2 | 6 | 19 | 4 | 6 |
| R_RFD_GetSeqErrorStatus | 8 | 8 | 4 | 4 |
| R_RFD_ClearSeqRegister | 11 | 10 | 4 | 4 |
| R_RFD_ForceStopSeq | 6 | 5 | 4 | 4 |
| R_RFD_ForceReset | 2 | 2 | 4 | 4 |
| R_RFD_SetBootAreaImmediately | 15 | 19 | 4 | 4 |
| R_RFD_GetSecurityAndBootFlags | 5 | 5 | 4 | 4 |
| R_RFD_GetFSW | 22 | 24 | 8 | 6 |
| r_rfd_wait_count | 19 | 19 | 6 | 6 |
| R_RFD_EraseCodeFlashReq | 34 | 43 | 4 | 4 |
| R_RFD_WriteCodeFlashReq | 28 | 58 | 4 | 6 |
| R_RFD_BlankCheckCodeFlashReq | 34 | 43 | 4 | 4 |
| R_RFD_IVerifyCodeFlashReq | 34 | 43 | 4 | 4 |
| R_RFD_EraseDataFlashReq | 29 | 41 | 4 | 4 |
| R_RFD_WriteDataFlashReq | 20 | 27 | 4 | 6 |
| R_RFD_BlankCheckDataFlashReq | 34 | 76 | 6 | 12 |
| R_RFD_IVerifyDataFlashReq | 34 | 76 | 6 | 12 |
| R_RFD_SetExtraEraseProtectReq | 24 | 29 | 4 | 4 |
| R_RFD_SetExtraWriteProtectReq | 24 | 29 | 4 | 4 |
| R_RFD_SetExtraBootAreaProtectReq | 24 | 29 | 4 | 4 |
| R_RFD_SetExtraBootAreaReq | 48 | 77 | 4 | 6 |
| R_RFD_SetExtraFSWReq | 21 | 30 | 4 | 4 |
| R_RFD_HOOK_EnterCriticalSection | 9 | 9 | 4 | 4 |
| R_RFD_HOOK_ExitCriticalSection | 11 | 10 | 4 | 4 |

# 3. API Functions of RFD RL78 Type 02

## 3.1 List of API Functions of RFD RL78 Type 02

### 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 02.

**Table 3-1 API Functions Used in Common for Flash Memory Control in RFD RL78 Type 02**

| | API Name | Overview |
|---|---|---|
| 1 | R_RFD_Init | Sets the frequency specified by the parameter in the flash memory sequencer and initializes RFD RL78 Type 02. |
| 2 | R_RFD_SetDataFlashAccessMode | Enables or disables access to the data flash memory according to the parameter setting. |
| 3 | R_RFD_SetFlashMemoryMode | Places the flash memory sequencer in the flash memory control mode specified by the parameter and then sets the specified CPU operating frequency in the flash memory sequencer. |
| 4 | R_RFD_CheckFlashMemoryMode | Checks if the flash memory sequencer is in the mode specified by the parameter. |
| 5 | R_RFD_CheckCFDFSeqEndStep1 | Checks if the operation of the activated code/data flash area sequencer has been completed. |
| 6 | R_RFD_CheckExtraSeqEndStep1 | Checks if the operation of the activated extra area sequencer has been completed. |
| 7 | R_RFD_CheckCFDFSeqEndStep2 | Checks if the command operation has been completed after the flash memory sequencer control register is cleared. |
| 8 | R_RFD_CheckExtraSeqEndStep2 | Checks if the command operation has been completed after the flash memory sequencer control register is cleared. |
| 9 | R_RFD_GetSeqErrorStatus | Acquires the information on errors that occurred during command execution in the code/data flash area sequencer or extra area sequencer. |
| 10 | R_RFD_ClearSeqRegister | Clears the registers for controlling the code/data flash area sequencer and extra area sequencer |
| 11 | R_RFD_ForceStopSeq | Forcibly stops the operation of the code/data flash area sequencer. |
| 12 | R_RFD_ForceReset | Generates an internal reset of the CPU. |
| 13 | R_RFD_SetBootAreaImmediately | Allocates the boot cluster specified by the parameter to the boot area (00000H to 03FFFH) immediately. |
| 14 | R_RFD_GetSecurityAndBootFlags | Acquires the information on the security flags (protection flags) and boot area switching flag. |
| 15 | R_RFD_GetFSW | Acquires the range of the flash shield window, the flash shield window mode, and the protection flag value. |
| 16 | r_rfd_wait_count | Executes a software loop to wait for the time specified by the parameter (time count in units of 1 μs). |

### 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 02.

Table 3-2 API Functions for Code Flash Memory Control in RFD RL78 Type 02

| | 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_IVerifyCodeFlashReq | Activates the code/data flash area sequencer and begins the internal verify of the code flash memory (one block). |

### 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 02.

Table 3-3 API Functions for Data Flash Memory Control in RFD RL78 Type 02

| | 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_IVerifyDataFlashReq | Activates the code/data flash area sequencer and begins the internal verify of the data flash memory (Size of the write data). |

### 3.1.4 API Functions for Extra Area Control

Table 3-4 shows the API functions for extra area control in RFD RL78 Type 02.

**Table 3-4 API Functions for Extra Area Control in RFD RL78 Type 02**

| | API Name | Overview |
|---|---|---|
| 1 | R_RFD_SetExtraEraseProtectReq | Activates the extra area sequencer and begins the setting of the block erase-prohibited flag. |
| 2 | R_RFD_SetExtraWriteProtectReq | Activates the extra area sequencer and begins the setting of the write-prohibited flag. |
| 3 | R_RFD_SetExtraBootAreaProtectReq | Activates the extra area sequencer and begins the setting of the boot area rewrite-prohibited flag. |
| 4 | R_RFD_SetExtraBootAreaReq | Activates the extra area sequencer and begins the setting of the boot area switching flag. |
| 5 | R_RFD_SetExtraFSWReq | Activates the extra area sequencer and begins the setting of the range and mode of the flash shield window specified by the parameters. |

### 3.1.5 Hook Functions

Table 3-5 shows the hook functions in RFD RL78 Type 02.

**Table 3-5 Hook Functions in RFD RL78 Type 02**

| | API Name | Overview |
|---|---|---|
| 1 | R_RFD_HOOK_EnterCriticalSection | Executes the instruction for disabling interrupts. |
| 2 | R_RFD_HOOK_ExitCriticalSection | Executes the instruction for enabling interrupts. |

## 3.2 Data Type Definitions

### 3.2.1 Data Types

Table 3-6 shows the data type definitions in RFD RL78 Type 02.

**Table 3-6 Data Type Definitions in RFD RL78 Type 02**

| 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 02.

(1) 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 (2 MHz to 40 MHz)<br>- Value of (CPU operating frequency – 1): 0x01u to 0x27u (1 to 39) |
| Definition file | r_rfd_common_api.c |

(2) g_u08_fset_cpu_frequency

| | |
|---|---|
| Type/Name | uint8_t g_u08_fset_cpu_frequency |
| Default value | 0x00 (R_RFD_VALUE_U08_INIT_VARIABLE) |
| Description | Value to be set to FSET bit of FSSET register. |
| Definition file | r_rfd_common_api.c |

(3) sg_u08_psw_ie_state

| | |
|---|---|
| Type/Name | static uint8_t sg_u08_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<br>- Interrupts are disabled: 0x00u<br>- Interrupts are enabled: 0x80u |
| Definition file | r_rfd_common_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_flash_memory_mode (enumerated-type variable name: e_rfd_flash_memory_mode_t)
  Flash memory control mode

| Symbol Name | Value | Description |
|---|---|---|
| R_RFD_ENUM_FLASH_MODE_CODE_PROGRAMMING | 0x01 | Code flash memory programming mode |
| R_RFD_ENUM_FLASH_MODE_DATA_PROGRAMMING | 0x02 | Data flash memory programming mode |
| R_RFD_ENUM_<br>FLASH_MODE_CODE_TO_NONPROGRAMMABLE | 0x03 | Non-programmable mode<br>[Transition from the code flash programming mode.] |
| R_RFD_ENUM_<br>FLASH_MODE_DATA_TO_NONPROGRAMMABLE | 0x04 | Non-programmable mode<br>[Transition from the data flash programming mode.] |

- 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_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 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 |

#### 3.2.4.2 Macro Definitions for Setting the Registers and Extra Area in the RL78/F23 and RL78/F24

- 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 select 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_IVERIFY_CF | 0x82u | Internal verify command for the code flash memory |
| R_RFD_VALUE_U08_FSSQ_IVERIFY_DF | 0x8Au | Internal verify command for the data 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 2 to 0] ESQMD2 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 | 0x82u | Command for setting the flash shield window function |
| R_RFD_VALUE_U08_FSSE_SECURITY_FLAG | 0x81u | 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_<br>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.

  Target register definition: R_RFD_REG_U08_FLPMC

| Symbol Name | Value | Description |
|---|---|---|
| R_RFD_VALUE_U08_FLPMC_MODE_ NONPROGRAMMABLE | 0x08u | The flash memory sequencer is in the non-programmable mode. |
| R_RFD_VALUE_U08_FLPMC_MODE_ CODE_FLASH_PROGRAMMING | 0x82u | Code flash memory programming mode |
| R_RFD_VALUE_U08_FLPMC_MODE_ DATA_FLASH_PROGRAMMING | 0x10u | Data flash memory programming mode |

| Symbol Name | Value | Description |
|---|---|---|
| R_RFD_VALUE_U08_FLPMC_ TRANSFER_1ST_LAYER | 0x12u | Control value used for the transition to code flash memory programming mode, or the transition to non-programmable mode from code flash memory programming mode. |
| R_RFD_VALUE_U08_FLPMC_ TRANSFER_1ST_LAYER_INVERSION | 0xEDu | Control value used for the transition to code flash memory programming mode, or the transition to non-programmable mode from code flash memory programming mode.<br>(Inverted 0x12u) |
| R_RFD_VALUE_U08_FLPMC_ TRANSFER_2ND_LAYER | 0x92u | Control value used for the transition to code flash memory programming mode, or the transition to non-programmable mode from code flash memory programming mode. |
| R_RFD_VALUE_U08_FLPMC_ TRANSFER_2ND_LAYER_INVERSION | 0x6Du | Control value used for the transition to code flash memory programming mode, or the transition to non-programmable mode from code flash memory programming mode.<br>(Inverted 0x92u) |

- 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 2] IVER: Error status of the internal verify command. IVER = 1 indicates an internal verify 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_<br>MASK1_FSASTL_ERROR_FLAG | 0x3Fu | 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 swap setting bit, temporary boot swap setting bit, or other setting bits are masked by ANDing with 0.

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

  [Bit 6] TMBTSEL: Temporary boot swap 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_MASK1_<br>FSSET_TMSPMD_AND_TMBTSEL | 0xC0u | The boot swap setting and temporary boot swap setting are masked. |
| R_RFD_VALUE_U08_MASK0_<br>FSSET_TMSPMD_AND_TMBTSEL | 0x3Fu | The bits other than the boot swap setting or temporary boot swap setting are masked. |
| R_RFD_VALUE_U08_MASK1_FSSET_TMSPMD | 0x80u | The bits other than the boot swap setting are masked. |
| R_RFD_VALUE_U08_<br>FSSET_BOOT_CLUSTER_0 | 0x80u | Value for specifying boot cluster 0 for temporary boot swap. |
| R_RFD_VALUE_U08_<br>FSSET_BOOT_CLUSTER_1 | 0xC0u | Value for specifying boot cluster 1 for temporary boot swap. |

- 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, and the correction shift value 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 | 2u | Lowest allowable operating frequency (2 MHz) |
| R_RFD_VALUE_U08_FREQUENCY_UPPER_LIMIT | 40u | Highest allowable operating frequency (40 MHz) |
| R_RFD_VALUE_U08_FREQUENCY_ADJUST | 1u | Correction value (-1) for conversion of the FSSET register setting |
| R_RFD_VALUE_U08_FREQUENCY_SHIFT_ADJUST | 1u | Correction shift value for conversion of the FSSET register setting |
| R_RFD_VALUE_U08_FREQUENCY_ CALC_THRESHOLD | 23u | Threshold for calculating the FSSET register setting |

- 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 monitor registers START and END)
  The mask values used to acquire or make the FSW settings are defined.
  FLFSWE [bits 9 to 0]: The end block number +1 of FSW is specified.
  FLFSWS [bits 9 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 | 0x03FFu | Mask value for acquiring the block number setting |

(2) Mask value for making FSW settings

| Symbol Name | Value | Description |
|---|---|---|
| R_RFD_VALUE_U16_MASK1_FSW_BLOCK_INFO | 0x03FFu | 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 = 1 Kbyte) 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_<br>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_<br>DATA_FLASH_ADDR_HIGH | 0x0Fu | Value for the upper bits of the start address of a data flash area (8 bits) |
| R_RFD_VALUE_U08_<br>DATA_FLASH_BLOCK_ADDR_LOW | 0x3Fu | Mask value for the lower bits of the start address of a data flash block (8 bits). |
| R_RFD_VALUE_U16_<br>DATA_FLASH_BLOCK_ADDR_END | 0x03FFu | Value for the lower bits of the end address of a data flash block (16 bits) |
| R_RFD_VALUE_U08_<br>DATA_FLASH_SHIFT_LOW_ADDR | 10u | 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 = 1 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_<br>CODE_FLASH_BLOCK_ADDR_LOW | 0x003Fu | Mask value for the lower bits of the start address of a code flash block (16 bits) |
| R_RFD_VALUE_U16_<br>CODE_FLASH_BLOCK_ADDR_HIGH | 0x03C0u | 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_<br>CODE_FLASH_BLOCK_ADDR_END | 0x03FFu | Lower address in 1-Kbyte units of the end of a code flash block (16 bits) |
| R_RFD_VALUE_U08_<br>CODE_FLASH_SHIFT_LOW_ADDR | 10u | Value for shifting the lower address bits to calculate the offset of a code flash area from the block number |
| R_RFD_VALUE_U08_<br>CODE_FLASH_SHIFT_HIGH_ADDR | 6u | 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:
0x002B -> 0xAC00 (shifted to the left by 10 bits)

R_RFD_VALUE_U16_CODE_FLASH_BLOCK_ADDR_HIGH:
0x0040 -> 0x0001 (shifted to the right by 6 bits)

Block start address = 0x0001_AC00

- Macro definitions for FLSEC (flash security flag monitor register)
  The mask values for extra area settings and security monitoring are defined.
  [Bit 12] WRPR: Write-prohibited flag. WRPR = 0 disables programming.
  [Bit 10] SEPR: Block erase-prohibited flag. SEPR = 0 disables block erasure.
  [Bit 9] BTPR: Flag for controlling the protection against reprogramming of the boot block cluster.
  BTPR = 0 disables reprogramming of the boot block cluster.
  [Bit 8] BTFLG: Boot area switching flag.
  BTFLG = 0: Boot cluster 1 is used as the boot area.
  BTFLG = 1: Boot cluster 0 is used as the boot area.
  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_<br>BOOT_CLUSTER_PROTECT_FLAG | 0xFDFFu | Mask value for setting the protection against reprogramming of the boot block cluster |
| 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 |

## 3.3 Specifications of API Functions

This section describes the detailed specifications of the API functions of Renesas Flash Driver (RFD) RL78 Type 02.

There are some prerequisites for using the API functions of RFD RL78 Type 02 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 02 only while the high-speed on-chip oscillator is active.
- To control the data flash memory, execute API functions of RFD RL78 Type 02 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<br>(Enumerated type, pointer type, etc.) | Enumerator (constant value) of the return value: Value<br>[Meaning of the constant: Detailed description] |
| | | Enumerator (constant value) of the return value: Value<br>[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 02.

#### 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 [2 to 40 (MHz)] |
| Parameters (IN/OUT) | N/A | |
| Parameters (OUT) | N/A | |
| Return Value | e_rfd_ret_t | R_RFD_ENUM_RET_STS_OK: 0x00<br>[Normal end: The frequency is within the allowable range.] |
| | | R_RFD_ENUM_RET_ERR_PARAMETER: 0x10<br>[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 02. | |
| 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:**

- Whether the value of the parameter (CPU operating frequency) is within the range from 2 MHz to 40 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”.
- The value which converted the CPU operating frequency inputted into the FSSET register is set to “g_u08_fset_cpu_frequency”.

  - When the argument (i_u08_cpu_frequency) is less than or equal to the threshold (23MHz)

    g_u08_fset_cpu_frequency = (i_u08_cpu_frequency -1)

  - When the argument (i_u08_cpu_frequency) exceeds threshold (23MHz)

    g_u08_fset_cpu_frequency = (i_u08_cpu_frequency + Threshold) >> 1u

**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 02 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 2 MHz, or 3 MHz can be used but a non-integer value such as 2.5 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<br>(e_rfd_df_access_t i_e_df_access); | |
| Reentrancy | Non-reentrant | |
| Parameters<br>(IN) | e_rfd_df_access_t | Control of access to the data flash memory |
| | i_e_df_access | R_RFD_ENUM_DF_ACCESS_ENABLE: 0x01<br>[Access to the data flash memory is enabled.]<br>R_RFD_ENUM_DF_ACCESS_DISABLE: 0x00<br>[Access to the data flash memory is disabled.] |
| Parameters<br>(IN/OUT) | N/A | |
| Parameters<br>(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: 4μsec). 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_SetFlashMemoryMode

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_SetFlashMemoryMode<br>(e_rfd_flash_memory_mode_t i_e_flash_mode); | |
| Reentrancy | Non-reentrant | |
| Parameters<br>(IN) | e_rfd_flash_memory_mode_t | Flash memory control mode |
| | i_e_flash_mode | R_RFD_ENUM_FLASH_MODE_CODE_PROGRAMMING : 0x01<br>[Code flash memory programming mode]<br>R_RFD_ENUM_FLASH_MODE_DATA_PROGRAMMING: 0x02<br>[Data flash memory programming mode]<br>R_RFD_ENUM_FLASH_MODE_CODE_TO_ NONPROGRAMMABLE : 0x03<br>[Non-programmable mode: Transition from the code flash memory programming mode]<br>R_RFD_ENUM_FLASH_MODE_DATA_TO_ NONPROGRAMMABLE : 0x04<br>[Non-programmable mode: Transition from the data flash memory programming mode] |
| Parameters<br>(IN/OUT) | N/A | |
| Parameters<br>(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<br>[Mode mismatch error]<br>(The flash memory sequencer is not placed in the specified mode.) |
| Description | Places the flash memory sequencer in the flash memory control mode specified by the parameter 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_EnterCriticalSection() is called to save the current interrupt disabled (DI) or enabled (EI) state and disable interrupts.
- The FLPMC register is set up according to the value of the parameter (i_e_flash_mode) to place the flash memory sequencer in the specified flash memory control mode.
- Before a transition to the specified mode, a wait time (tMS) is inserted. For the wait time (tMS), refer to the hardware manual of the target RL78 microcontroller.
- The hook function R_RFD_HOOK_ExitCriticalSection() 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_EnterCriticalSection() and R_RFD_HOOK_ExitCriticalSection(). If interrupts are enabled and an interrupt occurs in this period, the subsequent operation is indeterminate.
- If the value specified by the parameter is not a flash memory control mode value, the operation is same as that for the non-programmable mode (transition from data flash memory programming mode).
- 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 02, 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 or data flash memory programming mode, please transition from non-programmable mode.

### 3.3.1.4 R_RFD_CheckFlashMemoryMode

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_CheckFlashMemoryMode<br>(e_rfd_flash_memory_mode_t i_e_flash_mode); | |
| Reentrancy | Non-reentrant | |
| Parameters (IN) | e_rfd_flash_memory_mode_t | Flash memory control mode |
| | i_e_flash_mode | R_RFD_ENUM_FLASH_MODE_CODE_PROGRAMMING: 0x01<br>[Code flash memory programming mode]<br>R_RFD_ENUM_FLASH_MODE_DATA_PROGRAMMING: 0x02<br>[Data flash memory programming mode]<br>R_RFD_ENUM_FLASH_MODE_CODE_TO_<br>NONPROGRAMMABLE: 0x03<br>[Non-programmable mode]<br>R_RFD_ENUM_FLASH_MODE_DATA_TO_<br>NONPROGRAMMABLE: 0x04<br>[Non-programmable mode] |
| 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 | Checks if the flash memory sequencer is in the mode specified by the parameter. | |
| 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 if it matches the register value for the mode specified by the parameter (i_e_flash_mode).
  - Non-programmable mode: 0x08
  - Code flash memory programming mode: 0x82
  - Data flash memory programming mode: 0x10

**Notes:**

- If the control mode of the flash memory sequencer was specified by a function other than R_RFD_SetFlashMemoryMode(), 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.
- When the value other than flash memory control mode is specified for the argument, the same processing as when the non-programmable mode is set is performed.

### 3.3.1.5 R_RFD_CheckCFDFSeqEndStep1

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_CheckCFDFSeqEndStep1(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<br>[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_STS_BUSY is returned.<br>After confirming that R_RFD_ENUM_RET_STS_OK has been returned from this function, execute the R_RFD_CheckCFDFSeqEndStep2() 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 area sequencer has been completed, the flash memory sequencer control register is cleared (FSSQ = 0x00) 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_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_CheckCFDFSeqEndStep2() function.

### 3.3.1.6 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<br>[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_STS_BUSY is returned.<br>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, the flash memory sequencer control register is cleared (FSSE = 0x00) 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_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.

### 3.3.1.7 R_RFD_CheckCFDFSeqEndStep2

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC e_rfd_ret_t R_RFD_CheckCFDFSeqEndStep2(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<br>[Normal end: Sequencer operation has been completed.] |
| | | R_RFD_ENUM_RET_STS_BUSY: 0x01<br>[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_CheckCFDFSeqEndStep1() function. | |
| Remarks | Execute this function again if R_RFD_STS_BUSY is returned. | |

**Details of Specifications:**

- Whether the command operation in the code/data flash area sequencer has been completed (SQEND (bit 6 of FSASTH) = 0) is checked after the flash memory sequencer control register is cleared (FSSQ = 0x00).
- When the command execution in the 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_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_CheckCFDFSeqEndStep1() function, this function is not executed correctly.

### 3.3.1.8 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<br>[Normal end: Sequencer operation has been completed.] |
| | | R_RFD_ENUM_RET_STS_BUSY: 0x01<br>[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 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_STS_BUSY is returned. | |

**Details of Specifications:**

- Whether all command execution in the extra area sequencer has been completed (ESQEND (bit 7 of FSASTH) = 0) is checked after the flash memory sequencer control register is cleared (FSSE = 0x00).
- 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_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.

### 3.3.1.9 R_RFD_GetSeqErrorStatus

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC void R_RFD_GetSeqErrorStatus<br>(uint8_t __near * onp_u08_error_status); | |
| Reentrancy | Non-reentrant | |
| Parameters (IN) | N/A | |
| Parameters (IN/OUT) | N/A | |
| Parameters (OUT) | uint8_t __near *<br>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 to 6 are set to a fixed value of 0.
  Error information to be acquired (six 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 2: Internal verify 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.

#### 3.3.1.10 R_RFD_ClearSeqRegister

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC void R_RFD_ClearSeqRegister(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 or data flash memory programming mode.<br>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_CheckCFDFSeqEndStep2() or 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 or data flash memory programming mode, the subsequent operation is indeterminate.

### 3.3.1.11 R_RFD_ForceStopSeq

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC void R_RFD_ForceStopSeq(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).<br>Use this function before the R_RFD_CheckCFDFSeqEndStep1() function returns R_RFD_ENUM_RET_STS_OK (before the sequencer operation is completed). | |
| Remarks | Execute the R_RFD_CheckCFDFSeqEndStep1() 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, internal verify 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, internal verify, 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.

#### 3.3.1.12 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.

### 3.3.1.13 R_RFD_SetBootAreaImmediately

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC void R_RFD_SetBootAreaImmediately<br>(e_rfd_boot_cluster_t i_e_boot_cluster); | |
| Reentrancy | Non-reentrant | |
| Parameters<br>(IN) | e_rfd_boot_cluster_t<br>i_e_boot_cluster | Boot cluster number |
| | | R_RFD_ENUM_BOOT_CLUSTER_0: 0x01<br>[Boot cluster 0]<br>R_RFD_ENUM_BOOT_CLUSTER_1: 0x00<br>[Boot cluster 1] |
| Parameters<br>(IN/OUT) | N/A | |
| Parameters<br>(OUT) | N/A | |
| Return Value | N/A | |
| Description | Allocates the boot cluster specified by the parameter to the boot area (00000H to 03FFFH) immediately. | |
| Preconditions | Use this function in the code flash memory programming mode or data flash memory programming mode.<br>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 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.

**Notes:**

- If an unallowable value is specified by the parameter (i_e_boot_cluster), boot cluster 0 is allocated to the boot area.
- The boot cluster that is not selected as the boot area is allocated to the area (04000H to 07FFFH) immediately following the boot area (00000H to 03FFFH).
- 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.

### 3.3.1.14 R_RFD_GetSecurityAndBootFlags

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC void R_RFD_GetSecurityAndBootFlags<br>(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 *<br>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-prohibited flag
  Bit 11: -
  Bit 10 (SEPR): Block erase-prohibited flag
  Bit 9 (BTPR): Boot area rewrite-prohibited flag
  Bit 8 (BTFLG): Boot area switching flag
  Bits 7 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.

### 3.3.1.15 R_RFD_GetFSW

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC void R_RFD_GetFSW<br>(uint16_t __near * onp_u16_start_block_number,<br>uint16_t __near * onp_u16_end_block_number); | |
| 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 |
| Return Value | N/A | |
| Description | Acquires the range of the flash shield window. | |
| 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 values of the FLFSWS register (16 bits) and FLFSWE register (16 bits) that indicate the start block and end block +1 of the flash shield window 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 9 to 0 of FLFSWS. Bits 15 to 10 are masked with 0.)

    *onp_u16_end_block_number: End block +1
    (Setting in bits 9 to 0 of FLFSWE. Bits 15 to 10 are masked with 0.)

**Notes:**

- If this function is executed in the initial state of the device, onp_u16_start_block_number = 1023 and onp_u16_end_block_number = 1023 are acquired.
- 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.

#### 3.3.1.16 r_rfd_wait_count

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC void r_rfd_wait_count(uint8_t i_u08_count); | |
| Reentrancy | Non-reentrant | |
| Parameters (IN) | uint8_t<br>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.

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

This section describes the API functions for code flash memory control in RFD RL78 Type 02.

#### 3.3.2.1 R_RFD_EraseCodeFlashReq

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC void R_RFD_EraseCodeFlashReq<br>(uint16_t i_u16_block_number); | |
| Reentrancy | Non-reentrant | |
| Parameters (IN) | uint16_t<br>i_u16_block_number | Target block number for erasure [0 to 511]<br>Example: For RL78/F24, 0 to 255 (256 Kbytes max.)<br>Example: For RL78/F23, 0 to 127 (128 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_CheckCFDFSeqEndStep1() 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 (1 Kbyte) to be erased in the code flash memory is set in the sequencer.
  - The start address and end address of the target block (1 Kbyte) in the code flash memory are calculated from the block number for erasure specified by the parameter (i_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 10 bits of the 16-bit parameter (i_u16_block_number) are used; the upper 6 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.

#### 3.3.2.2 R_RFD_WriteCodeFlashReq

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC void R_RFD_WriteCodeFlashReq<br>(uint32_t i_u32_start_addr,<br>uint8_t __near * inp_u08_write_data); | |
| Reentrancy | Non-reentrant | |
| Parameters (IN) | uint32_t<br>i_u32_start_addr | Target start address for programming (4-byte boundary)<br>[Address in the code flash area] |
| | uint8_t __near *<br>inp_u08_write_data | Pointer to the variable that stores write data<br>[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_CheckCFDFSeqEndStep1() 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<br>(uint16_t i_u16_block_number); | |
| Reentrancy | Non-reentrant | |
| Parameters<br>(IN) | uint16_t<br>i_u16_block_number | Target block number for blank check [0 to 511]<br>Example: For RL78/F24, 0 to 255 (256 Kbytes max.)<br>Example: For RL78/F23, 0 to 127 (128 Kbytes max.) |
| Parameters<br>(IN/OUT) | N/A | |
| Parameters<br>(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_CheckCFDFSeqEndStep1() 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 (1 Kbyte) 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 (1024 bytes) 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 10 bits of the 16-bit parameter (i_u16_block_number) are used; the upper 6 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.

#### 3.3.2.4 R_RFD_IVerifyCodeFlashReq

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC void R_RFD_IVerifyCodeFlashReq<br>(uint16_t i_u16_block_number); | |
| Reentrancy | Non-reentrant | |
| Parameters (IN) | uint16_t<br>i_u16_block_number | Target block number for internal verify [0 to 511]<br>Example: For RL78/F24, 0 to 255 (256 Kbytes max.)<br>Example: For RL78/F23, 0 to 127 (128 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 internal verify 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_CheckCFDFSeqEndStep1() 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 top address of one block (1024byte) that executes internal verify of the code flash memory.
  - The start address and end address of the target block (1024 bytes) in the code flash memory are calculated from the block number for internal verify 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_IVERIFY_CF = 0x82 is set in the FSSQ register to start the internal verify.
  (SQST (bit 7) = 1, MDCH (bit 3) = 0, SQMD (bits 2 to 0) = 2 (0b010), and the other bits are set to 0.)

**Notes:**

- The lower 10 bits of the 16-bit parameter (i_u16_block_number) are used; the upper 6 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 internal verify can be executed only once on the target block immediately after writing. Do not execute internal verification more than once.

### 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 02.

#### 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<br>i_u08_block_number | Target block number for erasure [0 to 63]<br>Example: For RL78/F24, 0 to 15 (16 Kbytes max.)<br>Example: For RL78/F23, 0 to 7 (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).<br>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_CheckCFDFSeqEndStep1() 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 (1024 bytes) to be erased in the data flash memory is set in the sequencer.
  - The start address and end address of the target block (1024 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.

### 3.3.3.2 R_RFD_WriteDataFlashReq

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC void R_RFD_WriteDataFlashReq<br>(uint32_t i_u32_start_addr,<br>uint8_t __near * inp_u08_write_data); | |
| Reentrancy | Non-reentrant | |
| Parameters<br>(IN) | uint32_t<br>i_u32_start_addr | Target start address for programming<br>[Address in the data flash area] |
| | uint8_t __near *<br>inp_u08_write_data | Pointer to the variable that stores write data<br>[Size of the write data pointed to is 1 byte] |
| Parameters<br>(IN/OUT) | N/A | |
| Parameters<br>(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).<br>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_CheckCFDFSeqEndStep1() 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<br>(uint32_t i_u32_start_addr,<br>uint16_t i_u16_blankcheck_length); | |
| Reentrancy | Non-reentrant | |
| Parameters (IN) | uint32_t<br>i_u32_start_addr | Target start address for blank check<br>[Address in the data flash area] |
| | uint16_t<br>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).<br>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_CheckCFDFSeqEndStep1() 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_IVerifyDataFlashReq

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC void R_RFD_IVerifyDataFlashReq<br>(uint32_t i_u32_start_addr,<br>uint16_t i_u16_iverify_length); | |
| Reentrancy | Non-reentrant | |
| Parameters (IN) | uint32_t<br>i_u32_start_addr | Target start address for internal verify<br>[Address in the data flash area] |
| | uint16_t<br>i_u16_iverify_length | Target data length for internal verify |
| Parameters (IN/OUT) | N/A | |
| Parameters (OUT) | N/A | |
| Return Value | N/A | |
| Description | Activates the code/data flash area sequencer and begins the internal verify of the data flash memory (Size of the write data). | |
| Preconditions | Use this function while access to the data flash memory is enabled (DFLEN = 1).<br>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_CheckCFDFSeqEndStep1() 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 sets the address at which to initiate the internal verify of the data flash memory and the number of bytes of interest.
  - The start address of the data flash memory for the internal verify of an argument (i_u32_start_addr) is set in the FLAPL and FLAPH register.
  - The end address (start address + size of the write data) of the data flash memory is calculated, and it sets in the FLSEDL and FLSEDH register.
- R_RFD_VALUE_U08_FSSQ_IVERIFY_DF = 0x8A is set in the FSSQ register to start the blank check. (SQST (bit 7) = 1, MDCH (bit 3) = 1, SQMD (bits 2 to 0) = 2 (0b010), 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.
- The internal verify can be executed only once for the area immediately after writing. Do not execute internal verification more than once.

### 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 02.

#### 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-prohibited 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-prohibited 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 = 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 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-prohibited 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-prohibited 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 = 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 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-prohibited 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-prohibited 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 = 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 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<br>(e_rfd_boot_cluster_t i_e_boot_cluster); | |
| Reentrancy | Non-reentrant | |
| Parameters<br>(IN) | e_rfd_boot_cluster_t<br>i_e_boot_cluster | Boot cluster number<br>R_RFD_ENUM_BOOT_CLUSTER_0: 0x01<br>[Boot cluster 0]<br>R_RFD_ENUM_BOOT_CLUSTER_1: 0x00<br>[Boot cluster 1] |
| Parameters<br>(IN/OUT) | N/A | |
| Parameters<br>(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 swap is executed only after a reset instead of immediately after the setting of the BTFLG.
  - The FSSET and FLSEC registers are read.
  - 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 swap is executed according to the information in the extra area (BTFLG). |
| | 1: Boot swap 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. |

- 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 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 = 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:**

- 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.

  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).
- 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_SetExtraFSWReq

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC void R_RFD_SetExtraFSWReq<br>(uint16_t i_u16_start_block_number,<br>uint16_t i_u16_end_block_number); | |
| Reentrancy | Non-reentrant | |
| Parameters<br>(IN) | uint16_t<br>i_u16_start_block_number | Start block number<br>Example: For RL78/F24, 0 to 255 (256 Kbytes max.)<br>Example: For RL78/F23, 0 to 127 (128 Kbytes max.) |
| | uint16_t<br>i_u16_end_block_number | End block number +1<br>Example: For RL78/F24, 1 to 256 (256 Kbytes max.)<br>Example: For RL78/F23, 1 to 128 (128 Kbytes max.) |
| Parameters<br>(IN/OUT) | N/A | |
| Parameters<br>(OUT) | N/A | |
| Return Value | N/A | |
| Description | Activates the extra area sequencer and begins the setting of the range and mode 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 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 flash shield window and the flash shield window mode 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 10 (the bits other than the block address bits) are set to 0.
  - 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 10 (the bits other than the block address bits) are set to 0.
- R_RFD_VALUE_U08_FSSE_FSW = 0x82 is set in the FSSE register to start the setting.
  (ESQST (bit 7) = 1, ESQMD (bits 2 to 0) = 2 (0b010), and the other bits are set to 0.)

**Notes:**

- Bits 9 to 0 of a 16-bit parameter are used as the block number to be set (the maximum number is 1023); bits 15 to 10 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 of the desired window range plus 1.

  Examples:

  - To shield the areas outside the four blocks from block 12 to block 15:

    i_u16_start_block_number = 12, i_u16_end_block_number = 16
- 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.

### 3.3.5 Specifications of Hook Functions

This section describes the hook functions of RFD RL78 Type 02.

#### 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 | 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 | Executes the instruction for enabling interrupts. | |
| Preconditions | Execute this function to enable interrupts after the processing executed with interrupts disabled. | |
| Remarks | — | |

**Details of Specifications:**

- According to the value of the variable sg_u08_psw_ie_state, which saves the interrupt enable flag (IE) of the PSW, the macro instruction for enabling interrupts is executed.

  Value of sg_u08_psw_ie_state:

  - 0x00 (bit 7 = 0: interrupts are disabled): Nothing is done.
  - 0x80 (bit 7 = 1: interrupts are enabled): The macro instruction for enabling interrupts (R_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.

# 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_SetFlashMemoryMode

- 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 writing to other memory spaces or registers is attempted between steps 1 to 4 in the specific sequence, the FLPMC register is not written to. 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 |
|---|---|---|---|---|---|---|---|
| FLPMC[7 : 0] | | | | | | | |
| R/W | | | | | | | |

| Code and Data flash mode | FLPMC register setting |
|---|---|
| Non-programmable mode<br>(Read mode) | This state is the state after reset.<br>When transitioning from Code flash programming mode and Data flash programming mode, perform in a specific sequence. (See 4.1.4, Procedure for transition to the Non-programmable Mode). |
| Code flash programming mode | This mode can only be transitioned from Read mode.<br>Set FLPMC register in a specific sequence (See 4.1.2, Procedure for transition from non-programmable mode to the Code Flash Memory Programming Mode). |
| Data flash programming mode | This mode can only be transitioned from Read mode.<br>Set FLPMC register in a specific sequence (See 4.1.3, Procedure for transition from non-programmable mode to the Data Flash Memory Programming Mode). |

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 = 0x12 |
| Step 3: | FLPMC register = 0xED |
| Step 4: | FLPMC register = 0x12 |
| Step 5: | 3μs Wait. |
| Step 6: | PFCMD register = 0xA5 |
| Step 7: | PFCMD register = 0x92 |
| Step 8: | PFCMD register = 0x6D |
| Step 9: | PFCMD register = 0x92 |
| Step 10: | PFCMD register = 0xA5 |
| Step 11: | PFCMD register = 0x82 |
| Step 12: | PFCMD register = 0x7D |
| Step 13: | PFCMD register = 0x82 |
| Step 14: | 10μs Wait. |

- Steps 2 and 4
  FLPMC register setting (0x12)
- Step 3
  Inverted value or FLPMC register setting (0xED)

- Steps 7 and 9
  FLPMC register setting (0x92)
- Step 8
  Inverted value or FLPMC register setting (0x6D)

- Steps 11 and 13
  FLPMC register setting (0x82)
- Step 12
  Inverted value or FLPMC register setting (0x7D)

### 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 |
| Step 5: | 10μs Wait. |

- Steps 2 and 4
  FLPMC register setting (0x10)
- 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 wait time (10μs) 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 transition from data flash memory programming mode to non-programmable mode

| | |
|---|---|
| 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)
- 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 transition from code flash memory programming mode to non-programmable mode

| | |
|---|---|
| Step 1: | PFCMD register = 0xA5 |
| Step 2: | FLPMC register = 0x92 |
| Step 3: | FLPMC register = 0x6D |
| Step 4: | FLPMC register = 0x92 |
| Step 5: | 3μs Wait |
| Step 6: | PFCMD register = 0xA5 |
| Step 7: | FLPMC register = 0x12 |
| Step 8: | FLPMC register = 0xED |
| Step 9: | FLPMC register = 0x12 |
| Step 10: | PFCMD register = 0xA5 |
| Step 11: | FLPMC register = 0x08 |
| Step 12: | FLPMC register = 0xF7 |
| Step 13: | FLPMC register = 0x08 |

- Steps 2 and 4
  FLPMC register setting (0x92)
- Step 3
  Inverted value or FLPMC register setting (0x6D)

- Steps 7 and 9
  FLPMC register setting (0x12)
- Step 8
  Inverted value or FLPMC register setting (0xED)

- Steps 11 and 13
  FLPMC register setting (0x08)
- Step 12
  Inverted value or FLPMC register setting (0xF7)

Step 14: 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 2 MHz, or 3 MHz can be specified. A non-integer frequency such as 2.5 MHz cannot be used.

Target functions of this operation: R_RFD_Init and R_RFD_SetFlashMemoryMode

**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 (2MHz to 40 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/F23 and RL78/F24 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 select 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 | DCLR | 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 |
| 0x1 | CF: 0<br>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.<br>• Code flash memory programming (1 word (4 bytes)):<br>Specify data in the FLWH and FLWL registers.<br>• Data flash memory programming (1 byte):<br>Specify data in the FLW7 to FLW0 bits (bits 7 to 0) of the FLWL register. |
| 0x2 | CF: 0<br>DF: 1 | Internal verify |
| | | Internal verify 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. |
| 0x3 | CF: 0<br>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. |
| 0x4 | CF: 0<br>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: **1 Kbyte**
- Write unit: **1 word (4 bytes)**

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

**Operation Procedure:**

Block erase, write, blank check, and internal verify 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: 0x0023FF)

(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: 0x0023FF)

Note: To perform blank check only in a 1-word (4-byte) area, set FLAPH = FLSEDH and FLAPL = FLSEDL.

(4) Internal verify:

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: 0x0023FF)

Note: This command can be executed only once in the area immediately after writing. Do not execute it more than once.

- 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 Internal verify: 0x82
- 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: **1 Kbyte**
- Write unit: **1 byte**

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

**Operation Procedure:**

Block erase, write, blank check, and internal verify 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: 0x0F17FF)

(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: 0x0F17FF)

Note: To perform blank check only in a 1-byte area, set FLAPH = FLSEDH and FLAPL = FLSEDL.

(4) Internal verify:

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: 0x0F15FF)

Note: This command can be executed only once in the area immediately after writing. Do not execute it more than once.

- 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)**

  Internal verify: 0x8A **(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 ESQMD2 to ESQMD0 bits (bits 2 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 | 0 | ESQMD2 | ESQMD1 | ESQMD0 |
| R/W | R | R | R | R | 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**

| ESQMD2 to ESQMD0 | Function of Dedicated Command |
|---|---|
| | Description |
| 0x1 | 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. |
| 0x2 | 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 is set up. |
| 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, and R_RFD_SetExtraFSWReq

**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 ESQMD2 to ESQMD0 (bits 2 to 0) bits of the FSSE register and set the ESQST bit (bit 7) to 1.

(1) Programming of the security flags and the boot area switching flag: 0x81

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.

(2) Programming of the FSW-related data: 0x82

- 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 |
| - | - | - | - | - | - | FSWE9 | 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 |
| - | - | - | - | - | - | FSWS9 | 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 |

- FSWE9 to FSWE0 (bits 25 to 16): Specify the value of (end block +1) of the window range.
- FSWS9 to FSWS0 (bits 9 to 0): Specify the start block of the window range.

(2) 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 swap 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)**

### 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/F23 and RL78/F24, 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), IVER (bit 2), WRER (bit 1), and ERER (bit 0)) of the FSASTL register to check whether an error has occurred in the execution of the respective commands.

FSASTH register (After reset: 0x00):

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

FSASTL register (After reset: 0x00):

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| MBTSEL | | ESEQER | SEQER | BLER | IVER | 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 | DCLR | 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 Swap 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 (00000H to 03FFFH), 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. The boot swap function is provided to avoid this situation.

### 4.5.2 Operation of the Boot Swap Function

The boot swap function replaces boot cluster 0 (00000H to 03FFFH), which is the boot area, with boot cluster 1 (04000H to 07FFFH), which is the target area of boot swap. Before starting the reprogramming processing, write a new boot program to boot cluster 1 (04000H to 07FFFH). Swap boot cluster 1 (04000H to 07FFFH) and boot cluster 0 (00000H to 03FFFH) so that boot cluster 1 is allocated to the boot area (00000H to 03FFFH). Even if a temporary power failure occurs during reprogramming of the boot area after this swap, booting by the next reset is done in boot cluster 1 (00000H to 03FFFH), which stores the new boot program, and the user program can be executed correctly.

| | |
|---|---|
| Boot area | Logical area from 00000H to 03FFFH including the reset vector address |
| Boot clusters 0 and 1 | A boot cluster is a 16-Kbyte group of blocks and either boot cluster 0 or 1 is allocated to the boot area.<br>Physical area name:<br>Boot cluster 0: 00000H to 03FFFH (logical addresses at shipment)<br>Boot cluster 1: 04000H to 07FFFH (logical addresses at shipment) |

Note: The logical addresses of boot cluster 0 and boot cluster 1 are switched after boot swap.

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).

The operation of the boot swap 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 a 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 a 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 |

### 4.5.3 Execution of the Boot Swap Function

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

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.

#### 4.5.3.1 Immediate Execution of Boot Swap

The specified boot cluster is immediately allocated to the boot area (00000H to 03FFFH) (boot swap is performed immediately).

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

Target function of this operation: R_RFD_SetBootAreaImmediately

**Operation Procedures:**

(1) When the TMSPMD bit = 0 (boot swap 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 swap according to TMBTSEL) and the TMBTSEL bit to 0. Boot swap is executed immediately.

  b) When the TMBTSEL bit = 0:
    - Set the TMSPMD bit to 1 (boot swap according to TMBTSEL) and the TMBTSEL bit to 1. Boot swap is executed immediately.

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

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

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

#### 4.5.3.2 Boot Swap Execution after a Reset

Boot swap is not executed immediately after the BTFLG is written to but executed after a 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 swap is not executed.

Target function of this operation: R_RFD_SetExtraBootAreaReq

**Operation Procedures:**

(1) When the TMSPMD bit = 0 (boot swap 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 swap 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 = 0x1 in the FSSE register)
    - Boot swap 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 swap 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 = 0x1 in the FSSE register)
    - Boot swap 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 swap according to TMBTSEL):

- Write to the BTFLG bit in the **extra area**. (Specify the boot cluster to be used as the boot area. ESQMD = 0x1 in the FSSE register)
- Boot swap is executed after the reset operation and execution branches to the reset vector address in the specified boot cluster.

## 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 end block +1.

### 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 monitor 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 a reset or when the extra area is programmed):

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | FSWE9 | 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 a reset or when the extra area is programmed):

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | FSWE9 | 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 9 to 0) of the FLFSWE register: Specify the end block number +1 of the window range.
- FSWS (bits 9 to 0) of the FLFSWS register: Specify the start block number of the window range.

### 4.6.3 Execution of the Flash Shield Window Function

#### 4.6.3.1 Control of the Flash Shield Window Mode

Target function of this operation: R_RFD_SetExtraFSWReq

**Operation Procedure:**

- Write to the FSWE, and FSWS bits in the **extra area**. (ESQMD = 0x1 in the FSSE register)
  FSWE: End block number +1 of the FSW window range.
  FSWS: Start block of the FSW window range.

Note: Set reserved bits (bits 15 to 10) to 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.

Example: Target device = R7F124FPJ
Specify "start block = 03H", "end block = 05H" as window ranges.

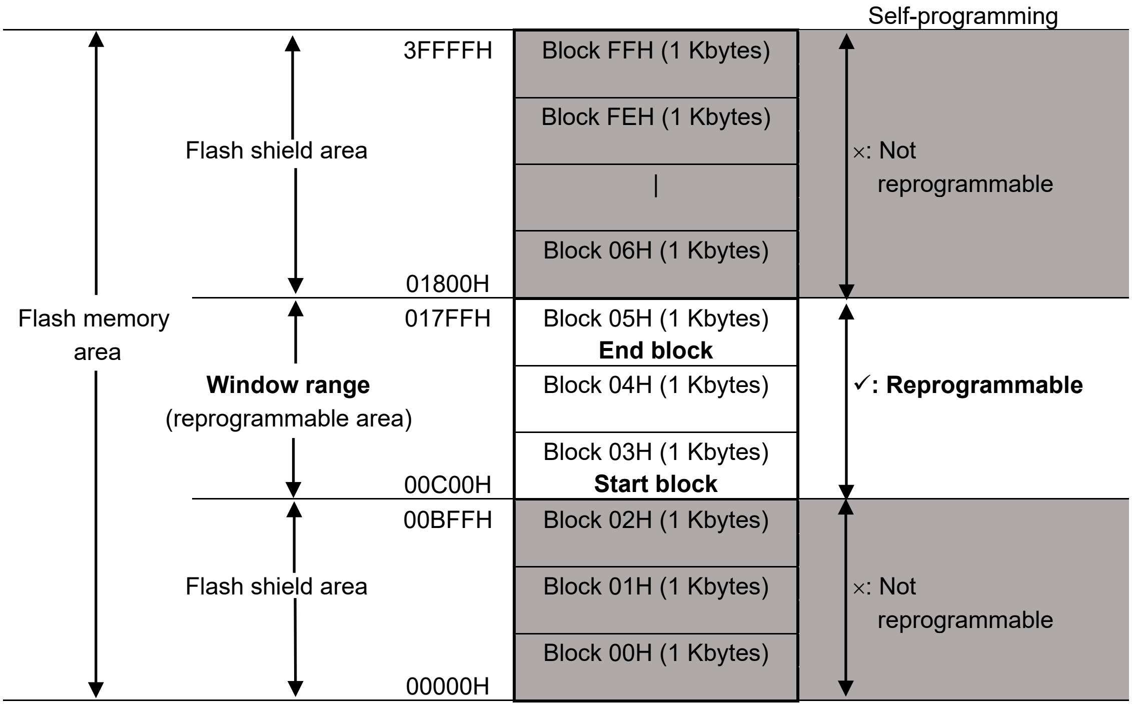


**Figure 4-1 Example of FSW Settings**

## 4.7 Examples of Command Execution for Reprogramming of Flash Areas

### 4.7.1 Example of Command Execution for Reprogramming of the Code Flash Area

Figure 4-2 shows a flowchart of command execution for reprogramming of the code flash area.

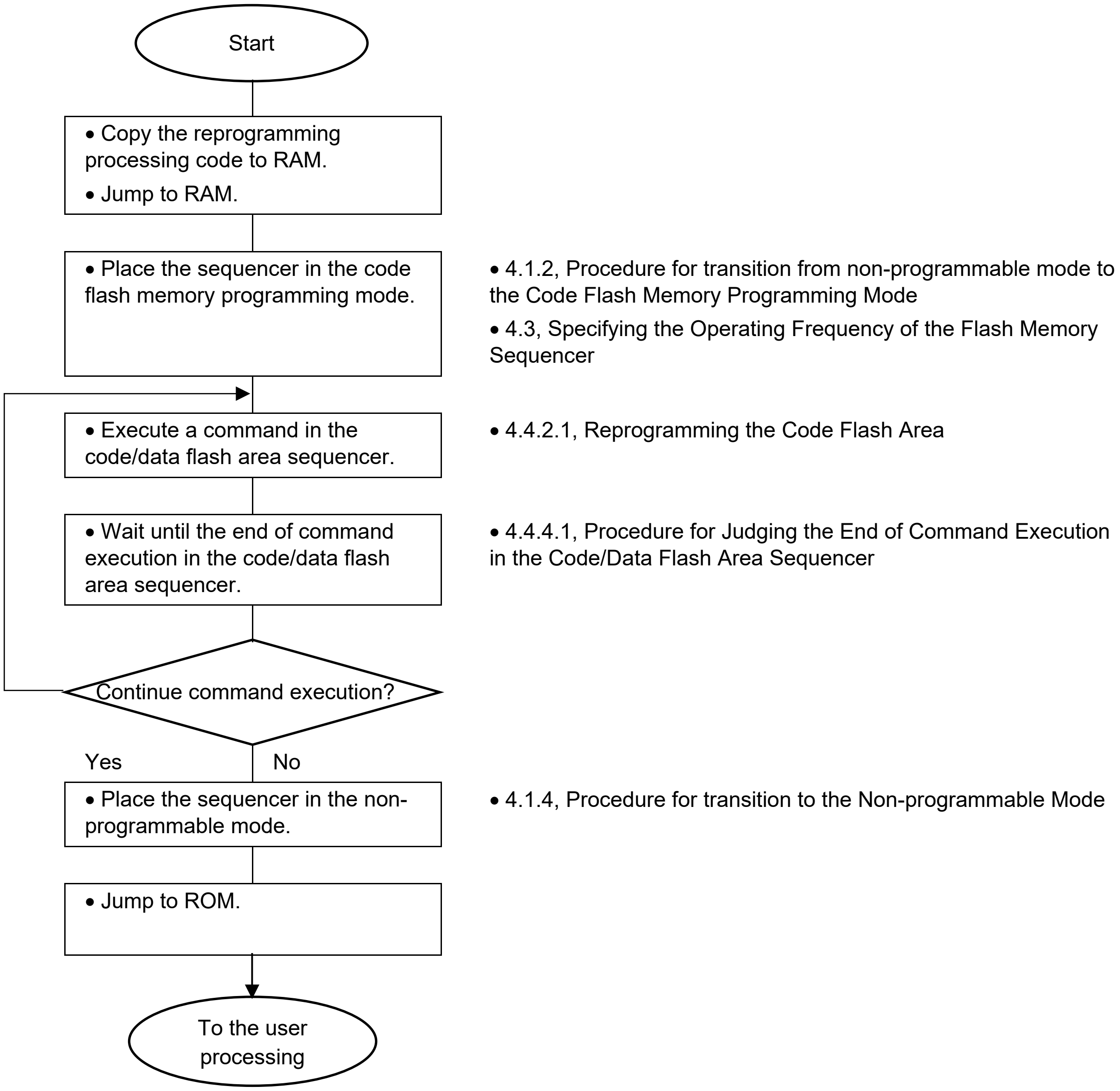
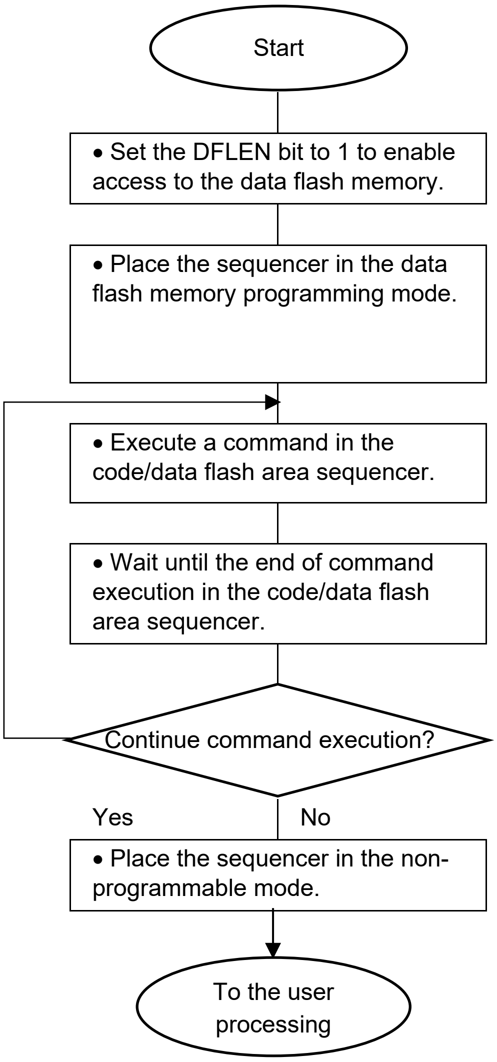

Start
• Copy the reprogramming processing code to RAM.
• Jump to RAM.
• Place the sequencer in the code flash memory programming mode.
• 4.1.2, Procedure for transition from non-programmable mode to the Code Flash Memory Programming Mode
• 4.3, Specifying the Operating Frequency of the Flash Memory Sequencer
• Execute a command in the code/data flash area sequencer.
• 4.4.2.1, Reprogramming the Code Flash Area
• Wait until the end of command execution in the code/data flash area sequencer.
• 4.4.4.1, Procedure for Judging the End of Command Execution in the Code/Data Flash Area Sequencer
Continue command execution?
Yes
No
• Place the sequencer in the non-programmable mode.
• 4.1.4, Procedure for transition to the Non-programmable Mode
• Jump to ROM.
To the user processing

**Figure 4-2 Flowchart of Command Execution for Reprogramming of the Code Flash Area**

### 4.7.2 Example of Command Execution for Reprogramming of the Data Flash Area

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

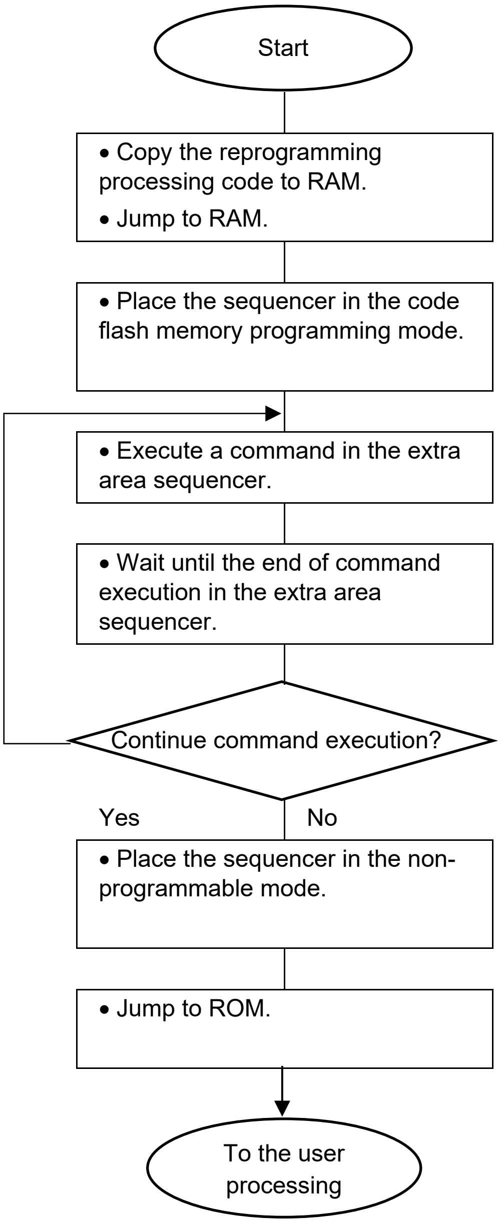

Start
• Set the DFLEN bit to 1 to enable access to the data flash memory.
• Place the sequencer in the data flash memory programming mode.
• Execute a command in the code/data flash area sequencer.
• Wait until the end of command execution in the code/data flash area sequencer.
Continue command execution?
Yes
No
• Place the sequencer in the non-programmable mode.
To the user processing

- 4.1.3, Procedure for transition from non-programmable mode to the Data Flash Memory Programming Mode
- 4.3, Specifying the Operating Frequency of the Flash Memory Sequencer
- 4.4.2.2, Reprogramming the Data Flash Area
- 4.4.4.1, Procedure for Judging the End of Command Execution in the Code/Data Flash Area Sequencer
- 4.1.4, Procedure for transition to the Non-programmable Mode

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

### 4.7.3 Example of Command Execution for Reprogramming of the Extra Area

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


Start
• Copy the reprogramming processing code to RAM.
• Jump to RAM.
• Place the sequencer in the code flash memory programming mode.
• Execute a command in the extra area sequencer.
• Wait until the end of command execution in the extra area sequencer.
Continue command execution?
Yes
No
• Place the sequencer in the non-programmable mode.
• Jump to ROM.
To the user processing

- 4.1.2, Procedure for transition from non-programmable mode to the Code Flash Memory Programming Mode
- 4.3, Specifying the Operating Frequency of the Flash Memory Sequencer

- 4.4.3.1, Reprogramming the Extra Area

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

- 4.1.4, Procedure for transition to the Non-programmable Mode

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

# 5. Sample Programs

This section describes the sample programs provided together with RFD RL78 Type 02.

## 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/F24. The installed “sample” folder contains a folder for each device group (e.g. RL78_F24).

The "RL78_F24" folder is used when using RL78/F23 and RL78/F24.

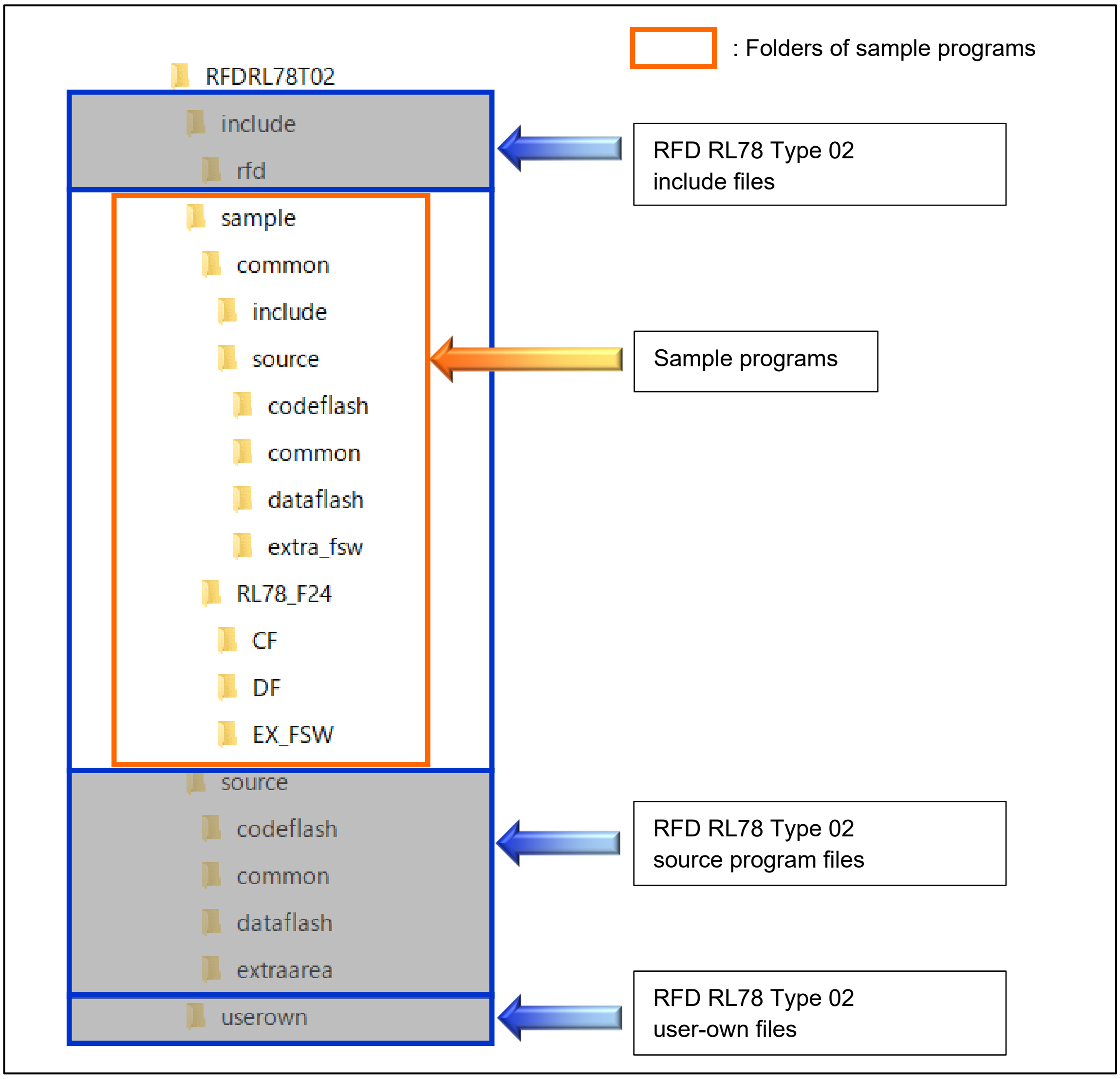


**Figure 5-1 Structure of Sample Program Folders**

### 5.1.2 List of Files

#### 5.1.2.1 List of Source Files

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

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

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

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

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

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

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

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

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

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

**Table 5-4 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 5-5 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_F24” folder.

- Main processing for controlling the code flash memory (CF):
  “sample\RL78_F24\CF\[compiler name]\source\” folder
- Main processing for controlling the data flash memory (DF):
  “sample\RL78_F24\DF\[compiler name]\source\” folder
- Main processing for controlling the FSW in the extra area (EX_FSW):
  “sample\RL78_F24\EX_FSW\[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) |

#### 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_common.h | This file defines the prototype declarations of the sample functions used in common for controlling the flash memory. |
| 2 | sample_control_data_flash.h | This file defines the prototype declarations of the sample functions for controlling the data flash memory. |
| 3 | sample_control_code_flash.h | This file defines the prototype declarations of the sample functions for controlling the code flash memory. |
| 4 | sample_control_extra_fsw.h | This file defines the prototype declarations of the sample functions for controlling the FSW in the extra area. |
| 5 | sample_defines.h | This file defines the macros of the sample functions for controlling the flash memory. |
| 6 | sample_memmap.h | This file defines the macros that describes the sections used by the sample program that controls the flash memory. |
| 7 | 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_ACT_IVERIFY | 0x25u | Internal verify 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_IVERIFY | 0x33u | Internal verify command error |
| SAMPLE_ENUM_RET_ERR_CMD_SET_EXTRA_AREA | 0x34u | Extra area command setting error |

## 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 | Sample_CheckCFDFSeqEnd | Waits for the completion of command execution in the code/data flash area sequencer. |
| 8 | Sample_CheckExtraSeqEnd | Waits for the completion of command execution in the extra area sequencer. |

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

The sample program for controlling the reprogramming of the code flash memory in RFD RL78 Type 02 erases block 28 (00007000H) in the code flash area and writes 256-word (1024-byte) data from the beginning of the block.

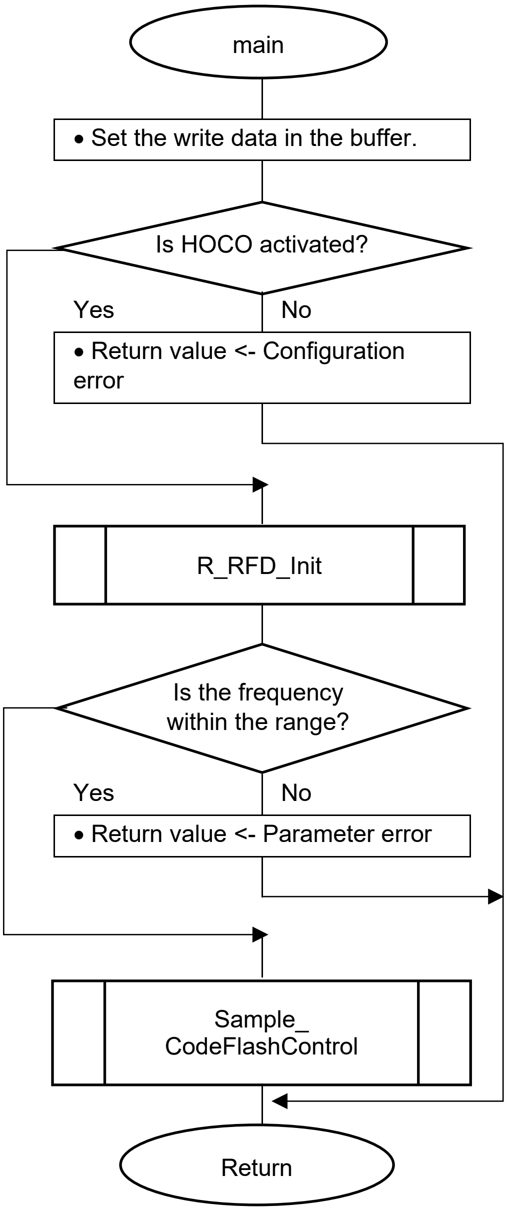
**Note: In the code flash memory programming mode, the programs in the code flash memory cannot be executed. Copy the Sample_CodeFlashControl function and the processing to be executed and data to be referenced within the function to RAM in advance, and execute and reference them in RAM.**

**Operating conditions (Example of a sample program for RL78/F24):**

- CPU operating frequency: 40 MHz (The high-speed on-chip oscillator clock is used for the main system clock.)
- Code flash memory address for erasure and programming: 00007000H
- Block number for erasure: 001CH
- Size of write data: 256 words (1024 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 02.

#### 5.3.1.1 main Function


main
• Set the write data in the buffer.
Is HOCO activated?
Yes
No
• Return value <- Configuration error
R_RFD_Init
Is the frequency within the range?
Yes
No
• Return value <- Parameter error
Sample_ CodeFlashControl
Return

• Initializes RFD RL78 Type 02 (specifies the operating frequency).

• CPU operating frequency range of the RL78/F24: 2 MHz to 40 MHz
• Within range: Returns “normal end” (0x00).
Out of range: Returns “parameter error” (0x10).

• Processing for controlling the code flash memory reprogramming
• Return value <- Value returned from the 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.

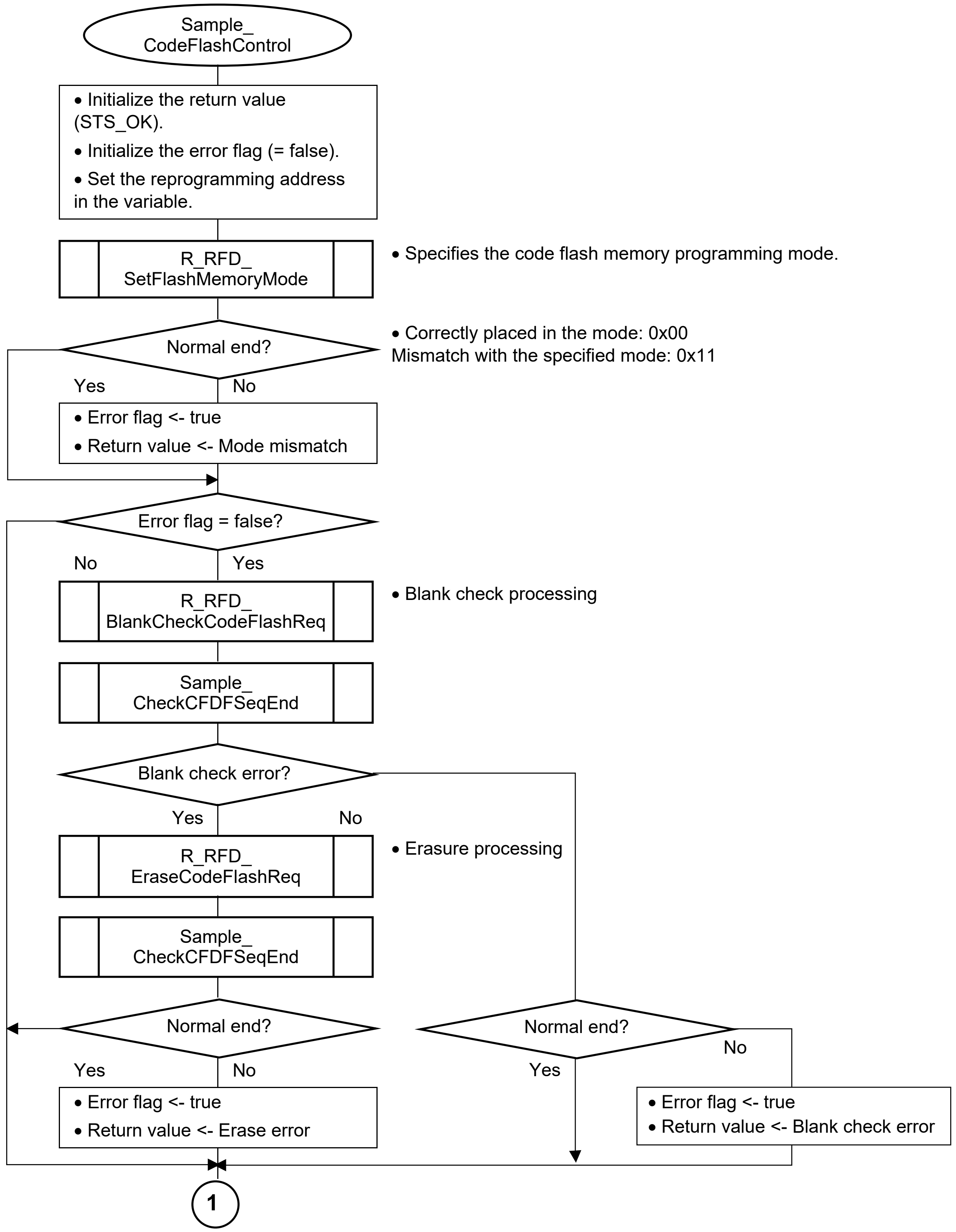
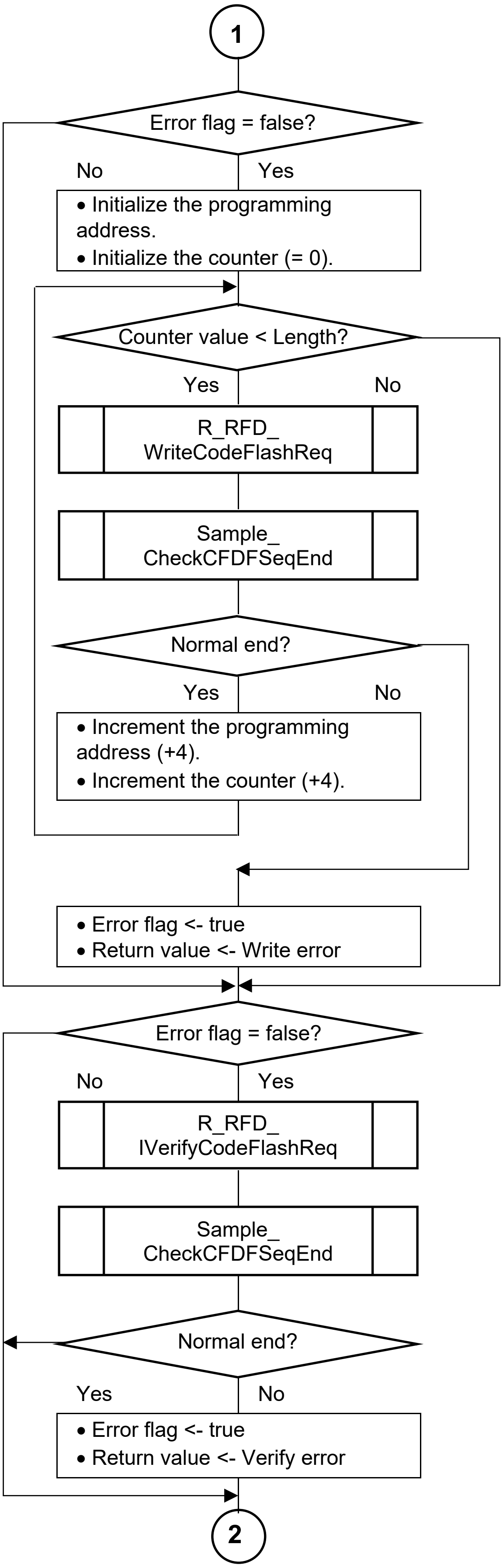

Sample_
CodeFlashControl
• Initialize the return value (STS_OK).
• Initialize the error flag (= false).
• Set the reprogramming address in the variable.
R_RFD_
SetFlashMemoryMode
• Specifies the code flash memory programming mode.
Normal end?
• Correctly placed in the mode: 0x00
Mismatch with the specified mode: 0x11
Yes
No
• Error flag <- true
• Return value <- Mode mismatch
Error flag = false?
No
Yes
R_RFD_
BlankCheckCodeFlashReq
• Blank check processing
Sample_
CheckCFDFSeqEnd
Blank check error?
Yes
No
R_RFD_
EraseCodeFlashReq
• Erasure processing
Sample_
CheckCFDFSeqEnd
Normal end?
Yes
No
• Error flag <- true
• Return value <- Erase error
Normal end?
No
Yes
• Error flag <- true
• Return value <- Blank check error
1

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

- Programming, and internal verify are executed.


1
Error flag = false?
No
Yes
• Initialize the programming address.
• Initialize the counter (= 0).
Counter value < Length?
Yes
No
R_RFD_ WriteCodeFlashReq
Sample_ CheckCFDFSeqEnd
Normal end?
Yes
No
• Increment the programming address (+4).
• Increment the counter (+4).
• Error flag <- true
• Return value <- Write error
Error flag = false?
No
Yes
R_RFD_ IVerifyCodeFlashReq
Sample_ CheckCFDFSeqEnd
Normal end?
Yes
No
• Error flag <- true
• Return value <- Verify error
2

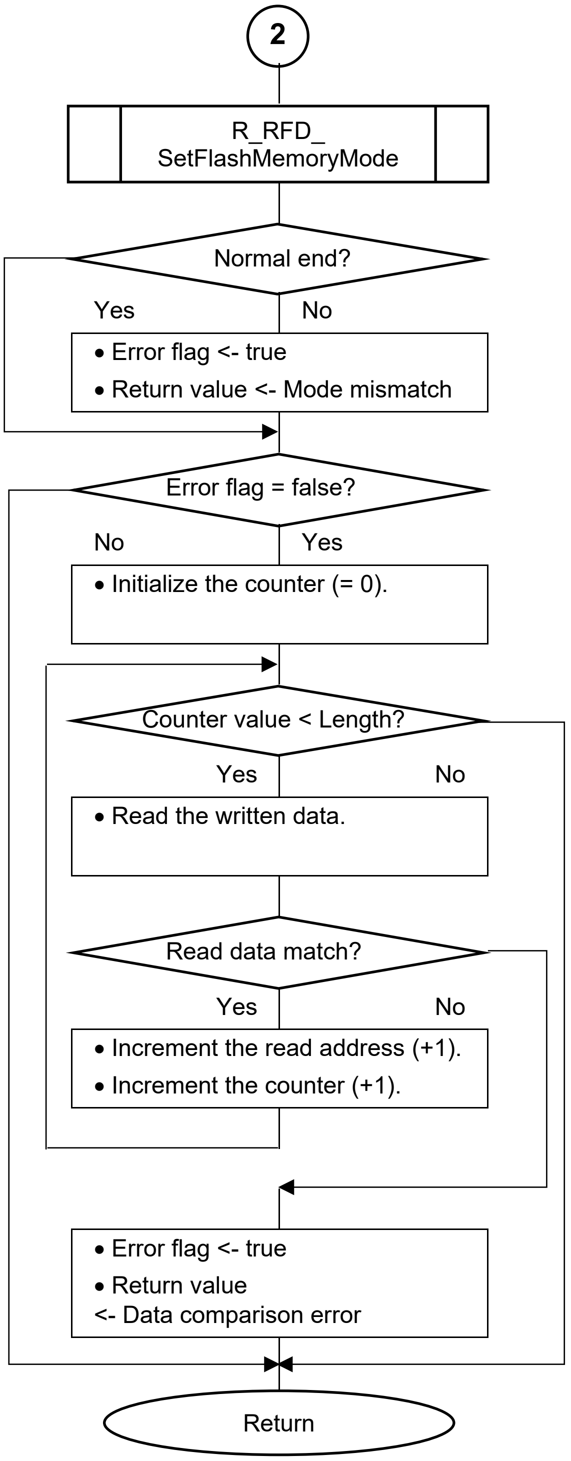
- Programming processing

- Processing for controlling the code flash memory reprogramming

- Internal verify processing

**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 02 erases block 0 (000F1000H) in the data flash area and writes 64-byte data from the beginning of the block.

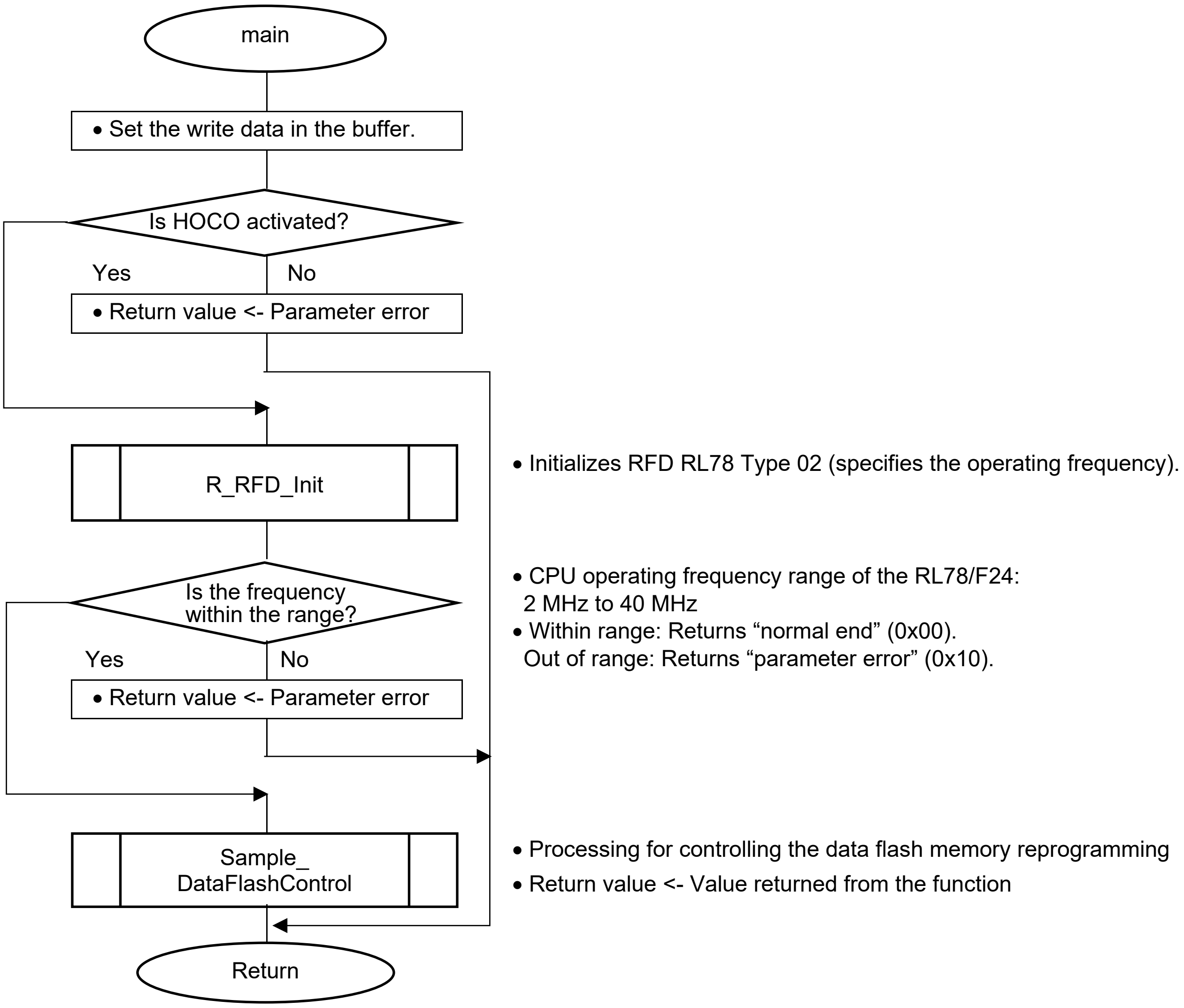
**Note: In the data flash memory programming mode, the data in the data flash memory cannot be referenced. Copy the Sample_DataFlashControl function and the data to be referenced within the function to RAM in advance, and reference them in RAM.**

**Operating conditions (Example of a sample program for RL78/F24):**

- CPU operating frequency: 40 MHz (The high-speed on-chip oscillator clock is used for the main system clock.)
- Data flash memory address for erasure and programming: 000F1000H
- Block number for erasure: 0000H
- 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 02.

#### 5.3.2.1 main Function


main
• Set the write data in the buffer.
Is HOCO activated?
Yes
No
• Return value <- Parameter error
R_RFD_Init
Is the frequency within the range?
Yes
No
• Return value <- Parameter error
Sample_ DataFlashControl
Return

- Initializes RFD RL78 Type 02 (specifies the operating frequency).

- CPU operating frequency range of the RL78/F24:
  2 MHz to 40 MHz
- Within range: Returns “normal end” (0x00).
  Out of range: Returns “parameter error” (0x10).

- Processing for controlling the data flash memory reprogramming
- Return value <- Value returned from the 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.

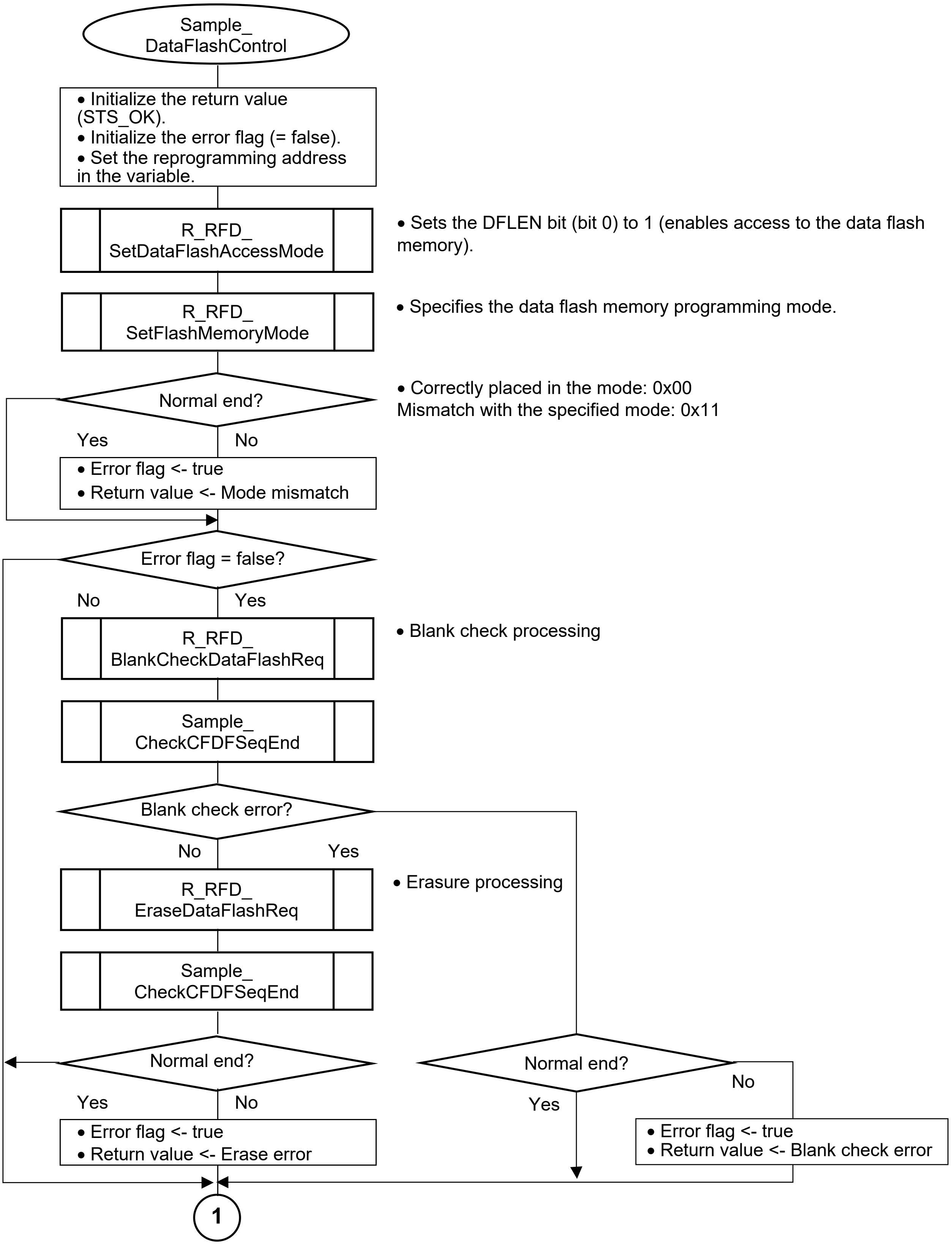

Sample_
DataFlashControl
• Initialize the return value (STS_OK).
• Initialize the error flag (= false).
• Set the reprogramming address in the variable.
R_RFD_
SetDataFlashAccessMode
• Sets the DFLEN bit (bit 0) to 1 (enables access to the data flash memory).
R_RFD_
SetFlashMemoryMode
• Specifies the data flash memory programming mode.
Normal end?
• Correctly placed in the mode: 0x00
Mismatch with the specified mode: 0x11
Yes
No
• Error flag <- true
• Return value <- Mode mismatch
Error flag = false?
No
Yes
R_RFD_
BlankCheckDataFlashReq
• Blank check processing
Sample_
CheckCFDFSeqEnd
Blank check error?
No
Yes
R_RFD_
EraseDataFlashReq
• Erasure processing
Sample_
CheckCFDFSeqEnd
Normal end?
Yes
No
• Error flag <- true
• Return value <- Erase error
Normal end?
No
Yes
• Error flag <- true
• Return value <- Blank check error
1

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

- Programming, and internal verify are executed.

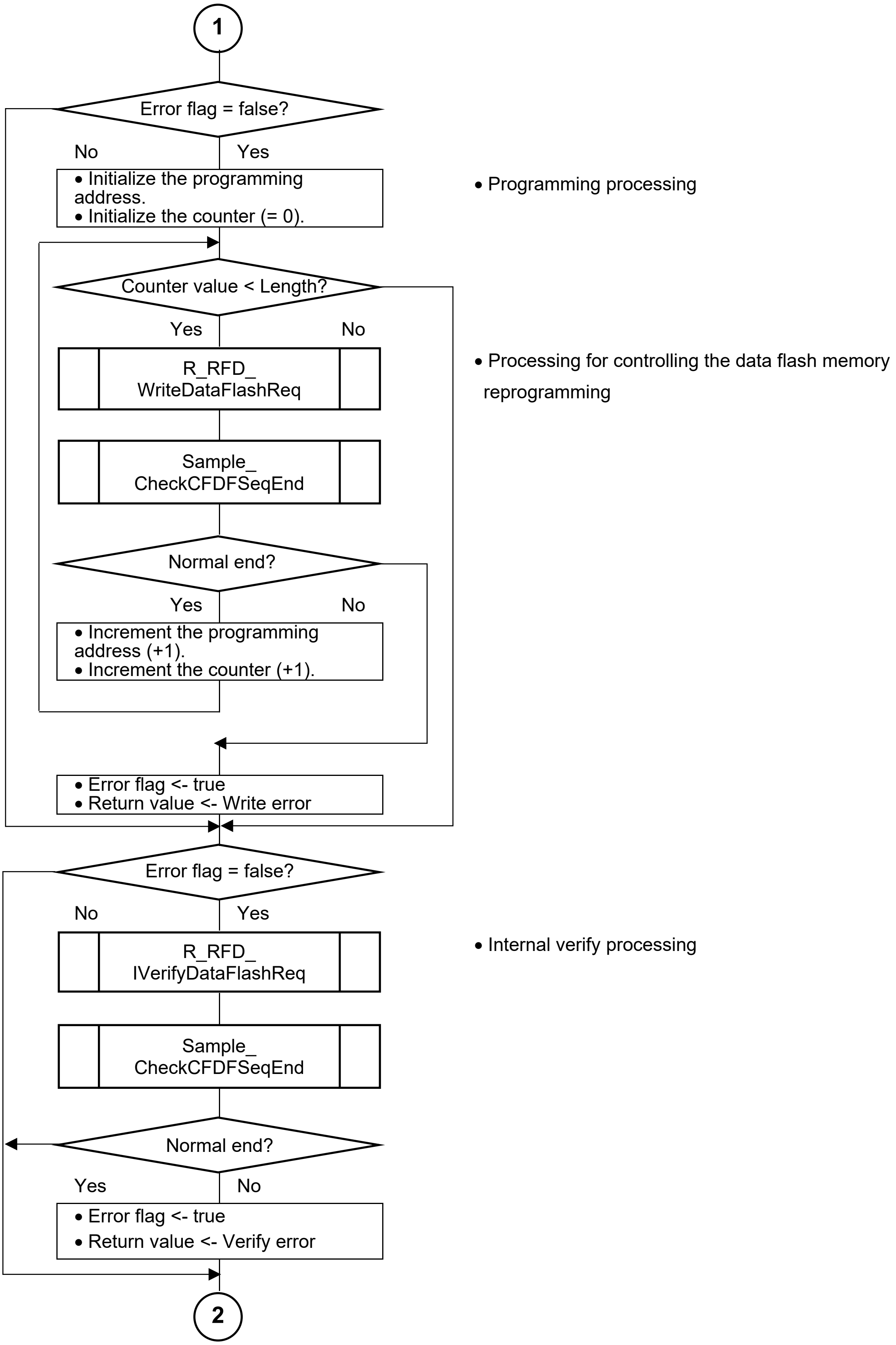
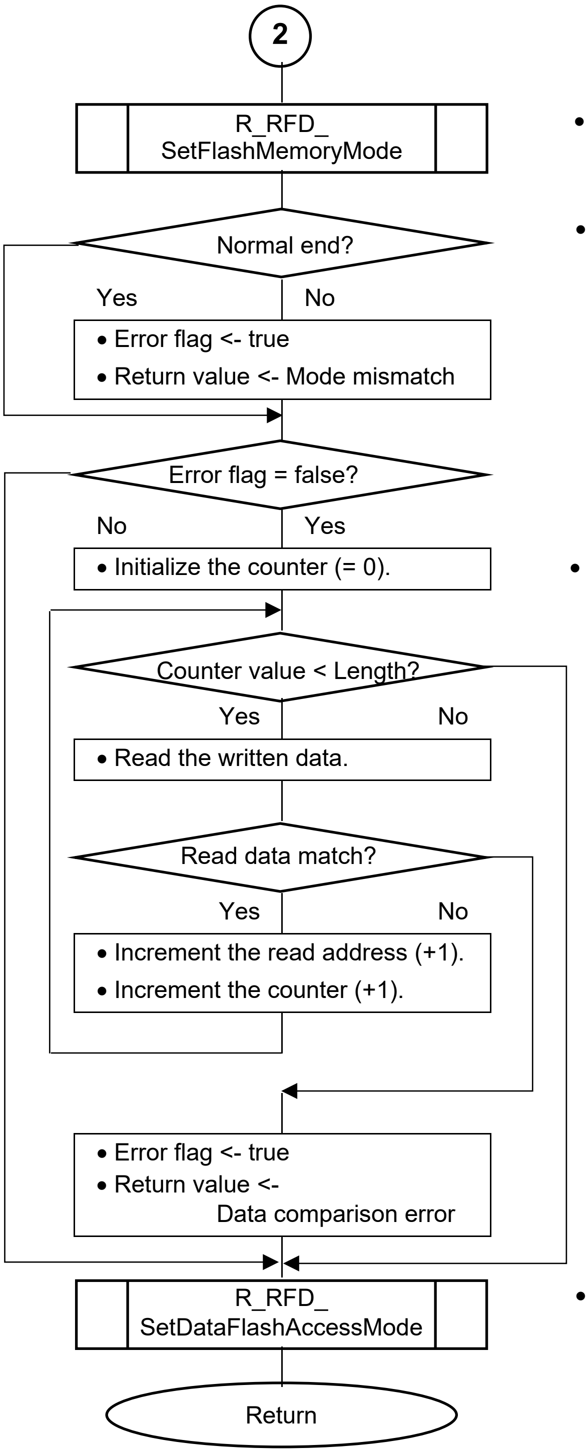

1
Error flag = false?
No
Yes
• Initialize the programming address.
• Initialize the counter (= 0).
• Programming processing
Counter value < Length?
Yes
No
R_RFD_ WriteDataFlashReq
• Processing for controlling the data flash memory reprogramming
Sample_ CheckCFDFSeqEnd
Normal end?
Yes
No
• Increment the programming address (+1).
• Increment the counter (+1).
• Error flag <- true
• Return value <- Write error
Error flag = false?
No
Yes
R_RFD_ IVerifyDataFlashReq
• Internal verify processing
Sample_ CheckCFDFSeqEnd
Normal end?
Yes
No
• Error flag <- true
• Return value <- Verify error
2

**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.



- Specifies the non-programmable mode.
- Correctly placed in the mode: 0x00
  Mismatch with the specified mode: 0x11
- Verification check through reading by the CPU
- Sets the DFLEN bit (bit 0) to 0 (disables access to the data flash memory).

**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 02 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.)

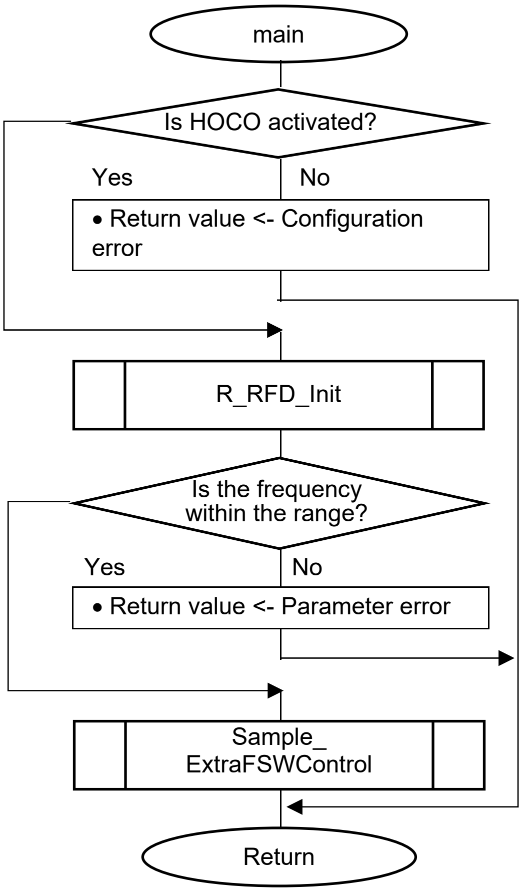
**Note: In the code flash memory programming mode for reprograming the extra area, the programs in the code flash memory cannot be executed. Copy the Sample_ExtraFSWControl function and the processing to be executed and data to be referenced within the function to RAM in advance, and execute and reference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

**Operating conditions (Example of a sample program for RL78/F24):**

- CPU operating frequency: 40 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 02.

#### 5.3.3.1 main Function


main
Is HOCO activated?
Yes
No
• Return value <- Configuration error
R_RFD_Init
Is the frequency within the range?
Yes
No
• Return value <- Parameter error
Sample_ExtraFSWControl
Return

• Initializes RFD RL78 Type 02 (specifies the operating frequency).

• CPU operating frequency range of the RL78/F24:
2 MHz to 40 MHz
• Within range: Returns “normal end” (0x00).
Out of range: Returns “parameter error” (0x10).

• Processing for controlling the extra area reprogramming
• Return value <- Value returned from the 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.

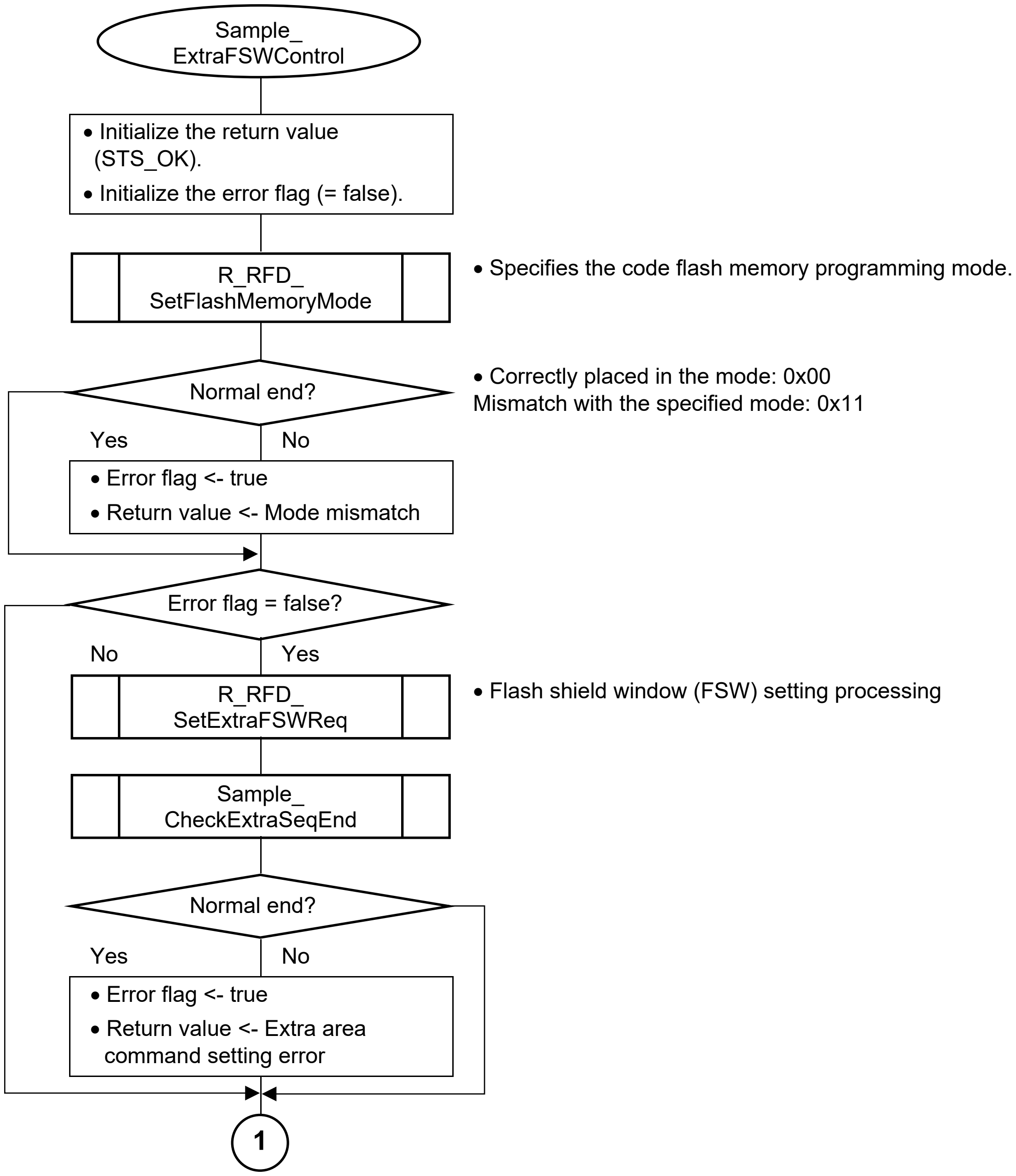
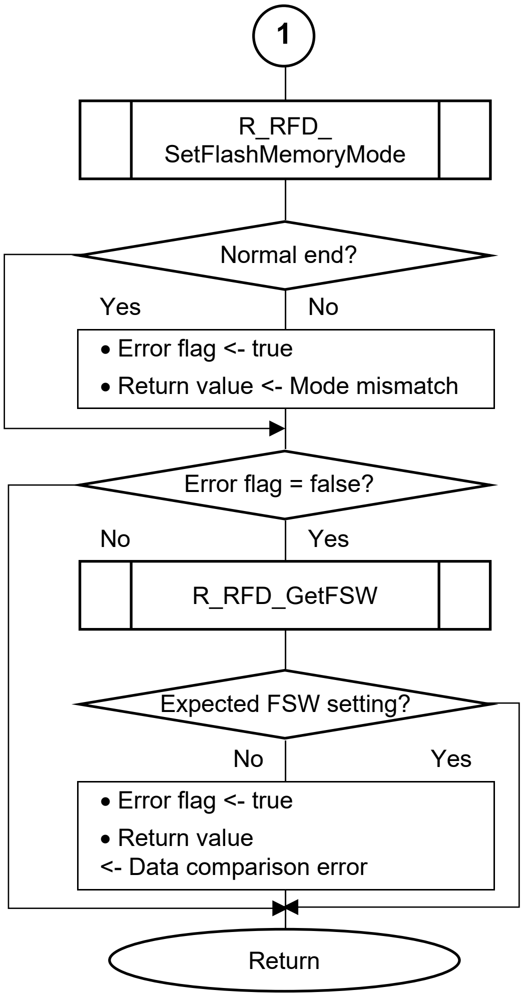

Sample_
ExtraFSWControl
• Initialize the return value (STS_OK).
• Initialize the error flag (= false).
R_RFD_
SetFlashMemoryMode
• Specifies the code flash memory programming mode.
Normal end?
• Correctly placed in the mode: 0x00
Mismatch with the specified mode: 0x11
Yes
No
• Error flag <- true
• Return value <- Mode mismatch
Error flag = false?
No
Yes
R_RFD_
SetExtraFSWReq
• Flash shield window (FSW) setting processing
Sample_
CheckExtraSeqEnd
Normal end?
Yes
No
• Error flag <- true
• Return value <- Extra area command setting error
1

**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 and end block +1 of the FSW.

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

### 5.3.4 Sample Program Used in Common for Controlling the Flash Memory

#### 5.3.4.1 Sample_CheckCFDFSeqEnd Function

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

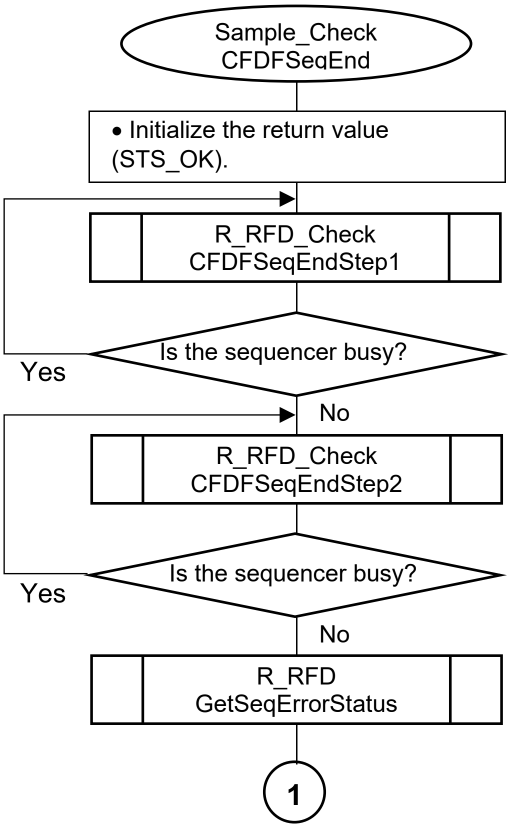


**Figure 5-13 Flowchart of Sample_CheckCFDFSeqEnd Function (1/2)**

- Returns the execution result.

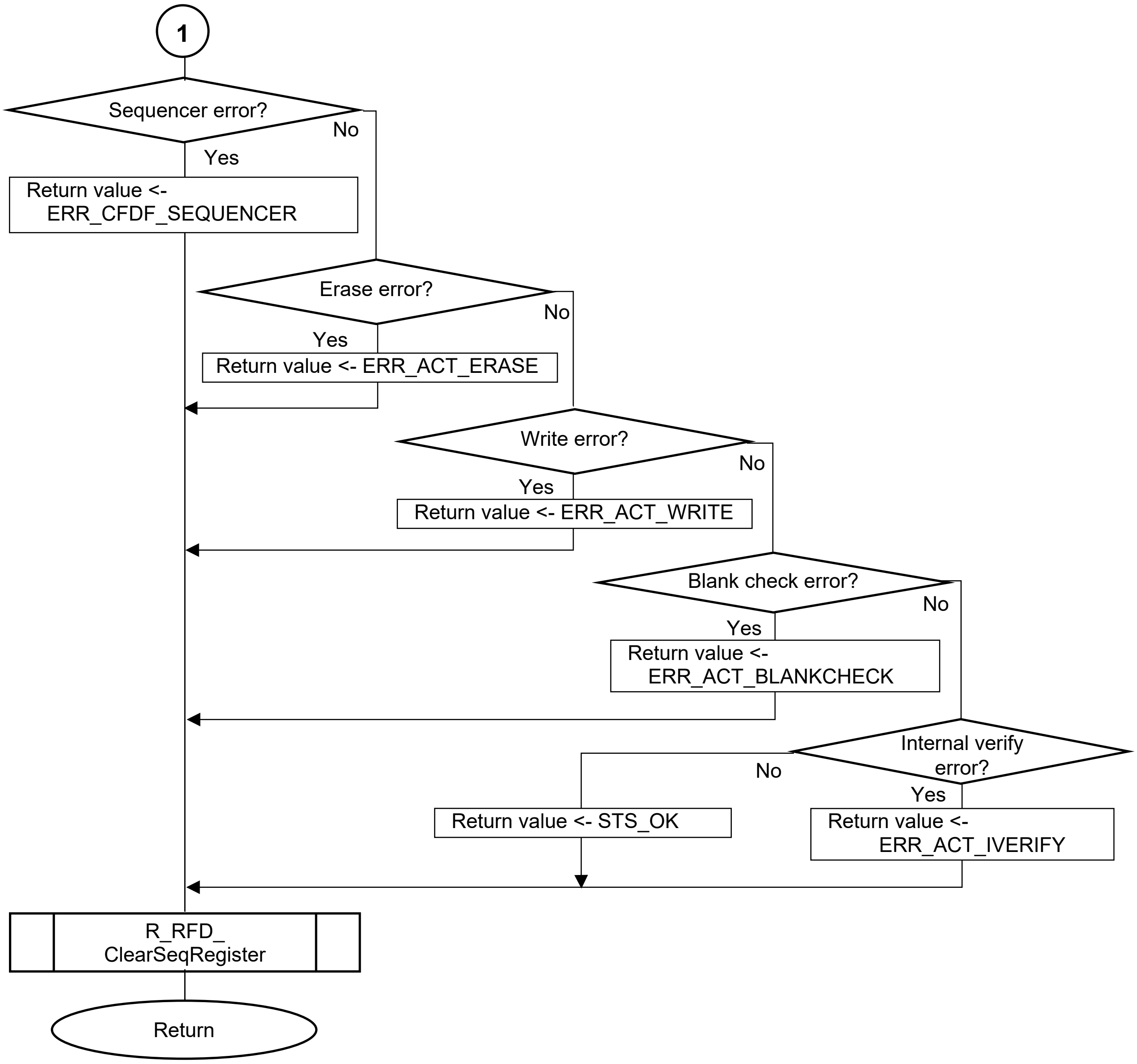

1
Sequencer error?
No
Yes
Return value <- ERR_CFDF_SEQUENCER
Erase error?
No
Yes
Return value <- ERR_ACT_ERASE
Write error?
No
Yes
Return value <- ERR_ACT_WRITE
Blank check error?
No
Yes
Return value <- ERR_ACT_BLANKCHECK
Internal verify error?
No
Yes
Return value <- STS_OK
Return value <- ERR_ACT_IVERIFY
R_RFD_ ClearSeqRegister
Return

**Figure 5-14 Flowchart of Sample_CheckCFDFSeqEnd Function (2/2)**

### 5.3.4.2 Sample_CheckExtraSeqEnd Function

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

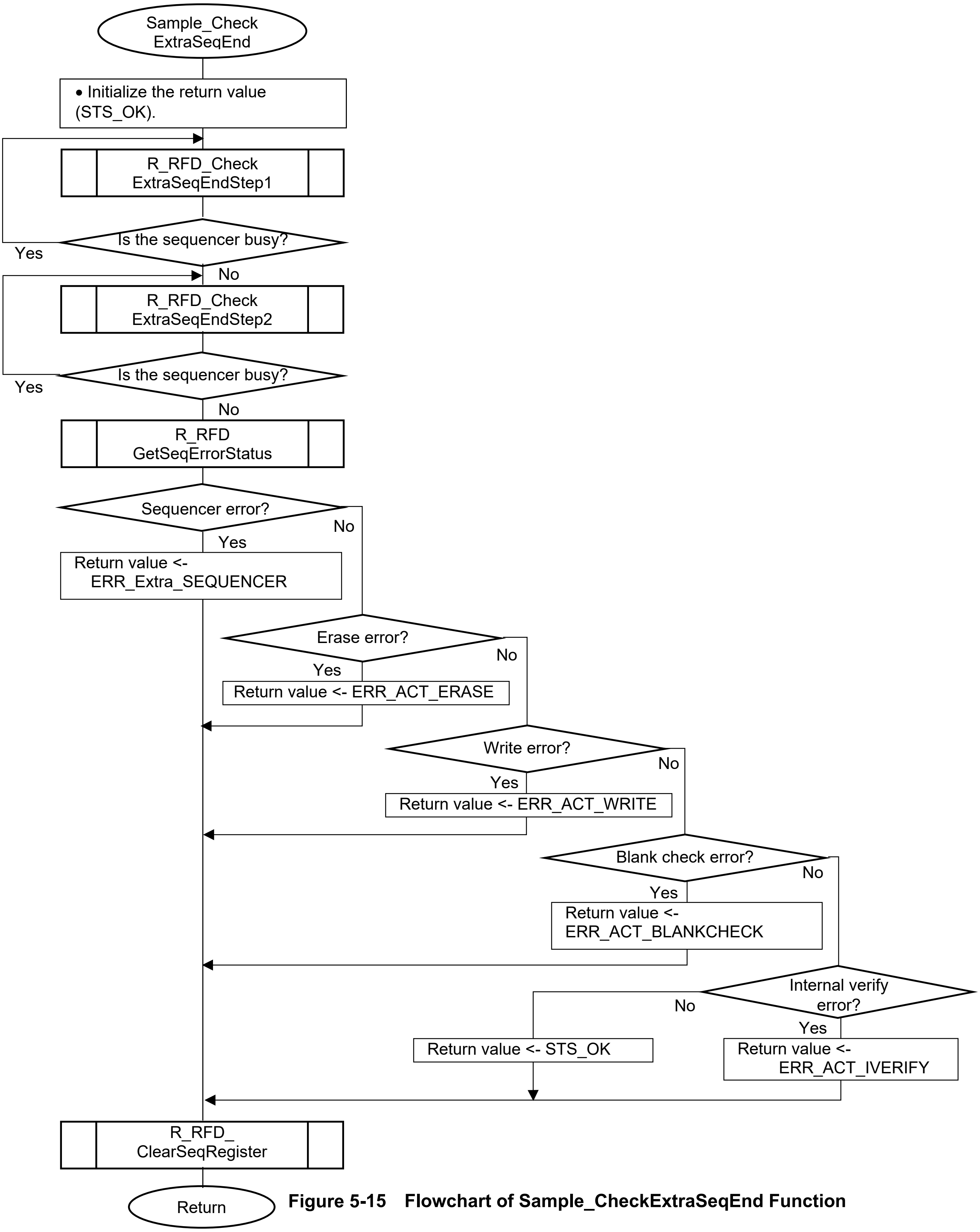


**Figure 5-15 Flowchart of Sample_CheckExtraSeqEnd Function**

## 5.4 Specifications of Sample Program Functions

This section describes the specifications of the functions in the sample programs for RFD RL78 Type 02.

The sample programs for RFD RL78 Type 02 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<br>(e_sample_ret_t) | SAMPLE_ENUM_RET_STS_OK: 0x00<br>[Normal end]<br>SAMPLE_ENUM_RET_ERR_PARAMETER: 0x10<br>[Parameter error]<br>SAMPLE_ENUM_RET_ERR_CONFIGURATION: 0x11<br>[Configuration error]<br>SAMPLE_ENUM_RET_ERR_MODE_MISMATCHED: 0x12<br>[Mode mismatch error]<br>SAMPLE_ENUM_RET_ERR_CHECK_WRITE_DATA: 0x13<br>[Written data comparison error]<br>SAMPLE_ENUM_RET_ERR_CMD_ERASE: 0x30<br>[Erase command error]<br>SAMPLE_ENUM_RET_ERR_CMD_WRITE: 0x31<br>[Write command error]<br>SAMPLE_ENUM_RET_ERR_CMD_BLANKCHECK: 0x32<br>[Blank check command error]<br>SAMPLE_ENUM_RET_ERR_CMD_IVERIFY: 0x33<br>[Internal verify 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<br>(uint32_t i_u32_start_addrr,<br>uint16_t i_u16_write_data_length,<br>uint8_t __near * inp_u08_write_data); | |
| Reentrancy | Non-reentrant | |
| Parameters (IN) | uint32_t<br>i_u32_start_addr | Start address of the area to be reprogrammed |
| | uint16_t<br>i_u16_write_data_length | Size of the reprogram data |
| | uint8_t __near *<br>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<br>[Normal end]<br>SAMPLE_ENUM_RET_ERR_MODE_MISMATCHED: 0x12<br>[Mode mismatch error]<br>SAMPLE_ENUM_RET_ERR_CHECK_WRITE_DATA: 0x13<br>[Written data comparison error]<br>SAMPLE_ENUM_RET_ERR_CMD_ERASE: 0x30<br>[Erase command error]<br>SAMPLE_ENUM_RET_ERR_CMD_WRITE: 0x31<br>[Write command error]<br>SAMPLE_ENUM_RET_ERR_CMD_BLANKCHECK: 0x32<br>[Blank check command error]<br>SAMPLE_ENUM_RET_ERR_CMD_IVERIFY: 0x33<br>[Internal verify command error] |
| Description | Executes the processing for reprogramming the code flash memory.<br>- The blank check, erase, write, and internal verify commands are executed in the code flash memory programming mode.<br>- 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<br>(e_sample_ret_t) | SAMPLE_ENUM_RET_STS_OK: 0x00<br>[Normal end]<br>SAMPLE_ENUM_RET_ERR_PARAMETER: 0x10<br>[Parameter error]<br>SAMPLE_ENUM_RET_ERR_CONFIGURATION: 0x11<br>[Configuration error]<br>SAMPLE_ENUM_RET_ERR_MODE_MISMATCHED: 0x12<br>[Mode mismatch error]<br>SAMPLE_ENUM_RET_ERR_CHECK_WRITE_DATA: 0x13<br>[Written data comparison error]<br>SAMPLE_ENUM_RET_ERR_CMD_ERASE: 0x30<br>[Erase command error]<br>SAMPLE_ENUM_RET_ERR_CMD_WRITE: 0x31<br>[Write command error]<br>SAMPLE_ENUM_RET_ERR_CMD_BLANKCHECK: 0x32<br>[Blank check command error]<br>SAMPLE_ENUM_RET_ERR_CMD_IVERIFY: 0x33<br>[Internal verify 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<br>(uint32_t i_u32_start_addrr,<br>uint16_t i_u16_write_data_length,<br>uint8_t __near * inp_u08_write_data); | |
| Reentrancy | Non-reentrant | |
| Parameters (IN) | uint32_t<br>i_u32_start_addr | Start address of the area to be reprogrammed |
| | uint16_t<br>i_u16_write_data_length | Size of the reprogram data |
| | uint8_t __near *<br>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<br>[Normal end]<br>SAMPLE_ENUM_RET_ERR_MODE_MISMATCHED: 0x12<br>[Mode mismatch error]<br>SAMPLE_ENUM_RET_ERR_CHECK_WRITE_DATA: 0x13<br>[Written data comparison error]<br>SAMPLE_ENUM_RET_ERR_CMD_ERASE: 0x30<br>[Erase command error]<br>SAMPLE_ENUM_RET_ERR_CMD_WRITE: 0x31<br>[Write command error]<br>SAMPLE_ENUM_RET_ERR_CMD_BLANKCHECK: 0x32<br>[Blank check command error]<br>SAMPLE_ENUM_RET_ERR_CMD_IVERIFY: 0x33<br>[Internal verify command error] |
| Description | Executes the processing for reprogramming the data flash memory.<br>- The blank check, erase, write, and internal verify commands are executed in the data flash memory programming mode.<br>- 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.<br>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<br>(e_sample_ret_t) | SAMPLE_ENUM_RET_STS_OK: 0x00<br>[Normal end]<br>SAMPLE_ENUM_RET_ERR_PARAMETER: 0x10<br>[Parameter error]<br>SAMPLE_ENUM_RET_ERR_CONFIGURATION: 0x11<br>[Configuration error]<br>SAMPLE_ENUM_RET_ERR_MODE_MISMATCHED: 0x12<br>[Mode mismatch error]<br>SAMPLE_ENUM_RET_ERR_CHECK_WRITE_DATA: 0x13<br>[Written data comparison error]<br>SAMPLE_ENUM_RET_ERR_CMD_SET_EXTRA_AREA: 0x34<br>[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<br>(uint16_t i_u16_start_block_number,<br>uint16_t i_u16_end_block_number); | |
| Reentrancy | Non-reentrant | |
| Parameters (IN) | uint16_t<br>i_u16_start_block_numberr | Start block number<br>Example: For RL78/F24, 0 to 255 (256 Kbytes max.) |
| | uint16_t<br>i_u16_end_block_number | End block number +1<br>Example: For RL78/F24, 1 to 256 (256 Kbytes max.) |
| Parameters (IN/OUT) | N/A | |
| Parameters (OUT) | N/A | |
| Return Value | e_sample_ret_t | SAMPLE_ENUM_RET_STS_OK: 0x00<br>[Normal end]<br>SAMPLE_ENUM_RET_ERR_MODE_MISMATCHED: 0x12<br>[Mode mismatch error]<br>SAMPLE_ENUM_RET_ERR_CHECK_WRITE_DATA: 0x13<br>[Written data comparison error]<br>SAMPLE_ENUM_RET_ERR_CMD_SET_EXTRA_AREA: 0x34<br>[Extra area command setting error] |
| Description | Executes the processing for reprogramming the extra area (FSW function settings).<br>- The write command for the extra area (FSW-related data programming command) is executed in the code flash memory programming mode.<br>- 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 Used in Common

#### 5.4.4.1 Sample_CheckCFDFSeqEnd

**Information:**

| | | |
|---|---|---|
| Syntax | R_RFD_FAR_FUNC e_sample_ret_t Sample_CheckCFDFSeqEnd(void); | |
| Reentrancy | Non-reentrant | |
| Parameters (IN) | N/A | |
| Parameters (IN/OUT) | N/A | |
| Parameters (OUT) | N/A | |
| Return Value | e_sample_ret_t | SAMPLE_ENUM_RET_STS_OK: 0x00<br>[Normal end]<br>SAMPLE_ENUM_RET_ERR_CFDF_SEQUENCER: 0x20<br>[Code/data flash area sequencer error]<br>SAMPLE_ENUM_RET_ERR_ACT_ERASE: 0x22<br>[Erase operation error]<br>SAMPLE_ENUM_RET_ERR_ACT_WRITE: 0x23<br>[Write operation error]<br>SAMPLE_ENUM_RET_ERR_ACT_BLANKCHECK: 0x24<br>[Blank check operation error]<br>SAMPLE_ENUM_RET_ERR_ACT_IVERIFY: 0x25<br>[Internal verify operation error] |
| Description | Waits for the completion of command execution in the code/data flash area sequencer. | |
| Preconditions | Use this function in the code flash memory programming mode or data flash memory programming mode while the high-speed on-chip oscillator is active.<br>When reprogramming the data flash memory, use this function while access to the data flash memory is enabled (DFLEN = 1). | |
| Remarks | - | |

### 5.4.4.2 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<br>[Normal end]<br>SAMPLE_ENUM_RET_ERR_EXTRA_SEQUENCER: 0x21<br>[Extra area sequencer error]<br>SAMPLE_ENUM_RET_ERR_ACT_ERASE: 0x22<br>[Erase operation error]<br>SAMPLE_ENUM_RET_ERR_ACT_WRITE: 0x23<br>[Write operation error]<br>SAMPLE_ENUM_RET_ERR_ACT_BLANKCHECK: 0x24<br>[Blank check operation error]<br>SAMPLE_ENUM_RET_ERR_ACT_IVERIFY: 0x25<br>[Internal verify operation error] |
| Description | Waits for the completion of command execution in the extra area sequencer. | |
| Preconditions | Execute this function in the code flash memory programming mode while the high-speed on-chip oscillator is active.<br>Use this function while access to the data flash memory is enabled. | |
| Remarks | - | |

# 6. Creating a Sample Project for RFD RL78 Type 02

RFD RL78Type 02 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 02 are a CC-RL compiler and an IAR 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/F24(R7F124FPJ) is explained to this section. When using other than RL78/F24(R7F124FPJ), section address settings must be changed by referring to the user's manual for the target device.

If the RL78/F23 is used, the RL78/F24 sample program is available.

**Note: The target Integrated Development Environment (IDE) and the compiler are premised on using the version for RL78/F23 and RL78/F24. Be sure to use them, after confirming that RL78/F23 and RL78/F24 are target products.**

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

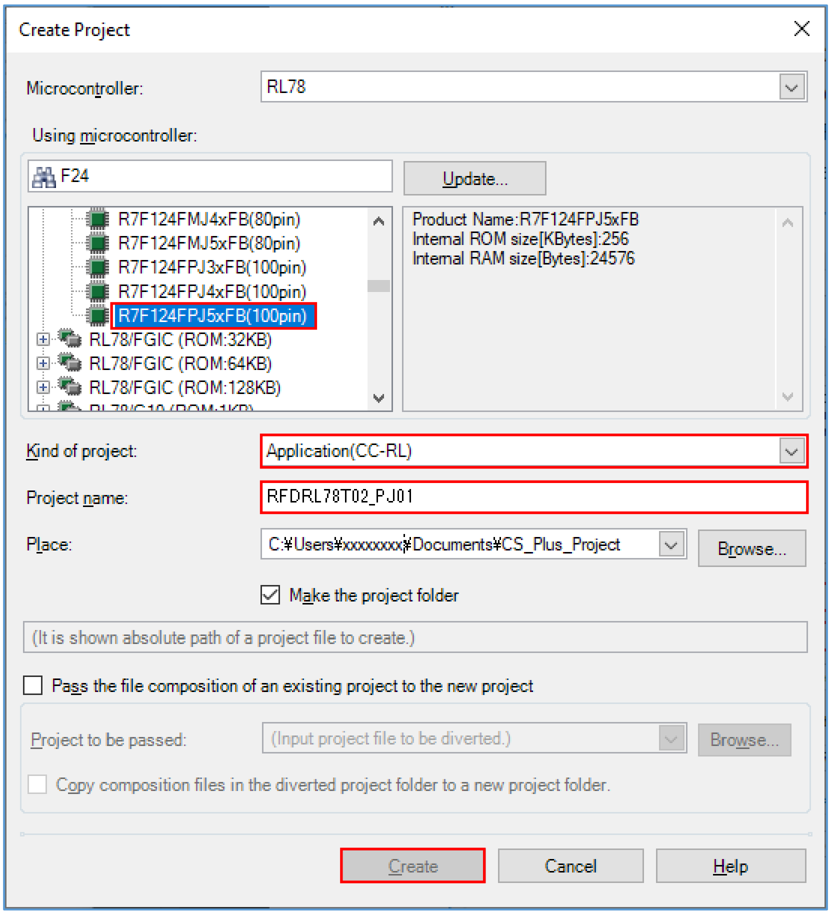
CS+ or e$^2$studio can be used for a RENESAS CC-RL compiler as an IDE. RFD RL78 Type 02 is registered and built in the project created by the IDE. An example of creating a sample project in case each IDE is used is shown. Because to understand a CC-RL compiler and each IDE, it is necessary to refer to 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/F24 (ROM: 256 Kbytes)” - “R7F124FPJ5xFB(100pin)” as [Using microcontroller].
- Select “Application (CC-RL)” as [Kind of project].
- [Project name] is temporarily set to “RFDRL78T02_PJ01”.
- When you click the [Create] button, the new project is created.



Microcontroller: RL78

Using microcontroller: F24

R7F124FMJ4xFB(80pin)
R7F124FMJ5xFB(80pin)
R7F124FPJ3xFB(100pin)
R7F124FPJ4xFB(100pin)
R7F124FPJ5xFB(100pin)
RL78/FGIC (ROM:32KB)
RL78/FGIC (ROM:64KB)
RL78/FGIC (ROM:128KB)

Product Name:R7F124FPJ5xFB
Internal ROM size[KBytes]:256
Internal RAM size[Bytes]:24576

Kind of project: Application(CC-RL)

Project name: RFDRL78T02_PJ01

Place: C:¥Users¥xxxxxxxx¥Documents¥CS_Plus_Project

☑ Make the project folder

(It is shown absolute path of a project file to create.)

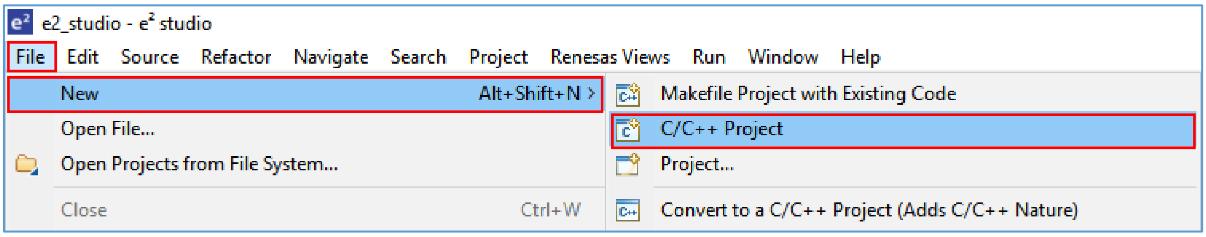
□ Pass the file composition of an existing project to the new project

Project to be passed: (Input project file to be diverted.)

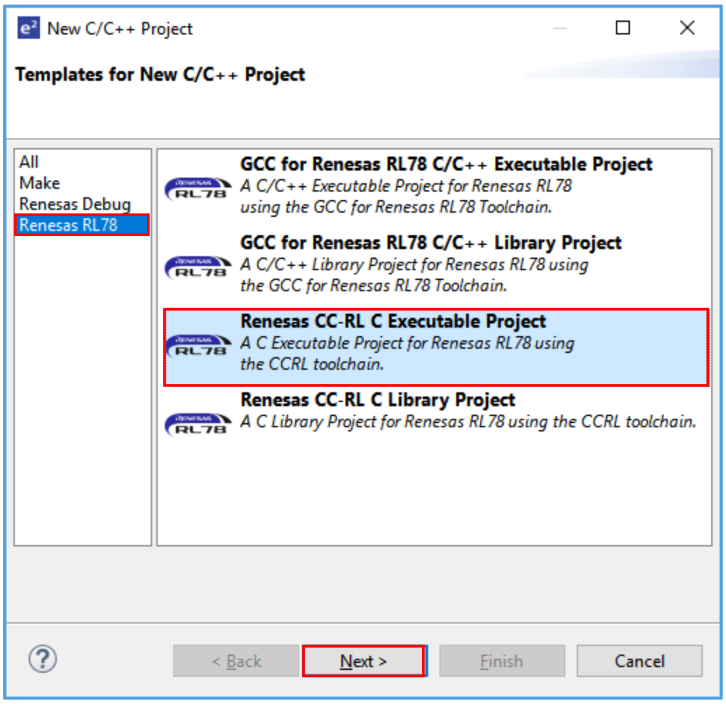
□ Copy composition files in the diverted project folder to a new project folder.

(2) An example of creating a sample project which used e$^2$studio (IDE)

- The e$^2$studio starts and from the [File] menu, select [New] – [C/C++ Project], the “Templates for New C/C++ Project” window will open.



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


New C/C++ Project

**Templates for New C/C++ Project**

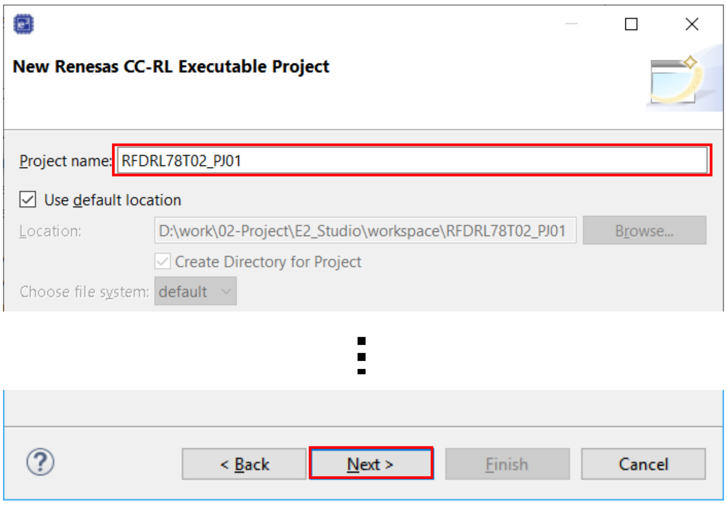
**GCC for Renesas RL78 C/C++ Executable Project**
*A C/C++ Executable Project for Renesas RL78 using the GCC for Renesas RL78 Toolchain.*

**GCC for Renesas RL78 C/C++ Library Project**
*A C/C++ Library Project for Renesas RL78 using the GCC for Renesas RL78 Toolchain.*

**Renesas CC-RL C Executable Project**
*A C Executable Project for Renesas RL78 using the CCRL toolchain.*

**Renesas CC-RL C Library Project**
*A C Library Project for Renesas RL78 using the CCRL toolchain.*

- Input “project name” on “New Renesas CC-RL Executable Project” window, and push “next” button. [Project name] is temporarily set to “RFDRL78T02_PJ01”.


**New Renesas CC-RL Executable Project**

Project name: RFDRL78T02_PJ01

☑ Use default location

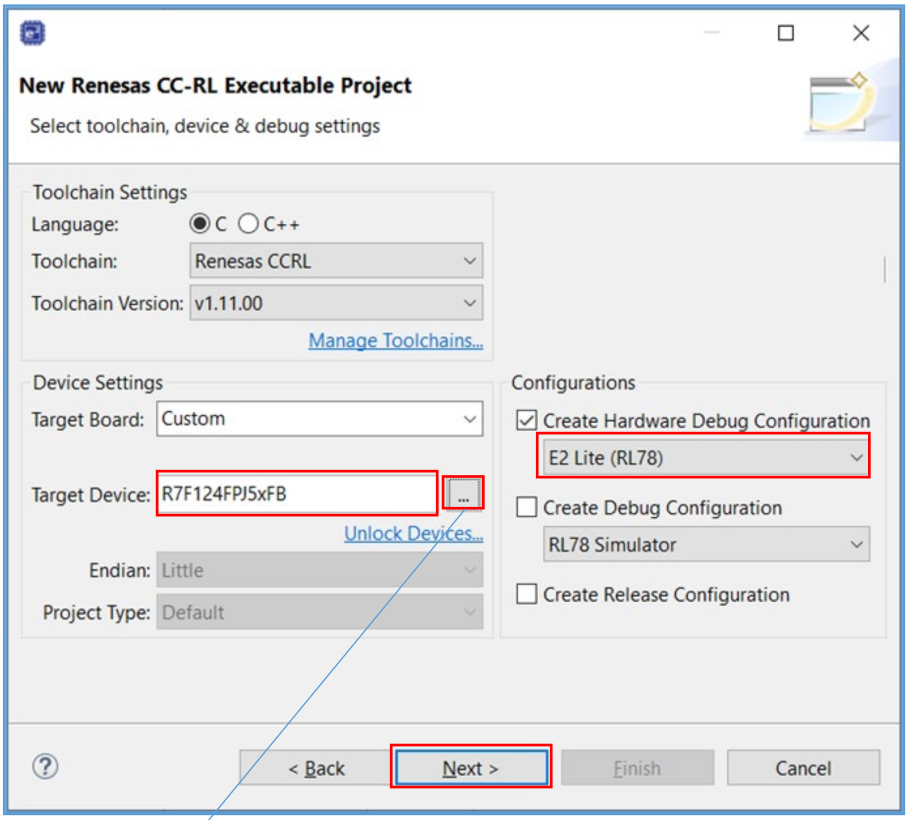
Location: D:\work\02-Project\E2_Studio\workspace\RFDRL78T02_PJ01 Browse...

☑ Create Directory for Project

Choose file system: default

⋮

- Select the [Target Device] of [Device Settings], and select “RL78 - F24” - “R7F124FPJ5xFB”.
- 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.



New Renesas CC-RL Executable Project

Select toolchain, device & debug settings

Toolchain Settings

Language: ● C ○ C++

Toolchain: Renesas CCRL

Toolchain Version: v1.11.00

Device Settings

Target Board: Custom

Target Device: R7F124FPJ5xFB

Endian: Little

Project Type: Default

Configurations

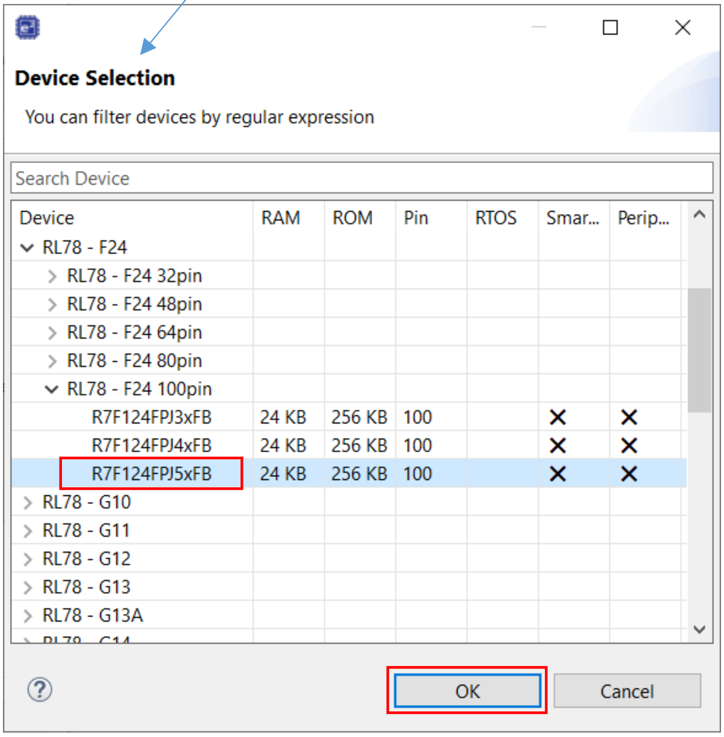
☑ Create Hardware Debug Configuration

E2 Lite (RL78)

□ Create Debug Configuration

RL78 Simulator

□ Create Release Configuration



Device Selection

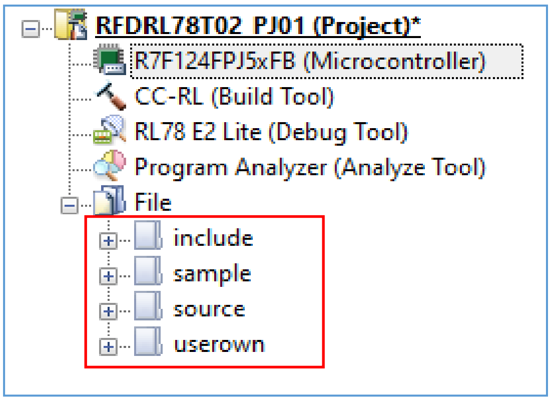
You can filter devices by regular expression

Search Device

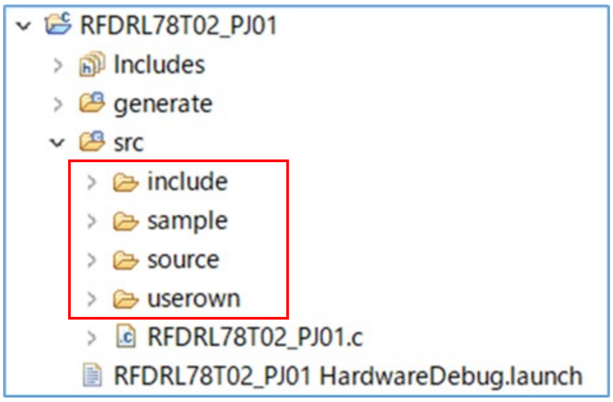
| Device | RAM | ROM | Pin | RTOS | Smar... | Perip... |
|---|---|---|---|---|---|---|
| RL78 - F24 | | | | | | |
| RL78 - F24 32pin | | | | | | |
| RL78 - F24 48pin | | | | | | |
| RL78 - F24 64pin | | | | | | |
| RL78 - F24 80pin | | | | | | |
| RL78 - F24 100pin | | | | | | |
| R7F124FPJ3xFB | 24 KB | 256 KB | 100 | | ✕ | ✕ |
| R7F124FPJ4xFB | 24 KB | 256 KB | 100 | | ✕ | ✕ |
| R7F124FPJ5xFB | 24 KB | 256 KB | 100 | | ✕ | ✕ |
| RL78 - G10 | | | | | | |
| RL78 - G11 | | | | | | |
| RL78 - G12 | | | | | | |
| RL78 - G13 | | | | | | |
| RL78 - G13A | | | | | | |

### 6.1.2 Example of Registration of Target Folders and Target Files

Using RFD RL78Type 02, when programming each area [(1) code flash memory, (2) data flash memory, (3) extra area], the example which registers necessary files is shown. Each folder of the RFD RL78Type 02 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).


RFDRL78T02_PJ01 (Project)*
R7F124FPJ5xFB (Microcontroller)
CC-RL (Build Tool)
RL78 E2 Lite (Debug Tool)
Program Analyzer (Analyze Tool)
File
include
sample
source
userown

The registration tree screen of RFD (CS+)


RFDRL78T02_PJ01
Includes
generate
src
include
sample
source
userown
RFDRL78T02_PJ01.c
RFDRL78T02_PJ01 HardwareDebug.launch

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” is an I/O header file which CS+ or e$^2$studio outputs to target products. Replacing instead of “iodefine.h” included in RFD RL78 Type 02 is recommended. Registration of target folders and target files is implemented. Then, a user replaces “iodefine.h” which IDE outputted with “iodefine.h” included in RFD RL78 Type 02.

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

The folder with which a user replaces the “iodefine.h” file:
- The case of code flash programming: “\[Project name]\sample\RL78_F24\CF\CCRL\include”
- The case of data flash programming: “\[Project name]\sample\RL78_F24\DF\CCRL\include”
- The case of extra area (FSW) programming:
“\[Project name]\sample\RL78_F24\EX_FSW\CCRL\include”

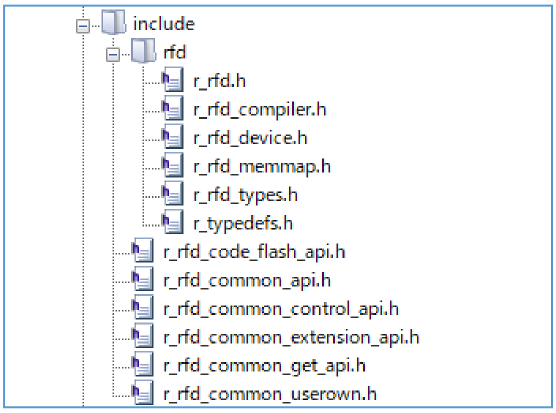
・Exclusion of the file automatically added by the function of IDE.
There are files added automatically in the created project. The same file as these exists also in the “sample” folder of RFD RL78 Type 02. Therefore, using the function of IDE, select those files from tree and excludes from a project.
- CS+: Click the right mouse button for the file of tree. And exclude target file using “Remove from Project” function. Targets are “cstart.asm, f24opt.asm, hdwinit.asm, stkinit.asm, main.c, and iodefine.h” in [project name] folder.

- e$^2$studio: Clicks the right mouse button for the file of tree. And on the [Settings] screen displayed by the “property”, put a check mark to [Exclude resource from build] and exclude a target file (target folder). (Exclusion of a folder is also possible)
  Target files are cstart.asm, f24opt.asm, hdwinit.asm, iodefine.h, and stkinit.asm in a [project name] / generate folder. And [project name] .c (“RFDRL78T02_PJ01.c”) in a [project name] / src folder is a target.

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

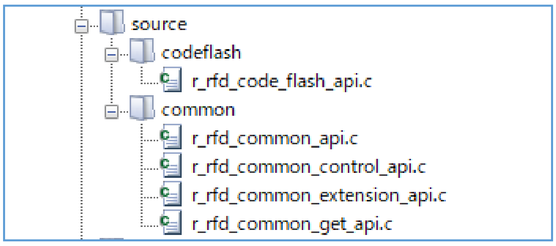
The folders (“include”, “source”, “userown”, “sample”) and source program file which are included in RFD RL78 Type 02 to register are shown below.

in the “include” folder



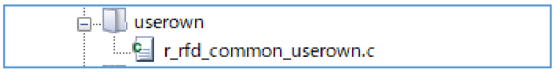
```
include
  rfd
    r_rfd.h
    r_rfd_compiler.h
    r_rfd_device.h
    r_rfd_memmap.h
    r_rfd_types.h
    r_typedefs.h
  r_rfd_code_flash_api.h
  r_rfd_common_api.h
  r_rfd_common_control_api.h
  r_rfd_common_extension_api.h
  r_rfd_common_get_api.h
  r_rfd_common_userown.h
```

in the “source” folder



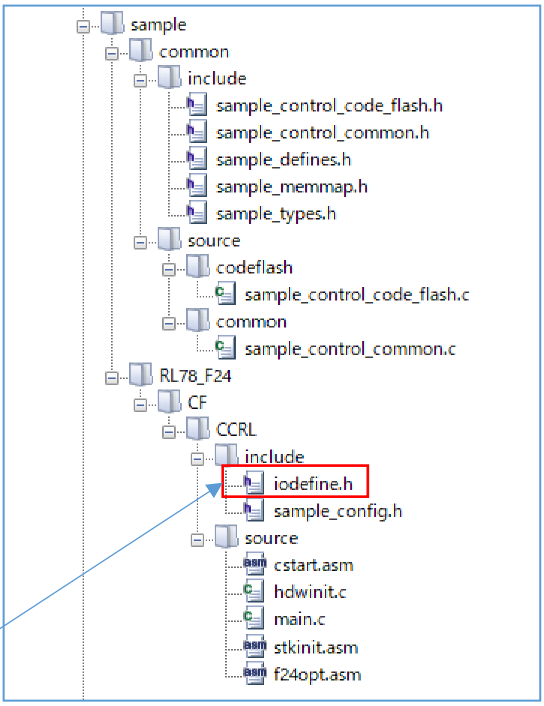
```
source
  codeflash
    r_rfd_code_flash_api.c
  common
    r_rfd_common_api.c
    r_rfd_common_control_api.c
    r_rfd_common_extension_api.c
    r_rfd_common_get_api.c
```

in the “userown” folder



```
userown
  r_rfd_common_userown.c
```

in the “sample” folder



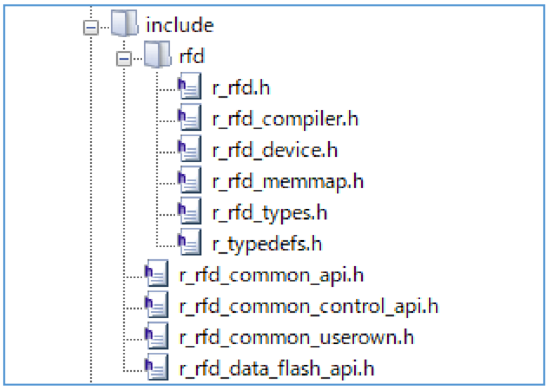
```
sample
  common
    include
      sample_control_code_flash.h
      sample_control_common.h
      sample_defines.h
      sample_memmap.h
      sample_types.h
    source
      codeflash
        sample_control_code_flash.c
      common
        sample_control_common.c
  RL78_F24
    CF
      CCRL
        include
          iodefine.h
          sample_config.h
        source
          cstart.asm
          hdwinit.c
          main.c
          stkinit.asm
          f24opt.asm
```

Transpose to “iodefine.h” outputted by CS+ or e$^2$studio.

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

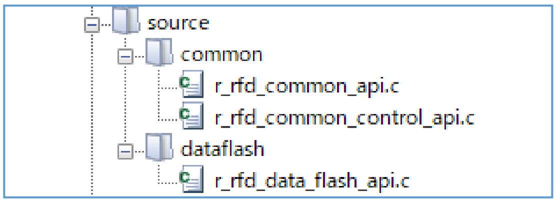
The folders (“include”, “source”, “userown”, “sample”) and source program file which are included in RFD RL78 Type 02 to register are shown below.

in the “include” folder



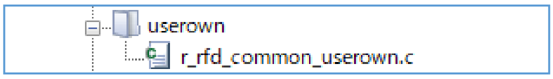
```
include
  rfd
    r_rfd.h
    r_rfd_compiler.h
    r_rfd_device.h
    r_rfd_memmap.h
    r_rfd_types.h
    r_typedefs.h
  r_rfd_common_api.h
  r_rfd_common_control_api.h
  r_rfd_common_userown.h
  r_rfd_data_flash_api.h
```

in the “source” folder



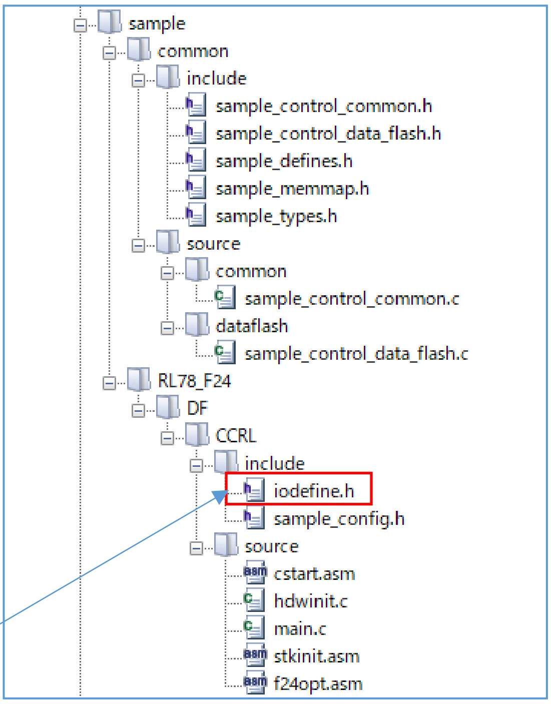
```
source
  common
    r_rfd_common_api.c
    r_rfd_common_control_api.c
  dataflash
    r_rfd_data_flash_api.c
```

in the “userown” folder



```
userown
  r_rfd_common_userown.c
```

in the “sample” folder



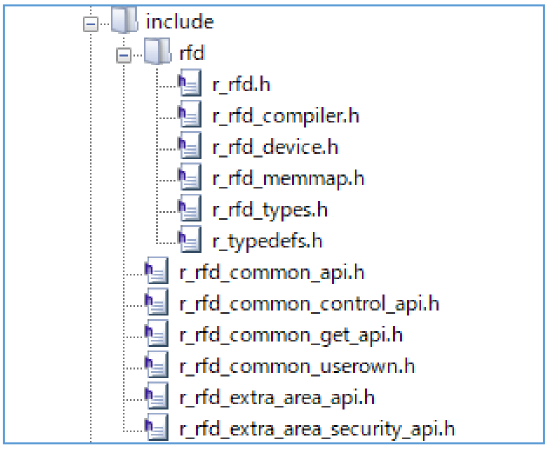
```
sample
  common
    include
      sample_control_common.h
      sample_control_data_flash.h
      sample_defines.h
      sample_memmap.h
      sample_types.h
    source
      common
        sample_control_common.c
      dataflash
        sample_control_data_flash.c
  RL78_F24
    DF
      CCRL
        include
          iodefine.h
          sample_config.h
        source
          cstart.asm
          hdwinit.c
          main.c
          stkinit.asm
          f24opt.asm
```

Transpose to “iodefine.h” outputted by CS+ or e$^2$studio.

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

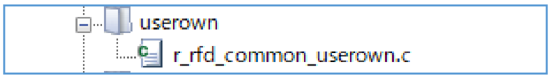
The folders (“include”, “source”, “userown”, “sample”) and source program file which are included in RFD RL78 Type 02 to register are shown below.

in the “include” folder



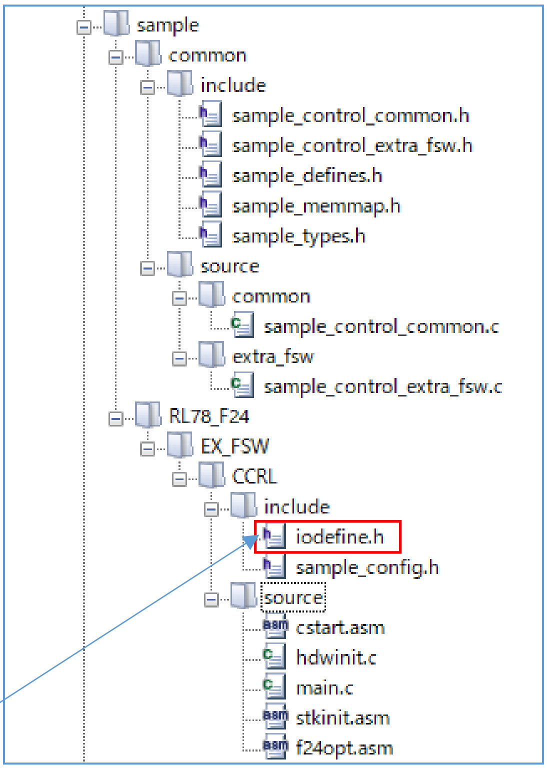
```
include
    rfd
        r_rfd.h
        r_rfd_compiler.h
        r_rfd_device.h
        r_rfd_memmap.h
        r_rfd_types.h
        r_typedefs.h
    r_rfd_common_api.h
    r_rfd_common_control_api.h
    r_rfd_common_get_api.h
    r_rfd_common_userown.h
    r_rfd_extra_area_api.h
    r_rfd_extra_area_security_api.h
```

in the “userown” folder



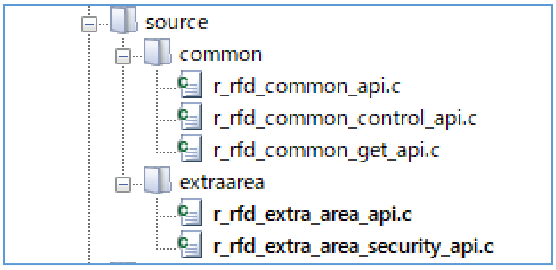
```
userown
    r_rfd_common_userown.c
```

in the “sample” folder



```
sample
    common
        include
            sample_control_common.h
            sample_control_extra_fsw.h
            sample_defines.h
            sample_memmap.h
            sample_types.h
        source
            common
                sample_control_common.c
            extra_fsw
                sample_control_extra_fsw.c
    RL78_F24
        EX_FSW
            CCRL
                include
                    iodefine.h
                    sample_config.h
                source
                    cstart.asm
                    hdwinit.c
                    main.c
                    stkinit.asm
                    f24opt.asm
```

in the “source” folder



```
source
    common
        r_rfd_common_api.c
        r_rfd_common_control_api.c
        r_rfd_common_get_api.c
    extraarea
        r_rfd_extra_area_api.c
        r_rfd_extra_area_security_api.c
```

Transpose to “iodefine.h” outputted by CS+ or $e^2$studio.

### 6.1.3 Build Tool Settings

Set IDE setting necessary in order to build RFD RL78 Type 02 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 (“RFDRL78T02_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

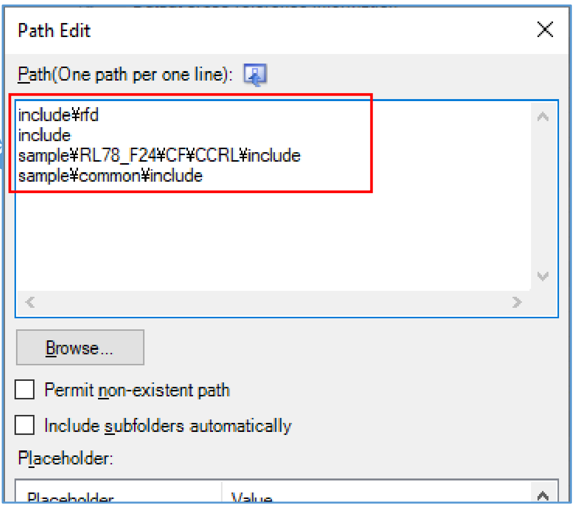
```
include\rfd
include
sample\RL78_F24\CF\CCRL\include
sample\common\include
```

(2) Data flash memory reprogramming

```
include\rfd
include
sample\RL78_F24\DF\CCRL\include
sample\common\include
```

(3) Extra area (FSW) reprogramming

```
include\rfd
include
sample\RL78_F24\EX_FSW\CCRL\include
sample\common\include
```



Path Edit
Path(One path per one line):
Browse...
□ Permit non-existent path
□ Include subfolders automatically
Placeholder:
Placeholder
Value

```
include¥rfd
include
sample¥RL78_F24¥CF¥CCRL¥include
sample¥common¥include
```

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

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

(1) Code flash memory reprogramming

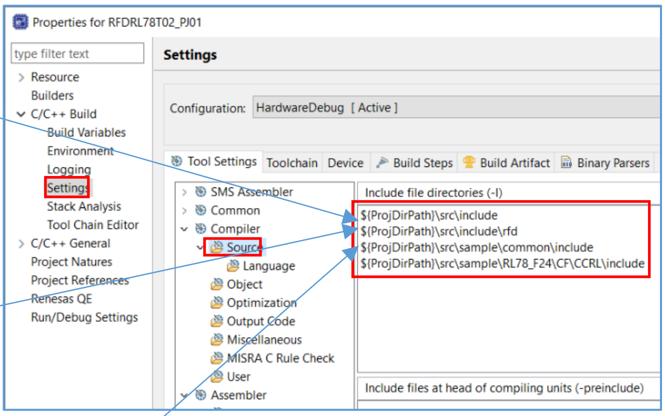
```
${ProjDirPath}\src\include\rfd
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_F24\CF\CCRL\include
${ProjDirPath}\src\sample\common\include
```

(2) Data flash memory reprogramming

```
${ProjDirPath}\src\include\rfd
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_F24\DF\CCRL\include
${ProjDirPath}\src\sample\common\include
```

(3) Extra area (FSW) reprogramming

```
${ProjDirPath}\src\include\rfd
${ProjDirPath}\src\include
${ProjDirPath}\src\sample\RL78_F24\EX_FSW\CCRL\include
${ProjDirPath}\src\sample\common\include
```



Configuration: HardwareDebug [ Active ]

```
${ProjDirPath}\src\include
${ProjDirPath}\src\include\rfd
${ProjDirPath}\src\sample\common\include
${ProjDirPath}\src\sample\RL78_F24\CF\CCRL\include
```

### 6.1.3.2 Device Item Settings

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

- Setting the [Device] items

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

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

Input the “**A5**” into [Option byte values for OCD]. [Example of permission of operation for on-chip debugging.]

Select “Yes (-SECURITY_OPT_BYTE)” in [Set security option byte].
Input the “**FE**” into [Security option byte value]. [Example of enables read of on-chip debug and flash serial programming security ID.] [The example for RL78/F24]

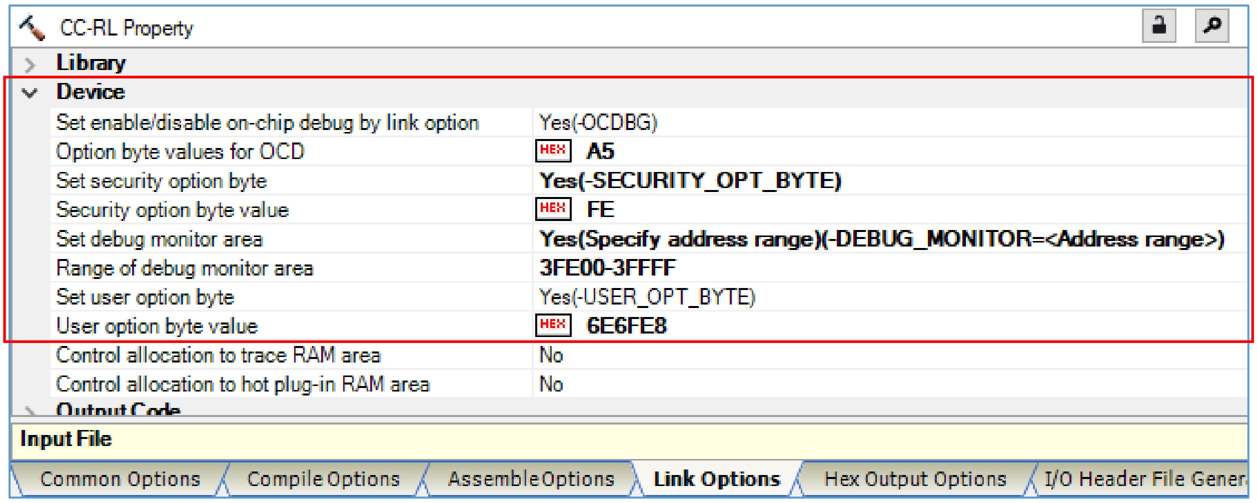
**Note: Be sure to confirm the contents of “On-Chip Debug Option Byte” and “Security Option Byte” in “Option Byte” chapter on the user's manual of a target device. And describe the set value used with user application.**

Select “Yes(Specify address range)(-OCDBG_MONITOR=<Address range>)” in [Set debug monitor area]. Set “**3FE00-3FFFF**” to [Range of debug monitor area]. [The example for RL78/F24]

**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 “**6E6FE8**” to [User option byte value]. (WDT stop, LVD [reset mode], 40MHz [The example for RL78/F24])

**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.**



CC-RL Property

Library

Device

| | |
|---|---|
| Set enable/disable on-chip debug by link option | Yes(-OCDBG) |
| Option byte values for OCD | HEX **A5** |
| Set security option byte | **Yes(-SECURITY_OPT_BYTE)** |
| Security option byte value | HEX **FE** |
| Set debug monitor area | **Yes(Specify address range)(-DEBUG_MONITOR=<Address range>)** |
| Range of debug monitor area | **3FE00-3FFFF** |
| Set user option byte | Yes(-USER_OPT_BYTE) |
| User option byte value | HEX **6E6FE8** |
| Control allocation to trace RAM area | No |
| Control allocation to hot plug-in RAM area | No |

Output Code

Input File

- 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 “**3FE00-3FFFF**” to [Memory area (-debug_monitor=<start address>-<end address>)]. [The example for RL78/F24]

  **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 “**6E6FE8**” to [User option byte value(-user_opt_byte=<value>)]. (WDT stop, LVD [reset mode], 40MHz [The example for RL78/F24])

  **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)].

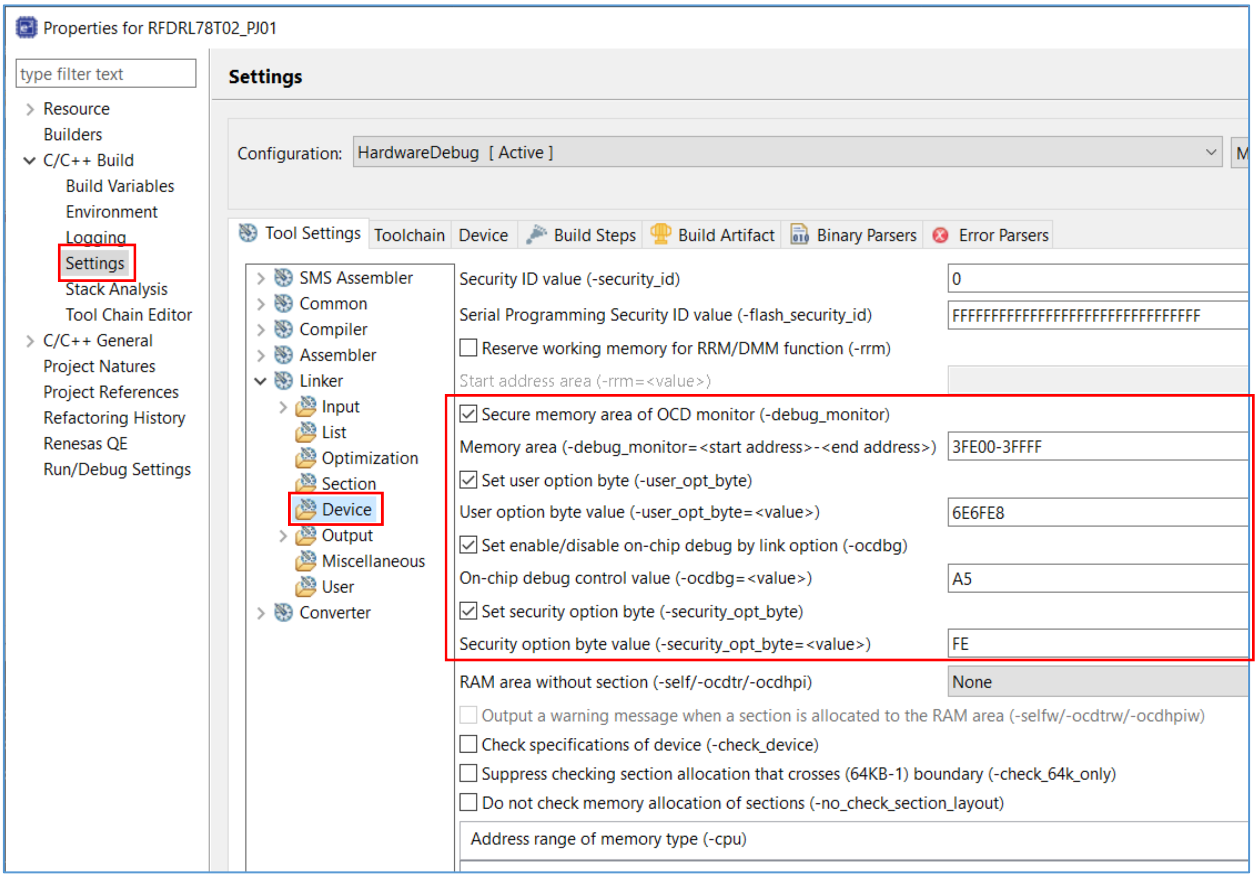
  **Note: The example of a setting on condition of on-chip debugging execution.**

  Input the “**A5**” into [On-chip debug control value(-ocdbg=<value>)]. (Example of permission of operation for on-chip debugging)

  Put a check mark to [Set security option byte (-security_opt_byte)].

  Input the “**FE**” into [Security option byte value (-security_opt_byte=<value>]. [Example of enables read of on-chip debug and flash serial programming security ID.] [The example for RL78/F24]

  **Note: Be sure to confirm the contents of “On-Chip Debug Option Byte” and “Security Option Byte” in “Option Byte” chapter on the user's manual of a target device. And describe the set value used with user application.**



Properties for RFDRL78T02_PJ01

type filter text

Resource
Builders
C/C++ Build
Build Variables
Environment
Logging
Settings
Stack Analysis
Tool Chain Editor
C/C++ General
Project Natures
Project References
Refactoring History
Renesas QE
Run/Debug Settings

Settings

Configuration: HardwareDebug [ Active ]

Tool Settings | Toolchain | Device | Build Steps | Build Artifact | Binary Parsers | Error Parsers

SMS Assembler
Common
Compiler
Assembler
Linker
Input
List
Optimization
Section
Device
Output
Miscellaneous
User
Converter

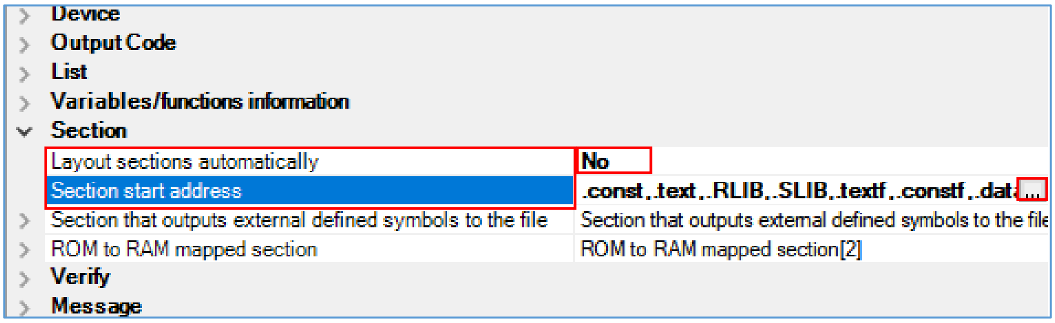
Security ID value (-security_id): 0
Serial Programming Security ID value (-flash_security_id): FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
☐ Reserve working memory for RRM/DMM function (-rrm)
Start address area (-rrm=<value>)
☑ Secure memory area of OCD monitor (-debug_monitor)
Memory area (-debug_monitor=<start address>-<end address>): 3FE00-3FFFF
☑ Set user option byte (-user_opt_byte)
User option byte value (-user_opt_byte=<value>): 6E6FE8
☑ Set enable/disable on-chip debug by link option (-ocdbg)
On-chip debug control value (-ocdbg=<value>): A5
☑ Set security option byte (-security_opt_byte)
Security option byte value (-security_opt_byte=<value>): FE
RAM area without section (-self/-ocdtr/-ocdhpi): None
☐ Output a warning message when a section is allocated to the RAM area (-selfw/-ocdtrw/-ocdhpiw)
☐ Check specifications of device (-check_device)
☐ Suppress checking section allocation that crosses (64KB-1) boundary (-check_64k_only)
☐ Do not check memory allocation of sections (-no_check_section_layout)
Address range of memory type (-cpu)

### 6.1.3.3 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]. Push the “ ... ” button of the right-hand side which sections are displaying, and a “Section settings” screen is displayed.



> Device
> Output Code
> List
> Variables/functions information
⌄ Section

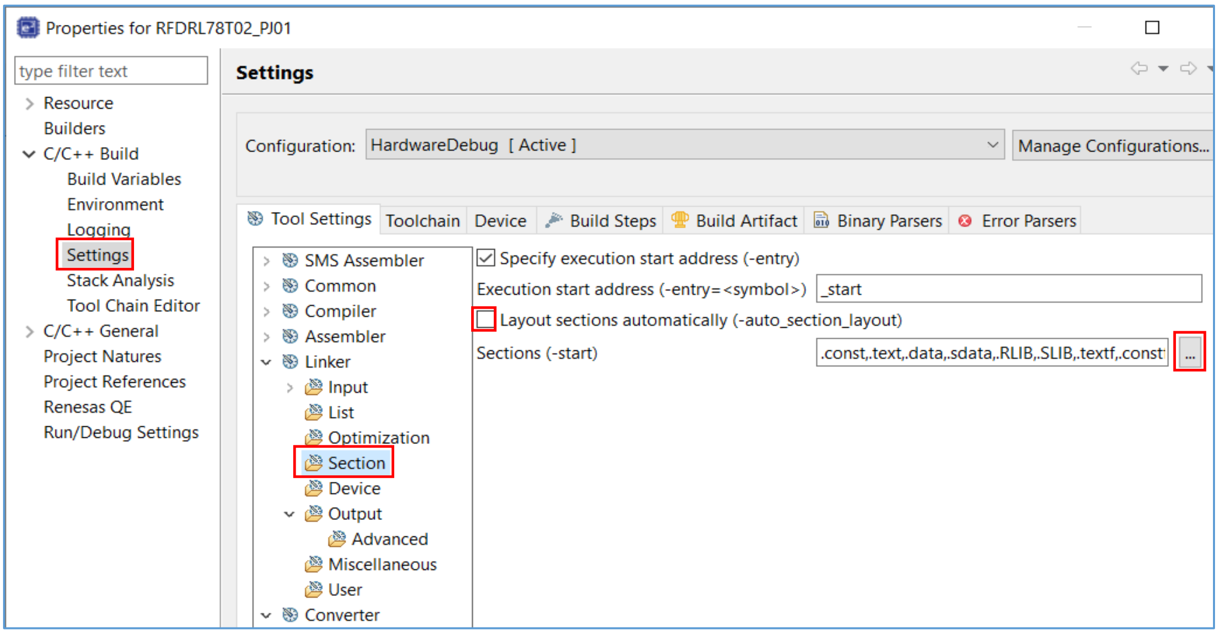
| | |
|---|---|
| Layout sections automatically | No |
| Section start address | .const,.text,.RLIB,.SLIB,.textf,.constf,.dat ... |
| Section that outputs external defined symbols to the file | Section that outputs external defined symbols to the file |
| ROM to RAM mapped section | ROM to RAM mapped section[2] |

> Verify
> Message

- 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)]. Push the “ ... ” button of the right-hand side which sections are displaying, and a “Section viewer” screen is displayed.



Configuration: HardwareDebug [ Active ]

☑ Specify execution start address (-entry)

Execution start address (-entry=<symbol>) _start

☐ Layout sections automatically (-auto_section_layout)

Sections (-start) .const,.text,.data,.sdata,.RLIB,.SLIB,.textf,.const ...

- Section setting operation for CS+ and e$^2$studio

  Set “0x05000” to a top address.
  Add the sections defined by “#pragma section” in RFD RL78 Type 02 to the program area (code flash memory) and the RAM area. Refer to “2.3.1 Sections in case of using RFD RL78 Type 02” for the details of each section.

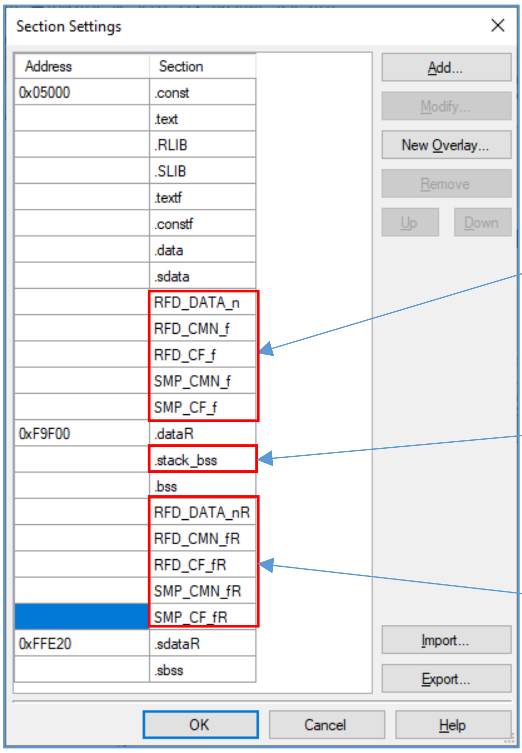
  **Note: In this description, it is a premise to select a medium model as Memory Model of Compile Options. (It is the same as the “auto select” in R7F124FPJ) The section names of each program on “#pragma section” of CC-RL are set to “section name +_f” with a “__far” attribute. The section names copied to RAM from ROM are “section name +_fR” with a “__far” attribute. Copy processing of the sections from ROM to RAM is executed in a cstart.asm file. 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_CMN_f, SMP_CF_f
Add to the RAM area: .stack_bss, RFD_DATA_nR, RFD_CMN_fR, RFD_CF_fR, SMP_CMN_fR, SMP_CF_fR



Section Settings

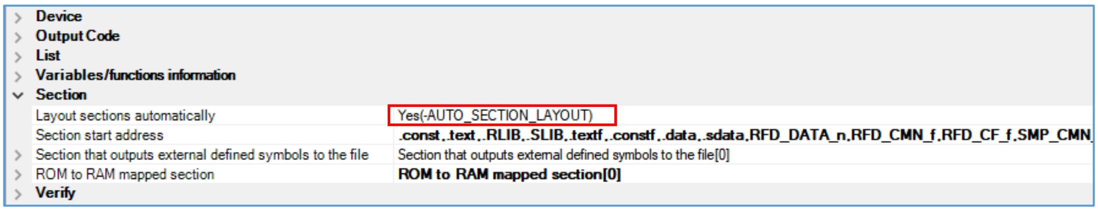
| Address | Section |
|---|---|
| 0x05000 | .const |
| | .text |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | .data |
| | .sdata |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_CF_f |
| | SMP_CMN_f |
| | SMP_CF_f |
| 0xF9F00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| | RFD_CMN_fR |
| | RFD_CF_fR |
| | SMP_CMN_fR |
| | SMP_CF_fR |
| 0xFFE20 | .sdataR |
| | .sbss |

| **Additional sections** |
|---|
| RFD_DATA_n |
| RFD_CMN_f |
| RFD_CF_f |
| SMP_CMN_f |
| SMP_CF_f |

| .stack_bss |
|---|

| RFD_DATA_nR |
|---|
| RFD_CMN_fR |
| RFD_CF_fR |
| SMP_CMN_fR |
| SMP_CF_fR |

Be sure to return [Layout sections automatically] to “Yes”, after pushing the “OK” button.

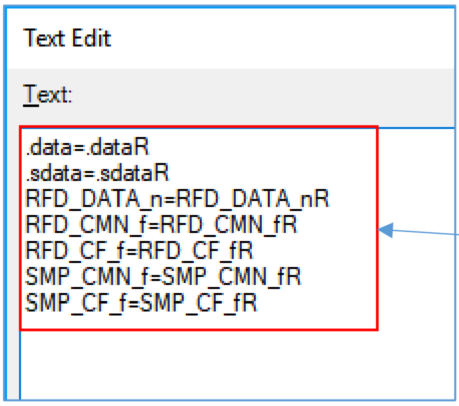


- Device
- Output Code
- List
- Variables/functions information
- Section

| | |
|---|---|
| Layout sections automatically | Yes(-AUTO_SECTION_LAYOUT) |
| Section start address | .const,.text,.RLIB,.SLIB,.textf,.constf,.data,.sdata,RFD_DATA_n,RFD_CMN_f,RFD_CF_f,SMP_CMN |
| Section that outputs external defined symbols to the file | Section that outputs external defined symbols to the file[0] |
| ROM to RAM mapped section | ROM to RAM mapped section[0] |

- Verify

Push 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.



Text Edit

Text:

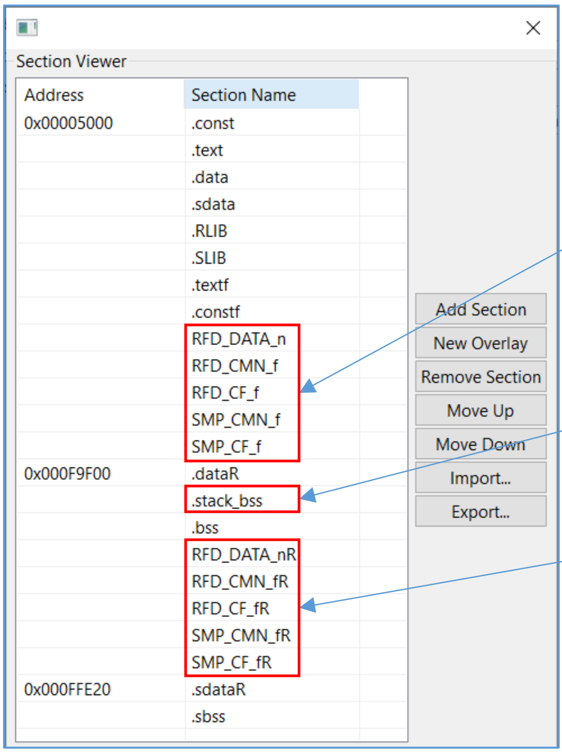
```
.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR
RFD_CMN_f=RFD_CMN_fR
RFD_CF_f=RFD_CF_fR
SMP_CMN_f=SMP_CMN_fR
SMP_CF_f=SMP_CF_fR
```

| **ROM to RAM mapped section (-rom)** |
|---|
| .data=.dataR |
| .sdata=.sdataR |
| RFD_DATA_n=RFD_DATA_nR |
| RFD_CMN_f=RFD_CMN_fR |
| RFD_CF_f=RFD_CF_fR |
| SMP_CMN_f=SMP_CMN_fR |
| SMP_CF_f=SMP_CF_fR |

- 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_CMN_f, SMP_CF_f
Add to the RAM area: .stack_bss, RFD_DATA_nR, RFD_CMN_fR, RFD_CF_fR, SMP_CMN_fR,
SMP_CF_fR



Section Viewer

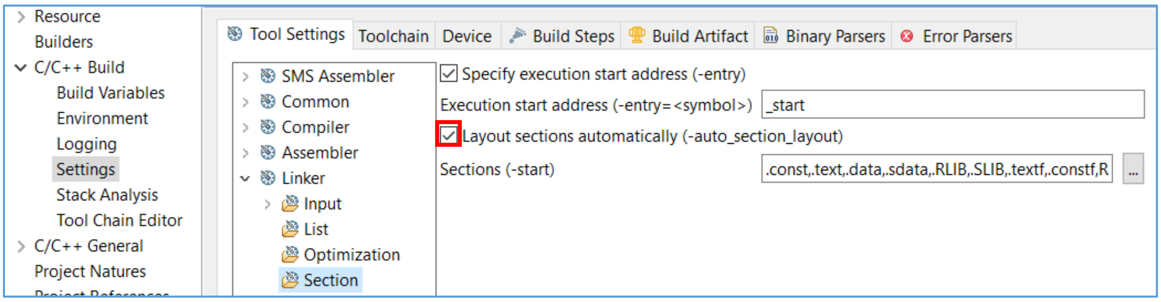
| Address | Section Name |
|---|---|
| 0x00005000 | .const |
| | .text |
| | .data |
| | .sdata |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_CF_f |
| | SMP_CMN_f |
| | SMP_CF_f |
| 0x000F9F00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| | RFD_CMN_fR |
| | RFD_CF_fR |
| | SMP_CMN_fR |
| | SMP_CF_fR |
| 0x000FFE20 | .sdataR |
| | .sbss |

| Additional sections |
|---|
| RFD_DATA_n |
| RFD_CMN_f |
| RFD_CF_f |
| SMP_CMN_f |
| SMP_CF_f |

| .stack_bss |
|---|

| RFD_DATA_nR |
|---|
| RFD_CMN_fR |
| RFD_CF_fR |
| SMP_CMN_fR |
| SMP_CF_fR |

Be sure to put a check mark to [Layout sections automatically (-auto_section_layout)], after pushing the “OK” button.



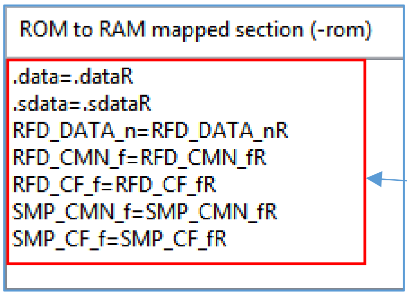
☑ Specify execution start address (-entry)

Execution start address (-entry=<symbol>) _start

☑ Layout sections automatically (-auto_section_layout)

Sections (-start) .const,.text,.data,.sdata,.RLIB,.SLIB,.textf,.constf,R ...

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 (-rom)

.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR
RFD_CMN_f=RFD_CMN_fR
RFD_CF_f=RFD_CF_fR
SMP_CMN_f=SMP_CMN_fR
SMP_CF_f=SMP_CF_fR

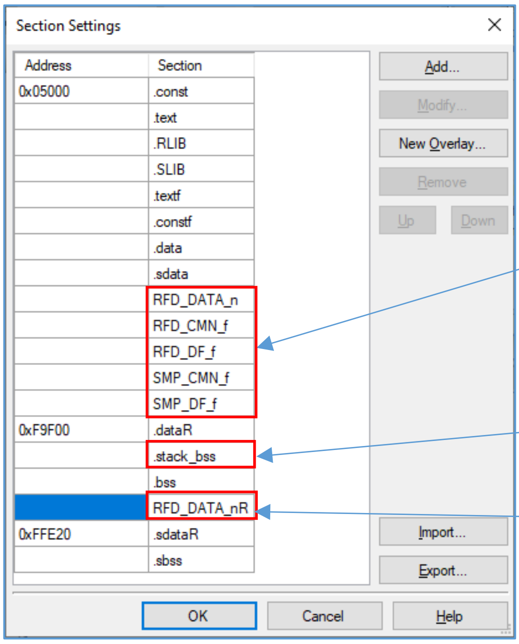
| ROM to RAM mapped section (-rom) |
|---|
| .data=.dataR |
| .sdata=.sdataR |
| RFD_DATA_n=RFD_DATA_nR |
| RFD_CMN_f=RFD_CMN_fR |
| RFD_CF_f=RFD_CF_fR |
| SMP_CMN_f=SMP_CMN_fR |
| SMP_CF_f=SMP_CF_fR |

(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_CMN_f, SMP_DF_f
Add to the RAM area: .stack_bss, RFD_DATA_nR



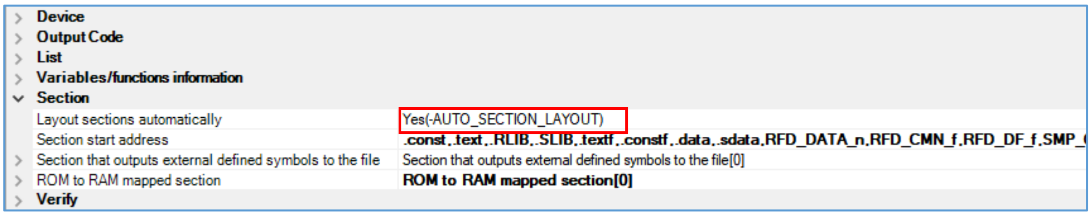
| Address | Section |
|---|---|
| 0x05000 | .const |
| | .text |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | .data |
| | .sdata |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_DF_f |
| | SMP_CMN_f |
| | SMP_DF_f |
| 0xF9F00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| 0xFFE20 | .sdataR |
| | .sbss |

| Additional sections |
|---|
| RFD_DATA_n |
| RFD_CMN_f |
| RFD_DF_f |
| SMP_CMN_f |
| SMP_DF_f |

| .stack_bss |
|---|

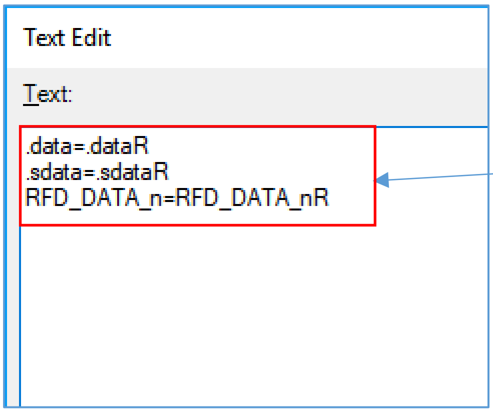
| RFD_DATA_nR |
|---|

Be sure to return [Layout sections automatically] to “Yes”, after pushing the “OK” button.



| | |
|---|---|
| Layout sections automatically | Yes(-AUTO_SECTION_LAYOUT) |
| Section start address | .const,.text,.RLIB,.SLIB,.textf,.constf,.data,.sdata,RFD_DATA_n,RFD_CMN_f,RFD_DF_f,SMP_( |
| Section that outputs external defined symbols to the file | Section that outputs external defined symbols to the file[0] |
| ROM to RAM mapped section | ROM to RAM mapped section[0] |

Push 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.



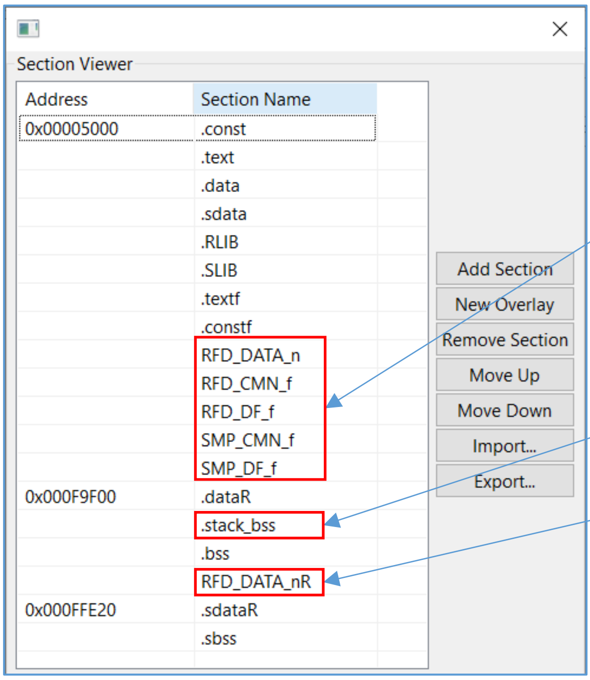
```
.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR
```

| ROM to RAM mapped section (-rom) |
|---|
| .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_CMN_f, SMP_DF_f
  Add to the RAM area: .stack_bss, RFD_DATA_nR



Section Viewer

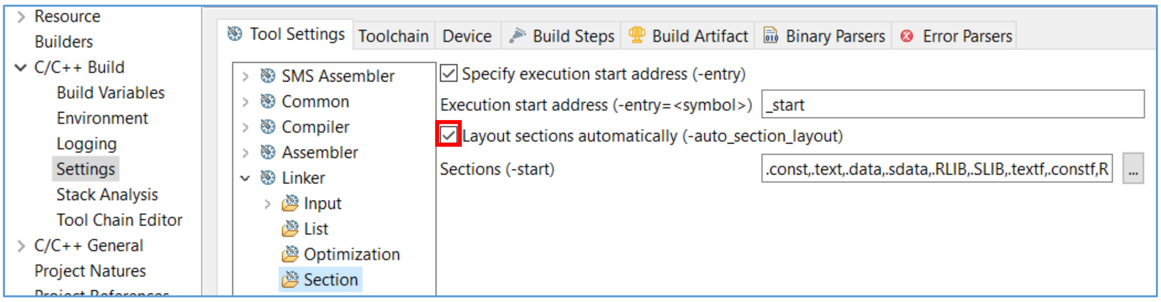
| Address | Section Name |
|---|---|
| 0x00005000 | .const |
| | .text |
| | .data |
| | .sdata |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_DF_f |
| | SMP_CMN_f |
| | SMP_DF_f |
| 0x000F9F00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| 0x000FFE20 | .sdataR |
| | .sbss |

| **Additional sections** |
|---|
| RFD_DATA_n |
| RFD_CMN_f |
| RFD_DF_f |
| SMP_CMN_f |
| SMP_DF_f |

| .stack_bss |
|---|

| RFD_DATA_nR |
|---|

Be sure to put a check mark to [Layout sections automatically (-auto_section_layout)], after pushing the “OK” button.



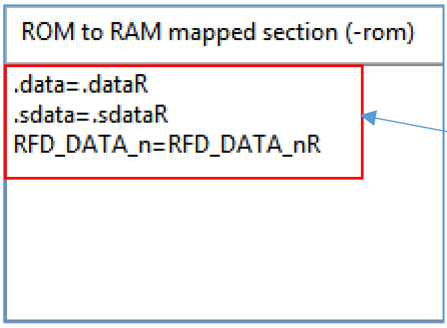
☑ Specify execution start address (-entry)

Execution start address (-entry=<symbol>) _start

☑ Layout sections automatically (-auto_section_layout)

Sections (-start) .const,.text,.data,.sdata,.RLIB,.SLIB,.textf,.constf,R

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 (-rom)

.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR

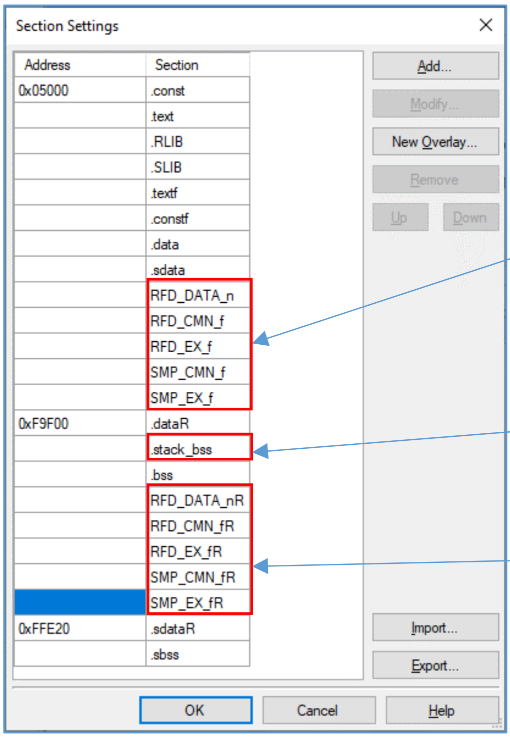
| **ROM to RAM mapped section (-rom)** |
|---|
| .data=.dataR |
| .sdata=.sdataR |
| RFD_DATA_n=RFD_DATA_nR |

(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_CMN_f, SMP_EX_f
  Add to the RAM area: .stack_bss, RFD_DATA_nR, RFD_CMN_fR, RFD_EX_fR, SMP_CMN_fR,
  SMP_EX_fR


Section Settings

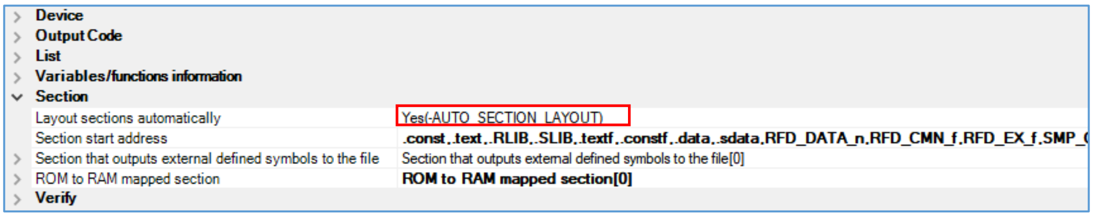
| Address | Section |
|---|---|
| 0x05000 | .const |
| | .text |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | .data |
| | .sdata |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_EX_f |
| | SMP_CMN_f |
| | SMP_EX_f |
| 0xF9F00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| | RFD_CMN_fR |
| | RFD_EX_fR |
| | SMP_CMN_fR |
| | SMP_EX_fR |
| 0xFFE20 | .sdataR |
| | .sbss |

| Additional sections |
|---|
| RFD_DATA_n |
| RFD_CMN_f |
| RFD_EX_f |
| SMP_CMN_f |
| SMP_EX_f |

| .stack_bss |
|---|

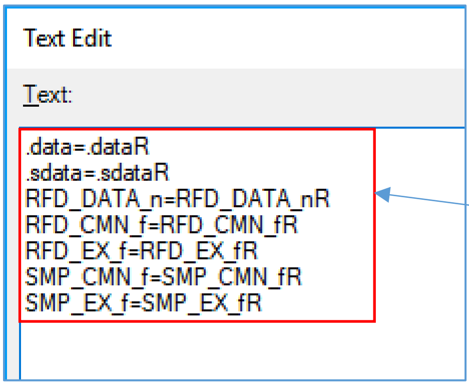
| RFD_DATA_nR |
|---|
| RFD_CMN_fR |
| RFD_EX_fR |
| SMP_CMN_fR |
| SMP_EX_fR |

Be sure to return [Layout sections automatically] to "Yes", after pushing the "OK" button.



| | |
|---|---|
| Layout sections automatically | Yes(-AUTO_SECTION_LAYOUT) |
| Section start address | .const,.text,.RLIB,.SLIB,.textf,.constf,.data,.sdata,RFD_DATA_n,RFD_CMN_f,RFD_EX_f,SMP_ |
| Section that outputs external defined symbols to the file | Section that outputs external defined symbols to the file[0] |
| ROM to RAM mapped section | ROM to RAM mapped section[0] |

Push 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.


Text Edit

Text:

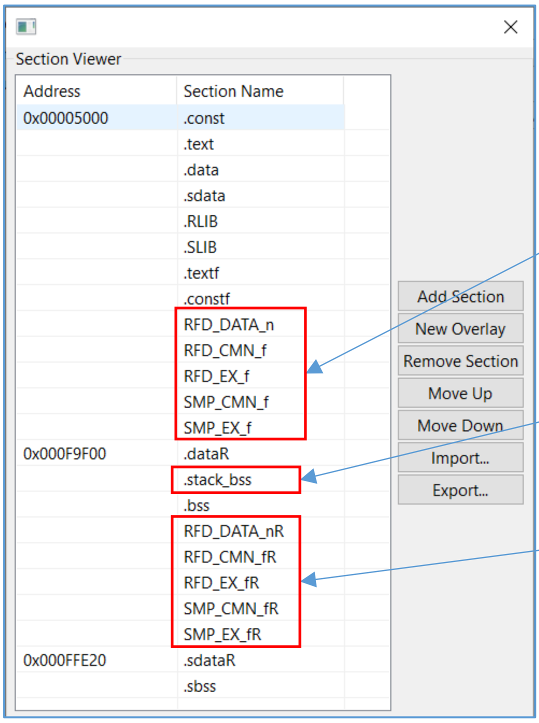
```
.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR
RFD_CMN_f=RFD_CMN_fR
RFD_EX_f=RFD_EX_fR
SMP_CMN_f=SMP_CMN_fR
SMP_EX_f=SMP_EX_fR
```

| ROM to RAM mapped section (-rom) |
|---|
| .data=.dataR |
| .sdata=.sdataR |
| RFD_DATA_n=RFD_DATA_nR |
| RFD_CMN_f=RFD_CMN_fR |
| RFD_EX_f=RFD_EX_fR |
| SMP_CMN_f=SMP_CMN_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_CMN_f, SMP_EX_f
Add to the RAM area: .stack_bss, RFD_DATA_nR, RFD_CMN_fR, RFD_EX_fR, SMP_CMN_fR,
SMP_EX_fR



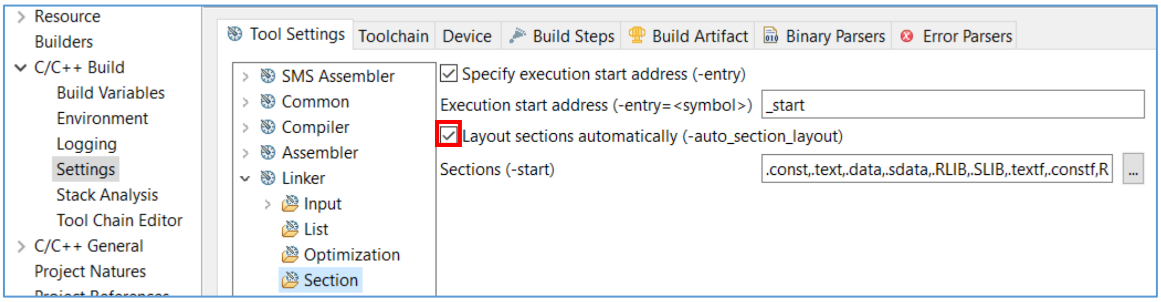
| Address | Section Name |
|---|---|
| 0x00005000 | .const |
| | .text |
| | .data |
| | .sdata |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_EX_f |
| | SMP_CMN_f |
| | SMP_EX_f |
| 0x000F9F00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| | RFD_CMN_fR |
| | RFD_EX_fR |
| | SMP_CMN_fR |
| | SMP_EX_fR |
| 0x000FFE20 | .sdataR |
| | .sbss |

| **Additional sections** |
|---|
| RFD_DATA_n |
| RFD_CMN_f |
| RFD_EX_f |
| SMP_CMN_f |
| SMP_EX_f |

| .stack_bss |
|---|

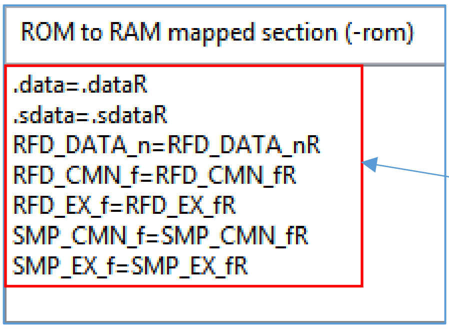
| RFD_DATA_nR |
|---|
| RFD_CMN_fR |
| RFD_EX_fR |
| SMP_CMN_fR |
| SMP_EX_fR |

Be sure to put a check mark to [Layout sections automatically (-auto_section_layout)], after pushing the “OK” button.



☑ Specify execution start address (-entry)
Execution start address (-entry=<symbol>) _start
☑ Layout sections automatically (-auto_section_layout)
Sections (-start) .const,.text,.data,.sdata,.RLIB,.SLIB,.textf,.constf,R

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 (-rom)

.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR
RFD_CMN_f=RFD_CMN_fR
RFD_EX_f=RFD_EX_fR
SMP_CMN_f=SMP_CMN_fR
SMP_EX_f=SMP_EX_fR

| **ROM to RAM mapped section (-rom)** |
|---|
| .data=.dataR |
| .sdata=.sdataR |
| RFD_DATA_n=RFD_DATA_nR |
| RFD_CMN_f=RFD_CMN_fR |
| RFD_EX_f=RFD_EX_fR |
| SMP_CMN_f=SMP_CMN_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 (“RFDRL78T02_PJ01 HardwareDebug”) of [Renesas GDB Hardware Debugging]. And the displayed “Debugger” tab performs debugging tool setting.

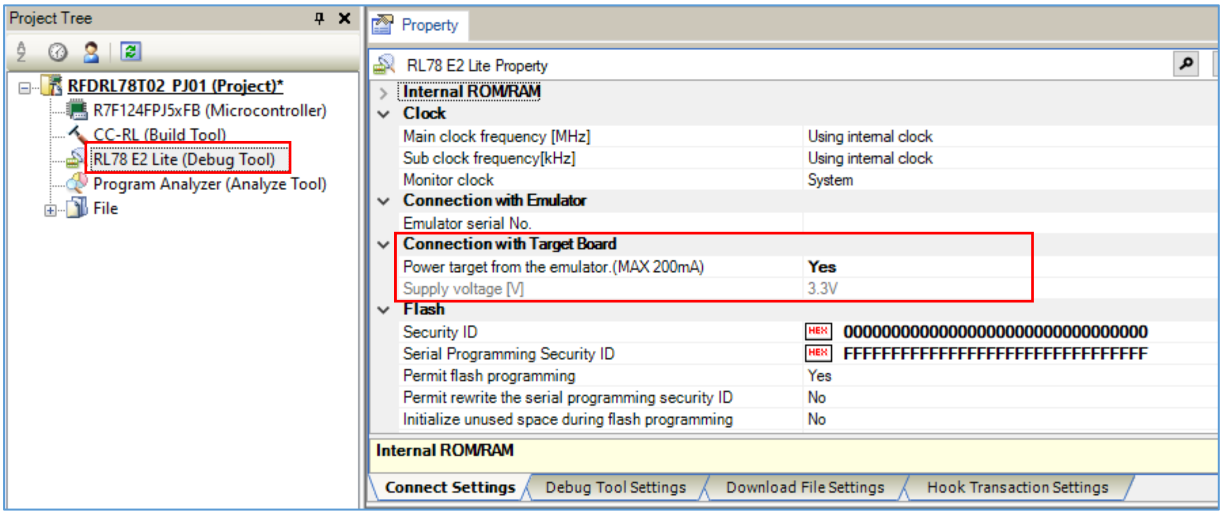
**Note: The power is already supplied to the target board, or when power supply capacity is insufficient, the emulator including E2 Lite may be unable to supply power to a target board. Be sure to refer to “the user's manual and Additional Document for User's Manual (Notes on Connection of RL78)” for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


Project Tree
RFDRL78T02_PJ01 (Project)*
R7F124FPJ5xFB (Microcontroller)
CC-RL (Build Tool)
RL78 E2 Lite (Debug Tool)
Program Analyzer (Analyze Tool)
File
Property
RL78 E2 Lite Property
Connect Settings
Debug Tool Settings
Download File Settings
Hook Transaction Settings

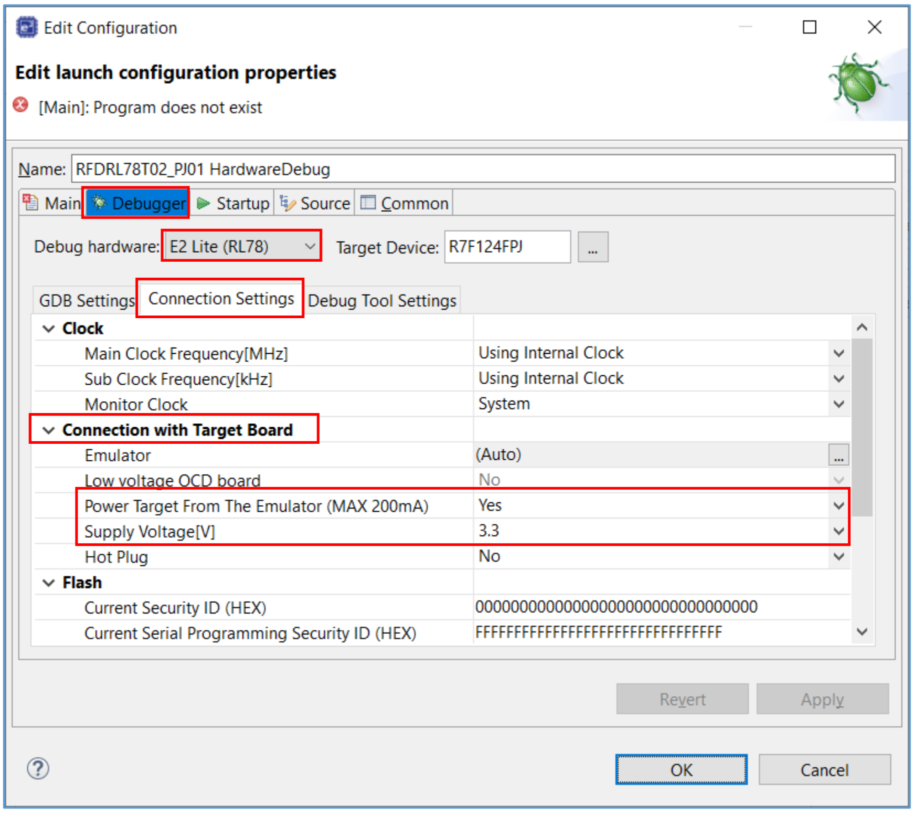
| | |
|---|---|
| **Internal ROM/RAM** | |
| **Clock** | |
| Main clock frequency [MHz] | Using internal clock |
| Sub clock frequency[kHz] | Using internal clock |
| Monitor clock | System |
| **Connection with Emulator** | |
| Emulator serial No. | |
| **Connection with Target Board** | |
| Power target from the emulator.(MAX 200mA) | **Yes** |
| Supply voltage [V] | 3.3V |
| **Flash** | |
| Security ID | HEX 00000000000000000000000000000000 |
| Serial Programming Security ID | HEX FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF |
| Permit flash programming | Yes |
| Permit rewrite the serial programming security ID | No |
| Initialize unused space during flash programming | No |

Internal ROM/RAM

- 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)].



Edit Configuration

**Edit launch configuration properties**

[Main]: Program does not exist

Name: RFDRL78T02_PJ01 HardwareDebug

Debug hardware: E2 Lite (RL78) Target Device: R7F124FPJ

| | |
|---|---|
| **Clock** | |
| Main Clock Frequency[MHz] | Using Internal Clock |
| Sub Clock Frequency[kHz] | Using Internal Clock |
| Monitor Clock | System |
| **Connection with Target Board** | |
| Emulator | (Auto) |
| Low voltage OCD board | No |
| Power Target From The Emulator (MAX 200mA) | Yes |
| Supply Voltage[V] | 3.3 |
| Hot Plug | No |
| **Flash** | |
| Current Security ID (HEX) | 00000000000000000000000000000000 |
| Current Serial Programming Security ID (HEX) | FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF |

## 6.2 Creating a Project in the Case of Using IAR Compiler

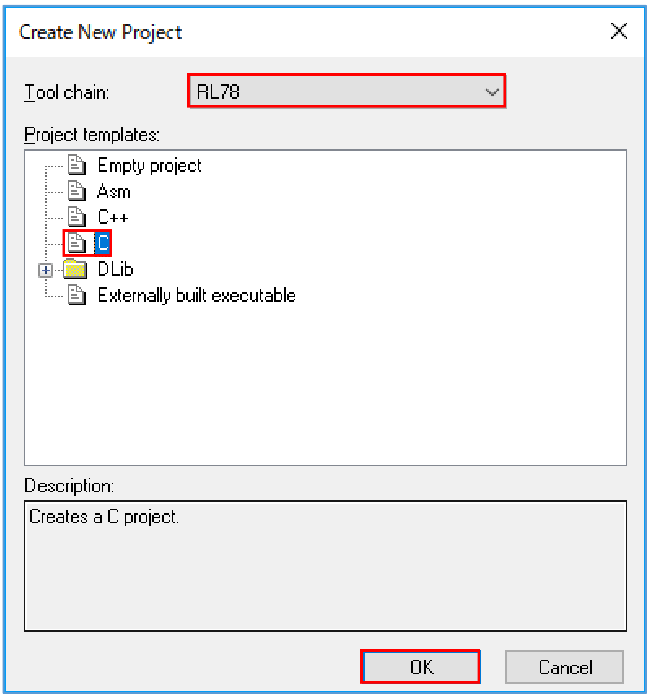
IAR Embedded Workbench can be used for a IAR compiler as an IDE. RFD RL78 Type 02 is registered and built in the project created by the IDE. An example of creating a sample project in case each IDE is used is shown. Because to understand a IAR compiler and each IDE, it is necessary to refer to the user's manual of each tool product.

**IAR Systems, IAR Embedded Workbench, C-SPY, IAR, and the logotype of IAR Systems are trademarks or registered trademarks owned by IAR Systems AB.**

### 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.



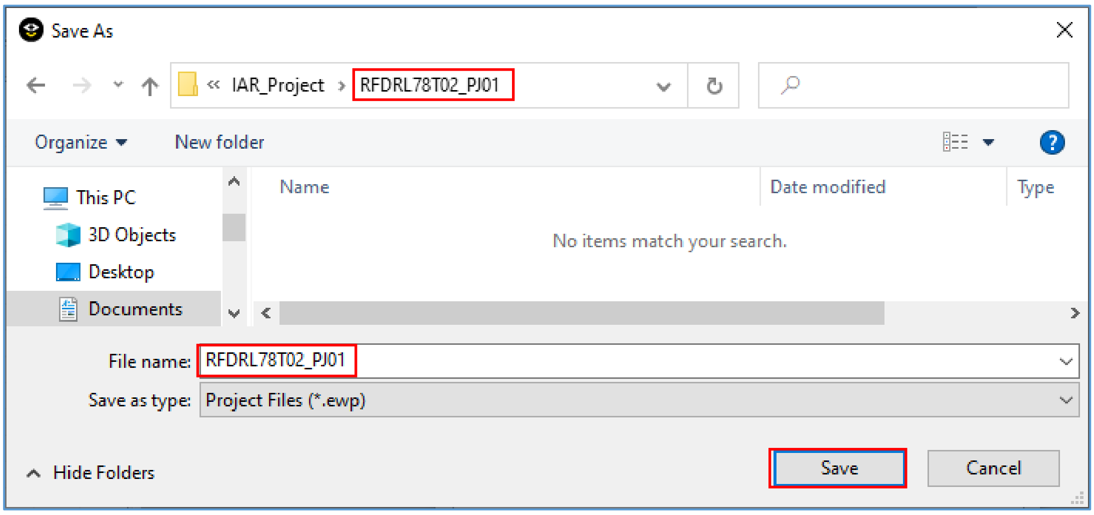
Create New Project

Tool chain: RL78

Project templates: C

Description:
Creates a C project.

- Create “RFDRL78T02_PJ01” folder temporarily, and move into a folder.
- The Project File name is temporarily set to “RFDRL78T02_PJ01”.



Save As

« IAR_Project › RFDRL78T02_PJ01

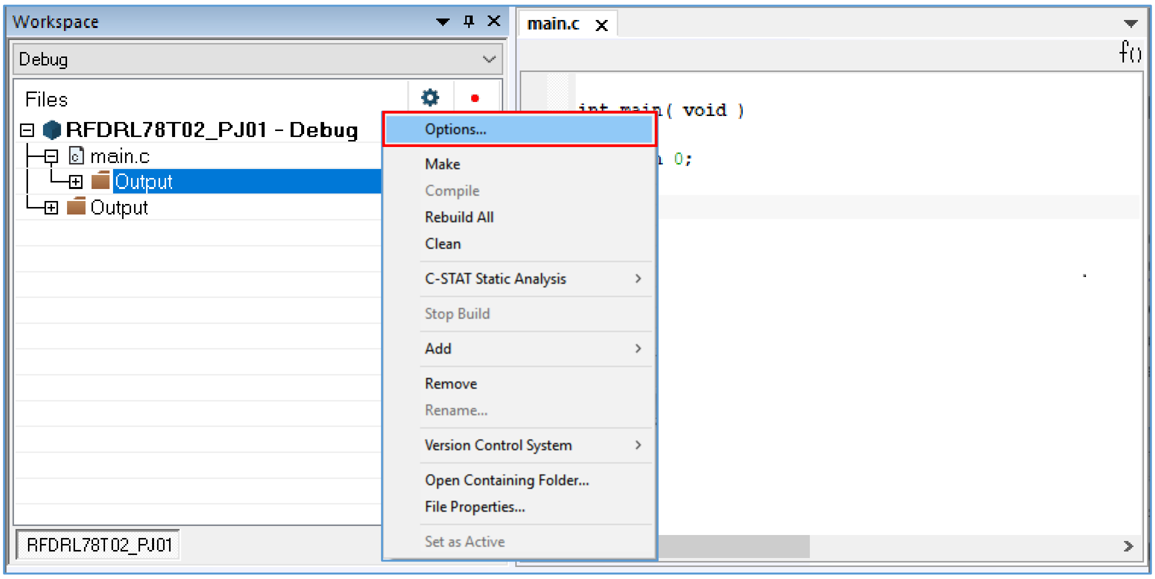
No items match your search.

File name: RFDRL78T02_PJ01

Save as type: Project Files (*.ewp)

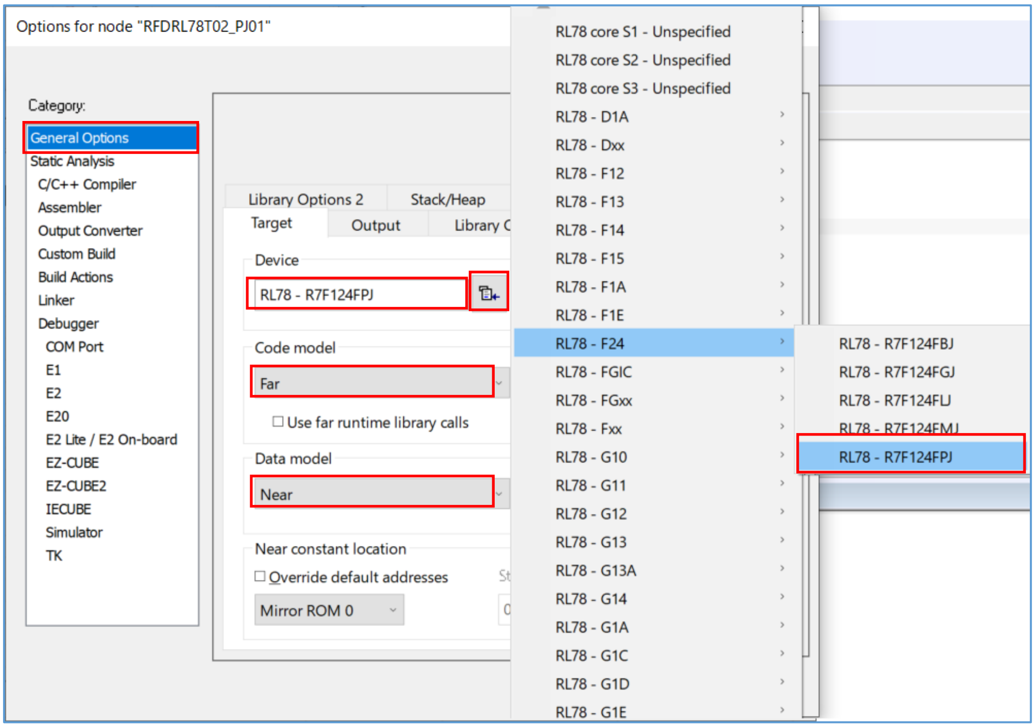
(2) Selection of a target device

・On IAR Embedded Workbench, I click the right mouse button of the project (“RFDRL78T02_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.

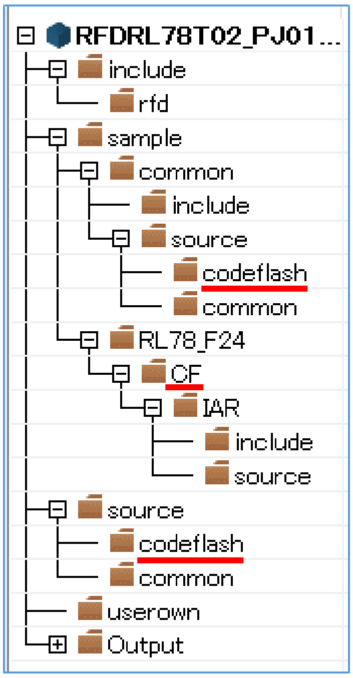
- Push “ ” button of [Device]. And select “RL78 - F24” - “RL78 - R7F124FPJ”. Select “Far” as [code model] and select “Near” as [Data model].



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

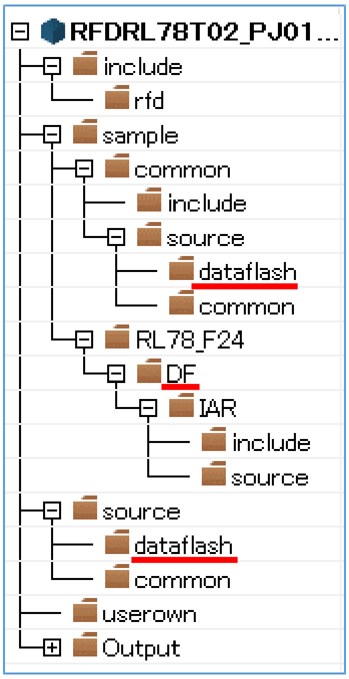
Using RFD RL78 Type 02, when programming each area [(1) code flash memory, (2) data flash memory, (3) extra area], the example which registers necessary files is shown. Each folder of a RFD RL78 Type 02 source-program file is “include”, “source”, “userown”, and “sample”. The target file in each folder 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 02, 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, and (3) Extra area] is shown. (The group name which changes with areas is shown by “ ▬ ”.)



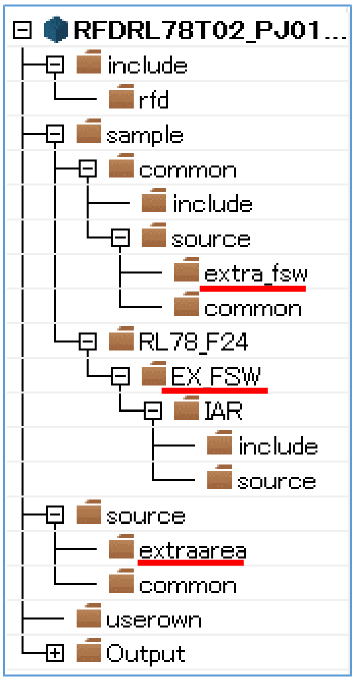
```
⊟ RFDRL78T02_PJ01...
├⊟ include
│ └── rfd
├⊟ sample
│ ├⊟ common
│ │ ├── include
│ │ └⊟ source
│ │   ├── codeflash
│ │   └── common
│ └⊟ RL78_F24
│   └⊟ CF
│     └⊟ IAR
│       ├── include
│       └── source
├⊟ source
│ ├── codeflash
│ └── common
├── userown
└⊞ Output
```

(1) Code flash memory



```
⊟ RFDRL78T02_PJ01...
├⊟ include
│ └── rfd
├⊟ sample
│ ├⊟ common
│ │ ├── include
│ │ └⊟ source
│ │   ├── dataflash
│ │   └── common
│ └⊟ RL78_F24
│   └⊟ DF
│     └⊟ IAR
│       ├── include
│       └── source
├⊟ source
│ ├── dataflash
│ └── common
├── userown
└⊞ Output
```

(2) Data flash memory



```
⊟ RFDRL78T02_PJ01...
├⊟ include
│ └── rfd
├⊟ sample
│ ├⊟ common
│ │ ├── include
│ │ └⊟ source
│ │   ├── extra_fsw
│ │   └── common
│ └⊟ RL78_F24
│   └⊟ EX_FSW
│     └⊟ IAR
│       ├── include
│       └── source
├⊟ source
│ ├── extraarea
│ └── common
├── userown
└⊞ Output
```

(3) Extra area

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

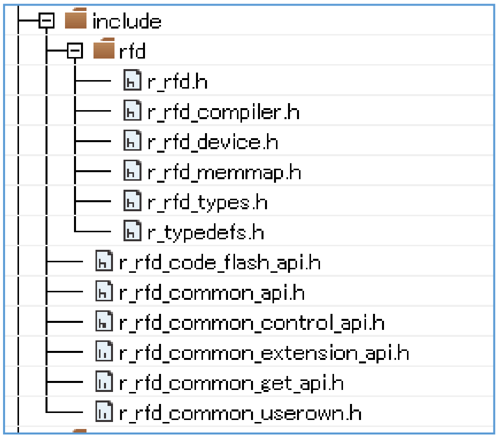
There are files added automatically in the created project. The same file as these exists also in the “sample” folder of RFD RL78 Type 02. Therefore, using the function of IDE, select those files from tree and excludes from a project.

- 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

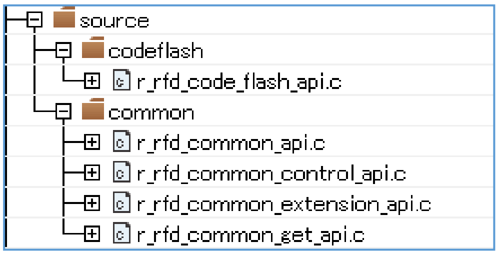
The groups (“include”, “source”, “userown”, “sample”) and source program file which are included in RFD RL78 Type 02 to register are shown below.

in the “include” group



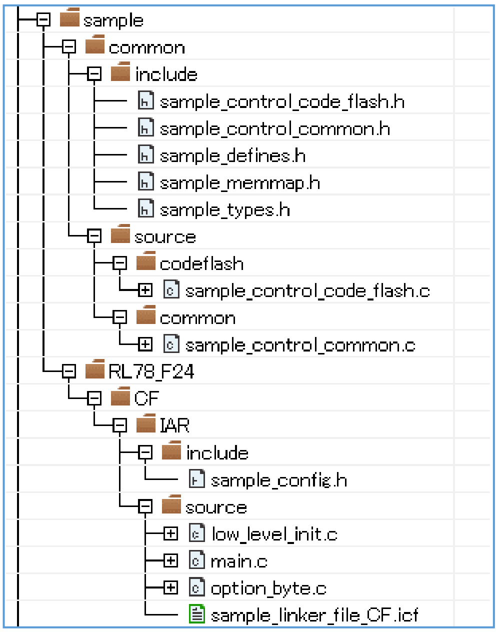
```
include
  rfd
    r_rfd.h
    r_rfd_compiler.h
    r_rfd_device.h
    r_rfd_memmap.h
    r_rfd_types.h
    r_typedefs.h
  r_rfd_code_flash_api.h
  r_rfd_common_api.h
  r_rfd_common_control_api.h
  r_rfd_common_extension_api.h
  r_rfd_common_get_api.h
  r_rfd_common_userown.h
```

in the “source” group



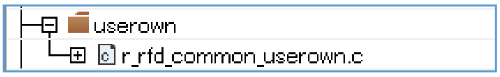
```
source
  codeflash
    r_rfd_code_flash_api.c
  common
    r_rfd_common_api.c
    r_rfd_common_control_api.c
    r_rfd_common_extension_api.c
    r_rfd_common_get_api.c
```

in the “sample” group



```
sample
  common
    include
      sample_control_code_flash.h
      sample_control_common.h
      sample_defines.h
      sample_memmap.h
      sample_types.h
    source
      codeflash
        sample_control_code_flash.c
      common
        sample_control_common.c
  RL78_F24
    CF
      IAR
        include
          sample_config.h
        source
          low_level_init.c
          main.c
          option_byte.c
          sample_linker_file_CF.icf
```

in the “userown” group

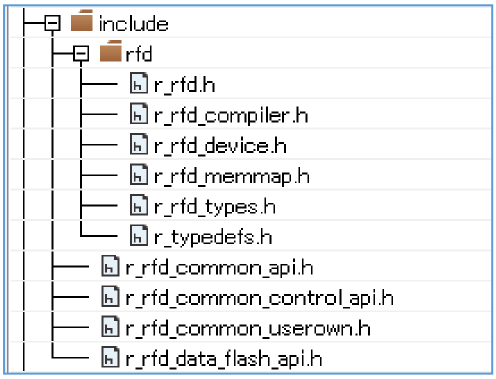


```
userown
  r_rfd_common_userown.c
```

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

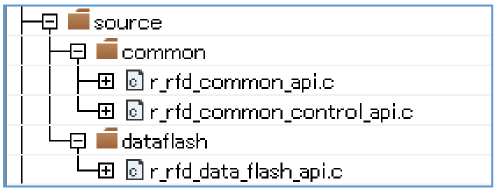
The groups (“include”, “source”, “userown”, “sample”) and source program file which are included in RFD RL78 Type 02 to register are shown below.

in the “include” group



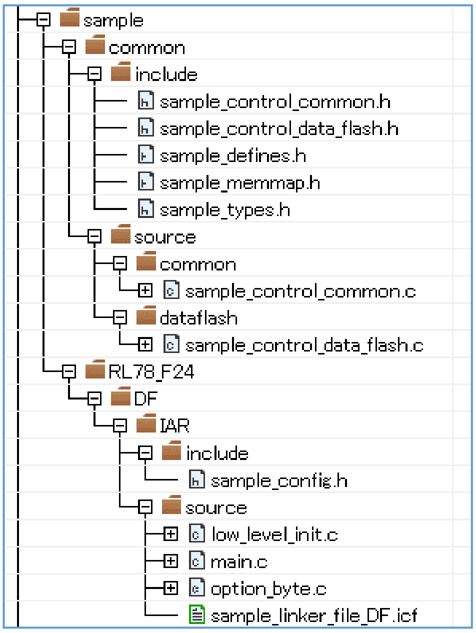
```
include
├─ rfd
│  ├── r_rfd.h
│  ├── r_rfd_compiler.h
│  ├── r_rfd_device.h
│  ├── r_rfd_memmap.h
│  ├── r_rfd_types.h
│  └── r_typedefs.h
├── r_rfd_common_api.h
├── r_rfd_common_control_api.h
├── r_rfd_common_userown.h
└── r_rfd_data_flash_api.h
```

in the “source” group



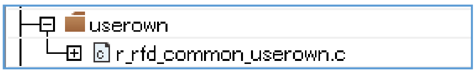
```
source
├─ common
│  ├─ r_rfd_common_api.c
│  └─ r_rfd_common_control_api.c
└─ dataflash
   └─ r_rfd_data_flash_api.c
```

in the “sample” group



```
sample
├─ common
│  ├─ include
│  │  ├── sample_control_common.h
│  │  ├── sample_control_data_flash.h
│  │  ├── sample_defines.h
│  │  ├── sample_memmap.h
│  │  └── sample_types.h
│  └─ source
│     ├─ common
│     │  └─ sample_control_common.c
│     └─ dataflash
│        └─ sample_control_data_flash.c
└─ RL78_F24
   └─ DF
      └─ IAR
         ├─ include
         │  └── sample_config.h
         └─ source
            ├─ low_level_init.c
            ├─ main.c
            ├─ option_byte.c
            └── sample_linker_file_DF.icf
```

in the “userown” group

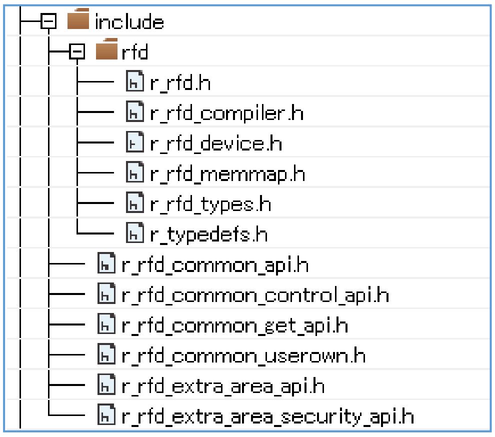


```
userown
└─ r_rfd_common_userown.c
```

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

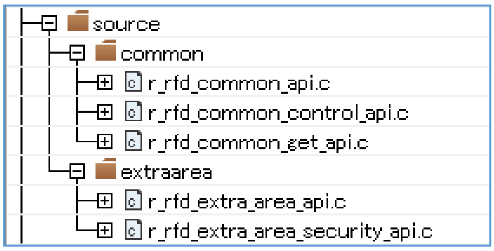
The groups ("include", "source", "userown", "sample") and source program file which are included in RFD RL78 Type 02 to register are shown below.

in the "include" group



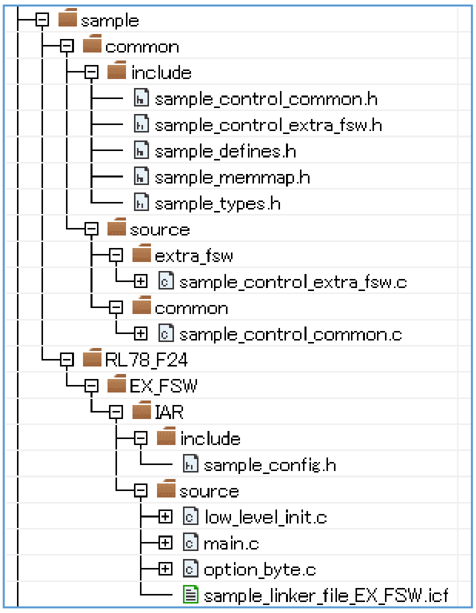
```
include
  rfd
    r_rfd.h
    r_rfd_compiler.h
    r_rfd_device.h
    r_rfd_memmap.h
    r_rfd_types.h
    r_typedefs.h
  r_rfd_common_api.h
  r_rfd_common_control_api.h
  r_rfd_common_get_api.h
  r_rfd_common_userown.h
  r_rfd_extra_area_api.h
  r_rfd_extra_area_security_api.h
```

in the "source" group



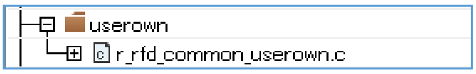
```
source
  common
    r_rfd_common_api.c
    r_rfd_common_control_api.c
    r_rfd_common_get_api.c
  extraarea
    r_rfd_extra_area_api.c
    r_rfd_extra_area_security_api.c
```

in the "sample" group



```
sample
  common
    include
      sample_control_common.h
      sample_control_extra_fsw.h
      sample_defines.h
      sample_memmap.h
      sample_types.h
    source
      extra_fsw
        sample_control_extra_fsw.c
      common
        sample_control_common.c
  RL78_F24
    EX_FSW
      IAR
        include
          sample_config.h
        source
          low_level_init.c
          main.c
          option_byte.c
          sample_linker_file_EX_FSW.icf
```

in the "userown" group



```
userown
  r_rfd_common_userown.c
```

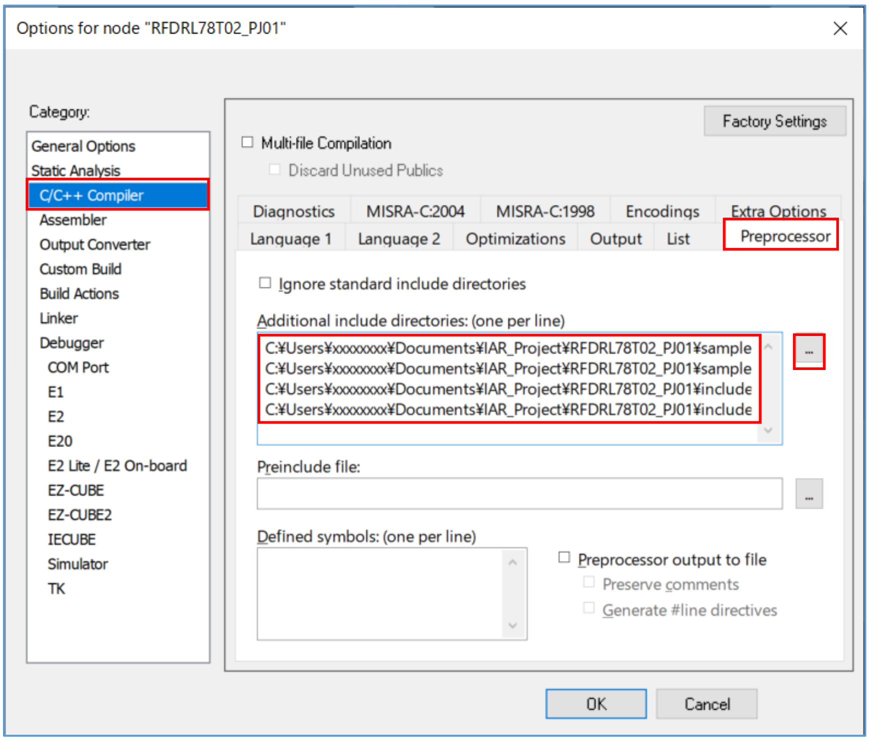
### 6.2.3 Integrated Development Environment (IDE) Settings

Set IDE setting necessary in order to build RFD RL78 Type 02 using an IAR compiler.
IAR Embedded Workbench: Click the right mouse button for the project (“RFDRL78T02_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)].



□ Multi-file Compilation
□ Discard Unused Publics
□ Ignore standard include directories

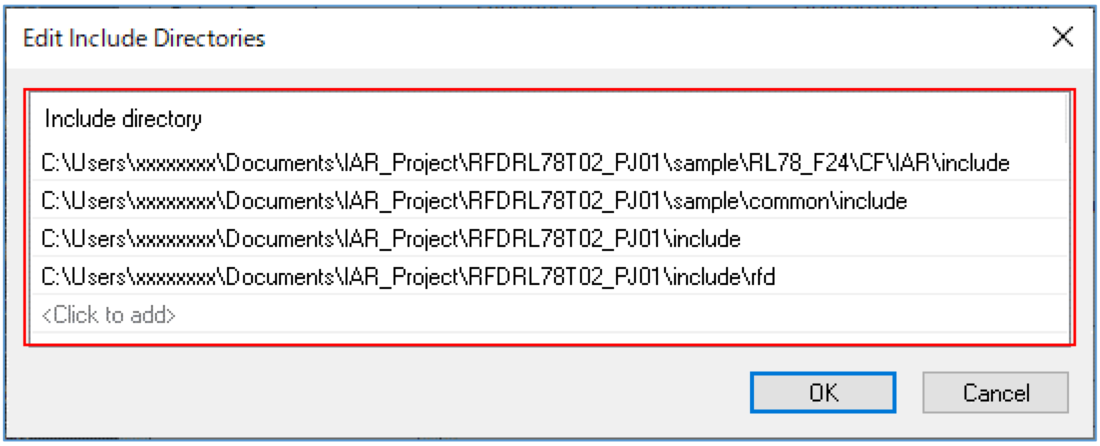
Additional include directories: (one per line)

```
C:¥Users¥xxxxxxxx¥Documents¥IAR_Project¥RFDRL78T02_PJ01¥sample
C:¥Users¥xxxxxxxx¥Documents¥IAR_Project¥RFDRL78T02_PJ01¥sample
C:¥Users¥xxxxxxxx¥Documents¥IAR_Project¥RFDRL78T02_PJ01¥include
C:¥Users¥xxxxxxxx¥Documents¥IAR_Project¥RFDRL78T02_PJ01¥include
```

Preinclude file:

Defined symbols: (one per line)

□ Preprocessor output to file
□ Preserve comments
□ Generate #line directives



Include directory

```
C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T02_PJ01\sample\RL78_F24\CF\IAR\include
C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T02_PJ01\sample\common\include
C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T02_PJ01\include
C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T02_PJ01\include\rfd
<Click to add>
```

- 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 02 on "C:\Users\xxxxxxxx\Documents\IAR_Project\".

(1) Code flash memory reprogramming

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T02_PJ01\sample\RL78_F24\CF\IAR\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T02_PJ01\sample\common\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T02_PJ01\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T02_PJ01\include\rfd

(2) Data flash memory reprogramming

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T02_PJ01\sample\RL78_F24\DF\IAR\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T02_PJ01\sample\common\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T02_PJ01\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T02_PJ01\include\rfd

(3) Extra area (FSW) reprogramming

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T02_PJ01\sample\RL78_F24\EX_FSW\IAR\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T02_PJ01\sample\common\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T02_PJ01\include

C:\Users\xxxxxxxx\Documents\IAR_Project\RFDRL78T02_PJ01\include\rfd

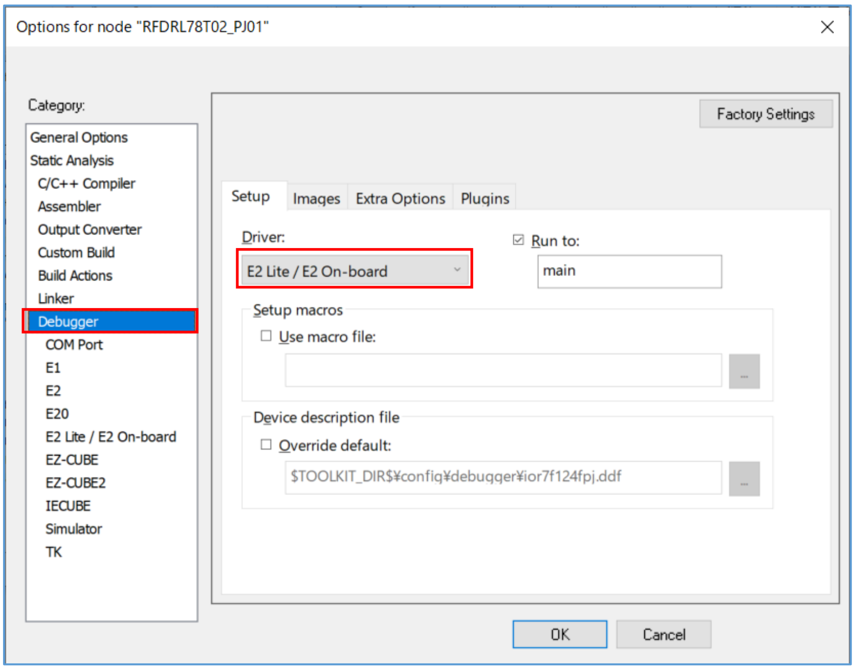
**Note: About the path setting of include directories.**

**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 Debugger Settings

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



Options for node "RFDRL78T02_PJ01"

Category: General Options, Static Analysis, C/C++ Compiler, Assembler, Output Converter, Custom Build, Build Actions, Linker, Debugger, COM Port, E1, E2, E20, E2 Lite / E2 On-board, EZ-CUBE, EZ-CUBE2, IECUBE, Simulator, TK

Tabs: Setup, Images, Extra Options, Plugins

Driver: E2 Lite / E2 On-board

☑ Run to: main

Setup macros

☐ Use macro file:

Device description file

☐ Override default:

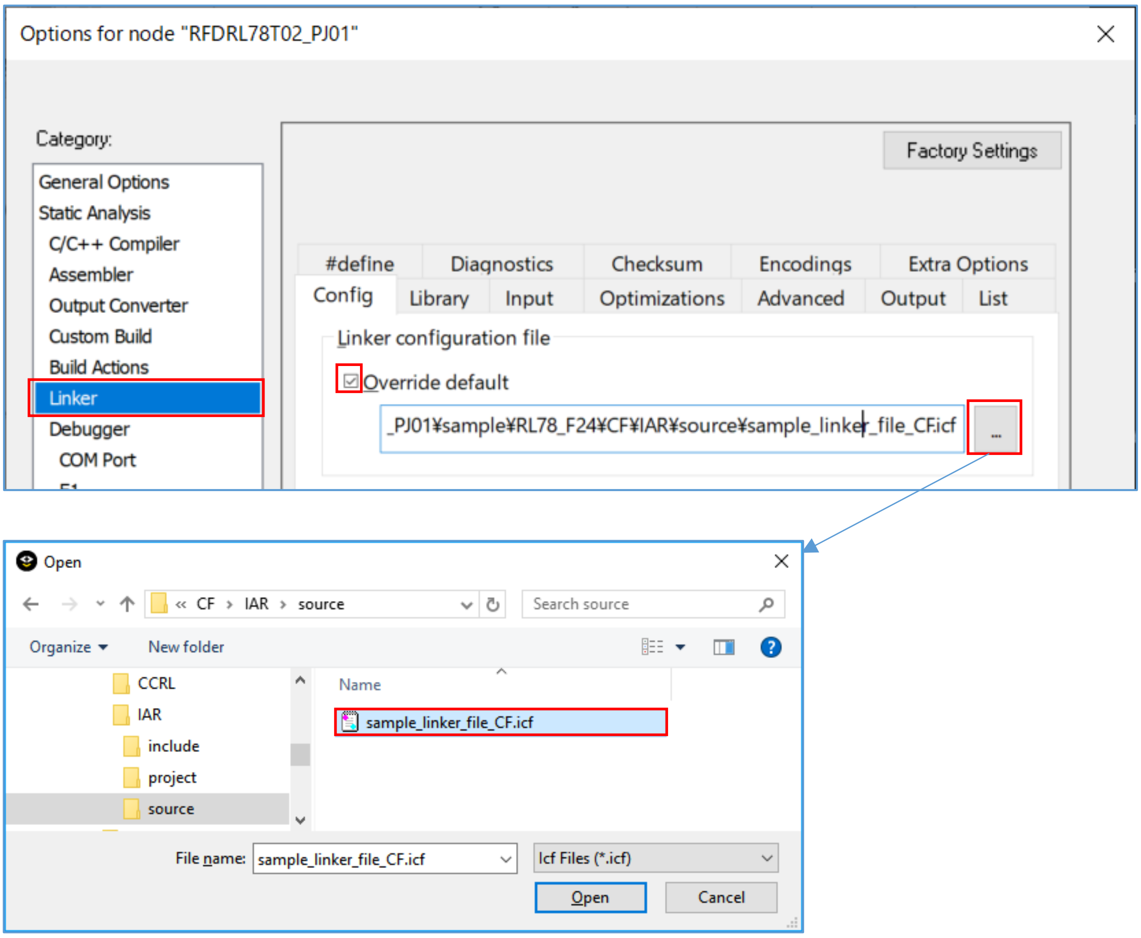
$TOOLKIT_DIR$¥config¥debugger¥ior7f124fpj.ddf

**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 02. Linker configuration file (*. icf) for every reprogramming area is as follows.

- For code flash memory reprogramming: sample_linker_file_CF.icf (\Sample\RL78_F24\CF\IAR\source\)
- For data flash memory reprogramming: sample_linker_file_DF.icf (\Sample\RL78_F24\DF\IAR\source\)
- For Extra area (FSW): sample_linker_file_EX_FSW.icf (\Sample\RL78_F24\EX_FSW\IAR\source\)



Options for node "RFDRL78T02_PJ01"

Category: Linker

Config tab — Linker configuration file

☑ Override default

_PJ01¥sample¥RL78_F24¥CF¥IAR¥source¥sample_linker_file_CF.icf

Open

« CF › IAR › source

sample_linker_file_CF.icf

File name: sample_linker_file_CF.icf

Icf Files (*.icf)

**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 02 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 02.

(1) The addition of the sections for code flash memory reprogramming

Add the initial value of each section of RFD_DATA, RFD_CMN, RFD_CF, SMP_CMN, and SMP_CF to ROM area (ROM_far). It is necessary to copy them to the section of RAM area (RAM_near, RAM_code).

- The additional section of the ROM_far area (The data and the program for copying to RAM area):
  RFD_DATA_init, RFD_CMN_init, RFD_CF_init, SMP_CMN_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_CMN, RFD_CF, SMP_CMN, SMP_CF

(2) The addition of the sections for data flash memory reprogramming

Add the initial value of each section of RFD_DATA, RFD_CMN, RFD_DF, SMP_CMN, and SMP_DF to ROM area (ROM_far). It is necessary to copy RFD_DATA to the section of RAM area (RAM_near).

- The additional section of the ROM_far area (The program and the data for copying to RAM area to be placed in ROM area):
  RFD_DATA_init, RFD_CMN, RFD_DF, SMP_CMN, 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_CMN, RFD_EX, SMP_CMN, and SMP_EX to ROM area (ROM_far). It is necessary to copy them to the section of RAM area (RAM_near, RAM_code).

- The additional section of the ROM_far area (The data and the program for copying to RAM area):
  RFD_DATA_init, RFD_CMN_init, RFD_EX_init, SMP_CMN_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_CMN, RFD_EX, SMP_CMN, SMP_EX

#### 6.2.4.2 Option Bytes Settings

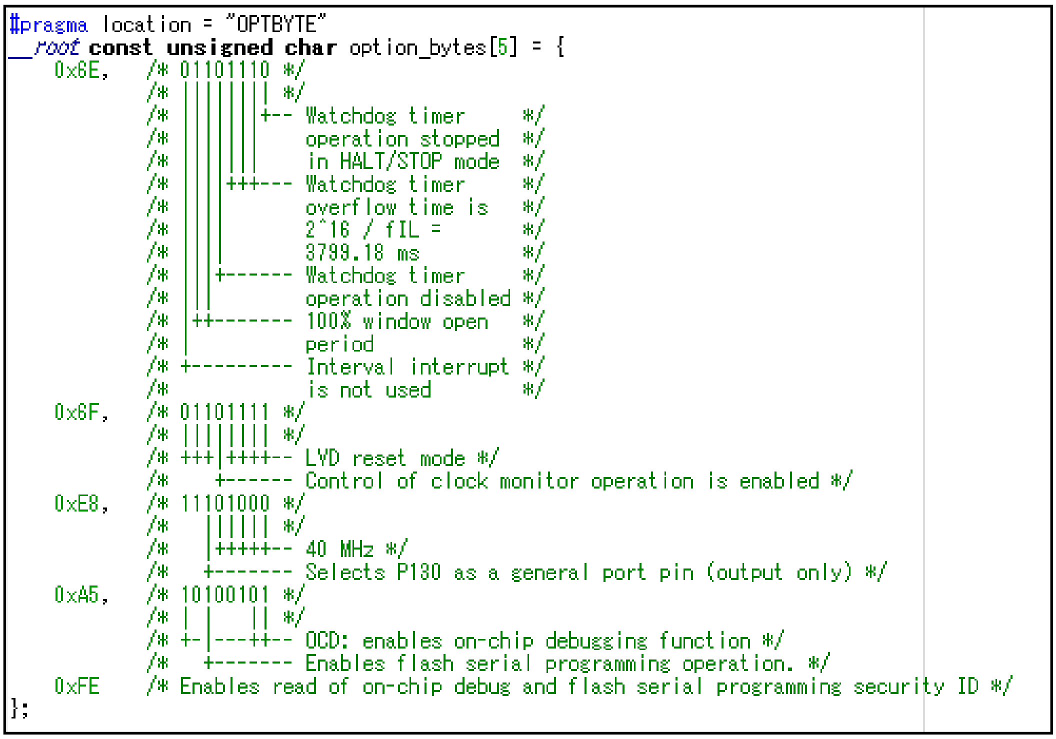
The Option bytes definition of RL78 is described in Linker configuration file (*. icf) of IAR Embedded Workbench attachment or the sample_linker_file_(area name).icf file prepared for RFD RL78 Type 02. The Option Bytes value for RFD RL78 Type 02 is described by the “option_byte.c” file.

**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 02 (*. icf).

```
define block OPT_BYTE with size = 5   { 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[5] = {
    0x6E,   /* 01101110 */
            /* |||||||| */
            /* |||||||+-- Watchdog timer     */
            /* |||||||    operation stopped  */
            /* |||||||    in HALT/STOP mode  */
            /* ||||+++--- Watchdog timer     */
            /* ||||       overflow time is   */
            /* ||||       2^16 / fIL =       */
            /* ||||       3799.18 ms         */
            /* |||+------ Watchdog timer     */
            /* |||        operation disabled */
            /* |++------- 100% window open   */
            /* |          period             */
            /* +--------- Interval interrupt */
            /*            is not used        */
    0x6F,   /* 01101111 */
            /* |||||||| */
            /* +++|++++-- LVD reset mode */
            /*    +------ Control of clock monitor operation is enabled */
    0xE8,   /* 11101000 */
            /*   |||||| */
            /*   |+++++-- 40 MHz */
            /*   +------- Selects P130 as a general port pin (output only) */
    0xA5,   /* 10100101 */
            /* | |   || */
            /* +-|---++-- OCD: enables on-chip debugging function */
            /*   +------- Enables flash serial programming operation. */
    0xFE    /* Enables read of on-chip debug and flash serial programming security ID */
};
```

- Description of user option byte value:

  The value of User option byte (000C0H-000C2H) in “option_byte.c” file is “0x6E6FE8”.
  (WDT Stop, LVD [reset mode], 40MHz [The example for RL78/F24])

  The value of on-chip debug option byte(000C3H/040C3H) in “option_byte.c” file is “0xA5”.
  (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.

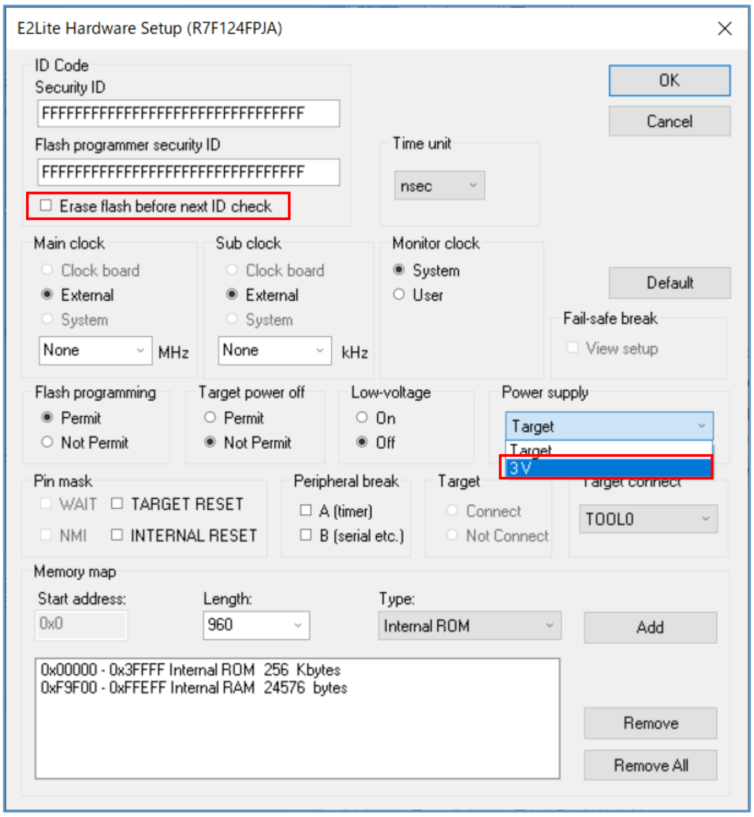
**Note: In cases where a target cannot be connected by other causes, please confirm each reference manual from [Help] of IAR Embedded Workbench.**

When selecting [Download and Debug] and starting debugging, an “E2Lite hardware setting” screen may be displayed. The cause may be ID code mismatch or power setting error.

- In the case of the ID code mismatch:
  “Cannot verify the ID code.” etc. may be displayed as a message. In this case, put a check mark to “Erase flash before next ID check” of the [ID code] in an “E2LiteHardwareSetup” 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 E2Lite), when the power is supplied to the target.**



E2Lite Hardware Setup (R7F124FPJA)

ID Code
Security ID: FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Flash programmer security ID: FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
☐ Erase flash before next ID check

Time unit: nsec

Main clock: ○ Clock board ● External ○ System None MHz
Sub clock: ○ Clock board ● External ○ System None kHz
Monitor clock: ● System ○ User
Fail-safe break: ☐ View setup

Flash programming: ● Permit ○ Not Permit
Target power off: ○ Permit ● Not Permit
Low-voltage: ○ On ● Off
Power supply: Target (Target / 3V)

Pin mask: ☐ WAIT ☐ TARGET RESET ☐ NMI ☐ INTERNAL RESET
Peripheral break: ☐ A (timer) ☐ B (serial etc.)
Target: ○ Connect ○ Not Connect
Target connect: TOOL0

Memory map
Start address: 0x0 Length: 960 Type: Internal ROM

```
0x00000 - 0x3FFFF Internal ROM  256  Kbytes
0xF9F00 - 0xFFEFF Internal RAM  24576  bytes
```

## 6.3 Setting Related to Changing Devices

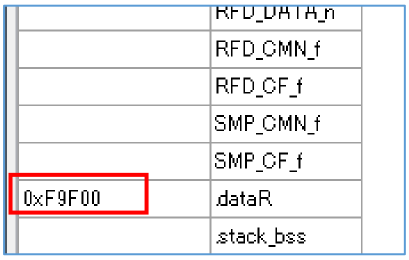
When using a device other than RL78/F24(R7F124FPJ), the address settings in the section and some of the sample programs must be modified. This section describes the where to modify and procedure to modify.

To modify the setting values, refer to "Renesas Flash Driver and EEPROM Emulation Software for RL78 Target MCU List - Automotive" (here after “Target MCU List”) and change the setting values according to the device you are using. An example of referencing the Target MCU List and an example of where to modify is shown below.

- Example of reference of the Target MCU List

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 Target MCU List and set the value of RL78/F23 (R7F123FxG).

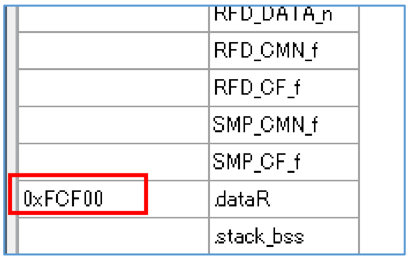
Example of where to modify the start address of RAM: RL78/F24(R7F124FPJ RAM: 24 Kbytes)



| | | |
|---|---|---|
| | RFD_DATA_n | |
| | RFD_CMN_f | |
| | RFD_CF_f | |
| | SMP_CMN_f | |
| | SMP_CF_f | |
| 0xF9F00 | dataR | |
| | stack_bss | |

[R-1] →

Example of setting the start address value of RAM when using RL78/F23 (R7F123FxG RAM: 12 Kbytes).

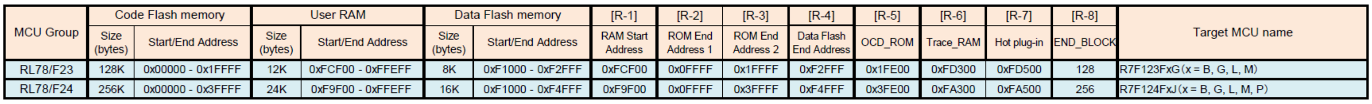


| | | |
|---|---|---|
| | RFD_DATA_n | |
| | RFD_CMN_f | |
| | RFD_CF_f | |
| | SMP_CMN_f | |
| | SMP_CF_f | |
| 0xFCF00 | dataR | |
| | stack_bss | |

The value to be set in [R-1] refers to the Target MCU List and sets the start address value of RAM of the target device.

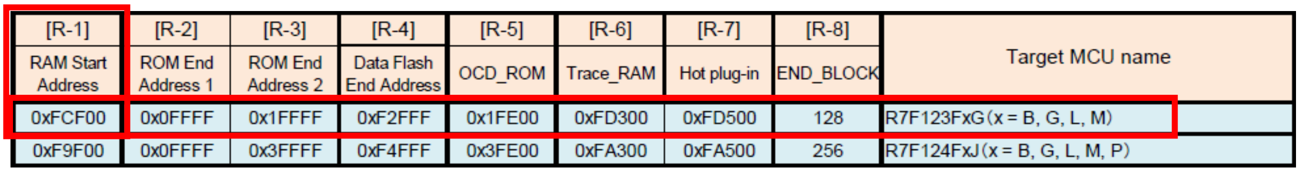
In the column "Target MCU name" of the Target MCU List, search for the row for R7F123FxG. Next, find the cell in the [R-1] column that intersects the row of R7F123FxG.

- Example of displaying the “Target MCU List”



| MCU Group | Code Flash memory | | User RAM | | Data Flash memory | | [R-1] | [R-2] | [R-3] | [R-4] | [R-5] | [R-6] | [R-7] | [R-8] | Target MCU name |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Size (bytes) | Start/End Address | Size (bytes) | Start/End Address | Size (bytes) | Start/End Address | RAM Start Address | ROM End Address 1 | ROM End Address 2 | Data Flash End Address | OCD_ROM | Trace_RAM | Hot plug-in | END_BLOCK | |
| RL78/F23 | 128K | 0x00000 - 0x1FFFF | 12K | 0xFCF00 - 0xFFEFF | 8K | 0xF1000 - 0xF2FFF | 0xFCF00 | 0x0FFFF | 0x1FFFF | 0xF2FFF | 0x1FE00 | 0xFD300 | 0xFD500 | 128 | R7F123FxG (x = B, G, L, M) |
| RL78/F24 | 256K | 0x00000 - 0x3FFFF | 24K | 0xF9F00 - 0xFFEFF | 16K | 0xF1000 - 0xF4FFF | 0xF9F00 | 0x0FFFF | 0x3FFFF | 0xF4FFF | 0x3FE00 | 0xFA300 | 0xFA500 | 256 | R7F124FxJ (x = B, G, L, M, P) |

Since "0xFCF00" applies, the setting value of [R-1] is RL78/F23 (R7F123FxG) value "0xFCF00".



| [R-1] | [R-2] | [R-3] | [R-4] | [R-5] | [R-6] | [R-7] | [R-8] | Target MCU name |
|---|---|---|---|---|---|---|---|---|
| RAM Start Address | ROM End Address 1 | ROM End Address 2 | Data Flash End Address | OCD_ROM | Trace_RAM | Hot plug-in | END_BLOCK | |
| 0xFCF00 | 0x0FFFF | 0x1FFFF | 0xF2FFF | 0x1FE00 | 0xFD300 | 0xFD500 | 128 | R7F123FxG (x = B, G, L, M) |
| 0xF9F00 | 0x0FFFF | 0x3FFFF | 0xF4FFF | 0x3FE00 | 0xFA300 | 0xFA500 | 256 | R7F124FxJ (x = B, G, L, M, P) |

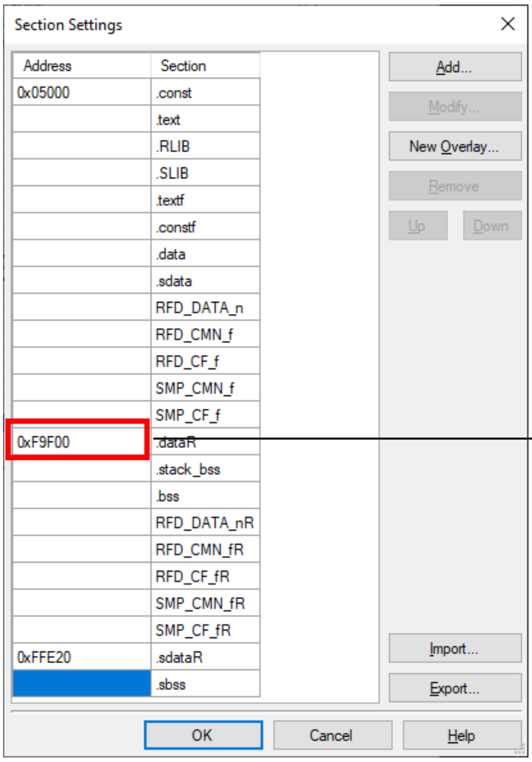
- Example of where to modify

Points that need to be modified from the RL78/F24(R7F124FPJ) settings are listed from "6.3.1". Points that need to be modified are indicated with "[R-x] →". Refer to the Target MCU List 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):

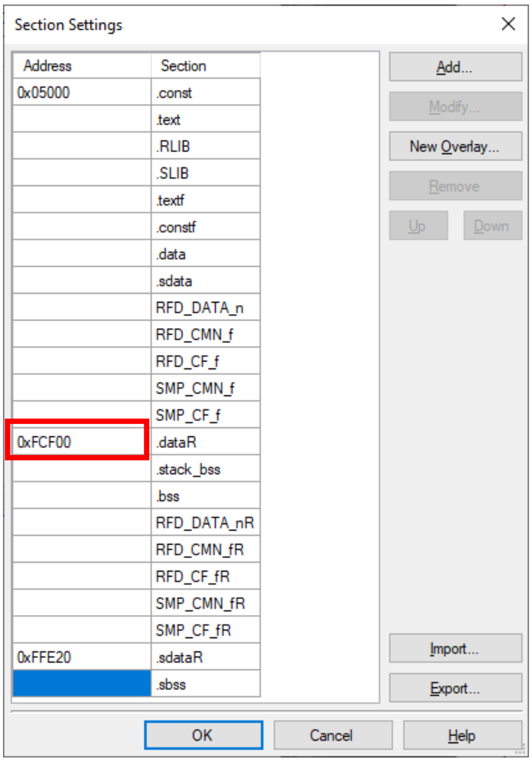
Setting for RL78/F24(RAM: 24 Kbytes)
Example: R7F124FPJ

[R-1] →



| Address | Section |
|---|---|
| 0x05000 | .const |
| | .text |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | .data |
| | .sdata |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_CF_f |
| | SMP_CMN_f |
| | SMP_CF_f |
| 0xF9F00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| | RFD_CMN_fR |
| | RFD_CF_fR |
| | SMP_CMN_fR |
| | SMP_CF_fR |
| 0xFFE20 | .sdataR |
| | .sbss |

Setting for RL78/F23(RAM: 12 Kbytes)
Example: R7F123FxG



| Address | Section |
|---|---|
| 0x05000 | .const |
| | .text |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | .data |
| | .sdata |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_CF_f |
| | SMP_CMN_f |
| | SMP_CF_f |
| 0xFCF00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| | RFD_CMN_fR |
| | RFD_CF_fR |
| | SMP_CMN_fR |
| | SMP_CF_fR |
| 0xFFE20 | .sdataR |
| | .sbss |

### 6.3.1 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.

#### 6.3.1.1 RAM Start Address Settings

Sets the starting address of RAM area.

"cstart.asm" stored in the “sample” folder is set to use a RAM size of 24 Kbytes on RL78/F24 (R7F124FxJ). Therefore, it is necessary to modify the setting when using a RAM size other than 24 Kbytes.

This section describes an example of modifying the settings when using RL78/F24 (R7F124FxJ) with a RAM size of 12 Kbytes.
To use the RL78/F23 (R7F123FxG) with a RAM size of 12 Kbytes, modify the value of the RAM start address setting register (RAMSAR) from "0x9F" to "0xCF" in "cstart.asm" stored in the "sample" folder.
For more information about the RAM Start Address Setting Register (RAMSAR), refer to the hardware manual of the target RL78.

- File path of “cstart.asm” stored in “sample” folder (area name: CF, DF, or EX_FSW)
  \sample\RL78_F24\(area name)\CCRL\source\cstart.asm

Setting for RL78/F24(RAM: 24 Kbytes) Example: R7F124FPJ

```
;--------------------------------------------------
; setting RAMSAR=0x9F RAM area is 24KB (RL78/F24)
;--------------------------------------------------
MOV !RAMSAR, #0x9F
```

Setting for RL78/F23(RAM: 12 Kbytes) Example: R7F123FxG

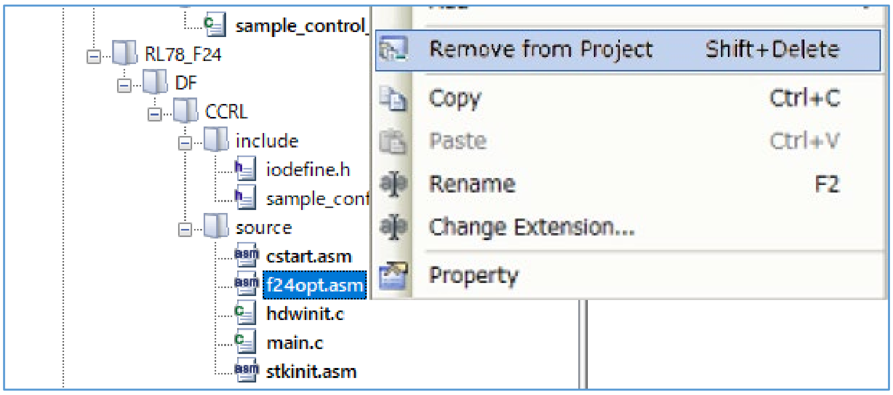
```
;--------------------------------------------------
; setting RAMSAR=0x9F RAM area is 12KB (RL78/F23)
;--------------------------------------------------
MOV !RAMSAR, #0xCF
```

#### 6.3.1.2 Exclude Unused File

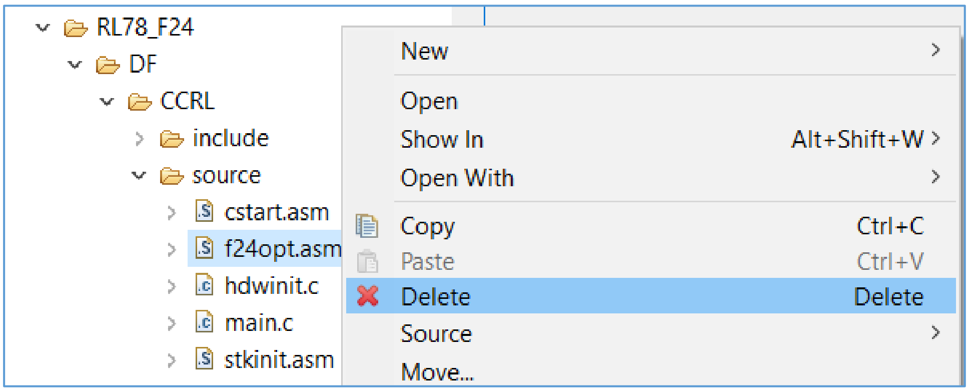
“f24opt.asm” stored in the “sample” folder is a file used only by RL78/F24, therefore, when using RL78/F23, remove “f24opt.asm” from the project.

- File path of "f24opt.asm" stored in “sample” folder (area name: CF, DF, or EX_FSW)
  \sample\RL78_F24\(area name)\CCRL\source\f24opt.asm

- Example of excluding "f24opt.asm" from a project In CS+



- Example of excluding "f24opt.asm" from a project In e$^2$studio



#### 6.3.1.3 Section Settings

Modify the start address of the RAM area in the section settings.
This example shows the change from RL78/F24 (R7F124FxJ) to RL78/F23 (R7F123FxG).
Since the RAM size is changed from 24 Kbytes to 12 Kbytes, modify the start address of the RAM from “0xF9F00” to “0xFCF00”.
Note: For the start address of the RAM for each product, refer to “R-1” column in the Target MCU List.

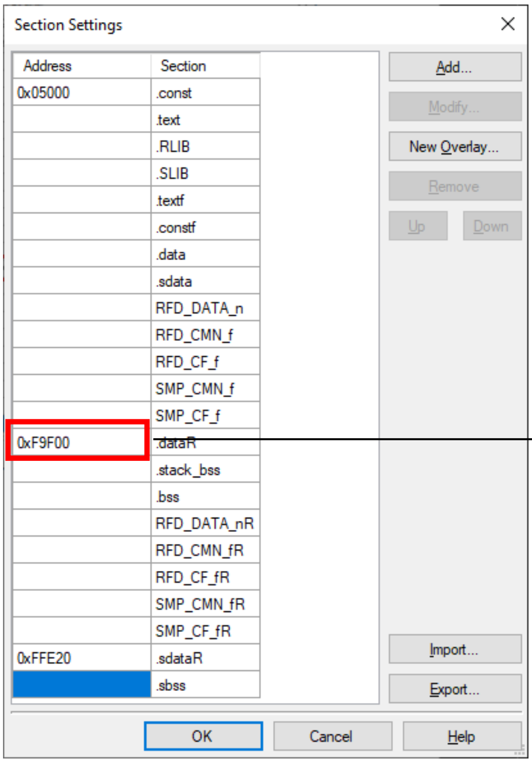
- Example of modifying CF, DF, and EX section settings (start address of the RAM) in CS+:

(1) The Case of Code Flash (CF) Memory Reprogramming

Setting for RL78/F24(RAM: 24 Kbytes)

Example: R7F124FPJ

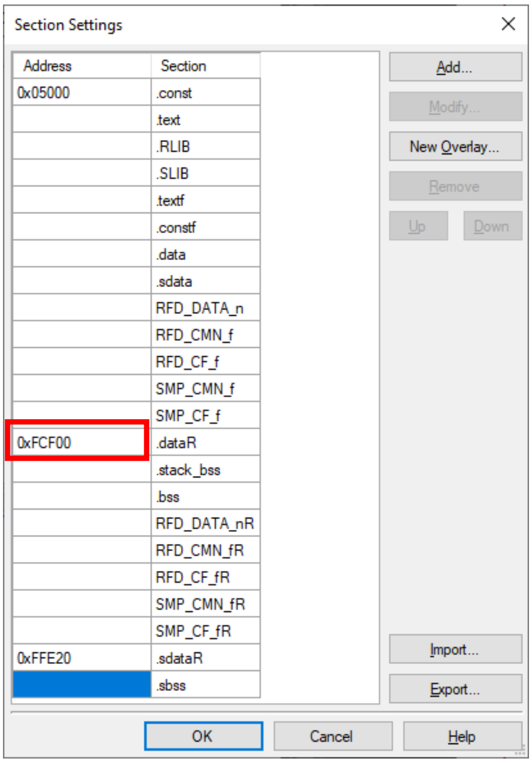
[R-1] →



| Address | Section |
|---|---|
| 0x05000 | .const |
| | .text |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | .data |
| | .sdata |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_CF_f |
| | SMP_CMN_f |
| | SMP_CF_f |
| 0xF9F00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| | RFD_CMN_fR |
| | RFD_CF_fR |
| | SMP_CMN_fR |
| | SMP_CF_fR |
| 0xFFE20 | .sdataR |
| | .sbss |

Setting for RL78/F23(RAM: 12 Kbytes)

Example: R7F123FxG



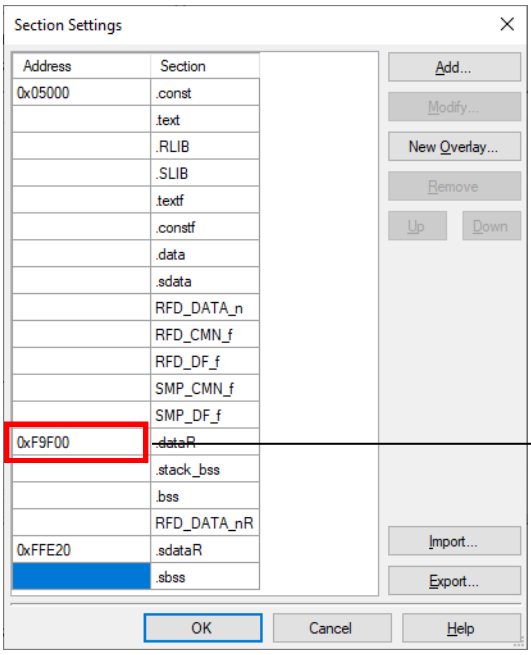
| Address | Section |
|---|---|
| 0x05000 | .const |
| | .text |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | .data |
| | .sdata |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_CF_f |
| | SMP_CMN_f |
| | SMP_CF_f |
| 0xFCF00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| | RFD_CMN_fR |
| | RFD_CF_fR |
| | SMP_CMN_fR |
| | SMP_CF_fR |
| 0xFFE20 | .sdataR |
| | .sbss |

(2) The case of data flash (DF) memory reprogramming

Setting for RL78/F24(RAM: 24 Kbytes)

Example: R7F124FPJ

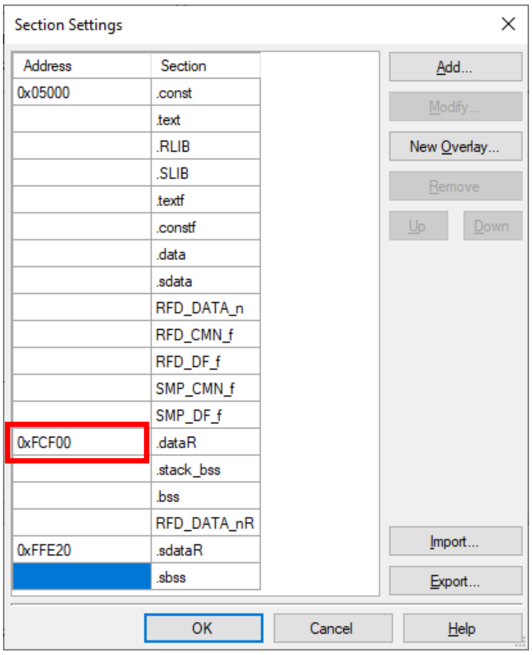
[R-1] →



| Address | Section |
|---|---|
| 0x05000 | .const |
| | .text |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | .data |
| | .sdata |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_DF_f |
| | SMP_CMN_f |
| | SMP_DF_f |
| 0xF9F00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| 0xFFE20 | .sdataR |
| | .sbss |

Setting for RL78/F23(RAM: 12 Kbytes)

Example: R7F123FxG

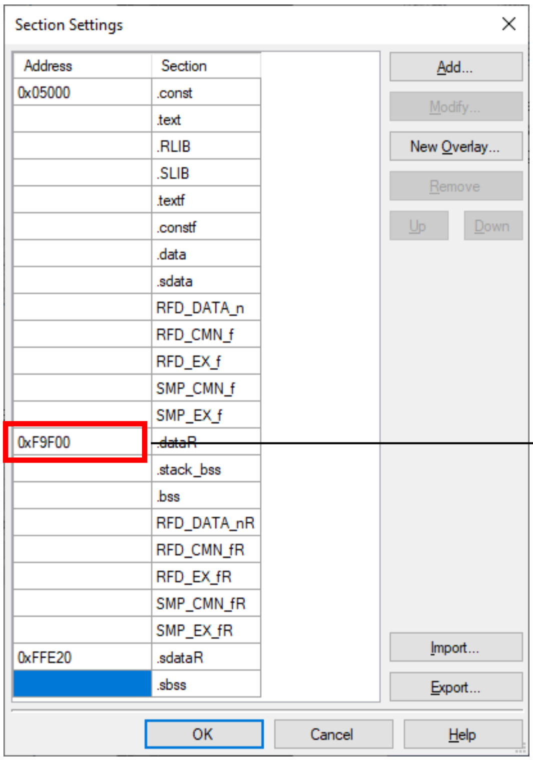


| Address | Section |
|---|---|
| 0x05000 | .const |
| | .text |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | .data |
| | .sdata |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_DF_f |
| | SMP_CMN_f |
| | SMP_DF_f |
| 0xFCF00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| 0xFFE20 | .sdataR |
| | .sbss |

(3) The case of extra area (EX) reprogramming

Setting for RL78/F24(RAM: 24 Kbytes)

Example: R7F124FPJ

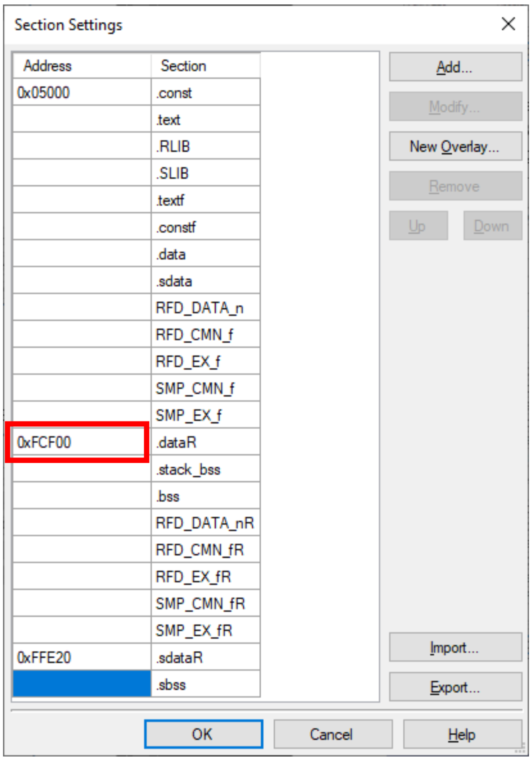

Section Settings

| Address | Section |
|---|---|
| 0x05000 | .const |
| | .text |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | .data |
| | .sdata |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_EX_f |
| | SMP_CMN_f |
| | SMP_EX_f |
| 0xF9F00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| | RFD_CMN_fR |
| | RFD_EX_fR |
| | SMP_CMN_fR |
| | SMP_EX_fR |
| 0xFFE20 | .sdataR |
| | .sbss |

[R-1] →

Setting for RL78/F23(RAM: 12 Kbytes)

Example: R7F123FxG


Section Settings

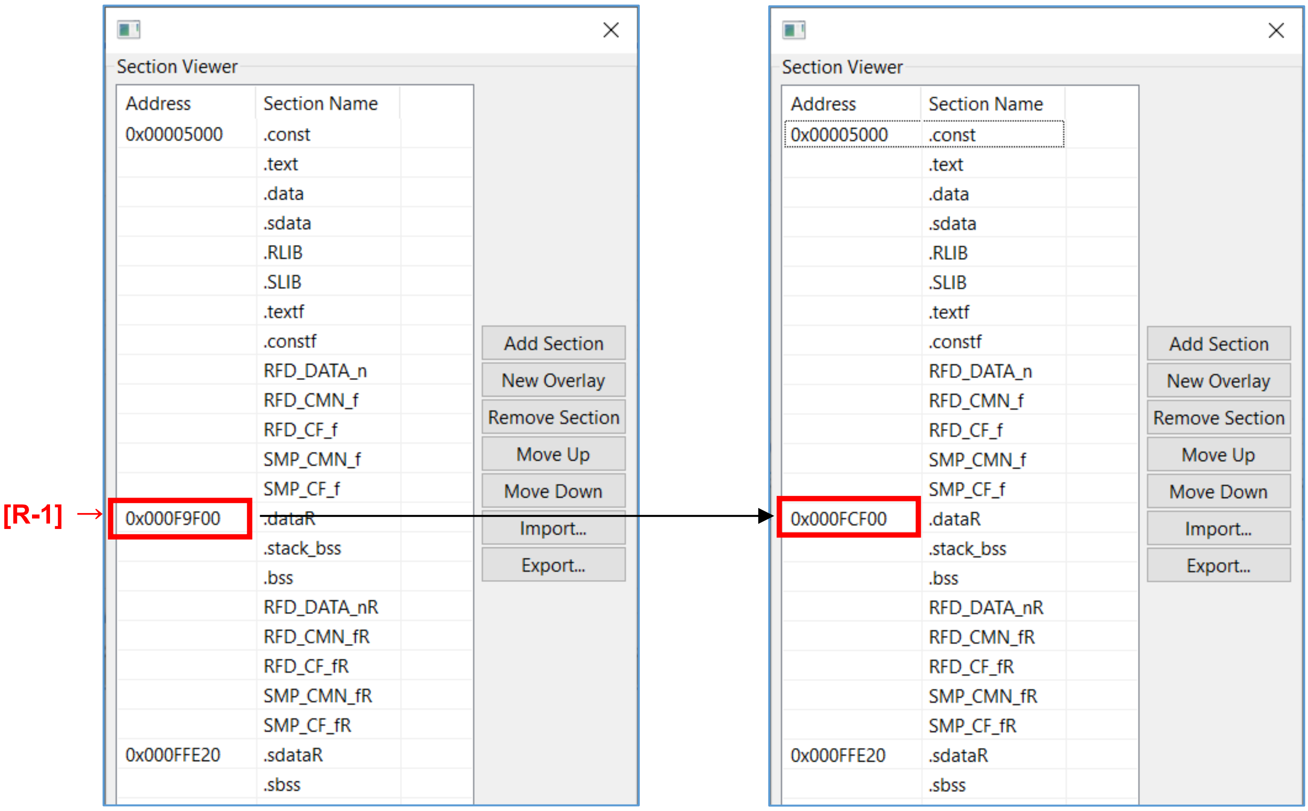
| Address | Section |
|---|---|
| 0x05000 | .const |
| | .text |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | .data |
| | .sdata |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_EX_f |
| | SMP_CMN_f |
| | SMP_EX_f |
| 0xFCF00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| | RFD_CMN_fR |
| | RFD_EX_fR |
| | SMP_CMN_fR |
| | SMP_EX_fR |
| 0xFFE20 | .sdataR |
| | .sbss |

- Example of modifying CF, DF, and EX section settings (start address of the RAM) in e$^2$studio:

(1) The Case of Code Flash (CF) Memory Reprogramming

Setting for RL78/F24(RAM: 24 Kbytes)
Example: R7F124FPJ

Setting for RL78/F23(RAM: 12 Kbytes)
Example: R7F123FxG



Setting for RL78/F24(RAM: 24 Kbytes):

| Address | Section Name |
|---|---|
| 0x00005000 | .const |
| | .text |
| | .data |
| | .sdata |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_CF_f |
| | SMP_CMN_f |
| | SMP_CF_f |
| 0x000F9F00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| | RFD_CMN_fR |
| | RFD_CF_fR |
| | SMP_CMN_fR |
| | SMP_CF_fR |
| 0x000FFE20 | .sdataR |
| | .sbss |

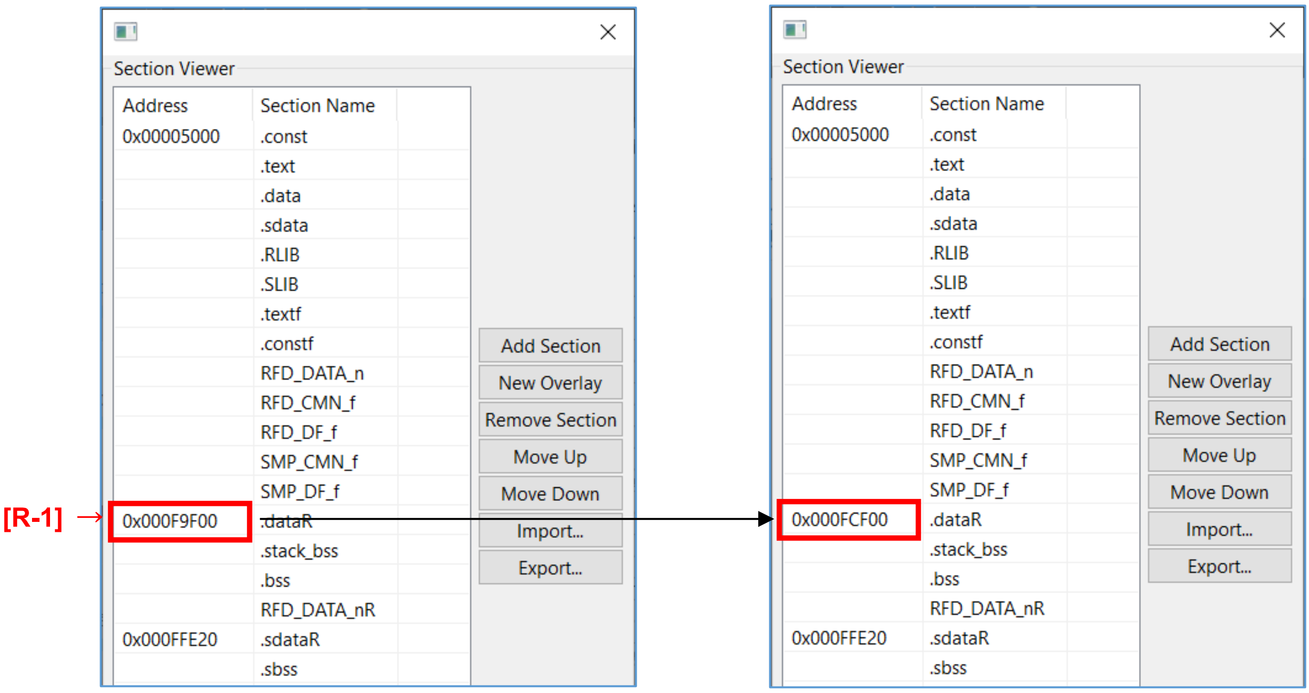
Setting for RL78/F23(RAM: 12 Kbytes):

| Address | Section Name |
|---|---|
| 0x00005000 | .const |
| | .text |
| | .data |
| | .sdata |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_CF_f |
| | SMP_CMN_f |
| | SMP_CF_f |
| 0x000FCF00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| | RFD_CMN_fR |
| | RFD_CF_fR |
| | SMP_CMN_fR |
| | SMP_CF_fR |
| 0x000FFE20 | .sdataR |
| | .sbss |

(2) The Case of Data Flash (DF) Memory Reprogramming

Setting for RL78/F24(RAM: 24 Kbytes)
Example: R7F124FPJ

Setting for RL78/F23(RAM: 12 Kbytes)
Example: R7F123FxG



Setting for RL78/F24(RAM: 24 Kbytes):

| Address | Section Name |
|---|---|
| 0x00005000 | .const |
| | .text |
| | .data |
| | .sdata |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_DF_f |
| | SMP_CMN_f |
| | SMP_DF_f |
| 0x000F9F00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| 0x000FFE20 | .sdataR |
| | .sbss |

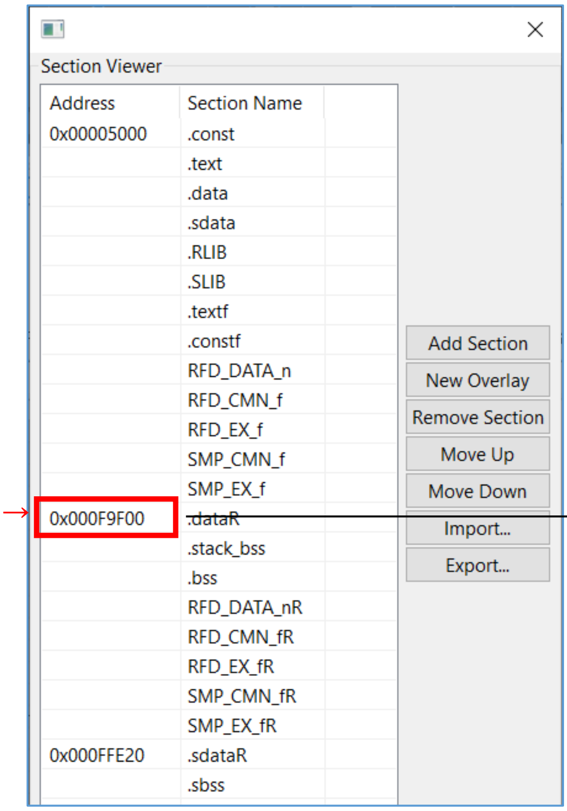
Setting for RL78/F23(RAM: 12 Kbytes):

| Address | Section Name |
|---|---|
| 0x00005000 | .const |
| | .text |
| | .data |
| | .sdata |
| | .RLIB |
| | .SLIB |
| | .textf |
| | .constf |
| | RFD_DATA_n |
| | RFD_CMN_f |
| | RFD_DF_f |
| | SMP_CMN_f |
| | SMP_DF_f |
| 0x000FCF00 | .dataR |
| | .stack_bss |
| | .bss |
| | RFD_DATA_nR |
| 0x000FFE20 | .sdataR |
| | .sbss |

(3) The Case of Extra Area (EX) Reprogramming

Setting for RL78/F24(RAM: 24 Kbytes)
Example: R7F124FPJ

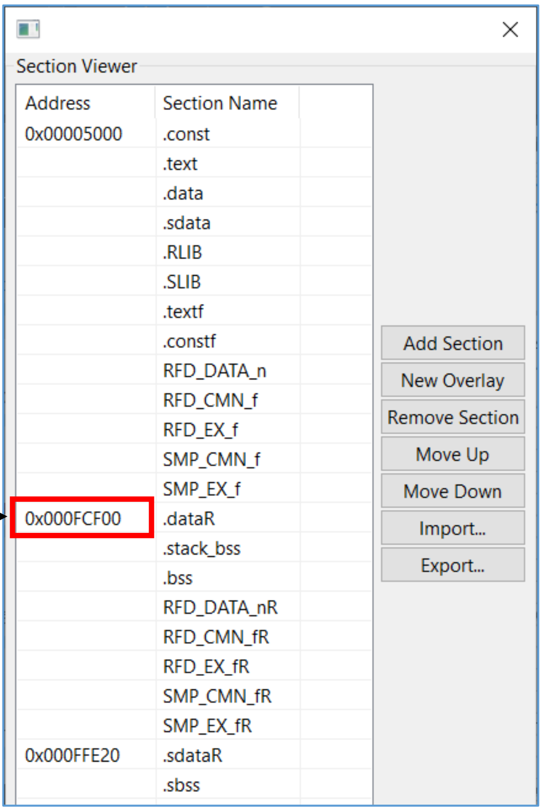
[R-1] →



Section Viewer

| Address | Section Name |
|---|---|
| 0x00005000 | .const |
|  | .text |
|  | .data |
|  | .sdata |
|  | .RLIB |
|  | .SLIB |
|  | .textf |
|  | .constf |
|  | RFD_DATA_n |
|  | RFD_CMN_f |
|  | RFD_EX_f |
|  | SMP_CMN_f |
|  | SMP_EX_f |
| 0x000F9F00 | .dataR |
|  | .stack_bss |
|  | .bss |
|  | RFD_DATA_nR |
|  | RFD_CMN_fR |
|  | RFD_EX_fR |
|  | SMP_CMN_fR |
|  | SMP_EX_fR |
| 0x000FFE20 | .sdataR |
|  | .sbss |

Setting for RL78/F23(RAM: 12 Kbytes)
Example: R7F123FxG



Section Viewer

| Address | Section Name |
|---|---|
| 0x00005000 | .const |
|  | .text |
|  | .data |
|  | .sdata |
|  | .RLIB |
|  | .SLIB |
|  | .textf |
|  | .constf |
|  | RFD_DATA_n |
|  | RFD_CMN_f |
|  | RFD_EX_f |
|  | SMP_CMN_f |
|  | SMP_EX_f |
| 0x000FCF00 | .dataR |
|  | .stack_bss |
|  | .bss |
|  | RFD_DATA_nR |
|  | RFD_CMN_fR |
|  | RFD_EX_fR |
|  | SMP_CMN_fR |
|  | SMP_EX_fR |
| 0x000FFE20 | .sdataR |
|  | .sbss |

#### 6.3.1.4 Debug Settings

When using the RL78/F23, 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 "0x3FFFF", set it to "0x3FE00".

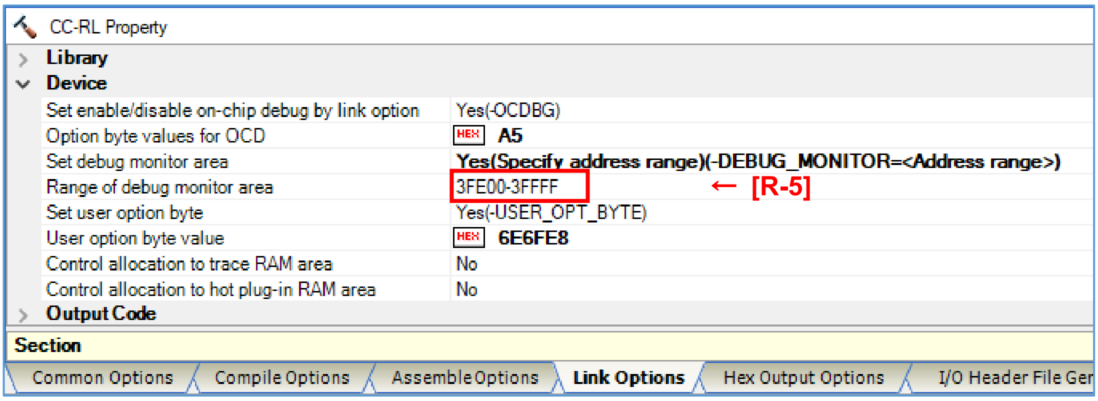
This example shows the modify from RL78/F24 (R7F124FPJ) to RL78/F23 (R7F123FMG).

- Set the debug monitor area range to "0x1FE00 - 0x1FFFF" for the RL78/F23 (R7F123FMG).

Note: For information on The start address of the "debug monitor area" for each product, refer to "[R-5]" column in the Target MCU List.

- To set the debug monitor area in CS+, select the [Device] on the "Link Options" tab.

Setting for RL78/F24 (ROM: 256 Kbytes) Example: R7F124FPJ

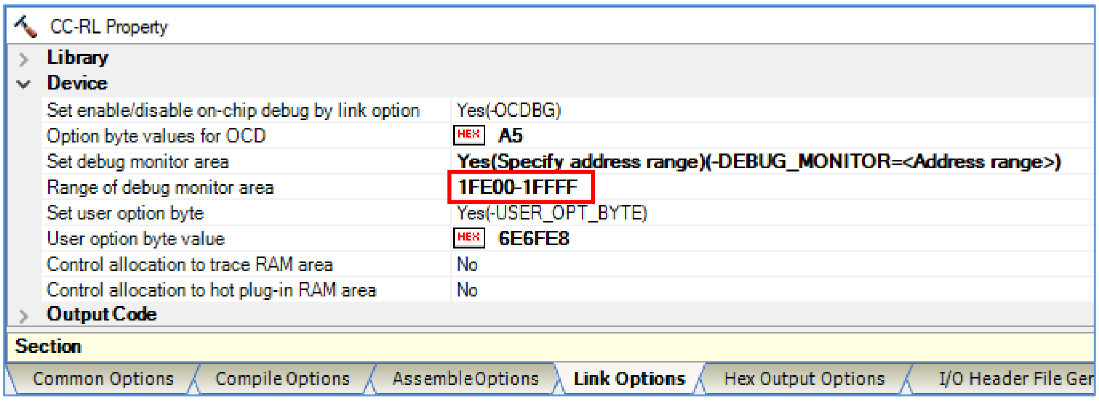


CC-RL Property

| | |
|---|---|
| Library | |
| Device | |
| Set enable/disable on-chip debug by link option | Yes(-OCDBG) |
| Option byte values for OCD | HEX A5 |
| Set debug monitor area | Yes(Specify address range)(-DEBUG_MONITOR=<Address range>) |
| Range of debug monitor area | 3FE00-3FFFF ← [R-5] |
| Set user option byte | Yes(-USER_OPT_BYTE) |
| User option byte value | HEX 6E6FE8 |
| Control allocation to trace RAM area | No |
| Control allocation to hot plug-in RAM area | No |
| Output Code | |

Section

Setting for RL78/F23 (ROM: 128 Kbytes) Example: R7F123FMG



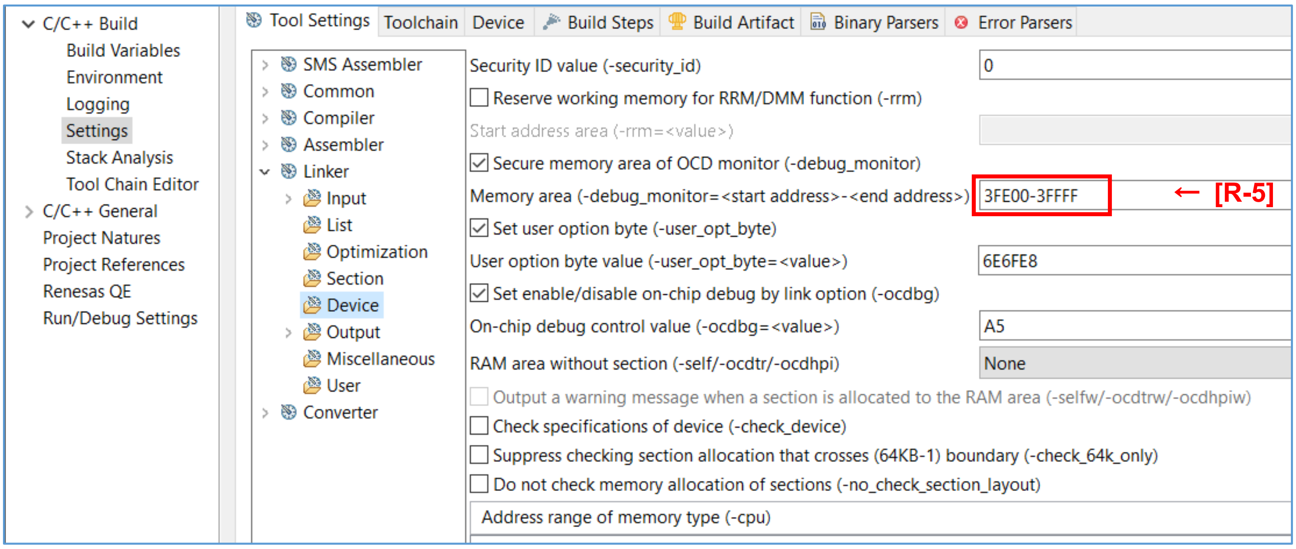
CC-RL Property

| | |
|---|---|
| Library | |
| Device | |
| Set enable/disable on-chip debug by link option | Yes(-OCDBG) |
| Option byte values for OCD | HEX A5 |
| Set debug monitor area | Yes(Specify address range)(-DEBUG_MONITOR=<Address range>) |
| Range of debug monitor area | 1FE00-1FFFF |
| Set user option byte | Yes(-USER_OPT_BYTE) |
| User option byte value | HEX 6E6FE8 |
| Control allocation to trace RAM area | No |
| Control allocation to hot plug-in RAM area | No |
| Output Code | |

Section

- To set the area of OCD monitor in e$^2$studio, select the [Device] in the “Linker”.

Setting for RL78/F24 (ROM: 256 Kbytes) Example: R7F124FPJ



Security ID value (-security_id): 0

☐ Reserve working memory for RRM/DMM function (-rrm)

Start address area (-rrm=<value>)

☑ Secure memory area of OCD monitor (-debug_monitor)

Memory area (-debug_monitor=<start address>-<end address>): 3FE00-3FFFF ← [R-5]

☑ Set user option byte (-user_opt_byte)

User option byte value (-user_opt_byte=<value>): 6E6FE8

☑ Set enable/disable on-chip debug by link option (-ocdbg)

On-chip debug control value (-ocdbg=<value>): A5

RAM area without section (-self/-ocdtr/-ocdhpi): None

☐ Output a warning message when a section is allocated to the RAM area (-selfw/-ocdtrw/-ocdhpiw)

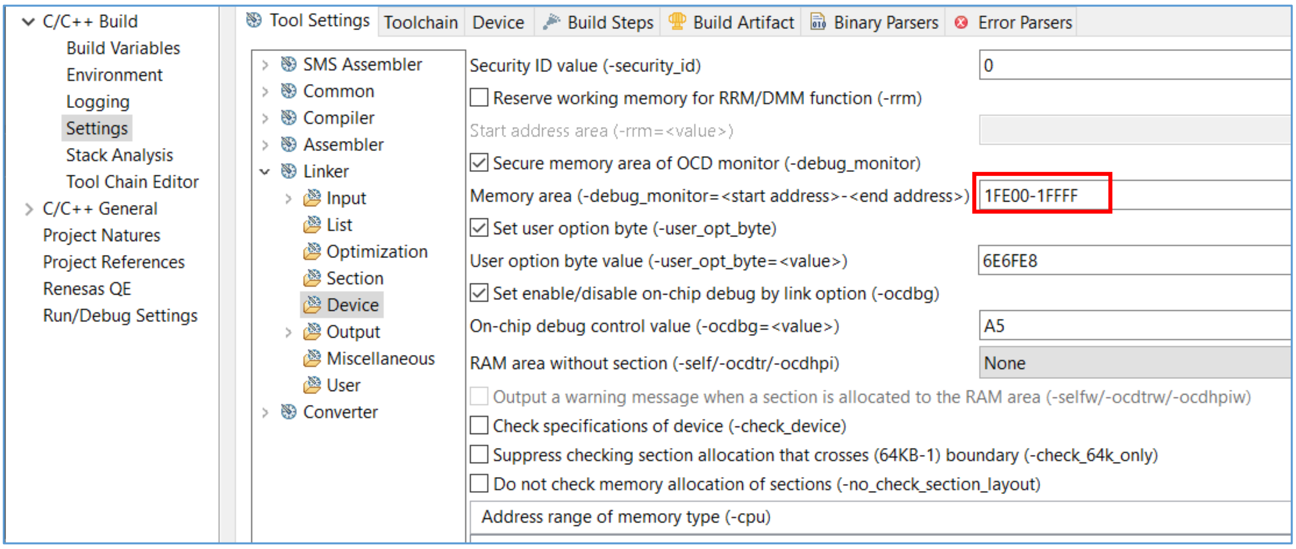
☐ Check specifications of device (-check_device)

☐ Suppress checking section allocation that crosses (64KB-1) boundary (-check_64k_only)

☐ Do not check memory allocation of sections (-no_check_section_layout)

Address range of memory type (-cpu)

Setting for RL78/F23 (ROM: 128 Kbytes) Example: R7F123FMG



Security ID value (-security_id): 0

☐ Reserve working memory for RRM/DMM function (-rrm)

Start address area (-rrm=<value>)

☑ Secure memory area of OCD monitor (-debug_monitor)

Memory area (-debug_monitor=<start address>-<end address>): 1FE00-1FFFF

☑ Set user option byte (-user_opt_byte)

User option byte value (-user_opt_byte=<value>): 6E6FE8

☑ Set enable/disable on-chip debug by link option (-ocdbg)

On-chip debug control value (-ocdbg=<value>): A5

RAM area without section (-self/-ocdtr/-ocdhpi): None

☐ Output a warning message when a section is allocated to the RAM area (-selfw/-ocdtrw/-ocdhpiw)

☐ Check specifications of device (-check_device)

☐ Suppress checking section allocation that crosses (64KB-1) boundary (-check_64k_only)

☐ Do not check memory allocation of sections (-no_check_section_layout)

Address range of memory type (-cpu)

### 6.3.2 IAR Compiler Environment Settings

Points of modifies and examples of modifies when using the IAR compiler environment (Embedded Workbench) is described.

#### 6.3.2.1 Setting up Header files for Target Device

The “main.c” and “low_level_init.c” provided with RFD RL78 Type 02 V1.00 includes the header files for the target device “RL78/F24: R7F124FPJ”. When using other RL78/F24 products or RL78/F23 products, the included header file must be changed to the header file for the device used.
This section describes when RL78/F23 (R7F123FMG) is used.

Target files name: main.c, low_level_init.c

- For RL78/F24 (R7F124FPJ):

< main.c >

```
#include "ior7f124fpj.h"
```

< low_level_init.c >

```
#include "ior7f124fpj.h"
#include "ior7f124fpj_ext.h"
```

- Example for RL78/F23 (R7F123FMG):

< main.c >

```
#include "ior7f123fmg.h"
```

< low_level_init.c >

```
#include "ior7f123fmg.h"
#include "ior7f123fmg_ext.h"
```

Note: For the device type name of the product, refer to “Target MCU name” column in the Target MCU List.

#### 6.3.2.2 Linker Configuration File Settings

In the sample program provided by RFD RL78 Type 02, The sections (ROM, RAM, and Data flash range) for RL78/F24 (R7F124FPJ) are set.
When using other RL78/F24 products, modify the contents of the linker configuration file “sample_linker_file_xx.icf (xx = CF or DF or EX_FSW)” provided for the RL78/F24 of RFD RL78 Type 02 V1.00 because the section settings are different. The modifications are shown in red text below, so refer to the Target MCU List and change the setting values for the target device.

Target files name: sample_linker_file_xx.icf (xx = CF or DF or EX_FSW)

This example shows the modify from RL78/F24 (R7F124FPJ) to RL78/F23 (R7F123FMG).

- Modify the ROM area to the range of 128 Kbytes [0x00000 - 0x1FFFF]
- Modify the start address to “0xFCF00” because the RAM area is 12 Kbytes [0x0FCF00 - 0x0FFEFF]
- Modify the end address to “0xF2FFF” because the data flash area is 8 Kbytes [0x0F1000 - 0x0F2FFF]

### (1) Section Settings

- sample_linker_file_CF.icf, and sample_linker_file_EX_FSW.icf

Setting for RL78/F24 (ROM: 256 Kbytes, RAM: 24 Kbytes, Data Flash: 16 Kbytes) Example: R7F124FPJ

```
define region ROM_near = mem:[from 0x00132 to 0x0FFFF];  ← [R-2]
define region ROM_far = mem:[from 0x00132 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF ]
                          | mem:[from 0x20000 to 0x2FFFF ] | mem:[from 0x30000 to 0x3FFFF ];← [R-2], [R-3] Notes 1
define region ROM_huge = mem:[from 0x00132 to 0x3FFFF ];  ← [R-2] or [R-3] Notes 2
define region SADDR = mem:[from 0xFFE20 to 0xFFEDF];
define region RAM_near = mem:[from 0xF9F00 to 0xFFE1F];  ← [R-1]
define region RAM_far = mem:[from 0xF9F00 to 0xFFE1F];  ← [R-1]
define region RAM_code = mem:[from 0xF9F00 to 0xFFE1F];  ← [R-1]
define region RAM_huge = mem:[from 0xF9F00 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 0xF4FFF ];  ← [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 list. In the case of “-“, set the value of [R-2].

Setting for RL78/F23 (ROM: 128 Kbytes, RAM: 12 Kbytes, Data Flash: 8 Kbytes) Example: R7F123FMG

```
define region ROM_near = mem:[from 0x00132 to 0x0FFFF];
define region ROM_far = mem:[from 0x00132 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF ];
define region ROM_huge = mem:[from 0x00132 to 0x1FFFF ];
define region SADDR = mem:[from 0xFFE20 to 0xFFEDF];
define region RAM_near = mem:[from 0xFCF00 to 0xFFE1F];
define region RAM_far = mem:[from 0xFCF00 to 0xFFE1F];
define region RAM_code = mem:[from 0xFCF00 to 0xFFE1F];
define region RAM_huge = mem:[from 0xFCF00 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 ];
```

- sample_linker_file_DF.icf

Setting for RL78/F24 (ROM: 256 Kbytes, RAM: 24 Kbytes, Data Flash: 16 Kbytes) Example: R7F124FPJ

```
define region ROM_near = mem:[from 0x00132 to 0x0FFFF]; ← [R-2]
define region ROM_far = mem:[from 0x00132 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF ]
                                        | mem:[from 0x20000 to 0x2FFFF ] | mem:[from 0x30000 to 0x3FFFF ];← [R-2], [R-3] Notes 1
define region ROM_huge = mem:[from 0x00132 to 0x3FFFF ]; ← [R-2] or [R-3] Notes 2
define region SADDR = mem:[from 0xFFE20 to 0xFFEDF];
define region RAM_near = mem:[from 0xF9F00 to 0xFFE1F];← [R-1]
define region RAM_far = mem:[from 0xF9F00 to 0xFFE1F];   ← [R-1]
define region RAM_huge = mem:[from 0xF9F00 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 0xF4FFF ]; ← [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 list. In the case of “-”, set the value of [R-2].

Setting for RL78/F23 (ROM: 128KB, RAM: 12KB, Data Flash: 8KB) Example: R7F123FMG

```
define region ROM_near = mem:[from 0x00132 to 0x0FFFF];
define region ROM_far = mem:[from 0x00132 to 0x0FFFF] | mem:[from 0x10000 to 0x1FFFF ];
define region ROM_huge = mem:[from 0x00132 to 0x1FFFF ];
define region SADDR = mem:[from 0xFFE20 to 0xFFEDF];
define region RAM_near = mem:[from 0xFCF00 to 0xFFE1F];
define region RAM_far = mem:[from 0xFCF00 to 0xFFE1F];
define region RAM_huge = mem:[from 0xFCF00 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 ];
```

### (2) Debug Settings

- The first 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 “0x3FFFF”, set it to “0x3FE00”.

- The first address of the TraceRAM area is set by adding "1 Kbyte(0x400))" to the first address of the RAM area. If the first address is "0xF9F00", set "0xFA300".

- The first address of the hot plug-in RAM area is set by adding "0x600" to the first address of the RAM area. If the first address is "0xF9F00", set "0xFA500".

As an example, modifying from RL78/F24 (R7F124FPJ) to RL78/F23 (R7F123FMG) is shown.

- Set the debug monitor area range to [from 0x1FE00 size 0x0200].
- Set the TraceRAM area range to [from 0xFD300 size 0x0200].
- Set the hot plug-in RAM area range to [from 0xFD500 size 0x0030].

- Modifies to the TraceRAM area, debug monitor area, and hot plug-in RAM area when using the debugger.

Setting for RL78/F24 (ROM: 256 Kbytes, RAM: 24 Kbytes) Example: R7F124FPJ

```
if (isdefinedsymbol(__RESERVE_OCD_ROM))
{
        if (__RESERVE_OCD_ROM == 1)
        {
        reserve region "OCD ROM area" = mem:[from 0x3FE00 size 0x0200];  ← [R-5]
        }
}
        [Omitted]
if (isdefinedsymbol(__RESERVE_OCD_TRACE_RAM))
{
        if (__RESERVE_OCD_TRACE_RAM == 1)
        {
        reserve region "OCD Trace RAM" = mem:[from 0xFA300 size 0x0200]; ← [R-6]
        }
}
        [Omitted]
if (isdefinedsymbol(__RESERVE_HOTPLUGIN_RAM))
{
   if (__RESERVE_HOTPLUGIN_RAM == 1)
   {
      reserve region "Hot Plugin RAM" = mem:[from 0xFA500 size 0x0030];  ← [R-7]
   }
}
```

Setting for RL78/F23 (ROM: 128 Kbytes, RAM: 12 Kbytes, Data Flash: 8 Kbytes) Example: R7F123FMG

```
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 0xFD300 size 0x0200];
        }
}
        [Omitted]
if (isdefinedsymbol(__RESERVE_HOTPLUGIN_RAM))
{
   if (__RESERVE_HOTPLUGIN_RAM == 1)
   {
      reserve region "Hot Plugin RAM" = mem:[from 0xFD500 size 0x0030];
   }
}
```

### (3) RAM Start Address Settings

Sets the starting address of RAM area.

“sample_linker_file_xx.icf (xx = CF or DF or EX_FSW)” is set to use a RAM size of 24 Kbytes on RL78/F24 (R7F124FPJ). Therefore, it is necessary to modify the setting when using a RAM size other than 24 Kbytes.

This section describes an example of modifying the settings when using RL78/F24 (R7F124FPJ) with a RAM size of 12 Kbytes.
To use RL78/F23 (R7F123FMG) with a RAM size of 12 Kbytes, modify the value of the RAM start address setting register (RAMSAR) from “0x9F” to “0xCF” in “cstart.asm” stored in the “sample” folder.
For more information about the RAM Start Address Setting Register (RAMSAR), refer to the hardware manual of the target RL78.

Setting for RL78/F24(RAM: 24 Kbytes) Example: R7F124FPJ

```
define exported symbol _RAMSAR_ADDR = 0xF0076;

if (!isdefinedsymbol(__RAMSAR_VAL))

{

    define exported symbol _RAMSAR_VAL = 0x9F;

}
```

Setting for RL78/F23(RAM: 12 Kbytes) Example: R7F123FMG

```
define exported symbol _RAMSAR_ADDR = 0xF0076;

if (!isdefinedsymbol(__RAMSAR_VAL))

{

    define exported symbol _RAMSAR_VAL = 0xCF;

}
```

### 6.3.3 Modifies in the Sample Program (Common to Compilers)

#### 6.3.3.1 Extra Area [FSW Range] Modifying the Reprogramming Sample Program

The number of the maximum blocks of the code flash of RL78/F23 differs from RL78/F24. 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-8])” of the FSW range. Also, the comment contains the value of “End block number ([R-8]) of the FSW range” and “End block number + 1 ([R-8]) of the FSW range”.

Example) For RL78/F24 (R7F124FPJ), [R-8] is 256 and “[R-8] -1” is 255.
As an example, modifying from RL78/F24 (R7F124FPJ) to RL78/F23 (R7F123FMG) is shown.

File Name: sample_config.h
File Path: \sample\RL78_F24\EX_FSW\IAR\include

Setting for RL78/F24 (ROM: 256 Kbytes) Example: R7F124FPJ

```
/*****************************************************************************************************************
   User configurable parameters
*****************************************************************************************************************/

/**** CPU frequency (MHz) ****/
/* It must be rounded up digits after the decimal point to form an integer (MHz). */
#define CPU_FREQUENCY (40u)

/**** 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-8]-1, [R-8]
#define END_BLOCK         (256u)    ← [R-8]
```

Setting for RL78/F23 (ROM: 128 Kbytes) Example: R7F123FMG

```
/*****************************************************************************************************************
   User configurable parameters
*****************************************************************************************************************/

/**** CPU frequency (MHz) ****/
/* It must be rounded up digits after the decimal point to form an integer (MHz). */
#define CPU_FREQUENCY (40u)

/**** 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 127 is the end of range for FSW, please specify 128. */
#define END_BLOCK         (128u)
```

# 7. Revision History

## 7.1 Major Modifications in this Revision

| Rev. | Date | Description | |
|---|---|---|---|
| | | Page | Summary |
| 1.00 | Jul.6.22 | - | Newly created. |
| 1.10 | Dec.28.22 | - | Add support of RL78/F23. |

Renesas Flash Driver RL78 Type 02 User's Manual

Publication Date: Rev.1.10 Dec.28.22

Published by: Renesas Electronics Corporation

Renesas Flash Driver

RL78 Type 02

Renesas Electronics Corporation