User's Manual

16

RL78 Family

EEPROM Emulation Library Pack02

Japanese Release

Installer name: RENESAS_RL78_EEL-FDL_T02_PACK02_xVxx.exe

16-Bit Single-Chip Microcontroller

Renesas Electronics
www.renesas.com

Rev.1.10 Apr 2024

Notice

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

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

www.renesas.com

Contact information

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

Trademarks

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

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

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

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

How to Use This Manual

Target Readers

This manual is intended for users who wish to understand the features of the RL78 family microcontrollers EEPROM Emulation Library Pack 02 and to use the library in designing and developing application systems.

<R>

Refer to the following list for the target MCUs.

"Self-Programming Library (Japanese Release) and Supported MCUs" (R20UT2861XJxxxx)

"RL78 Family Self RAM list of Flash Self Programming Library" (R20UT2944)

Purpose

This manual is intended to give users understanding of how to use EEPROM Emulation Library Pack 02 to rewrite the flash data memory in RL78 family microcontrollers (i.e. write constant data by the application).

Organization

The RL78 Family EEPROM Emulation Library Pack 02 user's manual is separated into the following parts:

- Overview of EEPROM Emulation
- Using EEPORM Emulation
- EEPROM Emulation Function

How to Read This Manual

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

- To gain a general understanding of features
  -> Read this manual in order of the table of contents.
- For details on the functions of the library
  -> Refer to chapter 5, User Interface, of this user's manual.

Conventions

Data significance: Higher-order digits to the left and lower-order digits to the right
Active low representations: $\overline{xxx}$ (overscore over pin and signal name)
Note: Footnote for item marked with **Note** in the text.
Caution: Information requiring particular attention
Remark: Supplementary information
Numeral representation: Binary ... xxxx or xxxxB
Decimal ... xxxx
Hexadecimal ... xxxxH or 0xXXXX

Contents

Chapter 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. During EEPROM emulation, the data flash library and EEPROM emulation library are used, and the data flash memory is written to and read from.

The data flash library is a software library used to perform operations on the data flash memory. The EEPROM emulation library is a software library used to execute EEPROM emulation from a user-created program. The data flash library and EEPROM emulation library are placed in the code flash memory for use.

The EEPROM emulation library is free software to rewrite the data flash through the user program.

In this user's manual, processing of the EEPROM emulation library includes processing of the data flash library.

Be sure to use this user's manual together with the release note supplied with the package of this EEPROM emulation library and the user's manual for the target device.

<R> 1. 2 Target Compilers

The Renesas CA78K0R, CC-RL and LLVM compilers are the target compilers for the product covered by this user's manual. The CA78K0R, CC-RL and LLVM compilers are respectively abbreviated as CA78, CCRL and LLVM.

1. 3 Definition of Terms

The terms used in this manual are defined below.

- **Pack**

"Pack" is an identification name representing an EEPROM emulation library type. Use the pack corresponding to your device.

- **EEL**

An abbreviation of the EEPROM emulation library.

In this user's manual, the RL78 EEPROM emulation library Pack02 is hereafter referred to as EEL.

- **FDL**

An abbreviation of the data flash library.

- **FSL**

An abbreviation of the flash self programming library.

- **EEL function**

  A generic term for the functions offered by the EEL.

- **FDL function**

  A generic term for the functions offered by the FDL.

- **FSL function**

  A generic term for the functions offered by the FSL.

- **Block number**

  A number which identifies a block of flash memory.

- **EEL Blocks**

  An abbreviation of blocks that the EEPROM emulation library accesses. In this user's manual, EEPROM emulation blocks are hereafter referred to as EEL blocks.

- **CF**

  Code flash memory

- **DF**

  Data flash memory

Chapter 2 EEPROM Emulation

2. 1 Specifications of EEPROM Emulation

By calling the EEL function provided by the EEL from a user-created program, use is possible without the awareness of data flash memory operations.

For the EEL, an one-bye identifier (data ID: 1 to 64) 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 EEL can handle up to 64 identifiers.)

Note that three or more continuous block areas of data flash memory (recommended) Note are used to store the data. These blocks are called EEL 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 EEL 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.

2. 2 Outline of Functions

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

- Up to 64 data items settable
- A data size of 1 to 255 bytes settable
- Supporting the back ground operation (BGO)
- Consumption of memory for management data
  (10 bytes per EEL block and 2 bytes per EEL block write data)
- Restoration by EEL_CMD_REFRESH when execution is stopped by a CPU reset while EEL_CMD_WRITE or EEL_CMD_REFRESH is running
- Block rotation (averaging data flash use frequency)

<R> Table 2-1 shows the range of settings when the EEL functions are used.

<R>

Table 2-1 Range of Settings when the EEL Functions are Used

Item	Range
User data length	1 to 255
Amount of stored user data Note 1	1 to 64
Data ID range	1 to 64 (The numbers assigned are from 1 to 64 in the order of registration, and the selection of settings is not possible.)
Number of EEPROM emulation blocks Note 2	3 to 255
Recommended user data size Note 1	1014/2 bytes

Note 1: The total size of user data must be within 1/2 of each block when all user data are written to an EEL 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 "4.2 Number of Stored User data items and total user data size".

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

2. 3 EEL Architecture

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

2. 3. 1 System Structure

The EEL offers interface for accessing the data flash area defined by the user. The arrows shown in the Figure 2-1 below indicate the flow of processing.

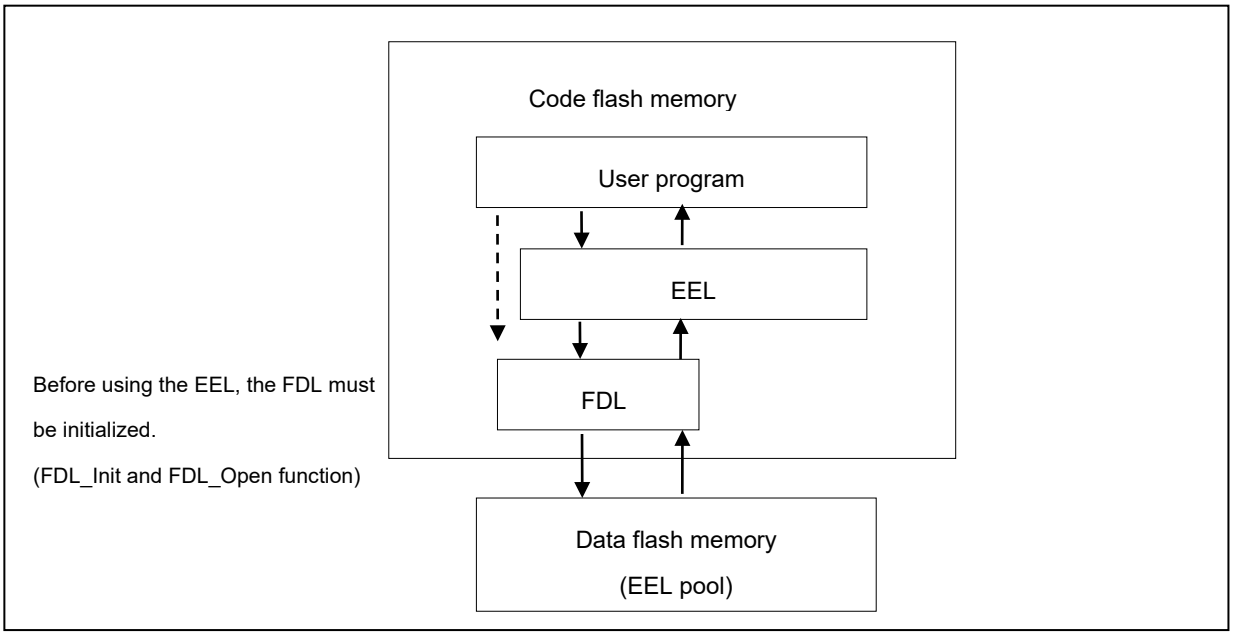


Figure 2-1 System Structure

2. 3. 2 EEL Pool

The EEL pool is a user-defined data flash area that is accessible by the EEL. The user program can access the data flash only by using this EEL pool in the data flash via the EEL. The EEL pool size must be specified with the number of blocks in the data flash of the target device. For the procedure to specify the number of blocks, see section "4.3 Initial Values to be Set by User".

The EEL pool is divided into 1024-byte blocks. Each block has a state which indicates the current usage of the block.

<R>

Table 2-2 States of the EEL Blocks

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

Figure 2-2 shows an example of pool configuration for a device with 8-KB data flash memory.

When no writable area is remaining in the active block (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 EEL_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.

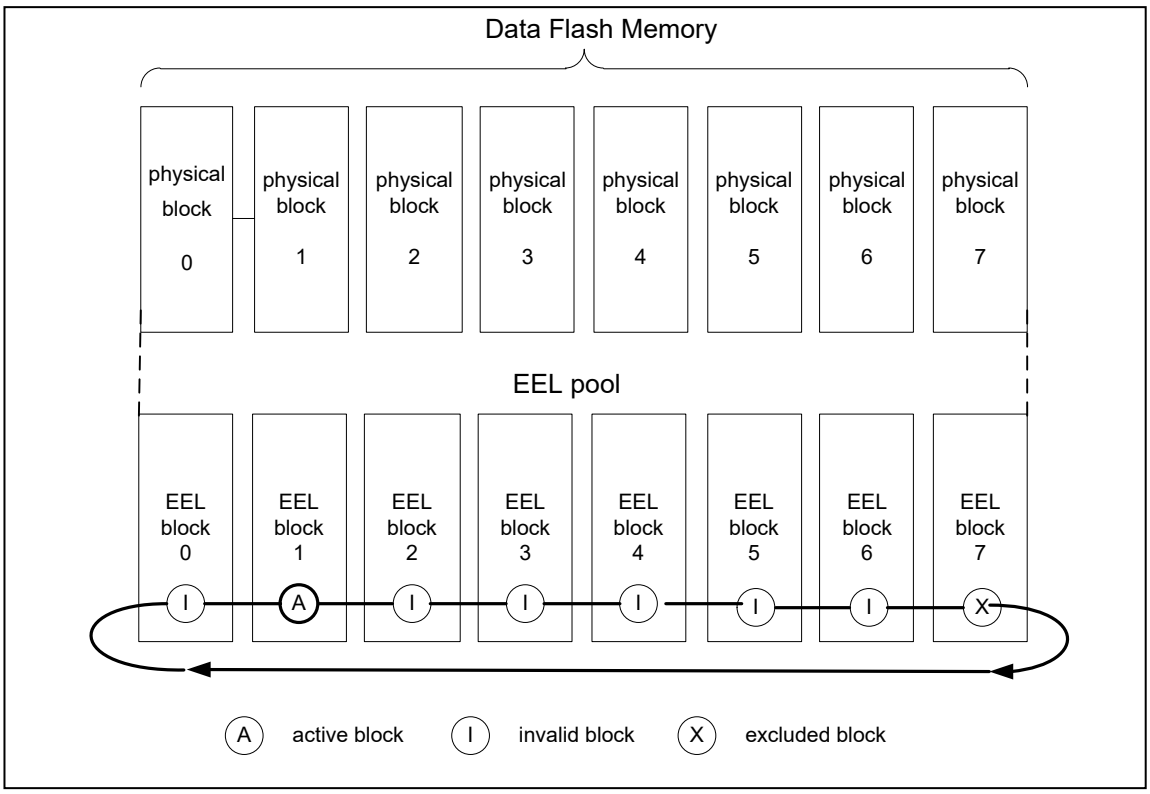


Figure 2-2 EEL pool structure

The overall life cycle of a block in the EEL pool is shown in Figure 2-3. During normal operation, the block switches between active and invalid state. When an error occurs during an access to the EEL block, the error EEL 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.

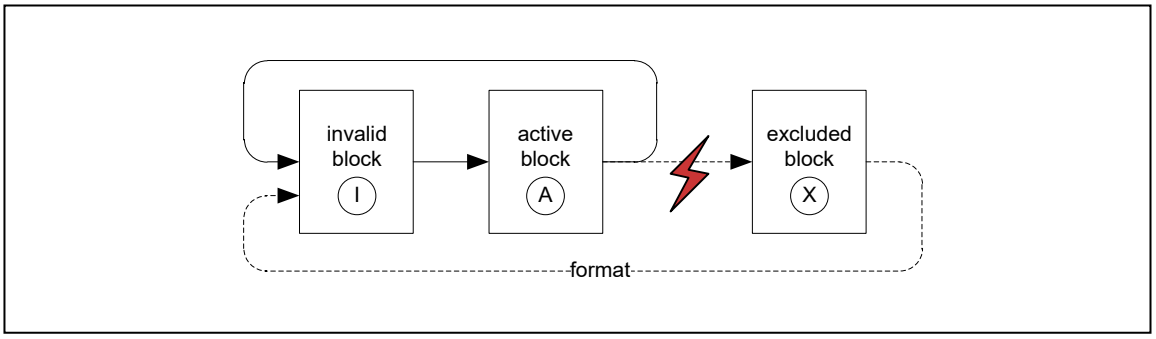


Figure 2-3 Life cycle of an EEL block

The EEL pool has the four states shown below.

Table 2-3 States of the EEL Pool

State	Description
Pool operational	This is the usual case during EEL 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 EEL block is left. (At least two blocks that are not excluded are necessary for EEL operations.)
Pool inconsistent	There is a mismatch in the pool state and the data structure in the EEL block does not match the user-set data structure. The EEL block is in the undefined state (e.g. no active block is present).

2. 3. 3 Structure of EEL Block

The detailed block structure used by the EEL is shown in Figure 2-4. In general, an EEL block is divided into three utilized areas: the block header, the reference area, and the data area.

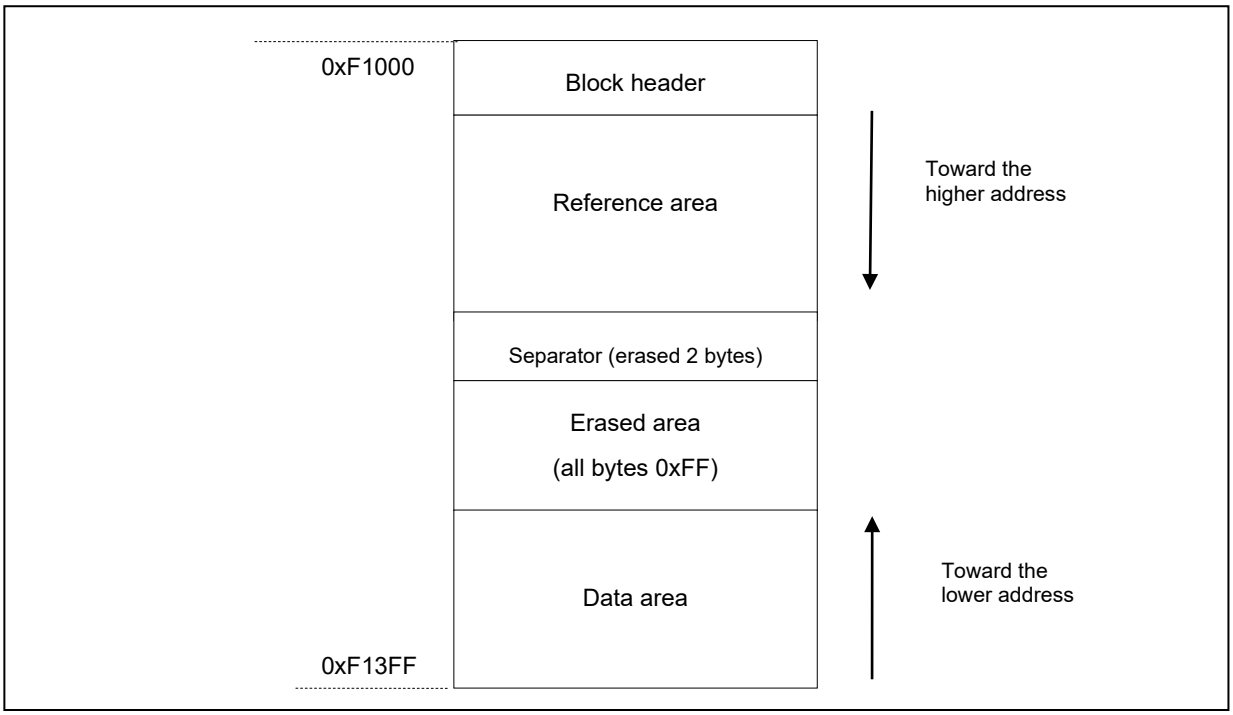


Figure 2-4 EEL Block Structure (Example of RL78/G13 Data Flash Block 0)

Table 2-4 Configuration of Each EEL Block

Name	Description
Block header	The block header contains all block status information needed for the block management within the EEL-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 EEL 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 2-4.

The EEL block header is detailed in section "2.3.4 EEL Block Header", while the structure of data stored in the reference and data area are described in section "2.3.5 Structure of Stored Data".

2. 3. 4 EEL Block Header

The structure of the block header is depicted in Figure 2-5. It is composed of 8 bytes, four of which are reserved for the system.

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

Figure 2-5 Structure of EEL Block Header

The block status flags start at the beginning of the block and include the A flag, B flag, I flag, and X flag, each of which is 1 byte, for a total of 4 bytes of data. The combination of flags indicates the EEL block status.

Figure 2-5 shows the placement status of flags, and Table 2-5 shows the combination status of flags.

Table 2-5 Overviews of Block Status Flags

Block Status Flag				State	Description
A Flag	B Flag	I Flag	X Flag		
0x01	0xFE	0xFF	0xFF	Active	Currently used block After the EEL_CMD_REFRESH command is executed, the A flag of a new active block is set to 0x02.
0x02	0xFD	0xFF	0xFF		Currently used block After the EEL_CMD_REFRESH command is executed, the A flag of a new active block is set to 0x03.
0x03	0xFC	0xFF	0xFF		Currently used block After the EEL_CMD_REFRESH command is executed, the A flag of a new active block is set to 0x01.
Data other than the above		0xFF	0xFF	Invalid	Invalid block
--		other than 0xFF	0xFF		
--		--	other than 0xFF	Excluded	Excluded block

2. 3. 5 Structure of Stored Data

The structure of stored data when user data is written to an EEL 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) field and the data field. The EEL descriptor table can be used to set data for use in the EEL. 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 EEL descriptor can be found in section "4.3 Initial Values to be Set by User".)

Each time data is written, stored data increase in the EEL block and multiple units of stored data exist in the EEL block, but only the most recent stored data is referenced.

SoR 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 2-6 shows the overview of the entire structure of stored data.

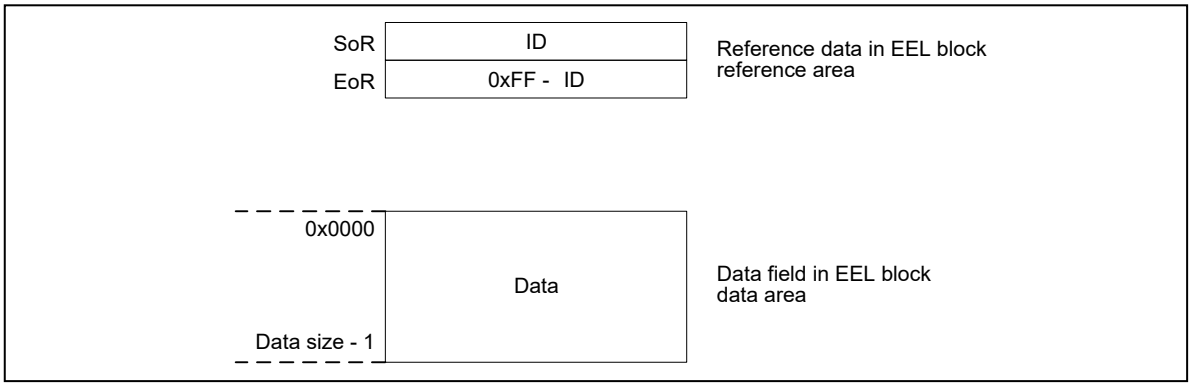


Figure 2-6 Structure of Stored Data

Table 2-6 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 EEL.
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 2-7).

Data is written to the EEL block in the order of SoR -> data field -> EoR. If write processing does not end successfully, the immediately previous data becomes valid.

Note 1: The total size of the reference data consumed by each stored data is 2 bytes. This should be considered when evaluating the free space in a block before writing the data through the EEL_GetSpace function.

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

2. 3. 6 EEL Block Overview

Figure 2-7 shows an example of an EEL 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.

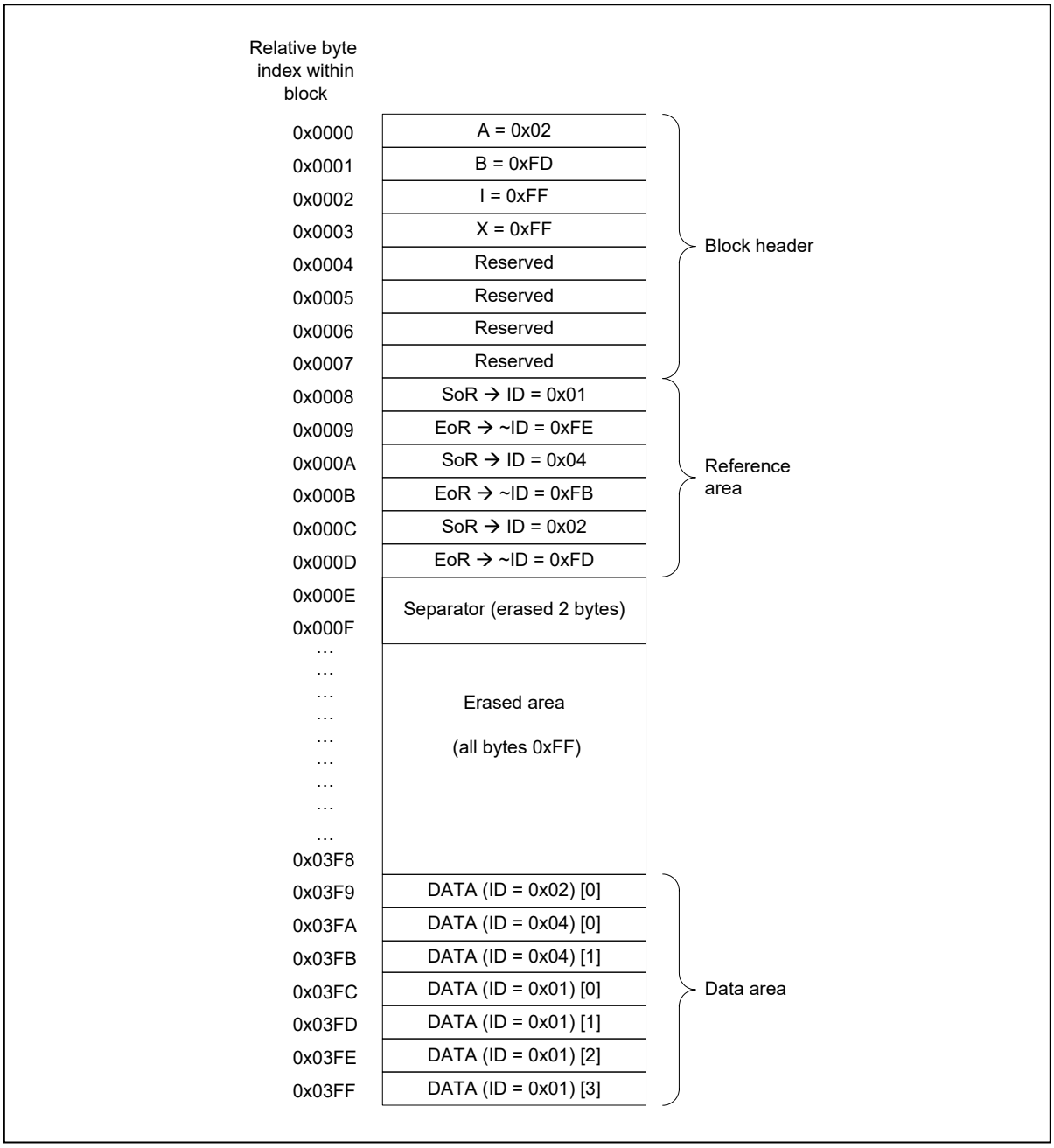


Figure 2-7 Example of an Active EEL Block

Chapter 3 EEL Functional Specifications

This chapter describes the functional specifications of the EEL required for the user to execute EEPROM emulation.

3.1 EEL Functions/Commands of the EEL_Execute Function

The table below summarizes the EEL functions offered by the EEL.

Table 3-1 EEL Functions

EEL function name	Functional overview
FDL_Init	Initializes the FDL.
FDL_Open	Preparation processing of FDL
FDL_Close	End processing of FDL
EEL_Init	Initializes the EEL
EEL_Open	Preparation processing of EEL
EEL_Close	End processing of EEL
EEL_Execute	Manipulates data flash memory with commands. Commands : EEL_CMD_STARTUP EEL_CMD_WRITE EEL_CMD_READ EEL_CMD_REFRESH EEL_CMD_VERIFY EEL_CMD_FORMAT EEL_CMD_SHUTDOWN
EEL_Handler	Controls the EEL while it is running.
EEL_GetSpace	Checks free space in the EEL block
EEL_GetVersionString	Obtains EEL version information.

With the EEL_Execute function, the following commands can be executed.

3. 1. 1 EEL_CMD_STARTUP Command [Startup Processing]

Check the block state and set the system to the EEPROM emulation start state (started).

3. 1. 2 EEL_CMD_SHUTDOWN Command [Shutdown Processing]

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

3. 1. 3 EEL_CMD_REFRESH Command [Refresh Processing]

Copy the latest stored data from the active block (copy source block) to the next block (copy destination block) in the EEL pool after the erase processing. This makes the copy destination block active.

3. 1. 4 EEL_CMD_FORMAT Command [Format Processing]

Initialize (erase) everything, including the data recorded in the EEL blocks. Be sure to use this command before using EEPROM emulation for the first time.

3. 1. 5 EEL_CMD_WRITE Command [Write Processing]

Write the specified data to an EEL block.

3. 1. 6 EEL_CMD_READ Command [Read Processing]

Read the specified data from an EEL block.

3. 1. 7 EEL_CMD_VERIFY Command [Verify Processing]

Perform verification (internal verification) to check signal levels of the active block.

3. 2 State Transitions

To use EEPROM emulation from a user-created program, it is necessary to initialize the EEL and execute functions that perform operations such as reading and writing on EEL blocks. Figure 3-1 shows the overall state transitions, and Figure 3-2 shows an operation flow for using basic features. When using EEPROM emulation, incorporate EEPROM emulation into user-created programs by following this flow.

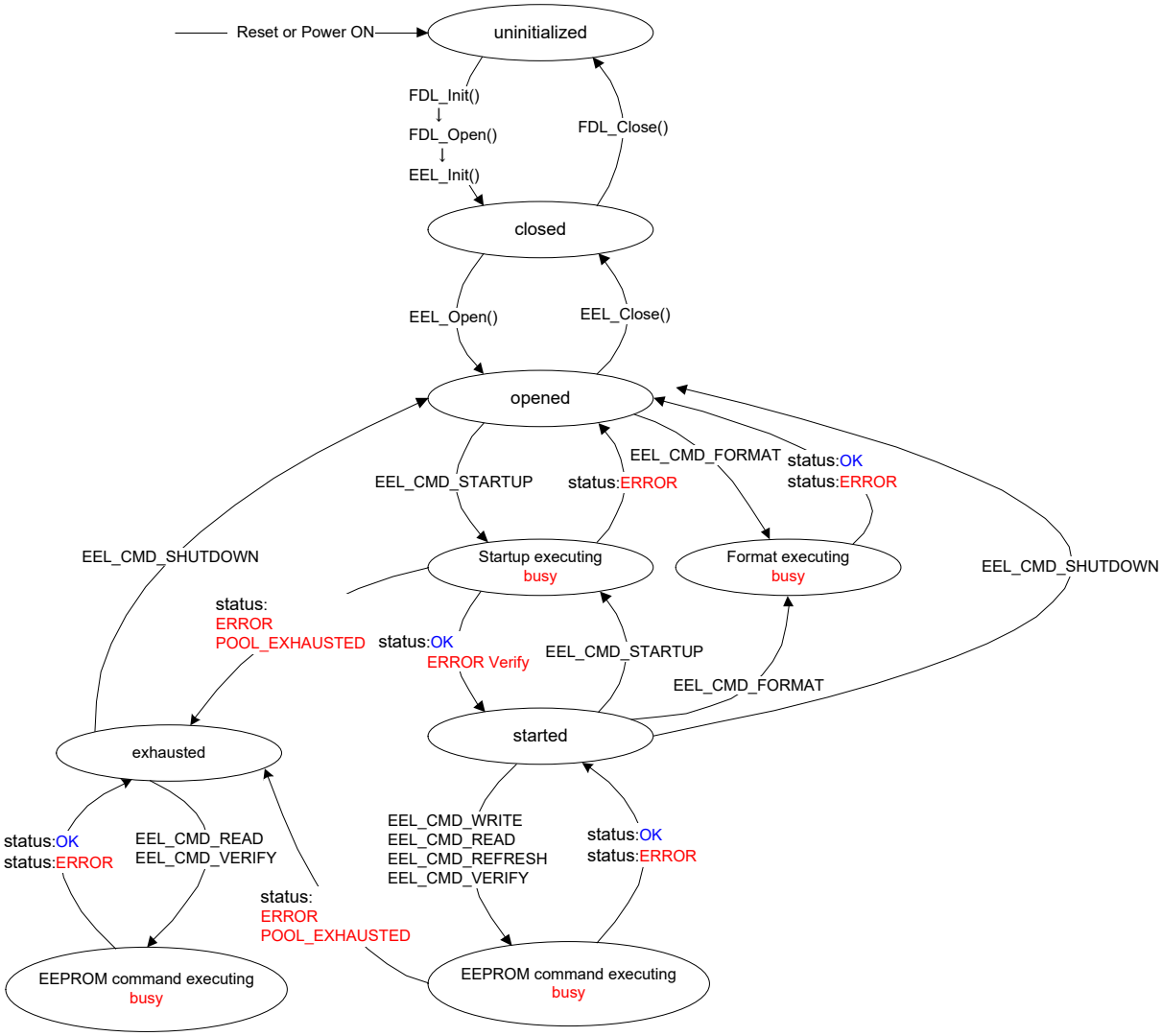


Figure 3-1 State Transitions Diagram

<R> Note 1: In case where discontinuing the execution of the EEL is urgent, the EEL_Close function can be called from any state. Continuing by calling the FDL_Close function enables the initialization of hardware that controls the flash memory and halting the EEL. In cases where restoring the EEL to its initial state is urgent, the EEL_Init function can be called from any state only when an EEL function is not being executed. However, since such urgent processing is only required in abnormal situations, the EEL may operate in an unpredictable way. In normal situations, we recommend executing the functions in accord with the state transition diagram.

Note 2: Once the EEL_CMD_FORMAT command has started running, execute the EEL_Handler function to check for its completion.

[Overview of state transitions diagram]

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

(1) uninitialized

This is the state after turning the power on or resetting.

(2) closed

This is the state in which the data to perform EEPROM emulation is initialized by executing the FDL_Init, FDL_Open, and EEL_Init functions (no ongoing operation to the data flash memory).

To execute FSL, STOP mode, or HALT mode processing after executing EEPROM emulation, execute EEL_Close in the opened state to switch to the closed state.

(3) opened

This state is switched to by executing EEL_Open in the closed state and makes it possible to perform operations on the data flash memory. It is not possible to execute FSL, STOP mode, or HALT mode processing until EEL_Close is executed and the system switches to the closed state.

(4) started

This state is switched to by executing the EEL_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 EEL blocks have been exhausted during command execution. In this state, only EEL_CMD_READ, EEL_CMD_VERIFY, and EEL_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.

3. 3 Basic Flowchart

Figure 3-2 below shows the basic procedure to perform read and write operations for the data flash by using the EEL.

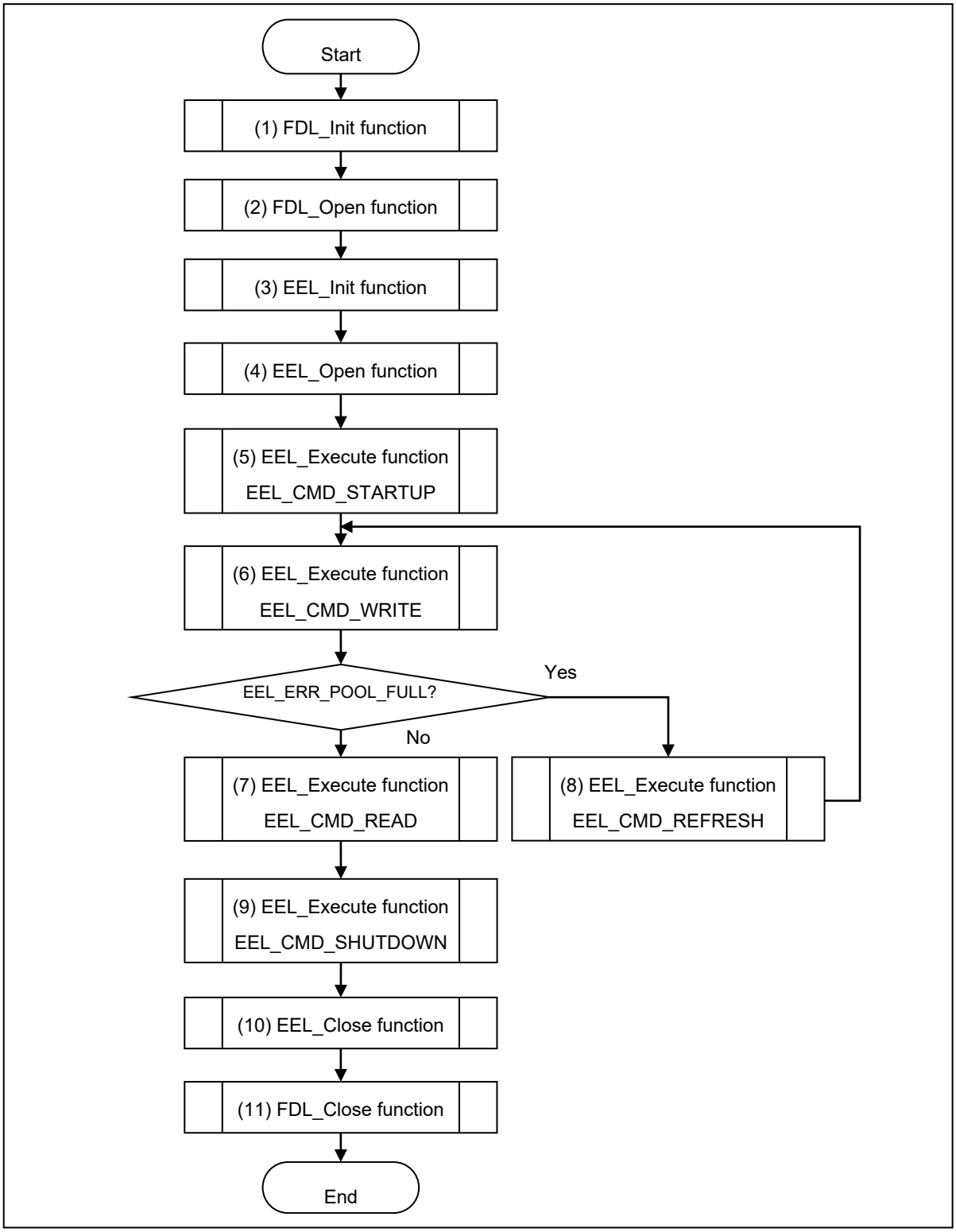


Figure 3-2 Basic Flowchart of EEL

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

Note 2: Error processing is omitted in the above flowchart.

[Overview of basic operation flow]

(1) FDL initialization processing (FDL_Init)

Because it is necessary to initialize the FDL parameters (RAM) if using the EEL to access the data flash memory, the FDL_Init function must be executed in advance. If FSL processing was executed after this initialization finished, the initialization processing must be re-executed.

(2) FDL preparation processing (FDL_Open)

Set the data flash control register (DFLCTL) to the state where accessing the data flash memory is permitted (DFLEN = 1).

(3) EEL initialization processing (EEL_Init)

Initialize the parameters (RAM) used by the EEL.

(4) EEPROM emulation preparation processing (EEL_Open)

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

(5) EEPROM emulation execution start processing (EEL_Execute: EEL_CMD_STARTUP command)

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

(6) EEPROM emulation data write processing (EEL_Execute: EEL_CMD_WRITE command)

Write the specified data to an EEL block.

(7) EEPROM emulation data read processing (EEL_Execute: EEL_CMD_READ command)

Read the specified data from an EEL block.

(8) EEPROM emulation refresh processing (EEL_Execute: EEL_CMD_REFRESH command)

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

(9) EEPROM emulation execution stop processing (EEL_Execute: EEL_CMD_SHUTDOWN command)

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

(10) EEPROM emulation end processing (EEL_Close)

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

(11) FDL end processing (FDL_Close)

Set the data flash control register (DFLCTL) to the state where accessing the data flash memory is inhibited (DFLEN = 0).

3. 4 Command Operation Flowchart

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

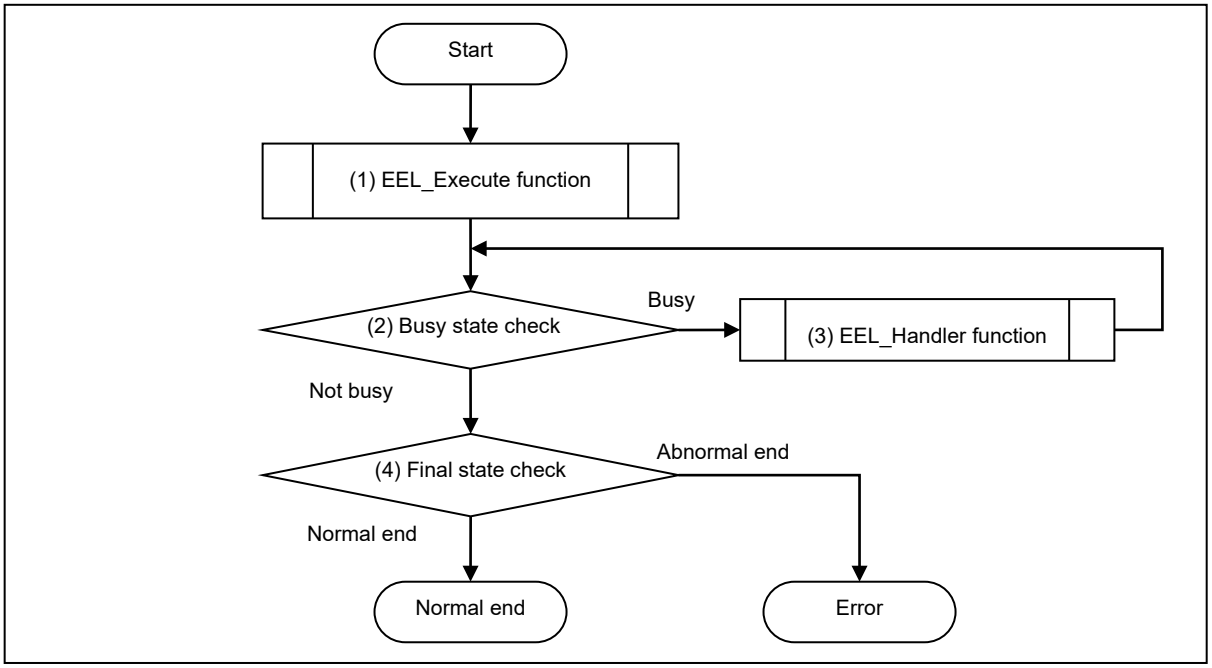


Figure 3-3 Command Operation Flowchart

(1) EEL_Execute function

Perform operations for data flash memory.

(2) Busy state check

Check status_enu of the request structure (eel_request_t). When status_enu is EEL_BUSY, continue the data flash operation. If the value of status_enu is other than EEL_BUSY, check the final state.

(3) EEL_Handler function

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

(4) Final state check

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

3. 5 BGO (Back Ground Operation) Function

The EEL_Execute function starts command processing and then immediately returns the control to the user program. This allows the user program to run during the data flash operation and so is called back ground operation (BGO).

The first round of processing of a command for writing data to or reading data from for the data flash memory is executed by calling the EEL_Execute function, and the second and later rounds are executed by calling the EEL_Handler function. For this reason, the EEL_Handler function must be called several times. For the processing to handle hardware divided into multiple devices, since other processing is suspended from the completion of processing until the next trigger, successive calls of the EEL_Handler function over a long interval will extend the overall processing time.

To see if the processing requested from the EEL_Execute function has been successfully completed, call the EEL_Handler function from the user program and check the status of ongoing processing.

The EEL_CMD_SHUTDOWN command does not require calling of the EEL_Handler function. However, we recommend that you follow the command operation flowchart shown in Figure 3-3.

Chapter 4 Using EEPROM Emulation

EEPROM emulation can store a maximum of 64 Note 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 EEL into a user-created program and executing that program.

Note: For details about the number of user data items that can be stored, see "4.2 Number of Stored User Data Items and Total User Data Size".

4. 1 Caution Points

EEPROM emulation is achieved by using a feature for manipulating the on-board microcontroller data flash memory. Therefore, it is necessary to note the following

Table 4-1 Points for Caution (1/3)

No	Caution Points
<R> 1	All EEL code and constants must be placed in the same 64-KB flash block such that EEL code and constants do not extend across a 64-KB boundary.
2	Initialization by the FDL_Init function must be performed before the FDL_Open, FDL_Close, or any EEL function is executed.
3	The EEL must be initialized by the EEL_Init function before any EEL function is executed.
4	The data flash memory cannot be read during data flash memory operation by the EEL.
5	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 EEL_Close function and FDL_Close function to finish EEPROM emulation.
6	The watchdog timer does not stop during execution of the EEL.
7	The request structure (eel_request_t) must be placed at an even address.
8	Do not destroy the request structure (eel_request_t) during command execution.
9	Initialize the argument (RAM) that is used by the EEPROM emulation library function. When not initialized, a RAM parity error is detected and the RL78 microcontroller might be reset. For a RAM parity error, refer to the "User's Manual: Hardware" of the target RL78 microcontroller.
10	All members of the request structure (eel_request_t) must be initialized once before a command is executed. If any unused member exists in the request structure (eel_request_t), set a desired value for the member. If any member is not initialized, the RL78 microcontroller may be reset due to a RAM parity error. For details, see the "User's Manual: Hardware" for the RL78 microcontroller in use.
11	The EEL does not support multitask execution. Do not execute the EEL functions during interrupt processing.

Table 4-2 Points for Caution (2/3)

No	Caution Points
12	After the FDL_Close and EEL_Close functions have been executed, the requested command and ongoing command stop and cannot be resumed. Before calling the FDL_Close and EEL_Close functions, finish all ongoing commands.
13	Before using the EEPROM emulation library, always close the FSL. Also, do not run the FSL while the EEPROM emulation library is being used. When using the FSL, be sure to execute all of the processing up to and including the EEL_Close function and FDL_Close function to finish EEPROM emulation. When using EEPROM emulation after executing FSL processing, it is necessary to start processing from the initializing function (the FDL_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	In address above 0xFFE20 (0xFE20), do not place data buffer (argument) or stack which is used by EEL and FDL functions.
16	When using data transfer controller (DTC) during EEPROM emulation, do not place RAM area used by DTC in self-RAM and in address above 0xFFE20 (0xFE20).
17	Until EEPROM emulation is finished, do not corrupt RAM area (including self-RAM) used by EEPROM emulation.
18	No checksum is added to user data. If a checksum is needed, add it to user data and check through the user program.
19	When the FDL descriptor or EEL descriptor is changed, the EEPROM emulation can no longer be executed. In that case, the EEL pool must be formatted by the EEL_CMD_FORMAT command in addition to initialization of FDL and EEL. When adding data, however, the EEPROM emulation can be continuously executed.
20	Do not operate the data flash control register (DFLCTL) during execution of the EEL.
21	To use the data flash memory for EEPROM emulation, it is necessary to execute the EEL_CMD_FORMAT command upon first starting up to initialize the data flash memory and make it usable as EEPROM emulation blocks.
22	It is recommended that at least three blocks be provided in the data flash memory to use the EEL.
23	Do not destroy EEL blocks by the user program that uses other EELs or FDLs.
24	The EEL does not support multitask execution. When executing an EEL function on the OS, do not execute in from two or more tasks.
25	About an operating frequency of RL78 microcontrollers and an operating frequency value set by the initializing function (FDL_Init), be aware of the following points: - When using a frequency lower than 4 MHz as an operating frequency of RL78 microcontrollers, only 1 MHz, 2 MHz and 3 MHz can be used (frequencies other than integer values like a 1.5 MHz cannot be used). Also, set an integer value 1, 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 microcontrollers, a certain frequency can be used as an operating frequency of RL78 microcontrollers. - This operating frequency is not the frequency of the high-speed on-chip oscillator.

Note: For a maximum frequency, see the target RL78 microcontroller "User's Manual: Hardware".

Table 4-3 Points for Caution (3/3)

No	Caution Points

<R> **26**	When using an assembler of the CC-RL compiler from Renesas Electronics, the hexadecimal prefix representation (0x..) cannot be mixed together with the suffix representation (..H). Specify the representation method by editing the symbol definition in eel_types.inc to match the user environment.
	
	eel_types.inc
	`;__EEL_TYPES_INC_BASE_NUMBER_SUFFIX .SET 1`
	When symbol "__EEL_TYPES_INC_BASE_NUMBER_SUFFIX" is not defined (initial state), the prefix representation will be selected.
	
	eel_types.inc
	`__EEL_TYPES_INC_BASE_NUMBER_SUFFIX .SET 1`
	When symbol "__EEL_TYPES_INC_BASE_NUMBER_SUFFIX" is defined, the suffix representation will be selected.

4. 2 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 EEL 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 of one block of data flash memory: 1024 bytes
Size required for EEPROM emulation block management: 8 bytes
Free space necessary as termination information (separator): 2 bytes

Maximum usable size of one block = 1024 bytes - 8 bytes - 2 bytes = 1014 bytes

[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 = the basic total user data size + maximum data size + 2 ≤ 1014
(Assumed that the largest data can be updated once after all data have been written.)

Recommended size = 1014 / 2
(Assumed that all data can be updated once after all data have been written.)

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

Size of each written user data item = data size + reference data size (2 bytes)
Note: For details, see 2.3.5 Structure of Stored Data.

[Calculating the basic total user data size**]**

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

4. 3 Initial Values to be Set by User

As the initial values for the EEL, be sure to set the items indicated below. In addition, before executing the EEL, 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)

<Data flash library user include file (fdl_descriptor.h)> Notes 1, 2

<R>

```
#define   FDL_SYSTEM_FREQUENCY    32000000    : (1) Operating frequency
#define   FDL_WIDE_VOLTAGE_MODE               : (2) Flash memory programming mode
#define   FDL_POOL_BLOCKS         0           : (3) FDL pool size
#define   EEL_POOL_BLOCKS         4           : (4) EEL pool size
```

<EEPROM emulation library user include file (eel_descriptor.h)> Notes 1, 2

```
#define   EEL_VAR_NO        8                 : (5) Number of stored data items
```

<EEPROM emulation library user program file (eel_descriptor.c)> Note 2

```
__far const eel_u08 eel_descriptor[EEL_VAR_NO+2] =        : (6) Data size of the identifier (data ID)
{
  (eel_u08)(EEL_VAR_NO),   /* variable count     */  ¥
  (eel_u08)(sizeof(type_A)),  /* id=1            */  ¥
  (eel_u08)(sizeof(type_B)),  /* id=2            */  ¥
  (eel_u08)(sizeof(type_C)),  /* id=3            */  ¥
  (eel_u08)(sizeof(type_D)),  /* id=4            */  ¥
  (eel_u08)(sizeof(type_E)),  /* id=5            */  ¥
  (eel_u08)(sizeof(type_F)),  /* id=6            */  ¥
  (eel_u08)(sizeof(type_X)),  /* id=7            */  ¥
  (eel_u08)(sizeof(type_Z)),  /* id=8            */  ¥
  (eel_u08)(0x00),            /* zero terminator */  ¥
};
```

Note 1: The macros that are being used are parameters which are common to the whole EEL, so any changes should only be to numerical values.

Note 2: After initializing the EEPROM emulation blocks (after executing the EEL_CMD_FORMAT command), do not change the values. If the values are changed, reinitialize the EEL blocks (by executing the EEL_CMD_FORMAT command).

(1) Operating frequency

Set an operating frequency which is used in RL78 microcontrollers. Note 1

The setting value is set to the FDL_Init frequency parameter by the following expressions (The frequency is calculated by raising its decimals. The result calculated omits its decimals.).

Setting value of FDL_Init operating frequency = ((FDL_SYSTEM_FREQUENCY + 999999) / 1000000)

Ex.1: When FDL_SYSTEM_FREQUENCY is 20000000 (20 MHz),
((20000000 + 999999) / 10000000) = 20.999999 = 20

Ex.2: When FDL_SYSTEM_FREQUENCY is 4500000 (4.5 MHz),
((4500000 + 999999) / 10000000) = 5.499999 = 5

Ex.3: When FDL_SYSTEM_FREQUENCY is 5000001 (5.000001 MHz),
((5000001 + 999999) / 10000000) = 6.000000 = 6

Note 1: This setting is a value required to control data flash memory. This setting does not change the operating frequency of RL78 microcontrollers. In addition, this operating frequency is not the frequency of the high-speed on-chip oscillator.

<R> (2) Flash memory programming mode Note 2

Set the flash memory programming mode of data flash memory. Note 3

When FDL_WIDE_VOLTAGE_MODE is not defined: Full-speed mode

When FDL_WIDE_VOLTAGE_MODE is defined: Wide voltage mode

Note 2: The FDL_WIDE_VOLTAGE_MODE is commented out and not defined in the initial setting. To use RL78 microcontrollers in the wide voltage mode, cancel the comment-out to define the mode.

Note 3: For details of the flash memory programming mode, see the corresponding RL78 microcontrollers user's manual.

(3) FDL pool size

Specify 0.

(4) EEL pool size Note 4

The number of blocks in the data flash memory of the target device must be specified as the number of blocks in the EEL pool.

Note 4: Specify 3 (3 blocks) or a greater value (recommended).

(5) Number of stored data items

Specify the number of data items to be used in the EEPROM emulation. A value of 1 to 64 can be set.

(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 EEL descriptor table. The EEL can only add identifiers while the program is running. Data to be written must be registered in the EEL descriptor table in advance.

__far const eel_u08 eel_descriptor [Number of stored data items + 2]

EEL_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 EEL Descriptor Table (When there are eight different data items)

- EEL_VAR_NO

  User-specified number of data items used in the EEL

- Byte size of data IDx

  User-specified size of user data (in bytes)

- Termination area (0x00)

  Specify 0 as the termination information.

Chapter 5 User Interface

5. 1 Request Structure (eel_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 EEL via the request structure (eel_request_t). Furthermore, the EEL state and error information are acquired via the request structure (eel_request_t).

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

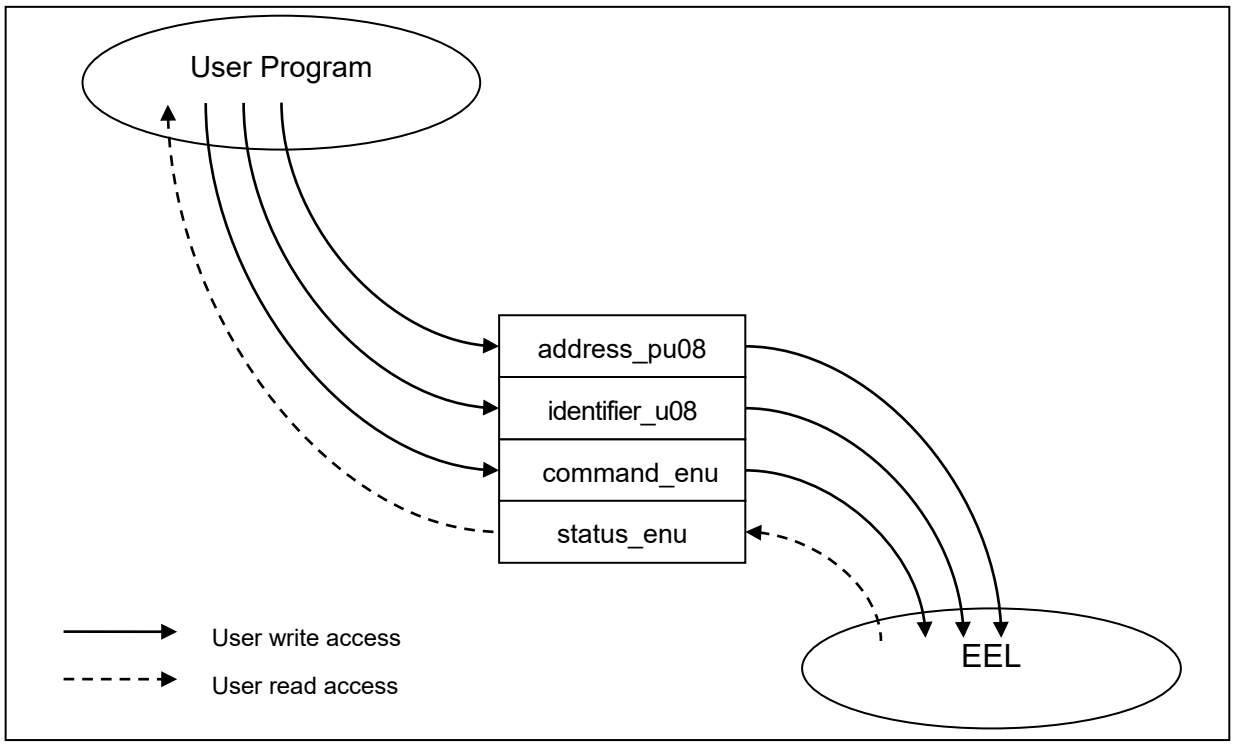


Figure 5-1 Request Structure (eel_request_t)

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

[Definition of the request structure (eel_request_t)]

<R>

Development Tool	C Language (Structure Definition)	Assembly Language (Example of definition)
RENESAS CA78K0R compiler	typedef struct { __near eel_u08* address_pu08; eel_u08 identifier_u08; eel_command_t command_enu; eel_status_t status_enu; } eel_request_t;	_request_pstr: _address_pu08: DS 2 _identifier_u08: DS 1 _command_enu: DS 1 _status_enu: DS 1
RENESAS CC-RL compiler	typedef struct { __near eel_u08* address_pu08; eel_u08 identifier_u08; eel_command_t command_enu; eel_status_t status_enu; } eel_request_t;	_request_pstr: _address_pu08: .DS 2 _identifier_u08: .DS 1 _command_enu: .DS 1 _status_enu: .DS 1
LLVM compiler	typedef struct { __near eel_u08* address_pu08; eel_u08 identifier_u08; eel_command_t command_enu; eel_status_t status_enu; } eel_request_t;	Check the compiler specifications.

__near eel_u08* address_pu08	
eel_u08 identifier_u08	eel_command_t command_enu
eel_status_t status_enu	

Bit 0 Bit 15

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

Note: The request structure (eel_request_t) must be placed at an even address.

5.1.1 User Write Access

(1) address_pu08

Specify the start address of the data buffer used for EEL_CMD_WRITE command and EEL_CMD_READ command execution.

Associated command (macro name)	Setting
EEL_CMD_WRITE	Start address of the data buffer Note 1
EEL_CMD_READ	Start address of the data buffer Note 2

Note 1: Buffer which contains data written by the user

Note 2: Buffer which contains data read from the data flash memory

(2) identifier_u08

Specify the data ID used for each command. For more information about how to do this, see the description of the EEL_Execute function in section "5.4 EEL Functions".

Associated command (macro name)	Setting
EEL_CMD_WRITE	ID of write data
EEL_CMD_READ	ID of read data

(3) command_enu

Commands to be set in the common executable function.

Associated command (macro name)	Description
EEL_CMD_STARTUP	Startup processing
EEL_CMD_WRITE	Write processing
EEL_CMD_READ	Read processing
EEL_CMD_REFRESH	Refresh processing
EEL_CMD_VERIFY	Verify processing
EEL_CMD_FORMAT	Format processing
EEL_CMD_SHUTDOWN	Shutdown processing

5.1.2 User Read Access

- **status_enu**

EEL status and error information. For information about the status and errors which might occur during execution of the functions, see the description of the EEL_Execute function in section "5.4 EEL Functions".

5. 2 EEL Function Calls

This section describes how to call the EEL functions from a user program written in C or assembly language.

- C language

When an EEL function is called from a user program in C language in the same way as a normal C function is called, the EEL function's parameters are passed to the EEL as arguments and the required processing is performed.

- Assembly language

Before calling an EEL function from a user program in assembly language, take necessary procedures (such as setting parameters or return addresses) based on the function-calling rules for the C compiler package used by the user as a development environment. The EEL function's parameters are passed to the EEL as arguments and the required processing is performed.

Remarks 1: To call the EEL functions offered by the EEL from a user program, you should define the following standard header file and include it in that program:

C language

fdl.h: FDL header file

fdl_types.h: FDL definition setting header file

eel.h: EEL header file

eel_types.h: EEL definition setting header file

Assembly language

fdl.inc: FDL header file

eel.inc: EEL header file

eel_types.inc: EEL definition setting header file

2: If an EEL function other than EEL_Init is called before EEL_Init is called, the correct operation is not guaranteed.

3: If an EEL function other than FDL_Init is called before FDL_Init is called, the correct operation is not guaranteed.

<R> **4**: The assembly language files for the LLVM compiler are not included in this product.

5. 3 Data Types

Below are the data types of the parameters to be specified for calling the EEL functions offered by the EEL.

Macro name	Description
eel_u08	Unsigned 8-bit integers (unsigned char)
eel_u16	Unsigned 16-bit integers (unsigned short)
eel_u32	Unsigned 32-bit integers (unsigned long)

5. 4 EEL Functions

The subsequent sections describe the EEL functions offered by the EEL. These functions appear in the following format.

Name

[Function]

Describes the function overview of this function.

[Format]

<C language>

<R> Describes the format for calling the given function from a user program written in the C language for compilation by the Renesas CA78K0R, Renesas CC-RL or LLVM compiler.

<Assembler>

Describes the format to call this function from a user program written in the assembly language.

[Pre-conditions]

Describes the precondition of this function.

[Post-conditions]

Describes the post condition of this function.

[Cautions]

Describes the cautions of this function.

[Register status after calling this function]

Describes the register status after this function is called.

<R> The general-purpose registers and registers to be corrupted which are used for the return values differ according to whether the Renesas CA78K0R, Renesas CC-RL or LLVM compiler is in use.

[Arguments]

Describes the argument of this function.

<R> Definitions of registers and arguments which are used in the assembly language may differ according to whether the Renesas CA78K0R, Renesas CC-RL or LLVM compiler is in use.

[Return values]

Describes the return values from this function.

FDL_Init

[Function]
FDL initialization processing

[Format]
<C language>
Renesas CA78K0R compiler

```
fdl_status_t __far FDL_Init(const __far fdl_descriptor_t* descriptor_pstr)
```

<R> Renesas CC-RL compiler

```
fdl_status_t __far FDL_Init(const __far fdl_descriptor_t* descriptor_pstr)
```

<R> LLVM compiler

```
fdl_status_t __far FDL_Init(const __far fdl_descriptor_t* descriptor_pstr)
                                                    __attribute__ ((section ("FDL_CODE")))
```

<Assembler>

```
CALL !FDL_Init or CALL !!FDL_Init
```

Remark Call this function with "!" if you are placing the FDL at 00000H to 0FFFFH or with "!!" otherwise.

[Pre-conditions]

1. The FSL and EEL processing must be either not executing or finished.
2. The high-speed on-chip oscillator has been started up.

[Post-conditions]

Execute the FDL_Open function.

[Cautions]

1. Be sure to execute this function when starting EEPROM emulation to make it possible to start accessing the data flash memory.
2. This function is mutually exclusive with the FSL. Before executing this function, be sure to end FSL. Also, never use any FSL functions during EEPROM emulation.
3. To use FSL after this function is executed, the RAM must be reinitialized, so always execute this function when restarting the EEL.
4. To execute this function again, always be sure to end EEL.
5. The descriptor table used for this function cannot be modified. Be sure to use a defined descriptor table.

[Register status after calling this function]

<R>

Development Tool	Registers Used for Return Value	Destructed Registers
Renesas CA78K0R compiler	C (General-purpose register)	AX, BC
Renesas CC-RL compiler	A (General-purpose register)	X, BC, DE, HL
LLVM compiler	A (General-purpose register)	X, BC, DE, HL

<R> [Arguments]

descriptor_pstr: Pointer to the descriptor table

For details of "descriptor table", refer to the description of “<Data flash library user include file (fdl_descriptor.h)>” in "4.3 Initial Values to be Set by User".

Contents of Argument Settings

Development Tool	C Language	Assembly Language (Register)
RENESAS CA78K0R compiler	const __far fdl_descriptor_t* descriptor_pstr	AX (0 to 15), C (16 to 23): The start address of the structure argument (24 bits)
RENESAS CC-RL compiler	const __far fdl_descriptor_t* descriptor_pstr	DE (0 to 15), A (16 to 23): The start address of the structure argument (24 bits)
LLVM compiler	const __far fdl_descriptor_t* descriptor_pstr	DE (0 to 15), A (16 to 23): The start address of the structure argument (24 bits)

[Return values]

Type	Symbol Definition	Description
fdl_status_t	FDL_OK	Normal end
	FDL_ERR_CONFIGURATION	Initialization error. The setting is incorrect. Or high-speed on-chip oscillator does not run. Make sure that the defined data has not been changed and the high-speed on-chip oscillator has been started up.

FDL_Open

[Function]

FDL preparation processing

Set the data flash control register (DFLCTL) to the state where accessing the data flash memory is permitted (DFLEN = 1).

[Format]

<C language>

Renesas CA78K0R compiler

```
void __far FDL_Open(void)
```

<R> Renesas CC-RL compiler

```
void __far FDL_Open(void)
```

<R> LLVM compiler

```
void __far FDL_Open(void) __attribute__ ((section ("FDL_CODE")))
```

<Assembler>

```
CALL !FDL_Open or CALL !!FDL_Open
```

Remark Call this function with "!" if you are placing the FDL at 00000H to 0FFFFH or with "!!" otherwise.

[Pre-conditions]

The FDL_Init function must have finisehed normally.

[Post-conditions]

Execute the EEL_Init function.

[Cautions]

None

[Register status after calling this function]

<R>

Development Tool	Registers Used for Return Value	Destructed Registers
Renesas CA78K0R compiler	None	None
Renesas CC-RL compiler	None	AX
LLVM compiler	None	AX

[Arguments]

None

[Return values]

None

FDL_Close

[Function]

FDL end processing

Set the data flash control register (DFLCTL) to the state where access to the data flash memory is inhibited (DFLEN = 0). All ongoing EEL processing stop.

[Format]

<C language>

Renesas CA78K0R compiler

```
void __far FDL_Close(void)
```

<R> Renesas CC-RL compiler

```
void __far FDL_Close(void)
```

<R> LLVM compiler

```
void __far FDL_Close(void) __attribute__ ((section ("FDL_CODE")))
```

<Assembler>

```
CALL !FDL_Close or CALL !!FDL_Close
```

Remark Call this function with "!" if you are placing the FDL at 00000H to 0FFFFH or with "!!" otherwise.

[Pre-conditions]

The FDL_Init, FDL_Open, EEL_Init, EEL_Open, and EEL_Close functions must have finished normally.

[Post-conditions]

None

[Cautions]

None

[Register status after calling this function]

<R>

Development Tool	Registers Used for Return Value	Destructed Registers
Renesas CA78K0R compiler	None	None
Renesas CC-RL compiler	None	C
LLVM compiler	None	C

[Arguments]

None

[Return values]

None

EEL_Init

[Function]
Processing to initialize the RAM used for EEPROM emulation

[Format]
<C language>

Renesas CA78K0R compiler

```
eel_status_t  __far EEL_Init (void)
```

<R> Renesas CC-RL compiler

```
eel_status_t  __far EEL_Init (void)
```

<R> LLVM compiler

```
eel_status_t  __far EEL_Init (void) __attribute__ ((section ("EEL_CODE")))
```

<Assembler>

```
CALL !EEL_Init or CALL !!EEL_Init
```

Remark Call this function with "!" if you are placing the EEL at 00000H to 0FFFFH or with "!!" otherwise.

[Pre-conditions]

1. The FSL and the EEL processing must be either not executed or finished.
2. The FDL_Init and FDL_Open functions must have finished normally.

[Post-conditions]

Execute the EEL_Open function.

[Cautions]

1. Be sure to execute this function when starting EEPROM emulation to initialize the RAM to be used.
2. This function is mutually exclusive with FSL. Before executing this function, be sure to close FSL. Also, never use any FSL functions during EEPROM emulation.
3. To use FSL after this function is executed, the RAM must be reinitialized, so always execute this function when restarting EEL.
4. To execute this function again, always close EEL.

[Register status after calling this function]

<R>

Development Tool	Registers Used for Return Value	Destructed Registers
Renesas CA78K0R compiler	C (General-purpose register)	None
Renesas CC-RL compiler	A (General-purpose register)	X, BC, D, HL
LLVM compiler	A (General-purpose register)	X, BC, D, HL

[Arguments]

None

[Return values]

Type	Symbol Definition	Description
eel_status_t	EEL_OK	Normal end
	EEL_ERR_CONFIGURATION	Initialization error. The EEL cannot be executed with the values set by the FDL_Init and EEL_Init functions. Check the current setting referring to section "4.3 Initial Values to be Set by User".

EEL_Open

[Function]

EEPROM emulation preparation processing

This function makes the EEPROM emulation executable.

[Format]

<C language>

Renesas CA78K0R compiler

```
void   __far EEL_Open(void)
```

<R> Renesas CC-RL compiler

```
void   __far EEL_Open(void)
```

<R> LLVM compiler

```
void   __far EEL_Open(void) __attribute__ ((section ("EEL_CODE")))
```

<Assembler>

```
CALL !EEL_Open or CALL !!EEL_Open
```

Remark Call this function with "!" if you are placing the EEL at 00000H to 0FFFFH or with "!!" otherwise.

[Pre-conditions]

1. The FDL_Init, FDL_Open, and EEL_Init functions must have finished normally.
2. If EEPROM emulation was executed, the processing up to EEL_Close and FDL_Close functions must be executed to stop the EEPROM emulation processing.

[Post-conditions]

None

[Cautions]

After the EEL_Open function is executed and the EEPROM emulation has been transitioned to the opened state, the FSL cannot be executed. In addition, STOP mode and HALT mode cannot be executed. To execute the FSL, STOP mode, or HALT mode, execute the EEL_Close and FDL_Close functions to transition the EEPROM emulation to the uninitialized state.

[Register status after calling this function]

No registers are corrupted.

[Arguments]

None

[Return values]

None

EEL_Close

[Function]

EEPROM emulation end processing

This function makes the EEPROM emulation unexecutable.

[Format]

<C language>

Renesas CA78K0R compiler

```
void   __far EEL_Close(void)
```

<R> Renesas CC-RL compiler

```
void   __far EEL_Close(void)
```

<R> LLVM compiler

```
void   __far EEL_Close(void) __attribute__ ((section ("EEL_CODE")))
```

<Assembler>

```
CALL !EEL_Close or CALL !!EEL_Close
```

Remark Call this function with "!" if you are placing the EEL at 00000H to 0FFFFH or with "!!" otherwise.

[Pre-conditions]

If EEPROM emulation was executed, the EEL_CMD_SHUTDOWN command must be used to set EEPROM emulation to the stopped state (the opened state).

[Post-conditions]

Execute the FDL_Close function to exit the EEPROM emulation.

[Cautions]

None

[Register status after calling this function]

<R>

Development Tool	Registers Used for Return Value	Destructed Registers
Renesas CA78K0R compiler	None	None
Renesas CC-RL compiler	None	A
LLVM compiler	None	A

[Arguments]

None

[Return values]

None

EEL_Execute

[Function]

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.

[Format]

<C language>

Renesas CA78K0R compiler

```
void __far EEL_Execute(__near eel_request_t* request_pstr)
```

<R> Renesas CC-RL compiler

```
void __far EEL_Execute(__near eel_request_t* request_pstr)
```

<R> LLVM compiler

```
void __far EEL_Execute(__near eel_request_t* request_pstr) __attribute__ ((section ("EEL_CODE")
```

<Assembler>

```
    CALL !EEL_Execute or CALL !!EEL_Execute
```

Remark Call this function with "!" if you are placing the EEL at 00000H to 0FFFFH or with "!!" otherwise.

[Pre-conditions]

The FDL_Init, FDL_Open, EEL_Init, and EEL_Open functions must have finished normally.

[Post-conditions]

1. While status_enu of the request structure (eel_request_t) is EEL_BUSY, execute the EEL_Handler function repeatedly.
2. The EEL_Execute function starts command processing and then immediately returns the control to the user program. The command processing is continued by executing the EEL_Handler function. Therefore, the EEL_Handler function must be executed continuously until the command processing is completed.

[Cautions]

None

[Register status after calling this function]

<R>

Development Tool	Registers Used for Return Value	Destructed Registers
Renesas CA78K0R compiler	None	AX
Renesas CC-RL compiler	None	AX, BC, DE, HL
LLVM compiler	None	AX, BC, DE, HL

<R> [Arguments]

request_pstr: Pointer to the request structure (eel_request_t)

Contents of Argument Settings

Development Tool	C Language	Assembly Language (Register)
RENESAS CA78K0R compiler	__near eel_request_t* request_pstr	AX (0 to 15) The start address of the structure argument (16 bits)
RENESAS CC-RL compiler	__near eel_request_t* request_pstr	AX (0 to 15) The start address of the structure argument (16 bits)
LLVM compiler	__near eel_request_t* request_pstr	AX (0 to 15) The start address of the structure argument (16 bits)

Details of eel_request_t

Member	Type	Description
eel_request_t.address_pu08	eel_u08 * (near)	Pointer to the data buffer for storing write and read data Note
eel_request_t.identifier_u08	eel_u08	Data ID number
eel_request_t.command_enu	eel_command_t	Command to be executed Refer to "Execution Commands (eel_command_t)" for details on each command
eel_request_t.status_enu	eel_status_t	Command execution status

Note: Specify this parameter only for a command that requires the parameter. Set up the data buffer size according to the byte sizes of the write and read data.

Execution Commands (eel_command_t)

Command	Description
EEL_CMD_STARTUP	This command checks the block status and sets the system to the EEPROM emulation start state (started). 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 EEL_CMD_FORMAT command and make sure that the command finishes normally.
EEL_CMD_WRITE Note 1	This command writes the specified data to the EEL blocks. * The following arguments must be specified prior to execution. • address_pu08: Specifies the start address of the RAM area where the write data is stored. • identifier_u08: Specifies the data ID of the write data.
EEL_CMD_READ Note 1	This command reads the latest data from the EEPROM emulation blocks corresponding to the specified data ID. *The following arguments must be specified prior to execution. • address_pu08: Specifies the start address of the RAM area where the read data is stored. • identifier_u08: Specifies the data ID of the read data.
EEL_CMD_VERIFY Notes 1, 2	This command performs internal verification to check signal levels of the active block. This command verifies whether signal levels of flash memory cells are appropriate or not.
EEL_CMD_REFRESH Notes 1, 3	This command copies the latest stored data from the active EEL block (copy source block) to the next block (copy destination block) in the EEL pool after the erase processing. This makes the copy destination block active.
EEL_CMD_FORMAT	This command initializes (erases) everything, including the data recorded in the EEL blocks. Be sure to issue 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 EEL block (such as a valid 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 EEL_CMD_STARTUP command to continue using EEPROM emulation.
EEL_CMD_SHUTDOWN Note 1	This command sets EEPROM emulation to the stopped state (opened).

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

2. This command is not used to perform processing for reading written data and compare it. To compare written data, use the EEL_Execute (EEL_CMD_READ) function through the user program.

3. The erase processing is performed by executing the EEL_CMD_REFRESH command.

Command Execution States of EEL_Execute/EEL_Handler (eel_status_t) (1/2)

Command Execution Status	Category	Description	Corresponding Commands
EEL_OK	Meaning	Normal end	All commands
	Cause	None	
	Action to be taken	None	
EEL_BUSY	Meaning	A command is being executed.	All commands
	Cause	None	
	Action to be taken	Keep calling the EEL_Handler function until the status changes.	
EEL_ERR_POOL_FULL	Meaning	Pool full error	EEL_CMD_WRITE
	Cause	There is no area that can be used to write the data.	
	Action to be taken	Execute the EEL_CMD_REFRESH command and restart writing data.	
EEL_ERR_INITIALIZATION	Meaning	Initialization error	All commands
	Cause	The FDL_Init, FDL_Open, EEL_Init, and EEL_Open functions have not been finished normally.	
	Action to be taken	Normally finish the FDL_Init, FDL_Open, EEL_Init, and EEL_Open functions.	
EEL_ERR_ACCESS_LOCKED	Meaning	EEPROM emulation lock error	Commands other than EEL_CMD_STARTUP and EEL_CMD_FORMAT
	Cause	EEPROM emulation cannot be executed.	
	Action to be taken	Make sure that the EEL_CMD_STARTUP command has finished normally.	
EEL_CMD_UNDEFINED		Command error A command that does not exist has been specified.	---
EEL_ERR_VERIFY	Meaning	When the EEL_CMD_STARTUP command was executed: An error occurred during the internal verification processing for the block header or the finally written data. When the EEL_CMD_VERIFY command was executed: An error occurred during the internal verification processing for the active block.	EEL_CMD_STARTUP EEL_CMD_VERIFY
	Cause	Some signal levels of flash memory cells were not appropriate.	
	Action to be taken	Execute the EEL_CMD_REFRESH command.	
EEL_ERR_PARAMETER	Meaning	Parameter error	All commands
	Cause	An incorrect command parameter has been specified.	
	Action to be taken	Check the specified parameter.	

Command Execution States of EEL_Execute/EEL_Handler (eel_status_t) (2/2)

Command Execution Status	Category	Description	Corresponding Commands
EEL_ERR_REJECTED	Meaning	Reject error	All commands
	Cause	A different command is being executed.	
	Action to be taken	Call the EEL_Handler function to terminate the ongoing command.	
EEL_ERR_NO_INSTANCE	Meaning	No-write-data error	EEL_CMD_READ
	Cause	The specified identifier data has not been written.	
	Action to be taken	Write data to the identifier specified using the EEL_CMD_WRITE command.	
EEL_ERR_POOL_INCONSISTENT	Meaning	EEL block inconsistency error	EEL_CMD_STARTUP
	Cause	An EEL block has the undefined state (such as there are no active blocks).	
	Action to be taken	Execute the EEL_CMD_FORMAT command to initialize the EEL blocks.	
EEL_ERR_POOL_EXHAUSTED	Meaning	EEL block exhaustion error	EEL_CMD_STARTUP EEL_CMD_FORMAT EEL_CMD_REFRESH EEL_CMD_WRITE
	Cause	There are no more EEL blocks that can be used to continue.	
	Action to be taken	Stop EEPROM emulation. You can try restoration by executing the EEL_CMD_FORMAT command (erasing all existing data) or read existing data.	
EEL_ERR_INTERNAL	Meaning	Internal error	Commands other than EEL_CMD_SHUTDOWN
	Cause	An unexpected error has occurred.	
	Action to be taken	Check the device state.	

[Return values]

None

EEL_Handler

[Function]

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 EEL_Execute function to continue.

[Format]

<C language>

Renesas CA78K0R compiler

```
void __far EEL_Handler(void)
```

<R> Renesas CC-RL compiler

```
void __far EEL_Handler(void)
```

<R> LLVM compiler

```
void __far EEL_Handler(void) __attribute__ ((section ("EEL_CODE")))
```

<Assembler>

```
CALL !EEL_Handler or CALL !!EEL_Handler
```

Remark Call this function with "!" if you are placing the EEL at 00000H to 0FFFFH or with "!!" otherwise.

[Pre-conditions]

1. The FDL_Init, FDL_Open, EEL_Init, and EEL_Open functions must have finished normally.
2. The EEL_Execute function should be executed Note, and status_enu of the request structure (eel_request_t) should be EEL_BUSY.

Note: Execution of the EEL_CMD_SHUTDOWN command does not need execution of the EEL_Handler function. However, we recommend that you follow the command operation flowchart shown in Figure 3-3.

[Post-conditions]

While status_enu of the request structure (eel_request_t) is EEL_BUSY, execute this function repeatedly. If the EEL_Handler function is executed while no command is executed, status_enu of of the request structure (eel_request_t) is not updated.

[Cautions]

The command execution status of the EEL_Handler function is set for the "eel_request_t* request" used as an argument of the EEL_Execute function. Therefore, when using the EEL_Handler function, do not free the "eel_request_t* request" variable. For the execution state of the command specified by the EEL_Handler function, see the list of the Command Execution States of EEL_Execute/EEL_Handler (eel_status_t) .

[Register status after calling this function]

<R>

Development Tool	Registers Used for Return Value	Destructed Registers
Renesas CA78K0R compiler	None	None
Renesas CC-RL compiler	None	AX, BC, DE, HL
LLVM compiler	None	AX, BC, DE, HL

[Arguments]

None

[Return values]

None

EEL_GetSpace

[Function]

This obtains the free EEL block space.

[Format]

<C language>

Renesas CA78K0R compiler

```
eel_status_t   __far EEL_GetSpace(__near eel_u16* space_pu16)
```

<R> Renesas CC-RL compiler

```
eel_status_t   __far EEL_GetSpace(__near eel_u16* space_pu16)
```

<R> LLVM compiler

```
eel_status_t   __far EEL_GetSpace(__near eel_u16* space_pu16)
                                                    __attribute__ ((section ("EEL_CODE")))
```

<Assembler>

```
CALL !EEL_GetSpace or CALL !!EEL_GetSpace
```

Remark Call this function with "!" if you are placing the EEL at 00000H to 0FFFFH or with "!!" otherwise.

[Pre-conditions]

The FDL_Init, FDL_Open, EEL_Init, EEL_Open, and EEL_Execute (EEL_CMD_STARTUP command) functions must have finished normally.

[Post-conditions]

None

[Cautions]

1. When the EEL pool has been exhausted, 0 is always returned to indicate that there is no free space.
2. When an error value is returned, the free space information remains unchanged.

[Register status after calling this function]

<R>

Development Tool	Registers Used for Return Value	Destructed Registers
Renesas CA78K0R compiler	C (General-purpose register)	AX
Renesas CC-RL compiler	A (General-purpose register)	X, C, HL
LLVM compiler	A (General-purpose register)	X, C, HL

<R> [Arguments]

space_pu16: The address at which the free space information of the current active block is input.

Contents of Argument Settings

Development Tool	C Language	Assembly Language (Register)
RENESAS CA78K0R compiler	__near eel_u16* space_pu16	AX(0 to 15) The start address of the argument (16 bits)
RENESAS CC-RL compiler	__near eel_u16* space_pu16	AX(0 to 15) The start address of the argument (16 bits)
LLVM compiler	__near eel_u16* space_pu16	AX(0 to 15) The start address of the argument (16 bits)

[Return values]

Type	Symbol Definition	Description
eel_status_t	EEL_OK	Normal end
	EEL_ERR_INITIALIZATION	EEL_Init has not been executed.
	EEL_ERR_ACCESS_LOCKED	The EEL_CMD_STARTUP command has not finished normally.
	EEL_ERR_REJECTED	A command is being executed.

EEL_GetVersionString

[Function]

This obtains the version information of the EEL.

[Format]

<C language>

Renesas CA78K0R compiler

```
__far eel_u08*  __far EEL_GetVersionString(void)
```

<R> Renesas CC-RL compiler

```
__far eel_u08*  __far EEL_GetVersionString(void)
```

<R> LLVM compiler

```
__far eel_u08*  __far EEL_GetVersionString(void) __attribute__ ((section ("EEL_CODE")))
```

<Assembler>

```
CALL !EEL_GetVersionString or CALL !!EEL_GetVersionString
```

Remark Call this function with "!" if you are placing the EEL at 00000H to 0FFFFH or with "!!" otherwise.

[Pre-conditions]

None

[Post-conditions]

None

[Cautions]

None

[Register status after calling this function]

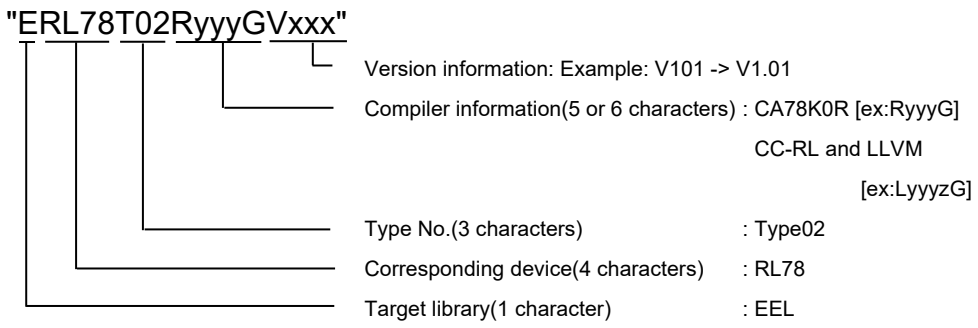
<R>

Development Tool	Registers Used for Return Value	Destructed Registers
Renesas CA78K0R compiler	BC (0 to 15), DE (16 to 31)	None
Renesas CC-RL compiler	DE (0 to 15), A (16 to 23)	None
LLVM compiler	DE (0 to 15), A (16 to 23)	None

[Arguments]

None

[Return values]

Type	Description
<R> eel_u08* (far)	The address at which the version information of the EEL is input (a 24-bit address area) The version information of the EEL consists of ASCII characters. Example: For EEPROM emulation library Pack02 V1.01 (ASCII code) 

Chapter 6 Software Resources and Processing Time

6. 1 Processing Time

This section describes the EEL processing time.

Figure 6-1 shows the concept of EEL function response time and total processing time. The total processing time in the figure is the case of successful completion and does not include the processing time in the case of abnormal end (such as incorrect input data or error). Delay time due to execution of the EEL_Handler function is not included either. If the EEL_Handler function calling interval time is extended, the maximum total processing time may be exceeded.

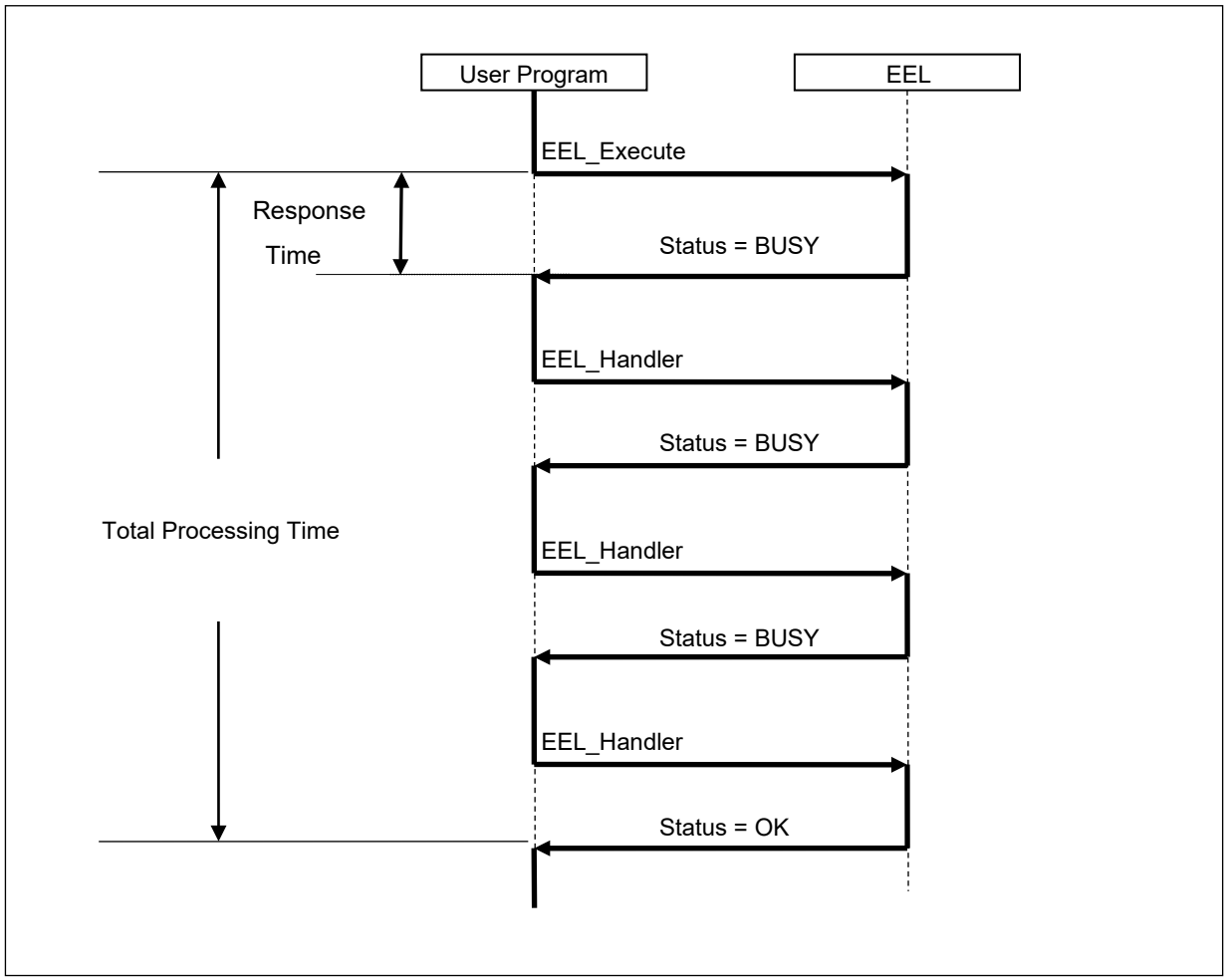


Figure 6-1 Overview of Processing Time

Table 6-1 EEL Function Response Time of EEPROM Emulation Library Pack02

Functions	MAX time (Full Speed Mode)	MAX time (Wide Voltage Mode)
FDL_Init	1199 / fcpu µs	1199 / fcpu µs
FDL_Open	27 / fcpu + 14 µs	27 / fcpu + 14 µs
FDL_Close	836 / fcpu + 444 µs	791 / fcpu + 969 µs
EEL_Init	3268 / fcpu µs	3268 / fcpu µs
EEL_Open	14 / fcpu µs	14 / fcpu µs
EEL_Close	17 / fcpu µs	17 / fcpu µs
EEL_GetSpace	47 / fcpu µs	47 / fcpu µs
EEL_GetVersionString	14 / fcpu µs	14 / fcpu µs
EEL_Execute	320 / fcpu µs	320 / fcpu µs
EEL_Handler	4582 / fcpu µs	4582 / fcpu µs

Table 6-2 Total Processing Time of EEPROM Emulation Library Pack 02

Functions	MAX time (Full Speed Mode)	MAX time (Wide Voltage Mode)
EEL_Execute / EEL_Handler		
EEL_CMD_STARTUP	(280530 + 235 * Block Num) / fcpu + 1612 µs	(277604 + 235 * Block Num) / fcpu + 8798 µs
EEL_CMD_FORMAT	(67102 + 288981 * Block Num) / fcpu + (266627 * Block Num) µs	(67102 + 256218 * Block Num) / fcpu + (303359 * Block Num) µs
EEL_CMD_REFRESH 1. Finished normally	5163828 / fcpu + 774424 µs	5072479 / fcpu + 1421917 µs
EEL_CMD_REFRESH 2. REFRESH processing failed until Block Num - 1	(1554000 + 7538406 * (Block Num - 1)) / fcpu + (1548404 * (Block Num - 1)) µs	(1554000 + 7355752 * (Block Num - 1)) / fcpu + (2842866 * (Block Num - 1)) µs
EEL_CMD_VERIFY	30869 / fcpu + 4126 µs	19605 / fcpu + 29754 µs
EEL_CMD_WRITE	303387 / fcpu + 111858 µs	289240 / fcpu + 253342 µs
EEL_CMD_READ	5102 / fcpu µs	5102 / fcpu µs
EEL_CMD_SHUTDOWN	219 / fcpu µs	219 / fcpu µs

Remarks fcpu: CPU/peripheral hardware clock frequency (for example, at 20 MHz, fcpu = 20)

Block Num: Number of EEPROM emulation blocks

6. 2 Software Resources

In the EEL, program areas corresponding to parts of the library to be used, RAM areas for variables to be used in the library, and RAM areas for work area (self-RAM) are used to assign an appropriate program to the user area. Also, since the FDL will be used, the EEL must have a separate area for use by the FDL.

<R> The self-RAM area used for EEL varies depending on the microcontroller, and the user RAM may be used in some devices. In such a device, the user needs to allocate the self-RAM area to the user RAM; be sure to allocate the self-RAM area at linkage.

CA78K0R compiler: The area can be specified in the link directive file.

CC-RL compiler: The area can be specified without allocating a section.

LLVM compiler: The area can be specified in the linker script file.

For the method of specifying the area in the link directive file, refer to the section "Defining the On-Chip RAM Area" in the release note.

Tables 6-3 and 6-4 list required software resources Notes 1, 2. Figures 6-2 and 6-3 show the images of allocating software resources to the RAM.

Table 6-3 Software Resources Used by EEPROM Emulation Library Pack02 Ver. 1.01

<R>

Item	Size (Byte)		Area Used by EEPROM Emulation Library Pack02 Note 1
	CA78K0R	CC-RL LLVM	
Self RAM Note 2	0 to 384 Note 2		Use of the self-RAM area by RL78 Family EEPROM Emulation Library Pack02 differs with the device. For details, refer to "RL78 Family Self RAM list of Flash Self Programming Library (R20UT2944)".
Stack	80	64	Can be allocated to a RAM area other than the self-RAM and the area from FFE20H to FFEFFH.
Data buffer Note 3	1 to 255		
Request structure	5		
SADDR RAM work area	SADDR : 3		Can be allocated to a short-addressing RAM area.
Library size	3400		Can be allocated to a program area other than the self RAM and the area from FFE20H to FFEFFH (ROM is recommended)
Data table	3 to 66		
Fixed-parameter area (default)	14		
EEL Blocks	3,072 or more (at least 3blocks)		Only data flash memory can be used.

Notes 1. For devices not shown in the RL78 Family Self RAM list of Flash Self Programming Library (R20UT2944), contact your Renesas sales agency.

2. An area used as the working area by the EEL is called self-RAM in this manual and the release note. The self-RAM requires no user setting because it is an area that is not mapped and automatically used at execution of the EEL (previous data is discarded). When the EEL is not used, the self-RAM can be used as a normal RAM space.
3. The data buffer is used as the working area for EEL internal processing or the area where the data to be set is allocated in the EEL_Execute function. The required size depends on the function to be used.

Table 6-4 Data Buffer Size Used by EEL Functions

Function Name	Byte	Function Name	Byte
FDL_Init	0	EEL_Close	0
FDL_Open	0	EEL_Execute Note	0 to 255
FDL_Close	0	EEL_Handler Note	0 to 255
EEL_Init	0	EEL_GetSpace	2
EEL_Open	0	EEL_GetVersionString	0

Note: An additional 5 bytes area is used by the request structure.

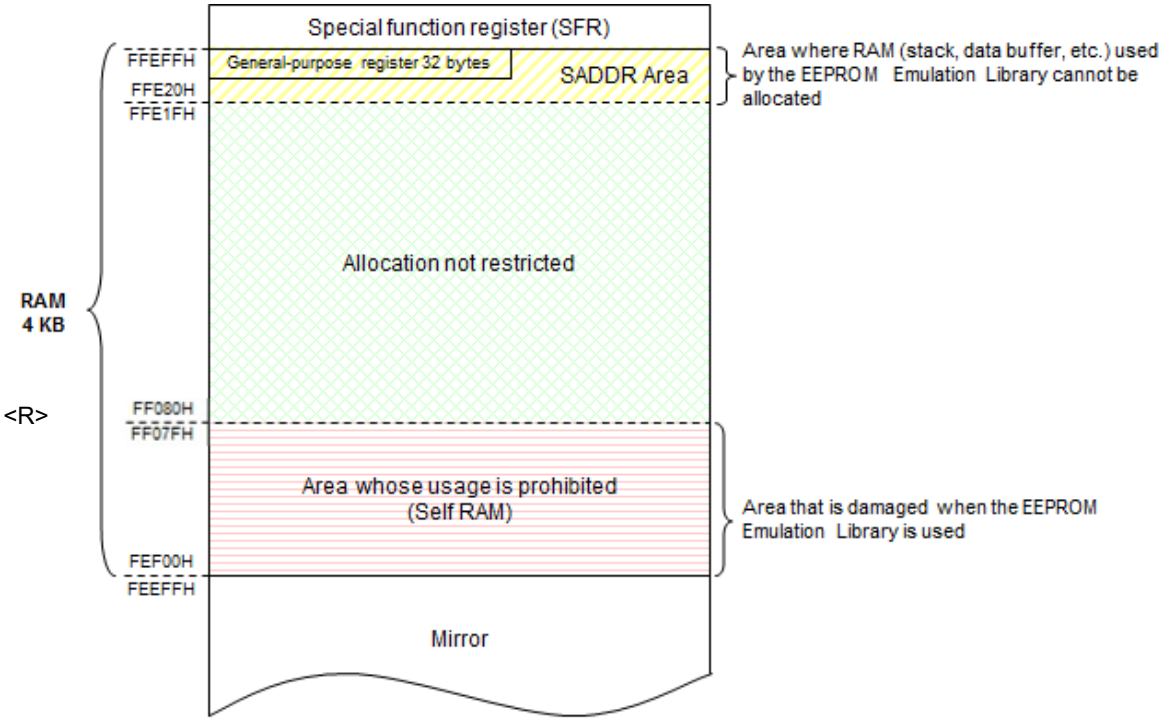


Figure 6-2 Example 1 of RAM Allocation with Self-RAM
(RL78/G13: product with 4 KB RAM and 64 KB ROM)

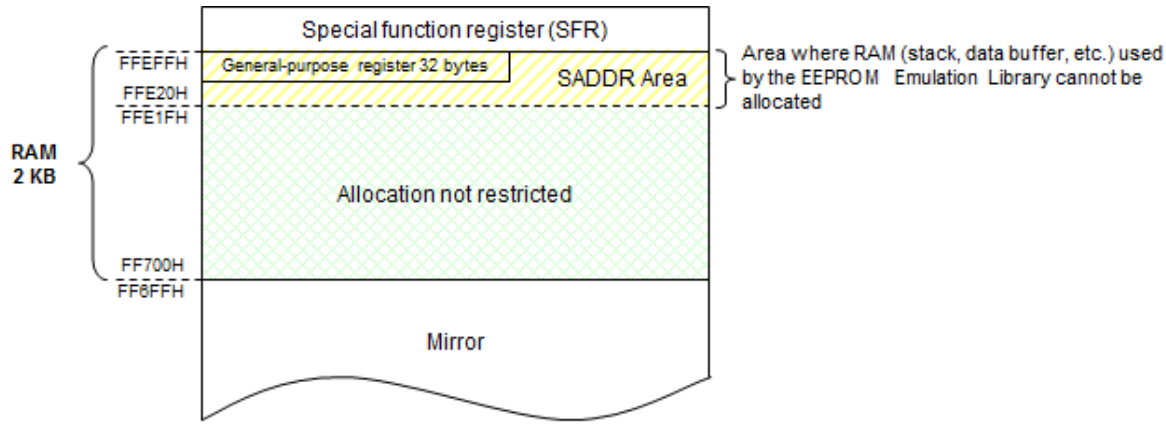


Figure 6-3 Example 2 of RAM Allocation without Self-RAM
(RL78/G13: product with 2 KB RAM and 32 KB ROM)

6. 2. 1 Sections

Functions, constants, and variables to be used are allocated to specified sections in the EEL and FDL.

The following table lists sections defined by the EEL and FDL.

Table 6-5 Sections Used in the EEL/FDL

Section name	Description
FDL_CODE	FDL's code section that contains FDL programs.
FDL_SDAT	FDL's variable data section that contains variable data used in the FDL. Place this section in the short addressing RAM area.
FDL_CNST	FDL's constant data section that contains constant data used in the FDL.
EEL_CODE	EEL's code section that contains EEL programs.
EEL_SDAT	EEL's variable data section that contains variable data used in the EEL. Place this section in the short addressing RAM area.
EEL_CNST	EEL's constant data section that contains constant data used in the EEL.

Appendix A Revision History

A.1 Major Revisions in This Edition

Page	Description	Classification
Throughout the document		
–	The user's manual of the EEPROM Emulation library for CC-RL was integrated into this manual.	(d)
–	Added information about the EEPROM Emulation library for the LLVM compiler.	(b)
–	A statement regarding reference to the RL78 Family Self RAM list of the target MCU was added (see HOW TO USE THIS MANUAL).	(e)
–	The English translation was reviewed and corrected.	(a)
–	The term "voltage mode" was changed to "flash memory programming mode" for consistency of terminology.	(d)
Chapter 1 Overview		
p. 1	The contents of section "1.2 Target Devices", were moved to the description of the target MCUs in How to Use This Manual. The description in the new section "1.2 Target Compilers", was added.	(c)
Chapter 2 EEPROM Emulation		
p. 3	The statements in Table 2-1 were added and those only relevant to EEL Pack 01 were deleted.	(c)
p. 4	The title of Table 2-1 was added and the descriptions on EEL Pack 01 were deleted.	(c)
p. 5	A title was added for Table 2-2.	(c)
Chapter 3 EEL Functional Specifications		
p. 13	Statements regarding urgent processing were added to Note 1.	(c)
Chapter 4 Using EEPROM Emulation		
p. 19	Item 1 in Table 4-1 was made more precise.	(c)
p. 21	A cautionary note on using the CC-RL compiler was added to No. 26 in Table 4-1.	(c)
Chapter 5 User Interface		
p. 27	Statements on the assembly language code for the CA78K0R compiler, the C and assembly languages of the CC-RL compiler and the C language code for the LLVM compiler were added to "Definition of the request structure (eel_request_t)".	(c)
p. 29	Statements on differences between the CA78K0R and CC-RL compilers were added.	(c)
p. 29	Added description of LLVM compiler to Remarks 4	(c)
p. 31 to p. 49	Statements on formats of the C language, register status after calling functions, and arguments in the case of the CC-RL and LLVM compilers were added.	(c)
Chapter 6 Software Resources and Processing Time		
p. 48	Statements on differences between the CA78K0R and CC-RL compilers were added. Statements of the sizes of individual items for the CC-RL compiler were added and that of the size of the self RAM was changed in Table 6-3.	(c)
p. 49	The address range of the area where usage is prohibited was corrected in Figure 6-2 to reflect the change to the self-RAM area.	(c)

Remark: Symbols under "Classification" in the above table are used to classify revisions as follows.

(a): Correction of errors, (b): addition or change to specifications, (c): addition or change to descriptions or notes, (d): addition or change to packages, part numbers, or management divisions, (e): addition or change to related documents

A.2 Revision History of Previous Editions

The following shows the revision history of the previous editions. The Chapter column indicates the chapter in the edition.

Rev. No.	Description	Chapter
1.01	Statements of the target devices were deleted.	Throughout the document
	References to the list of target MCUs were added.	
	The description in section 2.2, Outline of Function, was changed.	Chapter 2 EEPROM Emulation
	The description in table 2-1 was modified.	
	The description in figure 2-4 was changed.	
	The description in table 2-2 was changed.	
	The description of note 2 for figure 3-1 was changed.	Chapter 3 EEL Functional Specifications
	The description of section 3.5 was changed.	
	The description of execution commands was changed.	Chapter 5 User Interface
	The description of EEL_Handler was changed.	
	Items in table 6-1 were changed.	Chapter 6 Software Resources and Processing Time
	Items in table 6-2 were changed.	
	The heading and items in table 6-3 were changed.	
	The description of the self-RAM area in table 6-3 was changed.	
	The inquiry about device specifications in note 1 for table 6-3 was changed.	
	The description of note 2 in table 6-3 was removed.	
	The description of note 4 in table 6-3 was removed.	

Rev. No.	Description	Chapter
1.00	Newly created	Throughout the document

RL78 Family User’s Manual: EEPROM Emulation Library Pack 02

Publication Date: Rev.1.10 Apr 22, 2024

Published by: Renesas Electronics Corporation

RL78 Family
EEPROM Emulation Library Pack02

Renesas Electronics Corporation

R01US0068EJ0110