User's Manual

RL78 Family
EEPROM Emulation Software
RL78 Type 03

User's Manual

RENESAS Microcontrollers
RL78 / F22
RL78 / F25

Renesas Electronics
www.renesas.com

Rev.1.01 May 2025

Notice

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

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Trademarks

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

Contact information

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

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 users who wish to understand the features of the RL78 microcontrollers EEPROM Emulation and to use the EEPROM Emulation Software (EES) RL78 Type 03 in designing and developing application systems.

- Purpose

  This manual is intended to give users an understanding of the methods for using the EEPROM Emulation Software (EES) RL78 Type 03 to reprogram the data flash memory in the RL78/F22, F25 microcontrollers (i.e. write constant data by the application).

- Organization

  This manual is separated into the following sections.

  - Overview
  - System Configuration
  - EEPROM Emulation
  - Using EEPROM Emulation
  - User Interface
  - Sample Programs
  - Creating a Sample Project for EES RL78 Type 03

- 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/F22, F25:

  - Refer to the User's Manual of the target RL78/F22, F25 devices.

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

    Binary: ×××× or ××××B

    Decimal: ××××

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

    K (kilo) 2^{10} = 1024

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

- Related Documents

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

No	Document Title	Document Number
1	RL78/F22, F25 User's Manual Hardware	R01UH1061EJ
2	RL78 Family Renesas Flash Driver RL78 Type 03 User's Manual	R20UT5454EJ
3	E1/E20/E2 Emulator, E2 Emulator Lite Additional Document for User's Manual (Notes on Connection of RL78)	R20UT1994EJ

Table of Contents

1. Overview 10
1.1 Outline 10
1.1.1 Purpose 10
1.2 Contents 10
1.3 Features 11
1.4 Operating Environment 12
1.5 Points for Caution 13
1.6 C Compiler Definitions 16

2. System Configuration 17
2.1 System Configuration 17
2.2 EES Architecture 17
2.2.1 EES Block 17
2.2.2 EES Pool 18
2.3 File Structure 19
2.3.1 Folder Structure 19
2.3.2 List of Files 20
2.4 Resources of RL78/F22, F25 21
2.4.1 Memory Map 21
2.4.2 Allocation of Blocks 22
2.5 Resources Used in EES RL78 Type 03 23
2.5.1 Sections Used in EES RL78 Type 03 23
2.5.2 Software Resources 23

3. EEPROM Emulation 24
3.1 Specifications of EEPROM Emulation 24
3.2 Outline of Functions 24
3.3 EES Pool 25
3.3.1 EES Pool State 25
3.3.2 Structure of EES Block 27
3.3.3 EES Block Header 28
3.3.4 Structure of Stored Data 29
3.3.5 EES Block Overview 31

4. Using EEPROM Emulation 32
4.1 Number of Stored User Data Items and Total User Data Size 32
4.2 Initial Values to be Set by User 33

5. User Interface 36
5.1 Request Structure (st_ees_request_t) Settings 36
5.1.1 User Write Access 37
5.1.2 User Read Access 37
5.2 List of API Functions and R_EES_Execute function commands for the EES 38
5.2.1 API Functions for the EES 38
5.2.2 Commands for R_EES_Execute Function 39
5.2.3 RFD control API functions for EES 40
5.3 State Transitions 41
5.4 Basic Flowchart 43
5.5 Command Operation Flowchart 45

5.6 Data Type Definitions 46
5.6.1 Data Types 46
5.6.2 Global Variables 46
5.6.3 Enumerations 48
5.7 Specifications of API Functions 50
5.7.1 Specifications of API Functions for EES RL78 Type 03 51
5.7.2 RFD control API Functions for EES 59
5.7.3 Internal Functions for the EES 61

6. Sample Programs 64
6.1 File Structure 64
6.1.1 Folder Structure 64
6.1.2 List of Files 65
6.2 Data Type Definitions 65
6.2.1 Macro Defines 65
6.3 Sample Program Functions 66
6.3.1 Sample Program for Controlling the EEPROM Emulation 66
6.4 Specifications of Sample Program Functions 72
6.4.1 Sample Program Functions for Controlling the EEPROM Emulation 72

7. Creating a Sample Project for EES RL78 Type 03 74
7.1 Creating a Project in the Case of Using a CC-RL Compiler 74
7.1.1 Example of Creating a Sample Project 75
7.1.2 Example of Registration of Target Folders and Target Files 78
7.1.3 Build Tool Settings 80
7.1.4 Debug Tool Settings 87
7.2 Creating a Project in the Case of Using IAR Compiler 89
7.2.1 Example of Creating a Sample Project 90
7.2.2 Example of Registration of Target Folders and Target Files 92
7.2.3 Integrated Development Environment (IDE) Settings 94
7.2.4 Linker Configuration File (.icf) Settings 96
7.2.5 On-chip Debug Settings 99
7.3 Configurations Modify Procedure for Changing Device 100
7.3.1 CC-RL Compiler Environment Settings 102
7.3.2 IAR Compiler Environment Settings 106

8. Revision History 112
8.1 Major Modifications in this Revision 112

Abbreviations

Abbreviation	Description
EES	EEPROM Emulation Software
RFD	Renesas Flash Driver
API	Application Program Interface
BGO	Background Operation Instructions in the code flash memory can be executed during reprogramming of the data flash memory.
RAM	Random Access Memory Randomly accessible volatile memory. It is memory for holding values that are to be changed during program execution.
ROM	Read-Only Memory Non-volatile memory. It is memory whose contents cannot be changed. The code flash memory may be called ROM.

Terminology

Terminology	Description
Code flash memory	Flash memory for storing application code and constant data. Note that this memory may be abbreviated as “CF” in this document.
Data flash memory	Flash memory for storing data. Note that this memory may be abbreviated as “DF” in this document.
Extra area	Generic name of the configuration setting area, security setting area, lock 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. - Code flash memory programming mode - Data flash memory programming mode - Non-programmable mode
Code flash memory programming mode	The code flash memory (and extra area) can be reprogrammed in this mode.
Data flash memory programming mode	The data flash memory can be reprogrammed in this mode.
Non-programmable mode	The flash memory (and extra area) cannot be reprogrammed in this mode.
Self-programming	A method of reprogramming the flash memory by executing a user program instead of using an external flash memory programming tool.
RFD function	A generic term for the functions offered by the RFD.
EES function	A generic term for the functions offered by the EES.
RFD control functions for EES	A generic term for the RFD control functions offered by the EES.
EES Block	An abbreviation of blocks that the EEPROM emulation software accesses. In this user’s manual, EEPROM emulation blocks are hereafter referred to as EES block.

1. Overview

1.1 Outline

EEPROM emulation is a feature used to store data in the on-board flash memory in the same way as EEPROM. In EEPROM emulation, EEPROM Emulation Software RL78 Type 03 operates the Renesas Flash Driver (RFD) RL78 Type 03. And RFD writes and reads the data flash memory.

EEPROM Emulation Software RL78 Type 03 (hereafter called EES RL78 Type 03) is software for reprogramming the data flash memory in the RL78/F22, F25.

For information on Renesas Flash Driver (RFD) RL78 Type 03, refer to the RL78 Family Renesas Flash Driver RL78 Type 03 User's Manual.

1.1.1 Purpose

This manual is intended to give users an understanding of the methods for using the EEPROM Emulation Software (EES) RL78 Type 03 to reprogram the data flash memory in the RL78/F22, F25 microcontrollers (i.e. write constant data by the application).

1.2 Contents

The API function of EES RL78 Type 03 is called from the user program. And reprogramming of the data in the EEPROM emulation block (EES block) placed into the data flash memory is possible.

The EES RL78 Type 03 package includes the following.

- This user's manual
- Source code files of EES RL78 Type 03 for controlling the data flash memory incorporated in the RL78/F22, F25.
- Sample program for operating the EES RL78 Type 03.

1.3 Features

EES RL78 Type 03 calls API functions for RFD RL78 Type 03 to operate the flash memory sequencer. Each API function of EES RL78 Type 03 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 EES RL78 Type 03.

EES RL78 Type 03 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 EEPROM emulation processing in the user application.

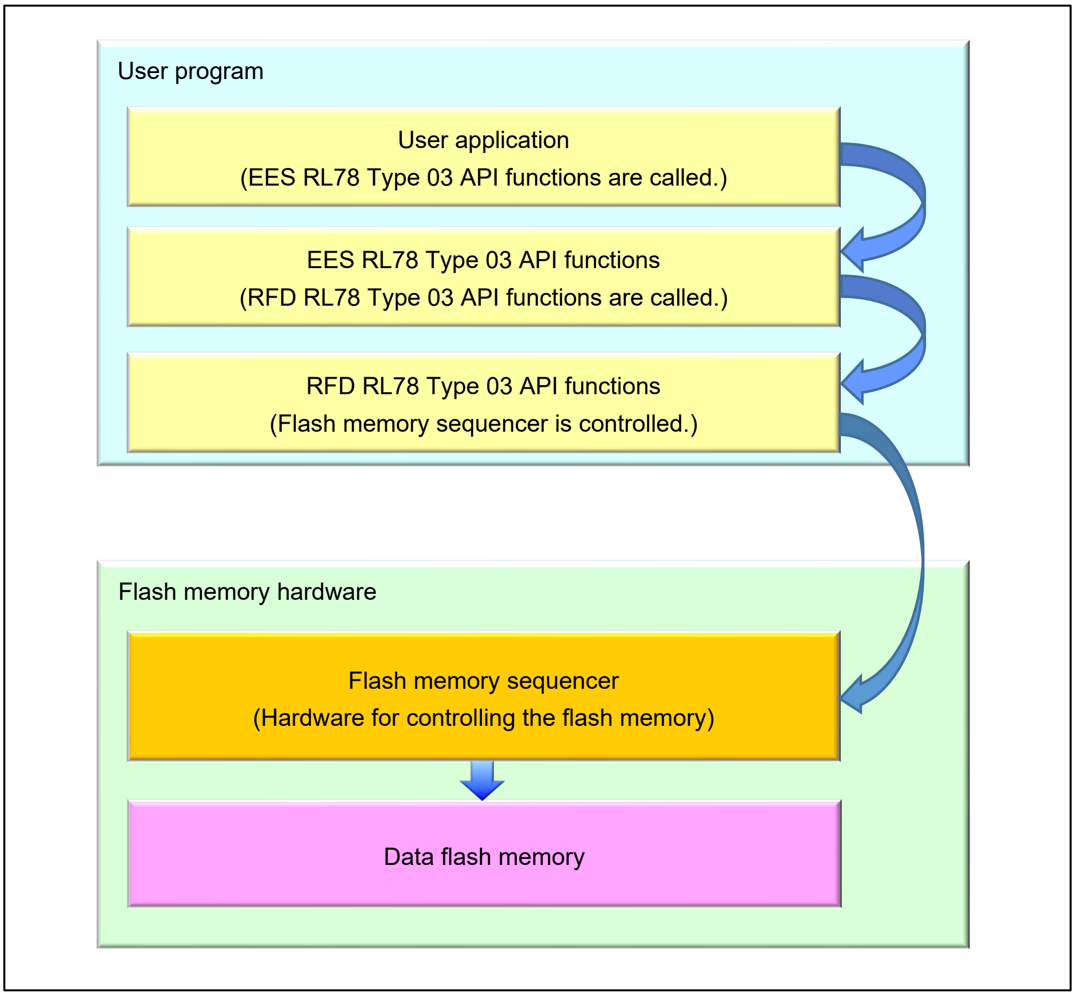


Figure 1-1 Data Flash Memory Control Using API Functions of EES RL78 Type 03

1.4 Operating Environment

- Host Computer
  The operation of EES RL78 Type 03 does not depend on the host computer but the appropriate environment for the C compiler package, debugger and emulator must be prepared. (EES RL78 Type 03 was developed and tested on Windows10 Enterprise.)

- **C Compiler Package**
  Table 1-1 shows the target C compiler packages for EES RL78 Type 03.

Table 1-1 The target C Compiler Packages for EES RL78 Type 03

Compiler	IDE（Integrated Development Environment）	Manufacturer	Version
CC-RL	CS+ or e^2 studio	Renesas Electronics	V1.13 or later
IAR	IAR Embedded Workbench® for Renesas RL78	IAR Systems®	V5.10.3 or later

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

- **Emulator**
  Table 1-2 shows the emulator on which the operation of EES RL78 Type 03 was confirmed.

Table 1-2 Emulator on which EES RL78 Type 03 operation was confirmed

Emulator	Manufacturer
E2 emulator	Renesas Electronics
E2 emulator Lite	Renesas Electronics

- **Target MCU**
  RL78/F22
  RL78/F25

- **EEPROM Emulation Software (EES)**
  Table 1-3 shows the EEPROM Emulation Software (EES) supported by this manual.

Table 1-3 EEPROM Emulation Software (EES) Supported by this Manual

Package	Manufacturer	Version
EES RL78 Type 03	Renesas Electronics	V1.00

Note. Use the version of RFD RL78 Type 03 listed in Table 1-4.

- **Renesas flash driver (RFD)**
  Table 1-4 shows the Renesas flash driver (RFD) used for EES RL78 Type 03.

Table 1-4 The Renesas flash driver (RFD) used for EES RL78 Type 03

Package	Manufacturer	Version
RFD RL78 Type 03	Renesas Electronics	V1.00

1.5 Points for Caution

EEPROM emulation is achieved by using a feature for manipulating the RL78/F22, F25 microcontroller data flash memory. Therefore, it is necessary to note the following.

(1) All EES code and constants must be placed in the same 64 Kbytes flash block such that EES code and constants do not extend across a 64-Kbyte boundary. (It dependent on each compiler.)

(2) The EES must be initialized by the R_EES_Init function before any EES function is executed.

(3) The data flash memory cannot be read during data flash memory operation by the EES.

(4) It is not allowed to call any RFD function during a command execution of the EES.

(5) It is not allowed to call any RFD control functions for EES directly from other than the EES.

(6) Do not execute STOP mode or HALT mode processing while the EEPROM emulation is being used. If it is necessary to execute STOP mode or HALT mode processing, be sure to execute all of the processing up to and including the R_EES_Close function to finish EEPROM emulation.

(7) The watchdog timer does not stop during execution of the EES.

(8) Do not destroy the request structure (st_ees_request_t) during command execution.

(9) Initialize the argument (RAM) that is used by the EEPROM emulation software function. When not initialized, a RAM parity error is detected and the RL78/F22, F25 microcontrollers might be reset. For a RAM parity error, refer to “User’s Manual: Hardware” of a target device.

(10) All members of the request structure (st_ees_request_t) must be initialized once before a EES command is executed. If any unused member exists in the request structure (st_ees_request_t), set a desired value for the member. If any member is not initialized, the RL78/F22, F25 microcontrollers may be reset due to a RAM parity error. For details, refer to “User’s Manual: Hardware” of a target device.

(11) The EES does not support multitask execution. Do not execute the EES functions during interrupt processing.

(12) After the R_EES_Close function have been executed, the requested command and ongoing command stop and cannot be resumed. Before calling the R_EES_Close function, finish all ongoing commands.

(13) Do not operate the code flash memory by RFD RL78 Type 03 while the EEPROM emulation is executed. Before the code flash memory is operated, be sure to execute a “R_EES_Close function” necessary in order to finish the EEPROM emulation. When using EEPROM emulation after executing the code flash memory operations using the RFD RL78 Type 03, it is necessary to start processing from the initializing function (the R_EES_Init function).

(14) Before starting the EEPROM emulation, be sure to start up the high-speed on-chip oscillator first. The high-speed on-chip oscillator must also be activated when using the external clock.

(15) No checksum is added to user data. If a checksum is needed, add it to user data and check through the user program.

(16) Do not operate the data flash control register (DFLCTL) during execution of the EES.

(17) To use the data flash memory for EEPROM emulation, it is necessary to execute the R_EES_ENUM_CMD_FORMAT command upon first starting up to initialize the data flash memory and make it usable as EES blocks.

(18) In order to use the EES, it is recommended to set at least 3 blocks in the EES block (virtual block).

(19) Do not destroy the EES blocks (virtual block) by the user program operating the data flash memory using the RFD from other than the EES.

(20) EES descriptor is changed, the EEPROM emulation can no longer be executed. In that case, the EES pool must be formatted by the R_EES_ENUM_CMD_FORMAT command in addition to initialization of EES. When adding data, however, the EEPROM emulation can be continuously executed.

(21) About an operating frequency of RL78/F22, F25 microcontrollers and an operating frequency value set by the initializing function (R_EES_Init), be aware of the following points:

- When using a frequency lower than 4 MHz as an operating frequency of RL78/F22, F25 microcontrollers, only 2 MHz and 3 MHz can be used (frequencies other than integer values like a 2.5 MHz cannot be used). Also, set an integer value 2, or 3 to the operating frequency value set by the initializing function.
- When using a frequency of 4 MHz or higher [Note] as an operating frequency of RL78/F22, F25 microcontrollers, a certain frequency can be used as an operating frequency of RL78/F22, F25 microcontrollers.
- This operating frequency is not the frequency of the high-speed on-chip oscillator.

Note: For a maximum frequency, refer to “User’s Manual: Hardware” of a target device.

(22) 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”.

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

When using CC-RL compiler, the Sample_INITSCT_EES function is called from the main function of main.c file. This function copies the data for EES RL78 Type 03 to RAM from ROM.

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

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

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

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

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

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

1.6 C Compiler Definitions

The definitions of the target compiler written in the header file (r_ees_compiler.h) for EES RL78 Type 03 are shown below.

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

- Definition of CC-RL compiler:
  “__CCRL__” is defined.
  #define COMPILER_CC (1)
- Definition of IAR compiler:
  “__IAR_SYSTEMS_ICC__” is defined.
  #define COMPILER_IAR (2)

< Descriptions in the r_ees_compiler.h file >

```
/* Compiler definition */
#define EES_COMPILER_CC  (1)
#define EES_COMPILER_IAR (2)

#if defined (__CCRL__)
    #define EES_COMPILER EES_COMPILER_CC
#elif defined (__IAR_SYSTEMS_ICC__)
    #define EES_COMPILER EES_COMPILER_IAR
#else
    /* Unknown compiler error */
    #error   “Non-supported compiler.”
#endif

/* Compiler dependent definition */
#if (EES_COMPILER_CC == EES_COMPILER)
    #define R_EES_FAR_FUNC                    __far
#elif (EES_COMPILER_IAR == EES_COMPILER)
    #define R_EES_FAR_FUNC                    __far_func
#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 System Configuration

The EES offers interface for accessing the data flash area (the EES pool) defined by the user. The API functions provided by EES accesses the EES pool via the RFD control functions for EES, or RFD.

The arrows shown in the Figure 2-1 below indicate the flow of processing.

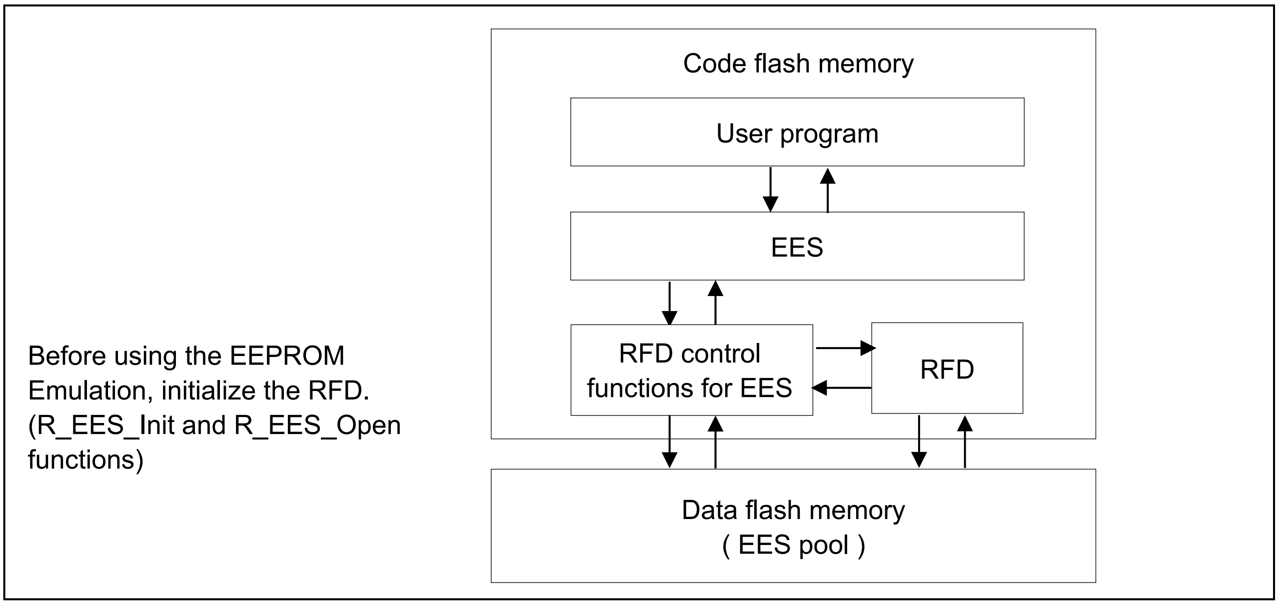


Figure 2-1 System Configuration

2.2 EES Architecture

This chapter describes the EES architecture required for the user to rewrite data flash memory (the EES pool) by using the EES.

2.2.1 EES Block

EES uses multiple blocks of the data flash memory as one virtual block. This area is called an EES block. The size of a block of the data flash memory mounted in RL78/F22, F25 are 1 Kbyte. When EES block size is set to a 2K-byte, two blocks of the data flash memory are gathered, and EES is handled as a 2K-byte's virtual block.

Be sure to set the size of an EES block in consideration of the size and the total number of blocks of the data flash memory mounted in the target device. Refer to "4.2 Initial Values to be Set by User" for the setting method. The schematic diagram for the EES block 0 when 1 K-byte or 2 K-byte are set by EES block is shown in "Figure 2-2 Schematic diagram of EES block 0".

Maximum number of blocks that can be set in the EES block of a product equipped with 16 Kbytes of data flash memory:

- When the EES block size is set to 1 K-byte , the maximum number of blocks is 16.
- When the EES block size is set to 2 K-byte , the maximum number of blocks is 8.

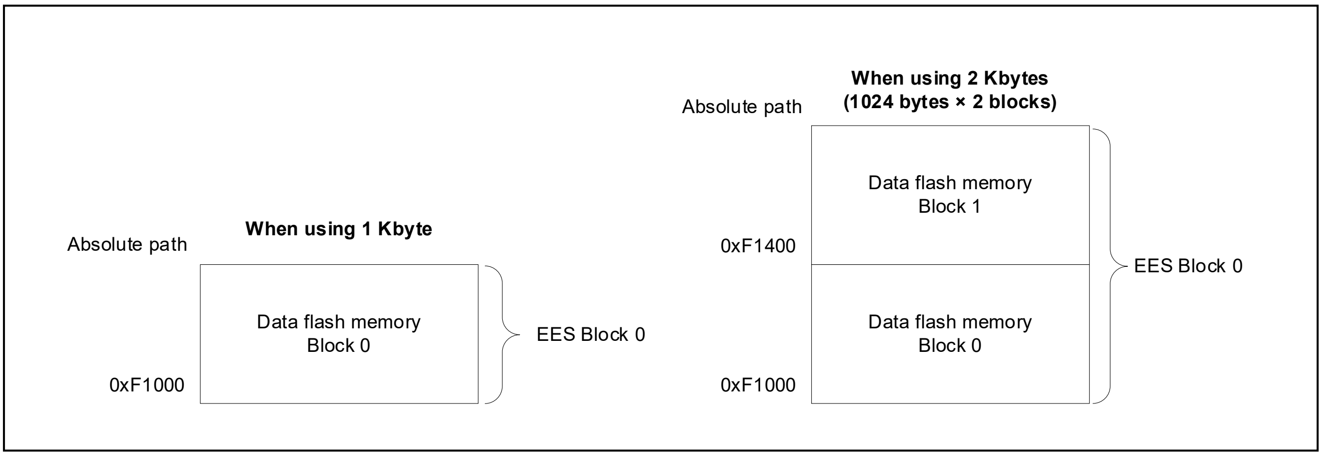


Figure 2-2 Schematic diagram of EES block 0

2.2.2 EES Pool

The EES pool is a user-defined data flash area that is accessible by the EES. The user program can access the data flash only by using this EES pool in the data flash via the RFD control functions for EES and the EES. The EES pool size must be specified with the number of size in the data flash of the target device. For the procedure to specify the number of blocks, see section 4.2 Initial Values to be Set by User.

Figure 2-3 shows an example of pool configuration for a device with 16 Kbytes data flash memory. （Example using 8Kbytes of 16Kbytes of data flash memory for EES block)

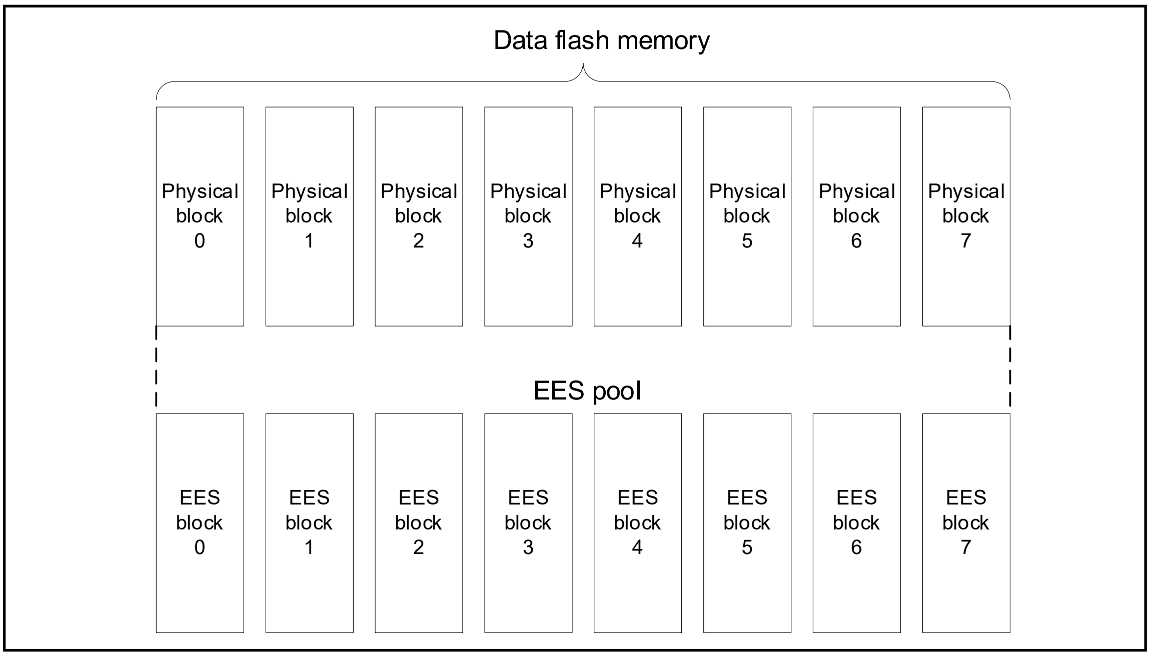


Figure 2-3 EES pool configuration example (EES block size: 1 Kbyte)

2.3 File Structure

2.3.1 Folder Structure

Figure 2-4 shows the folder structure of EES RL78 Type 03.

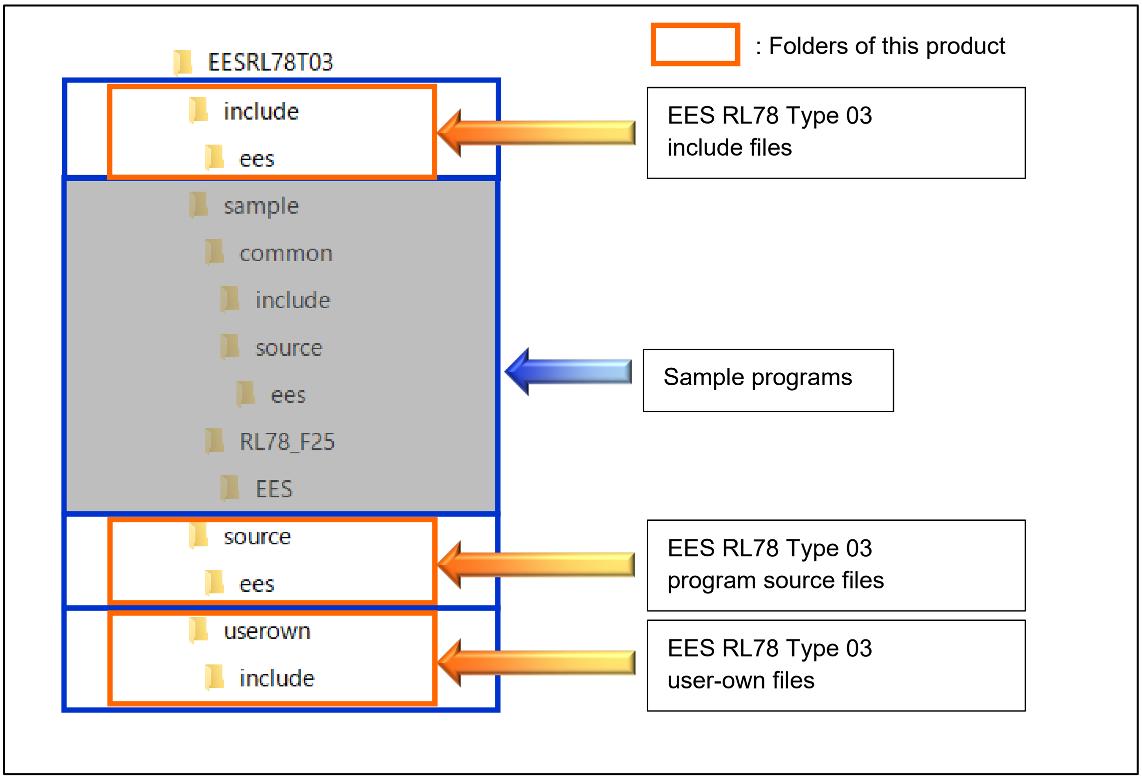


Figure 2-4 Folder Structure of EES RL78 Type 03

Note: Figure 2-4 shows an example of using RL78/F25. Refer to “6.1.1 Folder Structure” for the sample folder.

2.3.2 List of Files

2.3.2.1 List of Source Files

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

Table 2-1 Program Source Files in the “source\ees\” Folder

No.	Source File Name	Description
1	r_ees_api.c	This file contains the API functions for EEPROM emulation control.
2	r_ees_exrfd_api.c	This file contains the API functions RFD control functions for EES
3	r_ees_sub_api.c	This file contains API functions that are used as internal functions for EEPROM emulation control.

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

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

No.	Source File Name	Description
1	r_ees_descriptor.c	EES descriptor source file.

2.3.2.2 Header File List of Header Files

Table 2-3 shows the program header files in the “include\” folder.

Table 2-3 Program Header Files in the “include\” Folder

No.	Header File Name	Description
1	r_ees_api.h	This file defines the prototypes used in EEPROM control functions.
2	r_ees_exrfd_api.h	This file defines the prototypes used in RFD control functions for EES.
3	r_ees_sub_api.h	This file defines the prototypes for internal functions used in EEPROM emulation control functions.

Table 2-4 shows the program header files in the “userown\include\” folder.

Table 2-4 Program Header Files in the “userown\include\” Folder

No.	Header File Name	Description
1	r_ees_descriptor.h	EES descriptor header file.
2	r_ees_user_types.h	This file defines the types of user data used in EES.

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

Table 2-5 Program Header Files in the “include\ees” Folder

No.	Header File Name	Description
1	r_ees.h	Common header file.
2	r_ees_compiler.h	This file defines the compiler-dependent macros used in EES RL78 Type 03.
3	r_ees_defines.h	This file describes the definitions that differ between compilers used in EES RL78 Type 03.
4	r_ees_device.h	This file defines the hardware-specific macros used in EES RL78 Type 03.
5	r_ees_memmap.h	This file defines macros to describe sections used in EES RL78 Type 03.
6	r_ees_types.h	This file defines the types of variables used in EES RL78 Type 03.
7	r_typedefs.h	This file defines the types of data used in EES RL78 Type 03.

2.4 Resources of RL78/F22, F25

2.4.1 Memory Map

Table 2-6 shows the memory map (code flash memory [CF: 1 block = 2 Kbytes], data flash memory [DF: 1 block = 1 Kbyte], and RAM) of the RL78/F22, F25.

Table 2-6 Memory Map (Code Flash Memory, Data Flash Memory and RAM)

Device	Code Flash Memory: CF	Data Flash Memory: DF	RAM
RL78/F22 R7F122FxG (x=7, B, G)	128 Kbytes (00000H-1FFFFH)	8 Kbytes (F1000H-F2FFFH)	12 Kbytes (FCF00H-FFEFFH)
RL78/F25 R7F125FxL (x=G, L, M, P)	512 Kbytes (00000H-7FFFFH)	16 Kbytes (F1000H-F4FFFH)	40 Kbytes (F5F00H-FFEFFH)

2.4.2 Allocation of Blocks

Figure 2-5 shows the allocation of blocks in the code flash memory (CF).

RL78/F22 (Code flash memory: 128 Kbytes) RL78/F25 (Code flash memory: 512 Kbytes)

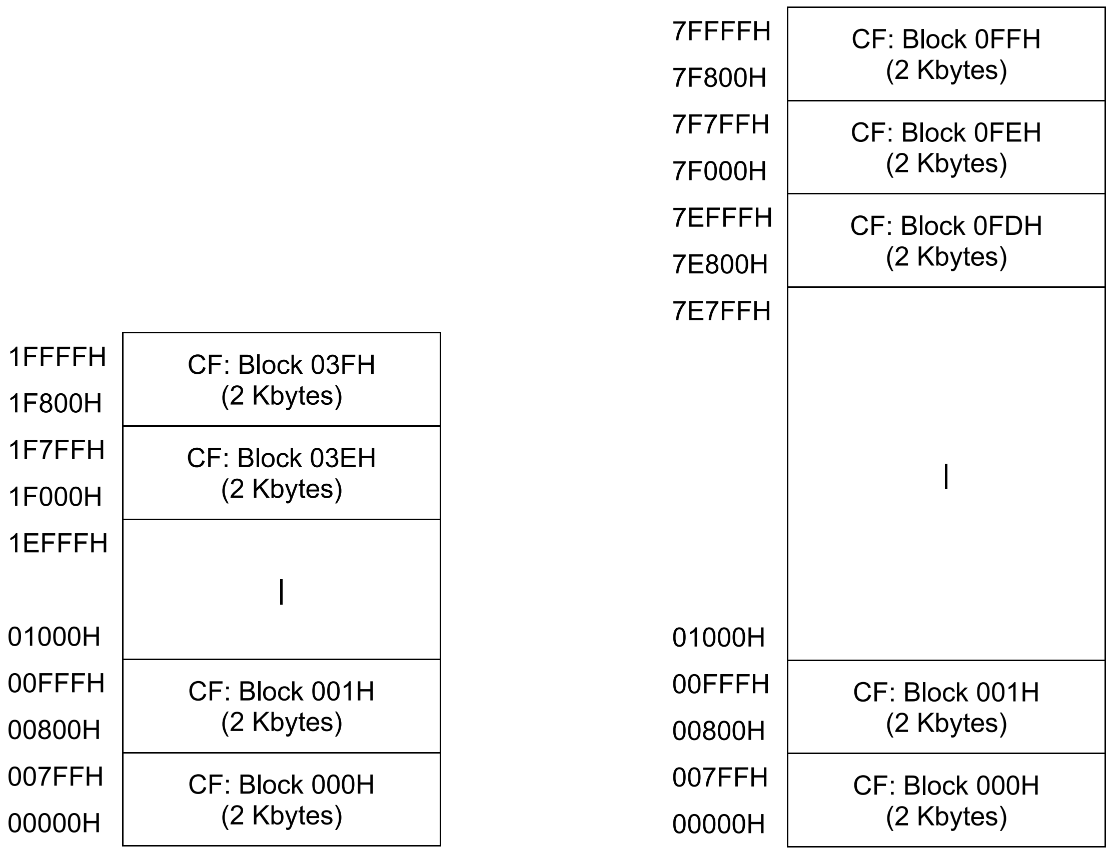


Figure 2-5 Blocks in the Code Flash Memory

Figure 2-6 shows the allocation of blocks in the data flash memory (DF).

RL78/F22 (Data flash memory: 8 Kbytes) RL78/F25 (Data flash memory: 16 Kbytes)

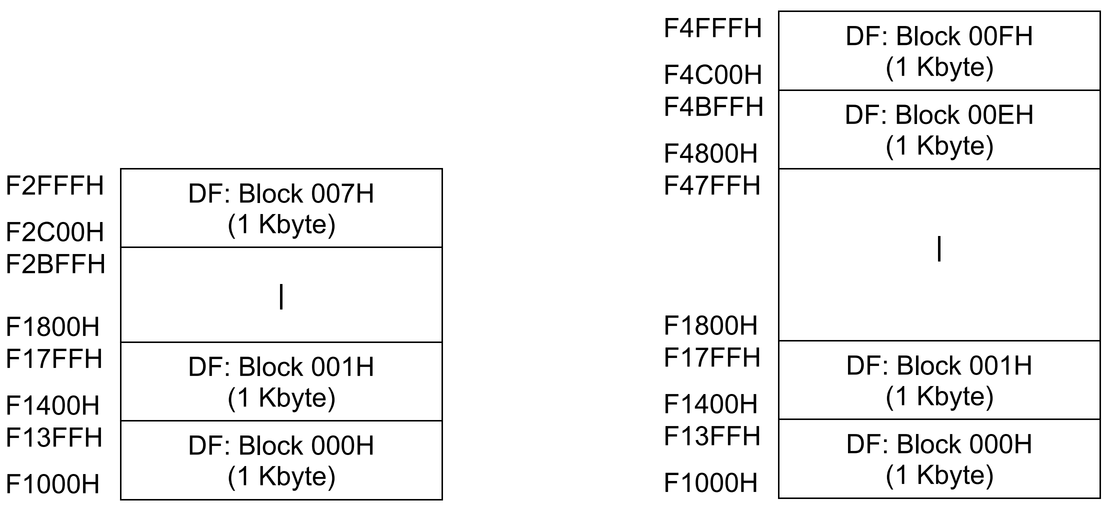


Figure 2-6 Blocks in the Data Flash Memory

2.5 Resources Used in EES RL78 Type 03

2.5.1 Sections Used in EES RL78 Type 03

Table 2-7 shows the sections used for EES and allocations of the sections.

Table 2-7 Sections Used in EES

Section Name	Description	Allocation
EES_CODE	Program section of API functions for EES control	ROM
EES_CNST	Constant variables section for EES initialized variables.	ROM
EES_VAR	Variables section for EES control	RAM
SMP_EES	Program section of sample functions for EES control	ROM
SMP_VAR	Variables section of sample functions for EES control	RAM

2.5.2 Software Resources

Table 2-8 shows software resources (Reference value).

Table 2-8 Software resources[Note1,2] (Reference value)

Item	Size (byte)	
	CC-RL	IAR
Stack	42	48
Code size[Note3]	4649	5221

Notes 1: These values are when using the compiler options described in “1.6 C Compiler Definitions”.
2: Does not include the stack and code size of the sample program.
3: Does not include code size of the RFD RL78 Type 03.

3. EEPROM Emulation

3.1 Specifications of EEPROM Emulation

By calling the EES functions provided by the EES RL78 Type 03 from a user-created program, use is possible without the awareness of data flash memory operations.
For the EES RL78 Type 03, a one-byte identifier (data ID: 1 to 254) is assigned by the user for each data item, and reading and writing using any unit from 1 to 255 bytes are possible on an assigned identifier basis. (The EES can handle up to 254 identifiers.)
Also, EES blocks (virtual block) for storing data use more than three blocks of area (recommended) [Note]. These blocks are called EES blocks. Data written by EEPROM emulation is divided into reference data and user-specified data, and the reference data is written to the target blocks from the lower block address, while the user data is written from the higher block address.

Note: At least two blocks are necessary for EEPROM emulation. When two blocks are specified, if a write error occurs even once, only reading of normally written data is possible but writing is no longer possible. After that, the two target blocks must be formatted when the EES is used to write data. Written data is erased completely. Since a contingency (such as voltage drop) may occur in the system, we recommend that you specify at least three blocks.

3.2 Outline of Functions

The EES provides basic read/write functions having the following features.

- The EES block size can be set to 1024 or 2048 bytes.
- Up to 254 data items settable.
- A data size of 1 to 255 bytes settable.
- Supporting the background operation (BGO).
- Memory consumption of data for EES management (Block header, Separator):
  10 bytes per EES block
- Memory consumption of reference data:
  3 bytes per EES block write data.
- Restoration by R_EES_ENUM_CMD_REFRESH when execution is stopped by a CPU reset while R_EES_ENUM_CMD_WRITE or R_EES_ENUM_CMD_REFRESH is running.
- Block rotation (averaging data flash use frequency).

Table 3-1 shows the range of settings when the EES functions are used.

Table 3-1 Range of Settings when the EES Functions are Used

Item	Range
EES block size	1024 or 2048 (bytes)
User data length	1 to 255
Amount of stored user data Note 1	1 to 254
Data ID range	1 to 254 (The numbers assigned are from 1 to 254 in the order of registration, and the selection of settings is not possible.)
Number of EES blocks Note 2	3 to 255
Recommended user data size Note 1	The EES block size is set to 1024 bytes: 1014 / 2 (bytes) or less The EES block size is set to 2048 bytes. 2038 / 2 (bytes) or less

Notes 1: The total size of user data must be within 1/2 of each block when all user data are written to an EES block. Therefore, the range used for the number of stored user data items differs depending on the size of the stored user data. It is also necessary to consider the size of the reference data provided for each data item for management use when determining the total size. For details about the number of stored user data items and total size, see “4.1 Number of Stored User Data Items and Total User Data Size”.

2: EES blocks cannot be set more than maximum number of blocks of on-board data flash memory.

3.3 EES Pool

This chapter describes the EES architecture required for the user to rewrite data flash memory (the EES pool) by using the EES.

3.3.1 EES Pool State

Each block has a state which indicates the current usage of the block. Table 3-2 shows States of the EES Blocks.

Table 3-2 States of the EES Blocks

State	Description
Active	Only a single EES block is active at a time to store defined data. The active block circulates in data flash blocks allocated in the EES pool.
Invalid	No data is stored in invalid blocks. EES blocks are marked as invalid by the EES or become invalid in the case of erasure blocks.
Excluded	If functional operation failed and possibility of a data flash failure is clarified, the EES excludes the relevant block and the block is no longer used for EEPROM emulation.

When no writable area is remaining in the active block (EES block 1 in the example) and data can no longer be stored (failure in write command), a new active block is selected in a cyclic manner and the current valid data set is copied to this new active block. This process is referred to as refresh. After the R_EES_ENUM_CMD_REFRESH command is executed, the previous active block becomes invalid and only a single active block exists. Excluded blocks (like block 7 in the example) are ignored during this process and not considered as candidates for the selection of the next active block.
Figure 3-1 shows an example of pool states (EES block size is set to 1 Kbyte).

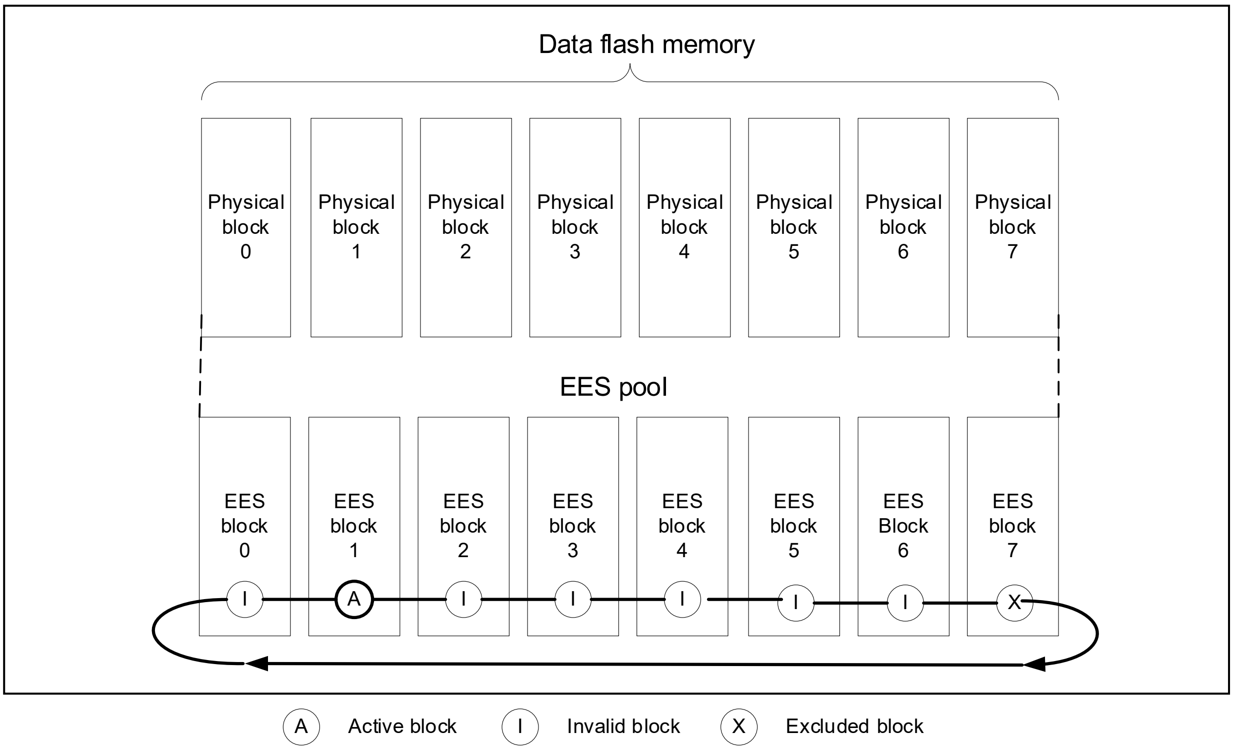


Figure 3-1 EES pool states example (The EES block size is set to 1 Kbyte)

The overall life cycle of a block in the EES pool is shown in Figure 3-2. During normal operation, the block switches between active and invalid state. When an error occurs during an access to the EES block, the error EES block is marked as excluded. This block will not enter the lifecycle again. However, the user can try to restore the block by a format of the complete pool which also erases all existing data content.

Caution: An EES block is a virtual block. Therefore, if even one of the physical blocks of data flash memory used in an EES block fails or otherwise becomes unusable, the EES block containing that block is considered a “excluded block”.

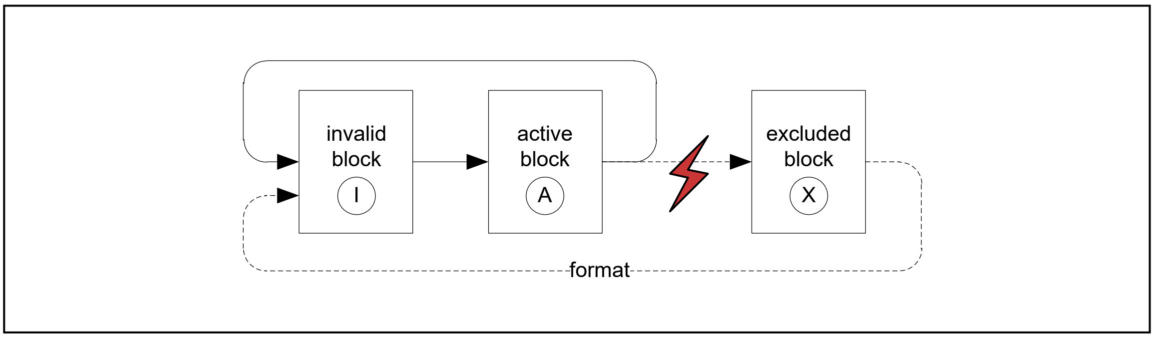


Figure 3-2 Life cycle of an EES block

The EES pool has the four states shown below.

Table 3-3 States of the EES Pool

State	Description
Pool operational	This is the usual case during EES operation. All commands are available and can be executed.
Pool full	Free space for data write is insufficient in the active block in use. This state indicates that a refresh needs to be executed.
Pool exhausted	No continuously usable EES block is left. (At least two blocks that are not excluded are necessary for EES operations.)
Pool inconsistent	There is a mismatch in the pool state and the data structure in the EES block does not match the user-set data structure. The EES block is in the undefined state (e.g. no active block is present).

3.3.2 Structure of EES Block

The detailed block structure used by the EES is shown in. In general, an EES block is divided into three utilized areas: the block header, the reference area, and the data area.

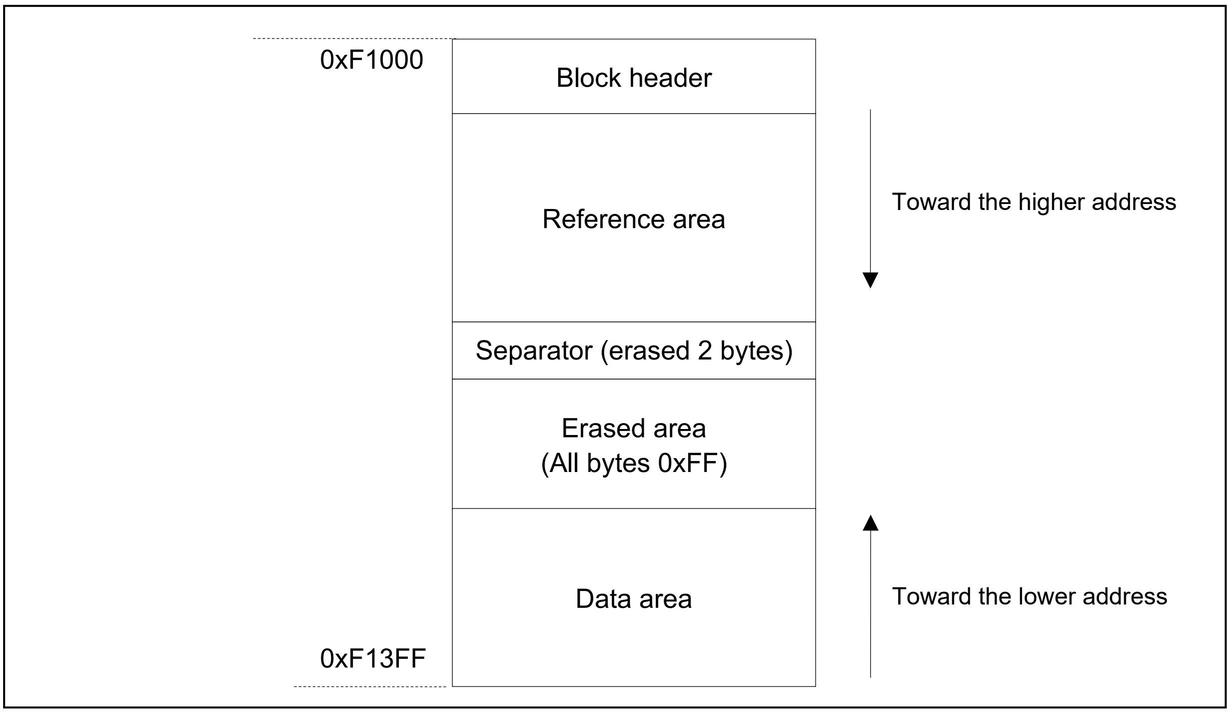


Figure 3-3 EES Block Structure(1 Kbyte)

Table 3-4 Configuration of Each EES Block

Name	Description
Block header	The block header contains all block status information needed for the block management within the EES-pool. It has a fixed size of 8 bytes.
Reference area	The reference area contains reference data which are required for the management of data. When data are written, this area expands in the direction of higher addresses.
Data area	The data area contains user data. When data are written, this area expands in the direction of lower addresses.

Between reference area and data area, there is an erased area. With each EES data update (i.e. the data is written), this area is reduced successively. However, at least 2 bytes of space always remain between reference area and data area for management and separation of these areas. This is indicated by the separator in Figure 3-3.
The EES block header is detailed in section "3.3.3 EES Block Header", while the structure of data stored in the reference and data area are described in section "3.3.4 Structure of Stored Data".

3.3.3 EES Block Header

The structure of the block header is depicted in Figure 3-4. It is composed of 8 bytes, three of which are reserved for the system.

Relative byte index within block		
0x0000	A	N
0x0001	B	0xFF - N
0x0002	B'	0x00
0x0003	I	0x00
0x0004	X	0x00
0x0005	-	Reserved
0x0006	-	Reserved
0x0007	-	Reserved

Figure 3-4 Structure of EES Block Header

The block status flags start at the beginning of the block and include the A flag, B flag, B' flag, I flag, and X flag, each of which is 1 byte, for a total of 5 bytes of data. The combination of flags indicates the EES block status.
Figure 3-4 shows the placement status of flags, and Table 3-5 shows the combination status of flags.

Table 3-5 Overviews of Block Status Flags

<table>
<tr><th colspan="5">Block Status Flag</th><th rowspan="2">State</th><th rowspan="2">Description</th></tr>
<tr><th>A Flag</th><th>B Flag</th><th>B’ Flag</th><th>I Flag</th><th>X Flag</th></tr>
<tr><td>0x01</td><td>0xFE</td><td>0x00</td><td>0xFF</td><td>0xFF</td><td rowspan="3">Active</td><td>Currently used block
After the R_EES_ENUM_CMD_REFRESH command is executed, the A flag of a new active block is set to 0x02.</td></tr>
<tr><td>0x02</td><td>0xFD</td><td>0x00</td><td>0xFF</td><td>0xFF</td><td>Currently used block
After the R_EES_ENUM_CMD_REFRESH command is executed, the A flag of a new active block is set to 0x03.</td></tr>
<tr><td>0x03</td><td>0xFC</td><td>0x00</td><td>0xFF</td><td>0xFF</td><td>Currently used block
After the R_EES_ENUM_CMD_REFRESH command is executed, the A flag of a new active block is set to 0x01.</td></tr>
<tr><td>0x01</td><td>0xFE</td><td rowspan="3">0x01 –
0xFE</td><td>0xFF</td><td>0xFF</td><td rowspan="3">Active</td><td rowspan="3">Currently used block. However, new data cannot be added because the writing for B' flag is not completed. (Read is possible.)
After executing the R_EES_ENUM_CMD_REFRESH command, the A flag of a new active block is set in the order of 0x01, 0x02, 0x03, 0x01,….</td></tr>
<tr><td>0x02</td><td>0xFD</td><td>0xFF</td><td>0xFF</td></tr>
<tr><td>0x03</td><td>0xFC</td><td>0xFF</td><td>0xFF</td></tr>
<tr><td colspan="2">--</td><td>0xFF</td><td>0xFF</td><td>0xFF</td><td rowspan="2">Invalid</td><td rowspan="2">Invalid block</td></tr>
<tr><td colspan="3">--</td><td>other than 0xFF</td><td>0xFF</td></tr>
<tr><td colspan="3">--</td><td>--</td><td>other than 0xFF</td><td>Excluded</td><td>Excluded block</td></tr>
</table>

3.3.4 Structure of Stored Data

The structure of stored data when user data is written to an EES block is shown in the figure below. A data is composed of three parts: the start-of-record (SoR) field and the end-of-record (EoR and EoR’) field and the data field. The EES descriptor table can be used to set data for use in the EES. Each data is referred to by an identification number (ID) and can have a size between 1 and 255 bytes. (The exact specification of the format of the EES descriptor can be found in section “4.2 Initial Values to be Set by User”.)

Each time data is written, stored data increase in the EES block and multiple units of stored data exist in the EES block, but only the most recent stored data is referenced.
SoR, EoR and EoR' build up the so-called reference data which is required for the management of the data. The reference data and user data values are stored in different sections of the active block, namely the reference area and the data area, respectively. Figure 3-6 shows the overview of the entire structure of stored data.

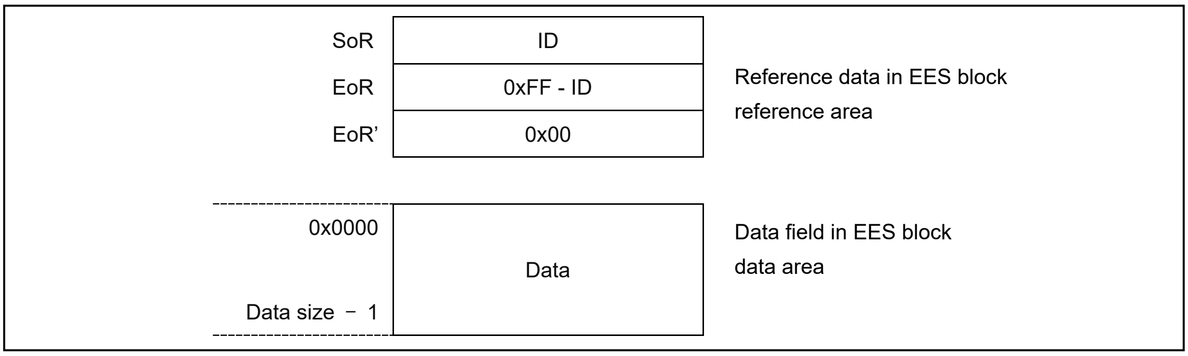


Figure 3-5 Structure of Stored Data

Table 3-6 Description of Each Field of Data Area

Name	Description
SoR field (Start of Record)	The 1 byte SoR field contains the ID of data. This field indicates the start of write processing. Data IDs 0x00 and 0xFF are not used to avoid patterns of erased cells.
EoR field (End of Record)	The 1 byte EoR field contains a 0xFF - data ID value. This field indicates successful end of write processing. If writing does not end normally due to a device reset or other reasons, the corresponding stored data is ignored by the EES.
EoR' field (End of Record')	The 1 byte EoR' field contains the completion of the write process to the EoR field. This field is written to 0x00 after the EoR field has been written. - When the value is between 0x01 - 0xFE, the stored data is valid, but the writing has not been completed. Therefore, the block is treated as a block to which data cannot be added. - When the value is 0xFF, EES judges with the execution result of the writing for the EoR field not having been a normal end.
Data field	The data field contains the user data. The size of user data is 1 to 255 bytes. When data of 2 bytes or more is stored, the smallest address of the data is allocated to the smallest address of the data field (as shown in Figure 3-6).

Data is written to the EES block in the order of SoR -> data field -> EoR -> EoR'. If the value of the EoR field is not written correctly, the immediately previous data becomes valid.

Notes 1: The total size of the reference data consumed by each stored data is 3 bytes. This should be considered when evaluating the free space in a block before writing the data through the R_EES_GetSpace function.

2: No checksum is added to user data. If a checksum is needed, add it to user data and check through the user program.

3.3.5 EES Block Overview

Figure 3-6 shows an example of an EES block that contains multiple units of stored data:

- Data ID 0x01 with size = 0x04
- Data ID 0x02 with size = 0x01
- Data ID 0x03 is defined but not written here.
- Data ID 0x04 with size = 0x02

The data have been written in the sequence ID 0x01 -> ID 0x04 -> ID 0x02.
In this example, the data with ID 0x03 has not been written yet.

Relative byte index within block		
0x0000	A = 0x02	Block header
0x0001	B = 0xFD	
0x0002	B' = 0x00	
0x0003	I = 0xFF	
0x0004	X = 0xFF	
0x0005	Reserved	
0x0006	Reserved	
0x0007	Reserved	
0x0008	SoR → ID = 0x01	Reference area
0x0009	EoR → ~ID = 0xFE	
0x000A	EoR' = 0x00	
0x000B	SoR → ID = 0x04	
0x000C	EoR → ~ID = 0xFB	
0x000D	EoR' = 0x00	
0x000E	SoR → ID = 0x02	
0x000F	EoR → ~ID = 0xFD	
0x0010	EoR' = 0x00	
0x0011 0x0012	Separator (erased 2 bytes)	
… 0x03F8	Erased area (All bytes 0xFF)	
0x03F9	DATA(ID = 0x02)[0]	Data area
0x03FA	DATA(ID = 0x04)[0]	
0x03FB	DATA(ID = 0x04)[1]	
0x03FC	DATA(ID = 0x01)[0]	
0x03FD	DATA(ID = 0x01)[1]	
0x03FE	DATA(ID = 0x01)[2]	
0x03FF	DATA(ID = 0x01)[3]	

Figure 3-6 Example of an Active EES Block

4. Using EEPROM Emulation

EEPROM emulation can store a maximum of 254 data items each consisting of 1 to 255 bytes in the flash memory by using three or more blocks (recommended) of flash memory.

EEPROM emulation can be executed by incorporating the EES into a user-created program and executing that program.

4.1 Number of Stored User Data Items and Total User Data Size

The total size of user data that can be used in the EEPROM emulation is limited. The size required for writing all user data to an EES block must be within 1/2 of the block. Therefore, the number of stored data items that can be used differs depending on the size of user data that is actually stored. The following shows how to calculate the size that can be used when actually writing user data, as well as the total user data size.

[Maximum usable size of one block that can be used to write the user data]

Size required for EEPROM emulation block management:	8 bytes
Free space necessary as termination information (separator):	2 bytes

- EES Block size: 1024 bytes
  Maximum usable size of one block = 1024 bytes - (8 bytes + 2 bytes) = 1014 bytes

- EES Block size: 2048 bytes
  EES block size: 1024 bytes * 2 = 2048 bytes
  Maximum usable size of one block = 2048 bytes - (8 bytes + 2 bytes) = 2038 bytes

[Calculating the size for writing each user data item] Note

Size of each written user data item = data size + reference data size (3 bytes)

Note: For details, see “3.3.4 Structure of Stored Data”.

[Calculating the basic total user data size]

Basic total size = (user data 1 + 3) + (user data 2 + 3) ... + (user data n + 3)

[Maximum size and recommended size]

Data must be held in one block. Therefore, the maximum size is the maximum usable size of one block but the following relational expression should be met. To enable all data to be updated at least once, we recommend that the data be within the half size of the maximum usable size of one block.

Maximum size: Assumed that the largest data can be updated once after all data have been written.
Recommended size: Assumed that all data can be updated once after all data have been written.

- EES Block size: 1024 bytes
  Maximum size = the basic total user data size + maximum data size + 3 ≤ 1014
  Recommended size = 1014 / 2 = 507 bytes or less

- EES Block size: 2048 bytes
  Maximum size = the basic total user data size + maximum data size + 3 ≤ 2038
  Recommended size = 2038 / 2 = 1019 bytes or less

4.2 Initial Values to be Set by User

As the initial values for the EES, be sure to set the items indicated below. In addition, before executing the EES, be sure to execute the high-speed on-chip oscillator. The high-speed on-chip oscillator must also be activated when using the external clock.

・Number of stored data items, and data size of the identifier (data ID)

< EEPROM emulation soft wear user include file (r_ees_descriptor.h) > Notes 2, 3

```
#define R_EES_EXRFD_VALUE_U16_PHYSICAL_BLOCK_SIZE   (1024u)
                                    : (1) The size of one block of data flash memory
                                          (Physical block size).
#define R_EES_EXRFD_VALUE_U08_PHYSICAL_BLOCKS_PER_VIRTUAL_BLOCK    (1u)
                                    : (2) The number of data flash memory blocks
                                          (Number of physical blocks) to set in the EES
                                          block (Per virtual block). Notes 1
#define R_EES_EXRFD_VALUE_U08_POOL_VIRTUAL_BLOCKS                  (4u)
                                    : (3) EES pool size (Number of virtual blocks).
#define R_EES_VALUE_U08_VAR_NO   (8u)  : (4) Number of stored data items.
```

Notes 1: The number of data flash memory blocks that can be set for the EES block is 1u or 2u.

< EEPROM emulation software user data definition file (r_ees_user_types.h) > Notes 3

```
typedef   uint8_t   type_A[2];        : (5) Data size definition of each data identifier
typedef   uint8_t   type_B[3];              (data ID).
typedef   uint8_t   type_C[4];
typedef   uint8_t   type_D[5];
typedef   uint8_t   type_E[6];
typedef   uint8_t   type_F[10];
typedef   uint8_t   type_X[20];
typedef   uint8_t   type_Z[255];
```

< EEPROM emulation software user program file (r_ees_descriptor.c)> Notes 3

```
__far const uint8_t g_ar_u08_ees_descriptor                : (6) Data size of each data identifier
[R_EES_VALUE_U08_VAR_NO + 2u] =                                (data ID).
{
  (uint8_t)( R_EES_VALUE_U08_VAR_NO ),  /* variable count      */  \
  (uint8_t)(sizeof(type_A)),        /* id=1              */  \
  (uint8_t)(sizeof(type_B)),        /* id=2              */  \
  (uint8_t)(sizeof(type_C)),        /* id=3              */  \
  (uint8_t)(sizeof(type_D)),        /* id=4              */  \
  (uint8_t)(sizeof(type_E)),        /* id=5              */  \
  (uint8_t)(sizeof(type_F)),        /* id=6              */  \
  (uint8_t)(sizeof(type_X)),        /* id=7              */  \
  (uint8_t)(sizeof(type_Z)),        /* id=8              */  \
  (uint8_t)(0x00),                  /* zero terminator   */  \
};
```

Notes 2: The macros that are being used are parameters which are common to the whole EES, so any changes should only be to numerical values.

3: After initializing the EEPROM emulation blocks (after executing the R_EES_ENUM_CMD_FORMAT command), do not change the values. If the values are changed, reinitialize the EES blocks (by executing the R_EES_ENUM_CMD_FORMAT command).

(1) The size of one block of data flash memory (Physical block size).
Set the size of one block of data flash memory installed (mounted) in the target device.

(2) The number of data flash memory blocks (Number of physical blocks) to set in the EES block.
Sets the number of data flash memory blocks to use for the EES block.

(3) EES pool size. Note
The number of blocks in the data flash memory of the target device must be specified as the number of blocks in the EES pool.
Note: Specify 3 (3 blocks) or a greater value (recommended).

(4) Number of stored data items
Specify the number of data items to be used in the EEPROM emulation. A value of 1 to 254 can be set.

(5) Data size definition of each data identifier (data ID).
Defines the data type name for the byte size of each user data. The EES descriptor table reflects the byte size of each user data.

(6) Data size of each data identifier (data ID)

A table to define the data size of each identifier is provided below. This is called an EES descriptor table. Data to be written must be registered in the EES descriptor table in advance.

__far const uint8_t g_ar_u08_ees_descriptor [Number of stored data items + 2]

R_EES_VALUE_U08_VAR_NO
Byte size of data ID1
Byte size of data ID2
Byte size of data ID3
Byte size of data ID4
Byte size of data ID5
Byte size of data ID6
Byte size of data ID7
Byte size of data ID8
0x00

Figure 4-1 EES Descriptor Table (When there are eight different data items)

- R_EES_VALUE_U08_VAR_NO
  User-specified number of data items used in the EES
- Byte size of data IDx
  User-specified size of user data (in bytes)
- Termination area (0x00)
  Specify 0 as the termination information.

5. User Interface

5.1 Request Structure (st_ees_request_t) Settings

Basic operations such as reading from and writing to the data flash memory are performed by a single function. The function transfers commands and data ID to the EES via the request structure (st_ees_request_t). Furthermore, the EES state and error information are acquired via the request structure (st_ees_request_t).

In subsequent sections, write access to the request structure (st_ees_request_t) from the user is called user write access, and read access to it from the user is called user read access.

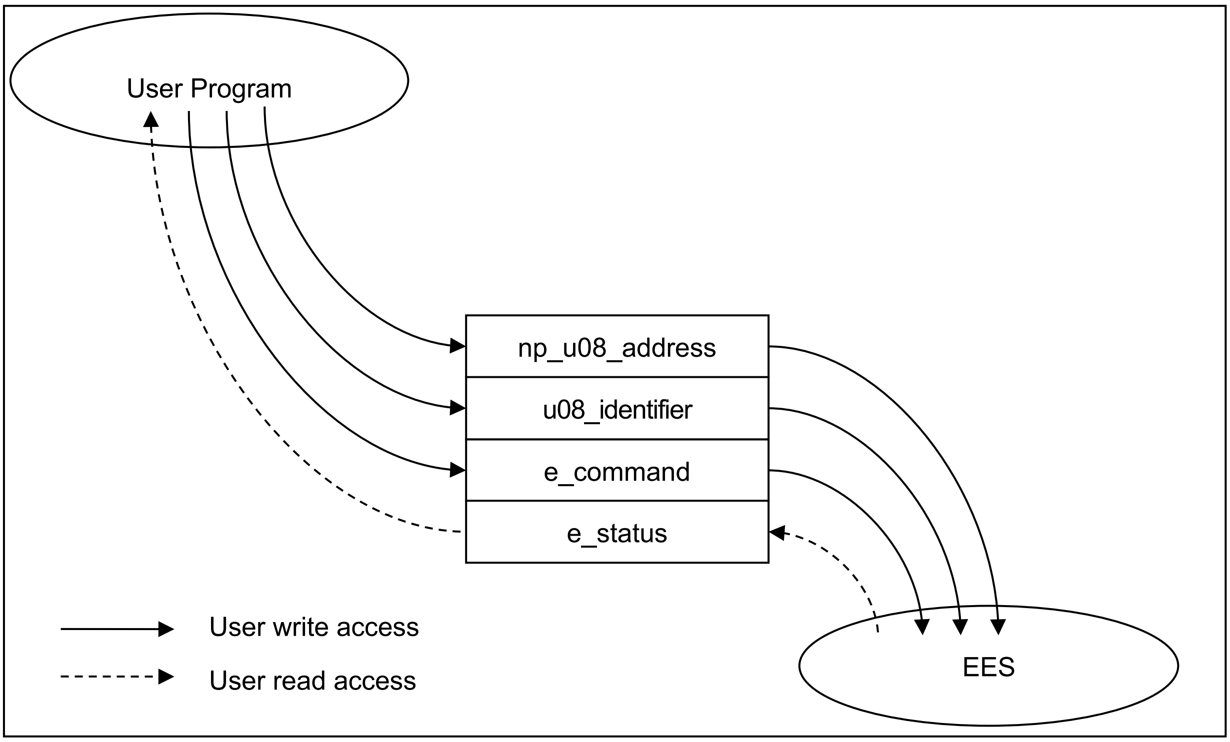


Figure 5-1 Request Structure (st_ees_request_t)

The request structure (st_ees_request_t) is defined in the r_ees_types.h file. It should not be changed by the user.

[Definition of the request structure (st_ees_request_t)]

```
typedef struct st_ees_request
{
  uint8_t __near *        np_u08_address;
  uint8_t                 u08_identifier;
  e_ees_command_t         e_command;
  e_ees_ret_status_t      e_status;
} st_ees_request_t;
```

uint8_t __near * np_u08_address	
uint8_t u08_identifier	e_ees_command_t e_command
e_ees_ret_status_t e_status	

Bit 0 Bit 15

Figure 5-2 Alignment of Variables of the Request Structure (st_ees_request_t)

5.1.1 User Write Access

(1) np_u08_address

Specifies a pointer to the start address of the data buffer used for R_EES_ENUM_CMD_WRITE command and R_EES_ENUM_CMD_READ command execution.

Associated command (macro name)	Setting
R_EES_ENUM_CMD_WRITE	Pointer to the start address of the data buffer. Note 1
R_EES_ENUM_CMD_READ	Pointer to the start address of the data buffer. Note 2

Notes 1: Buffer which contains data written by the user
2: Buffer which contains data read from the data flash memory

(2) u08_identifier

Specify the data ID used for each command. For more information about how to do this, see the description of the R_EES_Execute function in section “5.7 Specifications of API Functions”.

Associated command (macro name)	Setting
R_EES_ENUM_CMD_WRITE	ID of write data
R_EES_ENUM_CMD_READ	ID of read data

(3) e_command

Commands to be set in the common executable function.

Associated command (macro name)	Description
R_EES_ENUM_CMD_UNDEFINED	Undefined command (Initial value: It is used only for initialization.)
R_EES_ENUM_CMD_STARTUP	Startup processing
R_EES_ENUM_CMD_WRITE	Write processing
R_EES_ENUM_CMD_READ	Read processing
R_EES_ENUM_CMD_REFRESH	Refresh processing
R_EES_ENUM_CMD_FORMAT	Format processing
R_EES_ENUM_CMD_SHUTDOWN	Shutdown processing

5.1.2 User Read Access

- e_status

EES status and error information. For information about the status and errors which might occur during execution of the functions, see the description of the R_EES_Execute function in section “5.7 Specifications of API Functions”

5.2 List of API Functions and R_EES_Execute function commands for the EES

5.2.1 API Functions for the EES

Table 5-1 shows the API functions for EES RL78 Type 03.

Table 5-1 API Functions for EES RL78 Type 03

	API Name	Overview
1	R_EES_Init	Initializes internal data and variables and checks the descriptor configuration.
2	R_EES_Open	EEPROM emulation preparation processing. This function makes the EEPROM emulation executable.
3	R_EES_Close	EEPROM emulation end processing. This function makes the EEPROM emulation un-executable.
4	R_EES_Execute	EEPROM emulation execution function. Each type of processing for performing EEPROM emulation operations is specified for this function as an argument in the command format, and the processing is executed.
5	R_EES_Handler	Continuous EEPROM emulation execution processing. This function is used to check for the completion of processing while allowing processing of EEPROM emulation specified by the R_EES_Execute function to continue.
6	R_EES_GetSpace	Gets the free space of the active block.

5.2.2 Commands for R_EES_Execute Function

Table 5-2 shows commands for R_EES_Execute.

Table 5-2 List of commands for R_EES_Execute

	Command Name	Outline
1	R_EES_ENUM_CMD_STARTUP	[Startup Processing] This command checks the block status and sets the system to the EEPROM emulation (data access) valid state (Full Access). If two active blocks exist, the incorrect block is changed to an invalid block. Be sure to execute this command before executing commands other than the R_EES_ENUM_CMD_FORMAT command and make sure that the command finishes normally.
2	R_EES_ ENUM_CMD_WRITE Note1	[Write Processing] This command writes the specified data to the EES block. * The following arguments must be specified prior to execution. - np_u08_address: Specifies a pointer to the start address of the RAM area where the write data is stored. - u08_identifier: Specifies the data ID of the write data.
3	R_EES_ ENUM_CMD_READ Note1	[Read Processing] Read the specified data from an EES block. * The following arguments must be specified prior to execution. - np_u08_address: Specifies a pointer to the start address of the RAM area where the read data is stored. - u08_identifier: Specifies the data ID of the read data.
4	R_EES_ ENUM_CMD_REFRESH Note1,2	[Refresh Processing] Copy the latest stored data from the active block (copy source EES block) to the next block (copy destination EES block) in the EES pool after the erase processing. This makes the copy destination block active.
5	R_EES_ ENUM_CMD_FORMAT	[Format Processing] Initialize (erase) everything, including the data recorded in the whole EES pool. Be sure to use this command before using EEPROM emulation for the first time. Note that issuing this command is also necessary to initialize all blocks if a malfunction occurs in an EES block (such as an active block disappearing) or the values in the descriptor table (those which are fixed values that cannot be changed) are modified. Because EEPROM emulation switches to the stopped state (opened) regardless of the results after the processing finishes, execute the R_EES_ENUM_CMD_STARTUP command to continue using EEPROM emulation.
6	R_EES_ ENUM_CMD_SHUTDOWN Note1	[Shutdown Processing] Set the EEPROM emulation operation to the stopped state (opened).

Notes 1: Do not execute this command until the R_EES_ENUM_CMD_STARTUP command has finished normally.

2: The erase processing is performed by executing the R_EES_ ENUM_CMD_REFRESH command.

5.2.3 RFD control API functions for EES

Table 5-3 shows RFD control API functions for EES.
This function is used internally by EES. It does not need to be used directly by the user.

Table 5-3 List of RFD control API functions for EES

	API Name	Overview
1	R_EES_EXRFD_Init	Initializes RFD RL78 Type 03.
2	R_EES_EXRFD_Open	Set the data flash control register (DFLCTL) to the state where accessing the data flash memory is permitted (DFLEN = 1).
3	R_EES_EXRFD_Close	Set the data flash control register (DFLCTL) to the state where access to the data flash memory is inhibited (DFLEN = 0). All ongoing EES processing stop.
4	R_EES_EXRFD_Erase	Start erasing the EES block (one virtual block).
5	R_EES_EXRFD_Write	Starts writing to the specified the data flash memory address (one byte).
6	R_EES_EXRFD_BlankCheck	Starts Blank check to the specified the data flash memory address.
7	R_EES_EXRFD_Read	Reads the specified address in the data flash memory.
8	R_EES_EXRFD_Handler	Continues processing of the RFD control function for EES that is executing, and confirms termination.

5.3 State Transitions

To use EEPROM emulation from a user-created program, it is necessary to initialize the EES and execute functions that perform operations such as reading and writing on EES blocks. **Figure 5-3** shows the overall state transitions, and **Figure 5-4** shows an operation flow for using basic features. When using EEPROM emulation, incorporate EEPROM emulation into user-created programs by following this flow.

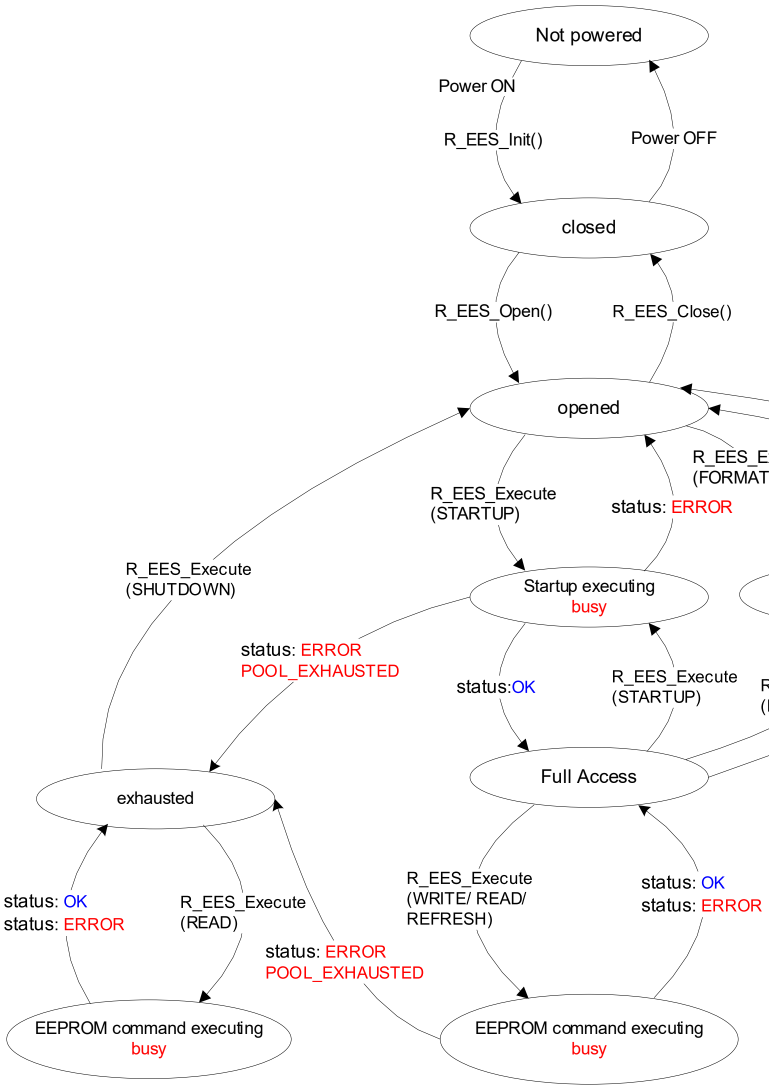


Figure 5-3 State Transitions Diagram

Note: Once the R_EES_ENUM_CMD_FORMAT command has started running, execute the R_EES_Handler function to check for its completion.

[Overview of state transitions diagram]

To use EES to manipulate the data flash memory, it is necessary to execute the provided functions in order to advance the processing.

(1) Not powered

Status is Power Off.

(2) closed

This is the state in which the data to perform EEPROM emulation is initialized by executing the R_EES_Init functions (no ongoing operation to the data flash memory).

Do not execute “operation of the code flash memory”, STOP mode or HALT mode while the EEPROM emulation is executing. In the case where they are executed, execute R_EES_Close function and change to a Closed state.

(3) opened

This state is switched to by executing R_EES_Open in the closed state and makes it possible to perform operations on the data flash memory. Even if the R_EES_Close function is executed, do not execute “operation of the code flash memory”, STOP mode, or HALT mode until a state change to “closed”.

(4) started

This state is switched to by executing the R_EES_ENUM_CMD_STARTUP command in the opened state and makes it possible to execute EEPROM emulation. Writes and reads that use EEPROM emulation are performed in this state.

(5) exhausted

This state is made from the opened or started state when continuously usable EES blocks have been exhausted during command execution. In this state, only R_EES_ENUM_CMD_READ, and R_EES_ENUM_CMD_SHUTDOWN commands are executable.

(6) busy

This is the state used when executing a specified command. The state that is switched to differ depending on which command is executed and how it terminates.

5.4 Basic Flowchart

Figure 5-4 below shows the basic procedure to perform read and write operations for the data flash by using the EES.

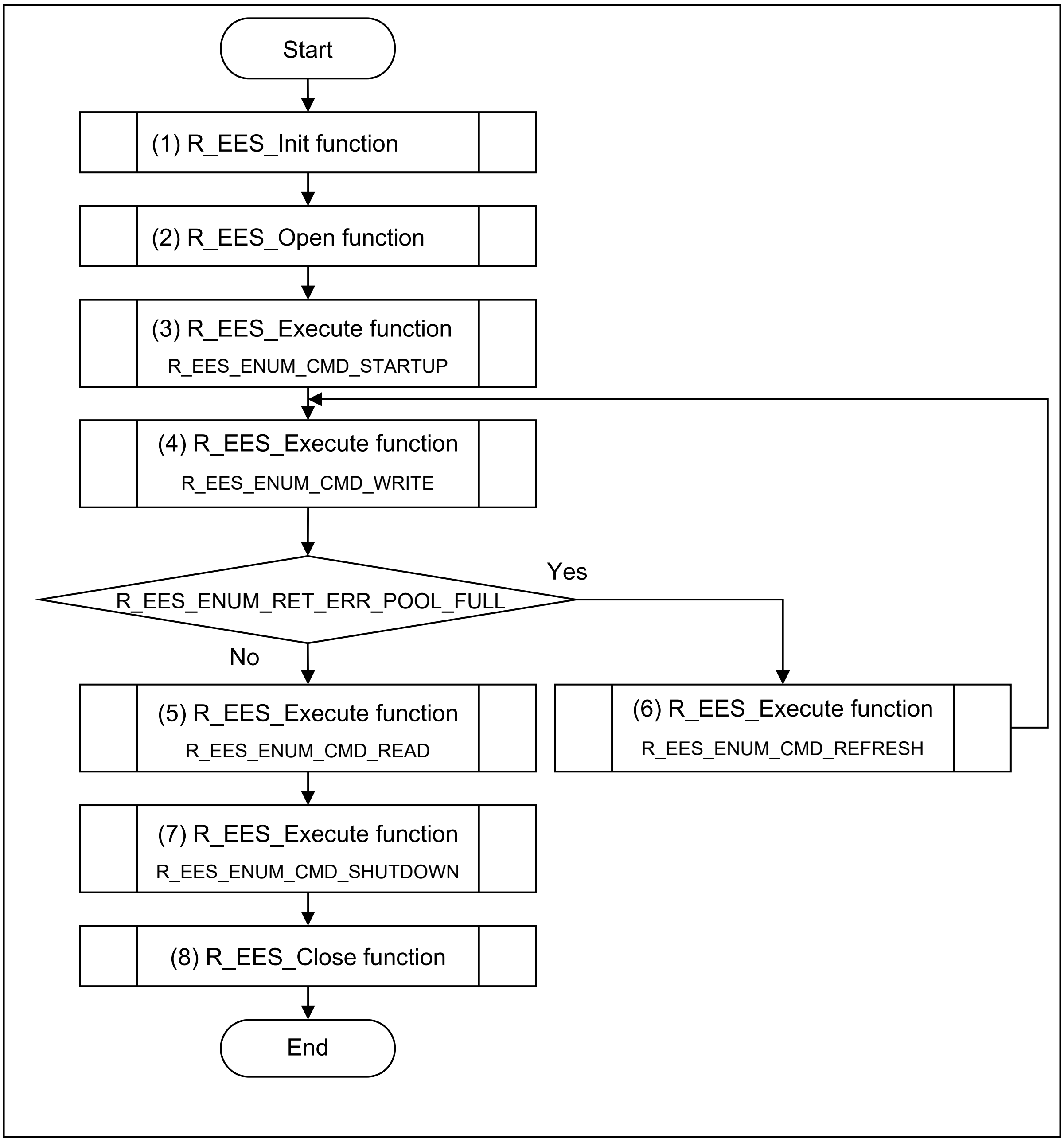


Figure 5-4 Basic Flowchart of EES

Notes 1: When using the EEPROM emulation for the first time, be sure to execute the R_EES_ENUM_CMD_FORMAT command.

2: This flowchart omits error handling and R_EES_Handler processing after command execution.

[Overview of basic operation flow]

(1) EES initialization processing (R_EES_Init)

Initialize the parameters used by the EES.

(2) EEPROM emulation preparation processing (R_EES_Open)

Set the data flash memory to a state (opened) for which control is enabled to execute EEPROM emulation.

(3) EEPROM emulation execution start processing (R_EES_Execute: R_EES_ENUM_CMD_STARTUP command)

Set the system to a state (Full Access) in which EEPROM emulation can be executed.

(4) EEPROM emulation data write processing (R_EES_Execute: R_EES_ENUM_CMD_WRITE command)

Write the specified data to an EES block.

(5) EEPROM emulation data read processing (R_EES_Execute: R_EES_ENUM_CMD_READ command)

Read the specified data from an EES block.

(6) EEPROM emulation refresh processing (R_EES_Execute: R_EES_ENUM_CMD_REFRESH command)

The latest stored data is copied from the active block (source block) to the next block (destination block) in the EES pool after the erase processing. This makes the copy destination block active.

(7) EEPROM emulation execution stop processing (R_EES_Execute: R_EES_ENUM_CMD_SHUTDOWN command)

Set the EEPROM emulation operation to the stopped state (opened).

(8) EEPROM emulation end processing (R_EES_Close)

Set the data flash memory to a state (closed) for which control is disabled to stop EEPROM emulation.

5.5 Command Operation Flowchart

The figure below shows the basic procedure to perform read and write operations for data flash by using the EES.

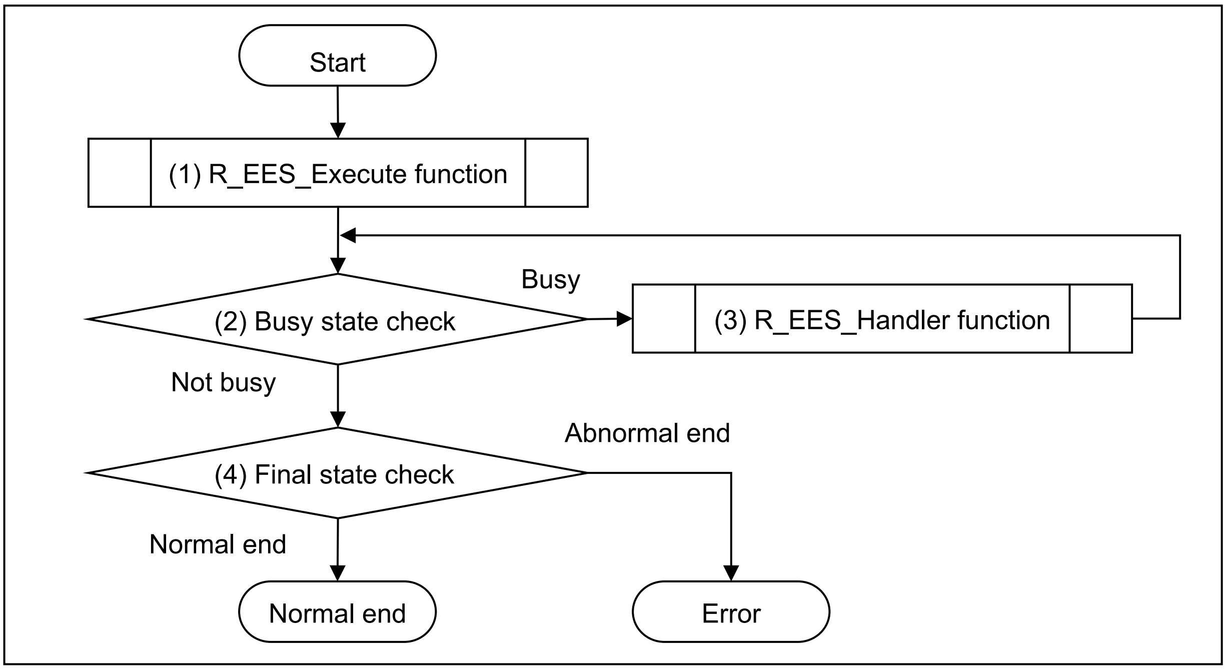


Figure 5-5 Command Operation Flowchart

(1) R_EES_Execute function

Perform operations for the data flash memory.

(2) Busy state check

Check e_status of the request structure (st_ees_request_t).

When e_status is R_EES_ENUM_RET_STS_BUSY, continue the data flash operation. If the value of e_status is other than R_EES_ENUM_RET_STS_BUSY, check the final state.

(3) R_EES_Handler function

Control the EES while it is running. By repeating the execution of the R_EES_Handler function, continue the data flash operation.

(4) Final state check

If the final state is R_EES_ENUM_RET_STS_OK, the operation ends normally. Otherwise, it will be terminated with an error.

5.6 Data Type Definitions

5.6.1 Data Types

Table 5-4 shows the data type definitions in EES RL78 Type 03.

Table 5-4 Data Type Definitions in EES RL78 Type 03

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”.

5.6.2 Global Variables

The following shows the global variables used in EES RL78 Type 03.

(1) g_ar_u08_ees_descriptor[R_EES_VALUE_U08_VAR_NO + 2u]

Type/Name	uint8_t g_ar_u08_ees_descriptor[]
Default value	(uint8_t)(R_EES_VALUE_U08_VAR_NO), /* variable count */ (uint8_t)(sizeof(type_A)), /* id=1 */ (uint8_t)(sizeof(type_B)), /* id=2 */ (uint8_t)(sizeof(type_C)), /* id=3 */ (uint8_t)(sizeof(type_D)), /* id=4 */ (uint8_t)(sizeof(type_E)), /* id=5 */ (uint8_t)(sizeof(type_F)), /* id=6 */ (uint8_t)(sizeof(type_X)), /* id=7 */ (uint8_t)(sizeof(type_Z)), /* id=8 */ (uint8_t)(0x00u) /* zero terminator */
Description	Stores the data size of each data identifier (Data ID).
Definition file	r_ees_descriptor.c

(2) g_st_ees_exrfd_descriptor

Type/Name	st_ees_exrfd_descriptor_t g_st_ees_exrfd_descriptor
Default value	(uint16_t) R_EES_EXRFD_VALUE_U16_PHYSICAL_BLOCK_SIZE (uint8_t) R_EES_EXRFD_VALUE_U08_PHYSICAL_BLOCKS_PER_VIRTUAL_BLOCK (uint8_t) R_EES_EXRFD_VALUE_U08_POOL_VIRTUAL_BLOCKS
Description	Contains settings that configure the EES pool ・uint16_t u16_ees_physical_block_size; The size of one block of data flash memory (Physical block size). Example: This value is fixed for RL78/F22, F25. (1024u) ・uint8_t u08_ees_physical_blocks_per_virtual_block; The number of data flash memory blocks to set in the EES block (Number of physical blocks). Example: When setting 1 Kbyte for EES block. Number of data flash memories. (1u) Example: When setting 2 Kbytes for EES block. Number of data flash memories. (2u) ・uint8_t u08_ees_pool_virtual_blocks; EES pool size (Number of virtual blocks) Example: Total EES blocks. (4u)
Definition file	r_ees_descriptor.c

(3) g_ar_u16_ram_ref_table[R_EES_VALUE_U08_VAR_NO]

Type/Name	uint16_t g_ar_u16_ram_ref_table[]
Default value	-
Description	Contains reference data for each data identifier (Data ID).
Definition file	r_ees_descriptor.c

5.6.3 Enumerations

- e_ees_command (enumerated-type variable name: e_ees_command_t)
  EES executable command

Symbol Name	Value	Description
R_EES_ENUM_CMD_UNDEFINED	0x00	Undefined command (Initial value)
R_EES_ENUM_CMD_STARTUP	0x01	Startup processing
R_EES_ENUM_CMD_WRITE	0x02	Write processing
R_EES_ENUM_CMD_READ	0x03	Read processing
R_EES_ENUM_CMD_REFRESH	0x04	Refresh processing
R_EES_ENUM_CMD_FORMAT	0x06	Format processing
R_EES_ENUM_CMD_SHUTDOWN	0x07	Shutdown processing

- e_ees_ret_status (enumerated-type variable name: e_ees_ret_status_t)
- EES return values

Symbol Name	Value	Description
R_EES_ENUM_RET_STS_OK	0x00	Normal end
R_EES_ENUM_RET_STS_BUSY	0x01	Busy
R_EES_ENUM_RET_ERR_CONFIGURATION	0x82	EES configuration error
R_EES_ENUM_RET_ERR_INITIALIZATION	0x83	EES initialization error
R_EES_ENUM_RET_ERR_ACCESS_LOCKED	0x84	EEPROM emulation lock error
R_EES_ENUM_RET_ERR_PARAMETER	0x85	Parameter error
R_EES_ENUM_RET_ERR_WEAK	0x86	Weak error
R_EES_ENUM_RET_ERR_REJECTED	0x87	Reject error
R_EES_ENUM_RET_ERR_NO_INSTANCE	0x88	No instance
R_EES_ENUM_RET_ERR_POOL_FULL	0x89	Pool full error
R_EES_ENUM_RET_ERR_POOL_INCONSISTENT	0x8A	EES block Inconsistency error
R_EES_ENUM_RET_ERR_POOL_EXHAUSTED	0x8B	EES block exhaustion error
R_EES_ENUM_RET_ERR_INTERNAL	0x8C	Internal error
R_EES_ENUM_RET_ERR_FLASH_SEQ	0x8D	Flash sequencer error

- e_ees_exrfd_ret_status (enumerated-type variable name: e_ees_exrfd_ret_status_t)
  These enumeration types are used internally by EES. It does not need to be used directly by the user.
  RFD control functions for EES return values

Symbol Name	Value	Description
R_EES_EXRFD_ENUM_RET_STS_OK	0x00	Normal end
R_EES_EXRFD_ENUM_RET_STS_BUSY	0x01	Busy
R_EES_EXRFD_ENUM_RET_ERR_CONFIGURATION	0x10	Configuration error
R_EES_EXRFD_ENUM_RET_ERR_INITIALIZATION	0x11	Initialization error
R_EES_EXRFD_ENUM_RET_ERR_REJECTED	0x12	Reject error
R_EES_EXRFD_ENUM_RET_ERR_PARAMETER	0x13	Parameter error
R_EES_EXRFD_ENUM_RET_ERR_INTERNAL	0x14	Internal error
R_EES_EXRFD_ENUM_RET_ERR_MODE_MISMATCHED	0x20	Mode mismatch error
R_EES_EXRFD_ENUM_RET_ERR_CFDF_SEQUENCER	0x21	Code/data flash area sequencer error
R_EES_EXRFD_ENUM_RET_ERR_ERASE	0x22	Erase operation error
R_EES_EXRFD_ENUM_RET_ERR_BLANKCHECK	0x23	Blank check operation error
R_EES_EXRFD_ENUM_RET_ERR_WRITE	0x24	Write operation error

5.7 Specifications of API Functions

This section describes the detailed specifications of the API functions of EEPROM Emulation Software (EES) RL78 Type 03.

There are some prerequisites for using the API functions of EES RL78 Type 03 to reprogram the data flash memory. If the prerequisites are not satisfied, execution of the API functions may result in indeterminate operation.

Prerequisites:

- Execute the R_EES_Init function once before starting the use of EES functions.
- The high-speed on-chip oscillator must be active while self-programming is in progress. Execute API functions of EES RL78 Type 03 only while the high-speed on-chip oscillator is active.
- To control the data flash memory, execute API functions of EES RL78 Type 03 while access to the data flash memory is enabled. For the method of enabling access to the data flash memory, refer to “User’s Manual: Hardware” of a target device.

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

Description format:

Information:

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

Details of Specifications:

The operation of this function is described.

Notes:

Conditions of usage or restrictions on this function are described.

5.7.1 Specifications of API Functions for EES RL78 Type 03

This section describes the API functions used for EES RL78 Type 03.

5.7.1.1 R_EES_Init

Information:

Syntax	R_EES_FAR_FUNC e_ees_ret_status_t R_EES_Init(uint8_t i_u08_cpu_frequency);	
Reentrancy	Non-reentrant	
Parameters (IN)	uint8_t i_u08_cpu_frequency	CPU operating frequency 〔2 - 40 (MHz)〕
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_ees_ret_status_t	R_RFD_ENUM_RET_STS_OK: 0x00 [Normal end〕 R_EES_ENUM_RET_ERR_CONFIGURATION: 0x82 [EES configuration error]
Description	Initializes internal data and variables and checks the descriptor configuration.	
Preconditions	Execute this function while the high-speed on-chip oscillator is active.	
Remarks	Execute this function once before starting the use of EES functions.	

Details of Specifications:

- Set the parameter (CPU operating frequency) to the R_EES_EXRFD_Init function and execute it.

Notes:

- When the configuration for executing the EEPROM emulation such as EES pool or EES block size is abnormal, the return value will return a EES configuration error (R_EES_ENUM_RET_ERR_CONFIGURATION).
- The high-speed on-chip oscillator needs to be kept active while EEPROM emulation is in progress. Execute this function while the high-speed on-chip oscillator is active.

  * EES RL78 Type 03 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 “User’s Manual: Hardware” of a target device.

5.7.1.2 R_EES_Open

Information:

Syntax	R_EES_FAR_FUNC e_ees_ret_status_t R_EES_Open(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_ees_ret_status_t	R_EES_ENUM_RET_STS_OK: 0x00 [Normal end] R_EES_ENUM_RET_ERR_ REJECTED: 0x87 [Reject error]
Description	EEPROM emulation preparation processing. This function makes the EEPROM emulation executable.	
Preconditions	R_EES_Init function must have finished normally.	
Remarks	-	

Details of Specifications:

- Execute the R_EES_EXRFD_Open function to make the data flash memory accessible.

Notes:

- When the R_EES_Init function is not executed and the internal variable has not been initialized, the return value will return a reject error (R_EES_ENUM_RET_ERR_REJECTED).

5.7.1.3 R_EES_Close

Information:

Syntax	R_EES_FAR_FUNC e_ees_ret_status_t R_EES_Close(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_ees_ret_status_t	R_EES_ENUM_RET_STS_OK: 0x00 [Normal end]
Description	EEPROM emulation end processing. This function makes the EEPROM emulation un-executable.	
Preconditions	-	
Remarks	-	

Details of Specifications:

- Executes the R_EES_EXRFD_Close function and finishes the EEPROM emulation.

Notes:

- If EEPROM emulation was executed, the R_EES_ENUM_CMD_SHUTDOWN command must be used to set EEPROM emulation to the stopped state (the open state).

5.7.1.4 R_EES_Execute

Information:

Syntax	R_EES_FAR_FUNC void R_EES_Execute(st_ees_request_t __near * ionp_st_ees_request);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	st_ees_request_t __near * ionp_st_ees_request	Pointer to the request structure (st_ees_request_t)
Parameters (OUT)	N/A	
Return Value	N/A	
Description	EEPROM emulation execution function. Each type of processing for performing EEPROM emulation operations is specified for this function as an argument in the command format, and the processing is executed.	
Preconditions	R_EES_Init and R_EES_Open function must have finished normally.	
Remarks	-	

Details of Specifications:

- Starts processing of the command set in the Request structure.

Notes:

- The R_EES_Execute function starts command processing and then immediately returns the control to the user program. The command processing is continued by executing the R_EES_Handler function. Therefore, the R_EES_Handler function must be executed continuously until the command processing is completed.
- Execute the repeat the R_EES_Handler function while the e_status of the Request structure(st_ees_request_t) is R_EES_ENUM_RET_STS_BUSY.
- It is not allowed to call R_EES_Execute function in an interrupt service routine.

Command Execution States (e_status) of R_EES_Execute and R_EES_Handler (1/2)

Command Execution Status	Category	Description	Corresponding Commands
R_EES_ENUM_RET_STS_OK	Meaning	Normal end	All commands
	Cause	None	
	Action to be taken	None	
R_EES_ENUM_RET_STS_BUSY	Meaning	A command is being executed.	Commands other than R_EES_ENUM_CMD_SHUTDOWN
	Cause	None	
	Action to be taken	Keep calling the R_EES_Handler function until the status changes.	
R_EES_ENUM_RET_ERR_INITIALIZATION	Meaning	Initialization error	All commands
	Cause	R_EES_Init, and R_EES_Open functions have not been finished normally.	
	Action to be taken	Normally finish the R_EES_Init, and R_EES_Open functions.	
R_EES_ENUM_RET_ERR_ACCESS_LOCKED	Meaning	EEPROM emulation lock error	Commands other than R_EES_ENUM_CMD_STARTUP and R_EES_ENUM_CMD_FORMAT.
	Cause	EEPROM emulation cannot be executed.	
	Action to be taken	Make sure that the R_EES_ENUM_CMD_STARTUP command has finished normally.	
R_EES_ENUM_RET_ERR_PARAMETER	Meaning	Parameter error	All commands
	Cause	An incorrect command parameter has been specified.	
	Action to be taken	Check the specified parameter.	
R_EES_ENUM_RET_ERR_WEAK	Meaning	The writing of an active block header or the last written stored data has not completed successfully.	R_EES_ENUM_CMD_STARTUP
	Cause	Write processing an active block header or stored data may have been interrupted.	
	Action to be taken	Execute the R_EES_ENUM_CMD_REFRESH command.	
R_EES_ENUM_RET_ERR_REJECTED	Meaning	Reject error	All commands
	Cause	A different command is being executed.	
	Action to be taken	Call the R_EES_Handler function to terminate the ongoing command.	

Command Execution States (e_status) of R_EES_Execute and R_EES_Handler (2/2)

Command Execution Status	Category	Description	Corresponding Commands
R_EES_ENUM_RET_ERR_NO_INSTANCE	Meaning	No-write-data error	R_EES_ENUM_CMD_READ
	Cause	The specified identifier data has not been written.	
	Action to be taken	Write data to the identifier specified using the R_EES_ENUM_CMD_WRITE command.	
R_EES_ENUM_RET_ERR_POOL_FULL	Meaning	Pool full error	R_EES_ENUM_CMD_WRITE
	Cause	There is no area that can be used to write the data.	
	Action to be taken	Execute the R_EES_ENUM_CMD_REFRESH command and restart writing data.	
R_EES_ENUM_RET_ERR_POOL_INCONSISTENT	Meaning	EES block inconsistency error	R_EES_ENUM_CMD_STARTUP
	Cause	An EES block has the undefined state (such as there are no active blocks).	
	Action to be taken	Execute the R_EES_ENUM_CMD_FORMAT command to initialize the EES blocks.	
R_EES_ENUM_RET_ERR_POOL_EXHAUSTED	Meaning	EES block exhaustion error	R_EES_ENUM_CMD_STARTUP R_EES_ENUM_CMD_FORMAT R_EES_ENUM_CMD_REFRESH R_EES_ENUM_CMD_WRITE
	Cause	There are no more EES blocks that can be used to continue.	
	Action to be taken	Stop EEPROM emulation. You can try restoration by executing the R_EES_ENUM_CMD_FORMAT command (erasing all existing data) or read existing data	
R_EES_ENUM_RET_ERR_INTERNAL	Meaning	Internal error	Commands other than R_EES_ENUM_CMD_SHUTDOWN
	Cause	An unexpected error has occurred.	
	Action to be taken	The EES should be stopped. Check the device state.	
R_EES_ENUM_RET_ERR_FLASH_SEQ	Meaning	Flash area sequencer error	Commands other than R_EES_ENUM_CMD_SHUTDOWN
	Cause	EES failed to change flash memory mode or start flash sequencer.	
	Action to be taken	The EES should be stopped. Check whether flash memory operation using RFD RL78 Type 03 is executed besides operation of an EEPROM emulation.	

5.7.1.5 R_EES_Handler

Information:

Syntax	R_EES_FAR_FUNC void R_EES_Handler(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	N/A	
Description	Continuous EEPROM emulation execution processing. This function is used to check for the completion of processing while allowing processing of EEPROM emulation specified by the R_EES_Execute function to continue.	
Preconditions	R_EES_Init and R_EES_Open function must have finished normally.	
Remarks	-	

Details of Specifications:

- Continues processing the EEPROM emulation initiated by the R_EES_Execute function.

Notes:

- While "e_status" of the request structure (st_ees_request_t) is R_EES_ENUM_RET_STS_BUSY, execute this function repeatedly.
- It is not allowed to call R_EES_Handler() in an interrupt service routine.
- The command execution status of the R_EES_Handler function is set for the "st_ees_request_t * ionp_st_ees_request" used as an argument of the R_EES_Execute function. Therefore, when using the R_EES_Handler function, do not free the "st_ees_request_t * ionp_st_ees_request" variable.

5.7.1.6 R_EES_GetSpace

Information:

Syntax	R_EES_FAR_FUNC e_ees_ret_status_t R_EES_GetSpace(uint16_t __near * onp_u16_space);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	uint16_t __near * onp_u16_space	Pointer to variable that contains free space information for the current active block.
Return Value	e_ees_ret_status_t	R_EES_ENUM_RET_STS_OK: 0x00 [Normal end] R_EES_ENUM_RET_ERR_INITIALIZATION: 0x83 [EES initialization error] R_EES_ENUM_RET_ERR_ACCESS_LOCKED: 0x84 [EEPROM emulation lock error] R_EES_ENUM_RET_ERR_REJECTED: 0x87 [Reject error]
Description	Gets the free space of the active block.	
Preconditions	R_EES_Init and R_EES_Open function must have finished normally. R_EES_Execute function and the R_EES_ENUM_CMD_STARTUP command must be executed successfully before.	
Remarks	-	

Details of Specifications:

- Calculate the free space of the active block.

Notes:

- When the R_EES_Init function is not executed and the internal variable has not been initialized, the return value will return a EES initialization error (R_EES_ENUM_RET_ERR_INITIALIZATION).
- When the R_EES_ENUM_CMD_STARTUP command does not finish normally with the R_EES_Execute function, the return value will return a EEPROM emulation lock error (R_EES_ENUM_RET_ERR_ACCESS_LOCKED).
- When the R_EES_Execute function is executing a EES command, the return value will return a Reject error (R_EES_ENUM_RET_ERR_REJECTED).
- In case the EES pool is exhausted the returned space value will always be 0x0000.
- When the write operation of the “active block header” or “stored data written” may have been interrupted, 0x0000 is returned to the free space.
- When an error value is returned, the free space information is not collected.

5.7.2 RFD control API Functions for EES

This section describes the RFD control API functions for EES. These functions are called from the EES control function. Do not call it directly from a user program.

Information:

Syntax	R_EES_FAR_FUNC e_ees_exrfd_ret_status_t R_EES_EXRFD_Init(uint8_t i_u08_cpu_frequency);
Description	Initializes RFD RL78 Type 03.

Information:

Syntax	R_EES_FAR_FUNC e_ees_exrfd_ret_status_t R_EES_EXRFD_Open(void);
Description	Set the data flash control register (DFLCTL) to the state where accessing the data flash memory is permitted (DFLEN = 1).

Information:

Syntax	R_EES_FAR_FUNC e_ees_exrfd_ret_status_t R_EES_EXRFD_Close(void);
Description	Set the data flash control register (DFLCTL) to the state where access to the data flash memory is inhibited (DFLEN = 0). All ongoing EES processing stop.

Information:

Syntax	R_EES_FAR_FUNC e_ees_exrfd_ret_status_t R_EES_EXRFD_Erase(uint8_t i_u08_virtual_block_number);
Description	Start erasing the EES block (one virtual block).

Information:

Syntax	R_EES_FAR_FUNC e_ees_exrfd_ret_status_t R_EES_EXRFD_Write(uint16_t i_u16_offset_addr, uint8_t __near * inp_u08_write_data, uint16_t i_u16_size);
Description	Starts writing to the specified the data flash memory address (one byte).

Information:

Syntax	R_EES_FAR_FUNC e_ees_exrfd_ret_status_t R_EES_EXRFD_BlankCheck(uint16_t i_u16_offset_addr, uint16_t i_u16_size);
Description	Starts Blank check to the specified the data flash memory address.

Information:

Syntax	R_EES_FAR_FUNC e_ees_exrfd_ret_status_t R_EES_EXRFD_Read(uint16_t i_u16_offset_addr, uint8_t __near * onp_u08_read_data, uint16_t i_u16_size);
Description	Reads the specified address in the data flash memory.

Information:

Syntax	R_EES_FAR_FUNC e_ees_exrfd_ret_status_t R_EES_EXRFD_Handler(void);
Description	Continues processing of the RFD control function for EES that is executing, and confirms termination.

Information:

Syntax	static R_EES_FAR_FUNC e_ees_exrfd_ret_status_t r_ees_exrfd_get_seq_error_status(void);
Description	Obtain the execution result from the flash memory sequencer.

Information:

Syntax	static R_EES_FAR_FUNC e_ees_exrfd_ret_status_t r_ees_exrfd_finish_state(void);
Description	Sets the RFD control functions for EES to the end status.

Information:

Syntax	static R_EES_FAR_FUNC e_ees_exrfd_ret_status_t r_ees_exrfd_check_cmd_executable(void);
Description	Check the status and flags of the RFD control functions for EES.

Information:

Syntax	static R_EES_FAR_FUNC bool r_ees_exrfd_is_valid_byte_parameter(uint16_t i_u16_offset_addr, uint16_t i_u16_size);
Description	Check the parameters used by the RFD Control functions for EES.

Information:

Syntax	static R_EES_FAR_FUNC void r_ees_exrfd_clear_cmd_workarea(void);
Description	Clears the data area used by the RFD control functions for EES.

5.7.3 Internal Functions for the EES

This section describes the functions used by the EES functions. These functions are internal functions called from the EES functions. Do not call it directly from a user program.

Information:

Syntax	R_EES_FAR_FUNC bool r_ees_is_valid_configuration(void);
Description	Check the EES configuration and initialize the internal status.

Information:

Syntax	R_EES_FAR_FUNC bool r_ees_is_valid_requester(st_ees_request_t __near * ionp_st_ees_request);
Description	Check “request structure” and “EES status” and update internal status.

Information:

Syntax	R_EES_FAR_FUNC void r_ees_fsm_startup_state_00(void); ～ R_EES_FAR_FUNC void r_ees_fsm_startup_state_09(void);
Description	Updates the internal status for startup processing.

Information:

Syntax	R_EES_FAR_FUNC void r_ees_fsm_write_state_00(void); ～ R_EES_FAR_FUNC void r_ees_fsm_write_state_04(void);
Description	Updates the internal status for write processing.

Information:

Syntax	R_EES_FAR_FUNC void r_ees_fsm_read_state_00(void); ～ R_EES_FAR_FUNC void r_ees_fsm_read_state_01(void);
Description	Updates the internal status for read processing.

Information:

Syntax	R_EES_FAR_FUNC void r_ees_fsm_refresh_state_00(void); ～ R_EES_FAR_FUNC void r_ees_fsm_refresh_state_17(void);
Description	Updates the internal status for refresh processing.

Information:

Syntax	R_EES_FAR_FUNC void r_ees_fsm_format_state_00(void); ～ R_EES_FAR_FUNC void r_ees_fsm_format_state_11(void);
Description	Updates the internal status for format processing.

Information:

Syntax	R_EES_FAR_FUNC void r_ees_fsm_shutdown_state_00(void);
Description	Execute the shutdown processing of the EEPROM emulation.

Information:

Syntax	R_EES_FAR_FUNC void r_ees_fsm_exrfd_cmd_erase_state_00(void);
Description	Start the erase processing.

Information:

Syntax	R_EES_FAR_FUNC void r_ees_fsm_exrfd_cmd_bw_state_00(void);
Description	Starts the blank check and write processing.

Information:

Syntax	R_EES_FAR_FUNC void r_ees_fsm_exrfd_cmd_inner_blankcheck_state_00(void);
Description	Start internal processing of the blank check.

Information:

Syntax	R_EES_FAR_FUNC void r_ees_fsm_exrfd_cmd_write_state_00(void);
Description	Start the write processing.

Information:

Syntax	R_EES_FAR_FUNC void r_ees_fsm_exrfd_cmd_inner_write_state_00(void);
Description	Start internal processing of the write.

Information:

Syntax	R_EES_FAR_FUNC void r_ees_fsm_exrfd_cmd_read_state_00(void);
Description	Start the read processing.

Information:

Syntax	R_EES_FAR_FUNC void r_ees_fsm_exrfd_cmd_state_01(void);
Description	Proceed with the internal processing of the executed RFD control functions for EES.

Information:

Syntax	R_EES_FAR_FUNC void r_ees_fsm_exit_state(void);
Description	Dummy processing.

Information:

Syntax	static R_EES_FAR_FUNC uint8_t r_ees_calculate_next_a_flag(uint8_t i_u08_a_flag_value);
Description	Calculates the value of the A flag.

Information:

Syntax	static R_EES_FAR_FUNC void r_ees_fsm_finish_command(void);
Description	Terminates the execution command.

Information:

Syntax	static R_EES_FAR_FUNC void r_ees_fsm_swap_acvive_block_info(void);
Description	Swaps the active block information.

Information:

Syntax	static R_EES_FAR_FUNC bool r_ees_fsm_exrfd_cmd_detect_fatal_error(e_ees_exrfd_ret_status_t i_e_ees_exrfd_ret_value);
Description	Check the results of the RFD control function for the EES for errors that make the EES unsustainable.

Information:

Syntax	static R_EES_FAR_FUNC e_ees_block_status_t r_ees_fsm_get_ees_block_status(void);
Description	Obtains the state of the EES block.

6. Sample Programs

This section describes the sample programs provided together with EES RL78 Type 03.

6.1 File Structure

6.1.1 Folder Structure

Figure 6-1 shows the structure of sample program folders.
Figure 6-1 shows an example of using RL78/F25. The installed “sample” folder contains a sample folder for each device group (e.g. RL78_F25).
The “RL78_F25” folder is used when using RL78/F22 or RL78/F25.

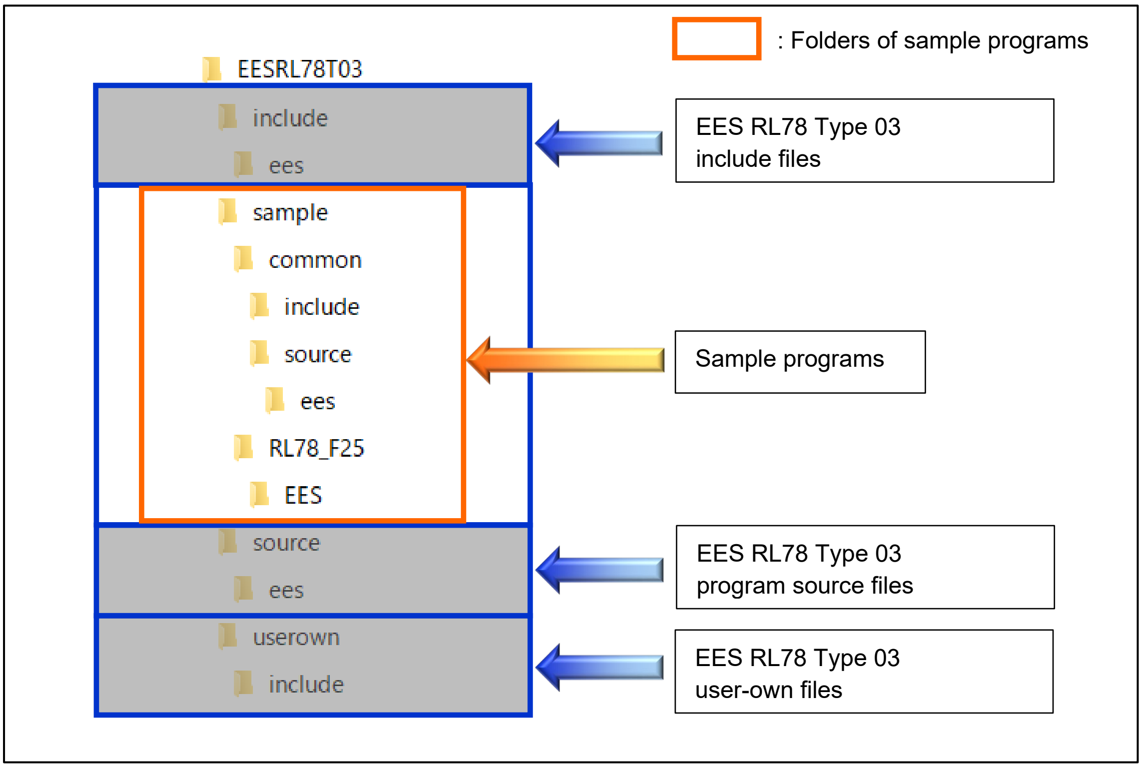


Figure 6-1 Structure of Sample Program Folders

6.1.2 List of Files

6.1.2.1 List of Source Files

Table 6-1 shows the program source file in the "sample\common\source\ees\" folder.

Table 6-1 Program Source File in the "sample\common\source\ees\" Folder

No.	Source File Name	Description
1	sample_control_ees.c	This file contains the functions for controlling the EEPROM emulation.

Table 6-2 shows the program source file of the main processing in the "sample\RL78_F25" folder.

"sample\RL78_F25\EES\[compiler name]\source\" folder

Table 6-2 Program Source File of the Main Processing

No.	Source File Name	Description
1	main.c	Sample file of the main processing functions

6.1.2.2 List of Header Files

Table 6-3 shows the program header files in the "sample\common\include\" folder.

Table 6-3 Program Header Files in the "sample\common\include\" Folder

No.	Header File Name	Description
1	sample_control_ees.h	This file defines the prototype declarations of the sample functions for controlling the EEPROM emulation.
2	sample_ees_defines.h	This file defines the macros of the sample functions for controlling the EEPROM emulation.
3	sample_ees_memmap.h	This file defines the macros that describes the sections used by the sample program that controls the EEPROM emulation.

Table 6-4 shows the program header files in the "sample\RL78_F25\EES\[compiler name]\include\" folder.

Table 6-4 Program Header Files in the "sample\RL78_F25\EES\[compiler name]\include\" Folder

No.	Header File Name	Description
1	sample_config.h	This File defines parameters value.

6.2 Data Type Definitions

6.2.1 Macro Defines

- Frequency setting macro

CPU frequency used in the sample program.

Symbol Name	Value	Description
SAMPLE_VALUE_U08_CPU_FREQUENCY	40u	CPU frequency

6.3 Sample Program Functions

Table 6-5 shows the sample program functions.

Table 6-5 List of Sample Program Functions

	API Function Name	Outline
1	main	Executes the main processing of the sample program for controlling the EES.
2	Sample_EES_Control	Write and read EES blocks according to the basic procedure for using EES.

6.3.1 Sample Program for Controlling the EEPROM Emulation

The EES RL78 Type 03 rewrite control sample follows the basic operation procedure for using EES and performs the rewrite and read processing of EES block.

Note: During EES command processing, the data in the data flash memory cannot be referenced. Copy 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/F25):

- CPU operating frequency: 40 MHz
  (The high-speed on-chip oscillator clock (HOCO) is used for the main system clock.)

Figure 6-2 shows a flowchart of the main processing of the sample program for the EES.

6.3.1.1 main Function

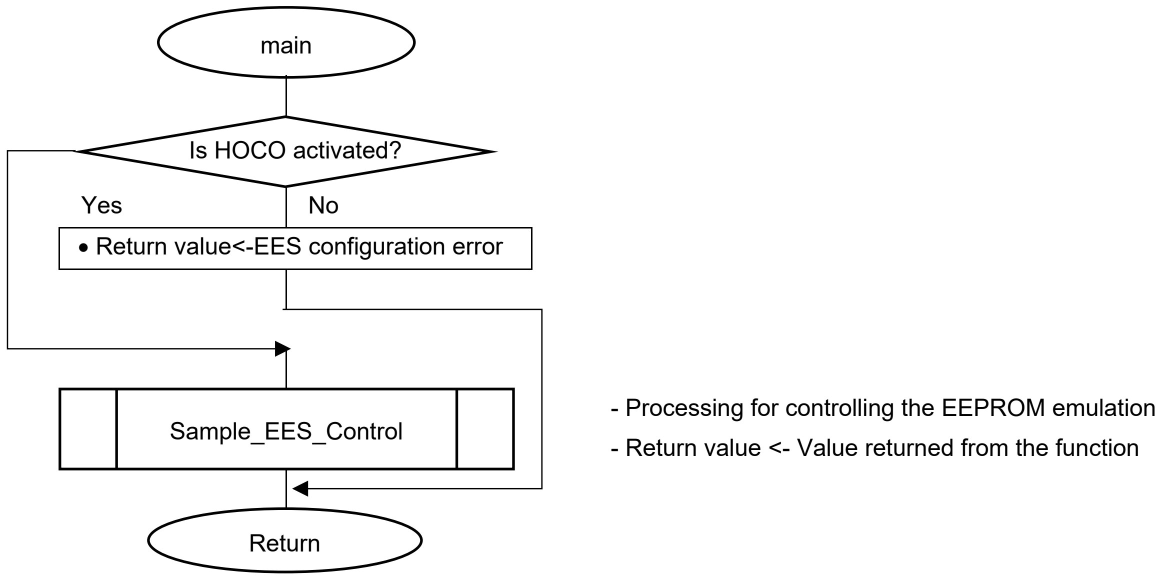


Figure 6-2 Flowchart of the Main Processing of the Sample Program for Controlling the EES

6.3.1.2 Sample_EES_Control Function

- Figure 6-3 shows the pre-processing required to use the EES and the write and read processing flow.
- Initialize the EES.

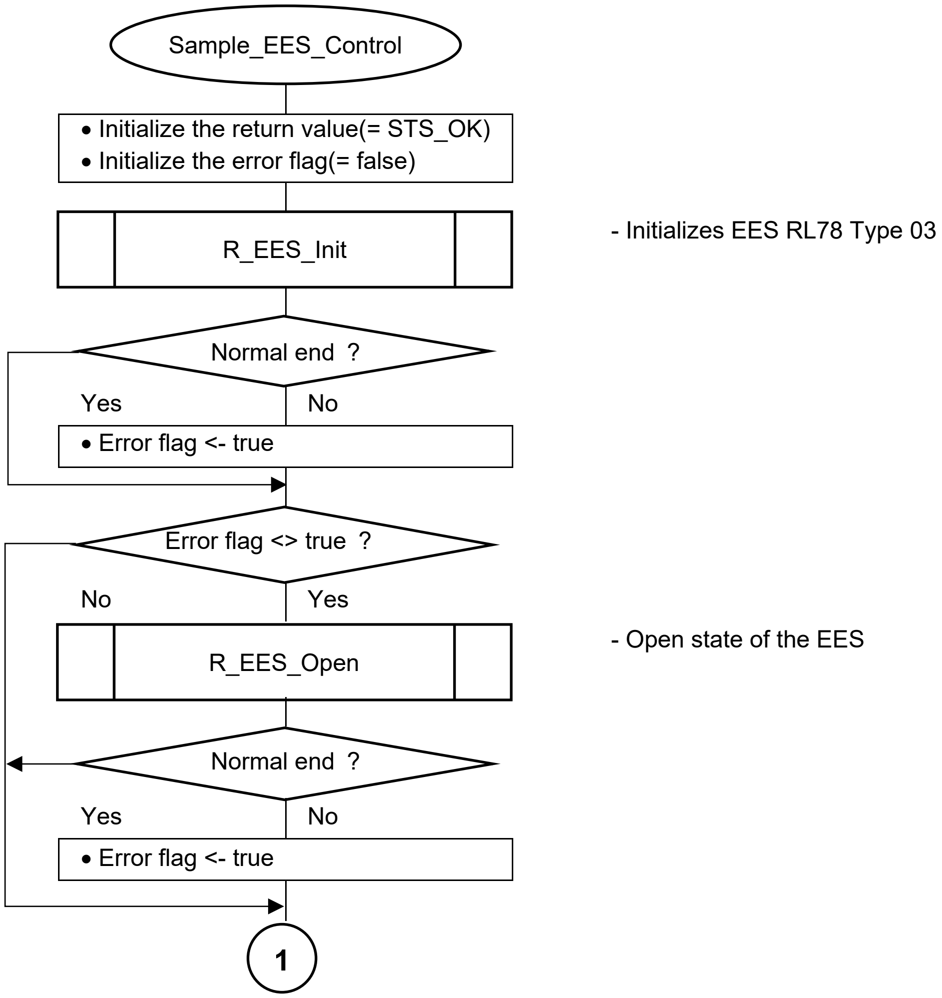


Figure 6-3 Flowchart of Sample Processing for Controlling EEPROM Emulation (1/5)

- EEPROM emulation execution startup processing.

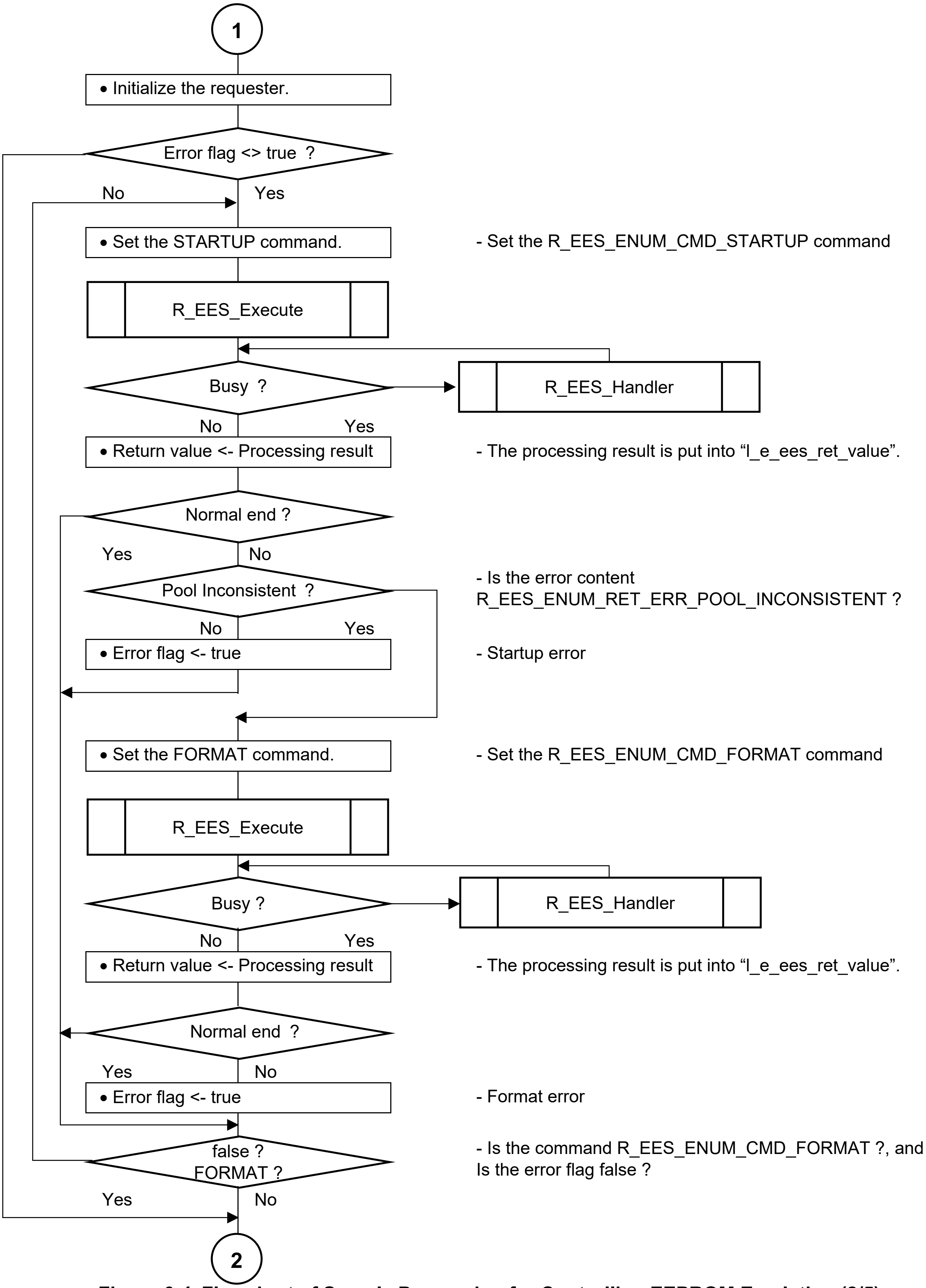


Figure 6-4 Flowchart of Sample Processing for Controlling EEPROM Emulation (2/5)

- EEPROM emulation data write processing.

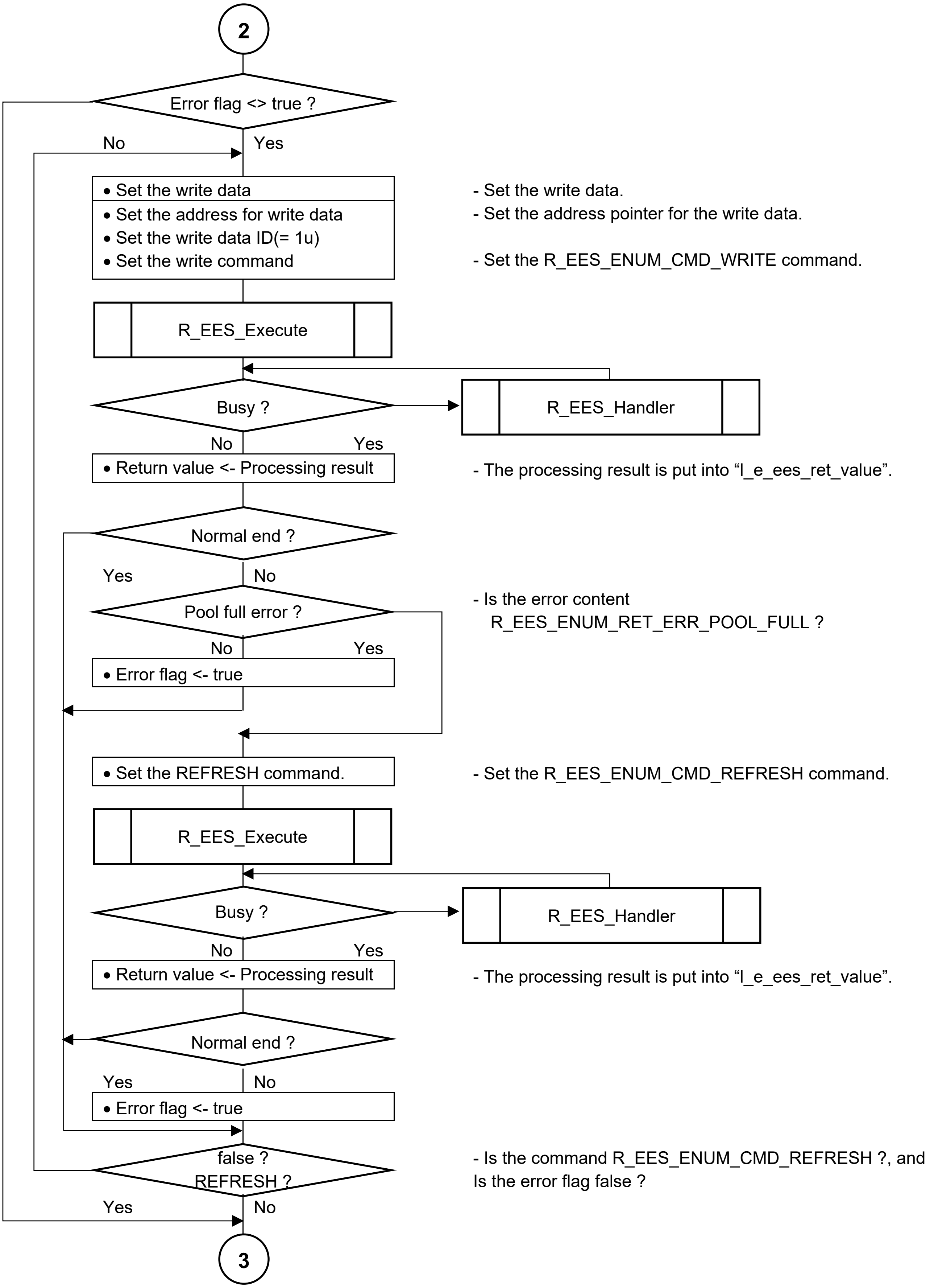


Figure 6-5 Flowchart of Sample Processing for Controlling EEPROM Emulation (3/5)

- EEPROM emulation data read processing.

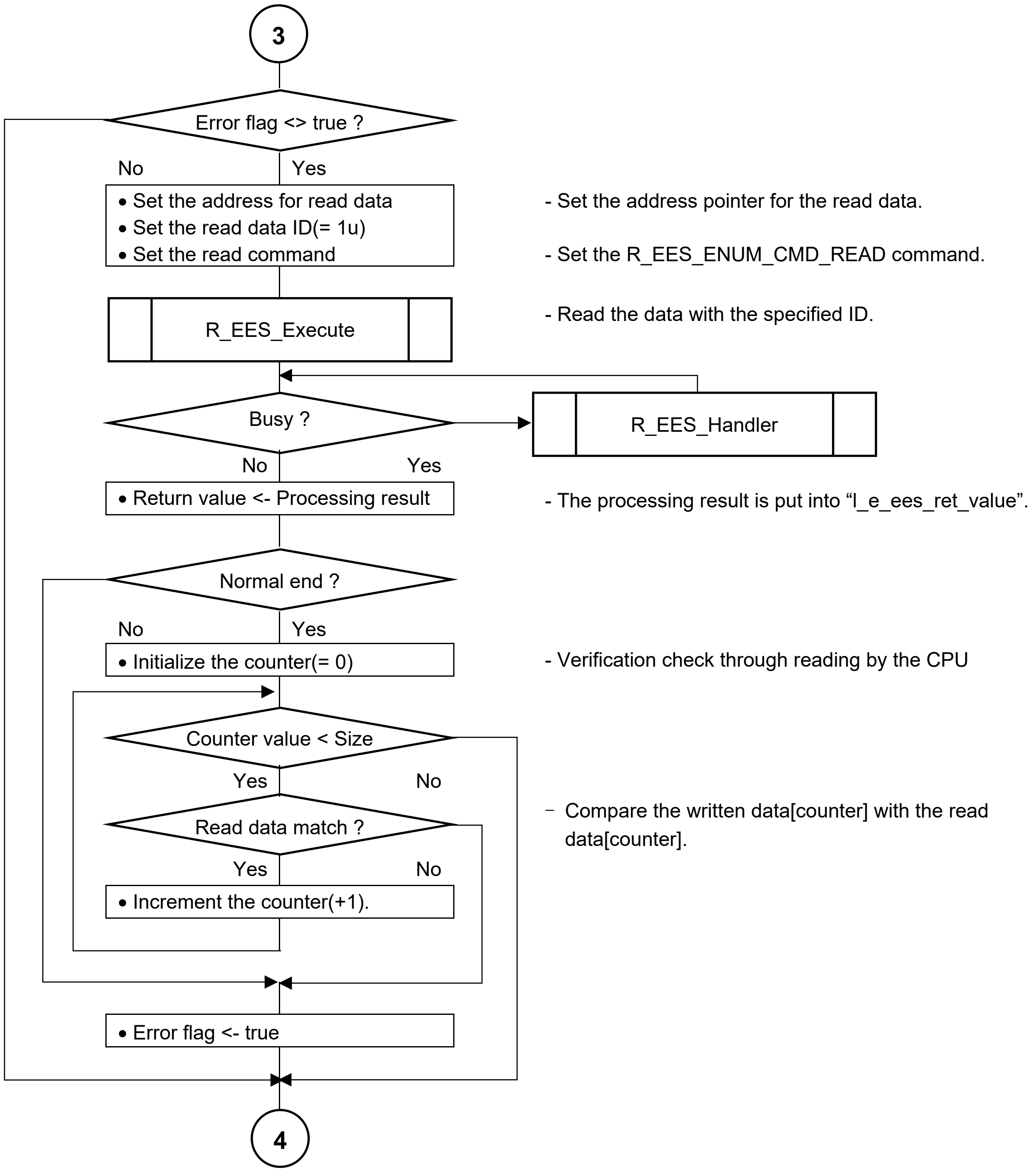


Figure 6-6 Flowchart of Sample Processing for Controlling EEPROM Emulation (4/5)

- EEPROM emulation shutdown processing.

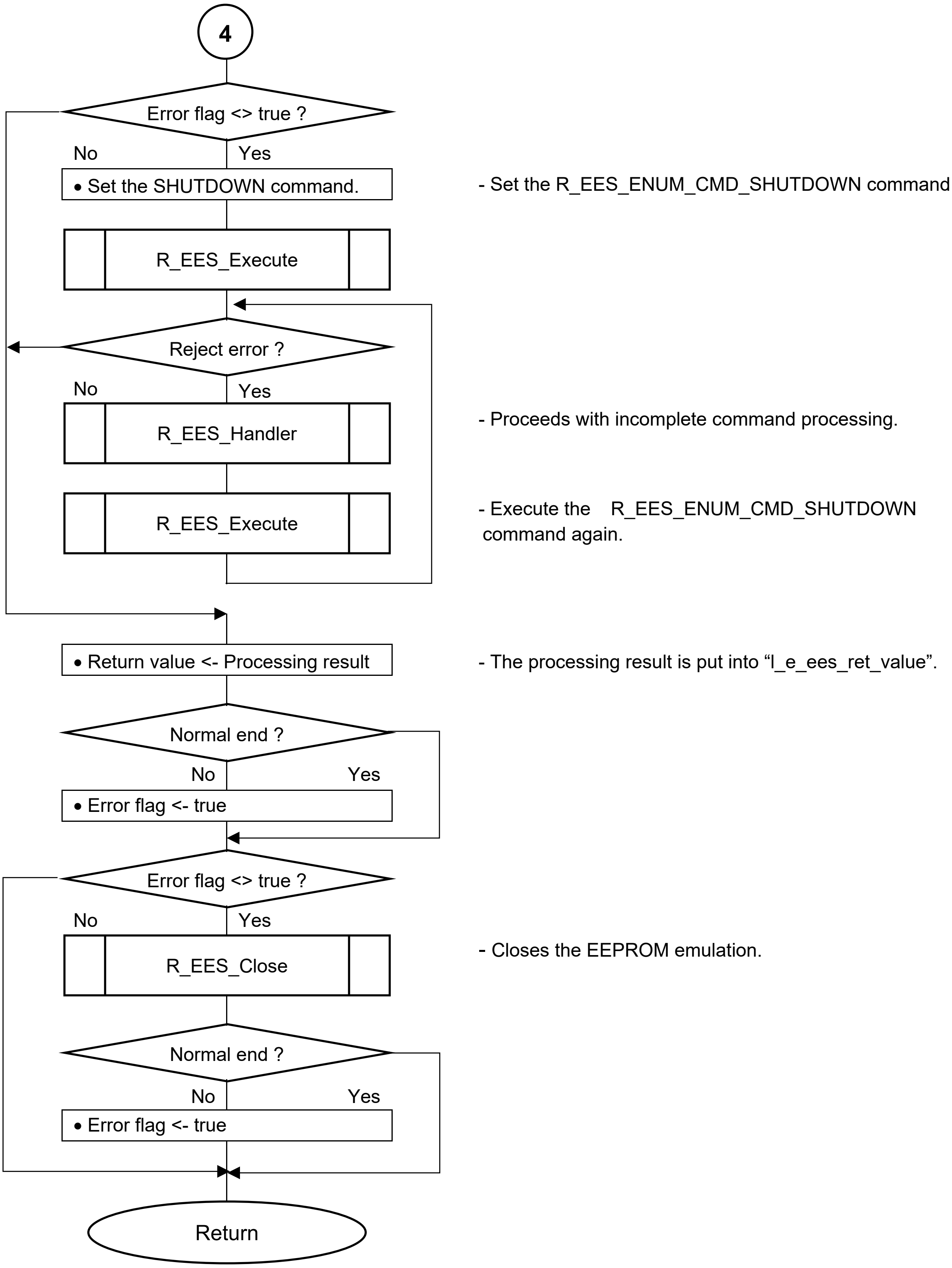


Figure 6-7 Flowchart of Sample Processing for Controlling EEPROM Emulation (5/5)

Note: Error handling and user processing for normal completion are omitted.

6.4 Specifications of Sample Program Functions

This section describes the specifications of the functions in the sample programs for EES RL78 Type 03. The sample programs for EEPROM emulation are examples of basic processing. The functions in the sample programs can be used as reference for developing an application program.

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

6.4.1 Sample Program Functions for Controlling the EEPROM Emulation

6.4.1.1 main

Information:

Syntax	int main(void);	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	int (e_ees_ret_status_t)	R_EES_ENUM_RET_STS_OK: 0x00 [Normal end] R_EES_ENUM_RET_STS_BUSY: 0x01 [Busy] R_EES_ENUM_RET_ERR_CONFIGURATION: 0x82 [EES configuration error] R_EES_ENUM_RET_ERR_INITIALIZATION: 0x83 [EES initialization error] R_EES_ENUM_RET_ERR_ACCESS_LOCKED: 0x84 [EEPROM emulation lock error] R_EES_ENUM_RET_ERR_PARAMETER: 0x85 [Parameter error] R_EES_ENUM_RET_ERR_WEAK: 0x86 [Weak error] R_EES_ENUM_RET_ERR_REJECTED: 0x87 [Reject error] R_EES_ENUM_RET_ERR_NO_INSTANCE: 0x88 [No instance] R_EES_ENUM_RET_ERR_POOL_FULL: 0x89 [Pool full error] R_EES_ENUM_RET_ERR_POOL_INCONSISTENT: 0x8A [EES block Inconsistency error] R_EES_ENUM_RET_ERR_POOL_EXHAUSTED: 0x8B [EES block exhaustion error] R_EES_ENUM_RET_ERR_INTERNAL: 0x8C [Internal error] R_EES_ENUM_RET_ERR_FLASH_SEQ: 0x8D [Flash sequencer error]
Description	Executes the main processing of the sample program for controlling the EES.	
Preconditions	-	
Remarks	-	

6.4.1.2 Sample_EES_Control

Information:

Syntax	R_EES_FAR_FUNC e_ees_ret_status_t Sample_EES_Control();	
Reentrancy	Non-reentrant	
Parameters (IN)	N/A	
Parameters (IN/OUT)	N/A	
Parameters (OUT)	N/A	
Return Value	e_ees_ret_status_t	R_EES_ENUM_RET_STS_OK: 0x00 [Normal end] R_EES_ENUM_RET_STS_BUSY: 0x01 [Busy] R_EES_ENUM_RET_ERR_CONFIGURATION: 0x82 [EES configuration error] R_EES_ENUM_RET_ERR_INITIALIZATION: 0x83 [EES initialization error] R_EES_ENUM_RET_ERR_ACCESS_LOCKED: 0x84 [EEPROM emulation lock error] R_EES_ENUM_RET_ERR_PARAMETER: 0x85 [Parameter error] R_EES_ENUM_RET_ERR_WEAK: 0x86 [Weak error] R_EES_ENUM_RET_ERR_REJECTED: 0x87 [Reject error] R_EES_ENUM_RET_ERR_NO_INSTANCE: 0x88 [No instance] R_EES_ENUM_RET_ERR_POOL_FULL: 0x89 [Pool full error] R_EES_ENUM_RET_ERR_POOL_INCONSISTENT: 0x8A [EES block Inconsistency error] R_EES_ENUM_RET_ERR_POOL_EXHAUSTED: 0x8B [EES block exhaustion error] R_EES_ENUM_RET_ERR_INTERNAL: 0x8C [Internal error] R_EES_ENUM_RET_ERR_FLASH_SEQ: 0x8D [Flash sequencer error]
Description	Write and read EES blocks according to the basic procedure for using EES.	
Preconditions	-	
Remarks	When the verification check of the read data results in an error, it is not reflected in the return value.	

7. Creating a Sample Project for EES RL78 Type 03

EES RL78 Type 03 includes a sample program to control EEPROM emulation. The compilers which can be used by EES RL78 Type 03 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/F25 (R7F125FPL) is explained to this section. When using other than RL78/F25 (R7F125FPL), section address settings must be changed by referring to the user's manual for the target device.

If the RL78/F22 is used, the RL78/F25 sample program is available.

Notes 1: The target Integrated Development Environment (IDE) and the compiler are premised on using the version for RL78/F22, F25. Be sure to use them, after confirming that RL78/F22, F25 are target products.

2: EES RL78 Type 03 uses the RFD RL78 Type 03 to control the data flash memory. However, it is not included in the EES RL78 Type 03 installer, RFD RL78 Type 03 must be installed before registering to the project. It describes the RFD RL78 Type 03 files and sections needed to register the project, however for more information on RFD RL78 Type 03, refer to the RFD RL78 Type 03 User's Manual.

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

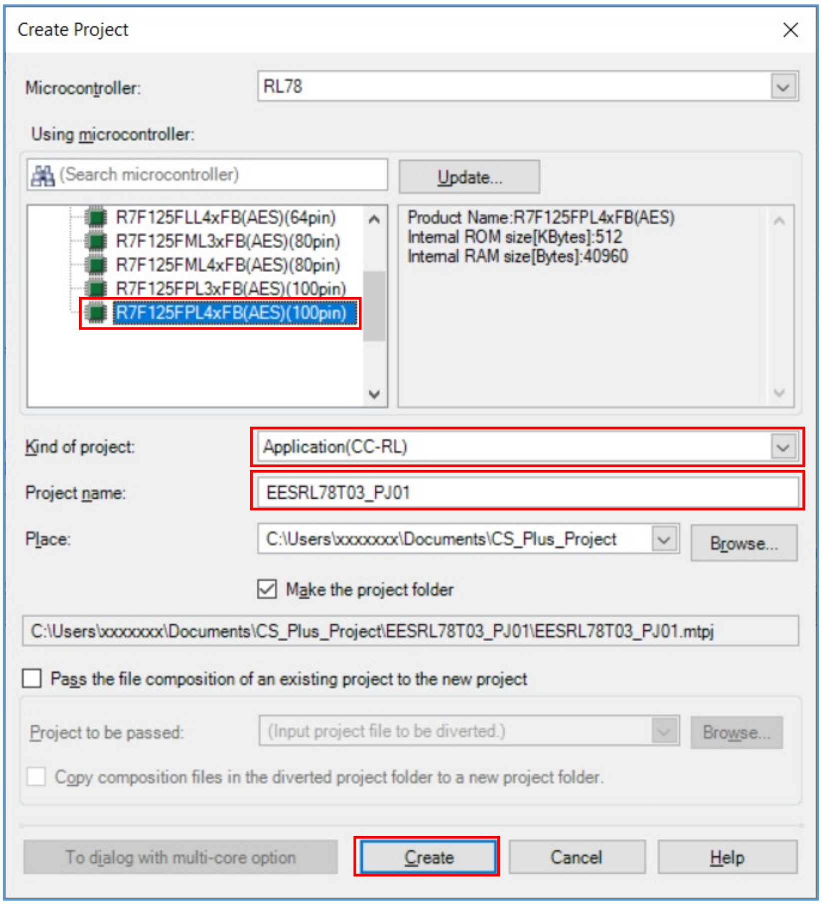
CS+ or e^2 studio can be used for a RENESAS CC-RL compiler as an IDE. EES RL78 Type 03 and RFD RL78 Type 03 are registered and built in the project created by the IDE. An example of creating a sample project in case each IDE is used is shown. Because to understand a CC-RL compiler and each IDE, it is necessary to refer to the user's manual of each tool product.

7.1.1 Example of Creating a Sample Project

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

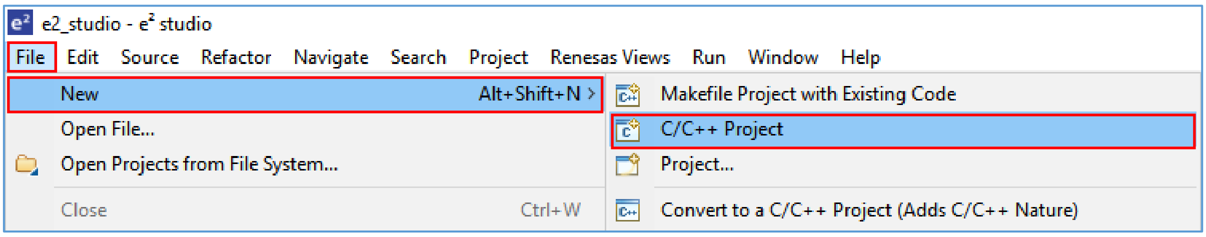
・The CS+ starts and from the [Project] menu, select [Create New Project...], the “Create Project” window will open.

- Select the product of “RL78/F25 (ROM: 512 Kbytes)” - “R7F125FPL4xFB(100pin)” as [Using microcontroller].
- Select “Application (CC-RL)” as [Kind of project].
- [Project name] is temporarily set to “EESRL78T03_PJ01”.
- When you click the [Create] button, the new project is created.

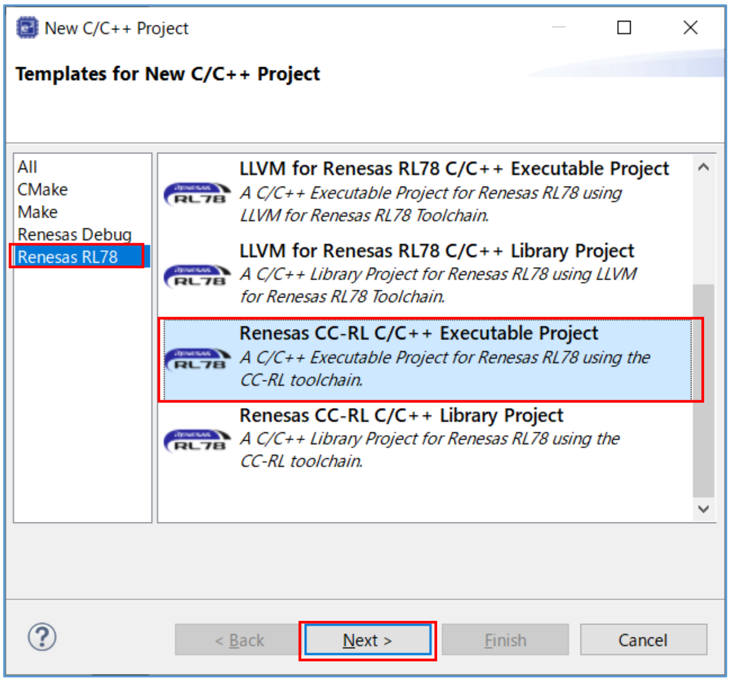


(2) An example of creating a sample project which used e² studio (IDE)

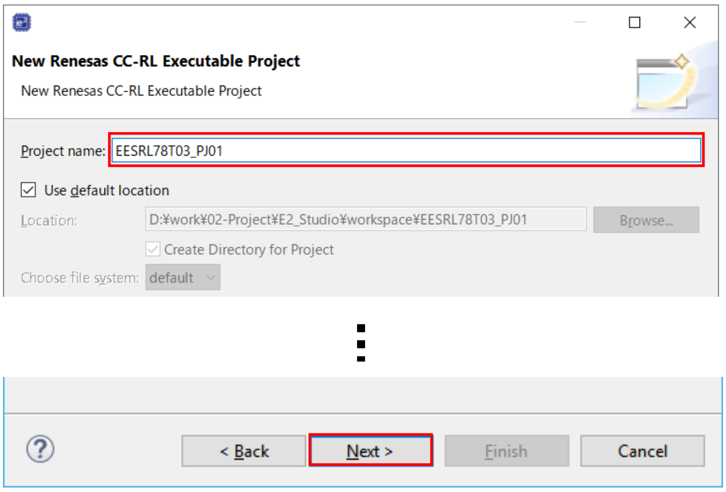
- The e² studio starts and from the [File] menu, select [New] – [C/C++ Project], the "Templates for New C/C++ Project" window will open.



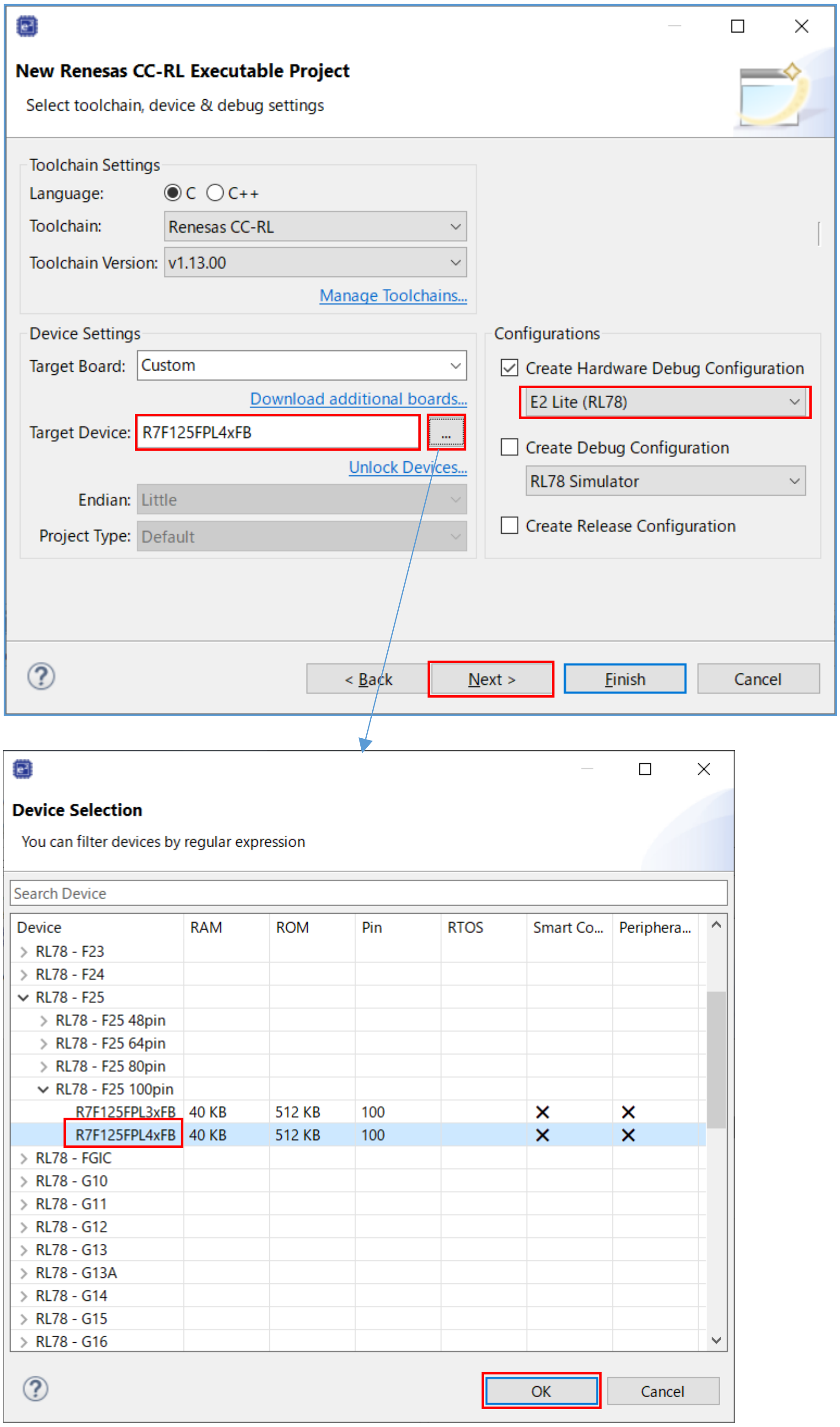
- Select [Renesas CC-RL C/C++ Executable Project] displayed after selection in [Renesas RL78], and press "next" button.



- Input "project name" on "New Renesas CC-RL Executable Project" window, and press "next" button. [Project name] is temporarily set to "EESRL78T03_PJ01".

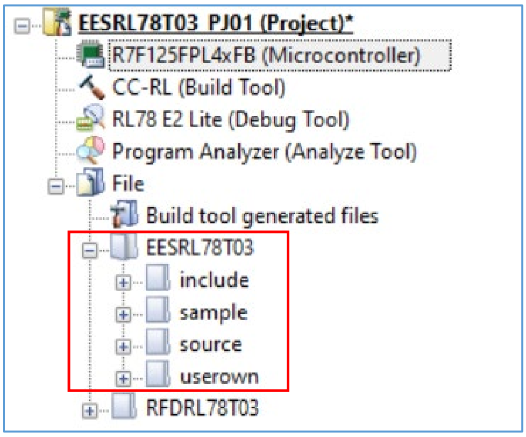


- Select the [Target Device] of [Device Settings], and select “RL78 - F25” - “RL78 - F25 100pin” - “R7F125FPL4xFB”.
- It is a premise that E2 Lite is selected as a debugging tool and on-chip debugging is executed. Put a check mark to “Create Hardware Debug Configuration” by [Configurations]. And select “E2 Lite (RL78)”.
- When press the [Next] button, the “Select Coding Assistant settings” window will be displayed, so press the [Finish] button.

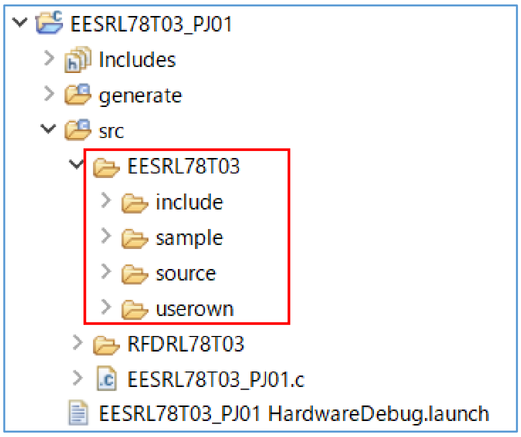


7.1.2 Example of Registration of Target Folders and Target Files

Using EES RL78 Type 03, when execute EEPROM emulation the example which registers necessary files is shown. Each folder of a “EESRL78T03” source program file is “include”, “source”, “userown”, and “sample”. As other registration methods, after all the folders of “include”, “source”, “userown”, and “sample” are registered, unnecessary files and folders can be removed using the function of “Remove from Project”(CS+) or [Resource Configuration] – [Exclude from Build] (e^2 studio).



The registration tree screen of EES (CS+)



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

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

The folder to which an I/O header file (iodefine.h) is outputted by IDE:

- CS+: [Project name] folder
- e^2 studio: [Project name]/generate folder

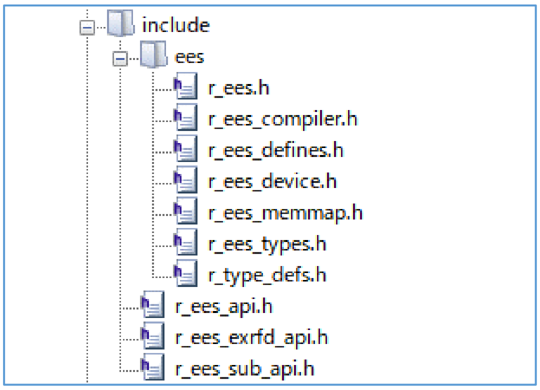
・Exclusion of the file automatically added by the function of IDE
There are files added automatically in the created project. The same file as these exists also in the “sample” folder of EES RL78 Type 03. Therefore, using the function of IDE, select those files from tree and excludes from a project.

- CS+: Click the right mouse button for the file of tree. And exclude target files using “Remove from Project” function. Target files are “hdwinit.asm and main.c” in [project name] folder.
- e^2 studio: Clicks the right mouse button for the file of tree. And on the [Settings] screen displayed by the “property”, put a check mark to [Exclude resource from build] and exclude target files (target folder). (Exclusion of a folder is also possible)
  Target files are a hdwinit.asm in [project name]/generate folder and a [project name].c (EESRL78T03_PJ01.c) in [project name]/src folder.

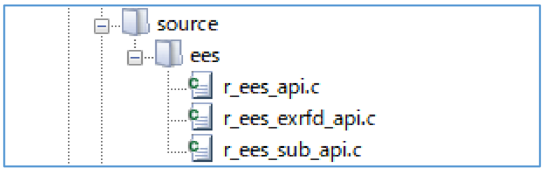
(1) Registration the EES RL78 Type 03 target folders and target files.

The folders ("include", "source", "userown", "sample") and source program files which are included in EES RL78 Type 03 to register are shown below.

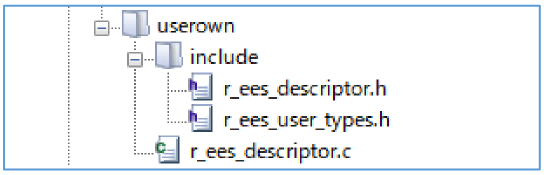
in the "include" folder



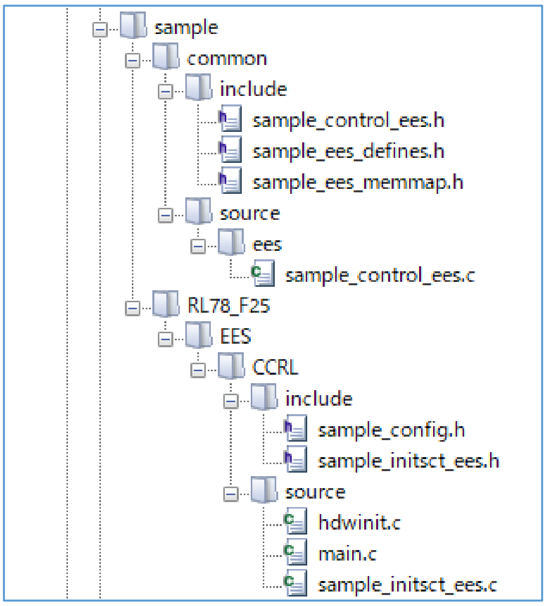
in the "source" folder



in the "userown" folder



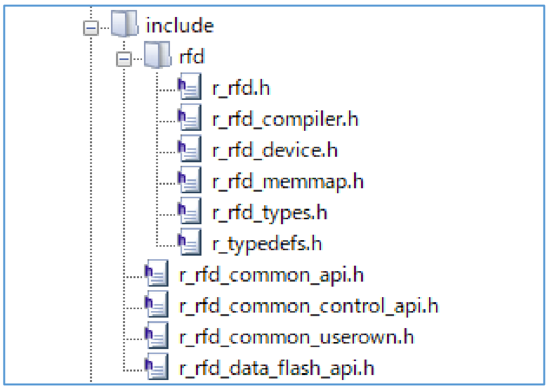
in the "sample" folder



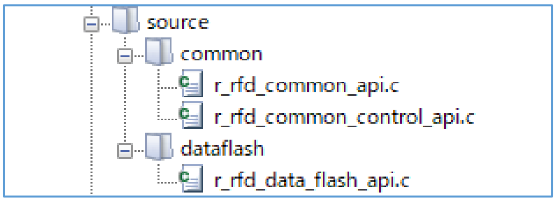
(2) Registration the RFD RL78 Type 03 target folders and target files.

The folders ("include", "source", "userown") and source program files which are included in RFD RL78 Type 03 to register are shown below.

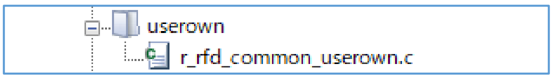
in the "include" folder



in the "source" folder



in the "userown" folder



7.1.3 Build Tool Settings

Set IDE setting necessary in order to build EES RL78 Type 03 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 (“EESRL78T03_PJ01”) in a tree, and select “Properties”. And set each setting of the build tool in the displayed window.

7.1.3.1 Include Path Settings

・Setting of the include path on CS+ inputs path in “Common Options” tab.

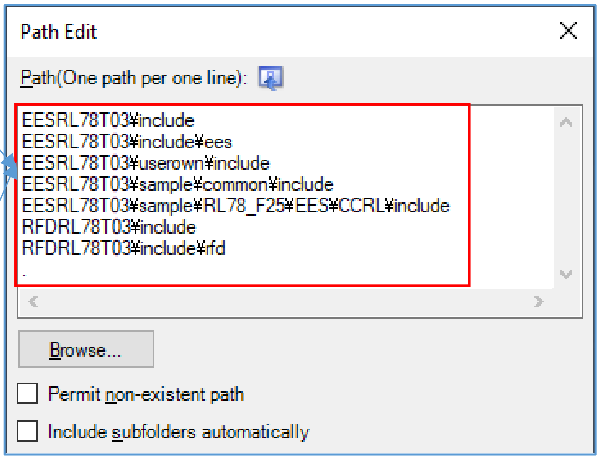
- Input the include directory path in the “Path Edit” window displayed by selection of [Frequently Used Options(for Compile)] - [Additional include paths].

(1) EES RL78 Type 03 include path

EESRL78T03\include
EESRL78T03\include\ees
EESRL78T03\userown\include
EESRL78T03\sample\common\include
EESRL78T03\sample\RL78_F25\EES\CCRL\include

(2) RFD RL78 Type 03 include path

RFDRL78T03\include
RFDRL78T03\include\rfd
.



・Setting of the include path on e[2] studio inputs path in “Properties” window.

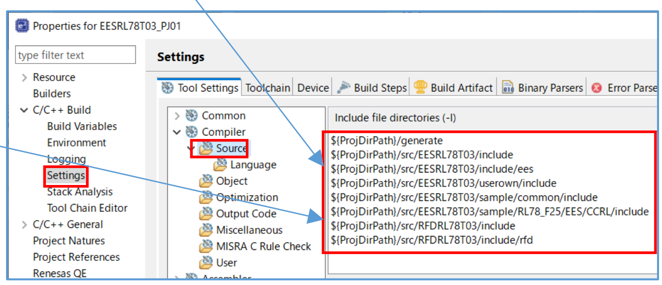
- Input the include directory path in the window displayed by selection of “C/C++ Build” [Settings] - “Compiler” [Source].

(1) EES RL78 Type 03 include path

${ProjDirPath}/generate
${ProjDirPath}/src/EESRL78T03/include
${ProjDirPath}/src/EESRL78T03/include/ees
${ProjDirPath}/src/EESRL78T03/userown/include
${ProjDirPath}/src/EESRL78T03/sample/common/include

(2) RFD RL78 Type 03 include path

${ProjDirPath}/src/RFDRL78T03/include
${ProjDirPath}/src/RFDRL78T03/include/rfd



7.1.3.2 Device Item Settings

・Setting of the device Items on CS+ inputs in the “Link Options” tab.

- 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] [The example for RL78/F25]

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/F25]

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.

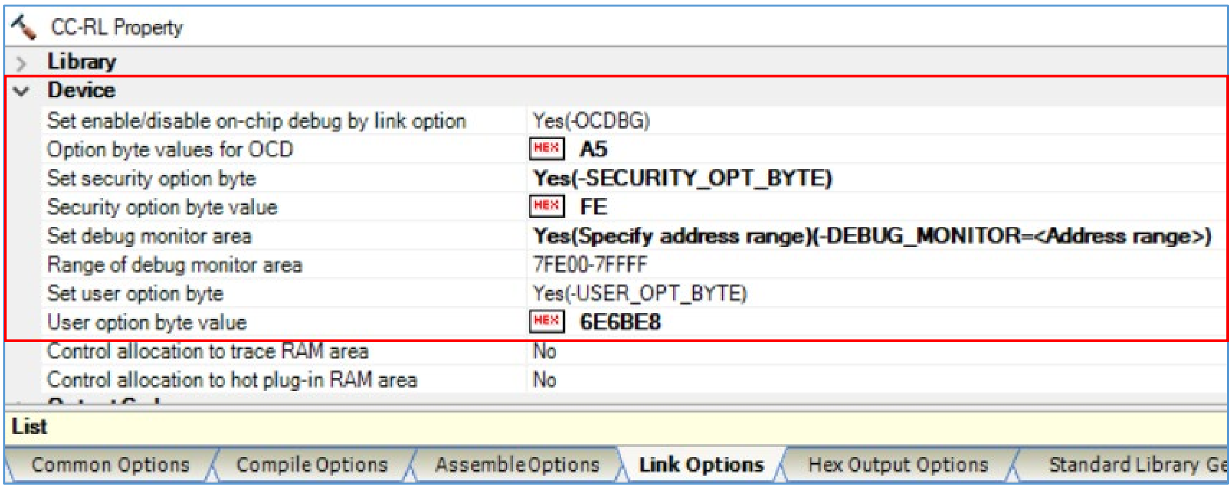
Select “Yes(Specify address range)(-OCDBG_MONITOR=<Address range>)” in [Set debug monitor area]. Set “**7FE00-7FFFF**” to [Range of debug monitor area].

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

Select “Yes(-USER_OPT_BYTE)” in [Set user option byte].

Set “**6E6BE8**” to [User option byte value]. (WDT stop, LVD reset mode, 40MHz [The example for RL78/F25])

Note: Be sure to confirm the contents of “User Option Byte” in “Option Byte” chapter on the user's manual of a target device. And describe the set value used with user application.



・Setting of the device Items on e[2] studio inputs in the “Properties” window.

- Select “C/C++ Build” [Settings] - “Linker” [Device]. And set device items on the displayed screen.

Put a check mark to [Secure memory area of OCD monitor(-debug_monitor)] in the screen.

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

Set “**7FE00-7FFFF**” to [Memory area(-debug_monitor=<start address>-<end address>)]. [The example for RL78/F25]

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 the user's manual.

Put a check mark to [Set user option byte(-user_opt_byte)].
Set “**6E6BE8**” to [User option byte value(-user_opt_byte=<value>)]. (WDT stop, LVD reset mode, 40MHz [The example for RL78/F25])

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.

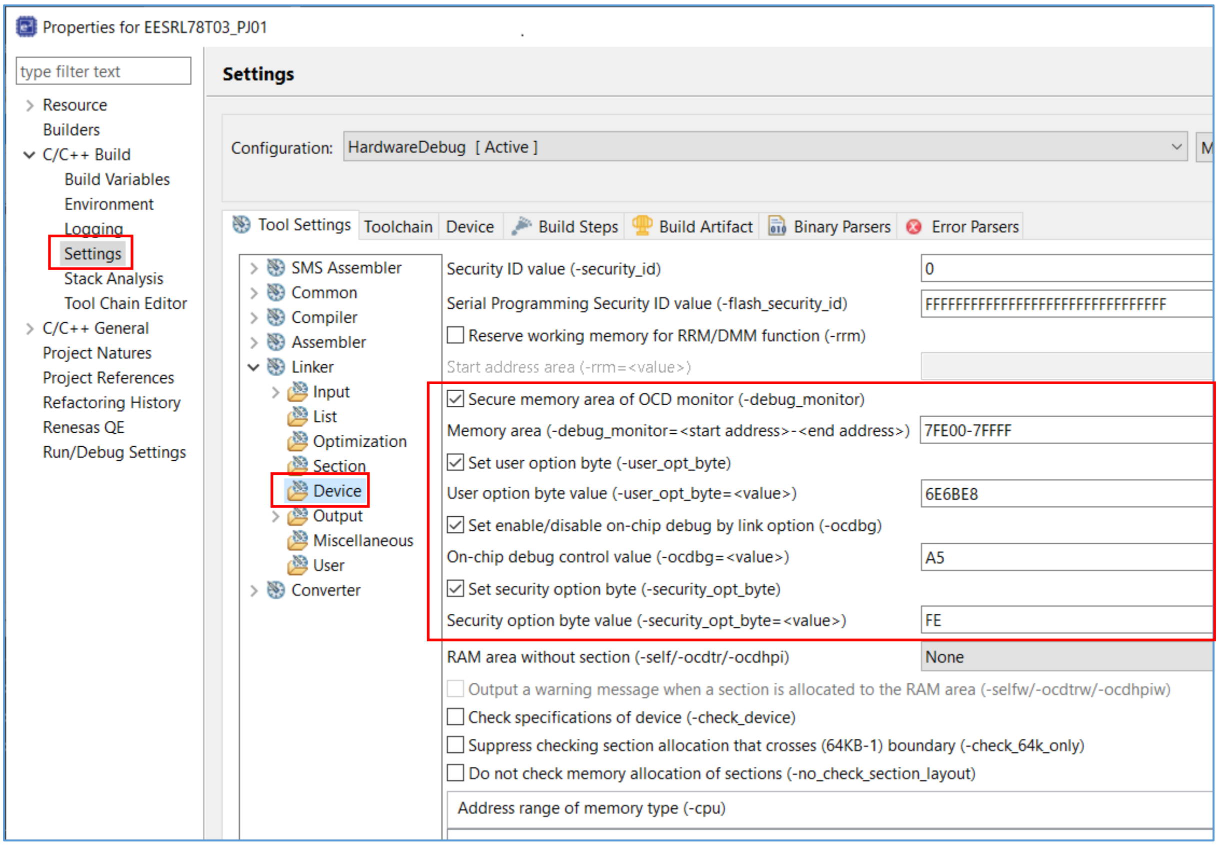
Put a check mark to [Set enable/disable on-chip debug by link option(-ocdbg)].

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/F25]

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.

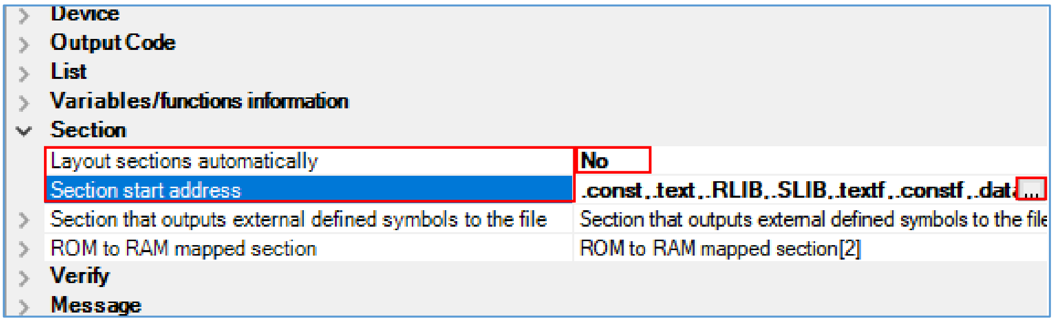


7.1.3.3 Section Item Settings

・Setting of the section Items on CS+ inputs in the “Link Options” tab.

- Setting the [Section] items

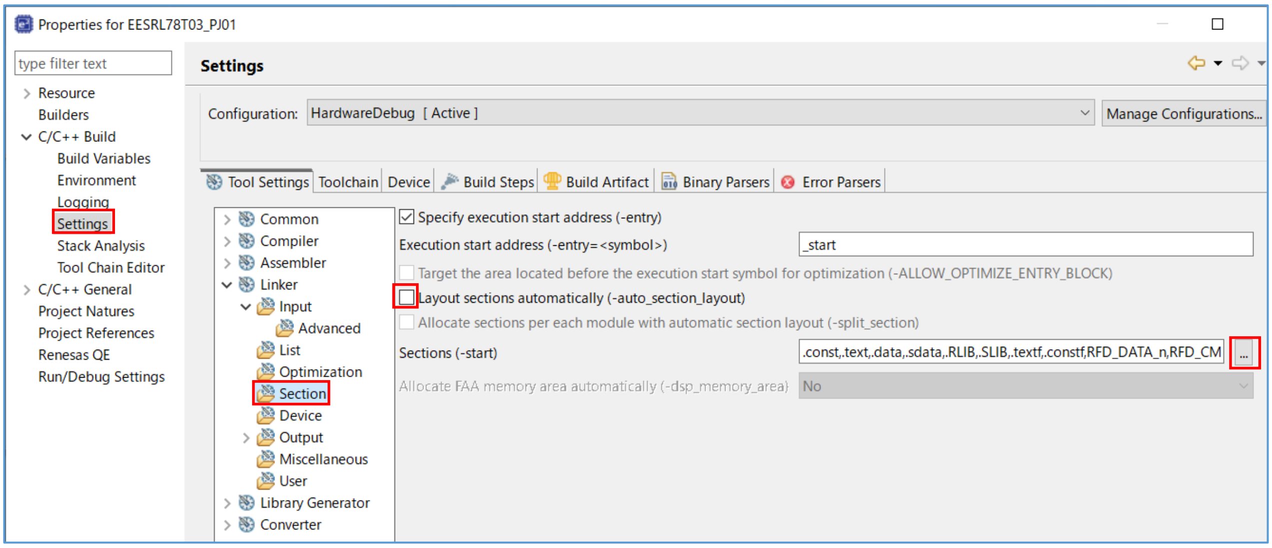
Set “No” to [Layout sections automatically]. And sections come to be displayed on [Section start address]. Press the “ ... ” button of the right-hand side which sections are displaying, and a “Section settings” screen is displayed.



・Setting of the section Items on e² studio inputs in the “Properties” window.

- Select “C/C++ Build” [Settings] - “Linker” [Section]. And set section items on the displayed screen.

Remove a check mark to [Layout sections automatically(-auto_section_layout)]. Press the “ ... ” button of the right-hand side which sections are displaying, and a “Section viewer” screen is displayed.



・Section setting operation for CS+ and e² studio

Set "0x05000" to a top address.
Add the sections defined by "#pragma section" in EES RL78 Type 03 to the program area (code flash memory) and the RAM area. Refer to "Table 2-7 Sections Used in EES" 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 R7F125FPL)
Refer to the user's manual of CC-RL for the section name of each program when a "small model" is selected.

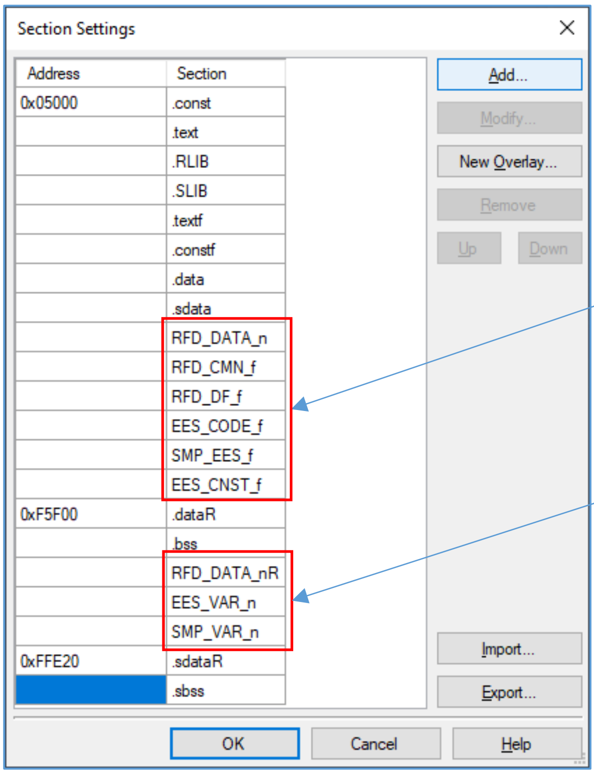
(1) The addition of the sections for EEPROM emulation

・The addition of the sections for EEPROM emulation on CS+

Add sections necessary for code flash memory reprogramming on a “Section Settings” screen. It also includes a section for the RFD RL78 Type 03.

Add to the program area: RFD_DATA_n, RFD_CMN_f, RFD_DF_f, EES_CODE_f, SMP_EES_f, EES_CNST_f

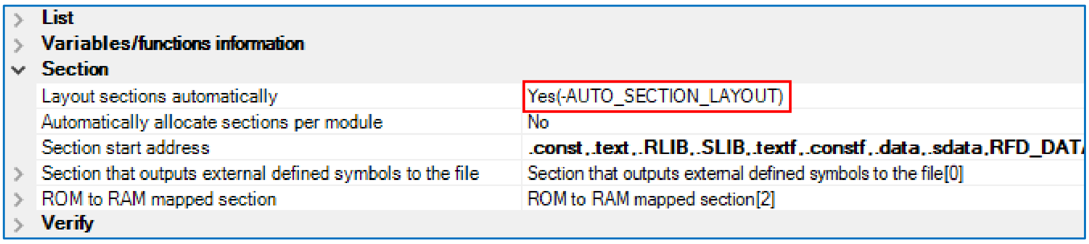
Add to the RAM area: RFD_DATA_nR, EES_VAR_n, SMP_VAR_n



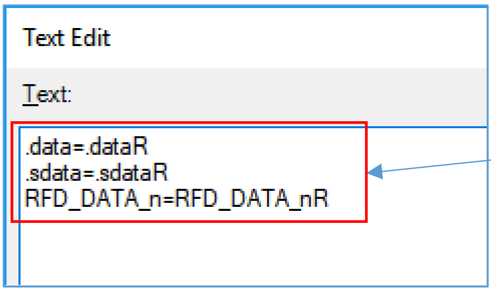
Additional sections
RFD_DATA_n
RFD_CMN_f
RFD_DF_f
EES_CODE_f
SMP_EES_f
EES_CNST_f

RFD_DATA_nR
EES_VAR_n
SMP_VAR_n

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



Press the right-hand side “ ... ” button by [ROM to RAM mapped section], display the “Text Edit” screen, and add the section for copying to RAM from ROM.



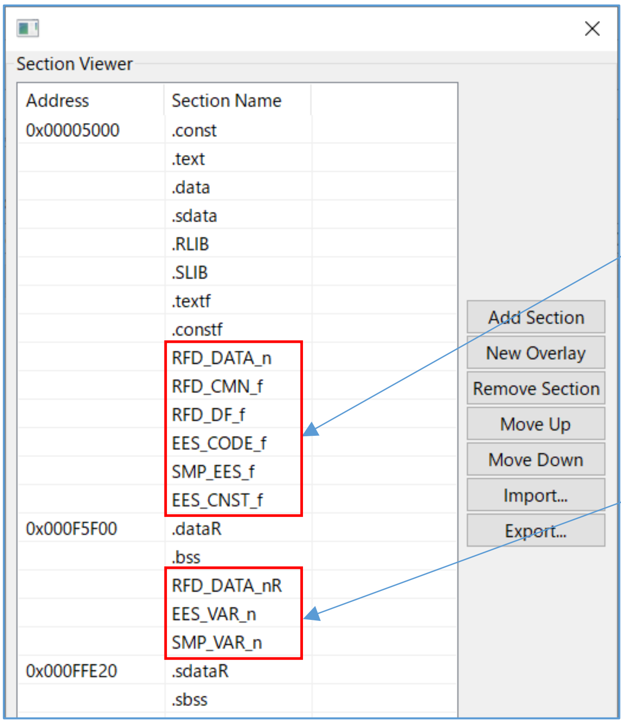
ROM to RAM mapped section (-rom)
.data=.dataR
.sdata=.sdataR
RFD_DATA_n=RFD_DATA_nR

・The addition of the sections for EEPROM emulation on e[2] studio

Add sections necessary for EEPROM emulation on a “Section Viewer”. It also includes a section for the RFD RL78 Type 03.

Add to the program area: RFD_DATA_n, RFD_CMN_f, RFD_DF_f, EES_CODE_f, SMP_EES_f, EES_CNST_f

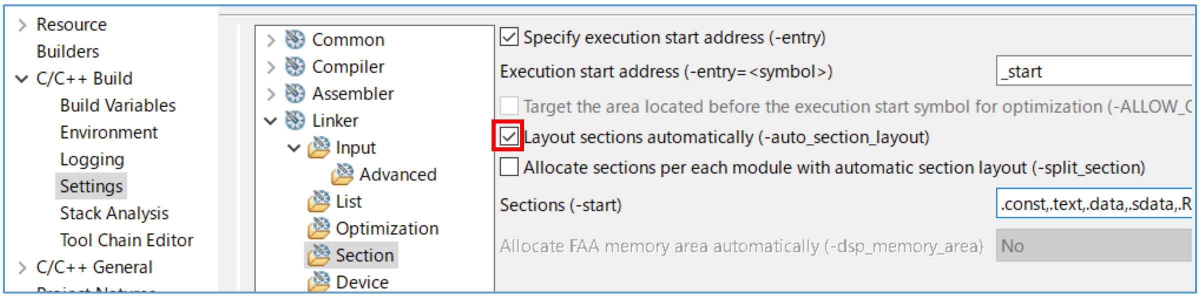
Add to the RAM area: RFD_DATA_nR, EES_VAR_n, SMP_VAR_n



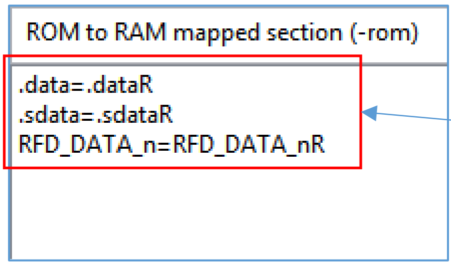
Additional sections
RFD_DATA_n
RFD_CMN_f
RFD_DF_f
EES_CODE_f
SMP_EES_f
EES_CNST_f

RFD_DATA_nR
EES_VAR_n
SMP_VAR_n

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



Select “C/C++ Build” [Settings] - “Linker” [Output], display the “ROM to RAM mapped section (-rom)” screen, and add the section for copying to RAM from ROM.



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

7.1.4 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 (“EESRL78T03_PJ01 HardwareDebug”) of [Renesas GDB Hardware Debugging]. And the displayed “Debugger” tab performs debugging tool setting.

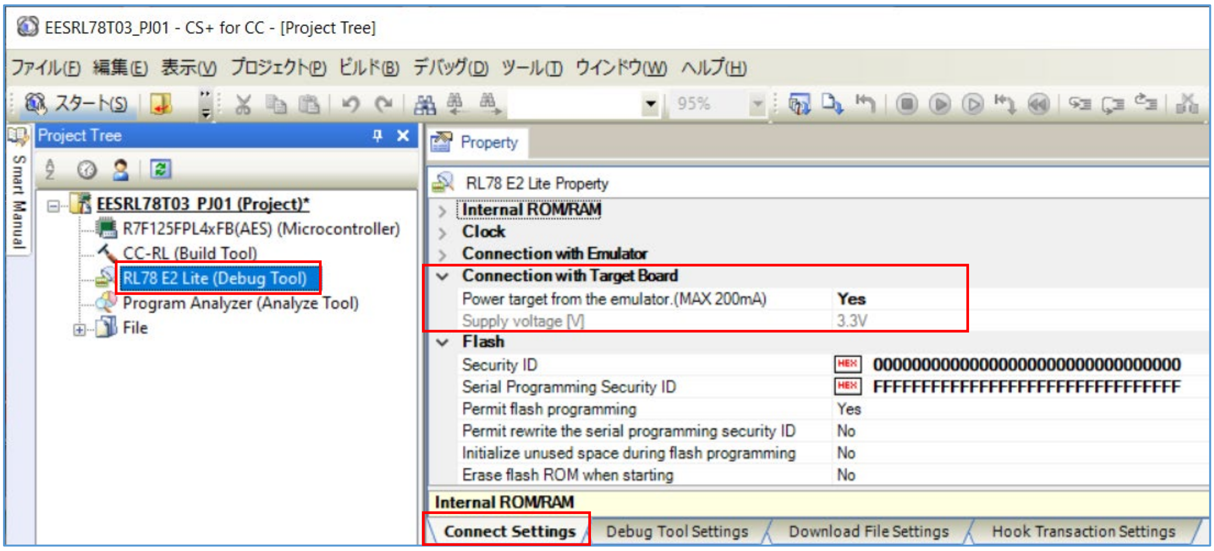
Note: The power is already supplied to the target board, or when power supply capacity is insufficient, the emulator including E2 Lite may be unable to supply power to a target board. Be sure to refer to “the user's manual and Additional Document for User's Manual (Notes on Connection of RL78)” for the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

7.1.4.1 Setting of Connection with Target Board

・On CS+, set up the connection with target board(via E2 Lite) with “Connect Settings” tab.

- [Connection with Target Board] item

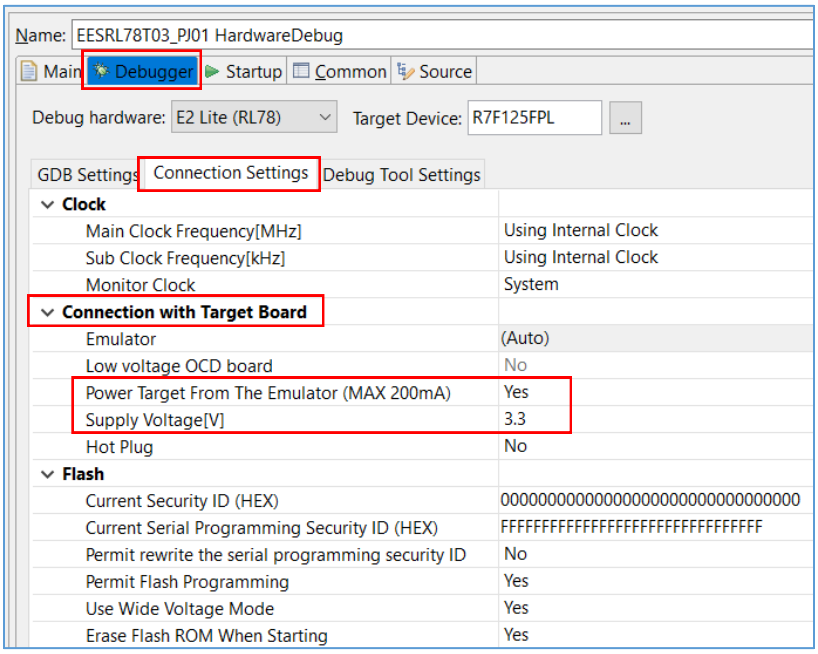
In order to let power supply(Supply voltage: 3.3V) from E2 Lite to a target board, it is necessary to set “Yes” to [Power target from the emulator (MAX 200mA)].



・On e[2] studio, set up the connection with target board(via E2 Lite) with “Connection Settings” tab.

- [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)].



7.2 Creating a Project in the Case of Using IAR Compiler

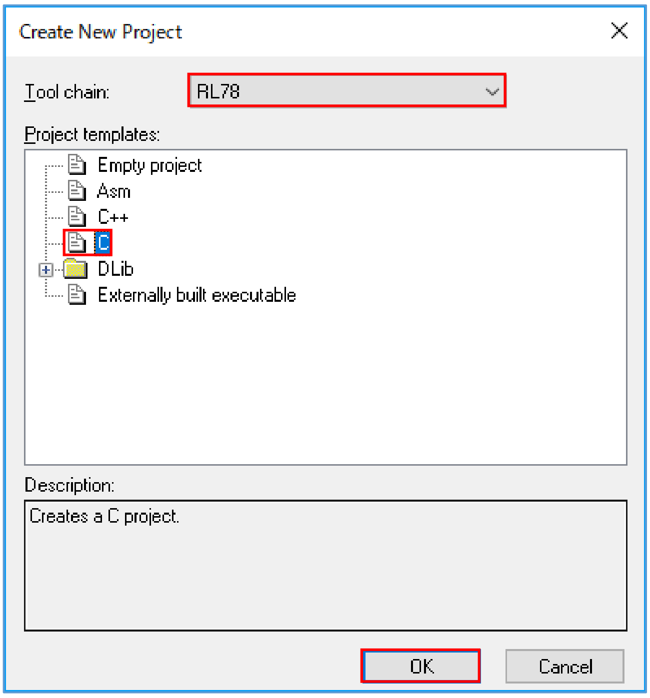
IAR Embedded Workbench can be used for an IAR compiler as an IDE. EES RL78 Type 03 and RFD RL78 Type 03 are registered and built in the project created by the IDE. An example of creating a sample project in case each IDE is used is shown. Because to understand an IAR compiler and each IDE, it is necessary to refer to the user's manual of each tool product.

7.2.1 Example of Creating a Sample Project

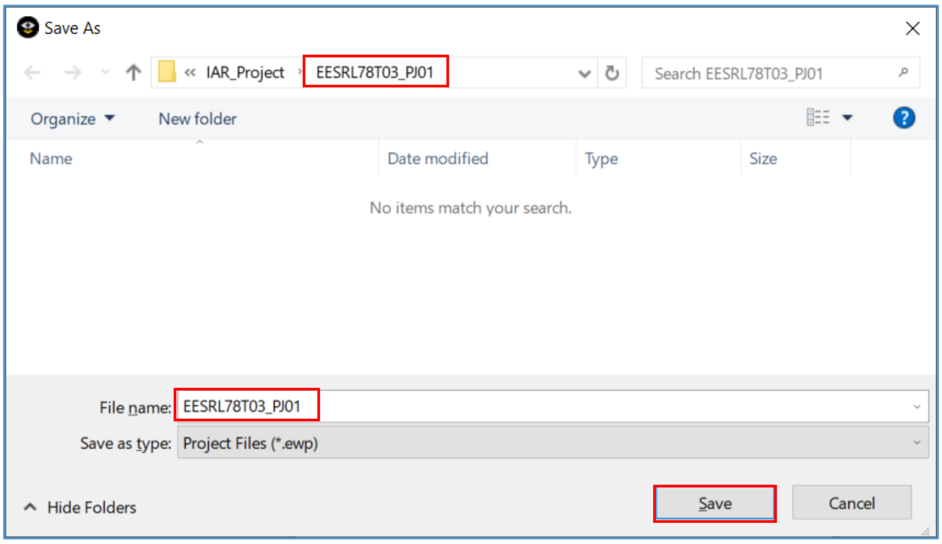
(1) An example of creating a sample project which used IAR Embedded Workbench (IDE)

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

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

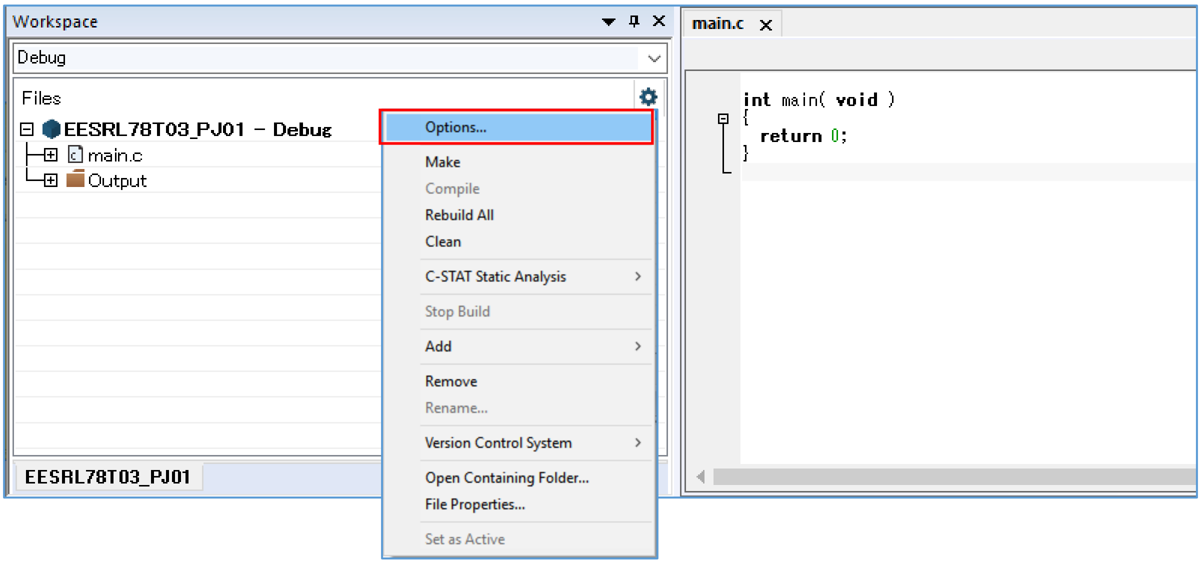


- Create "EESRL78T03_PJ01" folder temporarily, and move into a folder.
- The Project File name is temporarily set to "EESRL78T03_PJ01".



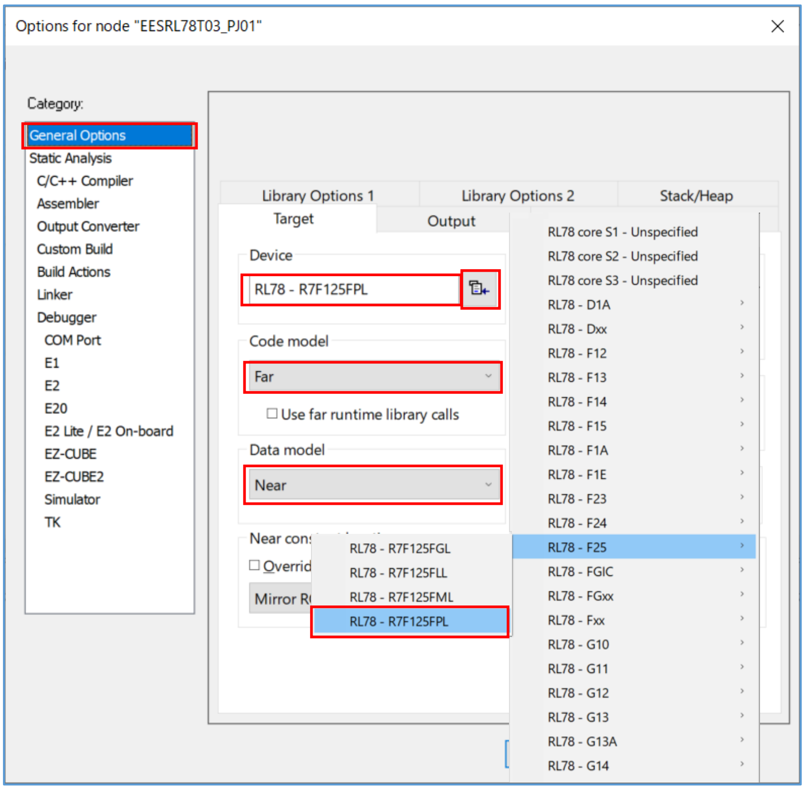
(2) Selection of a target device

・On IAR Embedded Workbench, I click the right mouse button of the project ("EESRL78T03_PJ01 - Debug") in a tree. When an "Options" is selected, the "Options for node [Project name]" window is displayed.



- Input setting in the [General Options] - [Target] tab of "Options for node [Project name]" window.

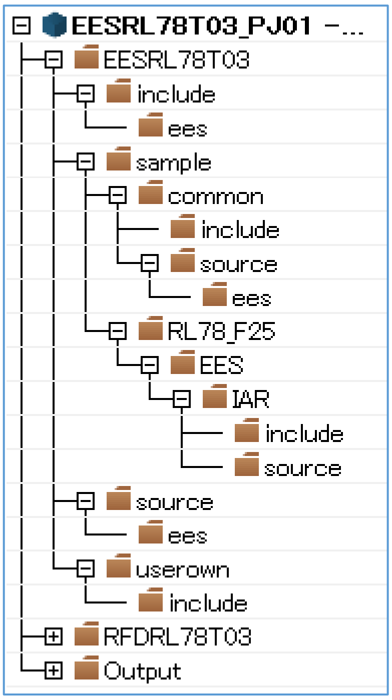
- Press "  " button of [Device]. And select "RL78 - F25" - "RL78 - R7F125FPL". Select "Far" as [Code model] and select "Near" as [Data model].



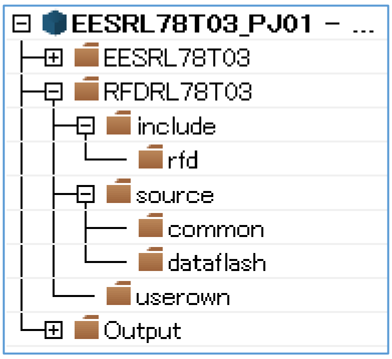
7.2.2 Example of Registration of Target Folders and Target Files

This describes an example of file registration required to execute EEPROM emulation.

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 EES RL78 Type 03 and RFD RL78 Type 03, and files are registered is shown.
The following example shows (1) EES RL78 Type 03 and (2) RFD RL78 Type 03 groups added:



(1) EES RL78 Type 03



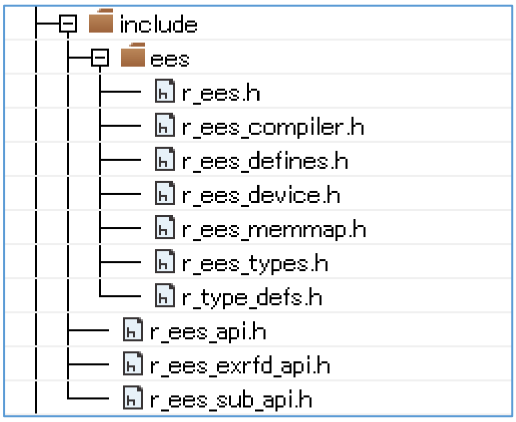
(2) RFD RL78 Type 03

・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 EES RL78 Type 03. 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.

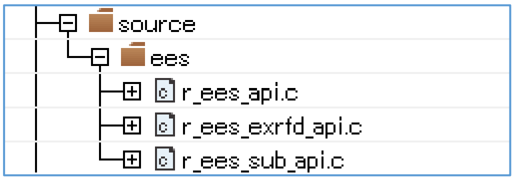
(1) Registration of the EES RL78 Type 03 files.

The groups (“include”, “source”, “userown”, “sample”) and source program files which are included in EES RL78 Type 03 to register are shown below.

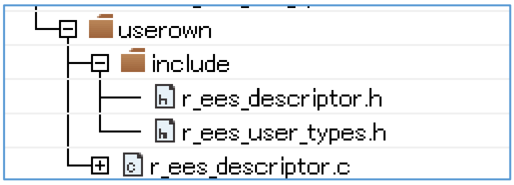
in the “include” group



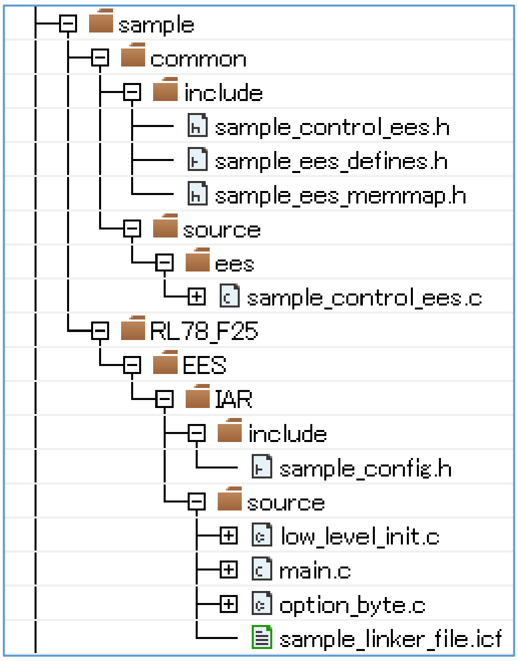
in the “source” group



in the “userown” group



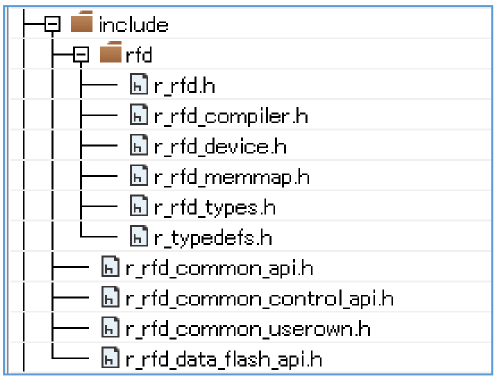
in the “sample” group



(2) Registration of the RFD RL78 Type 03 files

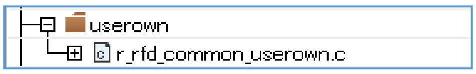
The groups (“include”, “source”, and “userown”) and source program files which are included in RFD RL78 Type 03 to register are shown below.

in the “include” group



in the “source” group

in the “userown” group

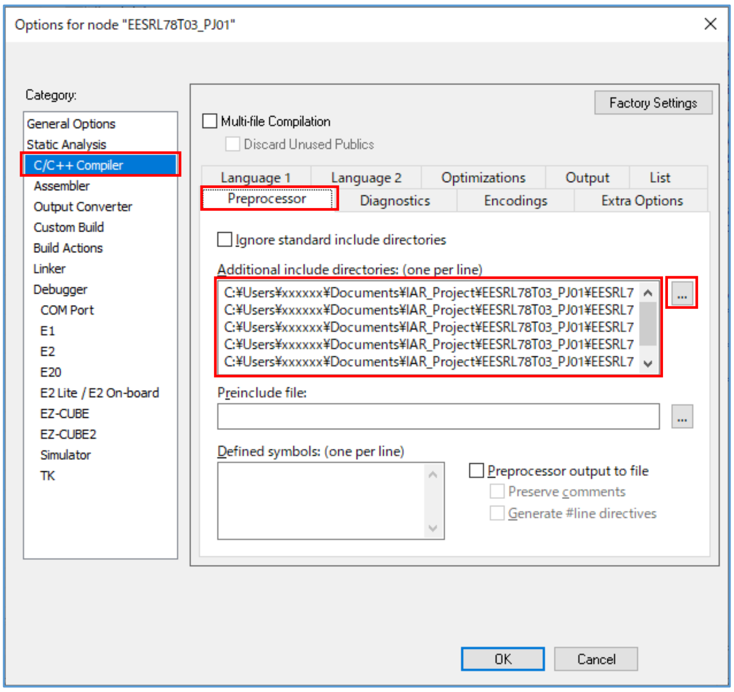


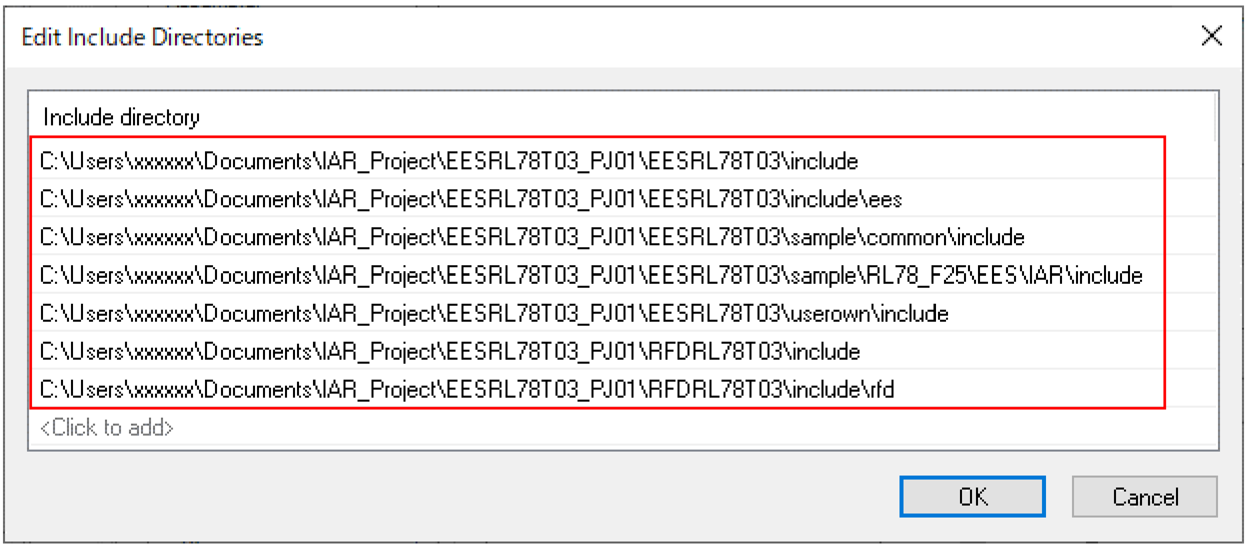
7.2.3 Integrated Development Environment (IDE) Settings

Set IDE setting necessary in order to build EEPROM emulation using an IAR compiler.
IAR Embedded Workbench: Click the right mouse button for the project (“EESRL78T03_PJ01”) in a tree, and select “Options”. And set each setting of the “Category” in the displayed window.

7.2.3.1 Include Path Settings

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





- The example of directory path setting.

It is the example when the project directory is placed in “C:\Users\xxxxxx\Documents\IAR_Project\”.

(1) EES RL78 Type 03 include directories

C:\Users\xxxxxx\Documents\IAR_Project\EESRL78T03_PJ01\EESRL78T03\include

C:\Users\xxxxxx\Documents\IAR_Project\EESRL78T03_PJ01\EESRL78T03\include\ees

C:\Users\xxxxxx\Documents\IAR_Project\EESRL78T03_PJ01\EESRL78T03\sample\common\include

C:\Users\xxxxxx\Documents\IAR_Project\EESRL78T03_PJ01\EESRL78T03\sample\RL78_F25\EES\IAR\include

C:\Users\xxxxxx\Documents\IAR_Project\EESRL78T03_PJ01\EESRL78T03\userown\include

(2) RFD RL78 Type 03 include directories

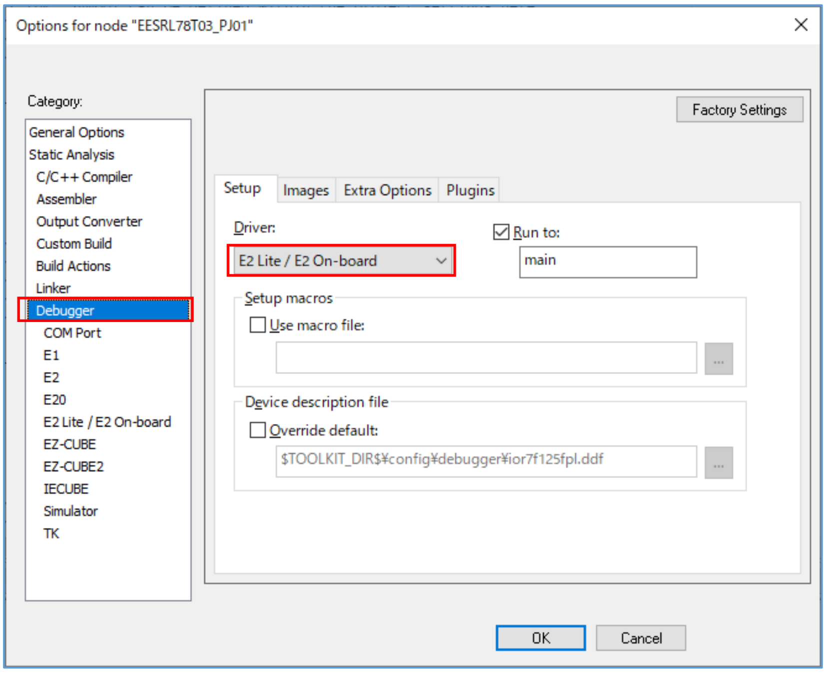
C:\Users\xxxxxx\Documents\IAR_Project\EESRL78T03_PJ01\RFDRL78T03\include

C:\Users\xxxxxx\Documents\IAR_Project\EESRL78T03_PJ01\RFDRL78T03\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.

7.2.3.2 Debugger Settings

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

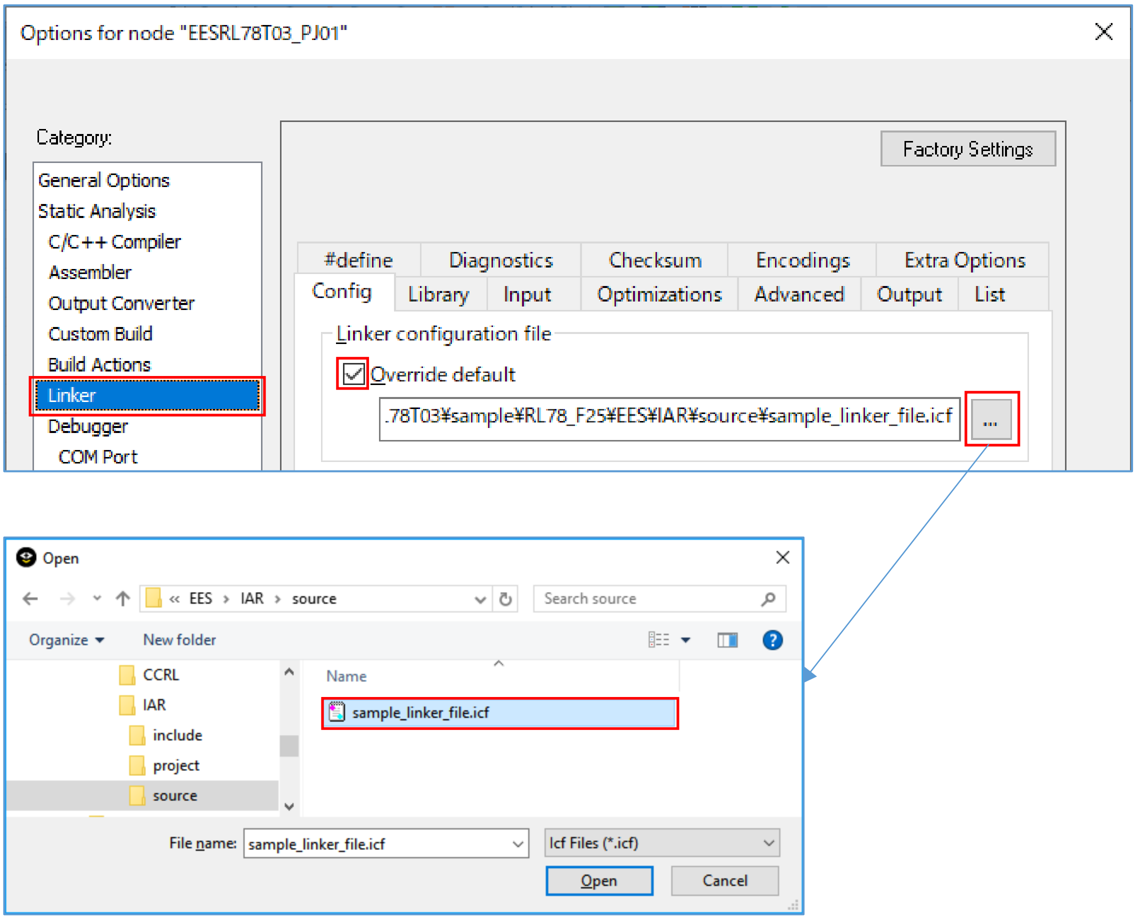


Note: Refer to each reference manual of IAR Embedded Workbench about the other items to be set.

7.2.4 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 button 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.icf” file prepared for EES RL78 Type 03.

- sample_linker_file.icf (\sample\RL78_F25\EES\IAR\source\)



Note: Refer to each reference manual of IAR Embedded Workbench about the descriptive content of Linker configuration file, and the details of the description method.

7.2.4.1 Section Settings

The outline of the section added to Linker configuration file (*. icf) currently prepared by EES RL78 Type 03 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.

(1) The addition of the sections for EES RL78 Type 03.
Add the initial value of each section of EES_CODE, SMP_EES, and EES_CNST to ROM area (ROM_far).

- The additional section of the ROM_far area:
  EES_CODE, SMP_EES, EES_CNST

- The additional section of RAM_near area:
  EES_VAR, SMP_VAR

(2) The addition of the sections for RFD RL78 Type 03.
Add the initial value of each section of RFD_DATA, RFD_CMN, and RFD_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

- The additional section of RAM_near area (Data copied from ROM area):
  RFD_DATA

7.2.4.2 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.icf” file prepared for EES RL78 Type 03. The Option Bytes value for EES RL78 Type 03 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 EES RL78 Type 03 (*. 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 the “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         */
    0x6B,    /* 01101011 */
             /* |||||||| */
             /* +++|++++-- 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 example for RL78/F25]:

The value of User option byte (000C0H-000C2H) in “option_byte.c” file is “**0x6E6BE8**”.
(WDT Stop, LVD reset mode, 40MHz)

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)

The value of security option byte (000C4H/040C4H) in “option_byte.c” file is “**0xFE**”.
(The example of enables read of on-chip debug and flash serial programming security ID.)

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

7.2.5 On-chip Debug Settings

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

7.2.5.1 Example of How to deal with Connection Errors

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

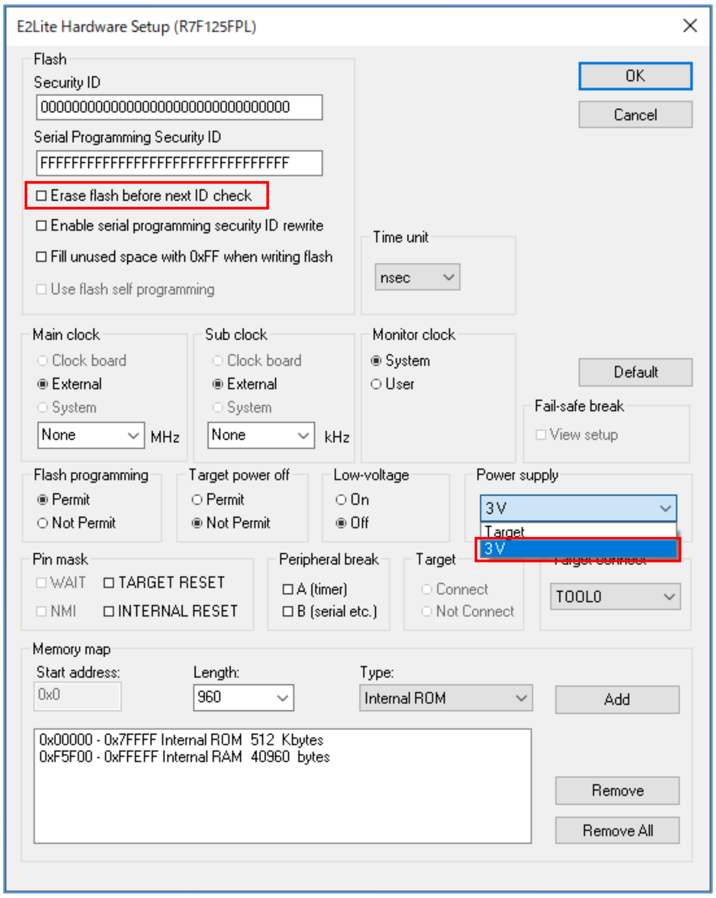
Note: In cases where a target cannot be connected by other causes, please confirm each reference manual from [Help] of IAR Embedded Workbench.

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

- In the case of the ID code mismatch:
  “Cannot verify the ID code.” etc. may be displayed as a message. In this case, put a check mark to “Erase flash before next ID check” of the [ID code] in an “E2 Lite HardwareSetup” window, and continue. And the flash memory is erased and debugger may be connected.

- In the case of power setting error:
  Initial setting of “Power supply” is “Target”. When supplying power supply from E2 Lite, select “3V” by the pull down menu for “Power supply”.

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



7.3 Configurations Modify Procedure for Changing Device

When using a device other than RL78/F25 (R7F125FPL), the address settings in the section and some of the sample programs must be modified. This section describes the where to modify and procedure to modify.

To modify the setting values, refer to MCU List for RL78/F22, F25 shown below and change the setting values according to the device you are using. An example of referencing the MCU List for RL78/F22, F25 and an example of where to modify is shown below.

- MCU List for RL78/F22, F25

MCU Group	Code Flash memory		User RAM		Data Flash memory		Target MCU name
	Size (bytes)	Start/End Address	Size (bytes)	Start/End Address	Size (bytes)	Start/End Address	
RL78/F22	128K	0x00000 - 0x1FFFF	12K	0xFCF00 - 0xFFEFF	8K	0xF1000 - 0xF2FFF	R7F122FxG (x = 7, B, G)
RL78/F25	512K	0x00000 - 0x7FFFF	40K	0xF5F00 - 0xFFEFF	16K	0xF1000 - 0xF4FFF	R7F125FxL (x = G, L, M, P)

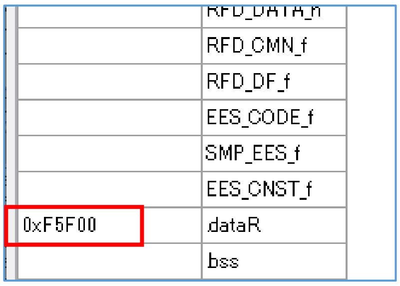
MCU Group	[R-1]	[R-2]	[R-3]	[R-4]	[R-5]	[R-6]	[R-7]	[R-8]	Target MCU name
	RAM Start Address	ROM End Address 1	ROM End Address 2	Data Flash End Address	OCD_ROM	Trace_RAM	Hot plug-in	END_BLOCK	
RL78/F22	0xFCF00	0x0FFFF	0x1FFFF	0xF2FFF	0x1FE00	0xFD300	0xFD500	64	R7F122FxG (x = 7, B, G)
RL78/F25	0xF5F00	0x0FFFF	0x7FFFF	0xF4FFF	0x7FE00	0xF6300	0xF6500	256	R7F125FxL (x = G, L, M, P)

- Example of reference of the MCU List for RL78/F22, F25

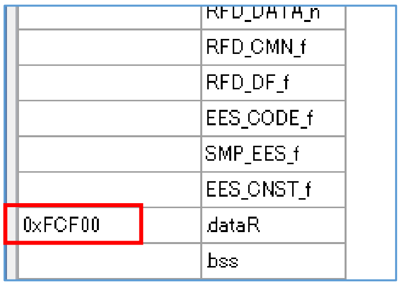
For example, when modifying the setting value indicated by [R-1] (the start address of RAM) as shown in the following figure. Here, refer to the setting value of the start address [R-1] (RAM Start Address) of RAM shown in the MCU List for RL78/F22, F25 and set the value of RL78/F22 (R7F122FGG).

Example of where to modify the start address of RAM: RL78/F25 (R7F125FPL RAM: 40 Kbytes)

[R-1] →



Example of setting the start address value of RAM when using RL78/F22 (R7F122FGG RAM: 12 Kbytes)



The value to be set in [R-1] refers to the MCU List for RL78/F22, F25 and sets the start address value of RAM of the target device.

In the column "Target MCU name" of the MCU List for RL78/F22, F25, search for the row for R7F122FxG. Next, find the cell in the [R-1] column that intersects the row of R7F122FxG.

Since "0xFCF00" applies, the setting value of [R-1] is RL78/F22 (R7F122FxG) value "0xFCF00".

MCU Group	[R-1] RAM Start Address	[R-2] ROM End Address 1	[R-3] ROM End Address 2	[R-4] Data Flash End Address	[R-5] OCD_ROM	[R-6] Trace_RAM	[R-7] Hot plug-in	[R-8] END_BLOCK	Target MCU name
RL78/F22	0xFCF00	0x0FFFF	0x1FFFF	0xF2FFF	0x1FE00	0xFD300	0xFD500	64	R7F122FxG (x = 7, B, G)
RL78/F25	0xF5F00	0x0FFFF	0x7FFFF	0xF4FFF	0x7FE00	0xF6300	0xF6500	256	R7F125FxL (x = G, L, M, P)

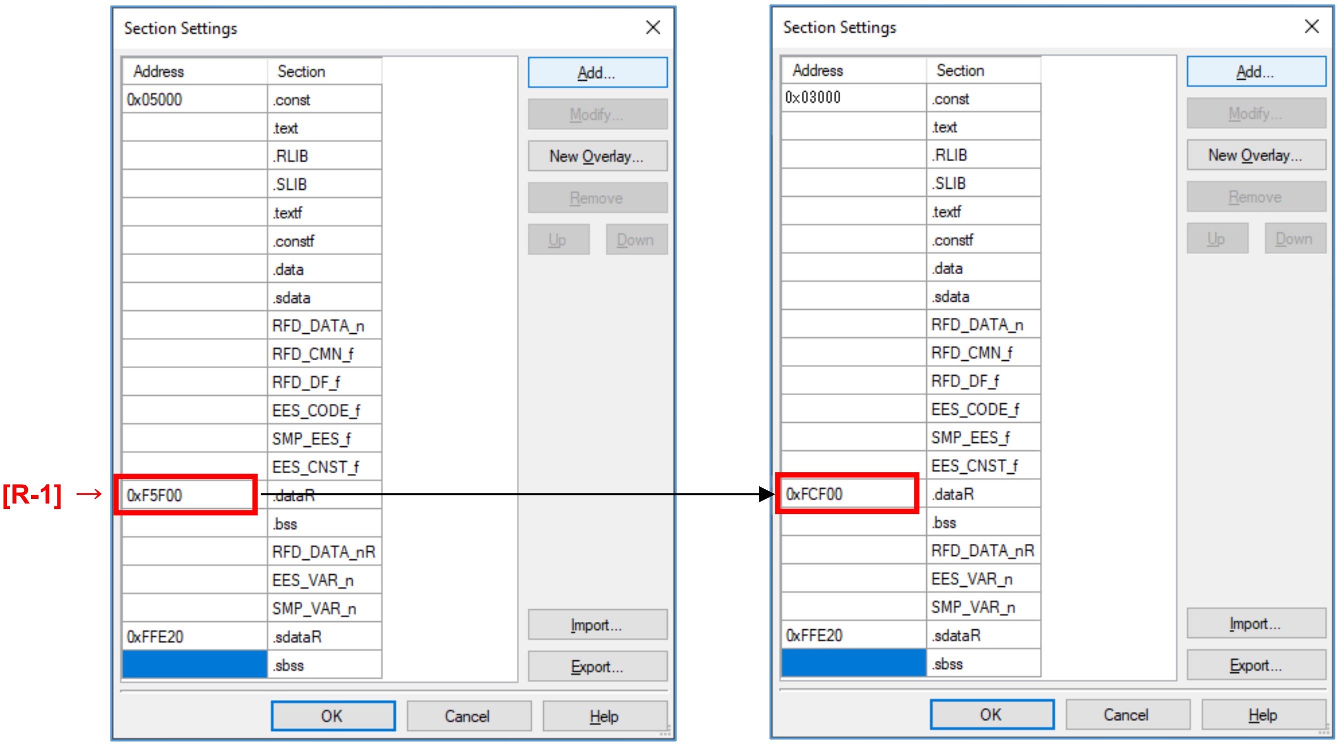
- Example of where to modify

Points that need to be modified from the RL78/F25 (R7F125FPL) settings are listed from "7.3.1 CC-RL Compiler Environment Settings". Points that need to be modified are indicated with "[R-x] →".
Refer to the MCU List for RL78/F22, F25 to find the appropriate [R-x] setting for the device.
Enter the searched value in [R-x]. (x = 1, 2, 3…)

- Example of modification the section setting [CS+: CC-RL compiler]:

Setting for RL78/F25(RAM: 40 Kbytes)
Example: R7F125FPL

Setting for RL78/F22(RAM: 12 Kbytes)
Example: R7F122FGG



7.3.1 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.

7.3.1.1 Section Settings

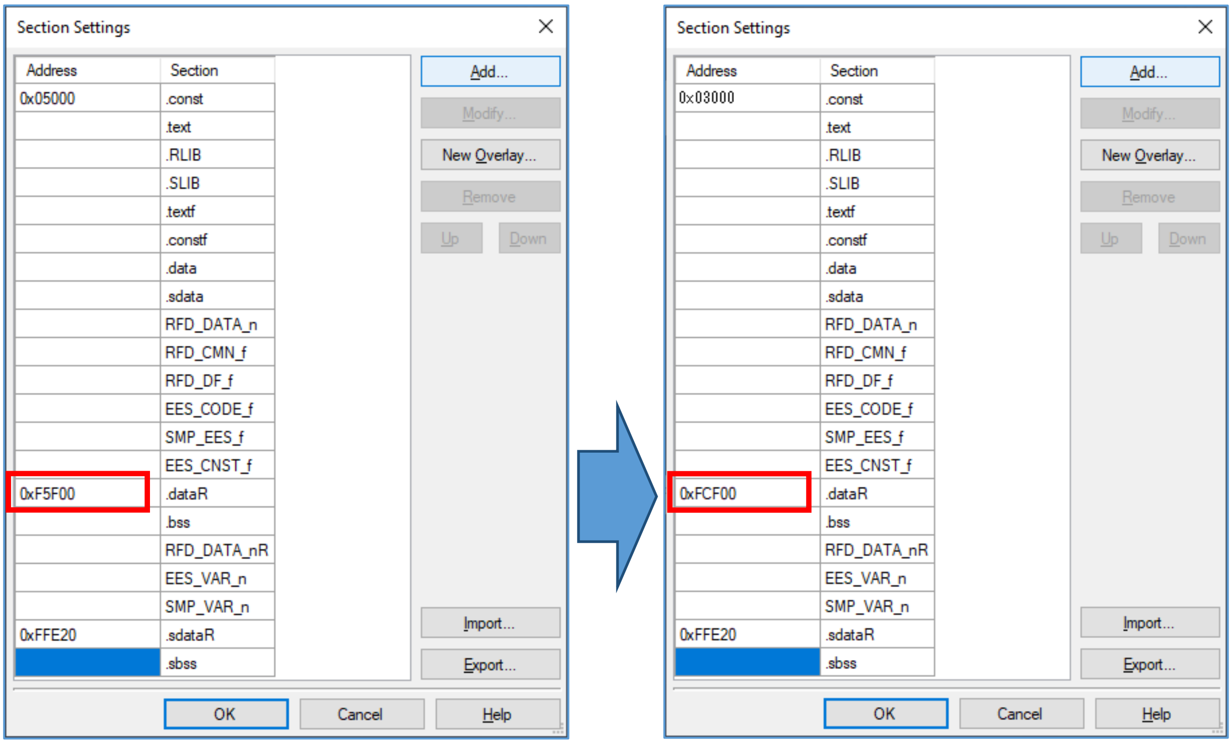
Modify the start address of the RAM area in the section settings.
This example shows the change from RL78/F25 (R7F125FPL) to RL78/F22 (R7F122FGG).
Since the RAM size is changed from 40 Kbytes to 12 Kbytes, modify the start address of RAM from "0xF5F00" to "0xFCF00".
Note: For the start address of the RAM for each product, refer to "R-1" column in the MCU List for RL78/F22, F25.

- Example of modifying section settings (start address of RAM) in CS+:

Setting for RL78/F25(RAM: 40 Kbytes)
Example: R7F125FPL

Setting for RL78/F22(RAM: 12 Kbytes)
Example: R7F122FGG

[R-1] →

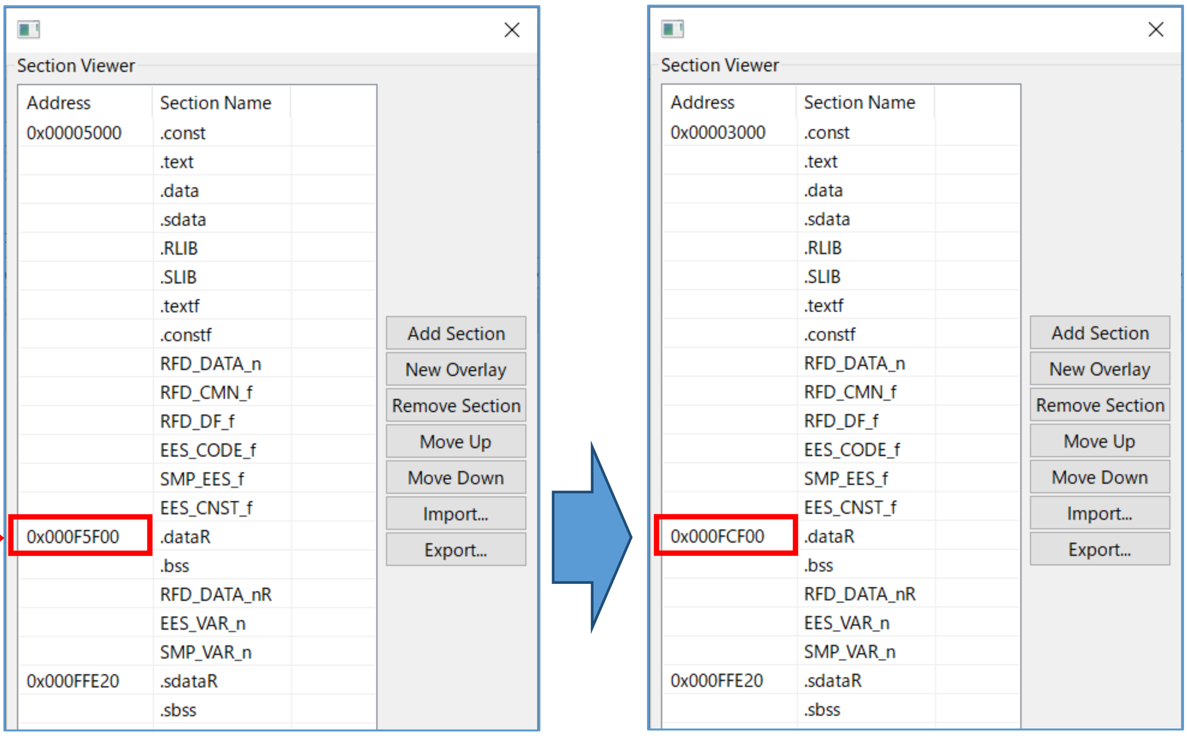


- Example of modifying section settings (start address of RAM) in e^2 studio:

Setting for RL78/F25(RAM: 40 Kbytes)
Example: R7F125FPL

Setting for RL78/F22(RAM: 12 Kbytes)
Example: R7F122FGG

[R-1] →



7.3.1.2 Debug Settings

When using the RL78/F22, the debug monitor area has a different range when using the debugger.

- The start of the "Debug monitor area" address sets the address obtained by subtracting "511 bytes (0x1FF)" from the end address of the ROM area. If the end address is "0x7FFFF", set it to "0x7FE00".

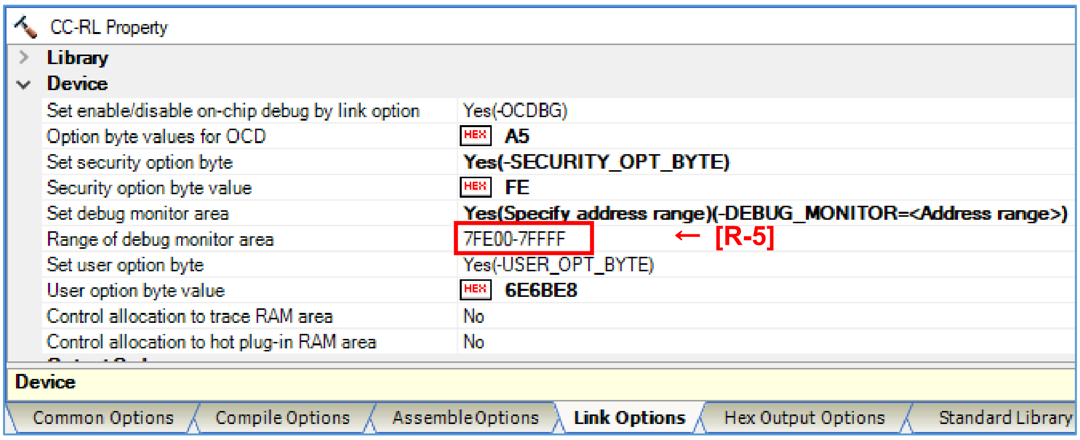
This example shows the modify from RL78/F25 (R7F125FPL) to RL78/F22 (R7F122FGG).

- Set the debug monitor area range to "0x1FE00 - 0x1FFFF" for the RL78/F22 (R7F122FGG).

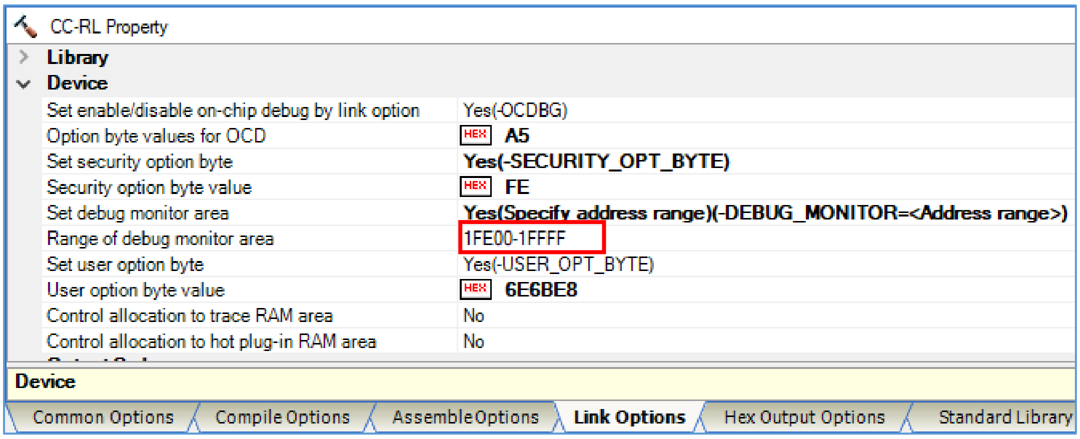
Note: For the start of the "Debug monitor area" address for each product, refer to "R-5" column in the MCU List for RL78/F22, F25.

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

Setting for RL78/F25 (ROM: 512 Kbytes) Example: R7F125FPL

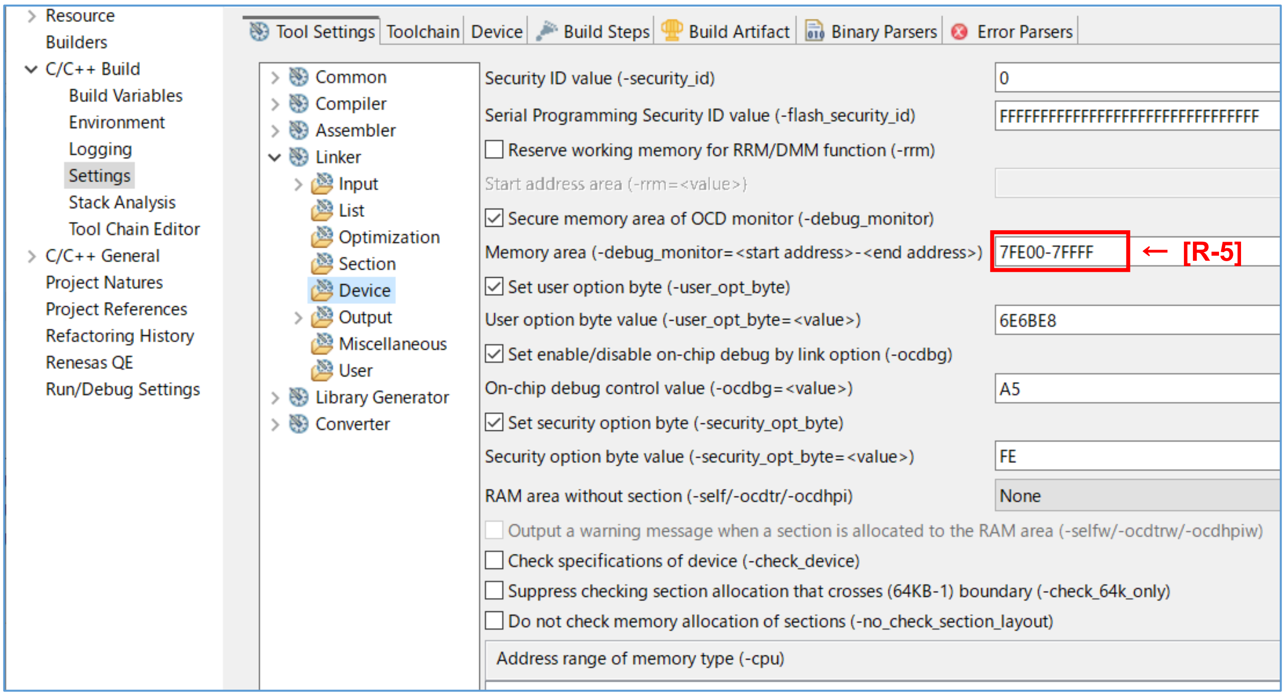


Setting for RL78/F22 (ROM: 128 Kbytes) Example: R7F122FGG

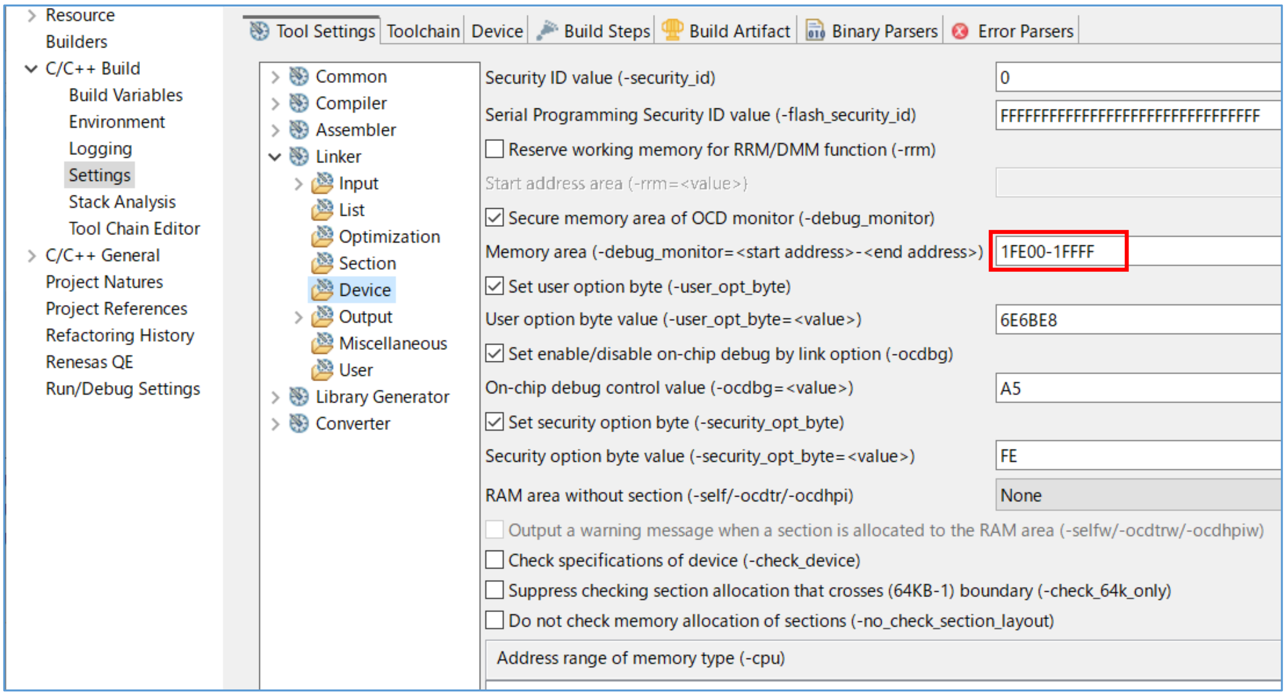


- To set the debug monitor area in e^2 studio, select the [Device] in the “Linker”.

Setting for RL78/F25 (ROM: 512 Kbytes) Example: R7F125FPL



Setting for RL78/F22 (ROM: 128 Kbytes) Example: R7F122FGG



7.3.2 IAR Compiler Environment Settings

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

7.3.2.1 Setting Up Header Files for Target Device

The “main.c” and “low_level_init.c” provided with EES RL78 Type 03 includes the header files for the target device “RL78/F25: R7F125FPL”. When using other RL78/F25 products or RL78/F22 products, the included header file must be changed to the header file for the device used.
This section describes when RL78/F22 (R7F122FGG) is used.

Target files name: main.c, low_level_init.c

- For RL78/F25 (R7F125FPL):
  - < main.c >
    - #include “ior7f125fpl.h”
  - < low_level_init.c >
    - #include “ior7f125fpl.h”
    - #include “ior7f125fpl_ext.h”

- Example for RL78/F22 (R7F122FGG):
  - < main.c >
    - #include “ior7f122fgg.h”
  - < low_level_init.c >
    - #include “ior7f122fgg.h”
    - #include “ior7f122fgg_ext.h”

Note: For the device type name of the product, refer to “Target MCU name” column in the MCU List for RL78/F22, F25.

7.3.2.2 Linker Configuration File Settings

In the sample program provided by EES RL78 Type 03, The sections (ROM, RAM, and Data flash range) for RL78/F25 (R7F125FPL) are set.

When using other RL78/F25 products or RL78/F22 products, modify the contents of the sample linker file “sample_linker_file.icf” provided for the RL78/F25 of EES RL78 Type 03 because the section settings are different. The modifications are shown in red text below, so refer to the MCU List for RL78/F22, F25 and change the setting values for the target device.

Target file name: sample_linker_file.icf

This example shows the modify from RL78/F25 (R7F125FPL) to RL78/F22 (R7F122FGG).

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

(1) **Section Settings**

- Modifies to the size of ROM, RAM, and Data Flash

Setting for RL78/F25 (ROM: 512 Kbytes, RAM: 40 Kbytes, Data Flash: 16 Kbytes) Example: R7F125FPL

```
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]
                        | mem:[from 0x40000 to 0x4FFFF] | mem:[from 0x50000 to 0x5FFFF]
                        | mem:[from 0x60000 to 0x6FFFF] | mem:[from 0x70000 to 0x7FFFF]; ← [R-2], [R-3] Note 1
define region ROM_huge = mem:[from 0x00132 to 0x7FFFF];   ← [R-2] or [R-3] Note 2
define region SADDR      = mem:[from 0xFFE20 to 0xFFEDF];
define region RAM_near = mem:[from 0xF5F00 to 0xFFE1F];   ← [R-1]
define region RAM_far   = mem:[from 0xF5F00 to 0xFFE1F];   ← [R-1]
define region RAM_huge = mem:[from 0xF5F00 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/F22 (ROM: 128 Kbytes, RAM: 12 Kbytes, Data Flash: 8 Kbytes) Example: R7F122FGG

```
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 “0x7FFFF”, set to “0x7FE00”.

- 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 “0xF5F00”, set to “0xF6300”.

- 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 “0xF5F00”, set to “0xF6500”.

- When debugging self-programming with an on-chip debugger, 128 bytes of area is used from the start address of RAM. Therefore, it is necessary to set the start address of a RAM area, and the address adding “127 bytes (0x7F) ”. If the start address is “0xF5F00”, set “0xF5F00” and “0xF5F7F”.

As an example, modifying from RL78/F25 (R7F125FPL) to RL78/F22 (R7F122FGG) 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].
- Set the area range required to debug the self-programming to [from 0xFCF00 to 0xFCF7F].

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

Setting for RL78/F25 (ROM: 512 Kbytes, RAM: 40 Kbytes) Example: R7F125FPL

```
if (isdefinedsymbol(__RESERVE_OCD_ROM))
{
        if (__RESERVE_OCD_ROM == 1)
        {
        reserve region “OCD ROM area” = mem:[from 0x7FE00 size 0x0200]; ← [R-5]
        }
}
      |
[Omitted]
      |
if (isdefinedsymbol(__RESERVE_OCD_TRACE_RAM))
{
        if (__RESERVE_OCD_TRACE_RAM == 1)
        {
        reserve region “OCD Trace RAM” = mem:[from 0xF6300 size 0x0200]; ← [R-6]
        }
}
      |
[Omitted]
      |
if (isdefinedsymbol(__RESERVE_HOTPLUGIN_RAM))
{
   if (__RESERVE_HOTPLUGIN_RAM == 1)
   {
      reserve region “Hot Plugin RAM” = mem:[from 0xF6500 size 0x0030];  ← [R-7]
   }
}
      |
[Omitted]
      |
if (isdefinedsymbol(__RESERVE_FLASH_SELF_PROGRAMMING_RAM))
{
   if (__RESERVE_FLASH_SELF_PROGRAMMING_RAM == 1)
   {
      reserve region “RESERVED_FLASH_SELF_PROGRAMMING_RAM” = mem:[from 0xF5F00 to 0xF5F7F];
   }                                                               ↑ [R-1]
}
```

Setting for RL78/F22 (ROM: 128 Kbytes, RAM: 12 Kbytes) Example: R7F122FGG

```
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];
   }
}
      |
[Omitted]
      |
if (isdefinedsymbol(__RESERVE_FLASH_SELF_PROGRAMMING_RAM))
{
   if (__RESERVE_FLASH_SELF_PROGRAMMING_RAM == 1)
   {
      reserve region "RESERVED_FLASH_SELF_PROGRAMMING_RAM" = mem:[from 0xFCF00 to 0xFCF7F];
   }
}
```

(3) RAM Start Address Settings

Sets the starting address of RAM area.

"sample_linker_file.icf" is set to use a RAM size of 40 Kbytes on RL78/F25 (R7F125FPL). Therefore, it is necessary to modify the setting when using a RAM size other than 40 Kbytes.

This section describes an example of modifying the settings when using RL78/F22 (R7F122FGG) with a RAM size of 12 Kbytes.
To use RL78/F22 (R7F122FGG) with a RAM size of 12 Kbytes, modify the value of the RAM start address setting register (RAMSAR) from "0x5F" 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/F25(RAM: 40 Kbytes) Example: R7F125FPL

```
define exported symbol _RAMSAR_ADDR = 0xF0076;
if (!isdefinedsymbol(__RAMSAR_VAL))
{
  define exported symbol _RAMSAR_VAL = 0x5F;
}
else
{
  define exported symbol _RAMSAR_VAL = __RAMSAR_VAL;
}
keep symbol __setup_ramsar;
```

↓

Setting for RL78/F22(RAM: 12 Kbytes) Example: R7F122FGG

```
define exported symbol _RAMSAR_ADDR = 0xF0076;
if (!isdefinedsymbol(__RAMSAR_VAL))
{
  define exported symbol _RAMSAR_VAL = 0xCF;
}
else
{
  define exported symbol _RAMSAR_VAL = __RAMSAR_VAL;
}
keep symbol __setup_ramsar;
```

8. Revision History

8.1 Major Modifications in this Revision

Rev.	Date	Description	
		Page	Summary
1.00	Aug.05.24	−	Newly created.
1.01	May.30.25	-	Add support of RL78/F22.

EEPROM Emulation Software RL78 Type 03 User's Manual

Publication Date: Rev.1.01 May. 30. 25

Published by: Renesas Electronics Corporation

EEPROM Emulation Software

RL78 Type 03

Renesas Electronics Corporation

R01UT5477EJ0101