User's Manual

16

# RL78 Family

EEPROM Emulation Library Pack01

Japanese Release

ZIP file name : JP_R_EEL_RL78_P01_Vx.xx_x_E

## 16-Bit Single-Chip Microcontrollers

Target Devices

RL78/D1A

RL78/F12

RL78/G13

RL78/G14

RL78/G1A

RL78/I1A

RL78/L13



Renesas Electronics
www.renesas.com

Rev.1.02 Aug 2013

# 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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of 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. 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 should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The recommended 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; and industrial robots etc.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor 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 damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may 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 should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

NOTES FOR CMOS DEVICES

(1) 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 VIL (MAX) and VIH (MIN) due to noise, etc., 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 VIL (MAX) and VIH (MIN).

(2) HANDLING OF UNUSED INPUT PINS: Unconnected CMOS device inputs can be cause of malfunction. If an input pin is unconnected, it is possible that an internal input level may be generated due to noise, etc., causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using pull-up or pull-down circuitry. Each unused pin should be connected to VDD or GND via a resistor if there is a possibility that it will be an output pin. All handling related to unused pins must be judged separately for each device and according to related specifications governing the device.

(3) PRECAUTION AGAINST ESD: A strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it when it has occurred. Environmental control must be adequate. When it is dry, a humidifier should be used. It is recommended to avoid using insulators that 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 should be grounded. The operator should be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with mounted semiconductor devices.

(4) STATUS BEFORE INITIALIZATION: Power-on does not necessarily define the initial status of a MOS device. Immediately after the power source is turned ON, devices with reset functions have not yet been initialized. Hence, power-on does not guarantee output pin levels, I/O settings or contents of registers. A device is not initialized until the reset signal is received. A reset operation must be executed immediately after power-on for devices with reset functions.

(5) POWER ON/OFF SEQUENCE: In the case of a device that uses different power supplies for the internal operation and external interface, as a rule, switch on the external power supply after switching on the internal power supply. When switching the power supply off, as a rule, switch off the external power supply and then the internal power supply. Use of the reverse power on/off sequences may result in the application of an overvoltage to the internal elements of the device, causing malfunction and degradation of internal elements due to the passage of an abnormal current. The correct power on/off sequence must be judged separately for each device and according to related specifications governing the device.

(6) INPUT OF SIGNAL DURING POWER OFF STATE : Do not input signals or an I/O pull-up power supply while the device is not powered. 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. Input of signals during the power off state must be judged separately for each device and according to related specifications governing the device.

# HOW TO USE THIS MANUAL

**Readers**

This manual is intended for user engineers who wish to understand the functions of the RL78 microcontrollers EEPROM Emulation Library Pack 01 and design and develop application systems and programs for these devices.
The target products are as follows.

RL78/D1A, RL78/F12, RL78/G13, RL78/G14, RL78/G1A, RL78/I1A, RL78/L13

**Purpose**

This manual is intended to give users an understanding of the methods (described in the **Organization** below) for using data flash memory library to rewrite the flash data memories.

**Organization**

The RL78 EEPROM Emulation Library Pack 01 user's manual is separated into the following parts:

- Overview of EEPROM Emulation
- Using EEPROM 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 functions:
  → Read this manual in the order of the **CONTENTS**.
- To know details of the RL78 Microcontroller instructions:
  → Refer to **CHAPTER 3 EEPROM EMULATION FUNCTION**.

The mark <R> shows major revised points.

**Conventions**

| | | |
|---|---|---|
| Data significance: | Higher digits on the left and lower digits on the right | |
| Active low representations: | $\overline{\times\times\times}$ (overscore over pin and signal name) | |
| **Note**: | Footnote for item marked with **Note** in the text | |
| **Caution**: | Information requiring particular attention | |
| **Remark**: | Supplementary information | |
| Numerical representations: | Binary | ···×××× or ××××B |
| | Decimal | ···×××× |
| | Hexadecimal | ···××××H |

# CONTENTS

# CHAPTER 1 OVERVIEW OF EEPROM EMULATION

## 1. 1 Basic Specifications of EEPROM Emulation

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.

By calling the user access function processing (functions) provided by the EEPROM emulation library from a user-created program, use is possible without the awareness of data flash memory operations.

For the EEPROM emulation library Pack01, a one-bye identifier (data ID: 1 to 255) 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. (Up to 255 data items assigned on an identifier basis can be handled.)

Note that four or more continuous block area of data flash memory are used to store the data. These blocks are called EEPROM emulation 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.

Figure 1-1 shows the relationship between the EEPROM emulation library and data flash library, Figures 1-2 and 1-3 show a memory map and data structure example, and Figures 1-4, 1-5, and 1-6 show block usage method and transition examples. Table 1-1 shows each item to be specified for EEPROM emulation and the range of the item.

**Figure 1-1. Relationship between EEPROM Emulation Library and Data Flash Library**

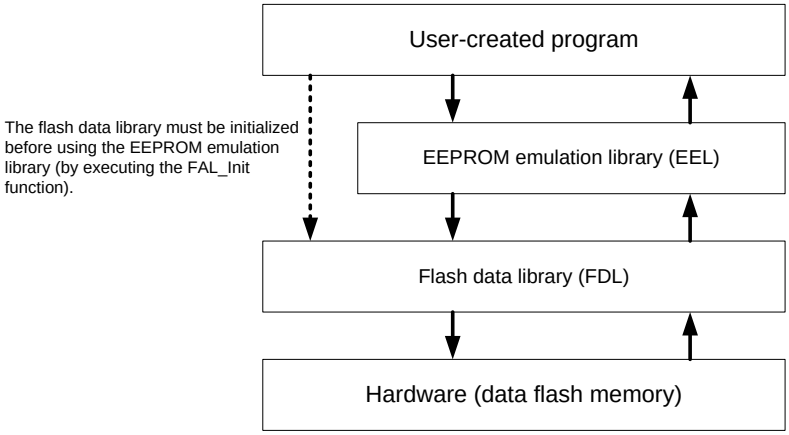



**Figure 1-2. Example of Memory Map**

- The following shows an example for the R5F100 where the user-created program, data flash library, and EEPROM emulation library are placed in the code flash memory, the EEPROM emulation blocks are specified for the data flash memory, and the defined user data (user data A, user data B, and user data C) is written in order to use the data flash memory as EEPROM emulation blocks.

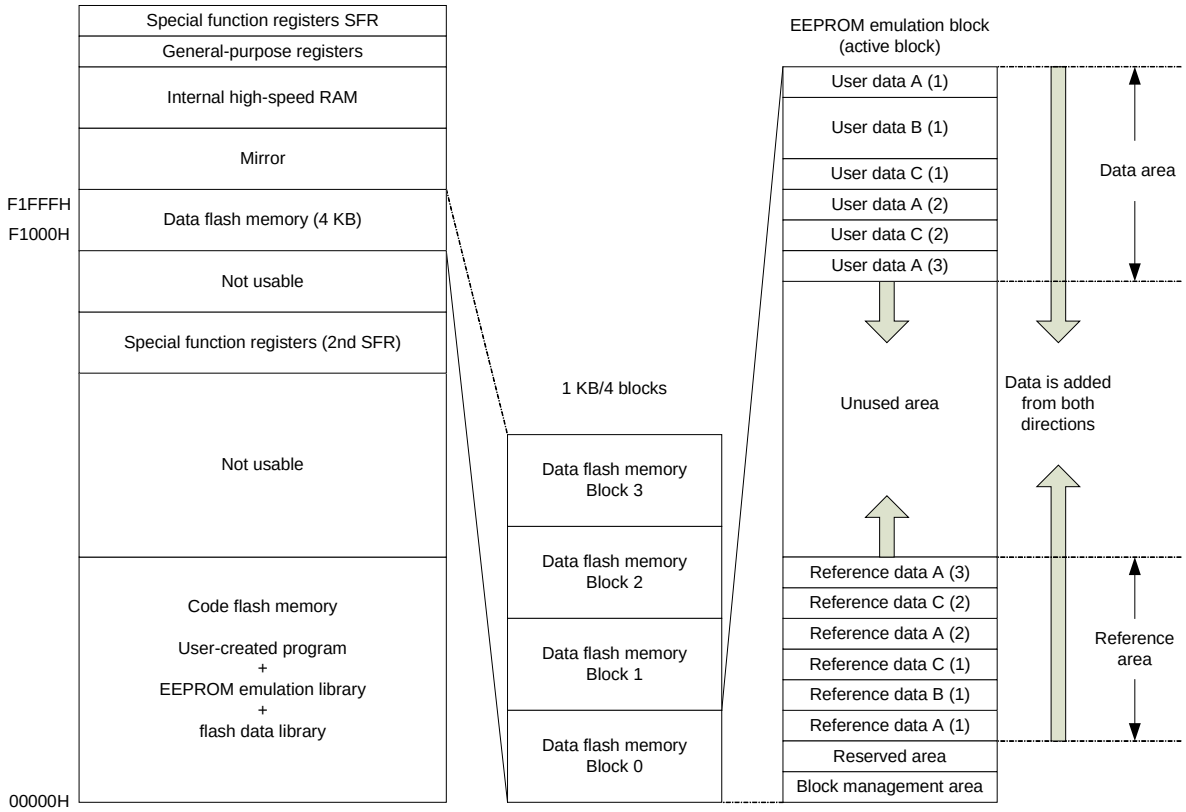


**Figure 1-3. EEPROM Emulation Block Details**

- In this data structure example, user data of various defined sizes (user data A, user data B, and user data C) are written in a specified sequence (write sequence: user data A → user data B → user data A → user data C). User data is written from the higher address and management data is written from the lower address, and the last written user data becomes valid.

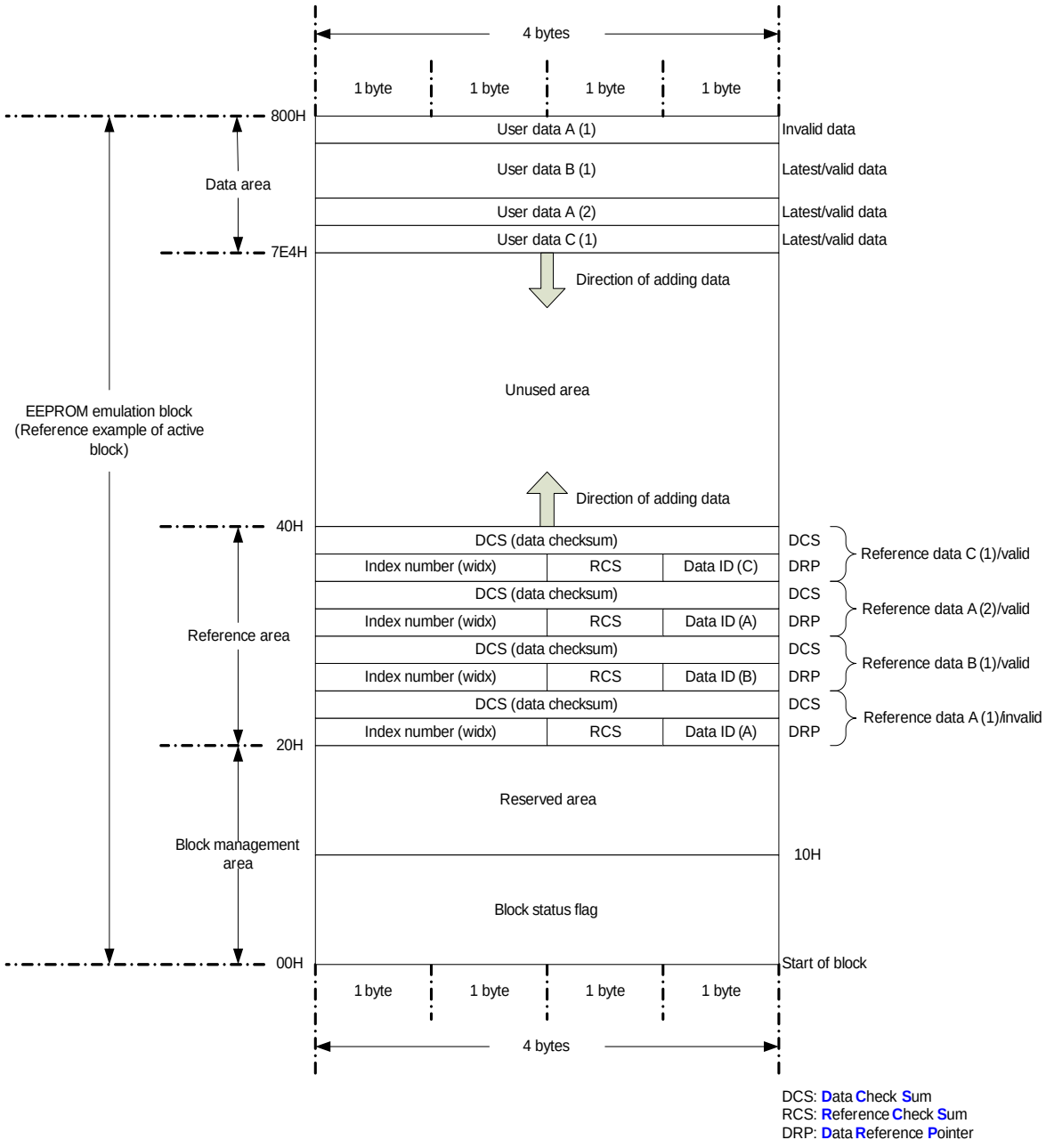


**Figure 1-4. Example of Expanding Active EEPROM Emulation Block**

- When writing specified data, if there is not enough free space in the active block being used to write the data, the number of active blocks is increased and the specified data is written to the new active block.

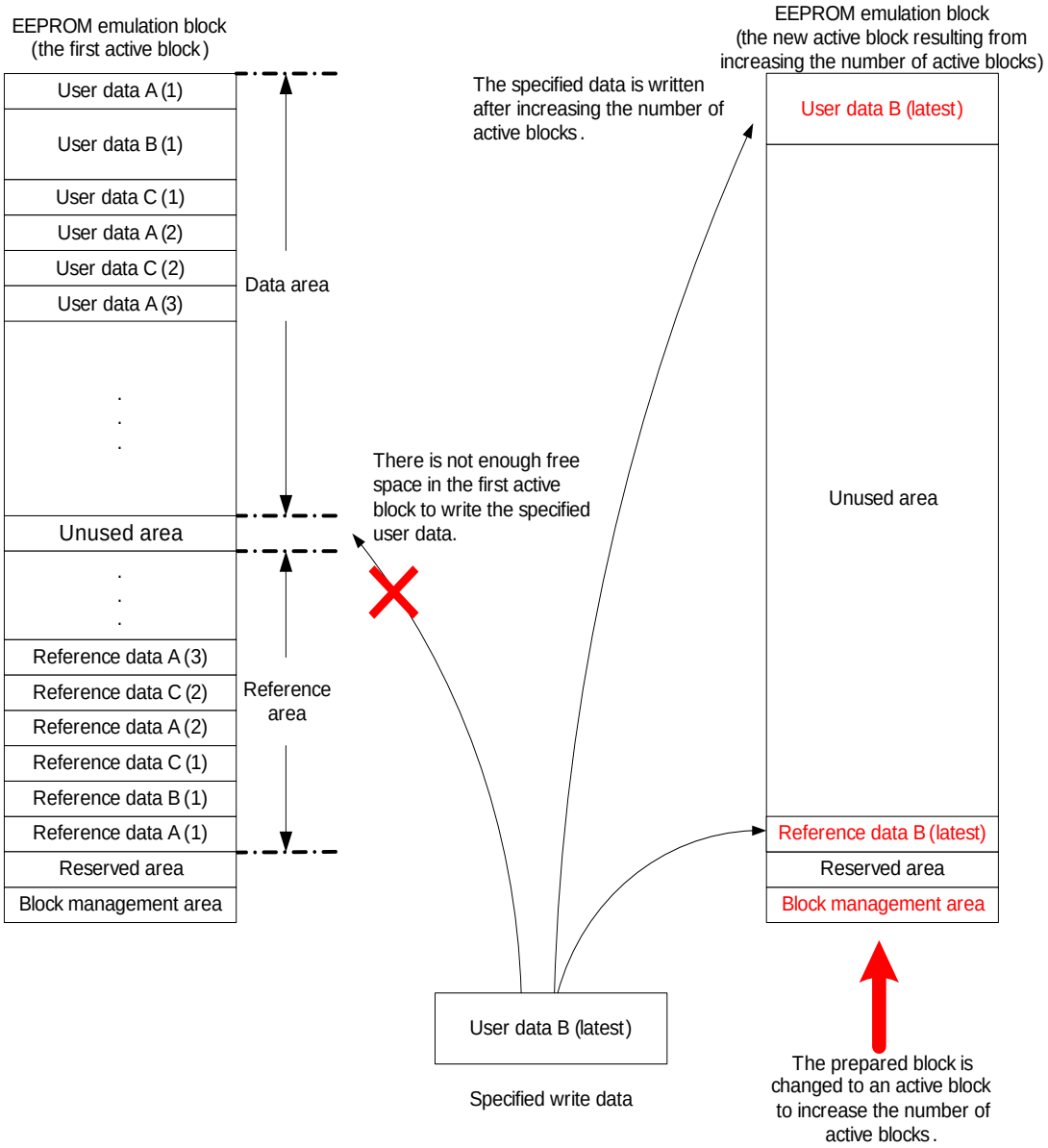
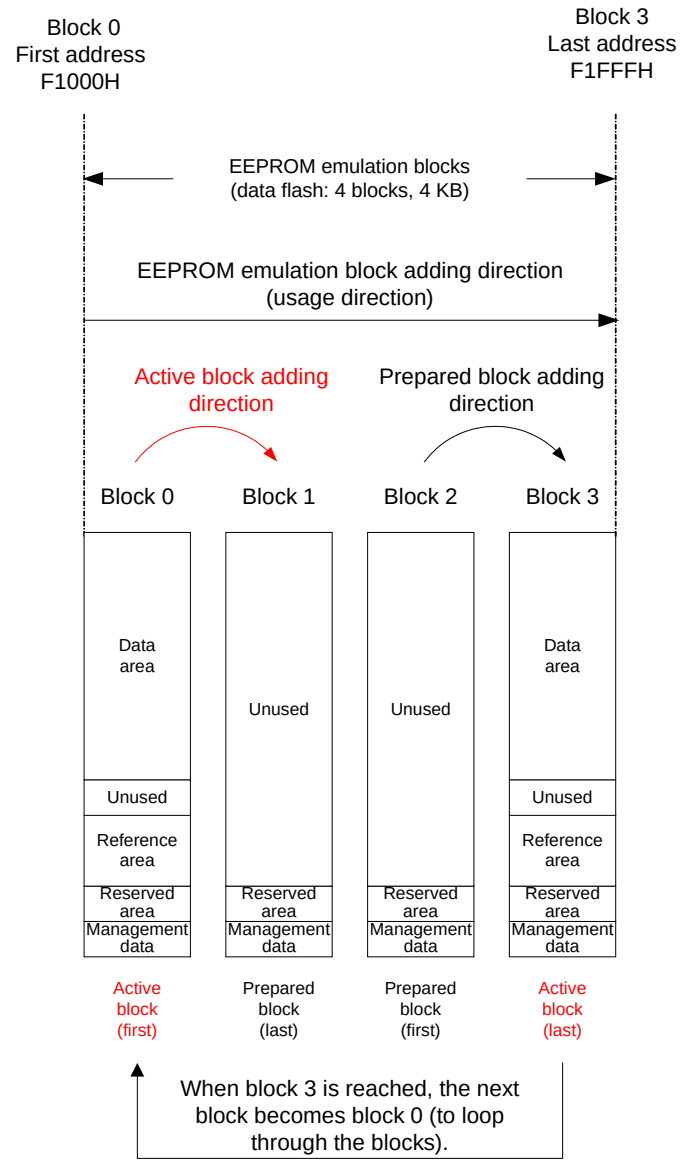


**Figure 1-5. EEPROM Emulation Block Usage Method (Four Blocks)**

- For EEPROM emulation, blocks 0 to 3 of the data flash memory are added in order as active or prepared blocks, and then, when the last block (block 3) is reached, the first block (block 0) is specified as the next block to loop through the blocks.




Active block: The currently used block
Prepared block: A block that has been prepared and is ready for use

**Figure 1-6 (A). EEPROM Emulation Block Transition Example**

- The following shows an example of EEPROM emulation blocks to which nothing was written after initializing them.

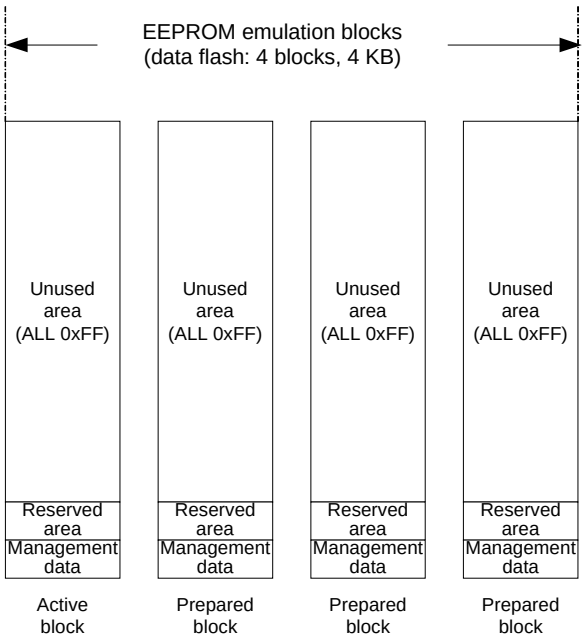



**Figure 1-6 (B). EEPROM Emulation Block Transition Example**

- If only writing processing is executed continuously, the number of active blocks is increased until a certain number of blocks (the number of active blocks such that there are two or fewer prepared blocks remaining) is reached.

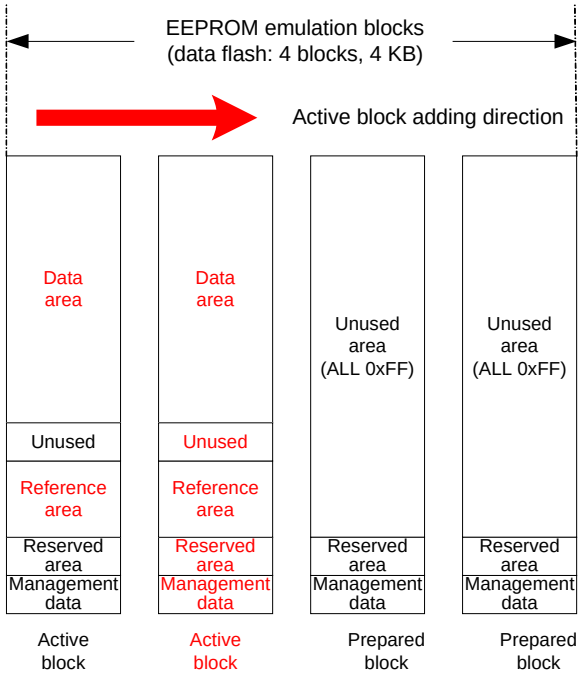


**Figure 1-6 (C). EEPROM Emulation Block Transition Example**

- If only writing processing is executed continuously during EEPROM emulation, and the number of active blocks is increased when two or fewer prepared blocks remain, the valid data in the oldest active block is copied to the latest active block, and the oldest active block is erased.

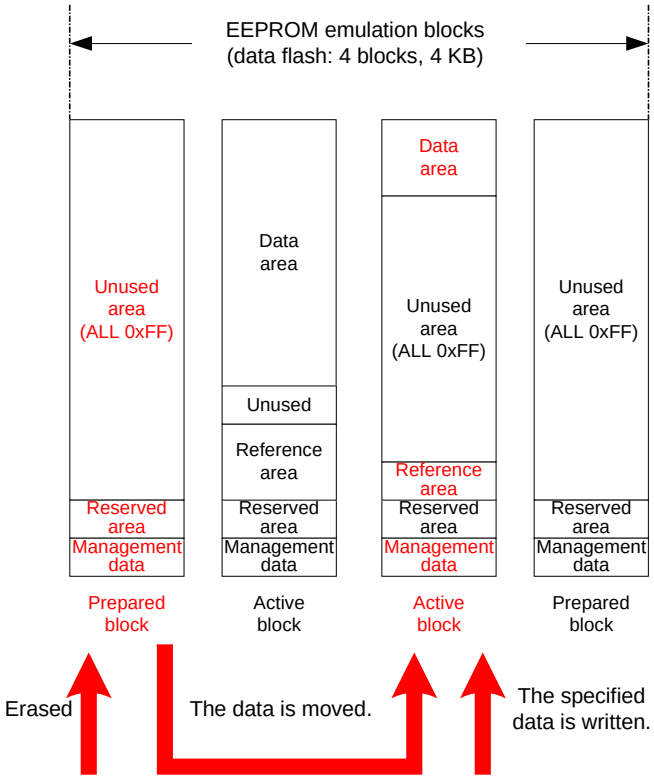



**Table 1-1. Settings of Written Data and Usable Ranges**

| Item | Range | Remark |
|---|---|---|
| User data length | 1 to 255 | – |
| Amount of stored user data Note 1 | 1 to 255 | Number of data types |
| Data ID range | 1 to 255 | – |
| Number of EEPROM emulation blocks Note 2 | 4 to 255 | – |
| <R> Recommended user data size Note 1 | 980 × total number of blocks × 1/4 – 980/2 bytes | This also includes the management reference data provided during writing. |

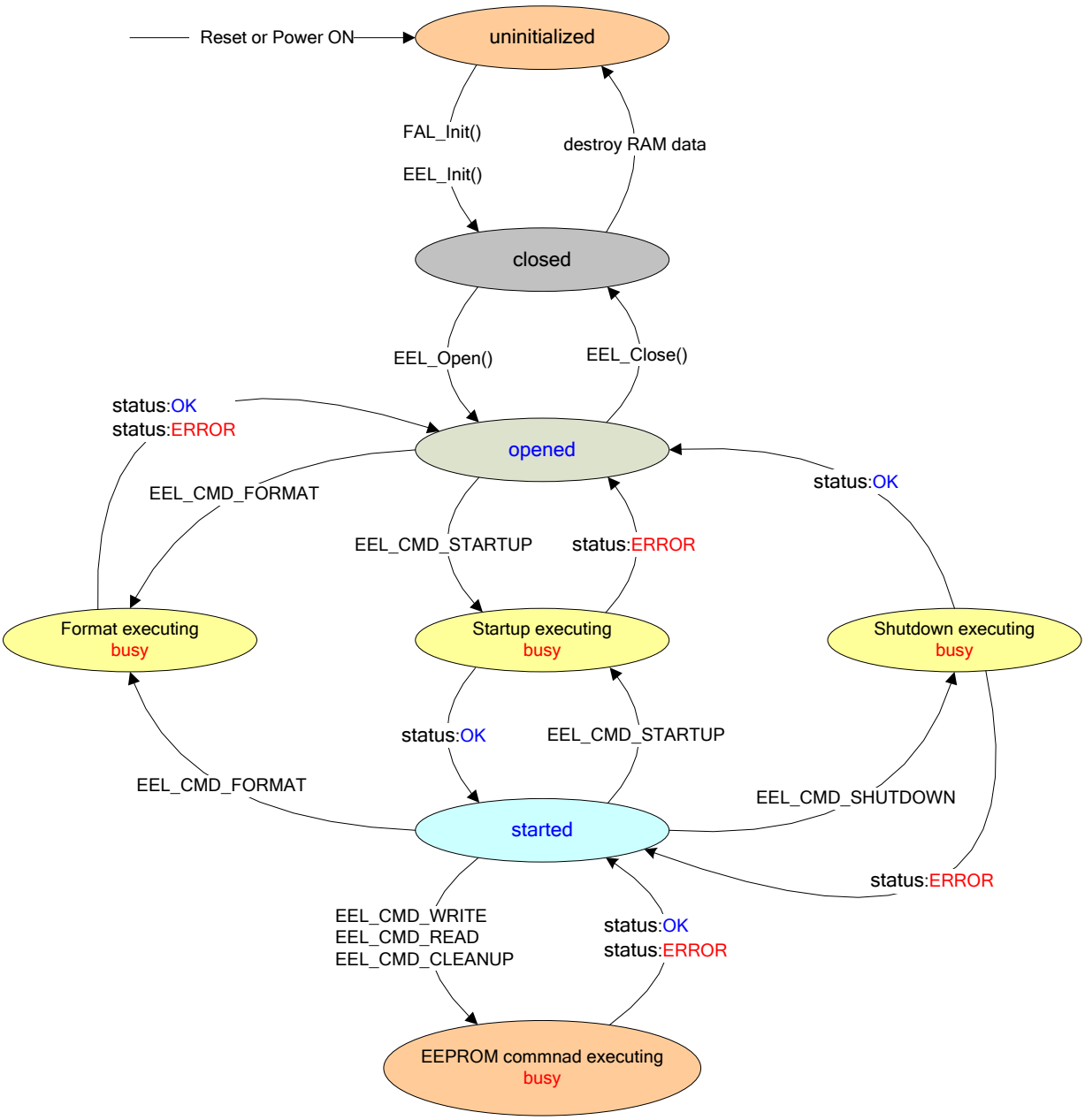
**Notes 1.** The total size of the user data must be such that it is possible to write all the data into within two EEPROM emulation blocks. Therefore, the range used for the number of stored user data items differs depending on the size of the stored user data. It is also necessary to consider the size 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 **1.2.4 Number of stored user data items and total user data size**.

**2.** EEPROM emulation blocks cannot be set more than maximum number of blocks of on-board data flash memory.

# 1. 2 EEPROM Emulation Operation Flow

To use EEPROM emulation from a user-created program, it is necessary to initialize the EEPROM emulation library and execute functions that perform operations such as reading and writing on EEPROM emulation blocks. Figure 1-7 shows the overall status transitions, and Figure 1-8 shows an operation flow for using basic features. When using EEPROM emulation, incorporate EEPROM emulation into user-created programs by following this flow.

**Figure 1-7. EEPROM Emulation Status Transitions**



[Overview of status transitions]

To use EEPROM emulation library 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 status after turning the power on or resetting. The system also transitions to this status after executing flash self programming library processing.

(2) closed

This is the status in which the data has been initialized (the status in which operations on the data flash memory are stopped) to execute the FAL_Init() and EEL_Init() functions and then EEPROM emulation. To execute flash self programming library, STOP mode, or HALT mode processing after executing EEPROM emulation, execute EEL_Close in the opened status to switch to the closed status.

(3) opened

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

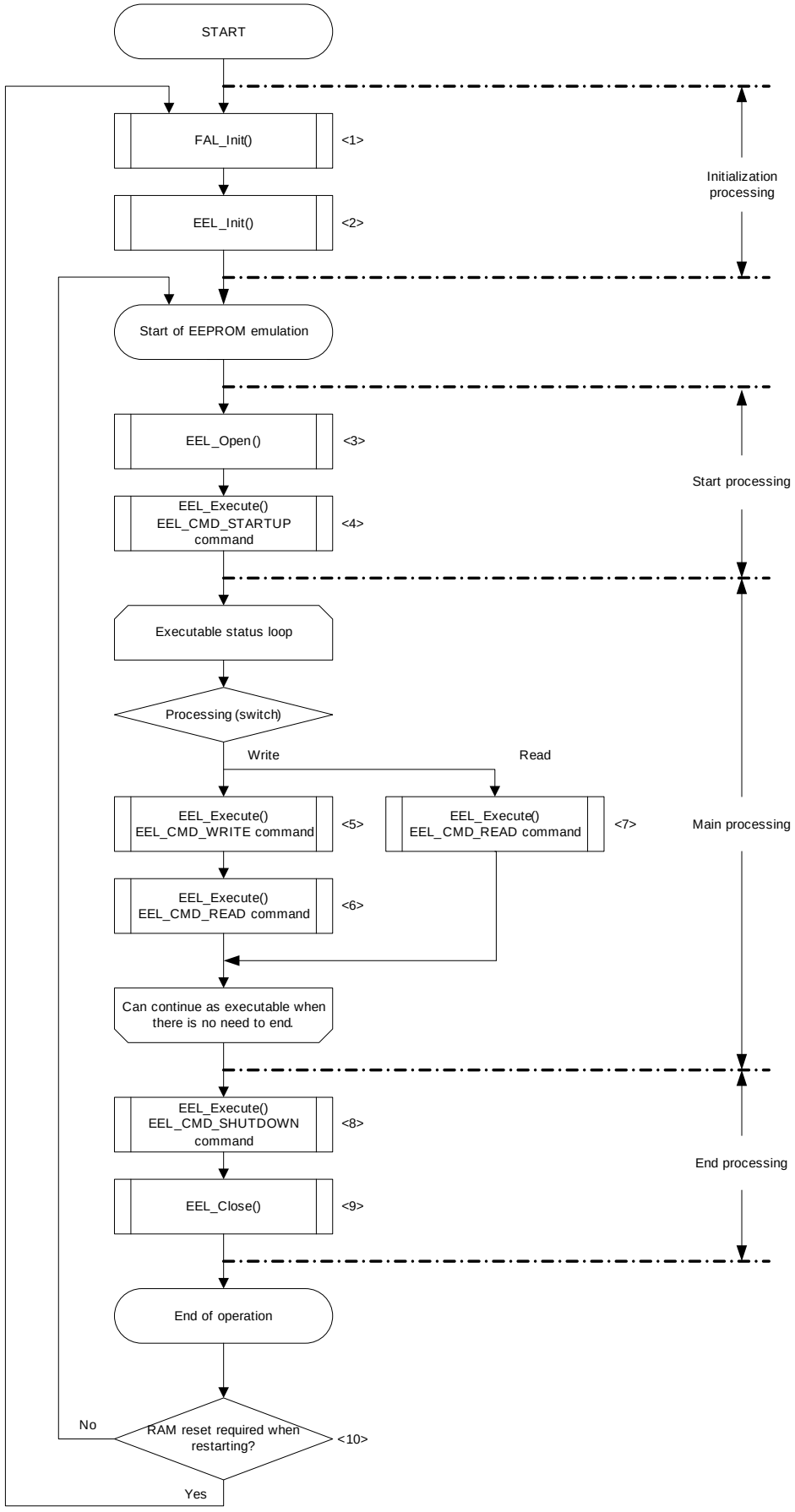
(4) started

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

(5) busy

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

**Figure 1-8. Basic Operation Flow of EEPROM Emulation (When Using Enforced Mode)**



[Overview of basic operation flow]

For EEPROM emulation, the method for executing the EEL_Execute function differs depending on the mode setting. The following three modes are available: the enforced mode, timeout mode, and polling mode. For details about the differences in the execution method for each mode, see EEL_Execute function in **3.2 EEPROM Emulation Library Functions**.

<1> data Flash library initialization processing (FAL_Init)
Because it is necessary to initialize the data flash library parameters (RAM) if using the EEPROM emulation library to access the data flash memory, the FAL_Init function must be executed in advance. If flash self programming library processing was executed after this initialization finished, the initialization processing must be re-executed.

<2> EEPROM emulation library initialization processing (EEL_Init)
Initialize the parameters (RAM) used by the EEPROM emulation library.

<3> EEPROM emulation preparation processing (EEL_Open)
Set the data flash memory to a status (opened) for which control is enabled to execute EEPROM emulation.

<4> EEPROM emulation execution start processing (EEL_Execute: EEL_CMD_STARTUP command)
Set the system to a status (started) in which EEPROM emulation can be executed.

<5> EEPROM emulation data write processing (EEL_Execute: EEL_CMD_WRITE command)
Write the specified data to an EEPROM emulation block.

<6> EEPROM emulation data confirmation processing (EEL_Execute: EEL_CMD_READ command)
Read data, and then make sure that the data was written correctly by comparing it to the original data.

<7> EEPROM emulation data read processing (EEL_Execute: EEL_CMD_READ command)
Read written data.

<8> EEPROM emulation execution stop processing (EEL_Execute: EEL_CMD_SHUTDOWN command)
Set the EEPROM emulation operation to the stopped status (opened).

<9> EEPROM emulation end processing (EEL_Close)
Set the data flash memory to a status (closed) for which control is disabled to stop EEPROM emulation.

<10> Confirmation before re-executing EEPROM emulation
If reinitializing the RAM is necessary before re-executing EEPROM emulation, such as when executing flash self programming after EEPROM emulation stops, use the FAL_Init function to re-execute the initialization processing.

## 1. 2. 1 EEPROM Emulation Blocks

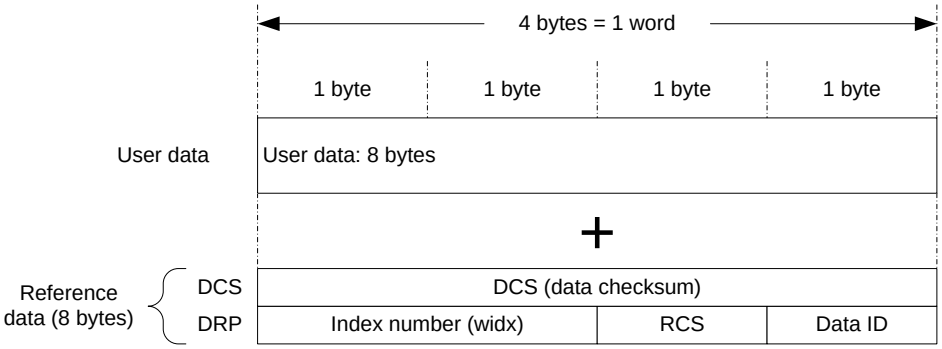
The EEPROM emulation library Pack01 uses four or more block data flash memory as EEPROM emulation blocks.

## 1. 2. 2 Data structure

The data flash memory is written to in word (four-byte) units. Therefore, when the EEPROM emulation library writes to the data flash memory, the data length is always adjusted to word (four-byte) units.

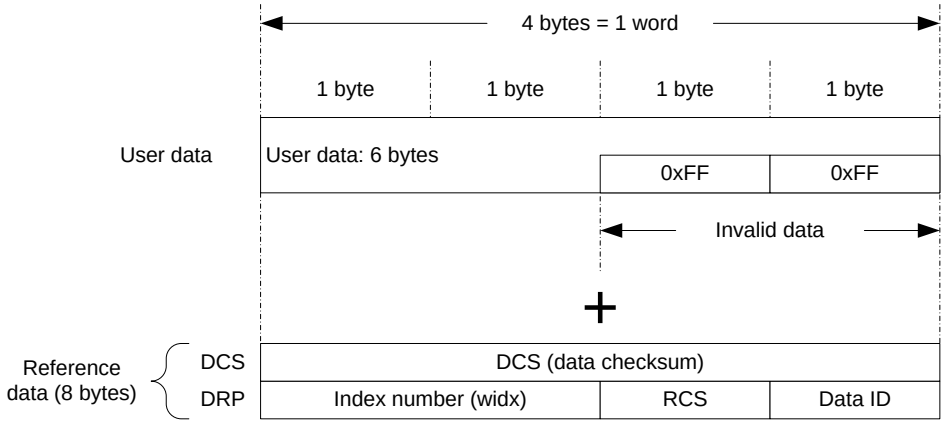
In addition, when writing user data, because the reference data for managing data is also written, when calculating the capacity required for writing, the two words (eight bytes) of management reference data must be added to the size of the user data in words. Figures 1-9 to 1-11 show an example of the data structure used when user data is written to the data flash memory.

**Figure 1-9. Data Length and Data Structure Example 1 (When User Data Is 8 Bytes)**



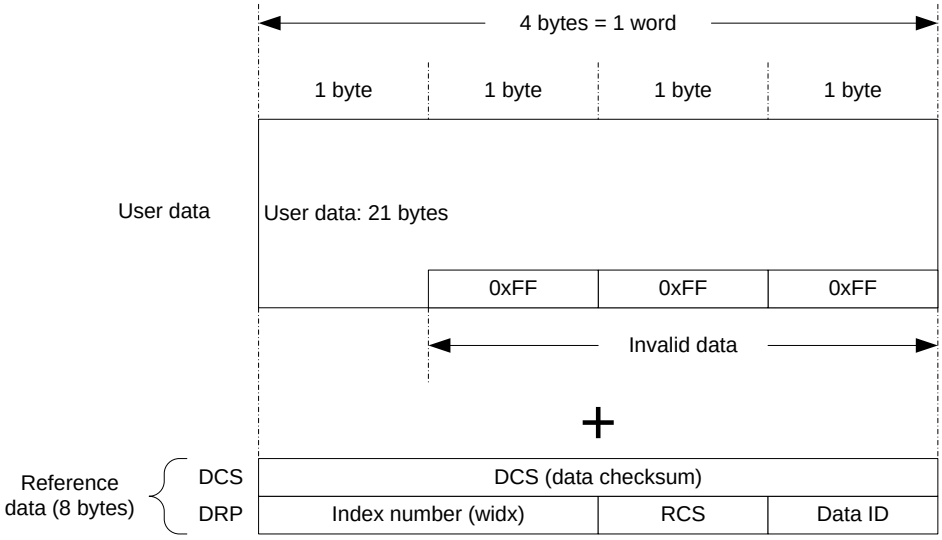

Total size: 4 words (16 bytes) = reference data: 2 words (8 bytes) + [user data (8 bytes)/word size (4 bytes)]
* Change the user data to word units.

**Figure 1-10. Data Length and Data Structure Example 2 (When User Data Is 6 Bytes)**




Total size: 4 words (16 bytes) = reference data: 2 words (8 bytes) + [user data (6 bytes)/word size (4 bytes)]
* Change the user data to word units. (Round any fractions up to the nearest integer.)

**Figure 1-11. Data Length and Data Structure Example 3 (When User Data Is 21 Bytes)**




Total size: 8 words (32 bytes) = reference data: 2 words (8 bytes) + [user data (21 bytes)/word size (4 bytes)]
* Change the user data to word units. (Round any fractions up to the nearest integer.)

(1) DRP

This stands for data reference pointer. This area records the ID of the recorded user data and the reference position.

(2) Index number (widx)

This is the user data index number (reference position).

(3) RCS

This stands for reference checksum. This is the (8-bit) checksum value for the DRP.

(4) Data ID

This is a unique ID for the data being used during EEPROM emulation. User-specified IDs[Note] are registered.

(5) DCS

This stands for data checksum. This is the (32-bit) checksum value for the user data and reference data.

**Note** Before specifying a data ID, it must be registered in the descriptor table.
For details, see **2.3 Initial Values to Be Set by User**.

## 1. 2. 3 Block status flags

The block status flags start at the beginning of the block and include the P flag, A flag, I flag, and X flag, each of which is four bytes, for a total of 16 bytes of data. This data indicates the EEPROM emulation block status, and the combination of flags indicates the block status.

Figure 1-12 shows the placement status of flags, and Table 1-2 shows the combination status of flags.

**Figure 1-12. Block Status Flag Placement Positions**

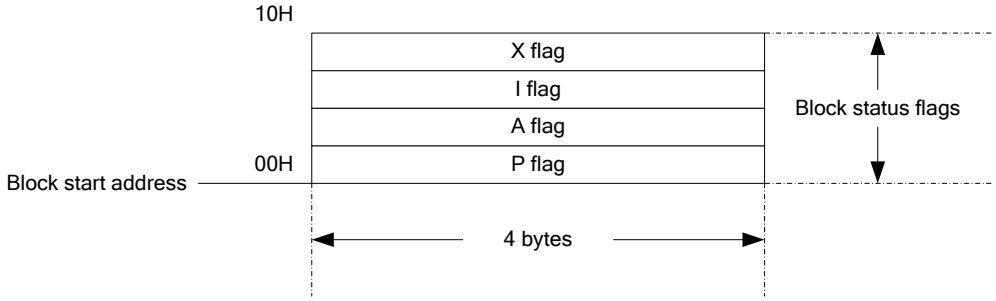



**Table 1-2. Overview of Block Status Flags**

| Block Status Flag | | | | Status | Description |
|---|---|---|---|---|---|
| P Flag | A Flag | I Flag | X Flag | | |
| 55555555H | FFFFFFFFH | FFFFFFFFH | FFFFFFFFH | Prepared | This block is ready to be written. |
| 55555555H | 55555555H | FFFFFFFFH | FFFFFFFFH | Active | This block is being used. |
| 55555555H | 55555555H | 00000000H | FFFFFFFFH | Inactive | Inactive block |
| Data other than the above | | | FFFFFFFFH | | |
| – | – | – | Other than FFFFFFFFH | Use prohibited[Note] | This block cannot be written. |

**Note** A block for which use has been prohibited cannot be reused.

## 1. 2. 4 Number of stored user data items and total user data size

The following restriction applies to the total size of user data that can be used for EEPROM emulation: The total size of the user data must be such that it is possible to write all the data into within two EEPROM emulation blocks. Therefore, the number of stored data items that can be used differs depending on the size of user data that is actually stored. In addition, because it is not possible to place stored user data such that one data item extends across multiple blocks, if the total size necessary to write the user data exceeds one block, it is also necessary to consider the maximum size of an area for which use might not be possible if one block is exceeded.

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, and Figure 1-13 shows the size concepts when the total user data size is more than one blocks.

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

  Size of one block of data flash memory: 1, 024 bytes
  Size required for EEPROM emulation block management: 32 bytes
<R> Free space necessary as termination information (separator): 12 bytes

<R> Maximum usable size of one block = 1, 024 bytes – 32 bytes – 12 bytes = 980 bytes

- Maximum size and recommended size

  The maximum size is the total of the usable sizes of EEPROM emulation blocks. The recommended size is less than the maximum size to account for problems such as writing not being possible due to momentary power loss and other issues. It is recommended to only use within a value subtracting half a block capacity from the overall capacity.

<R> Maximum size = 980 bytes ×number of EEPROM emulation blocks ×1/4
<R> Recommended size = maximum size – 980/2

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

  Size of each written user data item = data size (a size in bytes adjusted to word units) + reference data size (8 bytes)

- Calculating the basic total user data size

  Basic total size = (user data 1 + 8) + (user data 2 + 8) ... + (user data n + 8)

- Calculating the total size when the basic total user data size exceeds the maximum size of one block

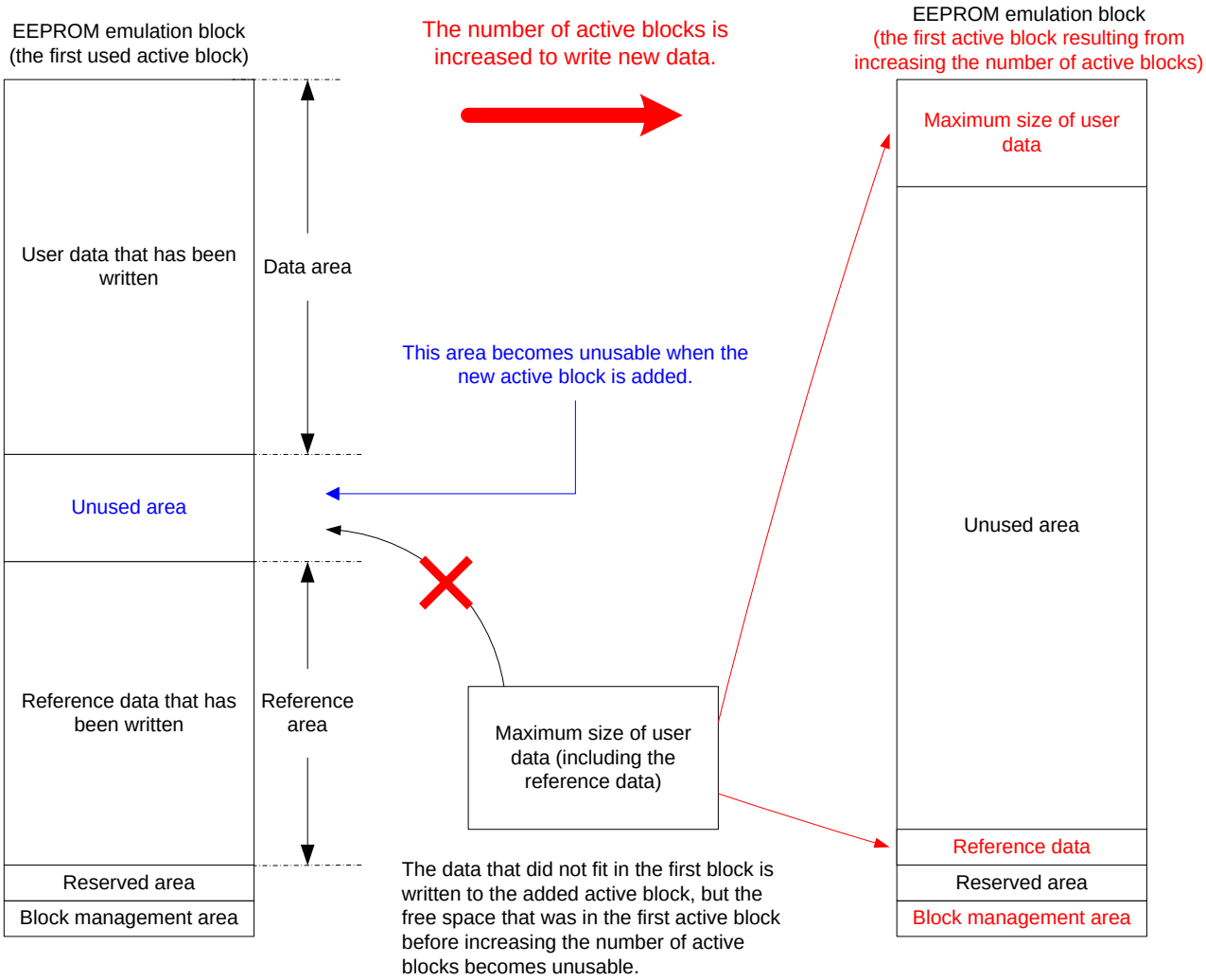
  If the basic total user data size exceeds 988 bytes, the maximum usable size of one block, it is necessary to include the maximum size that might become unusable in the calculation when increasing the number of active blocks.

Total size when more than one blocks are used = basic total size + ((largest user data size + 8) – minimum writing unit for EEPROM emulation (4 bytes)) × (number of necessary blocks – 1)

**Note** For details, see **1.2.2 Data structure**.

**Figure 1-13. Size Concepts When Total User Data Size Is Two Blocks (Which Exceeds One Block)**

- The size of the user data must fit within a quarter of EEPROM emulation blocks when recording all the data, but, if the total user data size does not fit in one block, it is necessary to include the maximum size that might become unusable in the calculation when increasing the number of active blocks.
  The maximum size that might become unusable is equal to the size of the largest user data that cannot be written.




Maximum size required for writing = largest user data size + reference data (8 bytes)
Maximum size that might become unusable = maximum size required for writing – size of the minimum writing unit for EEPROM emulation (4 bytes)
Total size when 2 blocks are used = basic total size + maximum size that might become unusable

# 1. 3 Initializing EEPROM Emulation Blocks

To use the data flash memory for the EEPROM emulation library, it is necessary to initialize the blocks as EEPROM emulation blocks. If there are blocks that need to be initialized, an EEL_ERR_POOL_INCONSISTENT error occurs when the EEL_CMD_STARTUP command is executed because there are no active EEPROM emulation blocks. To make the EEPROM emulation blocks usable, it is necessary to initialize the blocks by executing the EEL_CMD_FORMAT command.

In addition, if it becomes necessary to change the initial settings, such as because the data flash memory area is corrupted or because user data is added after initialization, or it otherwise becomes impossible to continue using the EEPROM emulation blocks in their current status, or if you just want to perform initialization, initialization can be performed at any time by executing the EEL_CMD_FORMAT command.

Figures 1-14 and 1-15 show the initialization flow and block status transitions used when the EEL_ERR_POOL_INCONSISTENT error occurs upon executing the EEL_CMD_STARTUP command, and Figures 1-16 and 1-17 show the flow and block status transitions used when performing initialization at an arbitrary time.

**Figure 1-14. Initialization Flow When EEL_ERR_POOL_INCONSISTENT Error Occurs (When Using Enforced Mode)**

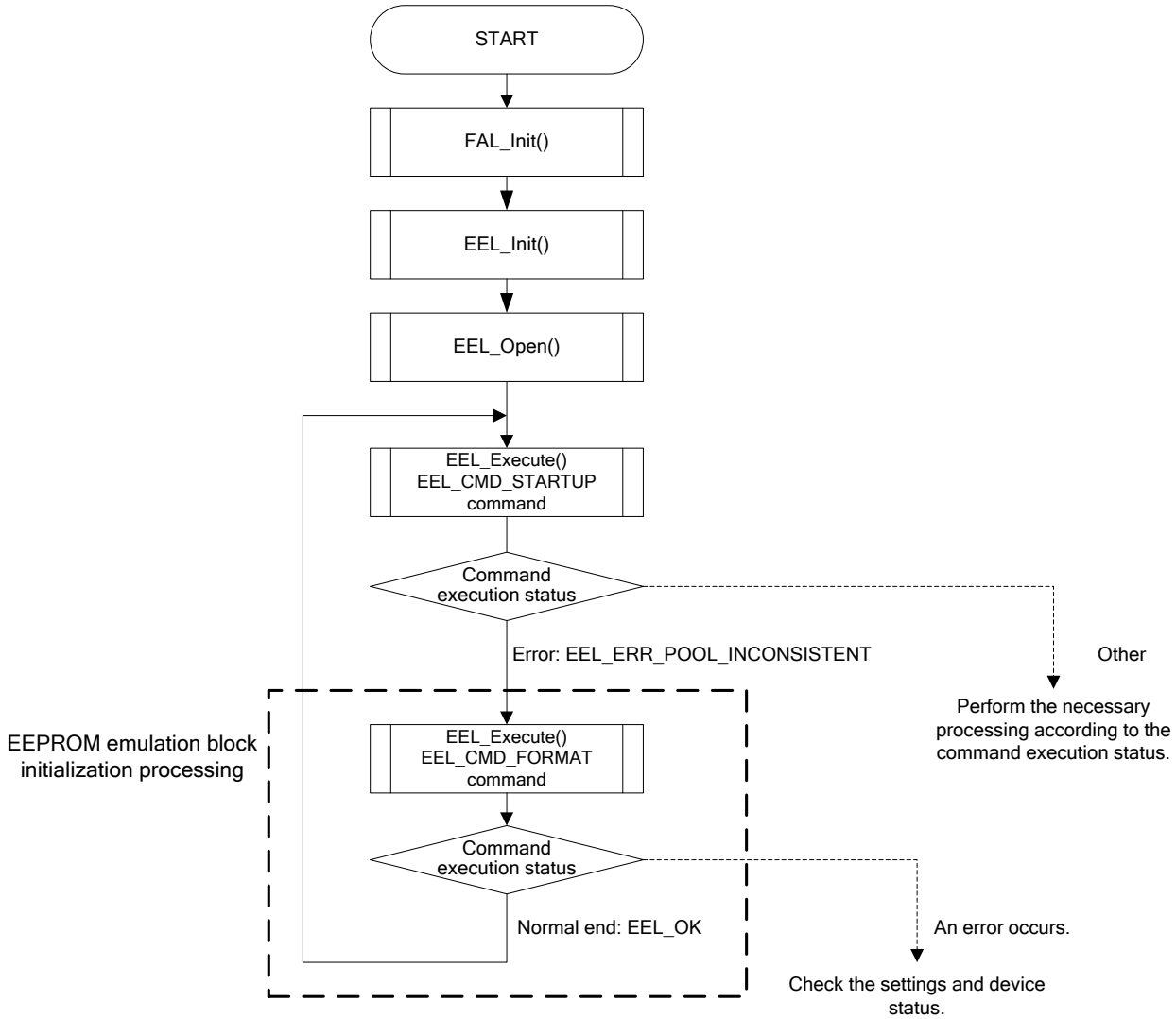


**Figure 1-15. Example Block Status Transitions When Initialization Is Performed Upon Occurrence of EEL_ERR_POOL_INCONSISTENT Error**

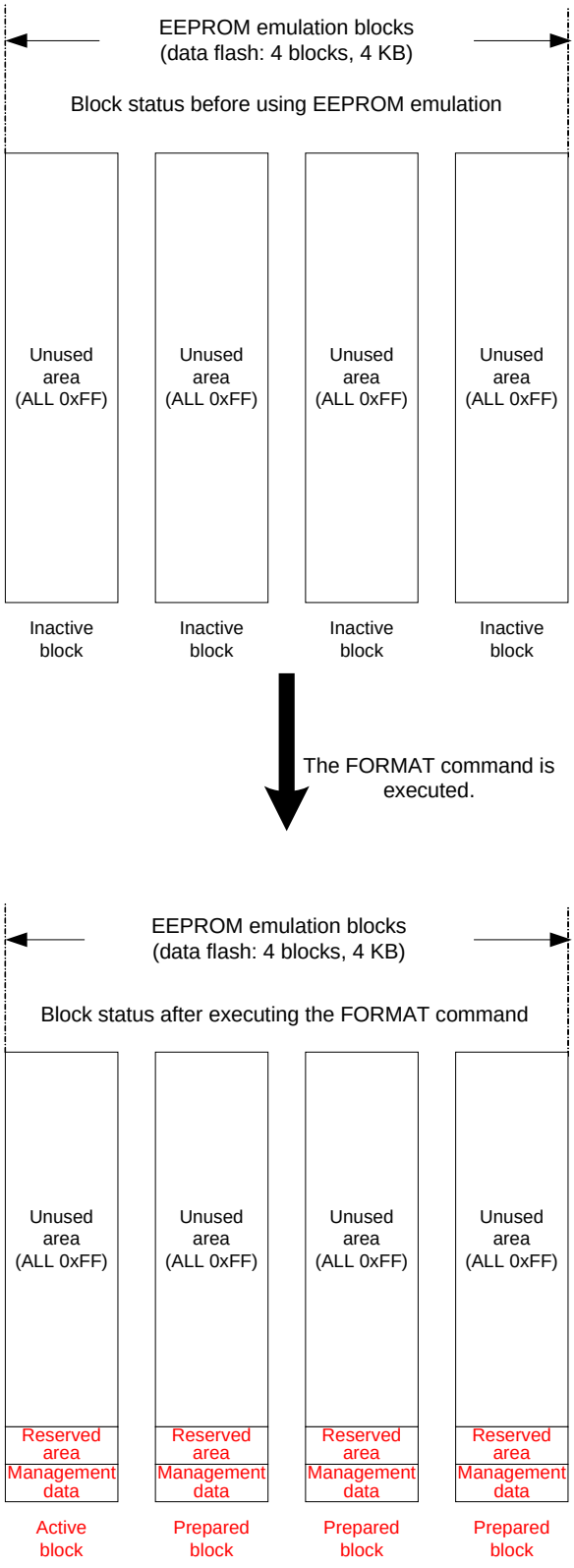


**Figure 1-16. Flow When Performing Initialization at Arbitrary Time (When Using Enforced Mode)**

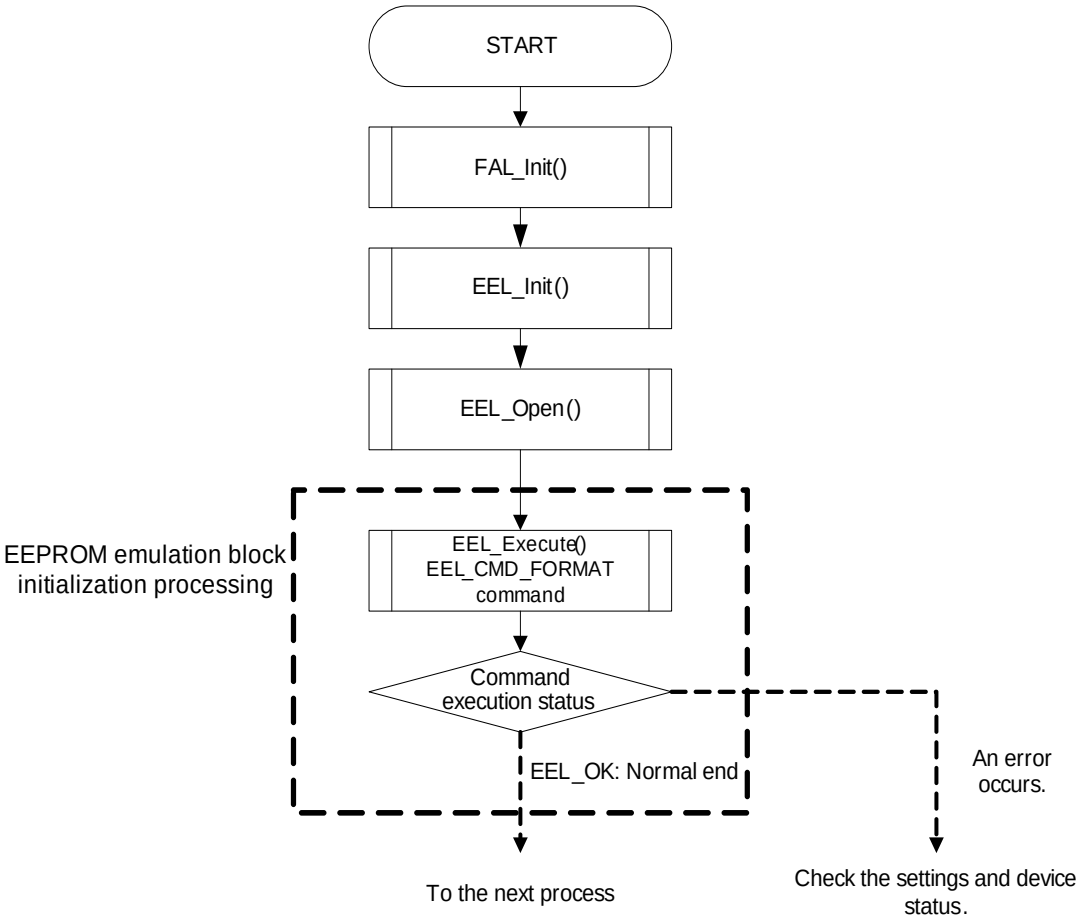
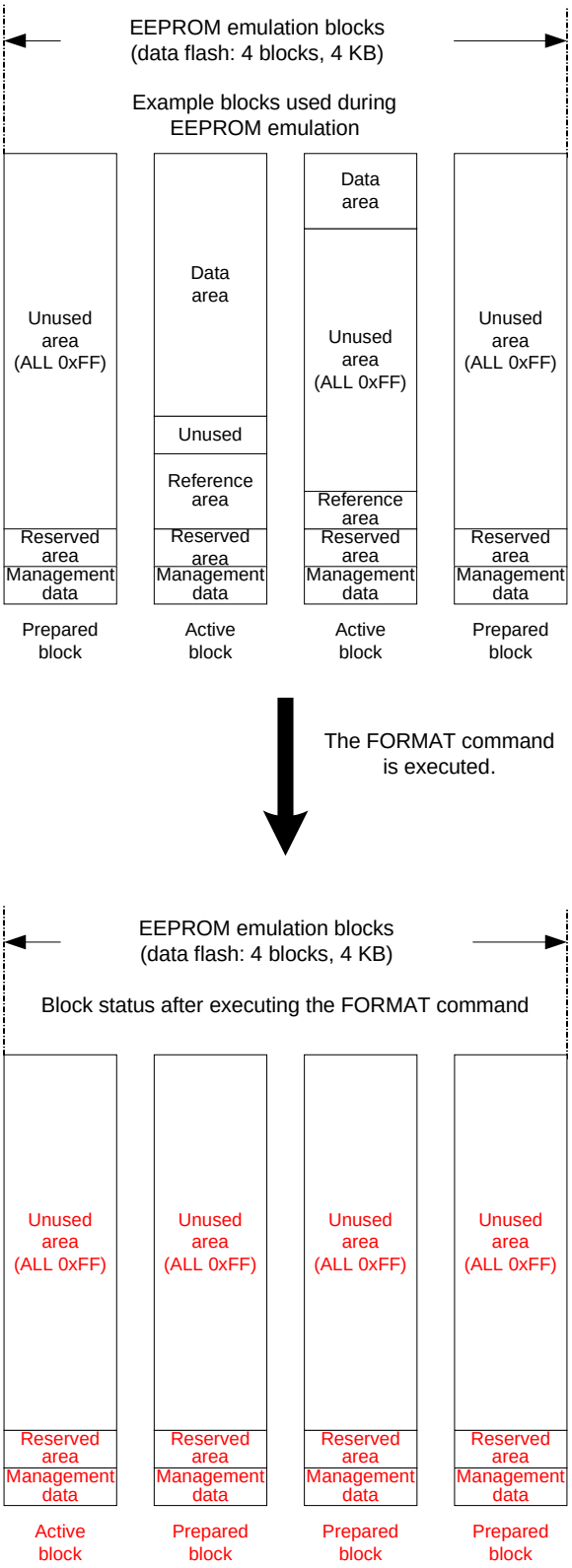


**Figure 1-17. Example Block Status Transitions When Used EEPROM Emulation Blocks Are Initialized at Arbitrary Time**



# 1. 4 Adjusting EEPROM Emulation Blocks

When writing to the EEPROM emulation blocks, a prepared block is erased each time an active block is added, but, if it is necessary to add an active block when there are two or fewer prepared blocks remaining, the active block is added, the valid data remaining in the old block is moved, and the old block is erased before writing the specified data. If writing is performed when erasing a block is required, the time necessary to move the data and erase the block is added to the time necessary for writing.

If this additional processing time is not permissible, it is possible to perform maintenance at a time that will not adversely affect the system in order to avoid moving and erasing data at the same time when high-priority data must be written.

To perform maintenance, either adjust blocks by executing the EEL_CMD_CLEANUP command or execute the maintenance mode processing of the EEL_Handler function.

Figure 1-18 shows an example of the block status when moving and erasing data upon adding an active block, and Figure 1-19 shows how the processing time is changed according to the block status when writing data.

**Figure 1-18. Example Block Status When Moving and Erasing Data upon Adding Active Block**

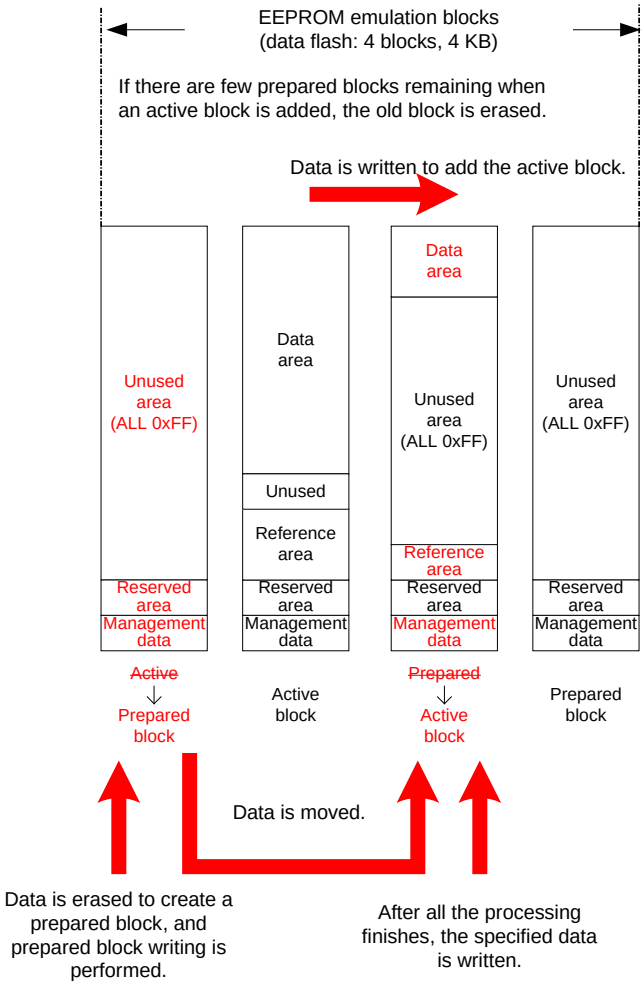
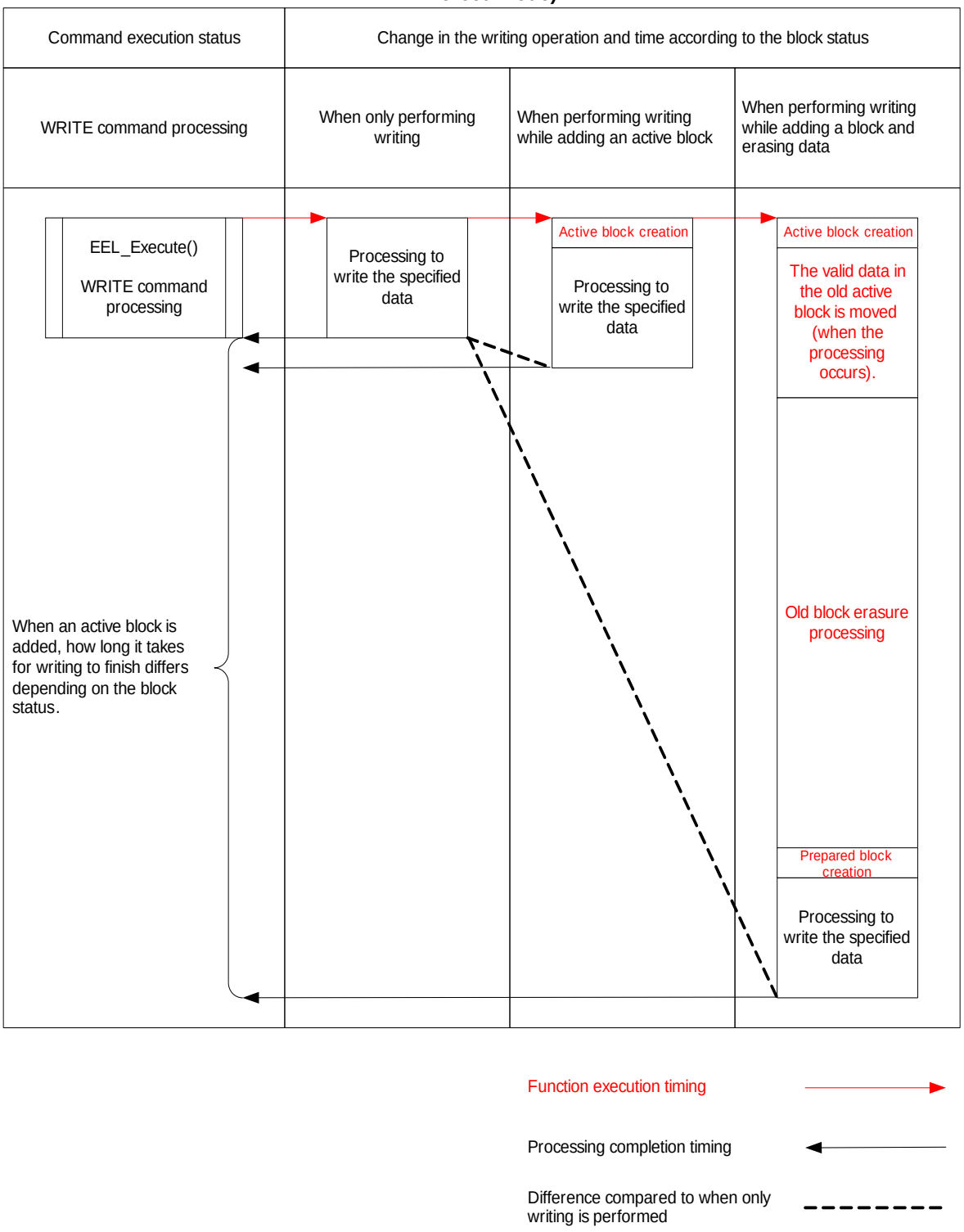


**Figure 1-19. Changes in Processing Time According to Block Status When Writing Data (When Using Enforced Mode)**



### 1. 4. 1 Adjusting blocks by using EEL_CMD_CLEANUP command

Blocks can be adjusted by executing the EEL_CMD_CLEANUP command from the EEL_Execute function.

Because this feature moves all the data to the latest newly created active block even if there is capacity remaining in the latest active block, the possible number of rewrites decreases by worth of data of blocks necessary to store data, but the feature makes it possible to change to a status in which there is only valid data (a status in which invalid data does not exist).

Figure 1-20 shows the flow when executing the EEL_CMD_CLEANUP command, and Figure 1-21 shows the block status after executing the EEL_CMD_CLEANUP command.

**Figure 1-20. Operation Flow of EEL_CMD_CLEANUP Command (When Using Enforced Mode)**

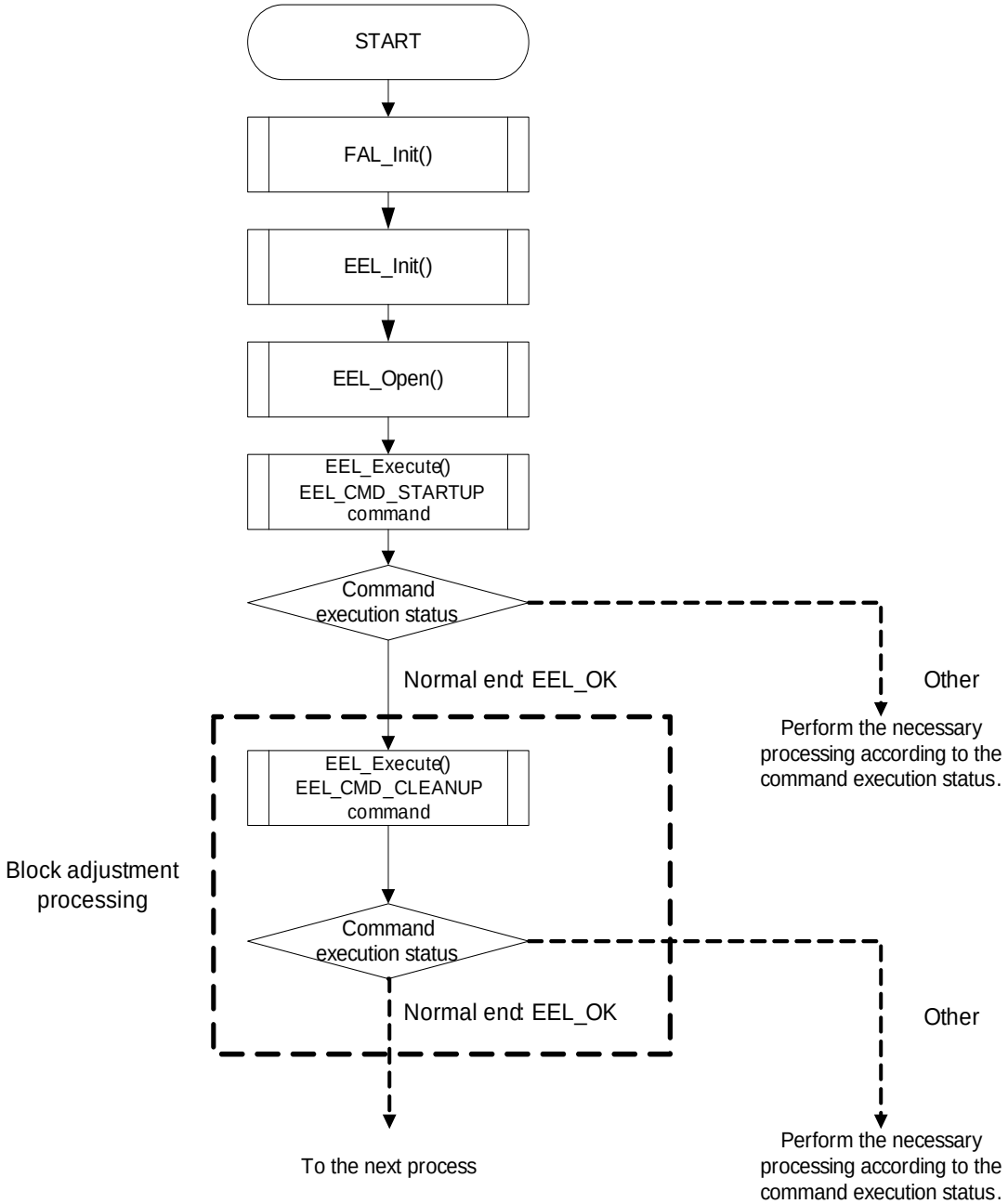
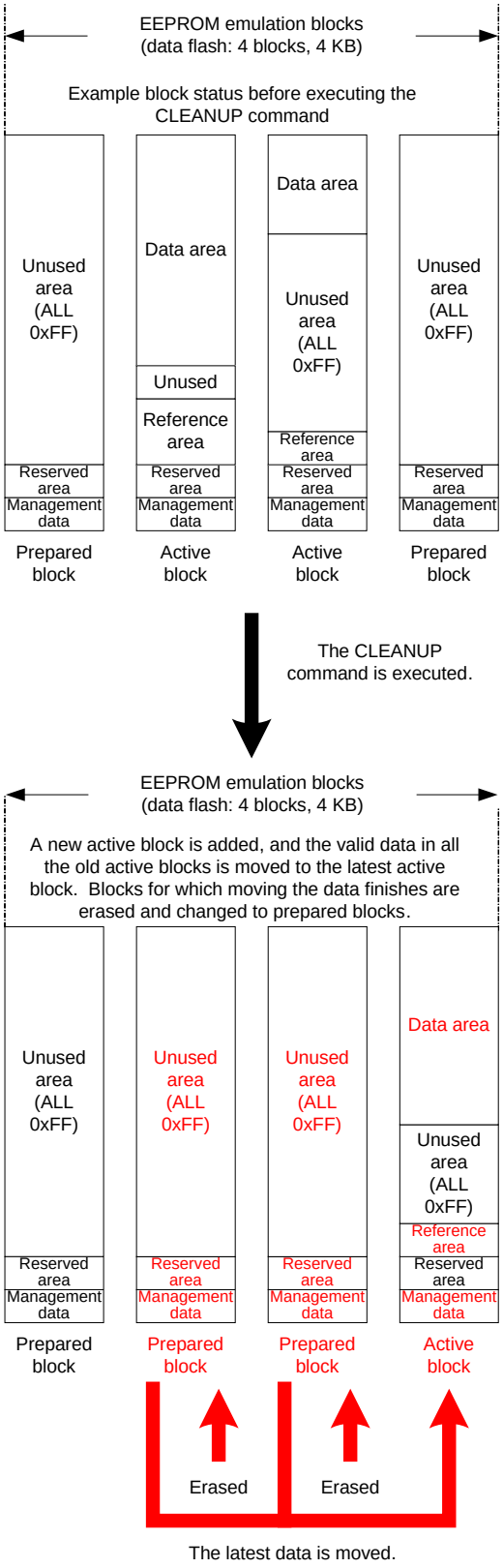


**Figure 1-21. Adjusting Blocks by Using EEL_CMD_CLEANUP Command**



## 1. 4. 2 Adjusting blocks by using EEL_Handler function (maintenance mode)

Block adjustment processing can be automatically performed by executing the EEL_Handler function while no commands are executing. This is called the maintenance mode. When the EEL_Handler function is executed while there are no commands to execute, the block status is checked, block adjustment is judged to be necessary if the number of active blocks exceeds the specified initial value EEL_REFRESH_BLOCK_THRESHOLD (described in **2.3 Initial Values to Be Set by User**), all necessary data in old active blocks is moved to the latest block, and the unnecessary blocks are erased and changed to prepared blocks.

The processing can be interrupted in the maintenance mode. If a new command is executed by the EEL_Execute function, the maintenance mode is paused, and executing the newly specified command is prioritized. To judge whether maintenance mode processing is being executed, it is necessary to execute the separate EEL_GetDriverStatus function and check the execution status of the EEL_Handler function. If nothing is being performed even when the EEL_Handler function is executed four times, adjustment is finished (the adjustment completion status).

Figure 1-22 and Table 1-3 show how to check the execution status of the EEL_Handler function, Figures 1-23 to 1-26 show an execution flow example, and Figure 1-27 shows the block transitions when the maintenance mode processing finishes (the adjustment completion status).

**Figure 1-22. Flow for Checking Execution Status of EEL_Handler Function**

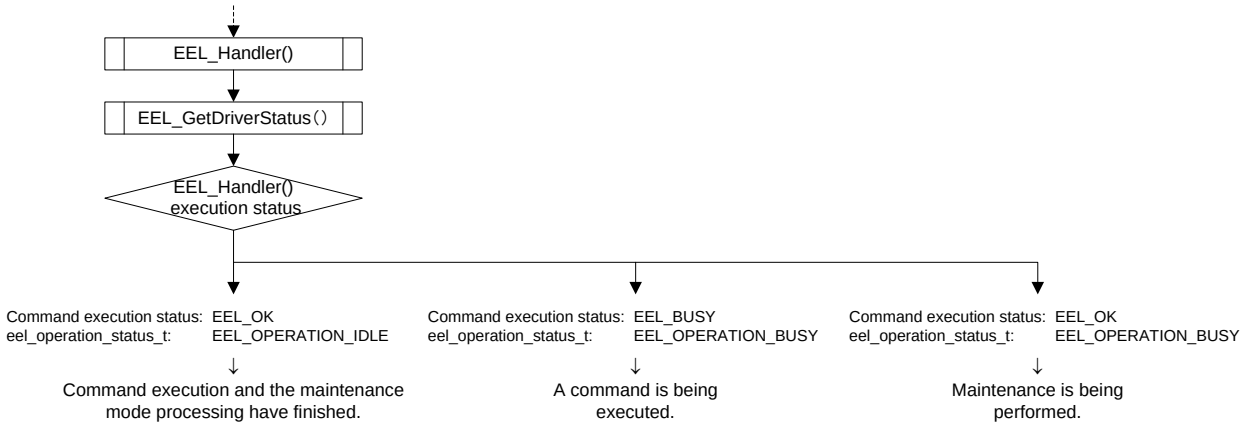



**Table 1-3. EEL_Handler Function Execution Status**

| EEL_Handler() Command Execution Status | EEL_GetDriverStatus() eel_operation_status_t Status | Description |
|---|---|---|
| EEL_OK | EEL_OPERATION_IDLE | This is the command execution and maintenance mode processing completion status.<br>(This includes when the maintenance mode processing is paused.)<br>If all the results are this status when the EEL_Handler function is executed four times, adjustment is complete. |
| EEL_OK | EEL_OPERATION_BUSY | Maintenance mode processing is being executed. |
| EEL_BUSY | EEL_OPERATION_BUSY | A command is being executed. |

**Figure 1-23. Maintenance Mode Execution Flow (Example of Execution Until System Enters Adjustment Completion Status)**

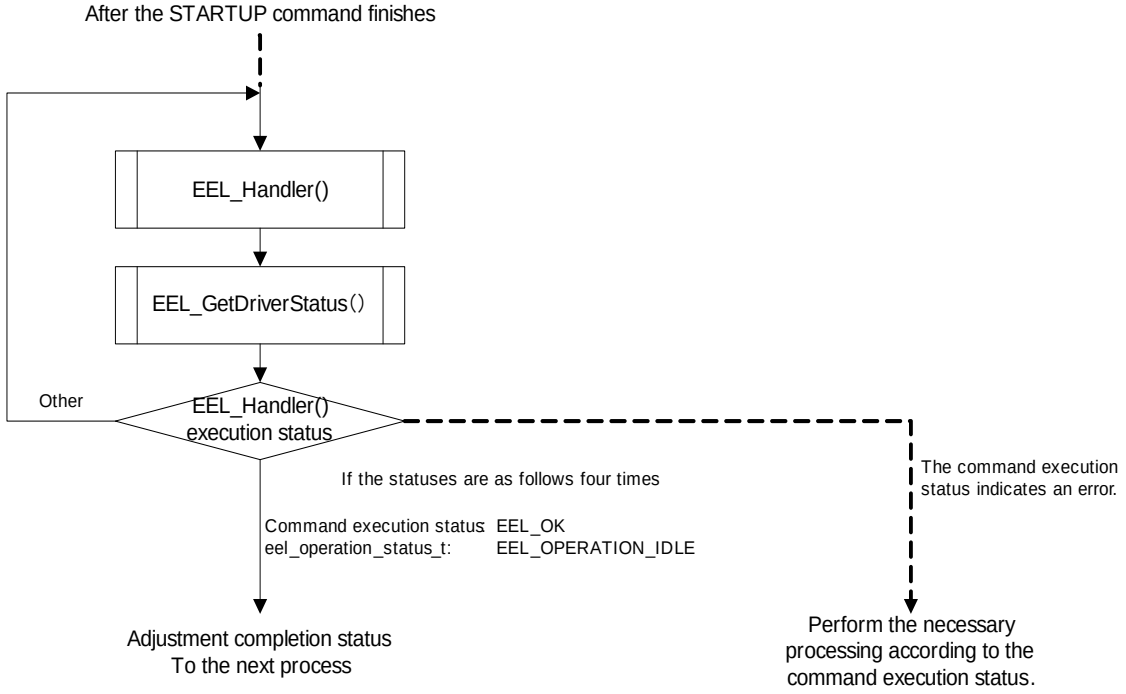



**Figure 1-24. Execution Flow When Performing Adjustment After Executing EEL_CMD_WRITE Command (Adjustment Completion Status)**

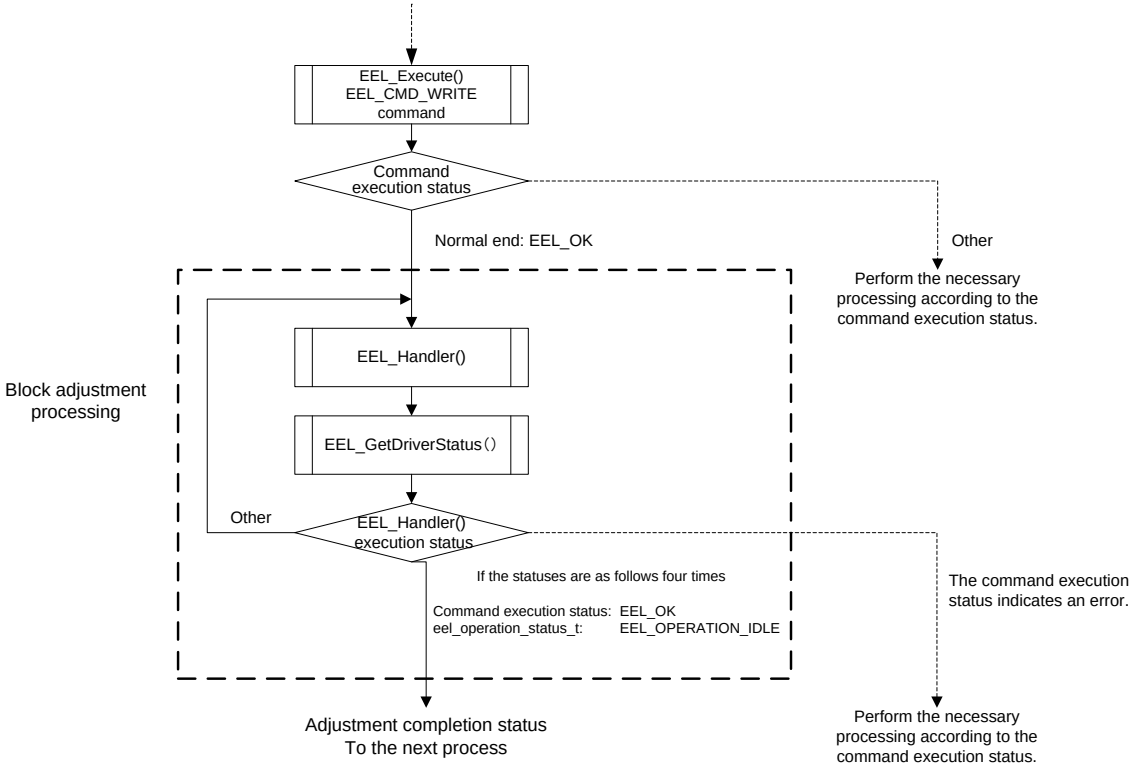


**Figure 1-25. Execution Flow When Performing Adjustment upon Starting EEPROM Emulation (Adjustment Completion Status)**

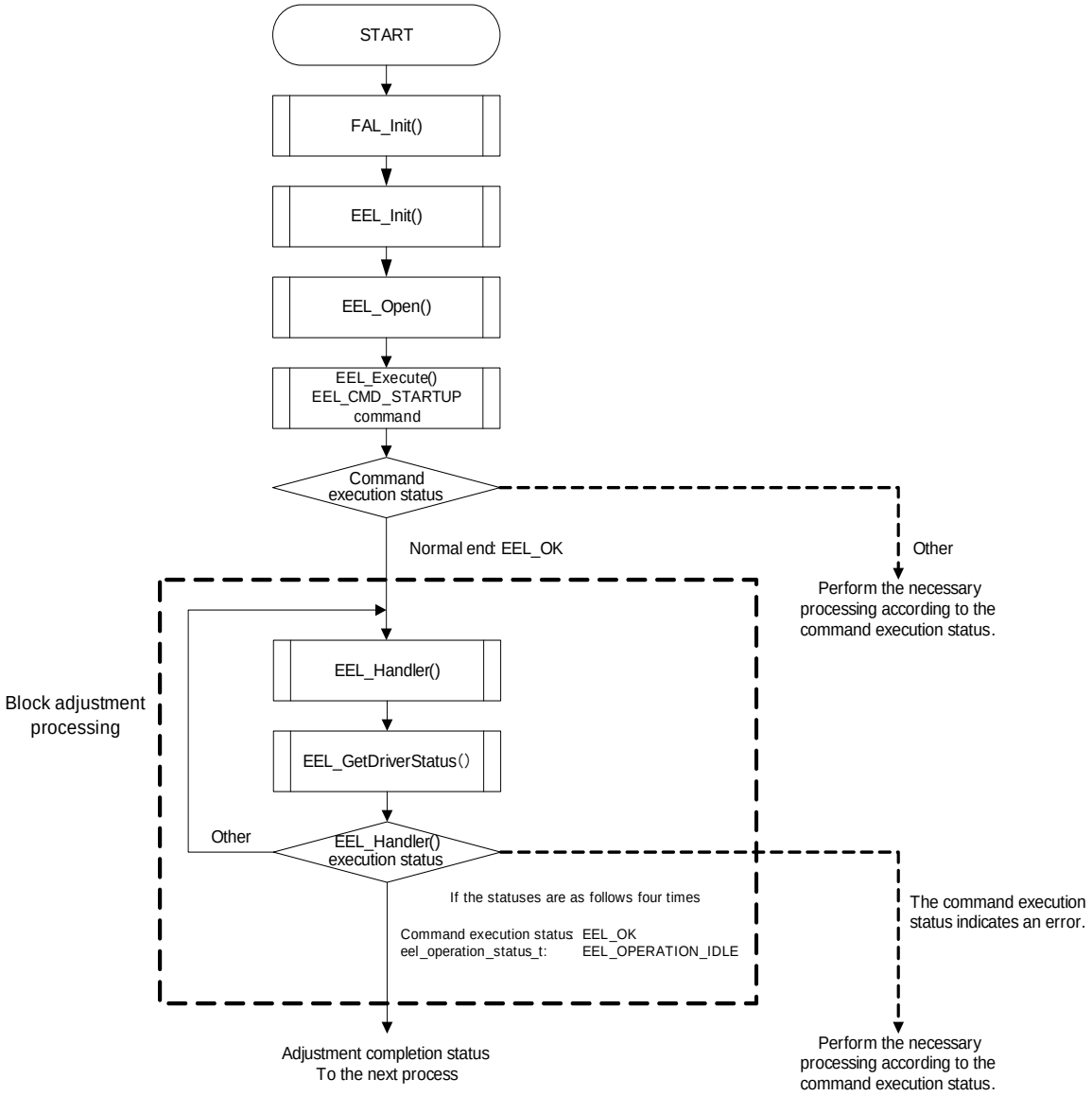


**Figure 1-26. Example Flow for Executing Command and Maintenance Mode Processing in Polling Mode**

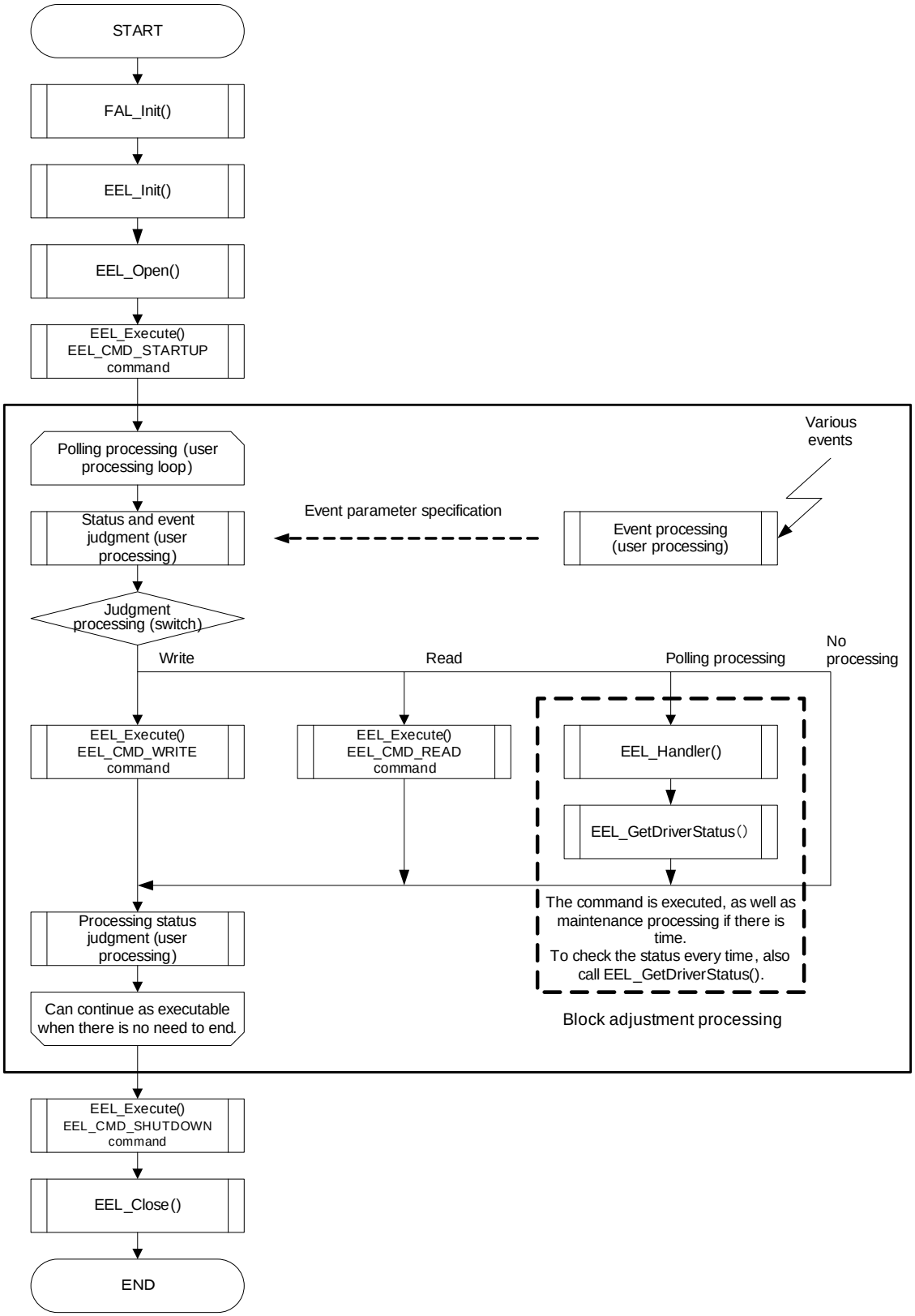
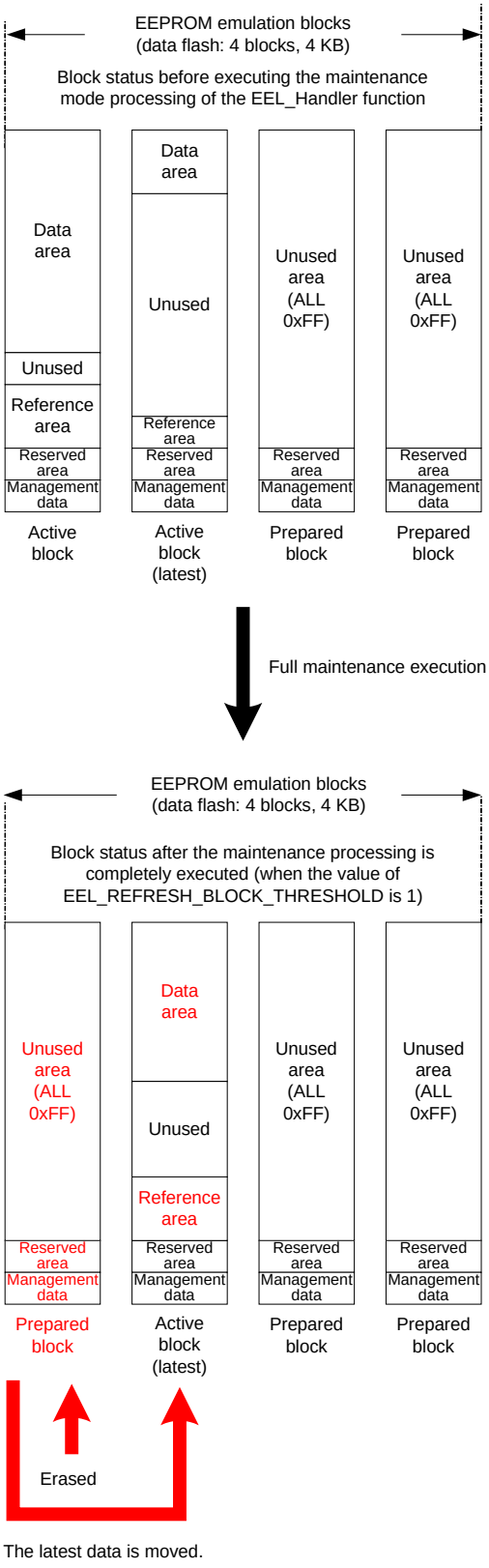


**Figure 1-27. Example of Block Adjustment in EEL_Handler Function Maintenance Mode (Adjustment Completion Status)**



# CHAPTER 2 USING EEPROM EMULATION

EEPROM emulation can store a maximum of 255 [Note] data items each consisting of 1 to 255 bytes in the flash memory by using eight blocks of flash memory.

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

**Note** For details about the number of user data items that can be stored, see **1.2.4 Number of stored user data items and total user data size**.

## 2. 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:

(1) Before using the EEPROM emulation library, always close the flash self programming library. Also, do not run the flash self programming library while the EEPROM emulation library is being used. When using the flash self programming library, be sure to execute all of the processing up to and including the EEL_Close function to finish EEPROM emulation.
When using EEPROM emulation after executing flash self programming library processing, it is necessary to start processing from the initializing function (the FAL_Init function).

(2) 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 to finish EEPROM emulation.

<R> (3) The watchdog timer does not stop during the execution of the EEPROM emulation Library Pack 01.

<R> (4) The data flash memory cannot be read during data flash memory operation by the EEPROM emulation Library Pack 01.

(5) In address above 0xFFE20 (0xFE20), do not place data buffer (argument) or stack which is used by EEPROM emulation library functions and data flash library functions.

(6) 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).

(7) Until EEPROM emulation is finished, do not corrupt RAM area (including self RAM) used by EEPROM emulation.

(8) Do not execute any of the EEPROM emulation library functions during interrupt servicing except for the EEL_TimeOut_CountDown function. Because the EEPROM emulation library functions do not support execution more than once at the same time, the operation is not guaranteed when executing these functions during interrupt servicing.

(9) When executing EEPROM emulation library processing in an OS, except for the EEL_TimeOut_CountDown function, do not execute any of the EEPROM emulation library functions from multiple tasks. Because the EEPROM emulation library functions do not support execution more than once at the same time, the operation is not guaranteed when executing these functions from multiple tasks.

(10) Before starting the EEPROM emulation, be sure to start up the high-speed on-chip oscillator first.

(11) About an operation frequency of RL78 microcontrollers and an operation frequency value set by the initializing function (FAL_Init), be aware of the following points:

- When using a frequency lower than 4 MHz as an operation 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 operation frequency value set by the initializing function.
- When using a frequency of 4 MHz or higher Note1 as an operation frequency of RL78 microcontrollers, a certain frequency can be used as an operation frequency of RL78 microcontrollers.

<R> - This operation frequency is not the frequency of the high-speed on-chip oscillator.

<R> (12) Do not operate the DFTCTL during the execution of the EEPROM emulation library .

<R> (13) 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 of the target RL78 microcontroller.

<R> (14) Initialize the SADDR area 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 of the target RL78 microcontroller.

(15) 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.

**Notes 1.** For a maximum frequency, see the target RL78 microcontroller user's manual.

(16) After initializing the EEPROM emulation blocks, do not change the initial values specified as fixed values for stored data and the blocks. If these values are changed, the specified parameters become inconsistent with the data written to the blocks, and there is a risk of no longer being able to correctly execute EEPROM emulation. If the values must be changed, execute the EEL_CMD_FORMAT command to reinitialize the EEPROM emulation blocks.

<R> (17) Four or more data flash memories are necessary to use the EEPROM Emulation Library Pack 01. If the number of data flash memory blocks is less than four, this library is unusable.

## <R> 2. 2 Total Processing Time

The total processing time is the time until successful termination. This does not include the time until abnormal termination due to errors in the input data or other errors.

**Figure 2-1 Overview of Total Processing Time**

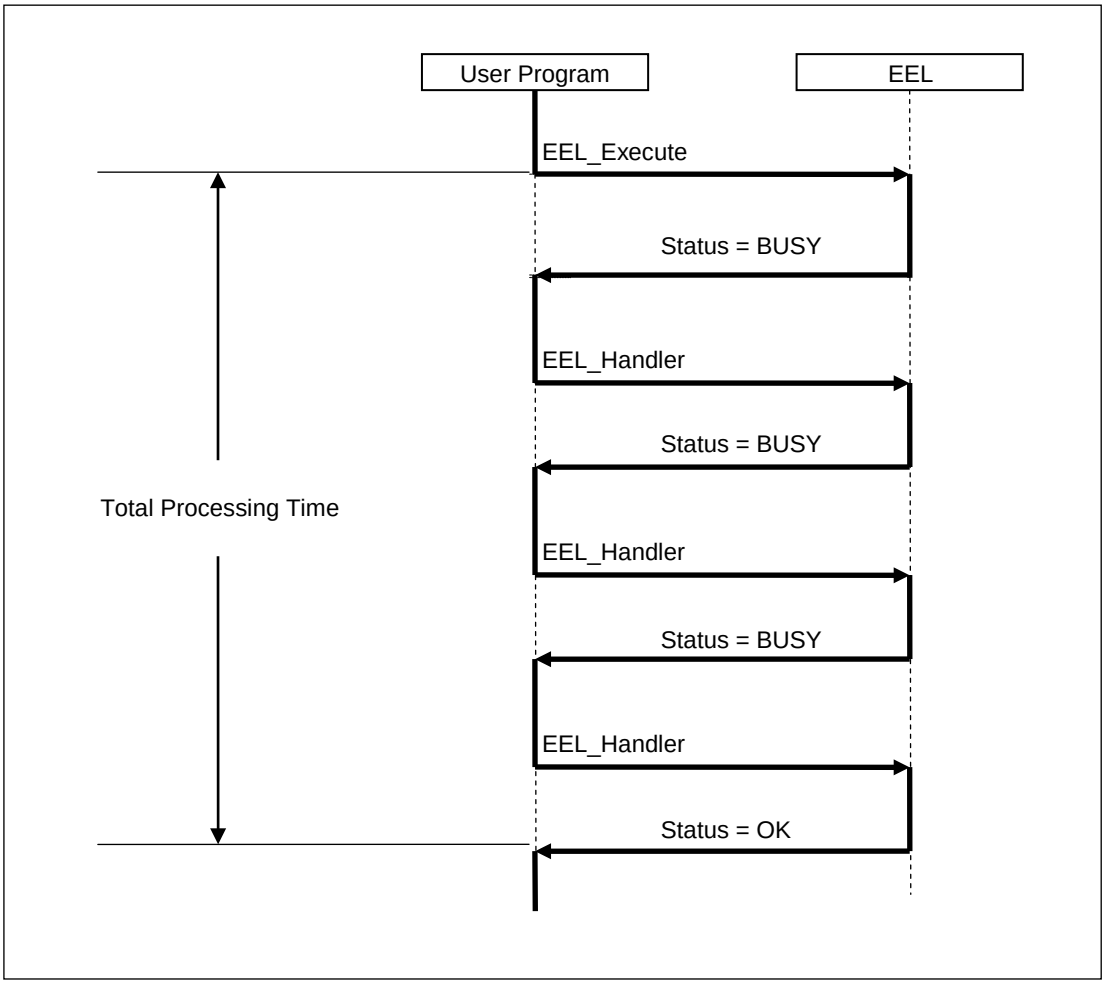


**Table 2-1 Total Processing Time of EEPROM Emulation Library Pack 01 (When All Data Fits in 1 Block)**

| Funcstions | | MAX time(Full Speed Mode) | MAX time(Wide Voltage Mode) |
|---|---|---|---|
| FAL_Init | | 2580 / fclk + 443 µs | 2536 / fclk + 968 µs |
| EEL_Init | | (2123 + 80 × Data Num) / fclk µs | (2123 + 80 × Data Num) / fclk µs |
| EEL_Open | | 87 / fclk + 12 µs | 87 / fclk + 12 µs |
| EEL_Close | | 866/ fclk + 443 µs | 822/ fclk + 968 µs |
| EEL_GetDriverStatus | | 47 / fclk µs | 47 / fclk µs |
| EEL_GetSpace | | 57 / fclk µs | 57 / fclk µs |
| EEL_GetVersionString | | 10 / fclk µs | 10 / fclk µs |
| EEL_TimeOut_CountDown | | 30 / fclk µs | 30 / fclk µs |
| EEL_Excute/EEL_Handler | | | |
| ・EEL_CMD_FORMAT | | (352494 / fclk + 330988)×Block Num µs | (311793 / fclk + 374134)×Block Num µs |
| ・EEL_CMD_START UP 1. Minimum data length is 1~4 bytes | Data Num 8 | (1082020+ 491768 ×(Block Num - 2)) / fclk + 330988 µs | (1054886+ 491768 × (Block Num - 2)) / fclk + 374134 µs |
| | Data Num 64 | (1926490+ 4482900 ×(Block Num - 2)) / fclk + 330988 µs | (1899356+ 4482900 × (Block Num - 2)) / fclk + 374134 µs |
| | Data Num 128 | (2891690+ 9044100 ×(Block Num - 2)) / fclk + 330988 µs | (2864556+ 9044100 × (Block Num - 2)) / fclk + 374134 µs |
| | Data Num 255 | (4806090+ 18097500×(Block Num - 2)) / fclk + 330988 µs | (4777956+ 18097500× (Block Num - 2)) / fclk + 374134 µs |
| 2. Minimum data length is 13~16 Byte | Data Num 8 | (771606 + 256268 × (Block Num - 2)) / fclk + 330988 µs | (744472 + 256268 × (Block Num - 2)) / fclk + 374134 µs |
| | Data Num 64 | (1189076+ 2363400 × (Block Num - 2)) / fclk + 330988 µs | (1161942+ 2363400 × (Block Num - 2)) / fclk + 374134 µs |
| | Data Num 128 | (1666676+ 4771500 × (Block Num - 2)) / fclk + 330988 µs | (1639542+ 4771500 × (Block Num - 2)) / fclk + 374134 µs |
| | Data Num 255 | (2613676+ 9550500 × (Block Num - 2)) / fclk + 330988 µs | (2586542+ 9550500 × (Block Num - 2)) / fclk + 374134 µs |
| 3.Minimum data length is 61~64 Byte | Data Num 8 | (569540 + 101109 × (Block Num - 2)) / fclk + 330988 µs | (542406 + 101109 × (Block Num - 2)) / fclk + 374134 µs |
| | Data Num 64 | (705816 + 967779 × (Block Num - 2)) / fclk + 330988 µs | (678682 + 967779 × (Block Num - 2)) / fclk + 374134 µs |
| | Data Num 128 | (861528 + 1958100 × (Block Num - 2)) / fclk + 330988 µs | (834394 + 1958100 × (Block Num - 2)) / fclk + 374134 µs |

Emarks. fclk: CPU/peripheral hardware clock frequency (for example, at 20 MHz, fclk = 20)

Data Num: Number of data entries registered

Block Num: Number of EEPROM emulation blocks

| Funcstions | MAX (Full Speed Mode) | MAX (Wide Voltage Mode) |
|---|---|---|
| EEL_Excute/EEL_Handler | | |
| ・EEL_CMD_CLEANUP | (352494 / fclk + 330988)× Block Num + 4236841 / fclk + 223893 µs | (311793 / fclk + 374134)× Block Num + 4219942 / fclk + 851467 µs |
| ・EEL_CMD_WRITE (Max data size) | 5763174 / fclk + 1024588 µs | 5650473 / fclk + 1906134 µs |
| ・EEL_CMD_READ | 58563 / fclk µs | 58563 / fclk µs |
| ・EEL_CMD_SHUTDOWN | 1674 / fclk µs | 1674 / fclk µs |

Emarks. fclk: CPU/peripheral hardware clock frequency (for example, at 20 MHz, fclk = 20)

Block Num: Number of EEPROM emulation blocks

<R> 

## 2. 3 Software Resources

In the EEPROM emulation library, 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 data flash library will be used, the EEPROM emulation library must have a separate area for use by the data flash library.

Tables 2-2 to 2-3 list the software resources required by each microcontroller.

**Table 2-2 Software Resources Used by EEPROM Emulation Library Pack01**

| Item | Size(Byte) | Restrictions on Allocation and Usage[Note1,2] | | |
|---|---|---|---|---|
| Self RAM[Note3] | 0 to 1024[Note3] | RL78/D1A | RAM 4KB ROM 64KB | FEF00H to FF2FFH |
| | | | RAM 16KB ROM 256KB | FBF00H to FC2FFH |
| | | RL78/F12 | RAM 4KB ROM 64KB | FEF00H to FF2FFH |
| | | RL78/G13 | RAM 4KB ROM 64KB | FEF00H to FF2FFH |
| | | | RAM 20KB ROM 256KB<br>*Excluding 128-pin products | FAF00H to FB2FFH |
| | | | RAM 32KB ROM 512KB | F7F00H to F82FFH |
| | | RL78/G14 | RAM 5.5KB ROM 64KB | FE900H to FECFFH |
| | | | RAM 24KB ROM 256KB | F9F00H to FA2FFH |
| | | RL78/G1A | RAM 4KB ROM 64KB | FEF00H to FF2FFH |
| | | RL78/I1A | RAM 4KB ROM 64KB | FEF00H to FF2FFH |
| | | RL78/L13 | RAM 8KB ROM 128KB | FDF00H to FE2FFH |
| | | ALL[Note1,2] | Products other than the above[Note1,2] | Contact us. |
| Stack | 100 | Can be allocated to a RAM area other than the self RAM and the area from FFE20H to FFEFFH | | |
| Data buffer [Note4] | 1 to 256 | | | |
| Requester(Argument) | 6 | | | |
| Library search area | 2 | | | |
| SADDR RAM work area | 11<br>(fdl:2)<br>(eel:9) | Can be allocated to a short-addressing RAM area | | |
| Library size | 8200<br>(fdl:1500)<br>(eel:6700) | Can be allocated to a program area other than the self RAM, the area from FFE20H to FFEFFH, and the internal ROM | | |
| Data table | (*n* + 1) * 4<br>n = number of data items stored | | | |
| Fixed-parameter area (default) | 72<br>(fdl:64)<br>(eel:4) | | | |
| EEPROM emulation block | 4,096 or more (at least 4 blocks) | Only data flash memory can be used. | | |

Notes: 1. Please contact us for the products that are newly added after this document has been issued.

2. The RL78/G12, L12 and G1C product does not support the EEPROM Emulation Library Pack01.

3. An area used as the working area by the EEPROM Emulation library Pack 01 is called self-RAM in this manual and the user's manual. The self-RAM requires no user setting because it is an area that is not mapped and automatically used at execution of the EEPROM Emulation library (previous data is

discarded). When the EEPROM Emulation library is not used, the self-RAM can be used as a normal RAM space.

For the RL78 microcontroller with self-RAM, the chapter of "memory space" in the user's manual of the RL78 microcontroller has a note on an area (self-RAM) whose usage is prohibited during EEPROM Emulation Library Pack01. If the above table does not include the target RL78 microcontroller, refer to the user's manual of the target RL78 microcontroller.

4. The data buffer is used as the working area for EEPROM Emulation Library Pack01 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 2-3. Data Buffer Size Used by EEPROM Emulation Library Functions**

| Function Name | Bytes |
|---|---|
| FAL_Init | 0 |
| EEL_Init | 0 |
| EEL_Open | 0 |
| EEL_Close | 0 |
| EEL_Execute[note] | 0 ～ 256 |
| EEL_Handler[note] | 0 ～ 256 |
| EEL_TimeOut_CountDown[note] | 0 |
| EEL_GetDriverStatus | 3 |
| EEL_GetSpace | 2 |
| EEL_GetVersionString | 0 |

Note. The data buffer area specified by the EEL_Execute function is used as is.

**Figure 2-2 Arrangement Example of Self-RAM and Addresses FFE20H to FFEFFH**

**(RL78/G13: product with 4-Kbyte RAM and 64-Kbyte ROM)**

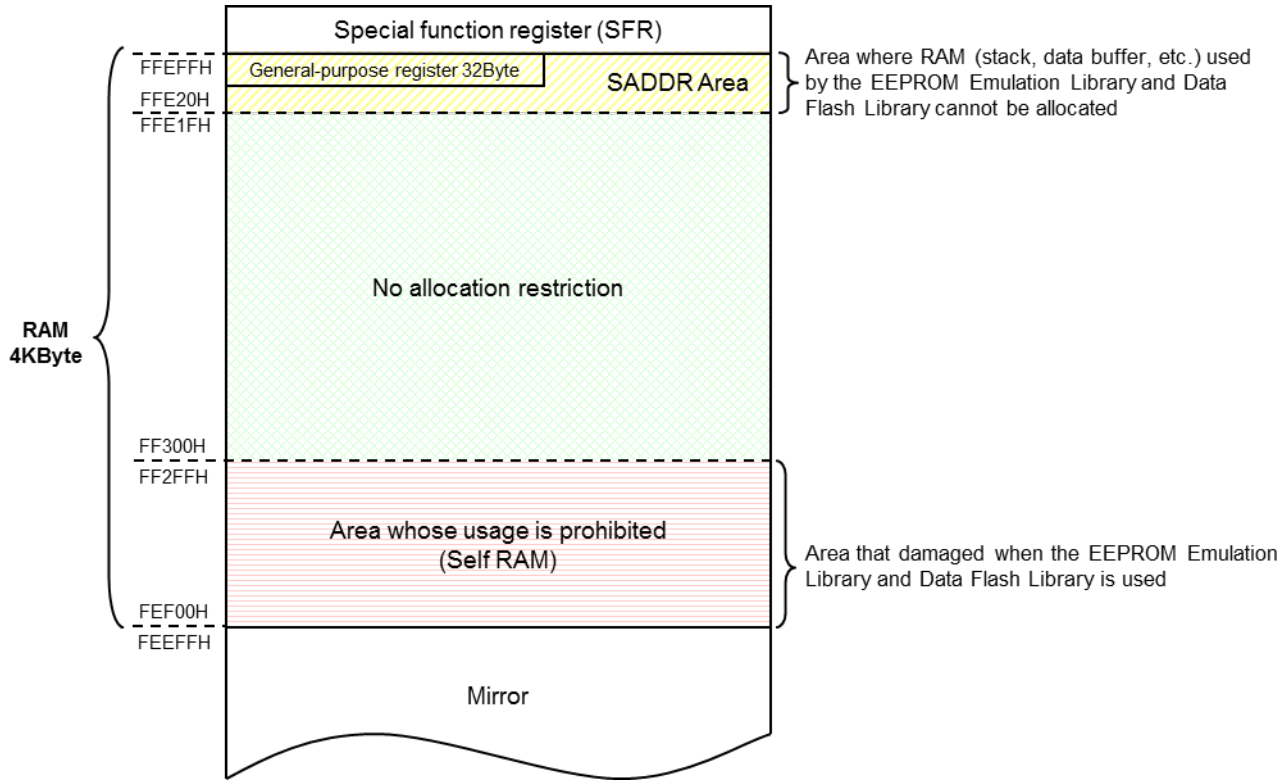
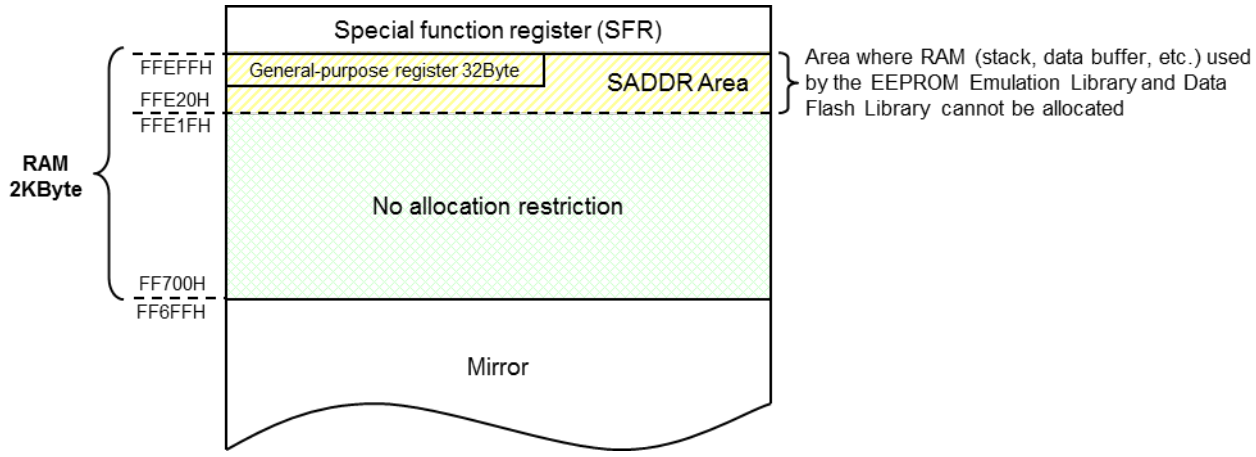



**Figure 2-3 Arrangement Example of Addresses FFE20H to FFEFFH**

**(RL78/G13: product with 2-Kbyte RAM and 32-Kbyte ROM)**



## 2. 4 Initial Values to Be Set by User

<R> As the initial values for the EEPROM emulation library, be sure to set the items indicated below. In addition, before executing the EEPROM emulation library, be sure to execute the high-speed on-chip oscillator.

- Number of stored data items, and specific data IDs and data size

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

```
#define   FDL_SYSTEM_FREQUENCY     2000000:     (1) Operation frequency
#define   FDL_WIDE_VOLTAGE_MODE:                (2) Voltage mode
#define   FAL_POOL_SIZE     4:                  (3) FAL pool size
#define   EEL_POOL_SIZE      0:                 (4) FEL pool size
```

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

```
#define   EEL_STORAGE_TYPE     'D':             Flash memory type (D: Data flash memory)
#define   EEL_VAR_NO       5:                   Number of stored data items
#define   EEL_REFRESH_BLOCK_THRESHOLD   1:      Threshold setting
```

<EEPROM emulation library user program file (eel_descriptor.c)>[Notes 1, 2]

```
__far const eel_u08 eel_descriptor[EEL_VAR_NO+1][4]:   (1) Data identifier (data ID) and size
                                                       = (eel_u08)EEL_REFRESH_BLOCK_THRESHOLD;
__far const eel_u08    eel_refresh_bth_u08:            (2) Threshold setting
                                                       = (eel_u08)EEL_STORAGE_TYPE;
__far const eel_u08    eel_storage_type_u08:           (3) Flash memory type
                                                       = (eel_u08)EEL_VAR_NO;
__far const eel_u08    eel_var_number_u08:             (4) Number of stored data items
```

**Notes 1.** The macros and macro names that are being used have common parameters with the EEPROM emulation library, so changes should be made to numerical values only.

**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 EEPROM emulation blocks.

(1) Operation frequency

This sets an operation frequency which is used in RL78 microcontrollers. Note

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

Setting value of FAL_Init operation 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** This setting is a value required to control data flash memory. This setting does not change the operation <R> frequency of RL78 microcontrollers. In addition, this operation frequency is not the frequency of the high-speed on-chip oscillator.

(2) Voltage mode Note 1

This sets the voltage mode of data flash memory. Note 2

When FDL_WIDE_VOLTAGE_MODE is not defined: Full-speed mode
When FDL_WIDE_VOLTAGE_MODE is defined: Wide voltage mode

**Notes 1.** 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.

**2.** For details of the voltage mode, see the corresponding RL78 microcontrollers user's manual.

(3) FAL pool size

Set the number of blocks of data flash memory mounted on your RL78 microcontrollers.

(4) EEL pool size Note

Set the number of blocks of data flash memory mounted on your RL78 microcontrollers.

**Note** Set the value of 4 (four blocks) or more.

(5) Data identifier (data ID) and size

This table specifies the data identifiers (data IDs) and sizes. It is called the EEL descriptor table. With RL78 EEPROM emulation library Pack01, it is not possible to add identifiers while the program is running. Accordingly, the data to be written must be registered to the EEL descriptor table in advance.

**Figure 2-4. EEL Descriptor Table (When There Are Four Instances of Different Data)**

**__far const eel_u08 eel_descriptor [Number of stored data items + 1][4]**

| | | | |
|---|---|---|---|
| Data ID (A) | Word size | Byte size | 0x01 |
| Data ID (B) | Word size | Byte size | 0x01 |
| Data ID (C) | Word size | Byte size | 0x01 |
| Data ID (D) | Word size | Byte size | 0x01 |
| 0x00 | 0x00 | 0x00 | 0x00 |

Data ID
The data ID is specified by the user.

Word size
This is the word size of the data to be written.

Byte size
This is the byte size of the data to be written.

RAM reference flag (0x01)
This flag is used for reference settings. Specify 1 as the registration data.

Termination area (0x00)
Specify 0 as the termination information.

(6) Threshold setting
This is the number of active blocks used as the adjustment reference when executing maintenance mode processing. The maintenance mode processing is executed when the number of active blocks exceeds this setting. For this setting, specify the number of blocks [Note] necessary for valid data storage + 1. When the capacity necessary for data storage is less than a half of maximum usable size of one block, + 1 is not required.

**Note** For details about the number of blocks necessary for valid data storage, see **1.2.4 Number of stored user data items and total user data size**.

(7) Flash memory type
This does not have to be changed. Use the initial value as is.

(8) Number of stored data items
This is set as the number of data items used for EEPROM emulation. The setting range is from 1 to 255.

# CHAPTER 3 EEPROM EMULATION FEATURES

This chapter describes the functions used for EEPROM emulation.

## 3. 1 Data Flash Library Functions

When executing EEPROM emulation by using the data flash memory, operations such as erasing or writing to the data flash memory are performed by manipulating the data flash library from the EEPROM emulation library. Therefore, before executing EEPROM emulation, it is necessary to initialize the data flash library, specify the settings for accessing the data flash memory, and make other preparations. Table 3-1 shows the data flash library function necessary for executing EEPROM emulation library processing.

**Table 3-1. Data Flash Library Function That Must Be Executed Before Using EEPROM Emulation**

| Function Name | Overview |
|---|---|
| FAL_Init | This initializes the data flash library. |

# FAL_Init

[Outline]

Data flash library initialization processing

[Format]

<C language>

```
fal_status_t  __far FAL_Init(const __far fal_descriptor_t* descriptor_pstr)
```

<Assembler>

```
CALL !_FAL_Init or CALL !!_FAL_Init
```

**Remark** Call this function by using ! if placing the data flash library at 00000H to 0FFFFH or by using !! if not.

[Advance setting]

- The flash self programming library and EEPROM emulation library processing must be either not executing or finished.
- <R> Before executing this function, be sure to execute the high-speed on-chip oscillator.

[Function]

This function initializes parameters for accessing the data flash memory.

**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 flash self programming library (FSL). Before executing this function, be sure to close the flash self programming library. Also, never use any flash self programming library functions during EEPROM emulation.

**3.** To use the flash self programming library after this function is executed, the RAM must be reinitialized, so always execute this function when restarting the EEPROM emulation library.

**4.** To execute this function again, always close the EEPROM emulation library.

**5.** The table used for this function cannot be modified. Be sure to use a defined table.

[Register status after calling this function]

Return value: C

Corrupted registers: AX (argument), BC (argument)

[Argument]

fal_descriptor_t [Note 1] Details (Defined Fixed Values Cannot Be Changed by Users)

| Argument | Type | Description |
|---|---|---|
| fal_pool_first_addr_u32 | fal_u32 | Data flash memory start address |
| eel_pool_first_addr_u32 | fal_u32 | Data flash memory start address used for EEL [Note 2] |
| user_pool_first_addr_u32 | fal_u32 | Unused |
| fal_pool_last_addr_u32 | fal_u32 | Data flash memory end address |
| eel_pool_last_addr_u32 | fal_u32 | Data flash memory end address used for EEL [Note 2] |
| user_pool_last_addr_u32 | fal_u32 | Unused |
| fal_pool_first_block_u16 | fal_u16 | Data flash memory first block number |
| eel_pool_first_block_u16 | fal_u16 | Data flash memory first block number used for EEL [Note 2] |
| user_pool_first_block_u16 | fal_u16 | Unused |
| fal_pool_last_block_u16 | fal_u16 | Data flash memory last block number |
| eel_pool_last_block_u16 | fal_u16 | Data flash memory last block number used for EEL [Note 2] |
| user_pool_last_block_u16 | fal_u16 | Unused |
| fal_first_widx_u16 | fal_u16 | Data flash memory access start number |
| eel_first_widx_u16 | fal_u16 | Data flash memory access start number used for EEL [Note 2] |
| user_first_widx_u16 | fal_u16 | Unused |
| fal_last_widx_u16 | fal_u16 | Data flash memory access end number |
| eel_last_widx_u16 | fal_u16 | Data flash memory access end number used for EEL [Note 2] |
| user_last_widx_u16 | fal_u16 | Unused |
| fal_pool_wsize_u16 | fal_u16 | Area size of all blocks (word units: 4-byte units) |
| eel_pool_wsize_u16 | fal_u16 | Area size of all blocks used for EEL [Note 2] (word units: 4-byte units) |
| user_pool_wsize_u16 | fal_u16 | Unused |
| block_size_u16 | fal_u16 | Area size of one block (byte units) |
| block_wsize_u16 | fal_u16 | Area size of one block (word units: 4-byte units) |
| fal_pool_size_u08 | fal_u08 | Data flash memory block size |
| eel_pool_size_u08 | fal_u08 | Data flash memory block size used for EEL [Note 2] |
| user_pool_size_u08 | fal_u08 | Unused |
| fx_MHz_u08 | fal_u08 | Setting of operation frequency of RL78 microcontroller |
| wide_voltage_mode_u08 | fal_u08 | Setting of voltage mode |

**Notes 1.** Users must not modify this defined table.

**2.** EEL: This stands for EEPROM emulation library.

| | Argument Type/Register | |
|---|---|---|
| | C Language | Assembly Language |
| RENESAS Small and medium model | const __far fal_descriptor_t* descriptor_pstr | AX (0 to 15), BC (16 to 23) |
| RENESAS Large model | const __far fal_descriptor_t* descriptor_pstr | AX (0 to 15), BC (16 to 23) |

[Return value]

<R>

| Type | Symbol Definition | Description |
|---|---|---|
| fal_status_t | FAL_OK | Normal end |
| | FAL_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. |

**Remark** Assembly language return values are stored in register C.

# 3. 2 EEPROM Emulation Library Functions

The EEPROM emulation library consists of the following library functions.

**Table 3-2. EEPROM Emulation Library Functions**

| Function Name | Overview |
|---|---|
| EEL_Init | Processing to initialize the RAM used for EEPROM emulation |
| EEL_Open | EEPROM emulation preparation processing |
| EEL_Close | EEPROM emulation end processing |
| EEL_Execute | EEPROM emulation command execution processing |
| EEL_Handler | Processing to continue executing EEPROM emulation commands or block adjustment processing<br>* This is used for command execution in modes other than the enforced mode. |
| EEL_TimeOut_CountDown | Timeout counting processing for EEPROM emulation command execution<br>* This is used only in the timeout mode. |
| EEL_GetDriverStatus | This obtains the EEPROM emulation library status. |
| EEL_GetSpace | This obtains the status indicating the number of free EEPROM emulation blocks. |
| EEL_GetVersionString | This obtains the version information of the EEPROM emulation library (EEL). |

# EEL_Init

[Outline]

Processing to initialize the RAM used for EEPROM emulation

[Format]

<C language>

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

<Assembler>

```
CALL !_EEL_Init or CALL !!_EEL_Init
```

**Remark** Call this function by using ! if placing the EEPROM emulation library at 00000H to 0FFFFH or by using !! if not.

[Advance setting]

1. The flash self programming library and EEPROM emulation library processing must be either not executing or finished.
2. The FAL_Init function must have finished normally.

[Function]

This function initializes parameters used to execute EEPROM emulation.

**Cautions 1.** When starting EEPROM emulation, always execute this function to initialize the RAM to be used.

**2.** This function is mutually exclusive with the flash self programming library (FSL). Before executing this function, be sure to close the flash self programming library. Also, never use any flash self programming library functions during EEPROM emulation.

**3.** To use the flash self programming library after this function is executed, the RAM must be reinitialized, so always execute this function when restarting the EEPROM emulation library.

**4.** To execute this function again, always close the EEPROM emulation library.

[Register status after calling this function]

Return value: C

[Argument]

None

[Return value]

| Type | Symbol Definition | Description |
|---|---|---|
| eel_status_t | EEL_OK | Normal end |
| | EEL_ERR_CONFIGURATION | Initialization error. Either the FAL_Init function is not being executed or the value specified for FAL_Init and EEL_Init functions cannot be used to execute EEL processing. |

**Remark** Assembly language return values are stored in register C.

# EEL_Open

[Outline]

EEPROM emulation preparation processing

[Format]

<C language>

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

<Assembler>

```
CALL !_EEL_Open or CALL !!_EEL_Open
```

**Remark** Call this function by using ! if placing the EEPROM emulation library at 00000H to 0FFFFH or by using !! if not.

[Advance setting]

1. The FAL_Init and EEL_Init functions must have finished normally.
2. If EEPROM emulation was executed, the processing up to EEL_Close must be executed to stop the processing (closed status).

[Function]

This function changes the system to a status in which the data flash memory can be manipulated to make it possible to execute EEPROM emulation.

**Note** After executing the EEL_Open function and switching to the EEPROM emulation start status (opened), flash self programming library processing cannot be executed. It also becomes impossible to execute STOP mode and HALT mode processing. If it is necessary to execute flash self programming library, STOP mode, or HALT mode processing, execute the EEL_Close function to switch EEPROM emulation to the stopped status (closed).

[Register status after calling this function]

No registers are corrupted.

[Argument]

None

[Return value]

None

# EEL_Close

[Outline]

EEPROM emulation end processing

[Format]

<C language>

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

<Assembler>

```
CALL !_EEL_Close or CALL !!_EEL_Close
```

**Remark** Call this function by using ! if placing the EEPROM emulation library at 00000H to 0FFFFH or by using !! if not.

[Advance setting]

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

[Function]

This function changes the flash memory to the operation completion status to make it impossible to execute EEPROM emulation.

[Register status after calling this function]

No registers are corrupted.

[Argument]

None

[Return value]

None

# EEL_Execute

[Outline]

EEPROM emulation execution function

[Format]

<C language>

```
void  __far EEL_Execute( eel_request_t* request )
```

<Assembler>

```
CALL !_EEL_Execute or CALL !!_EEL_Execute
```

**Remark** Call this function by using ! if placing the EEPROM emulation library at 00000H to 0FFFFH or by using !! if not.

[Advance setting]

The FAL_Init, EEL_Init, and EEL_Open functions must have finished normally.

[Function]

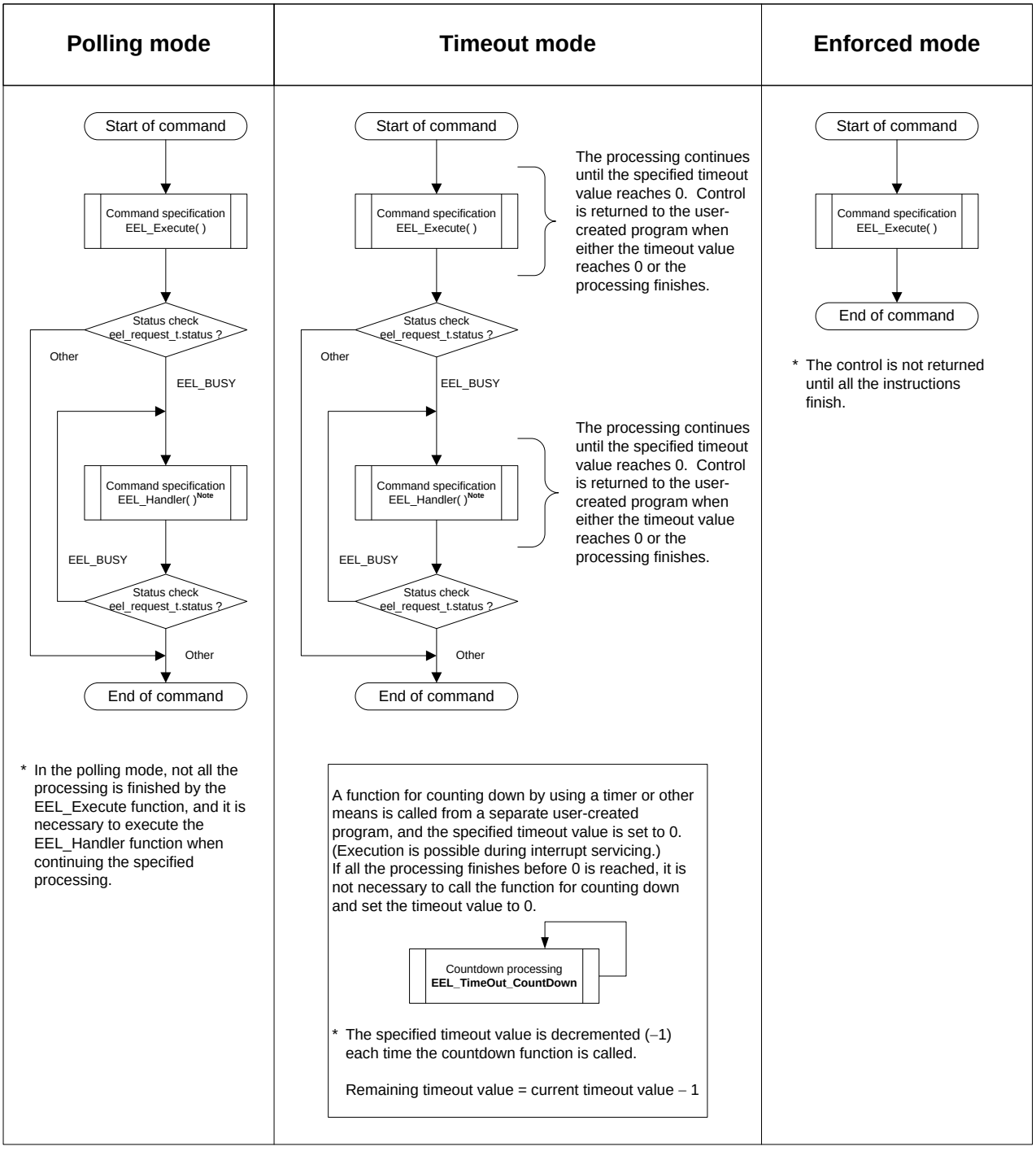
Each type of processing for performing EEPROM emulation operations is specified for this function as an argument in the command format, and the processing is executed. When executing the EEL_Execute function, it is possible to specify the execution mode and select the EEPROM emulation execution method. Table 3-3 and Figure 3-1 show the status of each mode.

**Table 3-3. Command Execution Method in Each Mode**

| Execution Mode | Description |
|---|---|
| Enforced mode | All the processing of specified commands is executed using only the EEL_Execute function. The control does not return from the function until all the command processing finishes. |
| Timeout mode [Note] | The processing of specified commands is continuously executed until the timeout value specified for the EEL_Execute function is set to 0 by the EEL_TimeOut_CountDown function. If there is remaining processing when the timeout occurs, the processing is continued by using the EEL_Handler function.<br>If all the processing finishes before the timeout value reaches 0, the function terminates and the control returns to the user-created program, but, when using this mode, be sure to use the EEL_TimeOut_CountDown function to make it possible for the timeout value to reach 0. |
| Polling mode [Note] | The processing of specified commands is executed using a certain unit. After executing the EEL_Execute function, the processing is continued by using the EEL_Handler function. |

**Note** The execution mode can be re-specified when executing EEL_Handler.

**Figure 3-1. Command Execution Method in Each Mode**




**Note** The execution mode can be re-specified when executing EEL_Handler.

[Register status after calling this function]
Corrupted register: AX (argument)

[Argument]

eel_request_t Details

| Argument | Type | Description |
|---|---|---|
| eel_request_t.address_pu08 | eel_u08 * | Data buffer for storing write and read data Note |
| eel_request_t.identifier_u08 | eel_u08 | Parameter for setting command to be executed |
| eel_request_t.timeout_u08 | eel_u08 | Timeout value (command execution mode setting) |
| eel_request_t.command_enu | eel_command_t | Command to be executed |
| 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.

Timeout Value (timeout_u08) Setting Details

| Timeout Value | Description |
|---|---|
| 0xFF | Processing is executed in the enforced mode. |
| 0x01 to 0xFE | Processing is executed in the timeout mode. |
| 0x00 | Processing is executed in the polling mode. |

Execution Commands (eel_command_t)

| Command | Description |
|---|---|
| EEL_CMD_STARTUP | This checks the block status and sets the system to the EEPROM emulation start (started) status. If all the blocks are active, the oldest active block is forcibly erased to create a prepared block.<br>Be sure to execute this command before executing commands other than the EEL_CMD_FORMAT command, which is used when initializing (erasing) Note 2 EEPROM emulation blocks, and make sure that the command finishes normally. |
| EEL_CMD_WRITE Note 1 | This writes the specified data to the EEPROM emulation blocks.<br>* The following arguments must be specified prior to execution.<br>• eel_request_t.address: Specifies the start address of the RAM area where the write data is stored.<br>• eel_request_t.identifier: Specifies the data ID of the write data. |
| EEL_CMD_READ Note 1 | This reads the specified data from the EEPROM emulation blocks.<br>* The following arguments must be specified prior to execution.<br>• eel_request_t.address: Specifies the start address of the RAM area where the read data is stored.<br>• eel_request_t.identifier: Specifies the data ID of the read data. |
| EEL_CMD_CLEANUP Notes 1, 3 | This moves only the latest EEPROM emulation block data to the newly created active block and initializes (erases) Note 2 all the other blocks, which are unnecessary.<br>If all the blocks are active, the oldest active block is forcibly erased to create a prepared block, and then the processing is executed. |

| Command | Description |
|---|---|
| EEL_CMD_FORMAT Note 4 | This initializes (erases) Note 2 everything, including the data recorded in the EEPROM emulation blocks. Be sure to use this command before using EEPROM emulation for the first time. Note that it is necessary to use this command to initialize all the blocks if an EEPROM emulation block abnormality occurs (such as an active block disappearing) or when modifying the initial values (fixed values that cannot be changed).<br>Because EEPROM emulation switches to the stopped status (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 sets EEPROM emulation to the stopped status (opened). |

**Notes 1.** Do not execute this command until the EEL_CMD_STARTUP command has finished normally.
**2.** Blocks for which usage is prohibited are not initialized (erased).
**3.** For the detailed operations, see **1.4.1 Adjusting blocks by using EEL_CMD_CLEANUP command**.
**4.** For the detailed operations, see **1.3 Initializing EEPROM Emulation Blocks**.

| | Argument Type/Register | |
|---|---|---|
| | C Language | Assembly Language |
| RENESAS Small and medium model | eel_request_t* request | AX (0 to 15) |
| RENESAS Large model | eel_request_t* request | AX (0 to 15) |

Command Execution Statuses (eel_status_t)

| Command Execution Status | Description | Corresponding Commands |
|---|---|---|
| EEL_OK | Normal end | All commands |
| EEL_BUSY | A command is being executed. | All commands |
| EEL_ERR_INITIALIZATION | Initialization error:<br>The FAL_Init(), EEL_Init(), or EEL_Open function has not finished normally. | All commands |
| EEL_ERR_ACCESS_LOCKED | EEPROM emulation lock error:<br>EEPROM emulation cannot be executed. Make sure that the EEL_CMD_STARTUP command has finished normally. | Commands other than EEL_CMD_STARTUP |
| EEL_ERR_COMMAND | Command error:<br>A command that does not exist has been specified. | – |
| EEL_ERR_PARAMETER | Parameter error:<br>An incorrect command parameter has been specified. Revise the specified parameter. | All commands |
| EEL_ERR_REJECTED | Reject error:<br>A different command is being executed. | All commands |
| <R> EEL_ERR_NO_INSTANCE | Identifier error:<br>The specified data is not in the descriptor table. Or data are not recorded. | EEL_CMD_READ |

| Command Execution Status | Description | Corresponding Commands |
|---|---|---|
| EEL_ERR_POOL_FULL | Pool full error:<br>There is no area that can be used to write the data. This error can be recovered from by using the EEL_CMD_STARTUP or EEL_CMD_CLEANUP command to forcibly erase the oldest block, but some of the data might disappear even if this is done. | EEL_CMD_WRITE |
| EEL_ERR_POOL_INCONSISTENT | EEPROM emulation block inconsistency error:<br>An EEPROM emulation block has the undefined status (such as because there are no active blocks). Execute the EEL_CMD_FORMAT command to initialize the EEPROM emulation blocks. | EEL_CMD_STARTUP |
| EEL_ERR_POOL_EXHAUSTED | EEPROM emulation block exhaustion error:<br>There are no more EEPROM emulation blocks that can be used to continue. Stop EEPROM emulation. | Commands other than EEL_CMD_READ and EEL_CMD_SHUTDOWN |
| EEL_ERR_INTERNAL | Internal error:<br>An unexpected error has occurred. Check the device status. In addition, if this error occurred during EEL_CMD_SHUTDOWN command execution, execute the EEL_Close function to stop EEPROM emulation. | Commands other than EEL_CMD_READ and EEL_CMD_WRITE |

[Return value]

None

# EEL_Handler

[Outline]

Processing to continue executing EEPROM emulation when EEL_Execute is executed in a mode other than the enforced mode, or the maintenance mode execution processing

[Format]

<C language>

```
void  __far EEL_Handler(eel_u08 timeout_u08);
```

<Assembler>

```
CALL !_EEL_Handler or CALL !!_EEL_Handler
```

**Remark** Call this function by using ! if placing the EEPROM emulation library at 00000H to 0FFFFH or by using !! if not.

[Advance setting]

The FAL_Init, EEL_Init, and EEL_Open functions must have finished normally.

[Function]

This function continues executing the EEPROM emulation processing specified for the EEL_Execute function. Note 1 In addition, by executing this function while no commands are being executed, it is possible to adjust EEPROM emulation blocks by using the maintenance mode. Note 2

**Notes 1.** The command execution status for the EEL_Handler function is specified for eel_request_t* request, which is used as the EEL_Execute function argument. Therefore, when using EEL_Handler, do not free the eel_request_t* request variable.

**2.** For details about the maintenance mode, see **1.4.2 Adjusting blocks by using EEL_Handler function**.

[Register status after calling this function]

Corrupted register: AX (argument)

[Argument]

Timeout Value (timeout_u08) Setting Details [Note]

| Timeout Value | Description |
|---|---|
| 0x01 to 0xFF | Executing processing continues in the timeout mode.<br>(It is necessary to use the EEL_TimeOut_CountDown function to set the timeout value to 0.) |
| 0x00 | Executing processing continues in the polling mode. |

**Note** For details about each mode, see the description of the EEL_Execute function.

| Development tool | Argument Type/Register | |
|---|---|---|
| | C Language | Assembly Language |
| RENESAS Small and medium model | eel_u08 timeout_u08 | X |
| RENESAS Large model | eel_u08 timeout_u08 | X |

[Return value]

Post-execution status information is specified for eel_request_t* request, an argument of the EEL_Execute function.

# EEL_TimeOut_CountDown

[Outline]

Timeout counting processing for EEPROM emulation command execution

[Format]

<C language>

```
void  __far EEL_TimeOut_CountDown(void)
```

<Assembler>

```
CALL !_EEL_TimeOut_CountDown or CALL !!_EEL_TimeOut_CountDown
```

**Remark** Call this function by using ! if placing the EEPROM emulation library at 00000H to 0FFFFH or by using !! if not.

[Advance setting]

The FAL_Init, EEL_Init, and EEL_Open functions must have finished normally.

[Function]

This function is used when executing EEPROM emulation in the timeout mode.

When this function is executed, the timeout value specified when executing an EEPROM emulation command is decremented (−1), and the EEL_Execute and EEL_Handler function loop processing ends when the timeout value reaches 0.

For details about handling the timeout value, see the sections that describe the EEL_Execute and EEL_Handler functions.

[Register status after calling this function]

No registers are corrupted.

[Argument]

None

[Return value]

None

# EEL_GetDriverStatus

[Outline]

This obtains the EEPROM emulation library status.

[Format]

<C language>

```
void  __far EEL_GetDriverStatus(__near eel_driver_status_t *driverStatus_pstr)
```

<Assembler>

```
CALL !_EEL_GetDriverStatus or CALL !!_EEL_GetDriverStatus
```

**Remark** Call this function by using ! if placing the EEPROM emulation library at 00000H to 0FFFFH or by using !! if not.

[Advance setting]

The FAL_Init and EEL_Init functions must have finished normally.

[Function]

This function obtains the status of the EEPROM emulation library. This makes it possible to check the status of the EEL_Handler function and EEPROM emulation.

[Register status after calling this function]

No registers are corrupted.

[Argument]

eel_driver_status_t Details

| Argument | Type | Description |
|---|---|---|
| eel_driver_status_t.operationStatus_enu | eel_operation_status_t | EEL_Handler() execution status |
| eel_driver_status_t.accessStatus_enu | eel_access_status_t | EEPROM emulation status |
| eel_driver_status_t.backgroundStatus_enu | eel_status_t | Library status (not available to users) |

eel_operation_status_t Details

| Type | Description |
|---|---|
| EEL_OPERATION_PASSIVE | The EEL_CMD_STARTUP command has not finished normally and no commands are being executed. |
| EEL_OPERATION_IDLE | No command or maintenance mode[Note] processing is being executed. |
| EEL_OPERATION_BUSY | Command or maintenance mode[Note] processing is being executed. |

**Note** For details about the maintenance mode, see **1.4.2 Adjusting blocks by using EEL_Handler function**.

eel_access_status_t Details

| Type | Description |
|---|---|
| EEL_ACCESS_LOCKED | Data reading and writing cannot be executed (opened, closed). |
| EEL_ACCESS_UNLOCKED | Data reading and writing can be executed (started). |

| Development tool | Argument Type/Register | |
|---|---|---|
| | C Language | Assembly Language |
| RENESAS Small and medium model | __near eel_driver_status_t *driverStatus_pstr | AX (0 to 15) |
| RENESAS Large model | __near eel_driver_status_t *driverStatus_pstr | AX (0 to 15) |

[Return value]

None

# EEL_GetSpace

[Outline]

This obtains the free EEPROM emulation block space (in word units).

[Format]

<C language>

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

<Assembler>

```
CALL !_EEL_GetSpace or CALL !!_EEL_GetSpace
```

**Remark** Call this function by using ! if placing the EEPROM emulation library at 00000H to 0FFFFH or by using !! if not.

[Advance setting]

The FAL_Init, EEL_Init, and EEL_Open functions, and the EEL_CMD_STARTUP command must have finished normally.

[Function]

This function obtains the free EEPROM emulation block space (in word units).

[Register status after calling this function]

Return value: C

Corrupted register: AX (argument)

[Argument]

| Type | Description |
|---|---|
| __near eel_u16* | The address at which the information about the total free space of the current active block and prepared blocks is input (a 2-byte area): The data is in word units (1 word = 4 bytes). |

| Development tool | Argument Type/Register | |
|---|---|---|
| | C Language | Assembly Language |
| RENESAS Small and medium model | __near eel_u16* space_pu16 | AX (0 to 15) |
| RENESAS Large model | __near eel_u16* space_pu16 | AX (0 to 15) |

[Return value]

| Type | Symbol Definition | Description |
|---|---|---|
| eel_status_t | EEL_OK | Acquisition successful |
| | 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. |

**Remark** Assembly language return values are stored in register C.

# EEL_GetVersionString

[Outline]

This obtains the version information of the EEPROM emulation library (EEL).

[Format]

<C language>

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

<Assembler>

```
CALL !_EEL_GetVersionString or CALL !!_EEL_GetVersionString
```

**Remark** Call this function by using ! if placing the EEPROM emulation library at 00000H to 0FFFFH or by using !! if not.

[Advance setting]

None

[Function]

This function obtains the version information of the EEPROM emulation library (EEL).

[Register status after calling this function]

Return value: BC, DE

Corrupted registers: BC, DE

[Argument]

None

[Return value]

<R>

| Type | Description |
|---|---|
| eel_u08* | The address at which the version information of the EEPROM emulation library (EEL) is input (a 24-bit address area)<br>Example: For EEPROM emulation library Pack01 V1.00 (ASCII code)<br><br>“ERL78T01R110GVxxx”<br>Version information: V110 → V1.10<br>Corresponding tool: Renesas Electronics version<br>Type name: Type 01<br>Corresponding device: RL78<br>Target library: EEL |

# APPENDIX A REVISION HISTORY

## A. 1 Major Revisions in This Edition

| Page | Description | Classification |
|---|---|---|
| Throughout the document | | |
| - | The document on the data flash library, which was classified as the application note (old version of R01AN0351), was changed to the user's manual. | (d) |
| - | The corresponding ZIP file name and release version were added to the cover page. | (d) |
| - | Contents of the processing time and software resources were moved from the usage note to this document. Accordingly, the reference destination described in this document was also changed. | (d) |
| - | The supported device was added. | (b) |
| - | The notation of high-speed OCO was deleted to unify the notation of high-speed on-chip oscillator. | (d) |
| - | The description of the operating frequency was unified to the CPU operating frequency since individual descriptions had different notations. | (d) |
| Chapter 1 Overview of EEPROM emulation | | |
| p.12 and 20 | Modified errors in the capacity of the separator and the calculation method for the maximum size | (a) |
| Chapter 2 Using EEPROM emulation | | |
| p.36 | Note on the frequency of the high-speed on-chip oscillator was added. | (c) |
| p.36 | Note on the RAM parity error was added. | (c) |
| p.36 | Note on the data flash control register (DFLCTL) was added. | (c) |
| p.37 | Note on the number of data flash memory block was added. | (c) |
| p.38-40 | Items regarding the processing time were added (the description of the processing time was moved from the usage note to this document). | (c) |
| p.41-43 | Items regarding the resources were added and description was also changed.(the description on the resources was moved from the usage note to this document). | (c) |
| P.44 and 45 and 48 | Note on the high-speed on-chip oscillator was added. | (c) |
| Chapter 3 EEPROM Emulation features | | |
| p.50 | Description of the return value was added. | (c) |
| p.58 | Description of the argument] was added. | (c) |
| p.66 | Description of the return value was changed | (c) |

Remark "Classification" in the above table classifies revisions as follows.

(a): Error correction, (b): Addition/change of specifications, (c): Addition/change of description or note,

(d): Addition/change of package, part number, or management division,

(e): Addition/change of related documents

RL78 Microcontrollers User’s Manual: EEPROM Emulation Library Pack 01

Publication Date: Rev.1.02 Aug 02, 2013

Published by: Renesas Electronics Corporation

**SALES OFFICES** Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.


**Renesas Electronics America Inc.**
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

**Renesas Electronics Canada Limited**
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

**Renesas Electronics Europe Limited**
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

**Renesas Electronics Europe GmbH**
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

**Renesas Electronics (China) Co., Ltd.**
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

**Renesas Electronics (Shanghai) Co., Ltd.**
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

**Renesas Electronics Hong Kong Limited**
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

**Renesas Electronics Taiwan Co., Ltd.**
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

**Renesas Electronics Singapore Pte. Ltd.**
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

**Renesas Electronics Malaysia Sdn.Bhd.**
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

**Renesas Electronics Korea Co., Ltd.**
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141





Colophon 1.3

RL78 Family

Renesas Electronics Corporation

R01US0054EJ0102