User's Manual

16

RENESAS

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

Renesas Electronics
www.renesas.com

Rev.1.03 Jan 2016

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

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

1. Handling of Unused Pins
   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.
   - The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.
2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   - The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
     In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.
3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   - The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.
4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   - When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.
5. Differences between Products
   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.
   - The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

HOW TO USE THIS MANUAL

Readers This manual is intended for users who wish to understand the functions of the RL78 microcontrollers EEPROM Emulation Library Pack 01 and design and develop application systems and programs for these devices.

<R> For the target MCUs, refer to the RL78 Family Self RAM list of Flash Self Programming Library (R20UT2944EJxxxx).

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 areas 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

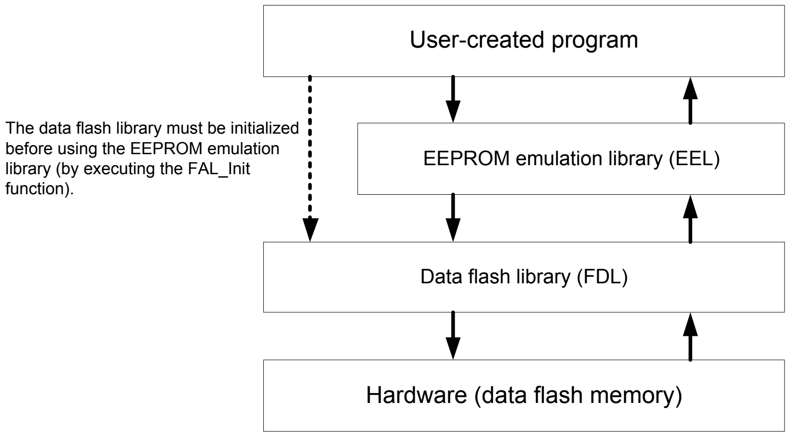


Figure 1-2. Example of Memory Map

- 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) are written in order to use the data flash memory as EEPROM emulation blocks.

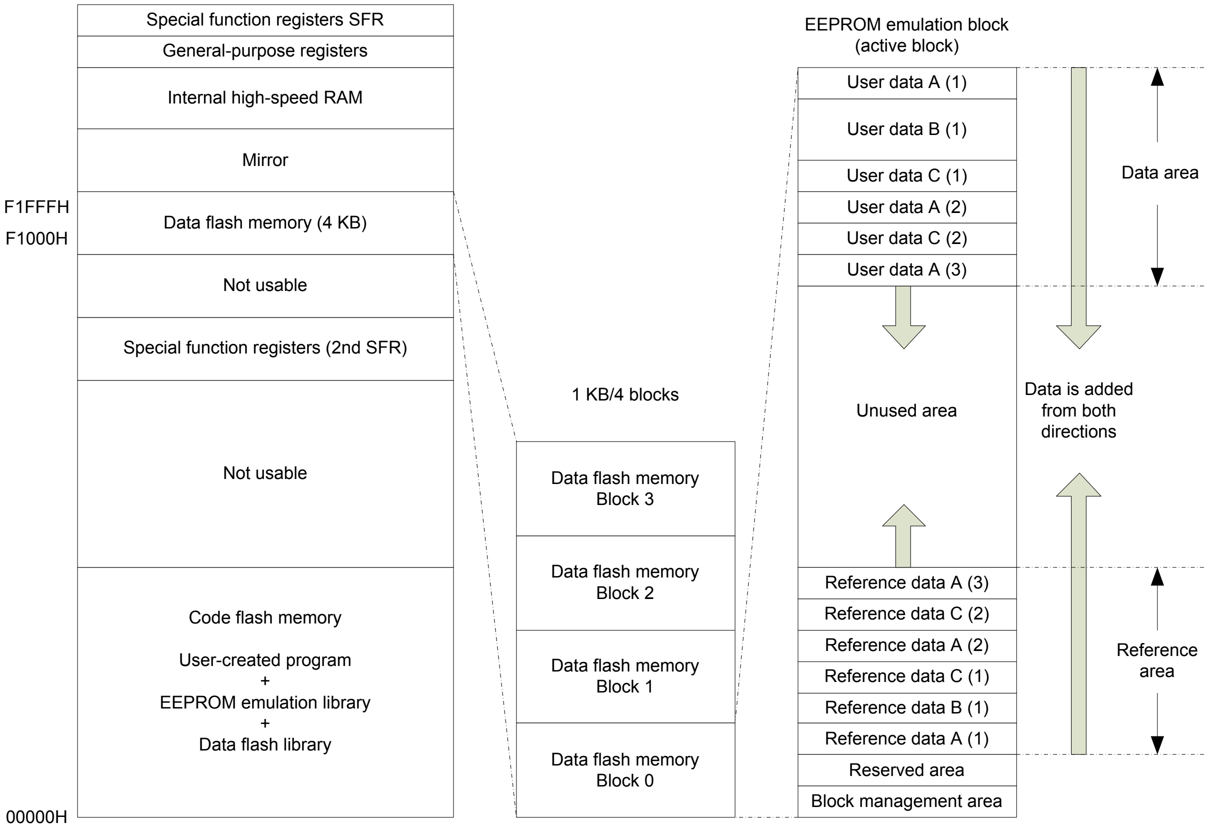


Figure 1-3. EEPROM Emulation Block Details

- 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.

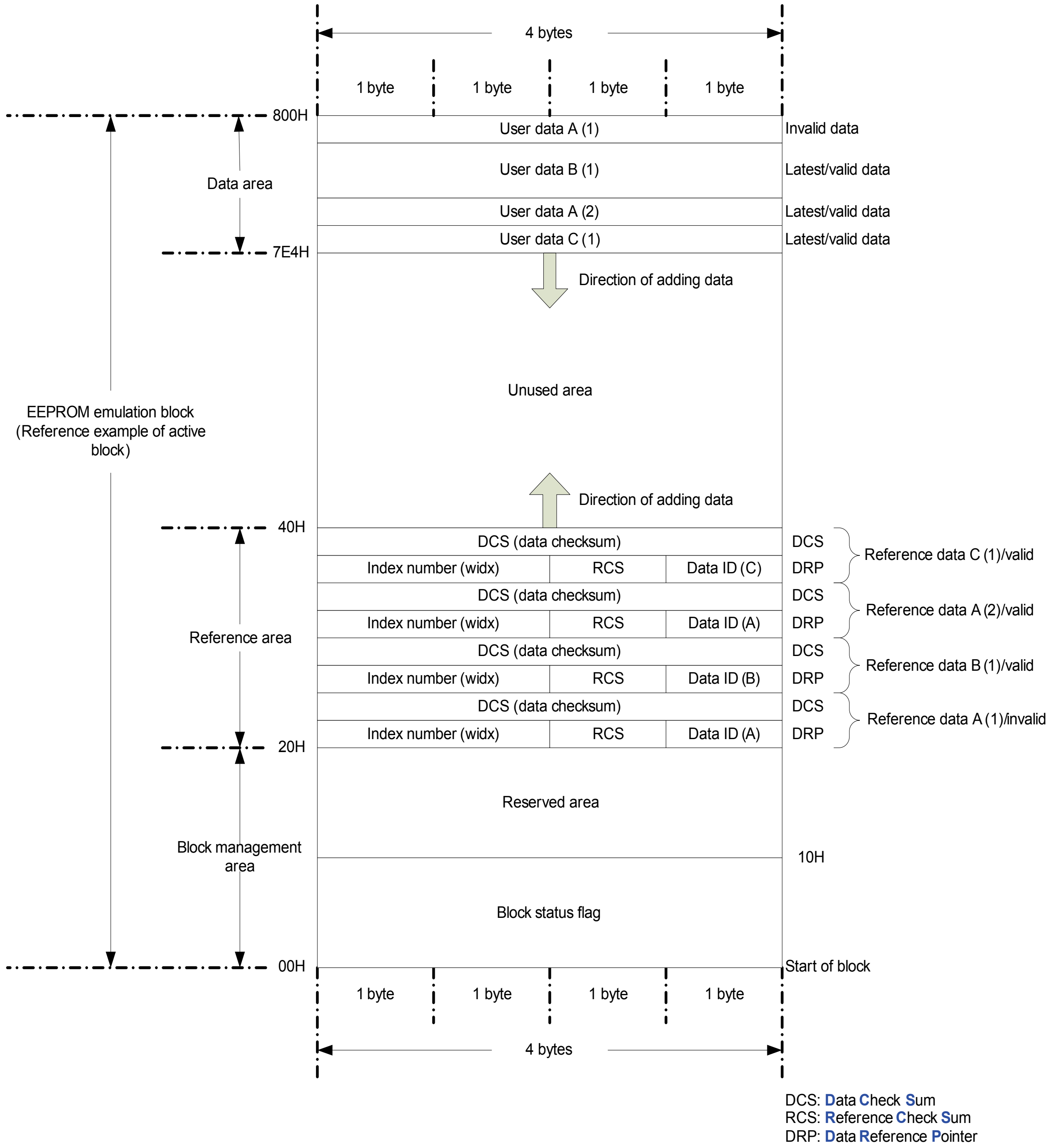


Figure 1-4. Example of Expanding Active EEPROM Emulation Block

- 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.

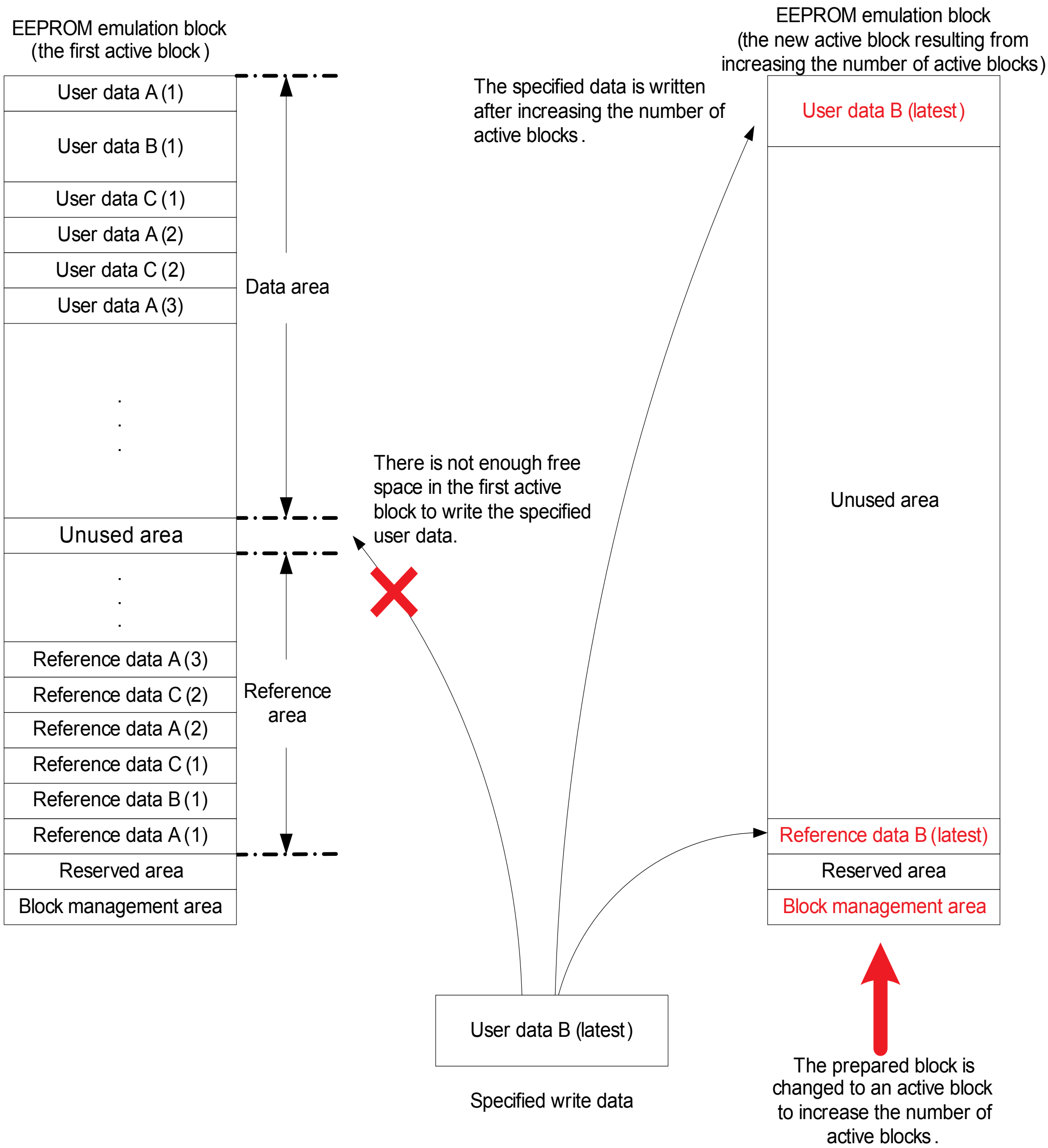
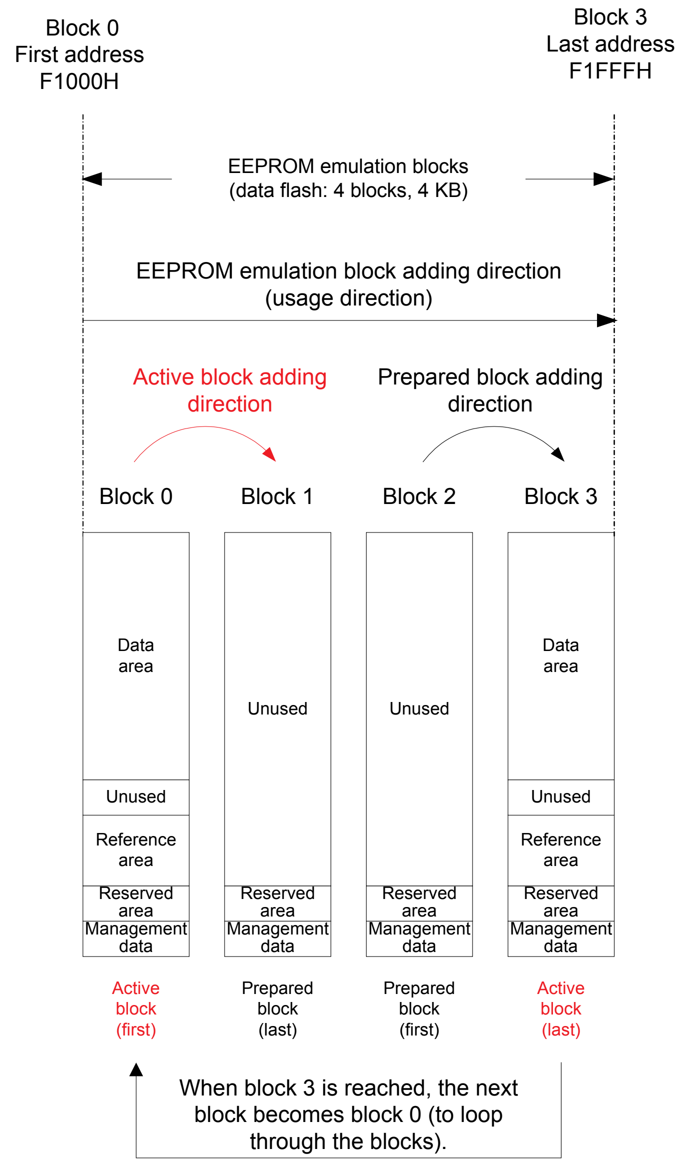


Figure 1-5. EEPROM Emulation Block Usage Method (Four Blocks)

- 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.

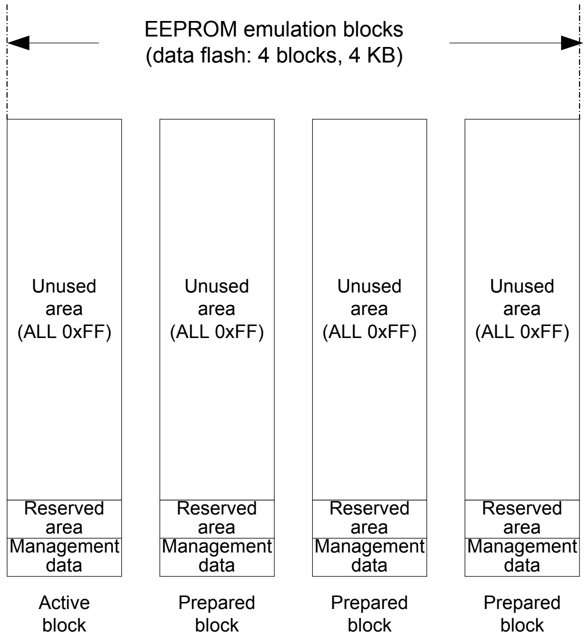


Figure 1-6 (B). EEPROM Emulation Block Transition Example

- 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.

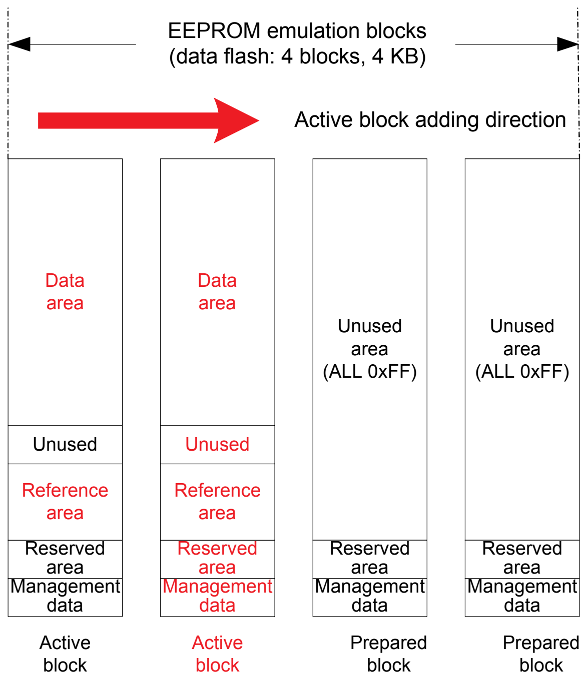


Figure 1-6 (C). EEPROM Emulation Block Transition Example

- 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.

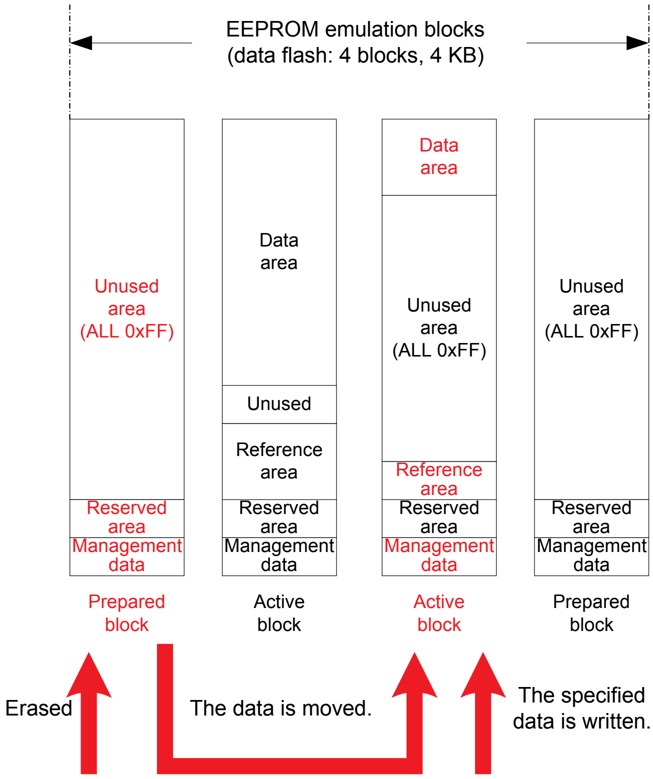


Table 1-1. Settings of Written Data and Usable Ranges

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	–
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.

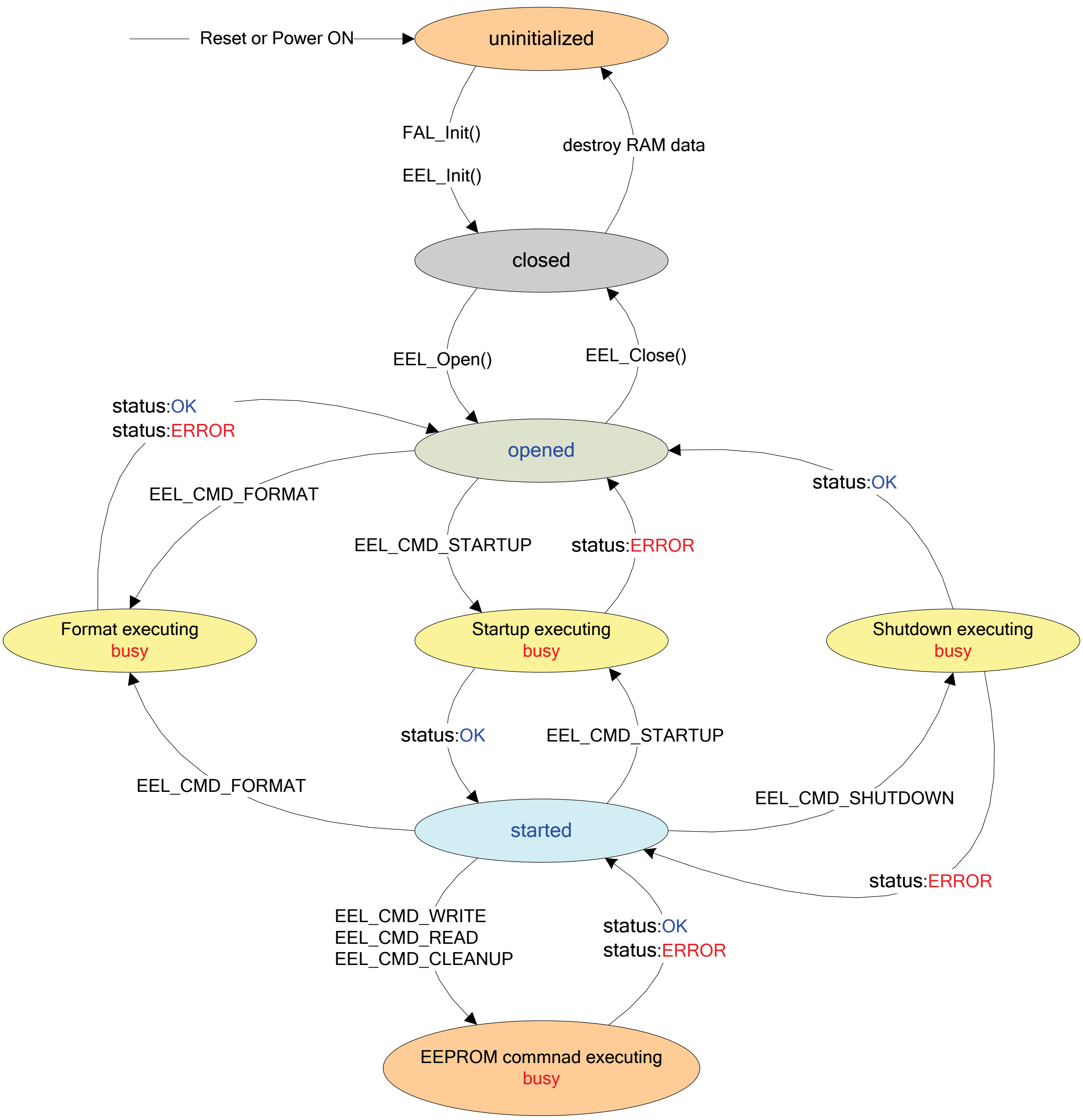
Notes 1. The total size of the user data must be such that it is possible to write all the data into within two EEPROM emulation blocks. Therefore, the range used for the number of stored user data items differs depending on the size of the stored user data. It is also necessary to consider the size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

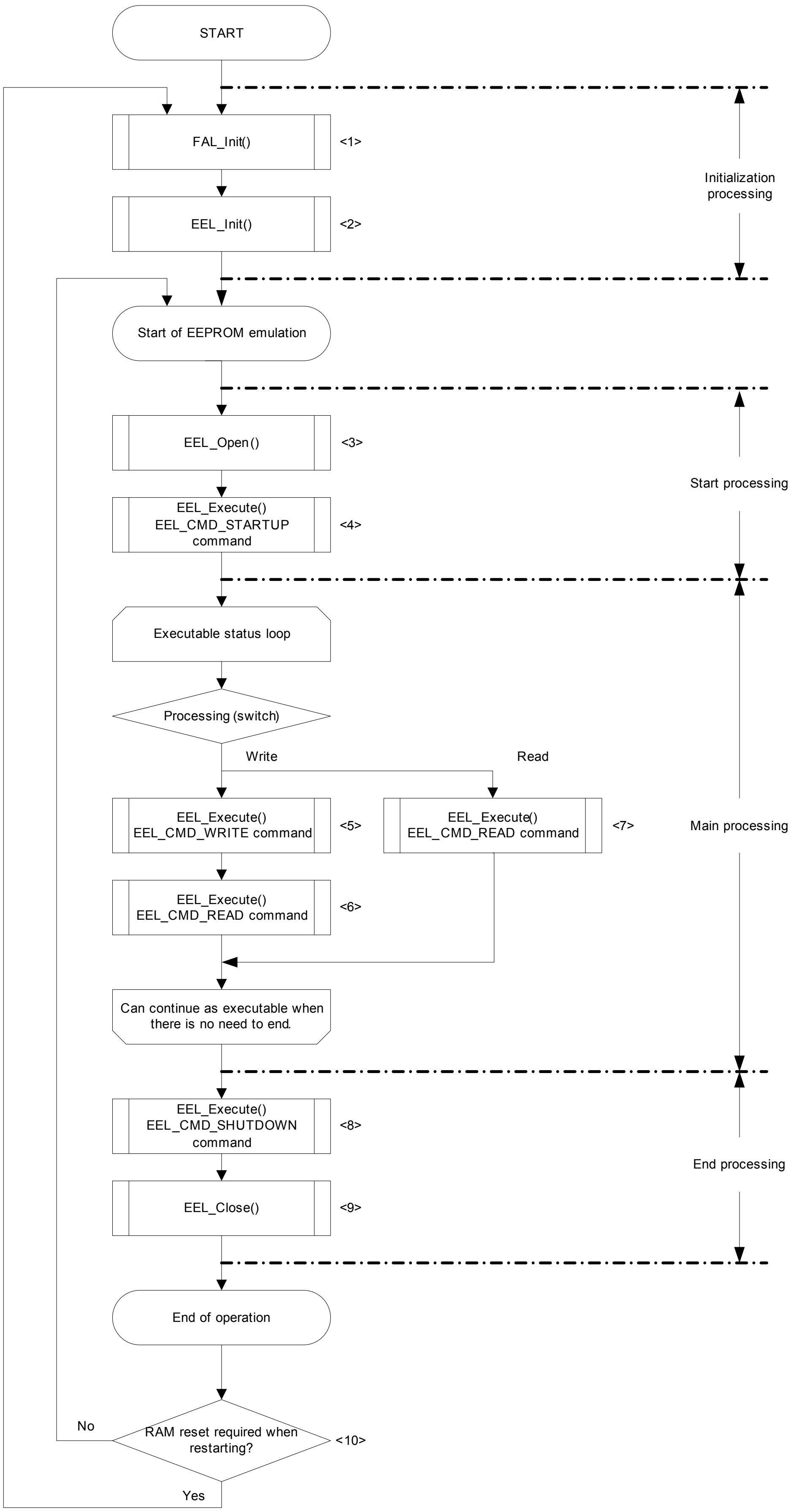
(4) started

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

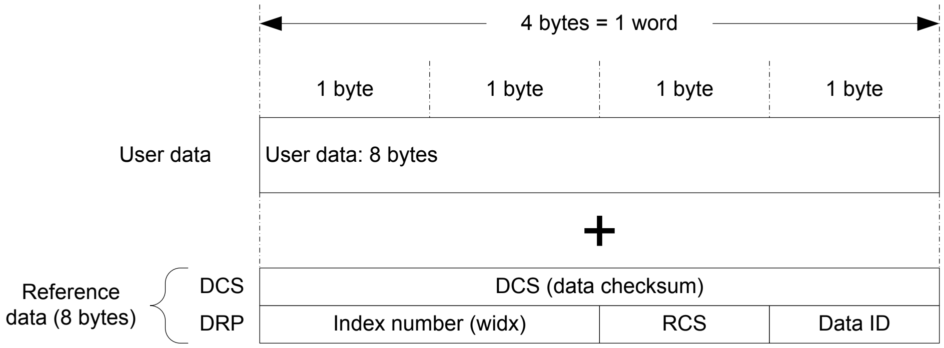
The EEPROM emulation library Pack01 uses four or more block data flash memory as EEPROM emulation blocks.

1. 2. 2 Data Structure

The data flash memory is written to in word (4 bytes) units. Therefore, when the EEPROM emulation library writes to the data flash memory, the data length is always adjusted to word (4 bytes) units.

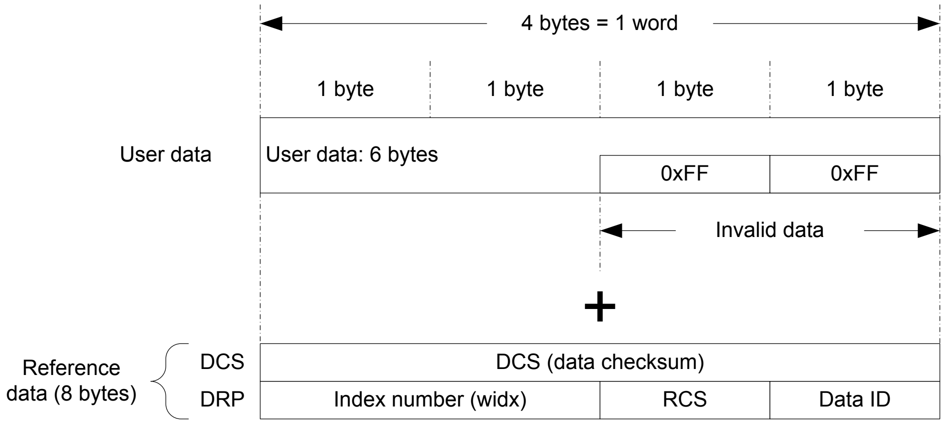
In addition, when writing user data, because the reference data for managing data is also written, when calculating the capacity required for writing, the 2 words (8 bytes) of management reference data must be added to the size of the user data in words. Figures 1-9 to 1-11 show an example of the data 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)



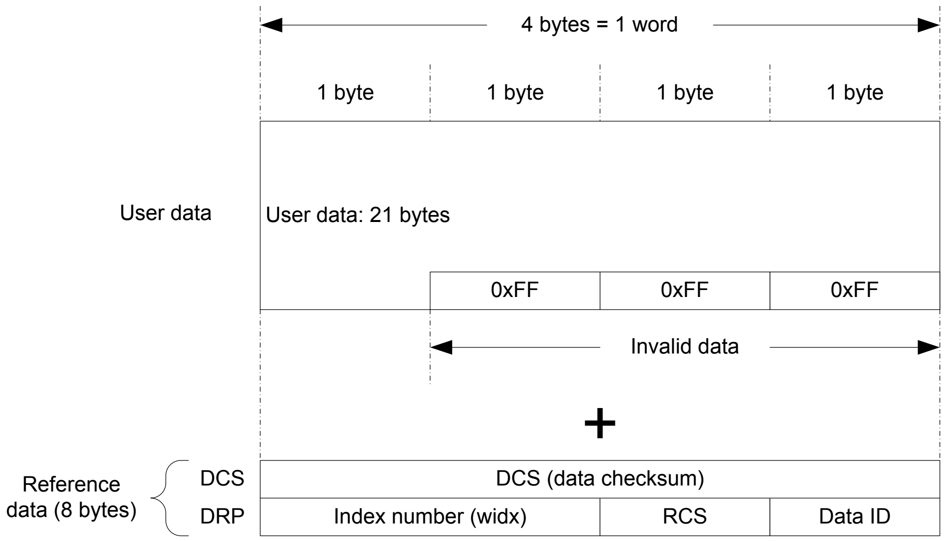
Total size: 4 words (16 bytes) = reference data: 2 words (8 bytes) + [user data (8 bytes)/word size (4 bytes)]
* Change the user data to word units.

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 **2.4 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, A, I, and X flags, each of which is 4 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

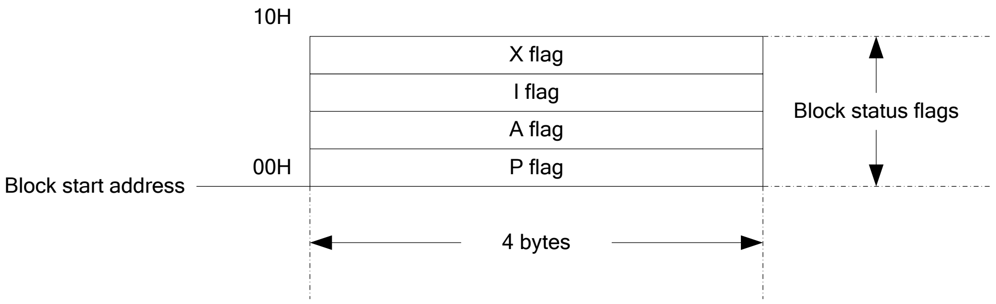


Table 1-2. Overview of Block Status Flags

<table>
<tr><th colspan="4">Block Status Flag</th><th rowspan="2">Status</th><th rowspan="2">Description</th></tr>
<tr><th>P Flag</th><th>A Flag</th><th>I Flag</th><th>X Flag</th></tr>
<tr><td>55555555H</td><td>FFFFFFFFH</td><td>FFFFFFFFH</td><td>FFFFFFFFH</td><td>Prepared</td><td>This block is ready to be written.</td></tr>
<tr><td>55555555H</td><td>55555555H</td><td>FFFFFFFFH</td><td>FFFFFFFFH</td><td>Active</td><td>This block is being used.</td></tr>
<tr><td>55555555H</td><td>55555555H</td><td>00000000H</td><td>FFFFFFFFH</td><td rowspan="2">Inactive</td><td rowspan="2">Inactive block</td></tr>
<tr><td colspan="3">Data other than the above</td><td>FFFFFFFFH</td></tr>
<tr><td>–</td><td>–</td><td>–</td><td>Other than FFFFFFFFH</td><td>Use prohibited[Note]</td><td>This block cannot be written.</td></tr>
</table>

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
Free space necessary as termination information (separator):	12 bytes

  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.

  Maximum size = 980 bytes × number of EEPROM emulation blocks × 1/4
  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

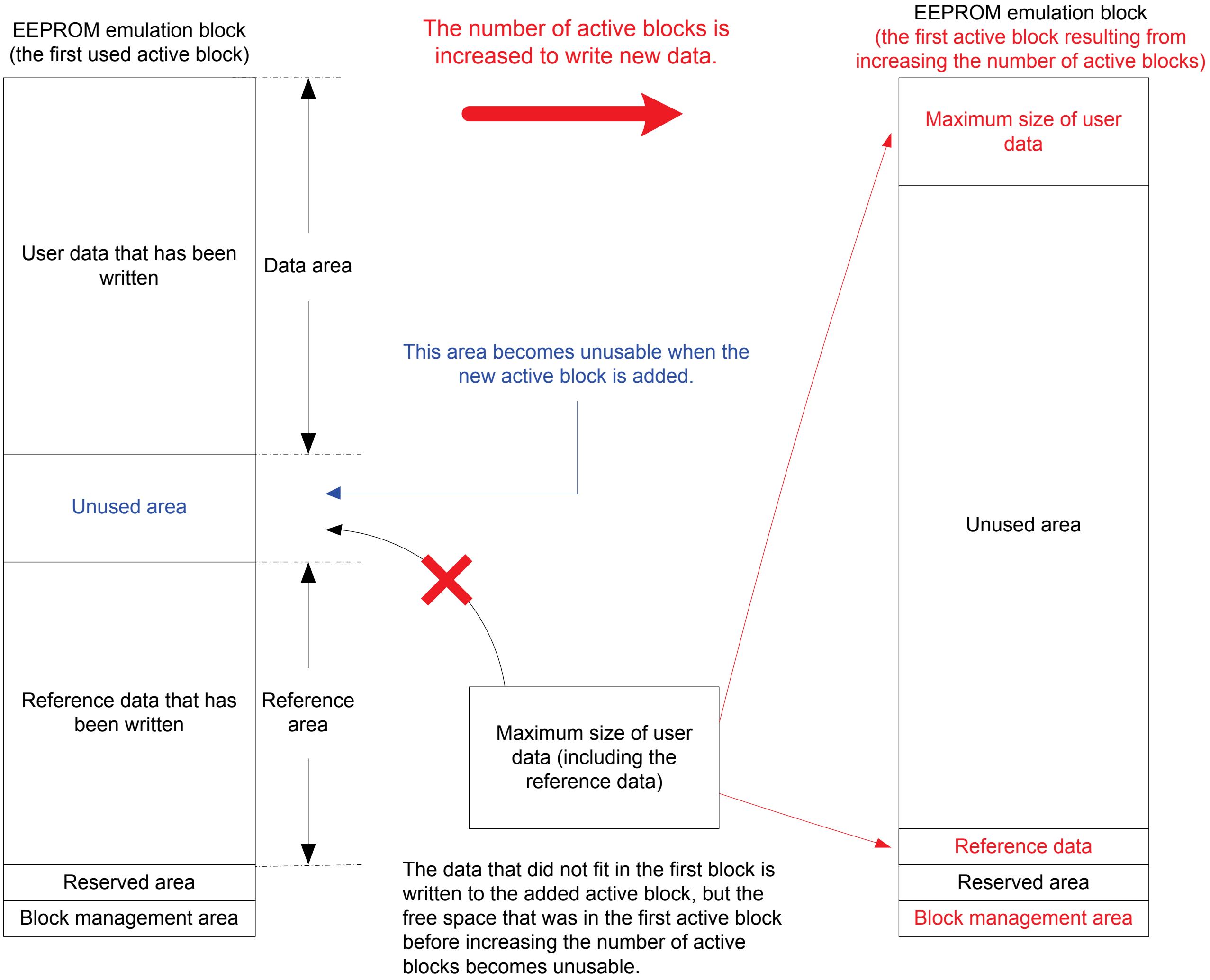
  If the basic total user data size exceeds 988 bytes, the maximum usable size of one block, it is necessary to include the maximum size that might become unusable in the calculation when increasing the number of active blocks.

  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 × (number of necessary blocks - 1)

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)

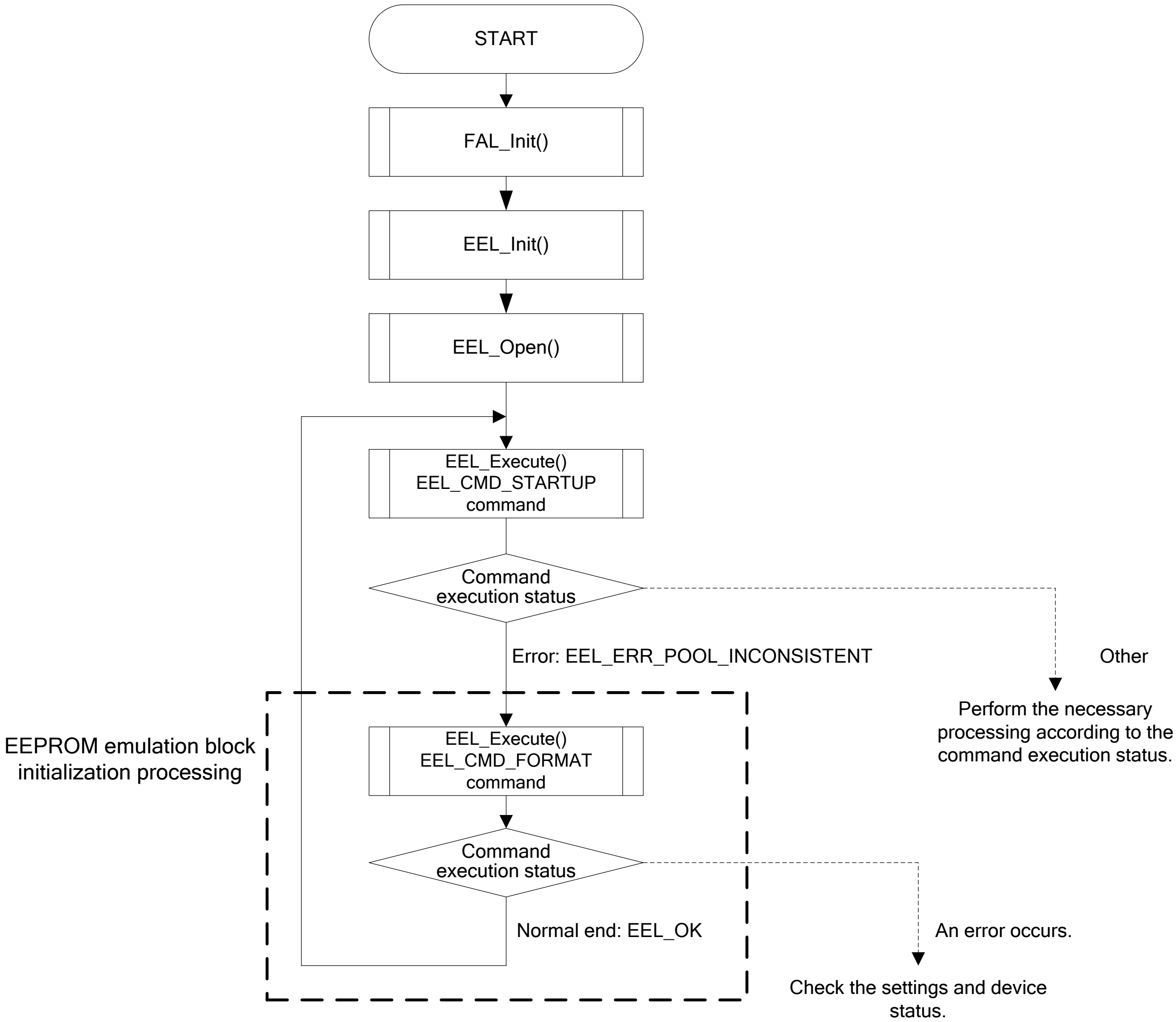


Figure 1-15. Example Block Status Transitions when Initialization is Performed upon Occurrence of EEL_ERR_POOL_INCONSISTENT Error

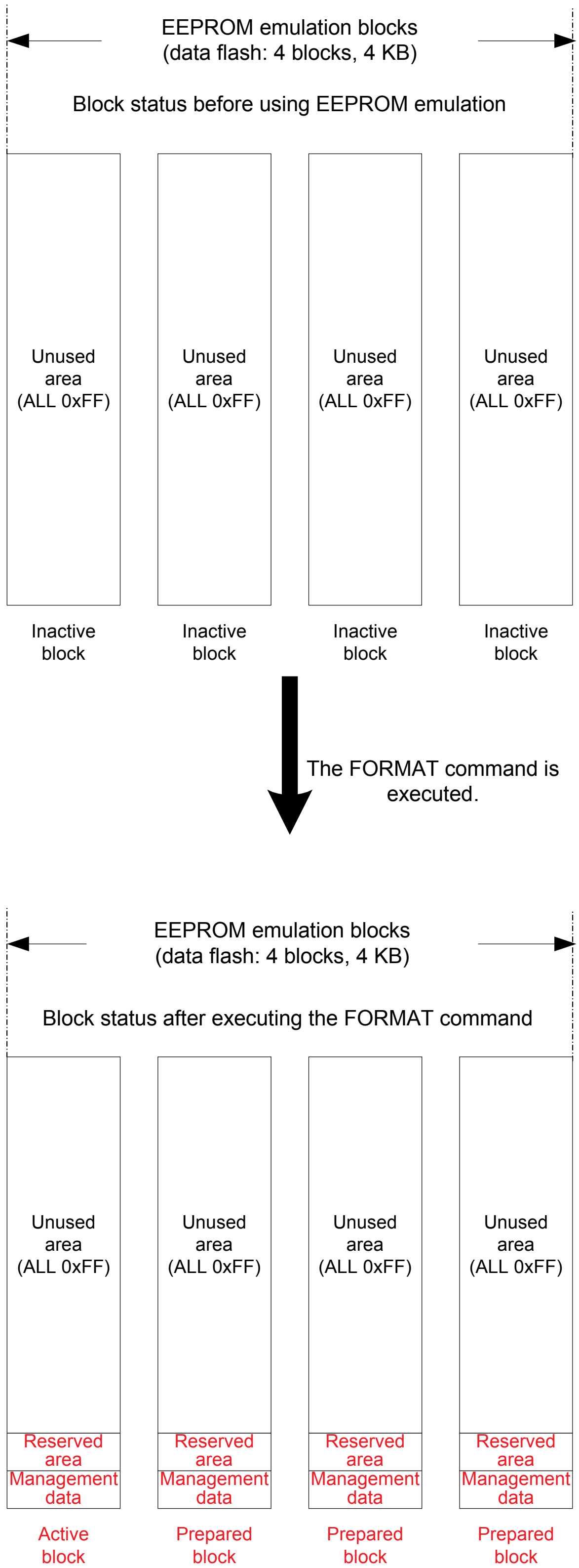


Figure 1-16. Flow when Performing Initialization at Arbitrary Time (when Using Enforced Mode)

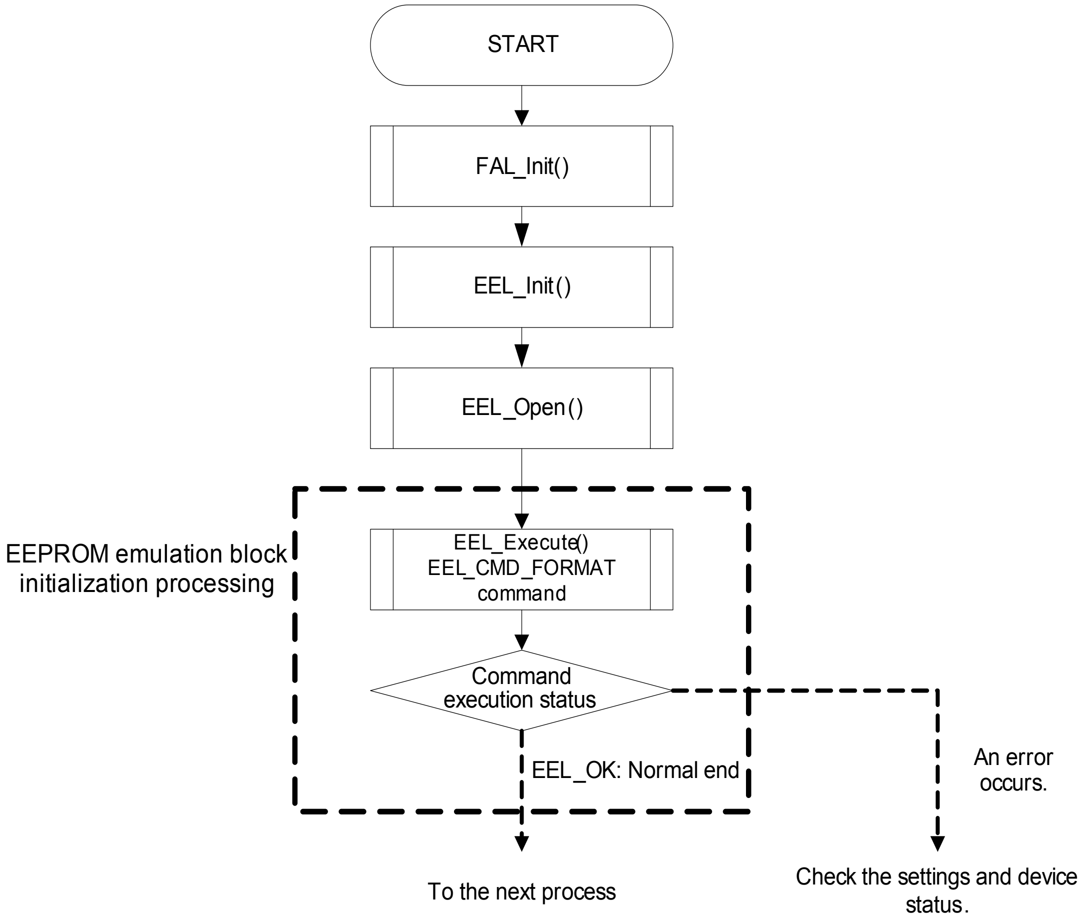
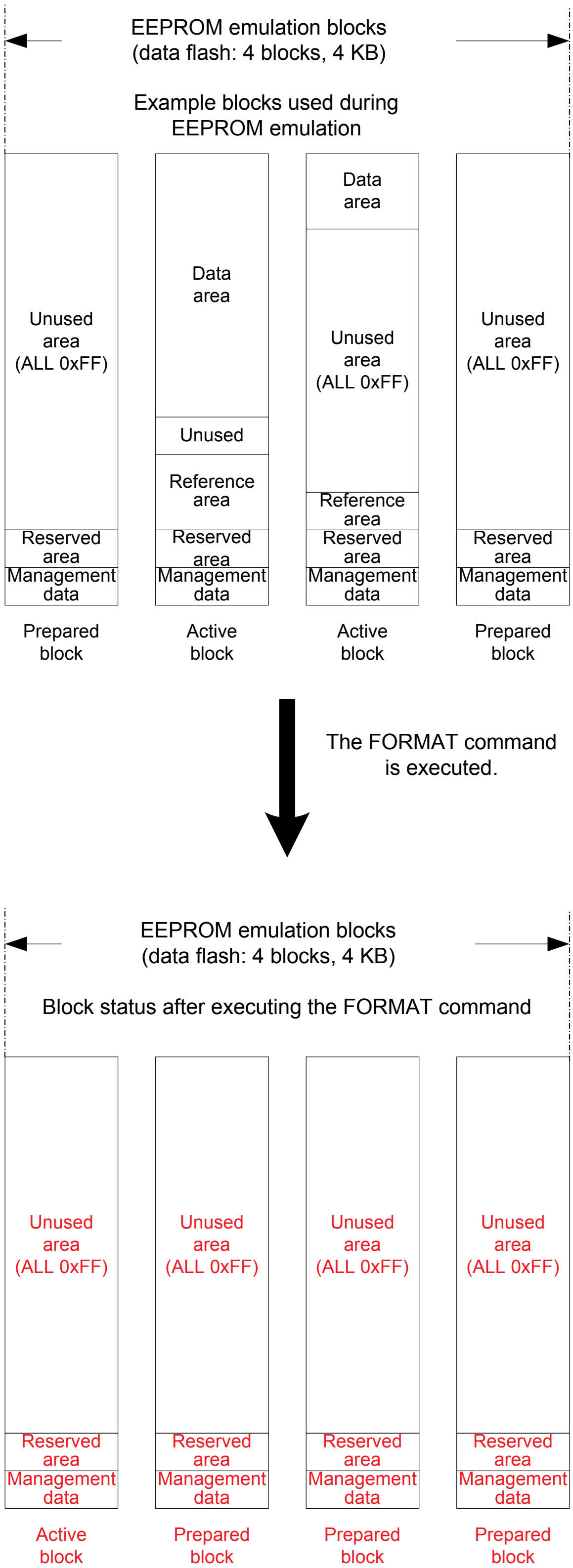


Figure 1-17. Example Block Status Transitions when EEPROM Emulation Blocks in Use are Initialized at Arbitrary Time



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

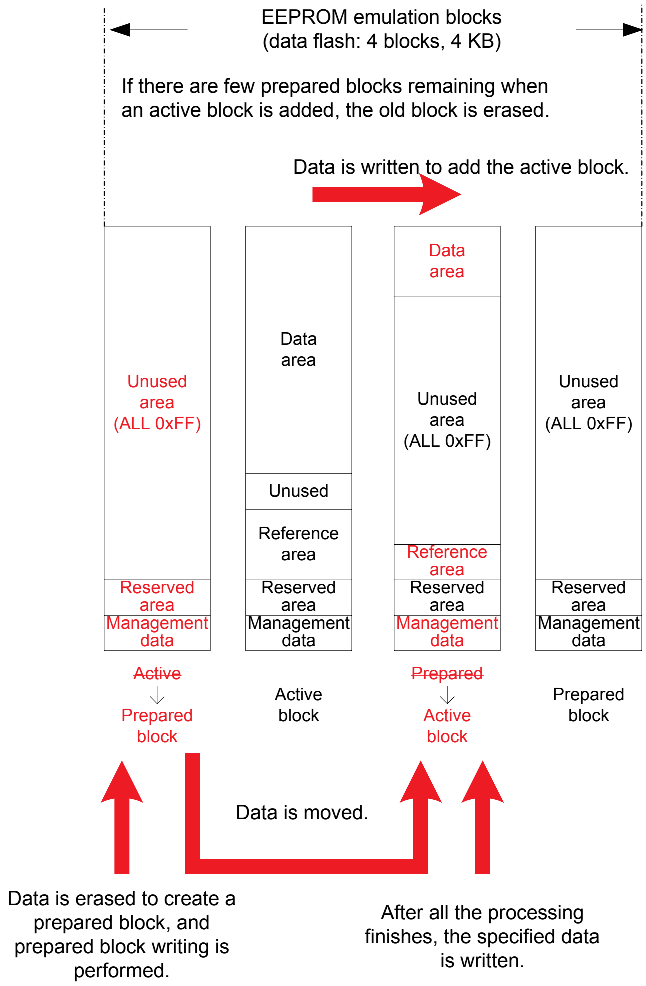
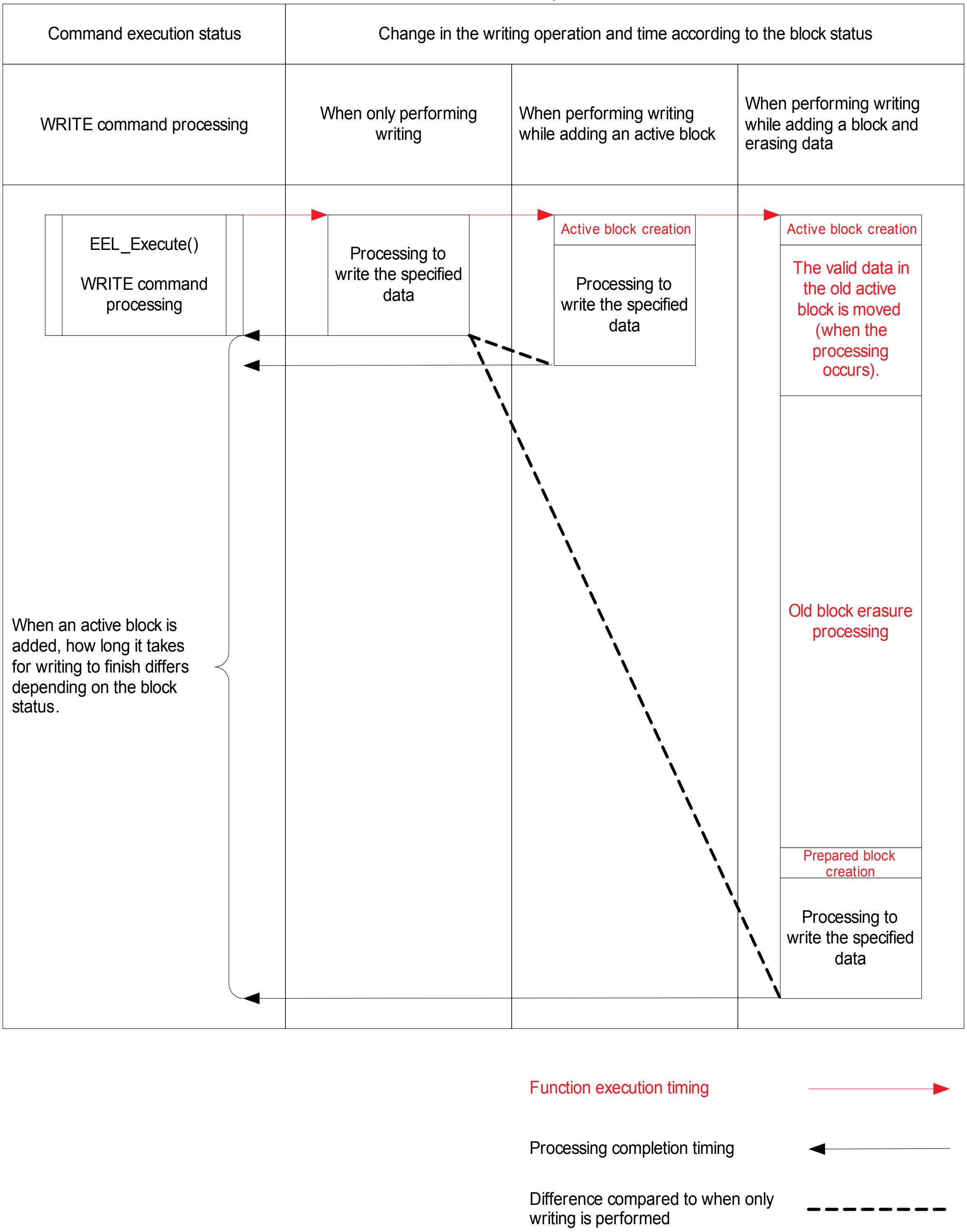


Figure 1-19. Changes in Processing Time according to Block Status when Writing Data (when Using Enforced Mode)



1. 4. 1 Adjusting Blocks by Using the 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 the EEL_CMD_CLEANUP Command (when Using Enforced Mode)

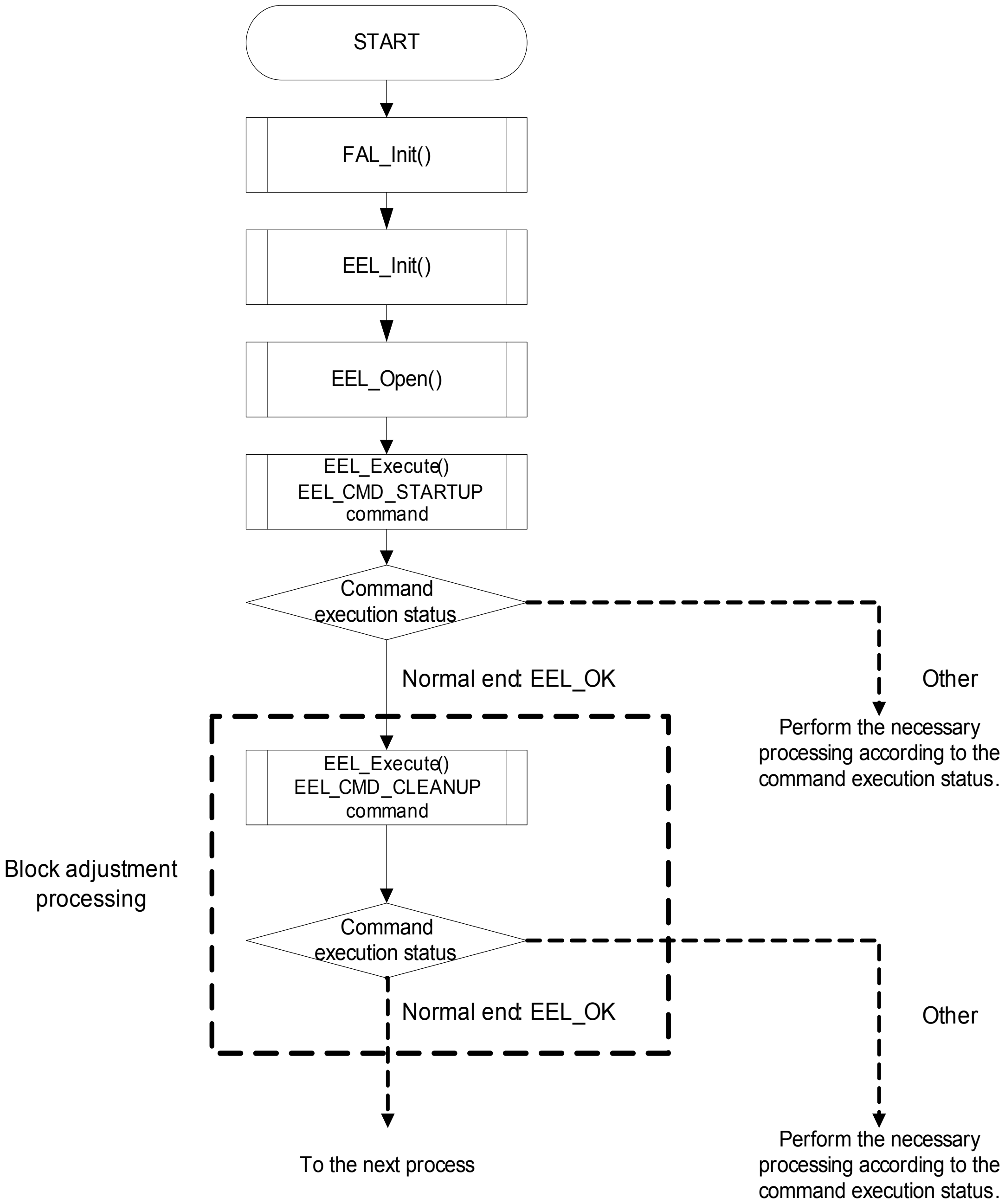
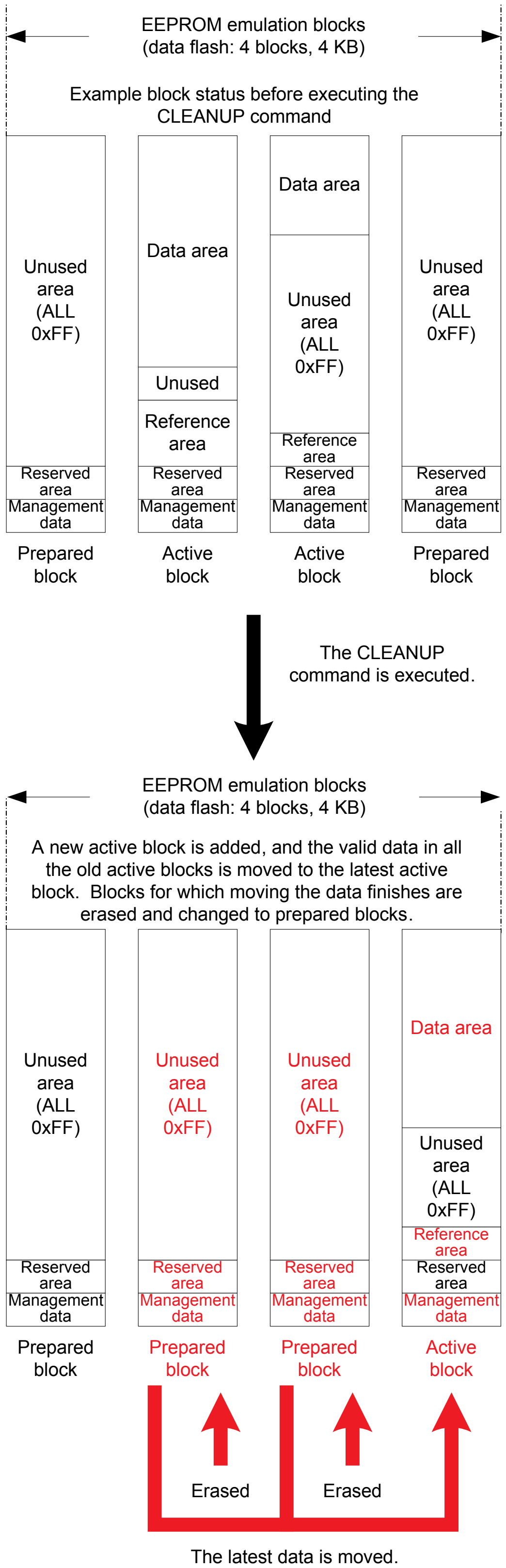


Figure 1-21. Adjusting Blocks by Using the EEL_CMD_CLEANUP Command



1. 4. 2 Adjusting Blocks by Using the 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 **2.4 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 the EEL_Handler Function

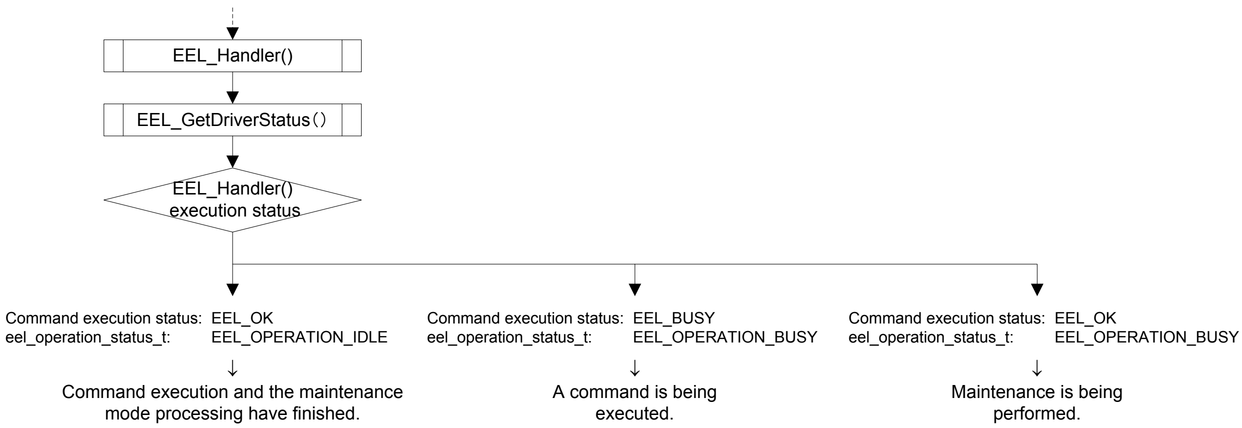


Table 1-3. EEL_Handler Function Execution Status

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. (This includes when the maintenance mode processing is paused.) 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)

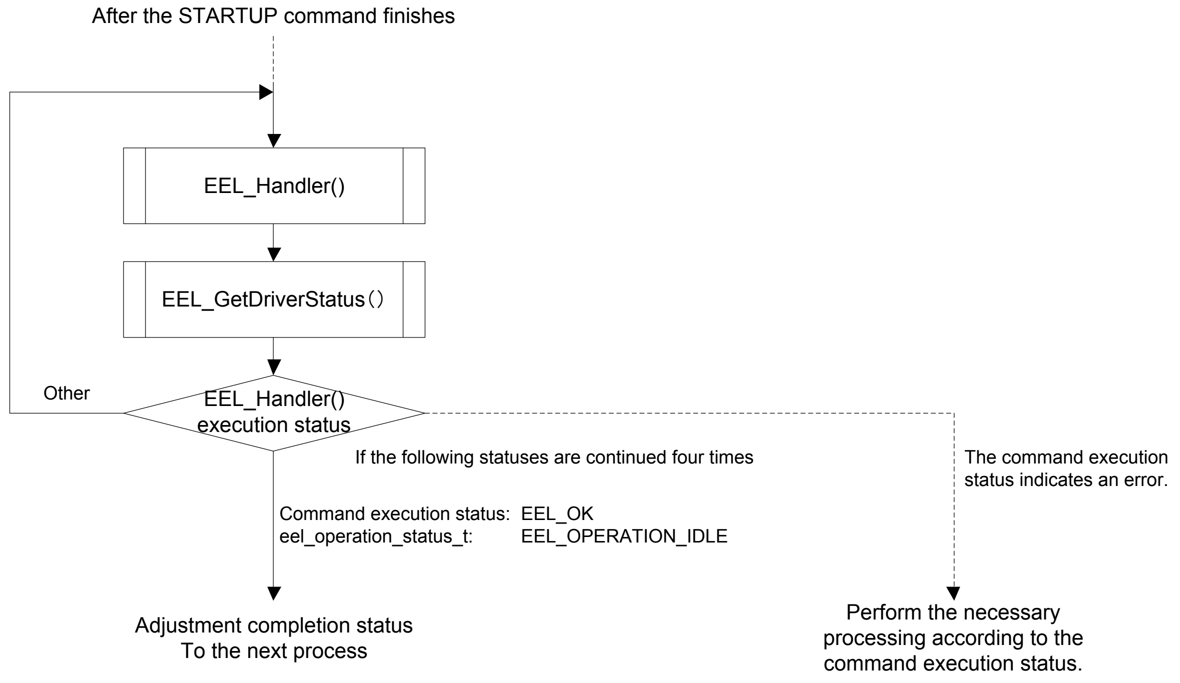


Figure 1-24. Execution Flow when Performing Adjustment after Executing the EEL_CMD_WRITE Command (Adjustment Completion Status)

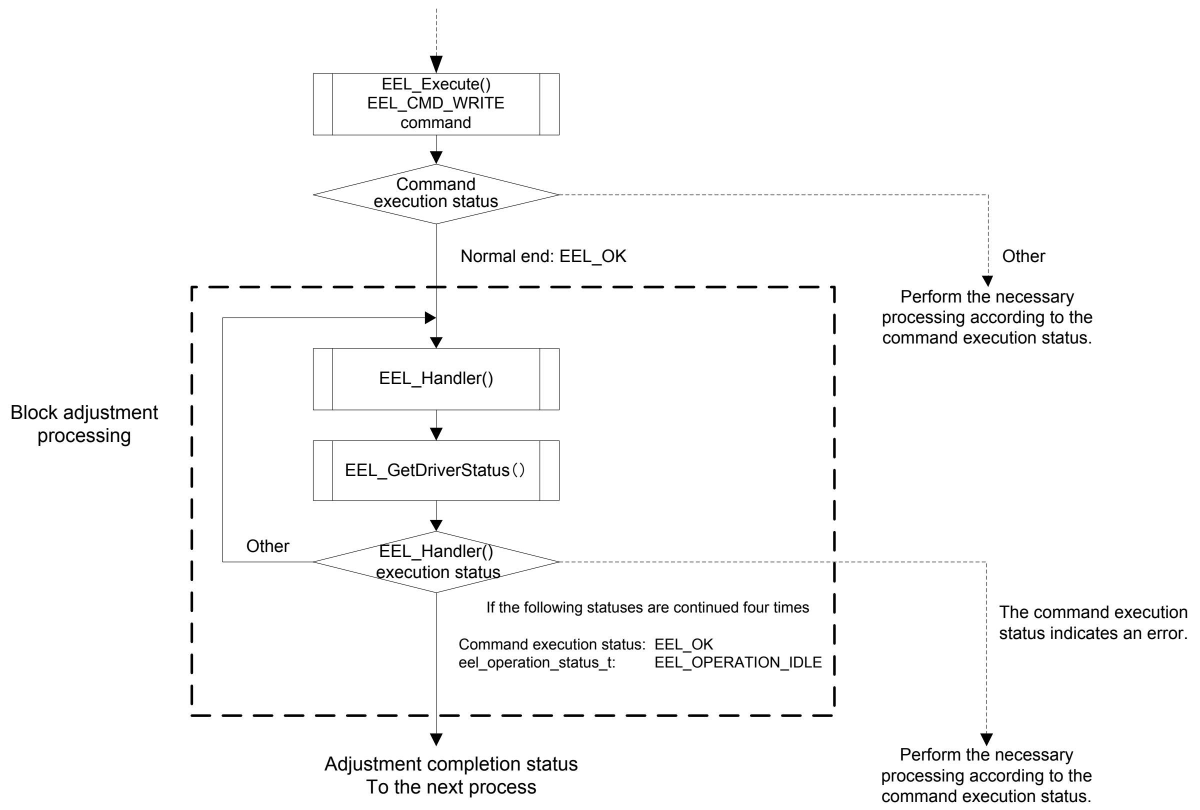


Figure 1-25. Execution Flow when Performing Adjustment upon Starting EEPROM Emulation (Adjustment Completion Status)

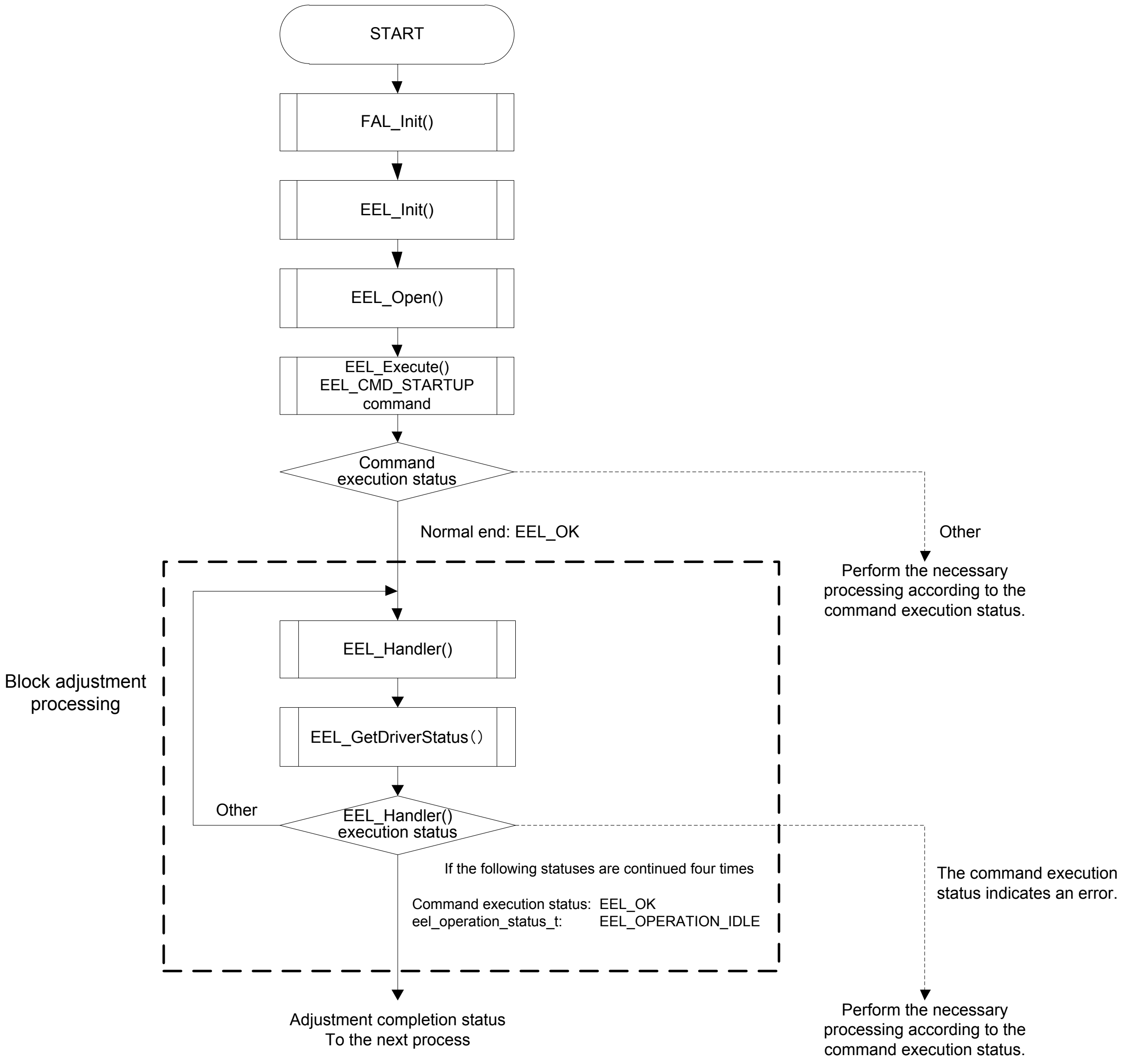


Figure 1-26. Example Flow for Executing Command and Maintenance Mode Processing in Polling Mode

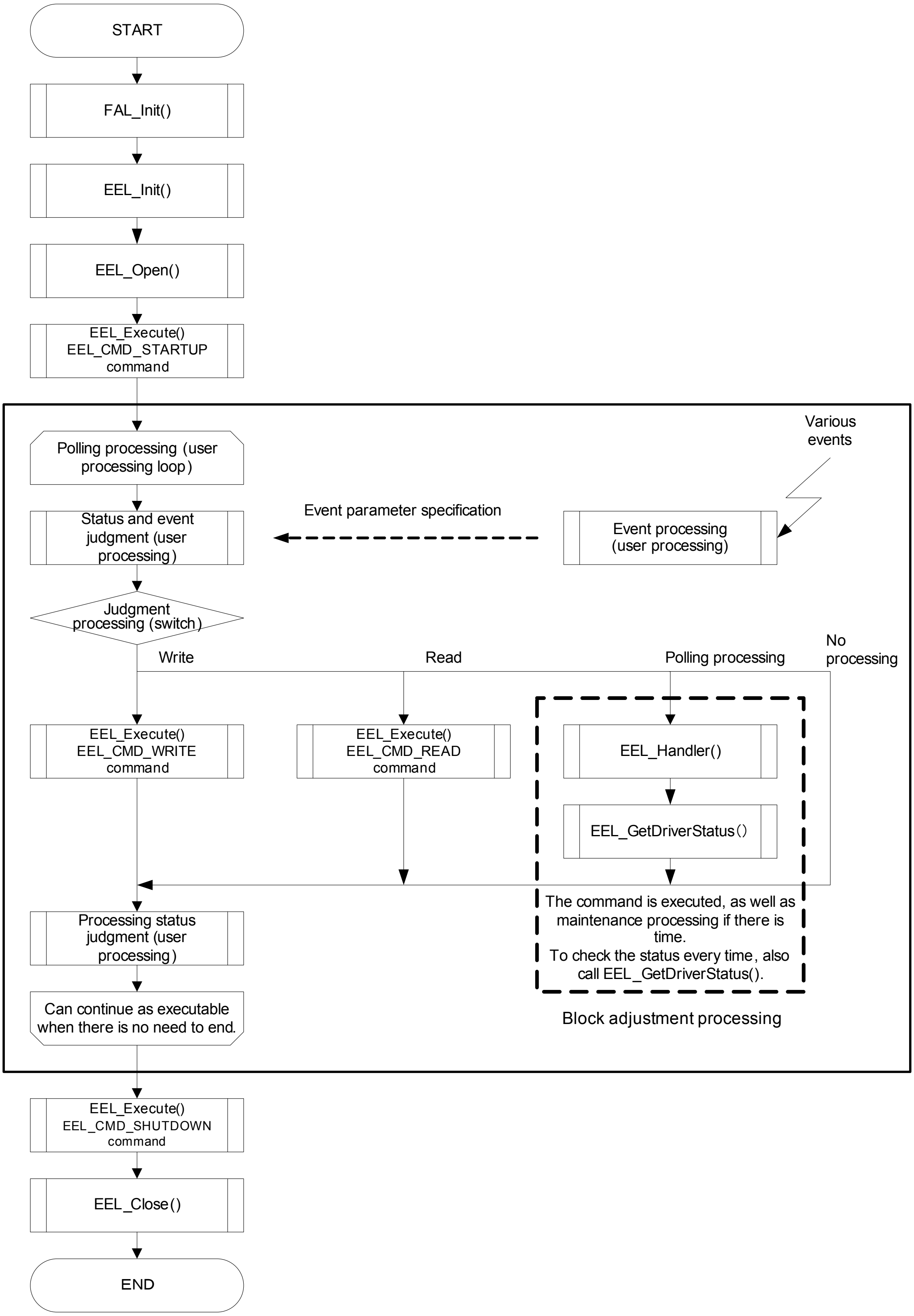
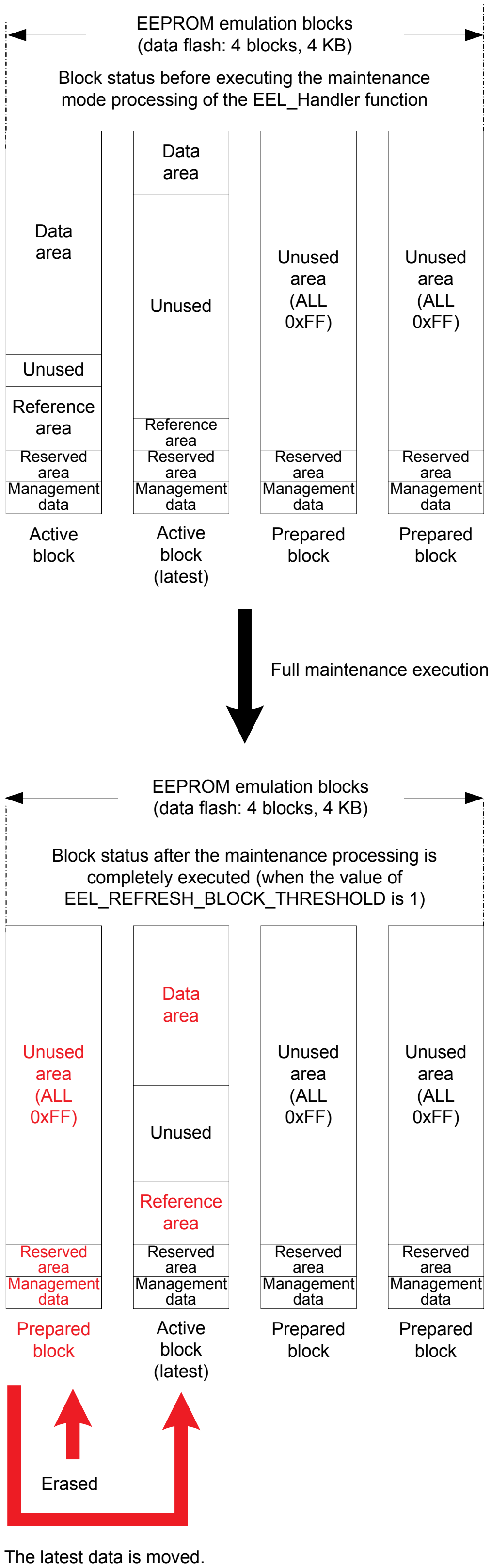


Figure 1-27. Example of Block Adjustment in EEL_Handler Function Maintenance Mode (Adjustment Completion Status)



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.

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

(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 processing 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 processing.

(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 operating frequency of RL78 microcontrollers and an operating 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 operating frequency of RL78 microcontrollers, only 1 MHz, 2 MHz and 3 MHz can be used (frequencies other than integer values like a 1.5 MHz cannot be used). Also, set an integer value 1, 2, or 3 to the operating frequency value set by the initializing function.
- When using a frequency of 4 MHz or higher [Note1] as an operating frequency of RL78 microcontrollers, a certain frequency can be used as an operating frequency of RL78 microcontrollers.
- This operating frequency is not the frequency of the high-speed on-chip oscillator.

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

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

(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.

(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.

(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.

(17) Four or more blocks of data flash memory 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.

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

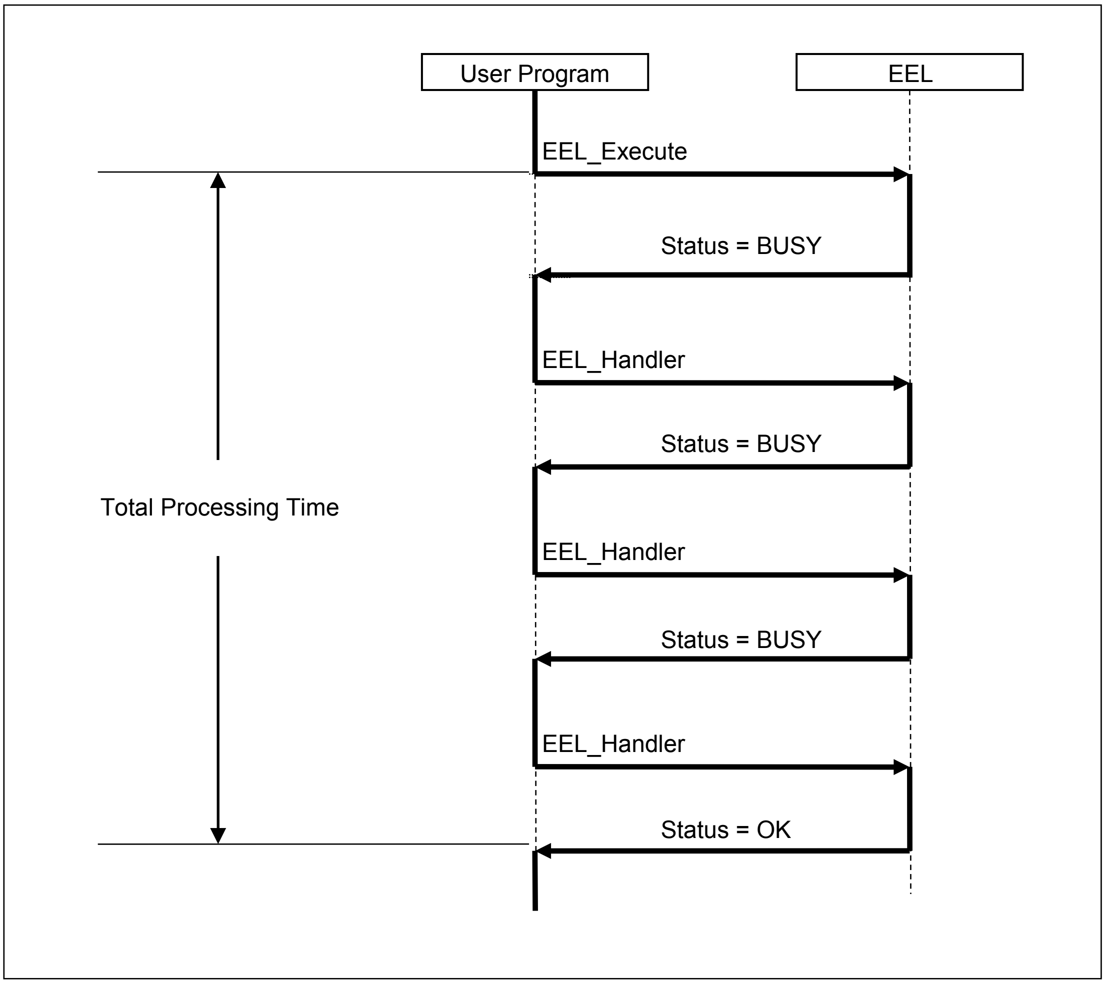


Table 2-1 Total Processing Time of EEPROM Emulation Library Pack 01 (when All Data Fits in 1 Block)

Functions		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_STARTUP 1. Minimum data length is 1 to 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 to 16 bytes Note	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 to 64 bytes Note	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

Remarks. 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

Note: If you do not intend to use any of the minimum data lengths listed above, check the processing time for a smaller minimum data length than the nearest longer range.

Functions	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

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

Block Num: Number of EEPROM emulation blocks

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 and 2-3 list the software resources[Notes 1, 2]. Figures 2-2 and 2-3 show examples of arrangements of RAM.

<R> **Table 2-2 Software Resources Used by EEPROM Emulation Library Pack01 Ver. 1.14**

Item	Size(Byte)	Restrictions on Allocation and Usage[Note 1]
Self RAM[Note 2]	0 to 1024[Note 2]	Use of the self-RAM area by RL78 Family EEPROM Emulation Library Pack 01 differs with the device. For details, refer to "RL78 Family Self RAM list of Flash Self Programming Library (R20UT2944)".
Stack	100	Can be allocated to a RAM area other than the self RAM and the area from FFE20H to FFEFFH.
Data buffer[Note 3]	1 to 256	
Requester (argument)	6	
Library search area	2	
SADDR RAM work area	11 (fdl:2) (eel:9)	Can be allocated to a short-addressing RAM area.
Library size	8200 (fdl:1500) (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 n = number of data items stored	
Fixed-parameter area (default)	72 (fdl:64) (eel:4)	
EEPROM emulation block	4,096 or more (at least 4 blocks)	Only data flash memory can be used.

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

2. An area used as the working area by the 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.

3. 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 Pack01 Functions

Function Name	Byte
FAL_Init	0
EEL_Init	0
EEL_Open	0
EEL_Close	0
EEL_Execute[note]	0 to 256
EEL_Handler[note]	0 to 256
EEL_TimeOut_CountDown[note]	0
EEL_GetDriverStatus	3
EEL_GetSpace	2
EEL_GetVersionString	0

Note. Another 6 bytes area of the requester is used.

Figure 2-2 Example 1 of RAM Allocation with Self-RAM

(RL78/G13: product with 4 KB RAM and 64 KB ROM)

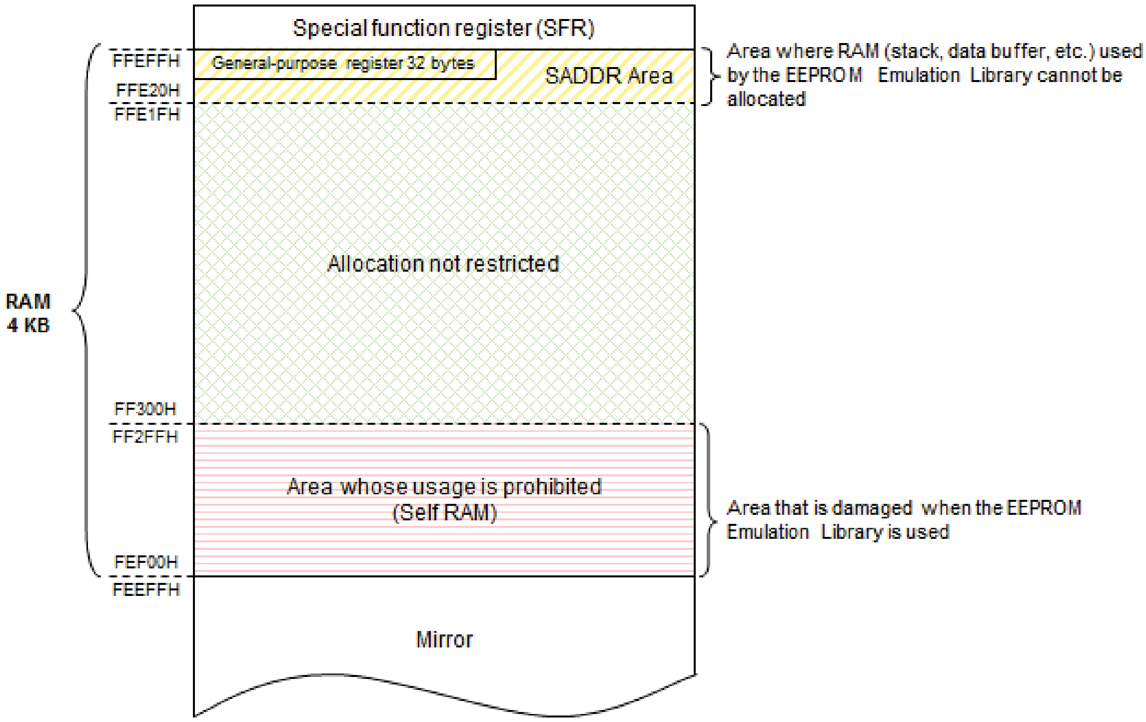
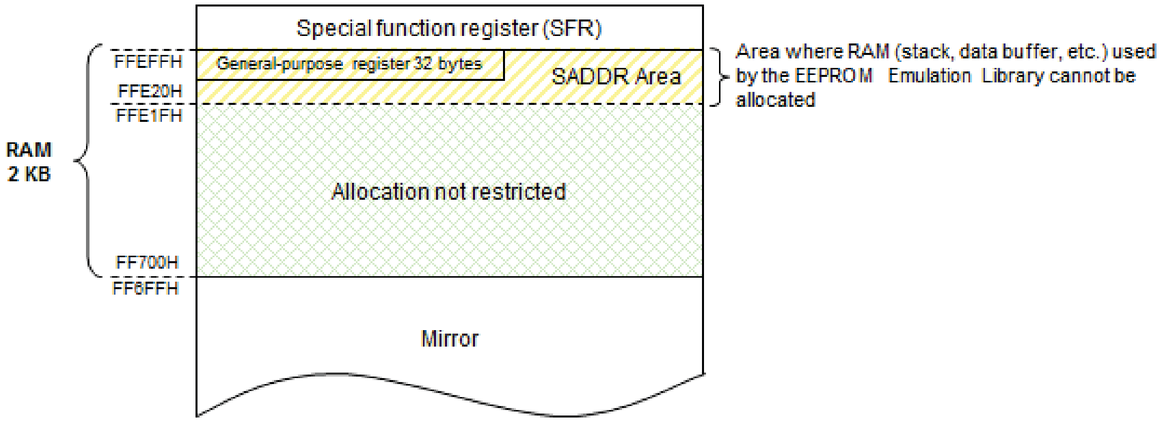


Figure 2-3 Example 2 of RAM Allocation without Self-RAM

(RL78/G13: product with 2 KB RAM and 32 KB ROM)



2. 4 Initial Values to be Set by User

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 (fdl_descriptor.h)>Notes 1, 2

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

Set an operating 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 operating frequency = ((FDL_SYSTEM_FREQUENCY + 999999) / 1000000)

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

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

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

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

(2) Voltage mode Note 1

Set 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
Set 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 with “!” if you are placing the data flash library at 00000H to 0FFFFH or with “!!” otherwise.

[Advance setting]

- The flash self programming library and EEPROM emulation library processing must be either not executing or finished.
- 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) and 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	Start address of data flash memory
eel_pool_first_addr_u32	fal_u32	Start address of data flash memory used for EEL Note 2
user_pool_first_addr_u32	fal_u32	Unused
fal_pool_last_addr_u32	fal_u32	End address of data flash memory
eel_pool_last_addr_u32	fal_u32	End address of data flash memory used for EEL Note 2
user_pool_last_addr_u32	fal_u32	Unused
fal_pool_first_block_u16	fal_u16	First block number of data flash memory
eel_pool_first_block_u16	fal_u16	First block number of data flash memory used for EEL Note 2
user_pool_first_block_u16	fal_u16	Unused
fal_pool_last_block_u16	fal_u16	Last block number of data flash memory
eel_pool_last_block_u16	fal_u16	Last block number of data flash memory used for EEL Note 2
user_pool_last_block_u16	fal_u16	Unused
fal_first_widx_u16	fal_u16	Access start number of data flash memory
eel_first_widx_u16	fal_u16	Access start number of data flash memory used for EEL Note 2
user_first_widx_u16	fal_u16	Unused
fal_last_widx_u16	fal_u16	Access end number of data flash memory
eel_last_widx_u16	fal_u16	Access end number of data flash memory 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 bytes units)
eel_pool_wsize_u16	fal_u16	Area size of all blocks used for EEL Note 2 (word units: 4 bytes 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 bytes units)
fal_pool_size_u08	fal_u08	Block size of data flash memory
eel_pool_size_u08	fal_u08	Block size of data flash memory used for EEL Note 2
user_pool_size_u08	fal_u08	Unused
fx_MHz_u08	fal_u08	Setting of operating 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 stands for EEPROM emulation library.

Development Tool	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]

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 or the high-speed on-chip oscillator is running.

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 * This is used for command execution in modes other than the enforced mode.
EEL_TimeOut_CountDown	Timeout counting processing for EEPROM emulation command execution * 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 with "!" if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with "!!" otherwise.

[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. Be sure to execute this function when starting EEPROM emulation 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 with “!” if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with “!!” otherwise.

[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 with “!” if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with “!!” otherwise.

[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 system to a status in which the data flash memory can be manipulated 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 with “!” if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with “!!” otherwise.

[Advance setting]

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

[Function]

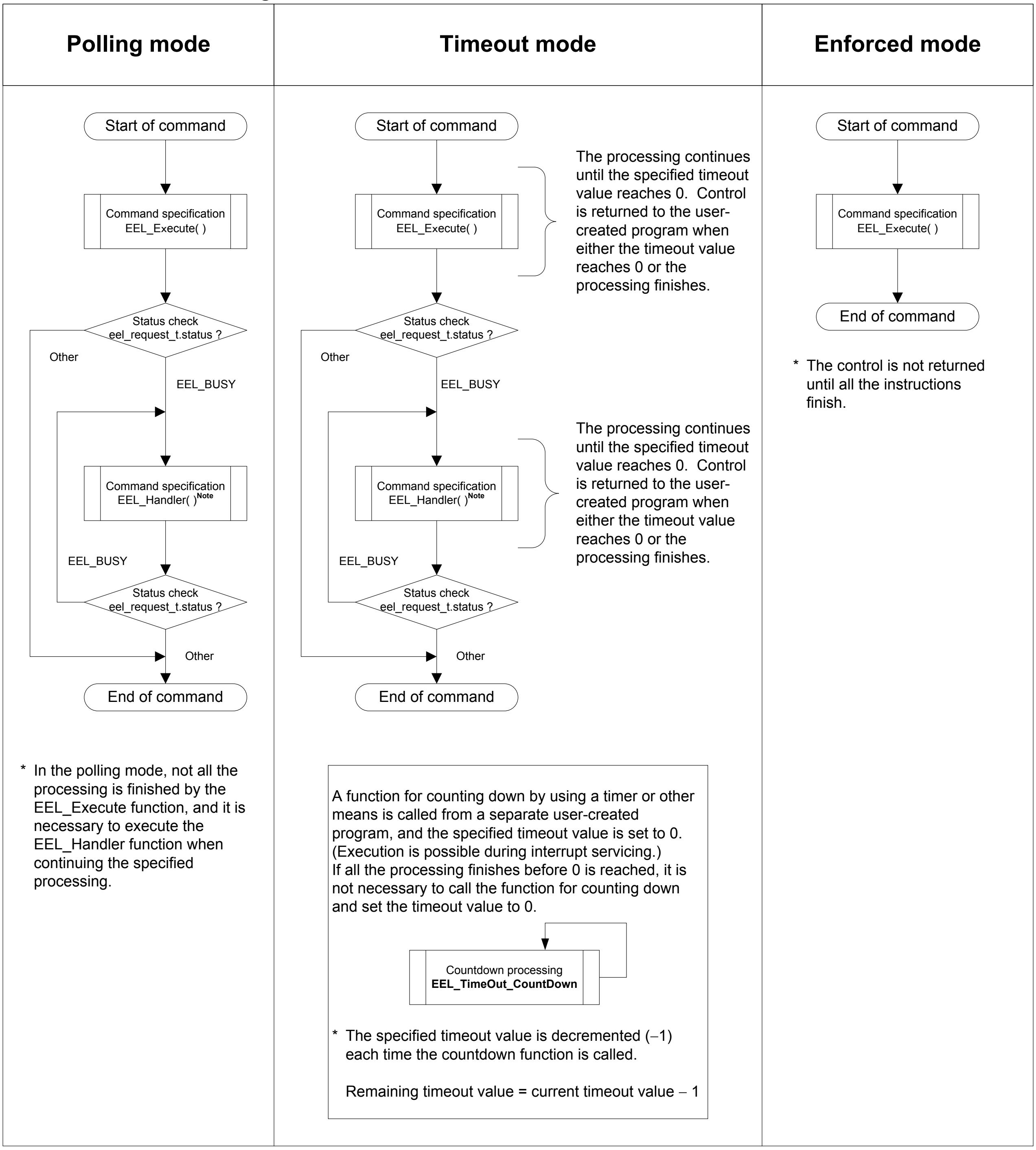
Each type of processing for performing EEPROM emulation operations is specified for this function as an argument in the command format, and the processing is executed. When executing the EEL_Execute function, it is possible to specify the execution mode and select the EEPROM emulation execution method. 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. 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 command 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. 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 command writes the specified data to the EEPROM emulation blocks. * The following arguments must be specified prior to execution. • eel_request_t.address: Specifies the start address of the RAM area where the write data is stored. • eel_request_t.identifier: Specifies the data ID of the write data.
<R> EEL_CMD_READ Note 1	This command reads the latest data from the EEPROM emulation blocks corresponding to the specified data ID. * The following arguments must be specified prior to execution. • eel_request_t.address: Specifies the address of the RAM area from which the data are to be read. • eel_request_t.identifier: Specifies the data ID of the data to be read.
EEL_CMD_CLEANUP Notes 1, 3	This command 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. 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 command initializes (erases) Note 2 everything, including the data recorded in the EEPROM emulation blocks. Be sure to issue this command before using EEPROM emulation for the first time. Note that issuing this command is also necessary to initialize all blocks if a malfunction occurs in an EEPROM emulation block (such as a valid block disappearing) or the initial values (those which are fixed values that cannot be changed) are modified. 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 command 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 the EEL_CMD_CLEANUP Command**.

4. For the detailed operations, see **1.3 Initializing EEPROM Emulation Blocks**.

Development Tool	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: The FAL_Init(), EEL_Init(), or EEL_Open function has not finished normally.	All commands
EEL_ERR_ACCESS_LOCKED	EEPROM emulation lock error: 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: A command that does not exist has been specified.	–
EEL_ERR_PARAMETER	Parameter error: An incorrect command parameter has been specified. Check the specified parameter.	All commands
EEL_ERR_REJECTED	Reject error: A different command is being executed.	All commands
EEL_ERR_NO_INSTANCE	Identifier error: 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: There is no area that can be used to write the data. This error can be recovered 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: An EEPROM emulation block has the undefined status (such as 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: 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: 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 with “!” if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with “!!” otherwise.

[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. (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.

Data to be Input to eel_request_t

Argument	Type	Description
eel_request_t.status	eel_status_t	Command execution status

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 with “!” if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with “!!” otherwise.

[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 loop processing of the EEL_Execute and EEL_Handler functions 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 with “!” if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with “!!” otherwise.

[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 for users)

eel_operation_status_t Details

Type	Description
EEL_OPERATION_PASSIVE	The EEL_CMD_STARTUP command has not finished normally when 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 the 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 with "!" if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with "!!" otherwise.

[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 bytes 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 succeeded.
	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 with “!” if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with “!!” otherwise.

[Advance setting]

The FAL_Init and EEL_Init functions must have finished normally.

[Function]

None

[Register status after calling this function]

Return value: BC and DE

Corrupted registers: BC and DE

[Argument]

None

[Return value]

Type	Description
eel_u08*	The address at which the version information of the EEPROM emulation library (EEL) is input (a 24-bit address area) Example: For EEPROM emulation library Pack01 V1.14 (ASCII code) “ERL78T01R110GVxxx” Version information: V114 → V1.14 Corresponding tool: Renesas Electronics version Type name: Type 01 Corresponding device: RL78 Target library: EEL

APPENDIX A REVISION HISTORY

A. 1 Major Revisions in This Edition

Page	Description	Classification
Throughout the document		
–	Statements of the target devices were deleted.	(c)
–	References to the list of target MCUs were added.	(e)
Chapter 1 OVERVIEW OF EEPROM EMULATION		
p.2	In figures 1-1 and 1-2, the description “flash data library” was corrected to “data flash library”.	(a)
Chapter 2 USING EEPROM EMULATION		
p.36	The version number was added to the heading for table 2-2.	(c)
p.36	The description of the self-RAM area in table 2-2 was changed.	(c)
p.36	Note 1 for table 2-2 was changed to include a statement regarding inquiries about device specifications.	(c)
p.36	The previous note 2 under table 2-2 was deleted.	(c)
p.36	The previous note 4 under table 2-2 was deleted.	(c)
Chapter 3 EEPROM EMULATION FEATURES		
p.52	The description in the table was changed.	(c)

Remark Symbols under “Classification” in the above table are used to classify revisions as follows.
(a): Correction of errors, (b): addition or change to specifications, (c): addition or change to descriptions or notes, (d): addition or change to packages, part numbers, or management divisions, (e): addition or change to related documents

A. 2 Revision History of Previous Editions

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

Rev. No.	Description	Chapter
1.02	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.	Throughout the document
	The corresponding ZIP file name and release version were added to the cover page.	
	Contents of the notes, 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.	
	The supported device was added.	
	The notation of high-speed OCO was deleted to unify the notation of high-speed on-chip oscillator.	
	The description of the operating frequency was unified to the CPU operating frequency since individual descriptions had different notations.	
	Errors in the capacity of the separator and the calculation method for the maximum size were modified.	Chapter 1 OVERVIEW OF EEPROM EMULATION
	Note on the frequency of the high-speed on-chip oscillator was added.	Chapter 2 USING EEPROM EMULATION
	Note on the RAM parity error was added.	
	Note on the data flash control register (DFLCTL) was added.	
	Note on the number of data flash memory blocks was added.	
	Items regarding the processing time were added (the description of the processing time was moved from the usage note to this document).	
	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).	
	Note on the high-speed on-chip oscillator was added.	
	Description of the return value of the Fal_Init function was added.	Chapter 3 EEPROM EMULATION FEATURES
	Description of the argument of the EEL_Execute function was added.	
	Description of the return value of the EEL_GetVersionString function was changed.	

RL78 Family User's Manual: EEPROM Emulation Library Pack 01

Publication Date: Rev.1.03 Jan 29, 2016

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.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, 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 1207, 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 India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 4.0

RL78 Family

Renesas Electronics Corporation

R01US0054EJ0103