R01AN2014EJ0110
Ver.1.10
2022.9.9

RL78/L13

Integrate External EEPROM IC Functionality into MCU by Using Data Flash Memory (EEPROM Emulation Library)

Introduction

Self-programming is a function that the microcontroller to rewrite the internal flash memory by itself. RL78/L13 is equipped with the data flash memory which is suitable for data storage. Rewriting of data flash memory can be realized by the Flash Data Library (FDL) and the EEPROM Emulation Library (EEL) from Renesas Electronics Corp.
This application note explains how to hold non-volatile data simply by data flash memory and the EEL without using external EEPROM IC. It also explains how to save data to data flash memory quickly after detecting low voltage to prepare for power interruption.
User can integrate the function of external EEPROM IC into microcontroller by applying this application note.

Correspondence between Compiler and EEL

This application note has a sample code (excluding the EEL). In order to operate this sample code, it is required to download and link EEL to the project. Refer to "6.9 How to import EEL" for details on method of linking EEL to the project.
EEL has a CubeSuite+ version. However, the version of EEL supported by each sales company (each area) is different. Confirm the supported version by selecting the area on the Renesas Electronics Website (http://www.renesas.com).
Please check the manual of the EEL, and the release note (or README.txt on the download source page) before using the EEL.

Correspondence between Compiler and EEL

	EEL	Download Link
CubeSuite+ version	EEPROM Emulation Library Pack02 for the RL78 Family Ver.1.01	https://www.renesas.com/software-tool/data-flash-libraries#overview
	RENESAS_EEL_RL78_T02E_V1.10 (Use it after linking with the following FDL which is available in the same place. FDLRENESAS_FDL_RL78_T02E_V1.10)	http://www.renesas.eu/updates?oc=EEPROM_EMULATION_RL78

Target Device

RL78/L13

EEL used in this application note supports other devices of RL78.
RL78/D1A, RL78/F12, RL78/F13, RL78/F14, RL78/G13, RL78/G14, RL78/G1A, RL78/G1E, RL78/I1A, RL78/L1C

Confirm by the latest user's manual of the EEL about the supported device of EEL.
When applying the sample program covered in this application note to another RL78 microcontroller, conduct an extensive evaluation of the modified program.

Contents

1. Overview

There are three types of Self Programming Library; the Flash Self Programming library (FSL), the FDL, and the EEL shown in .

As libraries using data flash memory, the outline of the EEL is indicated in **1.1 Outline of the EEL** and the outline of FDL is indicated in **1.2 Outline of the FDL**. This application note explains the EEL which is indicated with the bold font in .

Table 1.1 List of Self Programing Library

Name of library	Target flash memory	Description
FSL	Code flash memory	Rewrites data in code flash memory.
FDL	**Data flash memory**	Rewrites and reads data in data flash memory.
EEL		**Uses data flash library just like EEPROM to rewrite and read data.**

1.1 Outline of the EEL

The EEL is a software library to store the data in internal data flash memory of the RL78 microcontroller in the same way as EEPROM. In order to rewrite data flash memory with the EEL, the corresponding functions of EEL initialization and other purpose, would be called from the user-created program.

The EEL allows the user to assign a 1-byte identifier (data ID: 1 to 64) to each block of data and to perform read or write operations in units of 1 to 255 bytes for each ID that is assigned (a maximum of 64 data items can be assigned to an ID).

1.2 Outline of the FDL

The FDL is a software library to perform operations to the data flash memory with the firmware installed on the RL78 microcontroller. In order to rewrite data flash memory with the FDL, the corresponding functions of FDL initialization and other purpose, would be called from the user-created program.

The fundamental usage of the FDL is to write data byte by byte to the data flash memory’s address, which has not been written(in the blank state). However, it cannot overwrite the same address. In order to overwrite the same address, data erasing per block is required in advance.

1.3 Proper Use of the FDL and the EEL

There are some differences such as rewriting method, resources required, execution time, data management mechanism and so on between using the FDL and the EEL. Main features of the FDL and the EEL are shown in Since FDL is only a fundamental access function to data flash memory, it can be customized to manage data flexibly according to the user-created program. On the other hand, EEL has the feature that development load is low because the mechanism of data management was decided in advance by the EEL.

Select the FDL or the EEL according to the requirements for application.

Table 1.2 Features of the FDL and the EEL

	FDL	EEL
Rewriting method	Depends on user-created program.	Writes after changing address.
Resources required	Small	Large
Data size	Up to 1024 bytes	Up to 255 bytes
Execution time	Short	Long
Data management mechanism	None (User manage data by address)	Managed (By data number)

Caution: The feature of the FDL is dependent on the upper-class layer, the application (the specification of data management).

(1) Rewriting Method

Writing is permitted only when the target write address of data flash memory is in the blank state. It is necessary to erase data in units of one block in advance in order to overwrite the same address.

FDL itself does not have a mechanism in which data can be managed. It is necessary to consider how to manage data in the application layer (by user). On the other hand, EEL has a mechanism to manage data, and writes data with keeping changing the variable which contains an address that marks the memory in the blank state in data flash memory. Since data can be written in until the block for writing is filled with data, it is suitable for mass data storage and frequent data writing.

(2) Resources required

Software resources required by the FDL and the EEL are shown in . The Self-RAM, stack, and data buffer have to use RAM. Since the EEL uses the FDL, the amount of the EEL ROM resources is larger than the FDL ROM resources.

Table 1.3 Software Resources of FDL/EEL (e.g. RL78/L13)

Item	Size (byte)	
	FDL	EEL
Self-RAM Note 1	0 to 1024	0 to 1024
Stack	MAX 46	MAX 80
Data buffer Note 2	1 to 1024	1 to 255
Library size	ROM : MAX 177	ROM :MAX 3400 (FDL : 600, EEL : 2800)

Note 1: An area used as the working area by the EEL is called self-RAM. The self-RAM requires no user setting because it is an area that is not mapped and automatically used at execution of the EEL (previous data is discarded).

Note 2: A RAM space required in order to input the data read and written is called a data buffer. Required size changes by the reading and writing unit. When performing 1 byte of reading and writing, a needed data buffer is 1 byte.

Note 3: The resources given in this table are according to FDL RL78 Type04 Ver1.05 and EEL RL78 Pack02 Ver1.01. The library may change by upgrade etc. Confirm the manual of each library for the latest resource information.

(3) Data Size

The FDL is able to read and write data up to 1024 bytes (1 block of a data flash memory). The EEL is able to read and write data up to 255 bytes. The FDL has an advantage when saving big data.
The data buffer of expresses the size of the data which can be read and written at a time.

(4) Execution Time

The execution time of the library function of FDL and EEL is shown in **Table 1.4**. The FDL without data management mechanism can read and write data at high speed.

Table 1.4 The Execution Time of the Library Function of FDL/EEL

Processing	FDL (255 bytes)	EEL (255 bytes)
Write FDL : PFDL_Execute(Write) EEL : EEL_Execute(Write)	519.7[μs]	11399.7[μs]
Read FDL : PFDL_Execute(Read) EEL : EEL_Execute(Read)	167.7[μs]	179.7[μs]
Verify FDL : PFDL_Execute(IVerify) EEL : EEL_Execute(Verify)	959.7[μs]	3919.7[μs]

Remark. The execution time described in this application note is the actual measured value calculated on operating FDL RL78 Type04 Ver1.05 or EEL RL78 Pack02 Ver1.01 on the integrated development environment CubeSuite+. The value would be different according to the individual specificities of the device and the execution condition.

(5) Data Management Mechanism

The FDL uses address to access data flash memory. Since the address in which the newest data is stored is changed, it needs to manage the address. On the other hand, EEL manages data by data ID. Therefore, it is not necessary to manage the address in which the newest data is stored when using the EEL.

1.4 Benefits and Caution Points When EEPROM IC is Replaced

This section explains benefits when replacing the function of EEPROM IC with data flash memory by using EEL, and the difference from EEPROM IC.

1.4.1 Benefits form Replacing EEPROM IC

The benefits of replacing from EEPROM IC are shown below.

- Since external EEPROM IC becomes unnecessary, parts cost reduction and small footprint are realizable.
- Since it is the operation completed inside device, it is not necessary to perform serial communication. The serial communication pins of microcontroller can be used by other functions. In addition, the value which was written can be confirmed directly with a debugger at the time of the software development.
- Since serial communication is unnecessary, processing time can be reduced. (However, it is dependent on data structure.) In EEPROM IC, the serial communication time + the write completion time (several milliseconds) are taken for the processing time.
- Since data is managed by data ID, it is not necessary to care about address.

1.4.2 Difference from EEPROM IC

The difference with the case where EEPROM IC is used is shown below.

- Since it is emulation, the size of the flash memory which can be used by user decreases.
  Refer to “2.4 Number of stored user data items and total user data size” for how to calculate the space available to the user.
- The program which communicates with EEPROM IC is not required. Instead, a program just like the FDL or the EEL is necessary.
- The maximum number of data items is 64 and the maximum size of one data item is 255 byte. Refer to the user’s manual of the EEL for more information about the number of data items.

2. Specifications

In this application, LED0 or LED1 blinks 10 times by a keypress. The data used for LED blinking is saved to data flash memory when the supply voltage becomes too low. The saved data is read when system restarts, and the interrupted blinking processing is continued.

When reset is ended, the system reads the blinking state data (target LED for blinking, and the remaining times of LED blinking) by EEL from data flash memory where the data has been saved.

Next, after completing 10 times blinking at intervals of 500 ms according to the blinking state data, the LED stops blinking and the system becomes the waiting state for keypress.
If the key is pressed in a state in which no LED is blinking, the LED that had not been blinking just before will start to blink. The keypress becomes invalid while LED is blinking.

The fall of power supply voltage is detected by LVD function. If the fall of power supply voltage is detected, the LED blinking state data (target LED for blinking, and the remaining times of LED blinking) is saved to data flash memory by the EEL, LED3 which shows the completion of data saving will be lit up, and then the mode moves to the STOP mode.

Moreover, if an error occurs when accessing data flash memory with EEL functions, LED0 and LED1 will be lit up and the mode shifts into the STOP mode.

The structure of the data to be saved is shown in **Figure 2.1**. Higher 4 bits of this one byte user data indicates target LED for blinking and lower 4 bits indicate the remaining times of LED blinking. The example data in **Figure 2.1**shows that the rest of the LED1's blinking times is 5.
.

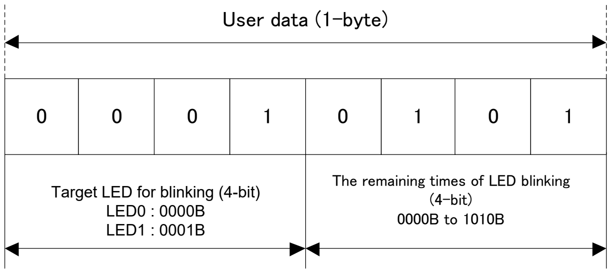


Figure 2.1 Stored Data

Table 2.1shows the required peripheral functions and their uses. **Figure 2.2** shows overall picture of application. **Figure 2.3** shows operation outline.

Table 2.1 Peripheral Functions to be Used and their Uses

Peripheral Function	Use
LVD	Supply voltage (VDD) monitoring
External interrupt (INTP0)	Key for operation switching
P05	LED lighting control (LED0)
P45	LED lighting control (LED1)
P41	LED lighting control (LED3)
Timer array unit (TAU) 0 channel 0	Generation of the wait time for chattering evasion of keypress（10ms）
TAU0 channel 1	Generation of LED blinking interval time (500ms)

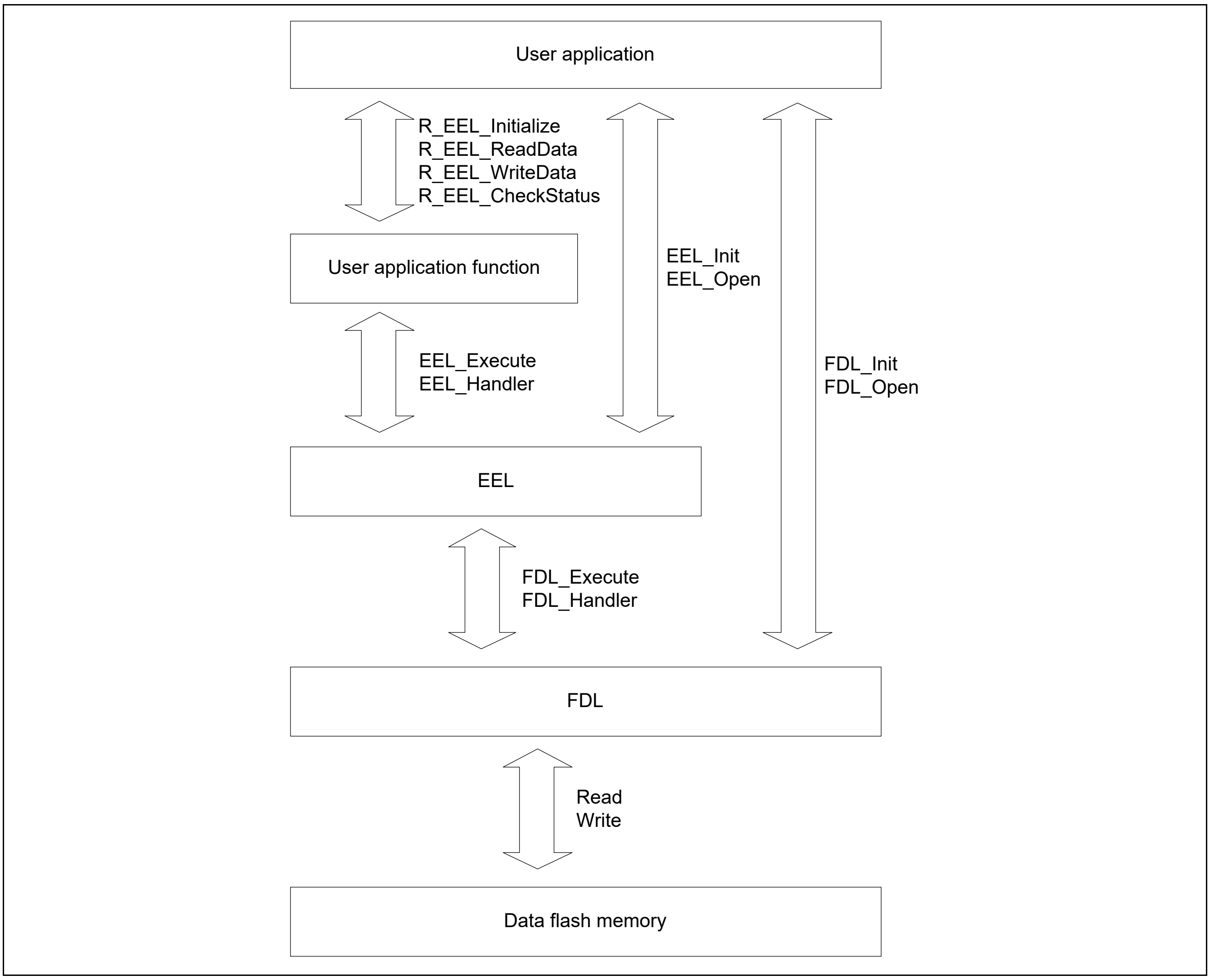


Figure 2.2 Overall Picture of Application

It is necessary to set data flash memory to the state that accessing data flash memory is permitted, or to secure the resource used by FDL/EEL by performing Init/Open of FDL/EEL, in order to access data flash memory from user application. The reading and writing to data flash memory are enabled by performing Startup of EEL after the Init/Open processing.

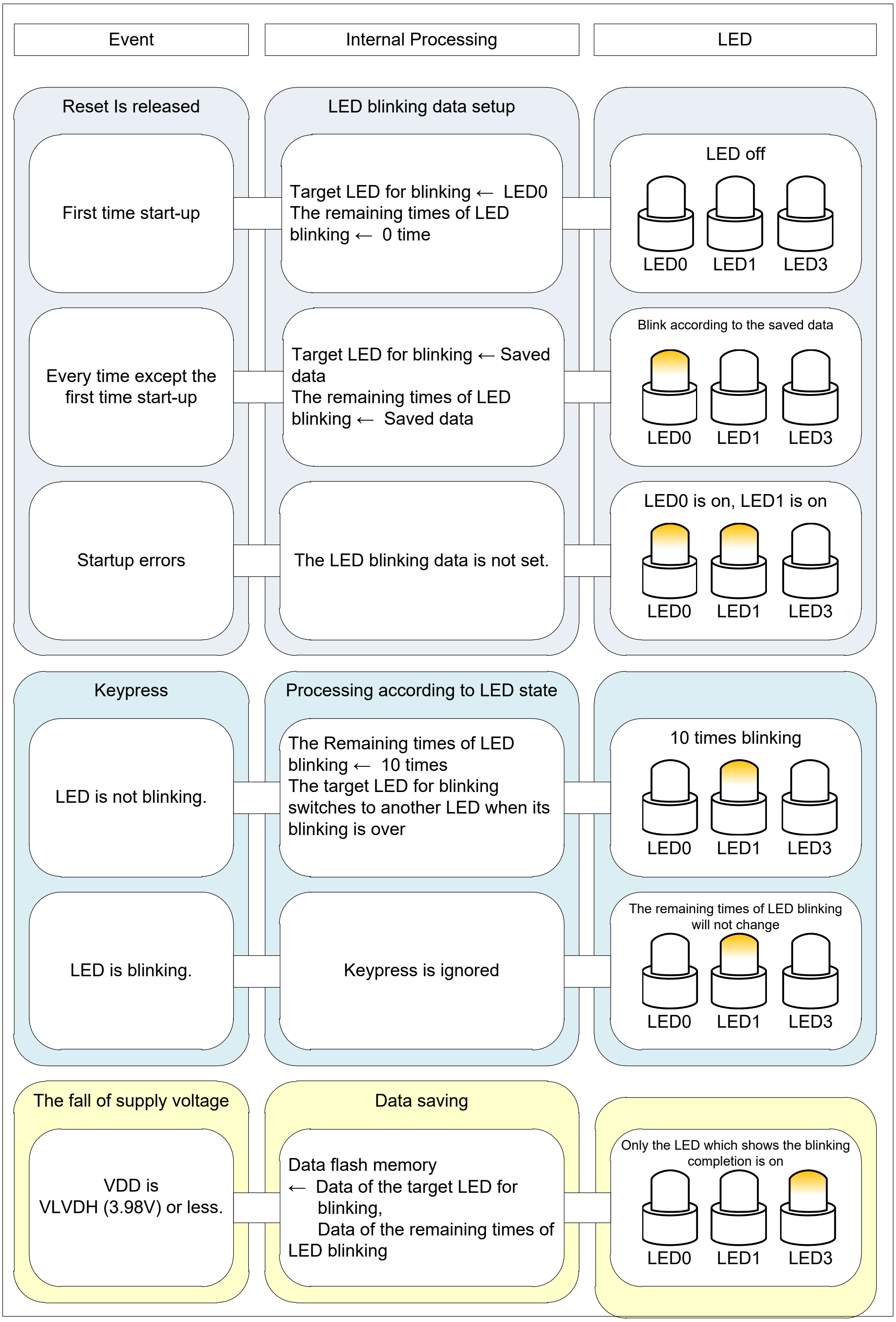


Figure 2.3 Operation Outline

2.1 Shortening of the Write Time of EEL

In order to access data flash memory from user application using the EEL, it is necessary to set data flash memory to the state that accessing the data flash memory is permitted, or to secure the resource used by FDL/EEL. Therefore, the EEL realizes the above-mentioned processes by calling some library functions. Required processes are as follows.

- FDL_Init function: Initialization of RAM used by FDL.
- FDL_Open function: Set data flash memory to the state that accessing the data flash memory is permitted.
- EEL_Init function: Initialization of RAM used by EEL.
- EEL_Open function: Set data flash memory to the state that can be managed.
- EEL_Execute function (STARTUP command): Changing into the state in which EEPROM emulation execution is possible.

However, if the above-mentioned preparation processings are performed at a low voltage, it may become power disconnect during the data saving processing. Therefore, in this application note, in order to shorten the data saving time, the data saving processing done by EEL is divided into two phases which are executed separately, the preparation phase and the saving phase.

Figure 2.4 shows the data saving processing when it is performed by a batch processing. **Figure 2.5** shows the data saving processsing when it is performed by a two-step processing. The saving phase takes 991[µs] in the case of batch processing, and takes 683[µs] in the case of two-step processing.

Remark. The measurements described in this application note are the actual measured value calculated by using EEL RL78 Pack02 Ver1.01 on the integrated development environment CubeSuite+.

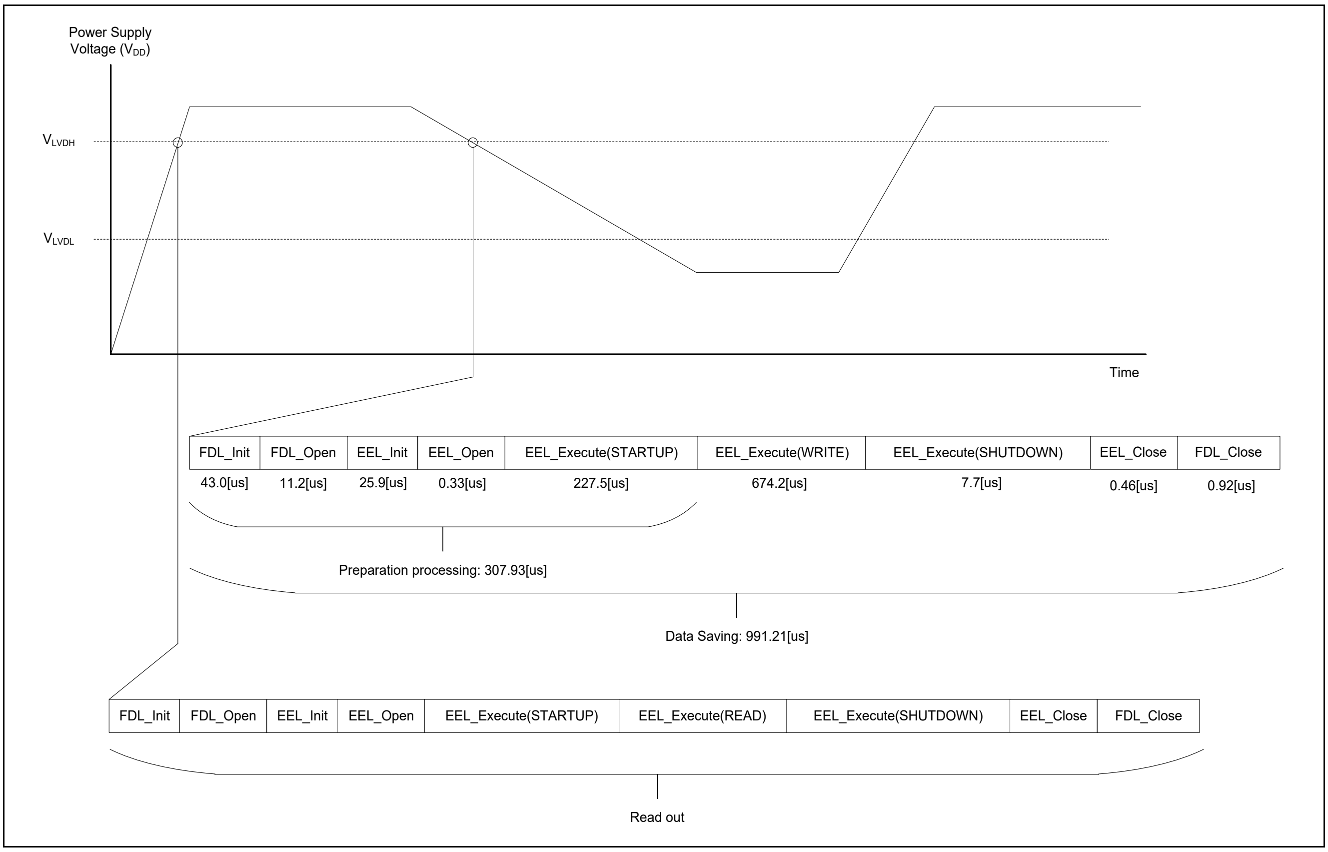


Figure 2.4 Data Saving Processing (by a batch processing)

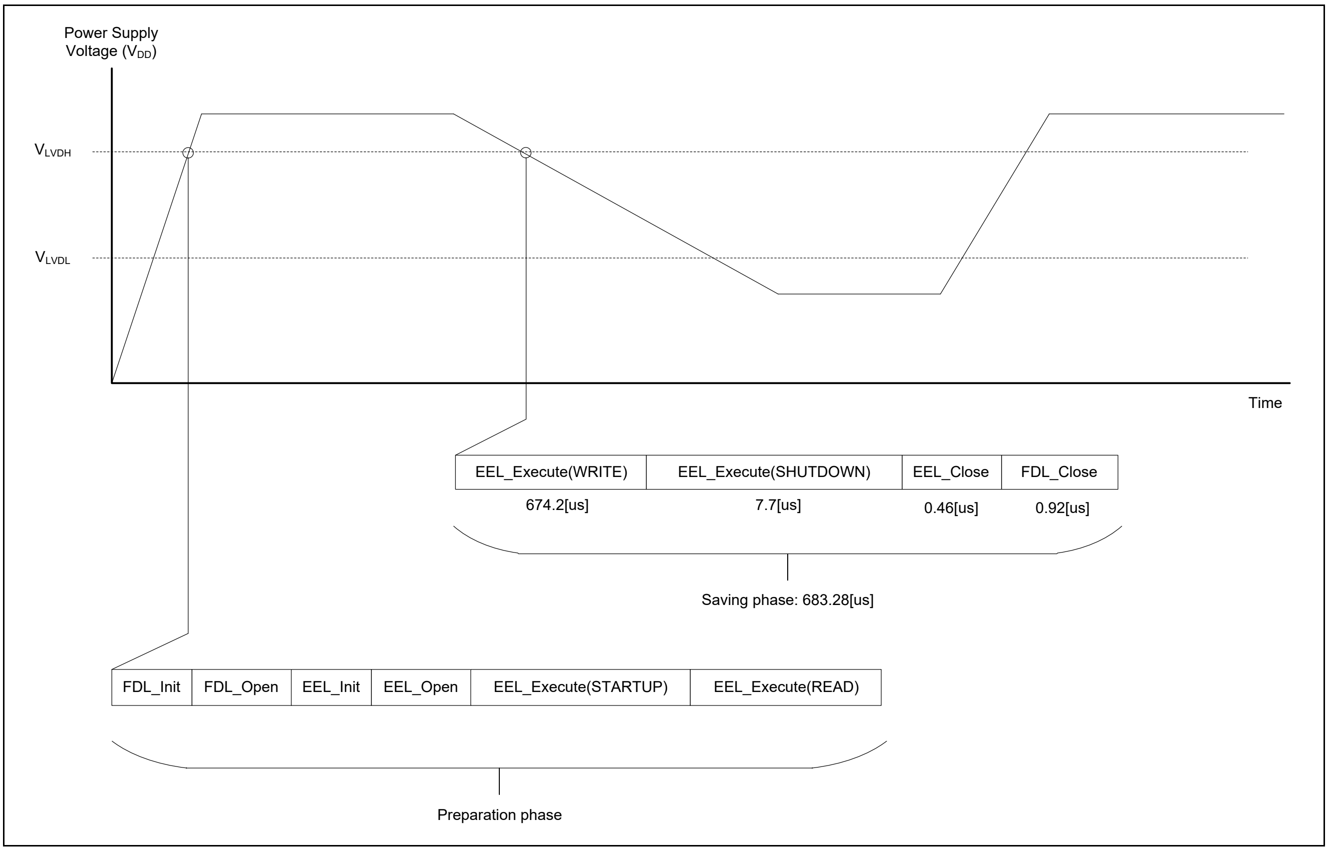


Figure 2.5 Data Saving Processing (by a two-step processing)

2.2 EEL Architecture

The operation principle of the EEL is explained. The EEL manages data in the data area and manages data ID in the reference area. These areas are configured in the same block and managed by each block. The next block will be used if unused area of the block currently used is run out. This chapter explains how to use the data flash memory in the EEL.

2.2.1 EEL Pool

The EEL pool is a user-defined data flash area that is accessible by the EEL. The user-created program can access the data flash only by using this EEL pool in the data flash via the EEL.
The EEL pool size must be specified with the number of blocks in the data flash of the target device. For the procedure to specify the number of blocks, see Section 2.3 EEL Initial Values to be set by User.

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

State	Description
Active	Only a single EEL block is active at a time to store defined data. The active block circulates in data flash blocks allocated in the EEL pool.
Invalid	No data is stored in invalid blocks. EEL blocks are marked as invalid by the EEL or become invalid in the case of erasure blocks.

Figure 2.6 shows an exemplary pool configuration for a device with 4 KB data flash.

When no writable area is remaining in the active block (block 1 in the example) and data can no longer be stored (failure in write command), a new active block is selected in a cyclic manner and the current valid data set is copied to this new active block. This process is referred to as refresh. After the EEL_CMD_REFRESH command is executed, the previous active block becomes invalid and only a single active block exists.

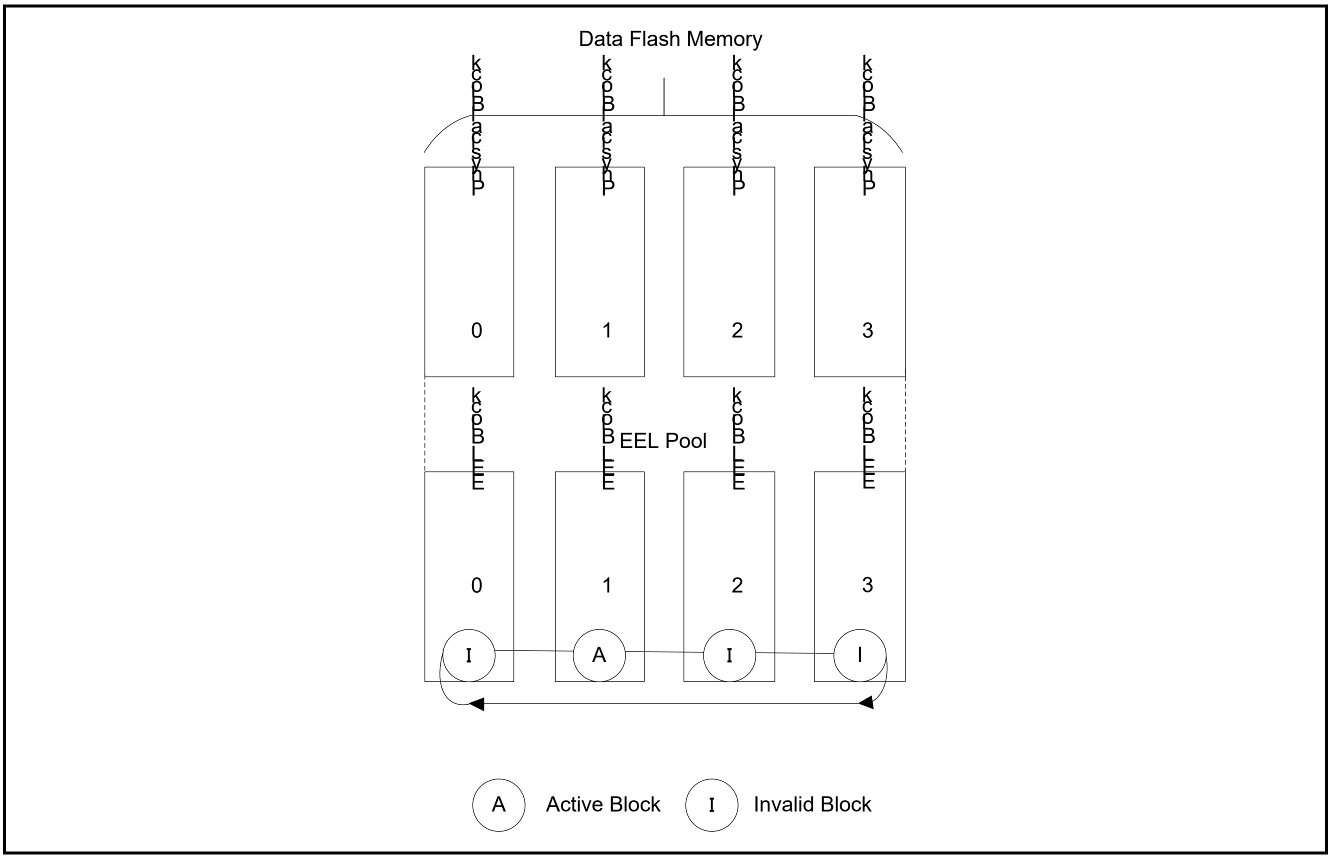


Figure 2.6 EEL Pool Structure

The overall life cycle of a block in the EEL pool is shown in **Figure 2.7**. The EEL block switches between active and invalid state.

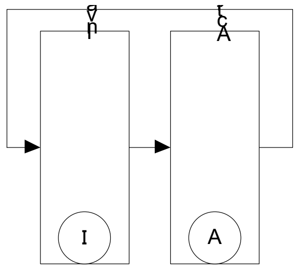


Figure 2.7 Life Cycle of an EEL Block

2.2.2 EEL Block

Figure 2.8 EEL Block Structure (Example of RL78/L13 (R5F10WMG)) shows detailed block structure used by the EEL. The EEL block is divided into three utilized areas: the block header, the reference area and the data area.

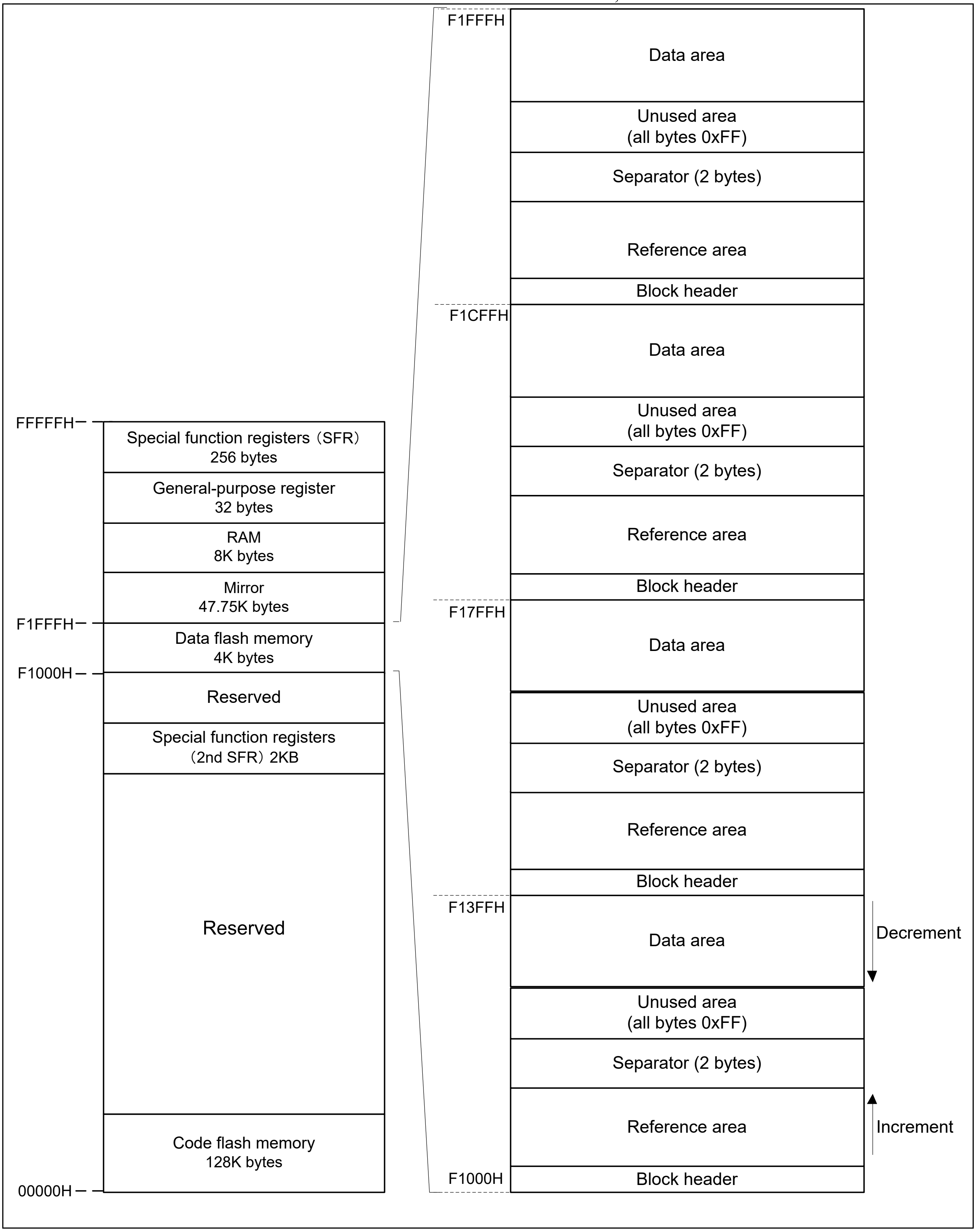


Figure 2.8 EEL Block Structure (Example of RL78/L13 (R5F10WMG))

Table 2.1 Components of EEL Block

Name	Description
Block header	The block header contains all block status information needed for the block management within the EEL-pool. It has a fixed size of 8 bytes.
Reference area	The reference area contains reference data which are required for the management of data. When data is written, this area extends in the address increment direction.
Data area	The data area contains user data. When data is written, this area extends in the address decrement direction.

Between reference area and data area, there is an unused area. With each EEL data update (i.e. the data is written), this area is reduced successively. However, at least two bytes of space is always required in between reference area and data area for management and separation of these areas. This is indicated by the separator in **Figure 2.8**.

2.3 EEL Initial Values to be Set by User

As the initial values for the EEL, be sure to set the items indicated below. In addition, before executing the EEL, be sure to execute the high-speed on-chip oscillator. The high-speed on-chip oscillator must also be activated when using the external clock. Parentheses indicated on the right-hand side of each setup are connected with the numbers in the following page.
Setup of each item has been tailored to this application.

<FDL user include file (fdl_descriptor.h)> Note 1,2

```
#define    FDL_SYSTEM_FREQUENCY     24000000      :(1) Operation frequency
#define    FDL_WIDE_VOLTAGE_MODE                  :(2) Voltage mode
#define    FDL_POOL_BLOCKS          0             :(3) FDL pool size
#define    EEL_POOL_BLOCKS          4             :(4) EEL pool size
```

<EEL user include file (eel_descriptor.h)>Note 1, 2

```
#define    EEL_VER_NO               1             :(5) Number of stored data items
```

＜EEL user type include file (eel_user_types.h)> Note 1, 2

```
typedef    eel_u08                  type_A;       :(6) Data size
```

<EEL user-created progaram file (eel_descriptor.c)> Note 1, 2

```
__far const eel_u08 eel_descriptor[EEL_VAR_NO+2] =    :(7) Data size of the data ID
{
   (eel_u08)(EEL_VAR_NO),  /* variable count */    \
   (eel_u08)(sizeof(type_A)),   /* id = 1 */        \
   (eel_u08) (0x00),        /* zero terminator */   \
};
```

Note 1: The macros and macro names that are being used have common parameters with the EEL, so changes should be made to numerical values only.
Note 2: After initializing the EEPROM emulation blocks (after executing the EEL_CMD_FORMAT command), do not change the values. If the values are changed, reinitialize the EEL blocks (by executing the EEL_CMD_FORMAT command).

(1) Operation frequency

This sets an operation frequency which is used in RL78 microcontrollers. Note1
The setting value is set to the FDL_Init frequency parameter by the following expressions.
In this application note, since the operation frequency of CPU is 24 MHz, it is set to 24.

Note1: This setting is a value required to control data flash memory. This setting does not change the operation frequency of RL78 microcontrollers. In addition, this operation frequency is not the frequency of the high-speed on-chip oscillator.

(2) Voltage mode

This sets the voltage mode of data flash memory. Note 2
FDL_WIDE_VOLTAGE_MODE is not defined: Full-speed mode
FDL_WIDE_VOLTAGE_MODE is defined: Wide voltage mode
This application note does not define FDL_WIDE_VOLTAGE_MODE because it is operated in full speed mode.

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

(3) FDL pool size **Note 3**

Specify 0.

Note 3: A user defined data flash area which is accessible by the FDL.

(4) EEL pool size **Note 4**

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

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

(5) Number of stored data items

Specify the number of data items to be used in the EEPROM emulation. A value of 1 to 64 can be set. In this application note, since one kind of data is managed, the number of stored data items is set to 1.

(6) Data size registration

The data size of every data ID is registered into an EEL descriptor table.
There are 8 standarized sizes: type_A, type_B, type_C, type_D, type_E, type_F, type_X, and type_Z. It is necessary to change size according to the size of the user data.
In this application note, since one data type of 1-byte (LED blinking state) is managed, type_A which is 1-byte long, is used for data ID1.

(7) Data size of data ID

A table to define the data size of each data ID is provided below. This is called an EEL descriptor table. The EEL can only add identifiers while the program is running. Data to be written must be registered in the EEL descriptor table in advance like the processing described in (6).

EEL Descriptor Table

__far const eel_u08 eel_descriptor [Number of stored data items (1) + 2]

EEL_VAR_NO
Byte size of data ID1 (type_A)
0x00

・EEL_VAR_NO

User-specified number of data items used in the EEL

・Byte size of Data IDx

User-specified size of user data (in bytes)

・Termination area (0x00)

Specify 0 as the termination information.

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

The total size of user data that can be used in the EEPROM emulation is limited. If refresh processing is taken into consideration, it is need to place all of the user data and an unused area which is big enough for one or more data in one block.
The number of stored data items that can be used differs depending on the size of user data that is actually stored.
The following shows the calculation method of the size which can actually be used in the writing of user data, and the maximum number of times one block can be written with user data.

[Maximum usable size of one block that can be used to write the user data]
Size of one block of data flash memory: 1024 bytes
Size required for EEPROM emulation block management: 8 bytes
Free space necessary as termination information (separator): 2 bytes

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

[Calculating the size for writing each user data item]
Size of each written user data item = data size + reference data size (2 bytes)
Since the size of the data written in in this application note is 1 byte, the size of user data will be 3 bytes.

[Number of times one block can be written]

Because the maximum usable size of one block is 1014 bytes and the user data size is 3 bytes:

Number of times one block can be written = 1014 / 3 = 338 times

It is necessary to perform refresh processing for every 338 times writing in this application note. Refresh processing is performed within preparation processing (Refer to 2.1 Shortening of the Write Time of EEL). At the time of doing refresh processing, preparation processing time is longer for 6.74[ms] in comparison with usual time.

2.5 Notes for Using EEL

Notes when using EEL are shown below.

- The data flash memory cannot be read during data flash memory operation by the EEL.
- The watchdog timer does not stop during the execution of the EEL.
- The EEL does not support multitask execution. Do not execute the EEL functions during interrupt processing.
- Before starting the EEPROM emulation, be sure to start up the high-speed on-chip oscillator first. The high-speed on-chip oscillator must also be activated when using the external clock.
- In address above 0xFFE20 (0xFE20), do not place data buffer (argument) or stack which is used by EEL functions and FDL functions.
- 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.
- It is recommended to use at least three blocks of the data flash memory in order to use the EEL.
- The EEL does not support multitask execution. When executing an EEL function on the OS, do not execute in from two or more tasks.
- About an operation frequency of RL78 microcontrollers and an operation frequency value set by the initializing function (FDL_Init), be aware of the following points:
  - When using a frequency lower than 4 MHz as an 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 Note as an operation frequency of RL78 microcontrollers, a certain frequency can be used as an operation frequency of RL78 microcontrollers.
  - This operation frequency is not the frequency of the high-speed on-chip oscillator.

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

3. Operation Check Conditions

The sample code described in this application note has been checked under the conditions listed in the table below.

Table 3.1 Operation Check Conditions

Item	Description
Microcontroller used	RL78/L13(R5F10WMGA)
Operating frequency	High-speed on-chip oscillator (fHOCO) clock: 24 MHz (Standard) CPU/peripheral hardware clock (f_{CLK}): 24 MHz
Operating voltage	5.0V (Operation is possible over a voltage range of 4.1V to 5.5V) LVD operation (V_{LVI}): Reset mode V_{LVDH}(rising edge 4.06V / falling edge 3.98V) V_{LVDL}(falling edge 2.75V)
CubeSuite+ Ver. development environment Integrated development environment C compiler ・EEL	 CubeSuite+ V2.01.00 from Renesas Electronics Corp. CA78K0R V1.70 from Renesas Electronics Corp. EELRL78 Pack02 Ver1.01 NOTE
Board to be used	Renesas Starter Kit for RL78/L13 (R0K5010WMC000BR)

Note: Use and evaluate the latest version.

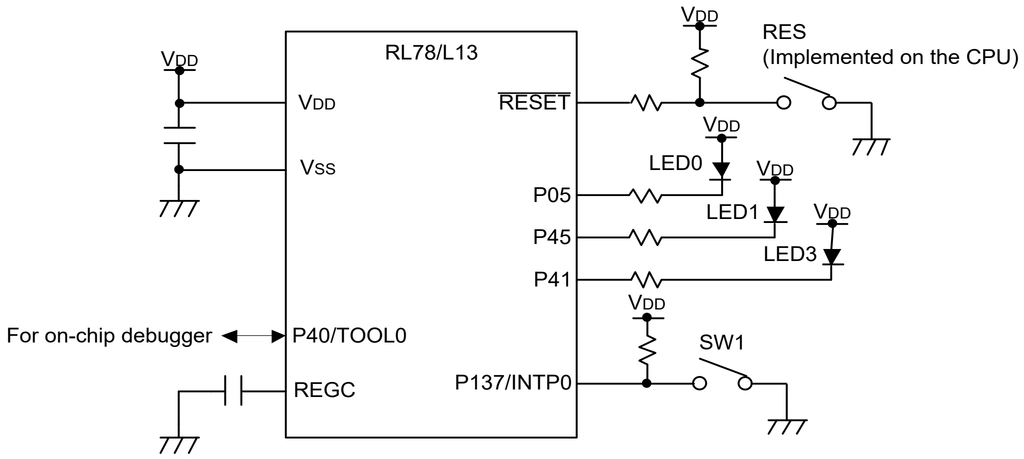
4. Related Application Notes

The application notes that are related to this application note are listed below for reference.
RL78 Family EEPROM Emulation Library Pack02 (R01US0068EJ) User Manual
Data Flash Access Library (Type T02 (Tiny), European Release) (R01US0061ED0100) Application Note
EEPROM Emulation Library (Type T02 (Tiny), European Release) (R01US0070ED0102) Application Note

5. Description of the Hardware

5.1 Hardware Configuration Example

Figure 5.1 shows an example of the hardware connection.



Cautions: 1. The purpose of this circuit is only to provide the connection outline and the circuit is simplified accordingly. When designing and implementing an actual circuit, provide proper pin treatment and make sure that the hardware's electrical specifications are met (connect the input-only ports separately to V_{DD} or V_{SS} via a resistor).

2. V_{DD} must be held at not lower than the reset release voltage (V_{LVDH}) that is specified as LVD.

Figure 5.1 Connection Example

5.2 List of Pins to be Used

Table 5.1 lists pins to be used and their functions.

Table 5.1 Pins to be Used and their Functions

Pin Name	I/O	Description
P05	Output	LED On (LED0) control port
P45	Output	LED On (LED1) control port
P41	Output	LED On (LED3) control port
P137/INTP0	Input	Key input (SW1) port

6. Description of Software

6.1 Operation Outline

In this application, LED0 or LED1 blinks 10 times by a keypress. The data used for LED blinking is saved to data flash memory when the supply voltage becomes too low. The saved data is read when system restarts, and the interrupted blinking processing is continued.

When reset is ended, the system reads the blinking state data (target LED for blinking, and the remaining times of LED blinking) by EEL from data flash memory where the data has been saved.

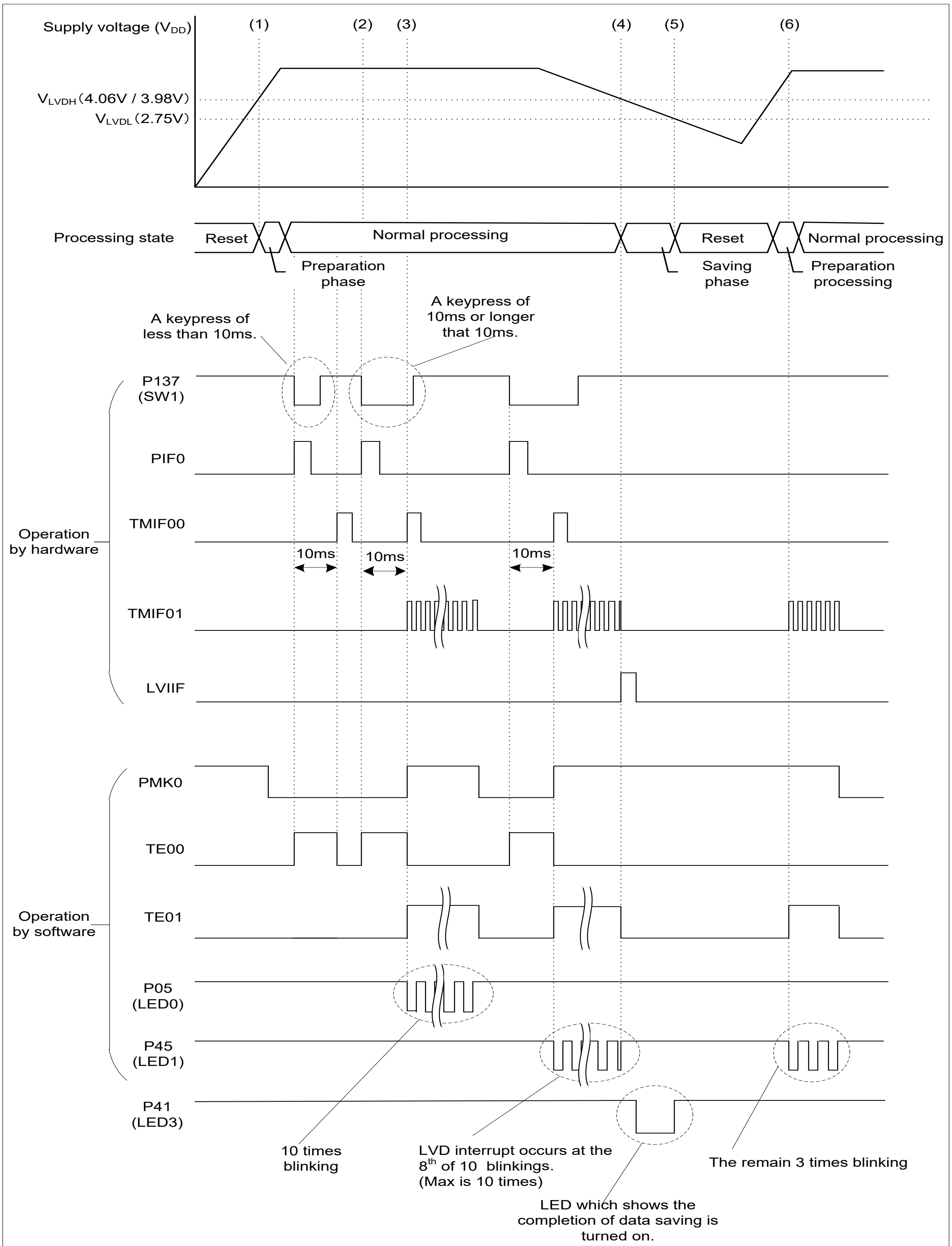
Next, after completing 10 times blinking at intervals of 500 ms according to the blinking state data, the LED stops blinking and the system becomes the waiting state for keypress.
If the key is pressed in a state in which no LED is blinking, the LED that had not been blinking just before will start to blink. The keypress becomes invalid while LED is blinking.

The fall of power supply voltage is detected by LVD function. If the fall of power supply voltage is detected, the LED blinking state data (target LED for blinking, and the remaining times of LED blinking) is saved to data flash memory by the EEL, LED3 which shows the completion of data saving will be lit up, and then the mode moves to the STOP mode.

Moreover, if an error occurs when accessing data flash memory with an EEL function, LED0 and LED1 will be lit up and the mode shifts into the STOP mode.

1. Sets the input and output ports.
   - LED lighting control (for LED0, LED1, LED3): Configure P05, P45, and P41as the output ports. (LED0, LED1, and LED3 are off.)
   - Keypress: Configure P137/INTP0 for detecting INTP0 falling edges. (Interrupt servicing disabled)
2. Start EEPROM emulation by doing the RAM initialization processing of FDL/EEL and the preparation processing.
   Specifically, functions are called in following order.
   FDL_Init, FDL_Open, EEL_Init, EEL_Open, EEL_Execute(Startup)
3. LED blinking state (Data ID: 1) is read and then the target LED will blinked at intervals of 500 ms according to this LED blinking state.
   - Higher 4 bits of the read data show the target LED for blinking (0000B: LED0, 0001B: LED1). And lower 4 bits show the data (Range: 0000B - 1010B) of remaining times of LED blinking.
   - The target LED for blinking to is set as LED0 and the data of remaining times of LED blinking is set as 0, when data does not exist.
   - Blinking according to the read data is started.
   - EEL_Execute (Read) function is used for reading of data.
4. Ensure the space for data writing via evaluating the free space in a block before writing the data.
   - If free space is lower than 3 bytes (smaller than the size of user data), perform refresh processing to secure a space in another block and move the latest data.
   - If free space is 3 bytes or more, refresh processing is not performed.
5. A push on a switch will blink LED 10 times.
   - Interrupt processing is started upon detection of a P137/INTP0 falling edge. Chattering is detected and, if the on state of the input lasts about 10 ms, it is recognized as a valid keypress and the LED blinking is started.
   - Target LED for blinking is changed at every keypress.
   - The next keypress can't be accepted during the period from pressing key to the end of the LED blinking.
6. When a LVD interrupt occurs, the remaining times of LED blinking and the number of the target LED for blinking will be saved to data flash memory. The LED3 turns on to show completion of data saving. Then FDL/EEL will be stopped and system will go into STOP mode. Specifically, after functions are called in following order, STOP command is executed.
   EEL_Execute(Write), EEL_Execute(Shutdown), EEL_Close、FDL_Close
7. If an error occurs when accessing data flash memory by the EEL, it will go to the stop mode after stopping FDL/EEL and turning on both LED0 and LED1.
   Specifically, after functions are called in following order, STOP command is executed.
   EEL_Execute(Shutdown), EEL_Close, FDL_Close
8. If reset occurs, it will return to 1.

Figure 6.1 shows the timing chart.



Remark 1. Preparation Processing: It get prepared to write data flash memory by using EEL for saving data. The saved data is read and the LED blinking is started. The free space of a block is checked. It will refresh if there is no space for writing.

Remark 2. Normal processing: Waiting for keypress. If a key is pressed, the target LED will blink 10 times and then waits for keypress again. The target LED for blinking changes to another by keypress.

Remark 3. Data saving: Dada of the remaining times of LED blinking and targeted LED for blinking is saved into data flash memory. It turns LED off and turns LED3 on which shows the completion of data saving. It invalidates INTP0 interruption and goes into STOP mode.

Figure 6.1 Timing Chart

(1) Release from the reset state

After reset is ended the CPU starts running, initialization of RAM used by FDL/EEL and the LED blinking data reading are performed. Then LED linking is started according to the read data.

(2) Keypress of SW1

The count of the interval timer for chattering evasion is started.

(3) Detection of keypress

It will be regarded as a valid keypress if the detection performed 10 ms after the previous keypress shows that SW1 is still being pressed. The interval timer of 500 ms is started, and LED goes to blink.

(4) Low voltage detection

LED blinking data (the remaining times of LED blinking, the target LED for blinking) will be written in data flash memory (Data ID: 1), and the blinking LED will be off. Moreover, after turning on LED (LED3) which shows the completion of data saving, It invalidates INTP0 interruption (operation of SW1 is ignored), and goes into STOP mode. In the example of Figure 6.1, LED blinking data is set to 03H (the rest of the LED0's blinking times is 3).

(5) Reset occurring

If voltage becomes below 2.75V (V_{LVDL} falling edge), reset by LVD will occur.

(6) Data saving

LED corresponding to the data saving blinks when the reset is ended. In the example of Figure 6.1, LED0 blinks 3 times.

6.2 File Configuration

The files used for the sample code is shown in **Table 6.1**. Files that are automatically generated by the integrated development environment are excluded.

Table 6.1 List of Additional Functions and Files

File Name	Outline	Remarks
r_eel_function.c	Source file for the data saving function	Additional functions: R_EEL_Initialize R_EEL_CheckStatus R_EEL_ReadData R_EEL_CheckDataRange R_EEL_WriteData
r_eel_function.h	Header file for the data saving function	-
fdl_descriptor.c	FDL descriptor source file	CubeSuite+ version
fdl_descriptor.h	FDL descriptor header file	CubeSuite+ version
eel_descriptor.c	EEL descriptor source file	CubeSuite+ version
eel_descriptor.h	EEL descriptor header file	CubeSuite+ version
fdl.h [Note 1]	Header file of FDL	CubeSuite+ version
eel.h [Note 1]	Header file of EEL	CubeSuite+ version
eel_types.h	Header file of EEL type definition	CubeSuite+ version
eel_user_types.h	Header file of EEL user type definition	CubeSuite+ version
fdl.lib [Note 1]	FDL	CubeSuite+ version
eel.lib [Note 1]	EEL	CubeSuite+ version
r_eel.dr [Note 2]	Link directive file	CubeSuite+ version

Note 1: It is a file which needs to be added separately. Refer to the cover sheet **"Correspondence between Compiler and EEL"** for more information.
Note 2: Depending on the device to be used, change may be required for the contents.

6.3 List of Option Byte Settings

Table 6.2 summarizes the settings of the option bytes.

Table 6.2 Option Byte Settings

Address	Setting	Description
000C0H/010C0H	11101111B	Disables the watchdog timer. (Stops counting after the release from the reset status.)
000C1H/010C1H	01110010B	LVD interrupt & Reset Mode Detection voltage V_{LVDH}: Rising edge 4.06V/Falling edge 3.98V V_{LVDL}: Rising edge 2.75V
000C2H/010C2H	11100000B	high-speed on-chip oscillator HS mode 24MHz
000C3H/010C3H	10000100B	Enables the on-chip debugger

6.4 List of Constants

Table 6.3 lists the constants for the sample program.

Table 6.3 Constants

Constant	Setting	Description
DATA_ID	0x01	Data ID of EEL
RET_OK	0x00	Normal response
RET_NG_DEVICE	0x01	Device exception
RET_NG_NODATA	0x02	No stored data
RET_NG_RANGE	0x03	The data is out of the valid range.
USER_DATA_SIZE	0x03	User data size
SHIFT_NUM	0x04	The number of bits to be shifted for LED blinking data
STATUS_PENDING	0xFF	Pending status
BLINK_LED_MAX	0x01	The maximum of the blinking LED's number
BLINK_NUM_MAX	0x0A	The maximum of LED blinking times
BLINK_LED0	0x00	Target LED for blinking: LED0
BLINK_LED1	0x01	Target LED for blinking: LED1
LED0	P0.5	LED0 control port (CubeSuite+ version)
	P0_bit.no5	LED0 control port (IAR version)
LED1	P4.5	LED1 control port (CubeSuite+ version)
	P4_bit.no5	LED1 control port (IAR version)
LED3	P4.1	LED3 control port (CubeSuite+ version)
	P4_bit.no1	LED3 control port (IAR version)
LED_ON	0	LED ON level
LED_OFF	1	LED OFF level
SW1	P13.7	SW1 control port (CubeSuite+ version)
	P13_bit.no7	SW1 control port (IAR version)
SW_ON	0	Keypress level of SW
SW_OFF	1	Not the keypress level of SW
BLINK_LED_MASK	0xF0	Mask for blinking target in LED blinking data
BLINK_NUM_MASK	0x0F	Mask for blinking times in LED blinking data

6.5 List of Variables

Table 6.4 lists the global variables.

Table 6.4 Global Variables

Type	Variable Name	Contents	Function Used
volatile uint8_t	**g_blink_led**	Target LED for blinking	**main** r_tau0_channel0_interrupt r_tau0_channel1_interrupt
volatile uint8_t	**g_blink_num**	Blink count	**main** r_tau0_channel0_interrupt r_tau0_channel1_interrupt
volatile uint8_t	g_lvd_flag	Supply voltage fall detection flag	main r_lvd_interrupt

6.6 List of Functions

Table 6.5 gives a list of functions that are used by this sample program.

Table 6.5 Functions

Function Name	Outline
R_Systeminit	Initialization of peripheral functions
R_PORT_Create	Initialization of the port
R_CGC_Create	Initialization of CPU clock
R_TAU0_Create	Initialization of TAU0
R_INTC_Create	Initialization of INTP
R_LVD_Create	Initialization of LVD
main	Main processing
R_MAIN_UserInit	Initialization of the main processing
R_EEL_Initialize	Initialization of EEL
R_EEL_ReadData	Read by EEL
R_EEL_CheckDataRange	Valid range check of LED blinking data
R_EEL_CheckStatus	EEL function status check
R_TAU0_Channel1_Start	Enabling TAU01
r_tau0_channel1_interrupt	TAU01 interrupt handler
R_TAU0_Channel1_Stop	Disabling TAU01
R_INTC0_Start	Enabling INTP0
r_intc0_interrupt	INTP0 interrupt handler
R_TAU0_Channel0_Start	Enabling TAU00
r_tau0_channel0_interrupt	TAU00 interrupt handler
R_EEL_WriteData	Write by EEL
R_TAU0_Channel0_Stop	Disabling TAU00
R_INTC0_Stop	Disabling INTP0
R_LVD_InterruptMode_Start	Enabling LVD interruption
r_lvd_interrupt	LVD interrupt handler
FDL_Init	Initialization of FDL
FDL_Open	FDL set up
FDL_Close	Stop FDL
EEL_Init	Initialization of RAM used by EEL
EEL_Open	EEL set up
EEL_Close	Stop EEL
EEL_Execute	Execution of the data flash operation by each command
EEL_Handler	Control of EEL during program execution
EEL_GetSpace	Confirmation of free space of EEL block

6.7 Function Specifications

This section describes the specifications for the functions that are used in the sample code.
Each function has included r_cg_macrodriver.hHeader.

R_Systeminit

Synopsis	Initialization of peripheral functions
Header	None
Declaration	void R_Systeminit(void)
Explanation	Initializes peripheral functions used in this application note.
Arguments	None
Return value	None

R_PORT_Create

Synopsis	Initialization of ports
Header	r_cg_port.h
Declaration	void R_PORT_Create(void)
Explanation	Initializes ports.
Arguments	None
Return value	None

R_CGC_Create

Synopsis	Initialization of CPU clock
Header	r_cg_cgc.h
Declaration	void R_CGC_Create(void)
Explanation	Initializes the CPU clock.
Arguments	None
Return value	None

R_TAU0_Create

Synopsis	Initialization of TAU0
Header	r_cg_timer.h
Declaration	void R_TAU0_Create(void)
Explanation	Initializes TAU0 in order to use TAU00 and TAU01 as interval timers.
Arguments	None
Return value	None

R_INTC_Create

Synopsis	Initialization of INTP
Header	r_cg_intc.h
Declaration	void R_INTC_Create(void)
Explanation	Initializes INTP.
Arguments	None
Return value	None

R_LVD_Create

Synopsis	Initialization of LVD
Header	r_cg_lvd.h
Declaration	void R_LVD_Create(void)
Explanation	Initializes LVD.
Arguments	None
Return value	None

main

Synopsis	Main Processing
Header	r_cg_tau.h r_cg_intc.h r_eel_function.h r_cg_userdefine.h
Declaration	void main(void)
Explanation	Main processing is performed.
Arguments	None
Return value	None

R_MAIN_UserInit

Synopsis	Initialization of the Main Processing
Header	r_lvd.h r_eel_function.h
Declaration	void R_MAIN_UserInit(void)
Explanation	Initializes the main function.
Arguments	None
Return value	None

R_EEL_Initialize

Synopsis	Initialization of EEL
Header	r_eel_function.h r_cg_userdefine.h
Declaration	uint8_t R_EEL_Initialize(void)
Explanation	Performs the preparation processing like initialization of RAM before starting EEPROM emulation.
Arguments	None
Return value	· Normal response: RET_OK · Abnormal termination: RET_NG_DEVICE

R_EEL_ReadData

Synopsis	Read by EEL	
Header	r_eel_function.h r_cg_userdefine.h	
Declaration	uint8_t R_EEL_ReadData(uint8_t id, uint8_t* pdata)	
Explanation	Reads data from data flash memory.	
Arguments	uint8_t id	Data ID to be read
	uint8_t* pdata	The pointer of the buffer where the read data is stored.
Return value	· Normal response: RET_OK · Abnormal termination: RET_NG_DEVICE · No data: RET_NG_NODATA	

R_EEL_CheckDataRange

Synopsis	Valid range check of LED blinking data	
Header	r_eel_function.h	
Declaration	uint8_t R_EEL_CheckDataRange(uint8_t data)	
Explanation	Checks whether LED blinking data is in the valid range.	
Arguments	uint8_t data	Object data of the check
Return value	Within the range: RET_OK Outside of the range: RET_NG_RANGE	

R_EEL_CheckStatus

Synopsis	EEL Function Status Check	
Header	r_eel_function.h r_cg_userdefine.h	
Declaration	uint8_t R_EEL_CheckStatus(eel_request_t* request_pstr)	
Explanation	Checks the execution status of EEL. It is called immediately after performing EEL_Execute.	
Arguments	eel_request_t* request_pstr	The argument when EEL_Execute is executed.
Return value	· Normal response: RET_OK · Abnormal termination: RET_NG_DEVICE · No data: RET_NG_NODATA	

R_TAU0_Channel1_Start

Synopsis	Enabling TAU01
Header	r_cg_timer.h
Declaration	void R_TAU0_Channel1_Start(void)
Explanation	Starts the count of TAU01.
Arguments	None
Return value	None

r_tau0_channel1_interrupt

Synopsis	TAU01 interrupt handler
Header	r_cg_timer.h r_eel_function.h
Declaration	__interrupt static void r_tau0_channel1_interrupt(void)
Explanation	Switches LED between ON/OFF and decrements the remaining times of LED blinking Target LED for blinking is switches to another LED when blinking is completed.
Arguments	None
Return value	None

R_TAU0_Channel1_Stop

Synopsis	Disabling TAU01
Header	r_cg_timer.h
Declaration	void R_TAU0_Channel1_Stop(void)
Explanation	Stops the count of TAU01.
Arguments	None
Return value	None

R_INTC0_Start

Synopsis	Enabling INTP0
Header	r_cg_intp.h
Declaration	void R_INTC0_Start(void)
Explanation	Enables INTP0 interruption.
Arguments	None
Return value	None

r_intc0_interrupt

Synopsis	INTP0 interrupt handler
Header	r_cg_intp.h r_cg_timer.h
Declaration	__interrupt static void r_intc0_interrupt(void)
Explanation	Starts the operation of TAU00.
Arguments	None
Return value	None

R_TAU0_Channel0_Start

Synopsis	Enabling TAU00
Header	r_cg_timer.h
Declaration	void R_TAU0_Channel0_Start(void)
Explanation	Starts the count of TAU00.
Arguments	None
Return value	None

r_tau0_channel0_interrupt

Synopsis	TAU00 interrupt handler
Header	r_cg_timer.h r_eel_function.h
Declaration	__interrupt static void r_tau0_channel0_interrupt(void)
Explanation	Checks the state of SW1 and starts to blink LED.
Arguments	None
Return value	None

R_EEL_WriteData

Synopsis	Write by EEL	
Header	r_eel_function.h r_cg_userdefine.h	
Declaration	uint8_t R_EEL_WriteData(uint8_t id, uint8_t* pdata)	
Explanation	Writes data to the data flash memory.	
Arguments	uint8_t id	Data ID to be written
	uint8_t* pdata	The pointer of the buffer where the data is written.
Return value	· Normal response: RET_OK · Abnormal termination: RET_NG_DEVICE	

R_TAU0_Channel0_Stop

Synopsis	Disabling TAU00
Header	r_cg_timer.h
Declaration	void R_TAU0_Channel0_Stop(void)
Explanation	Stops the count of TAU00.
Arguments	None
Return value	None

R_INTC0_Stop

Synopsis	Disabling INTP0
Header	r_cg_intp.h
Declaration	void R_INTC0_Stop(void)
Explanation	Disables INTP0 interruption.
Arguments	None
Return value	None

R_LVD_InterruptMode_Start

Synopsis	Enabling LVD interruption
Header	r_cg_lvd.h
Declaration	void R_LVD_InterruptMode_Start(void)
Explanation	Enables LVD interruption.
Arguments	None
Return value	None

r_lvd_interrupt

Synopsis	LVD interrupt handler
Header	r_cg_lvd.h r_eel_function.h
Declaration	__interrupt static void r_lvd_interrupt(void)
Explanation	Sets the low voltage detection flag
Arguments	None
Return value	None

FDL_Init

Synopsis	Initialization of FDL
Header	fdl.h
Declaration	fdl_status_t __far FDL_Init(const __far descriptor_t* descriptor_pstr)
Explanation	Initializes FDL. (This is a FDL library function.)
Arguments	const __far descriptor_t* descriptor_pstr — It is a pointer to the descriptor table.
Return value	· Normal termination: FDL_OK · Initialization error: FDL_ERR_CONFIGURATION

FDL_Open

Synopsis	FDL set up
Header	fdl.h
Declaration	void __far FDL_Open(void)
Explanation	Sets up FDL. (This is a FDL library function.)
Arguments	None
Return value	None

FDL_Close

Synopsis	Stop FDL
Header	fdl.h
Declaration	void __far FDL_Close(void)
Explanation	Stops FDL. (This is a FDL library function.)
Arguments	None
Return value	None

EEL_Init

Synopsis	Initialization of RAM used by EEL
Header	eel.h
Declaration	eel_status_t __far EEL_Init(void)
Explanation	Initializes RAM which is used for EEPROM emulation. (This is an EEL library function.)
Arguments	None
Return value	Normal termination: EEL_OK Initialization error: EEL_ERR_CONFIGURATION

EEL_Open

Synopsis	EEL set up
Header	eel.h
Declaration	void __far EEL_Open(void)
Explanation	Changes into the state where EEPROM emulation can be performed. (This is an EEL library function.)
Arguments	None
Return value	None

EEL_Close

Synopsis	Stop EEL
Header	eel.h
Declaration	void __far EEL_Close(void)
Explanation	This function makes the EEPROM emulation unexecutable. (This is an EEL library function.)
Arguments	None
Return value	None

EEL_Execute

Synopsis	Execution of the data flash operation by each command	
Header	eel.h	
Declaration	void __far EEL_Execute(__near eel_request_t* request_pstr)	
Explanation	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. (This is an EEL library function.)	
Arguments	__near eel_request_t* request_pstr	It is a pointer to the request structure.
Return value	None	

EEL_Handler

Synopsis	Controls the EEL while it is running
Header	eel.h
Declaration	void __far EEL_Handler(void)
Explanation	This function continues executing the EEPROM emulation processing specified for the EEL_Execute function. (This is an EEL library function.)
Arguments	None
Return value	None

EEL_GetSpace

Synopsis	Checks free space in the EEL block	
Header	eel.h	
Declaration	eel_status_t __far EEL_GetSpace(__near eel_u16* space_pu16)	
Explanation	This obtains the free EEL block space. (This is an EEL library function.)	
Arguments	__near eel_u16* space_pu16	The address at which the free space information of the current active block is input.
Return value	Normal termination: EEL_OK EEL_Init has not been executed: EEL_ERR_INITIALIZATION The EEL_CMD_STARTUP command has not finished normally: EEL_ERR_ACCESS_LOCKED A command is being executed: EEL_ERR_REJECTED	

6.8 Flowcharts

6.8.1 Overall Flowchart

Figure 6.2 shows the overall flow of the sample program described in this application note.

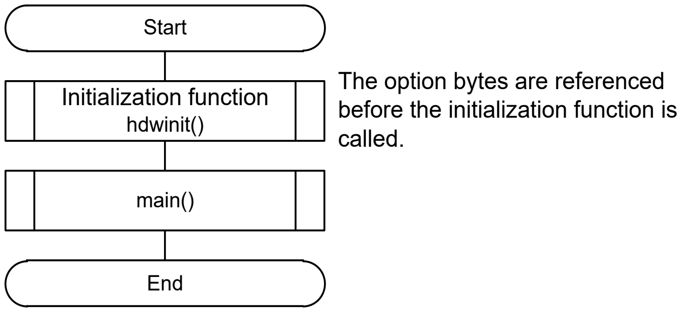


Figure 6.2 Overall Flowchart

6.8.2 Initialization of Peripheral Functions

Figure 6.3 shows the initialization of peripheral function.

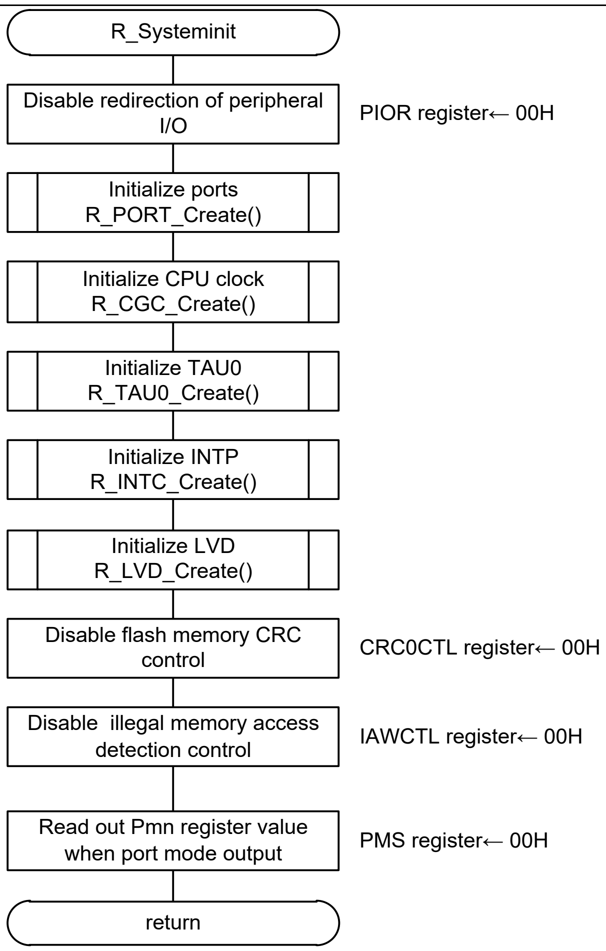


Figure 6.3 Initialization of Peripheral Functions

6.8.3 Initialization of Ports

Figure 6.4 shows the initialization of ports.

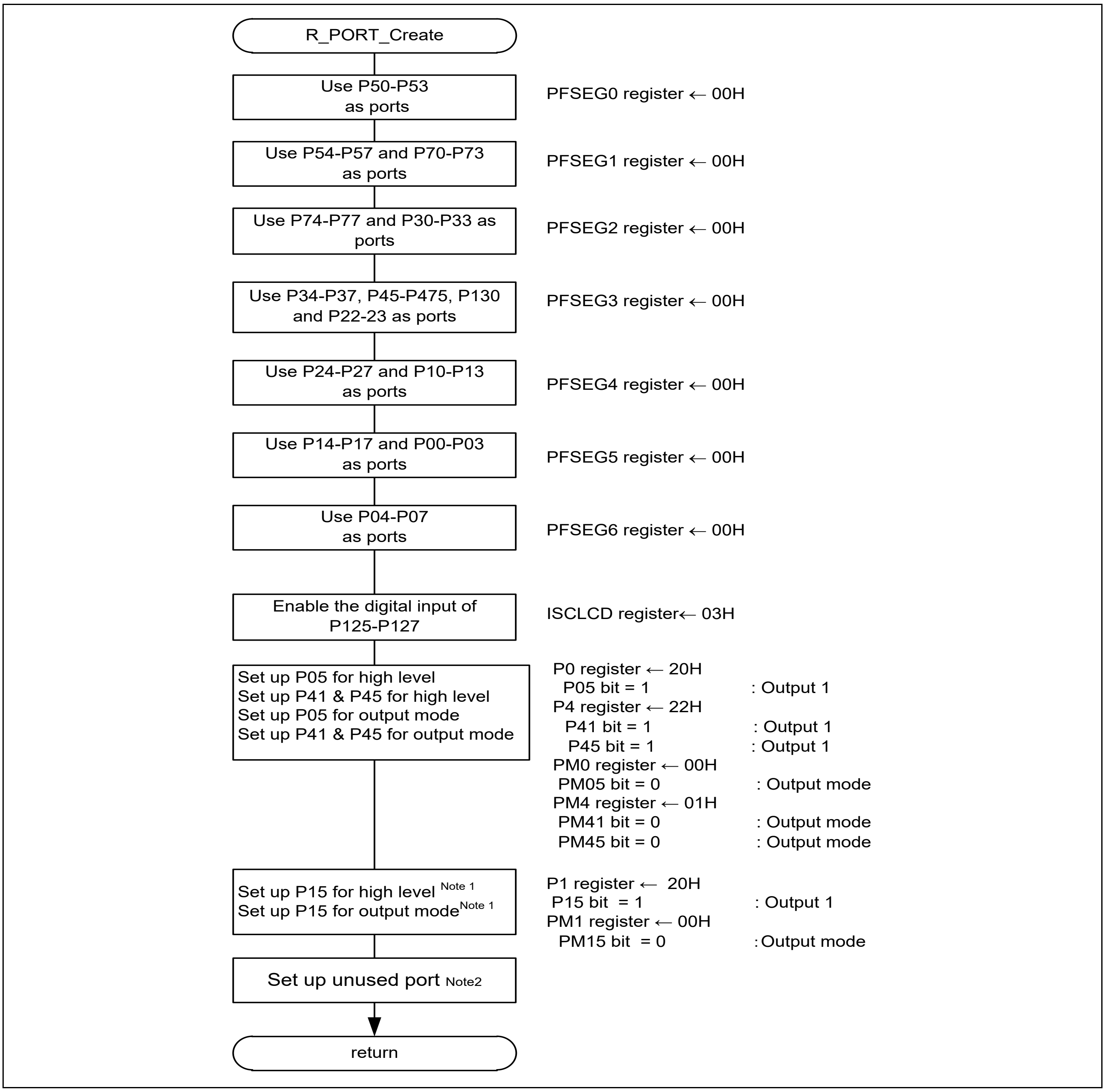


Figure 6.4 Initialization of ports

Note 1: This is a setup which makes unused LED switch off.
Note 2: Refer to "**RL78/L13 User's Manual: Hardware**" for the setup of the unused ports.
Caution: Provide proper treatment for unused pins so that their electrical specifications are observed. Connect each of any unused input-only ports to VDD or Vss via a separate resistor.

6.8.4 Initialization of CPU Clock

Figure 6.5 shows the initialization of CPU clock.

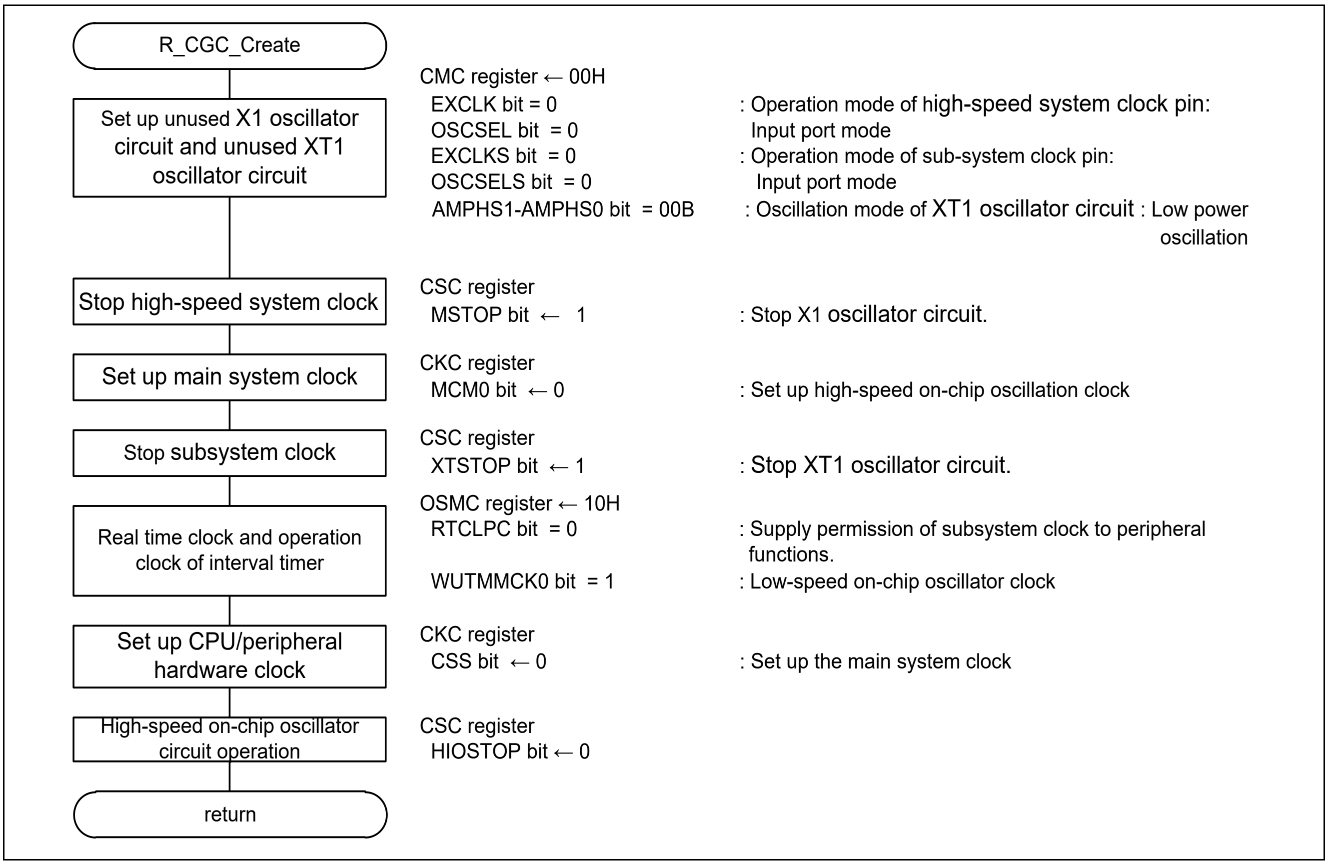


Figure 6.5 Initialization of CPU Clock

6.8.5 Initialization of TAU0

Figure 6.6 and **Figure 6.7** show the initialization of TAU0.

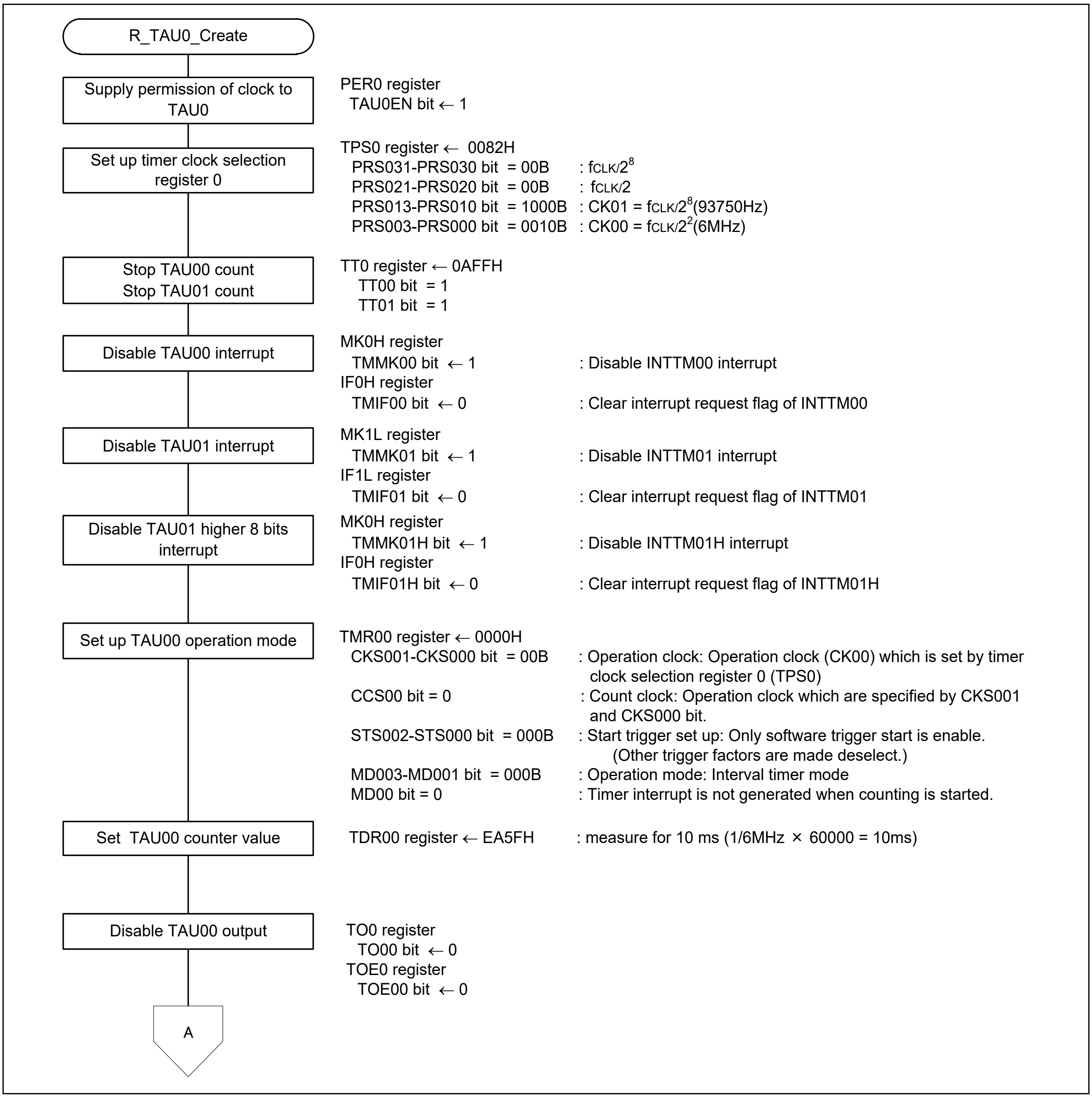


Figure 6.6 Initialization of TAU0 (1/2)

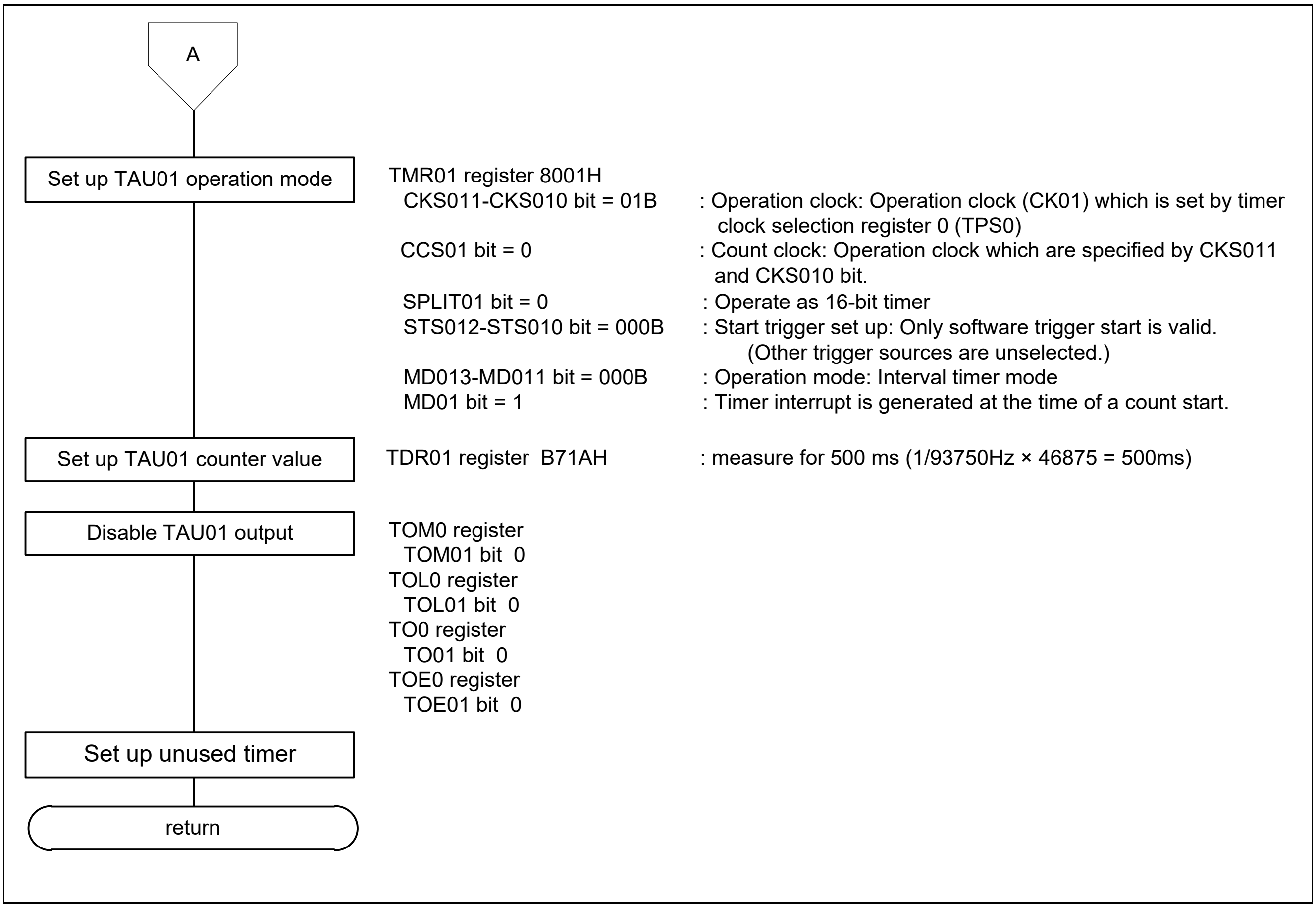


Figure 6.7 Initialization of TAU0 (2/2)

6.8.6 Initialization of INTP

Figure 6.8 shows the initialization of INTP.

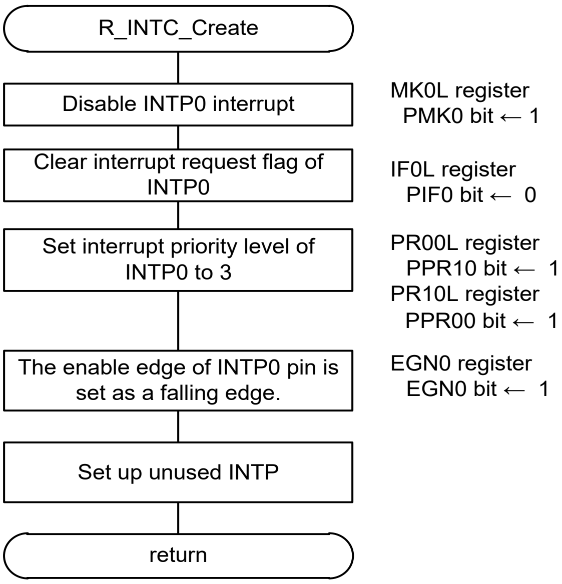


Figure 6.8 Initialization of INTP

6.8.7 Initialization of LVD

Figure 6.9 shows the initialization of LVD.

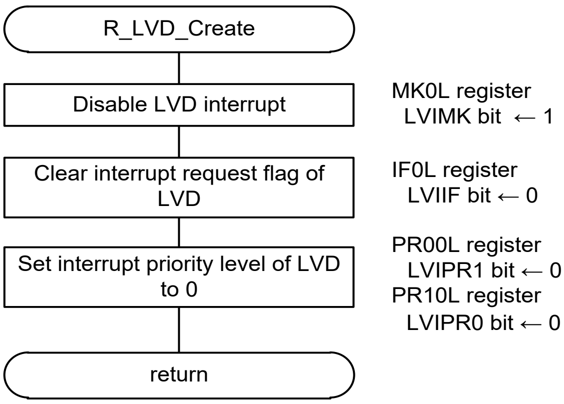


Figure 6.9 Initialization of LVD

6.8.8 Main Processing

Figure 6.10 and **Figure 6.11** show the main processing.

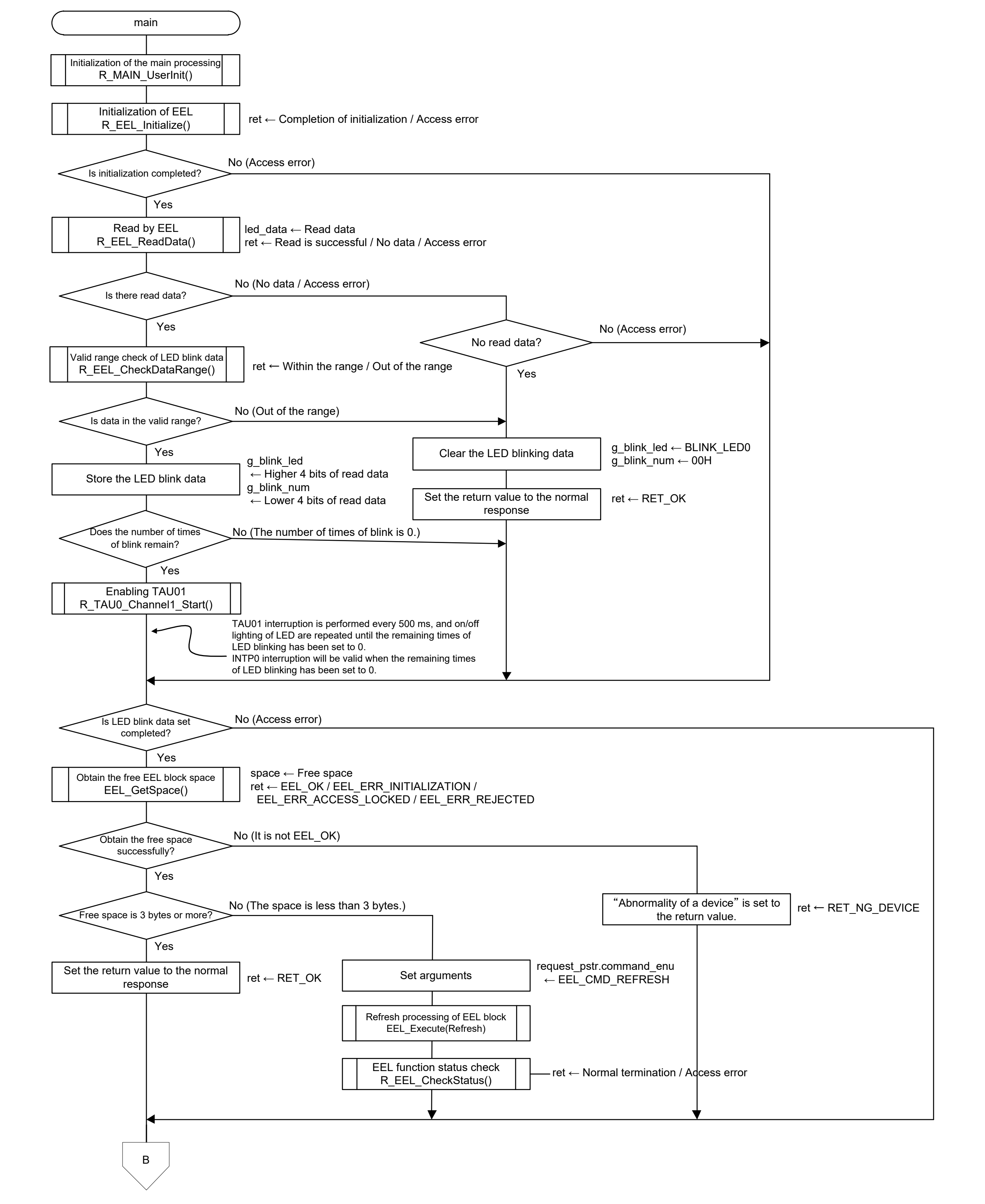


Figure 6.10 Main Processing (1/2)

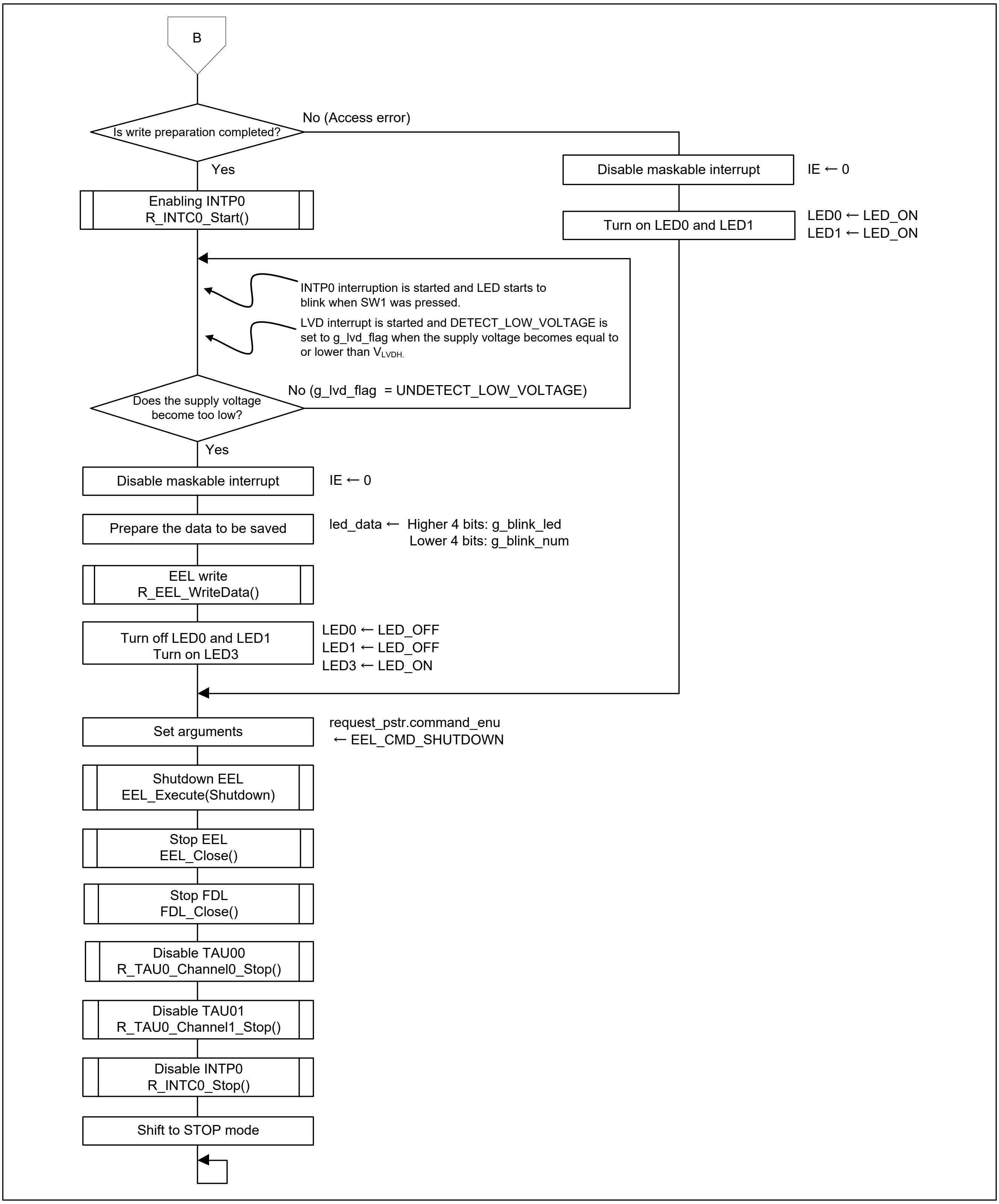


Figure 6.11 Main Processing (2/2)

6.8.9 Initialization of the Main Processing

Figure 6.12 shows the initialization of the main processing.

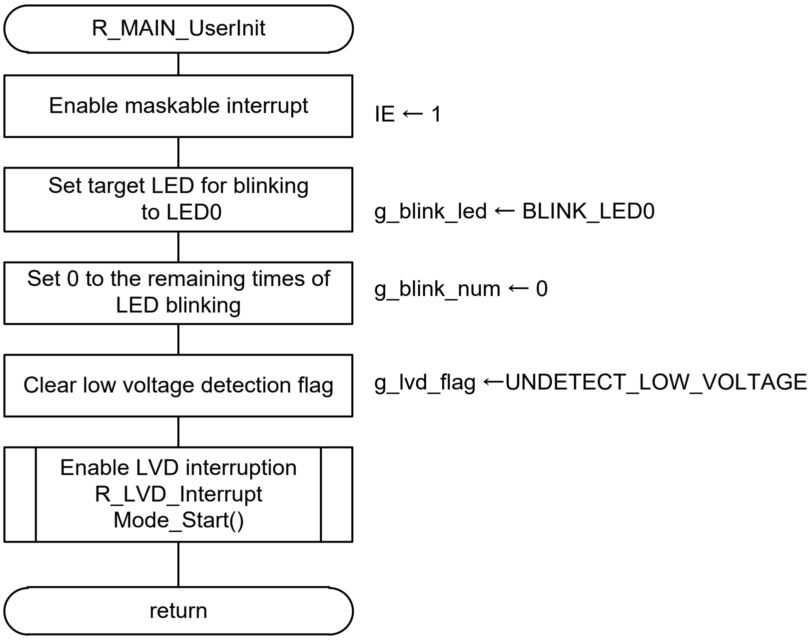


Figure 6.12 Initialization of the Main Processing

6.8.10 Initialization of EEL

Figure 6.13 shows the initialization of EEL.

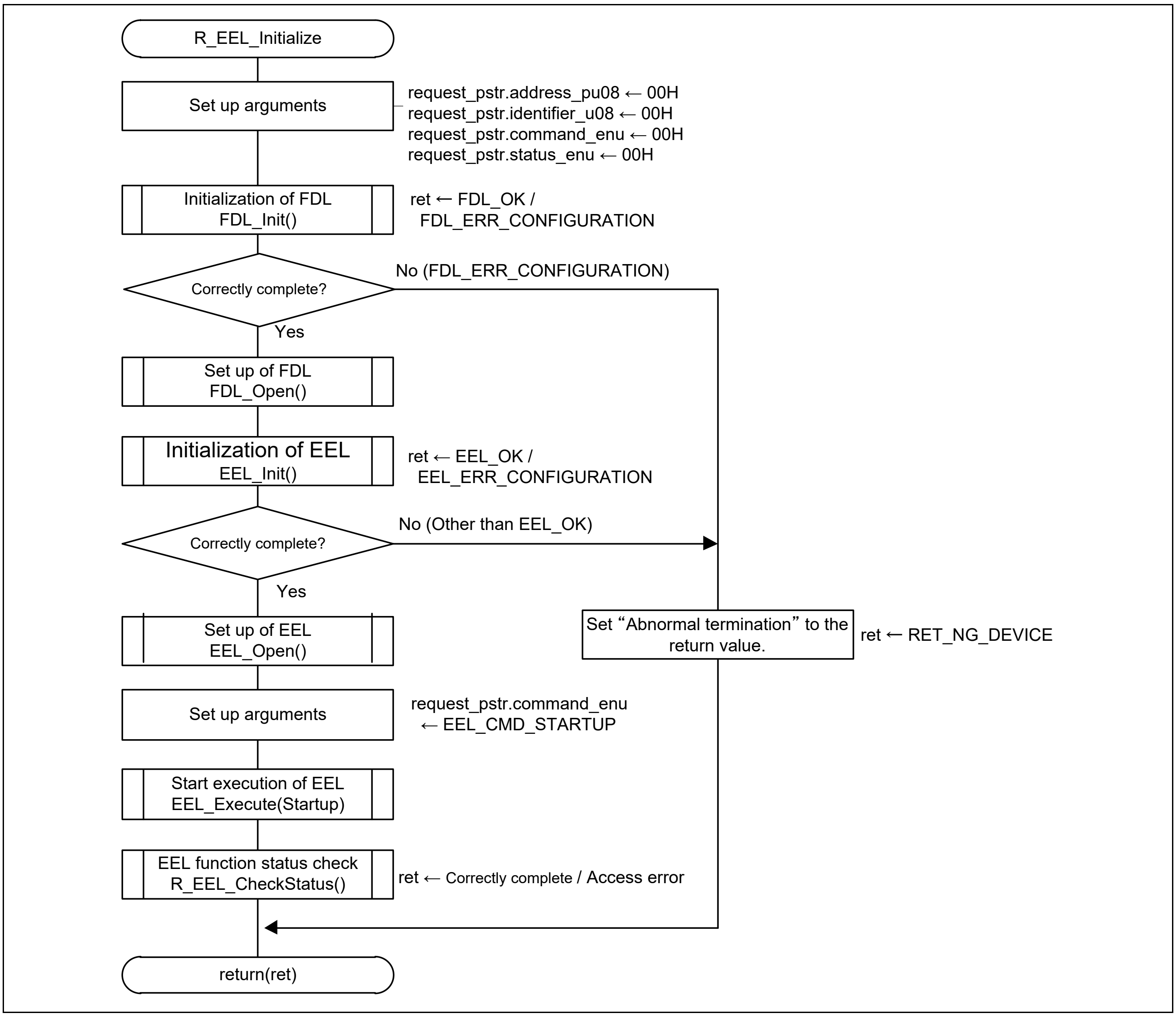


Figure 6.13 Initialization of EEL

6.8.11 Read Processing by EEL

Figure 6.14 shows how to read data by EEL.

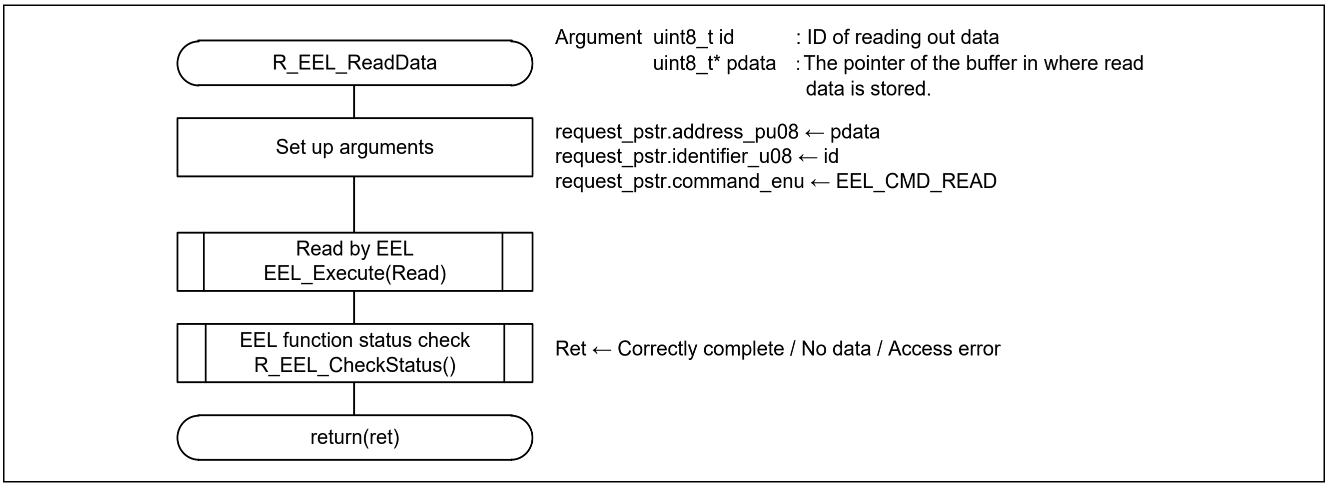


Figure 6.14 Read by EEL

6.8.12 Valid Range Check of LED blinking Data

Figure 6.15 shows the valid range check of LED blinking data.

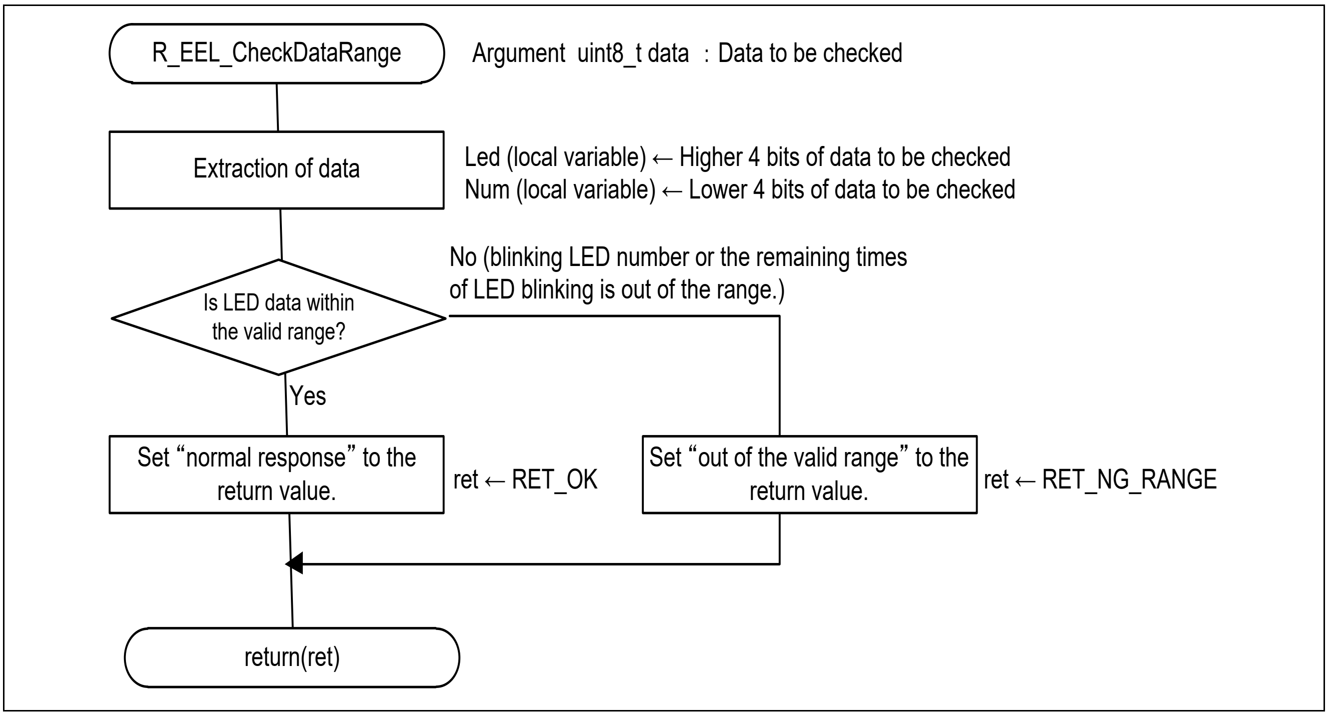


Figure 6.15 Valid Range Check of LED Blinking Data

6.8.13 EEL Function Status Check

Figure 6.16 shows the EEL function status check.

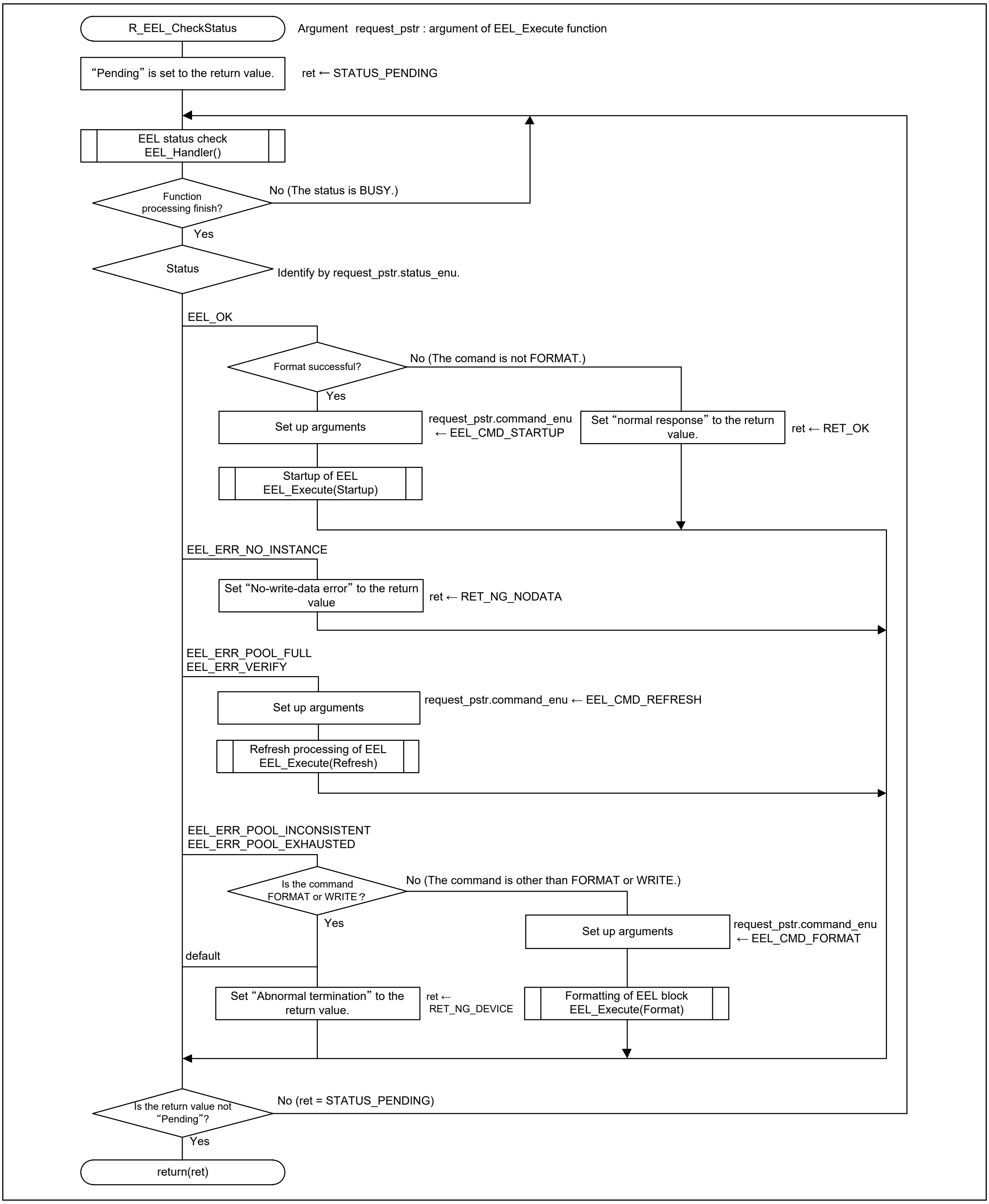


Figure 6.16 EEL Function Status Check

6.8.14 Enabling TAU01

Figure 6.17 shows how to enable TAU01.

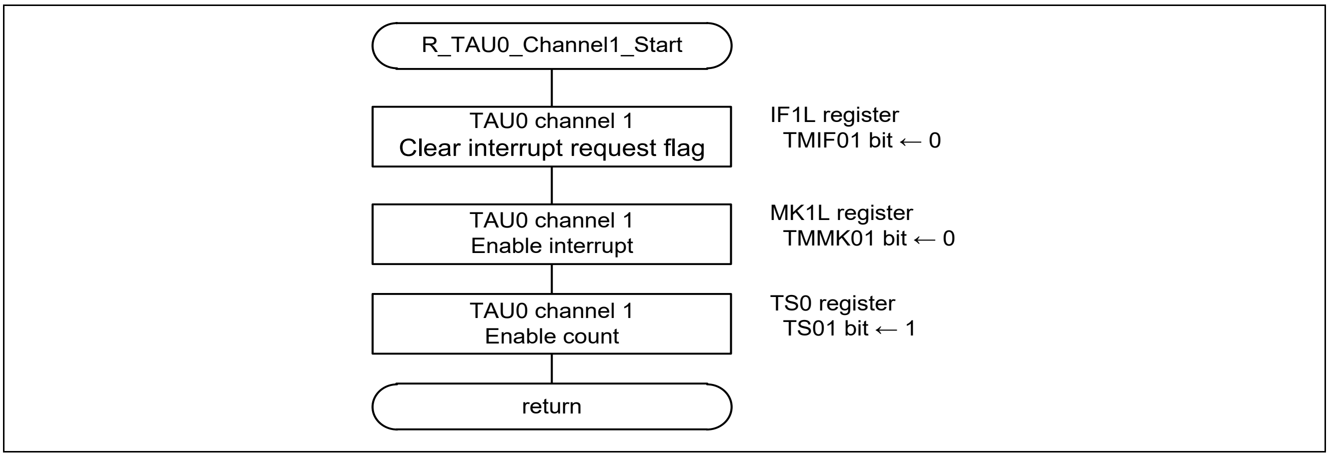


Figure 6.17 Enabling TAU01

6.8.15 TAU01 Interrupt Handler

Figure 6.18 shows the flowchart of the TAU01 interrupt handler.

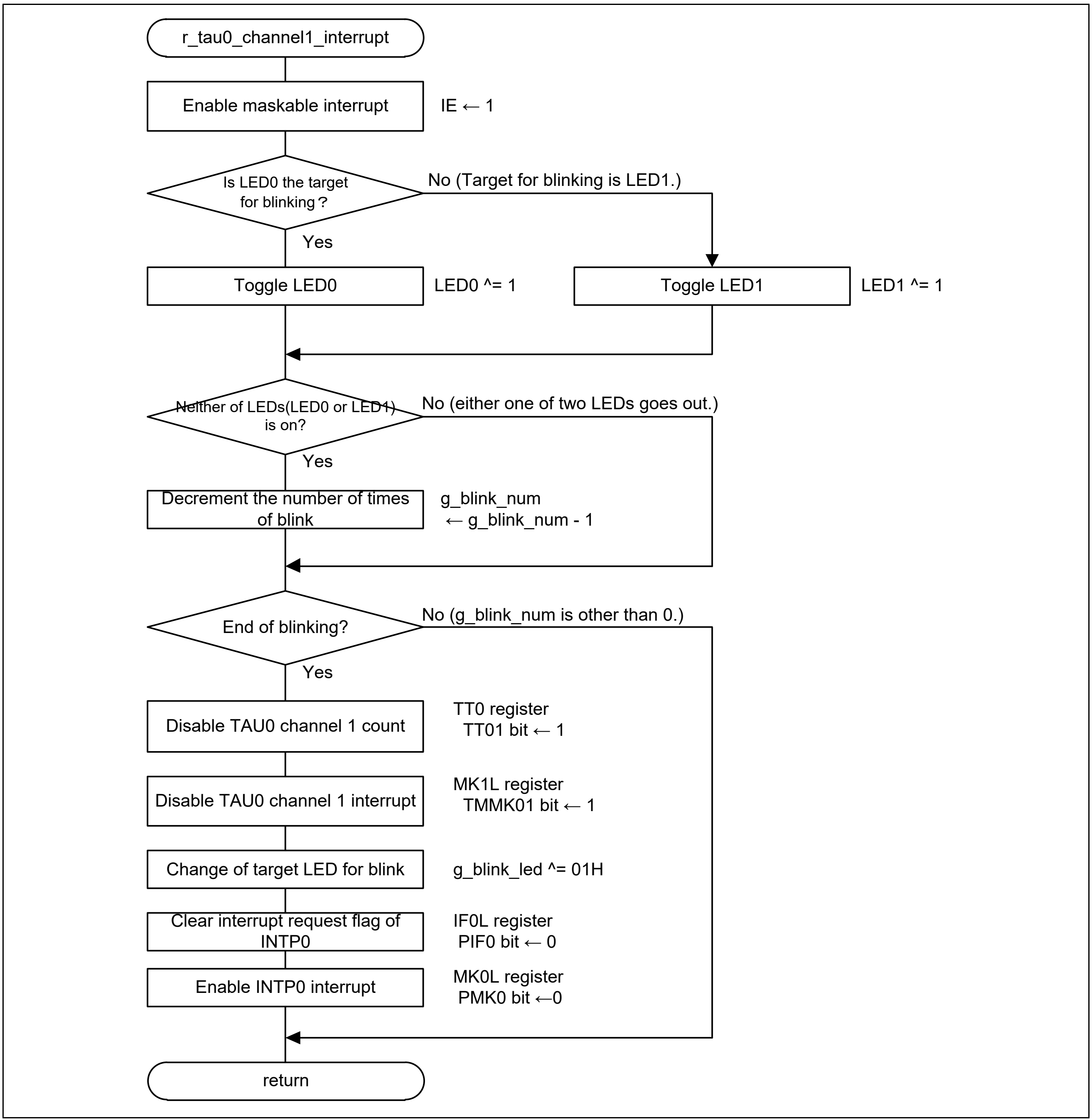


Figure 6.18 TAU01 Interrupt Handler

6.8.16 Disabling TAU01

Figure 6.19 shows how to disable TAU01.

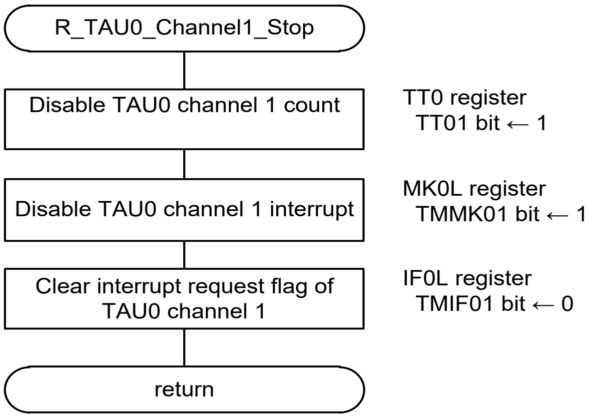


Figure 6.19 Disabling TAU01

6.8.17 Enabling INTP0

Figure 6.20 shows how to enable INTP0.

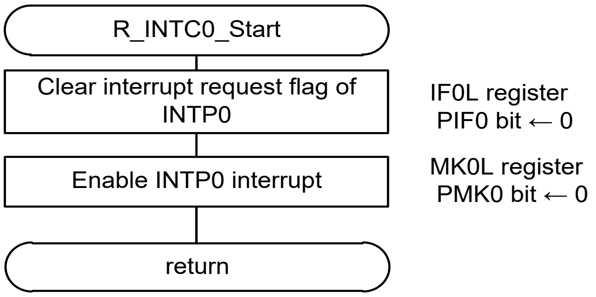


Figure 6.20 Enabling INTP0

6.8.18 INTP0 Interrupt Handler

Figure 6.21 shows the flowchart of the INTP0 interrupt handler.

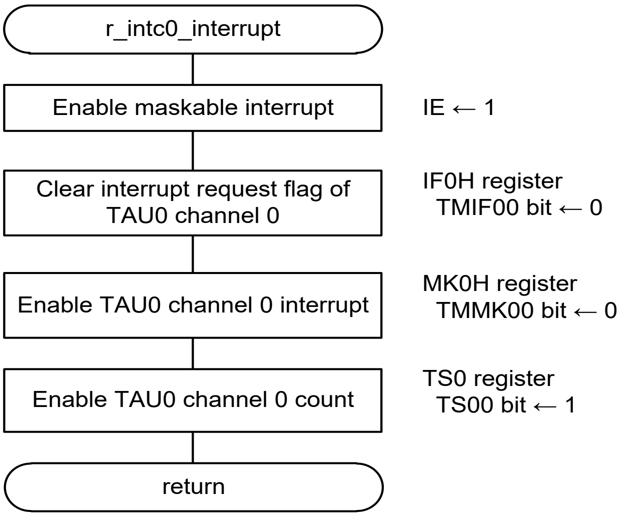


Figure 6.21 INTP0 Interrupt Handler

6.8.19 Enabling TAU00

Figure 6.22 shows how to enable TAU00.

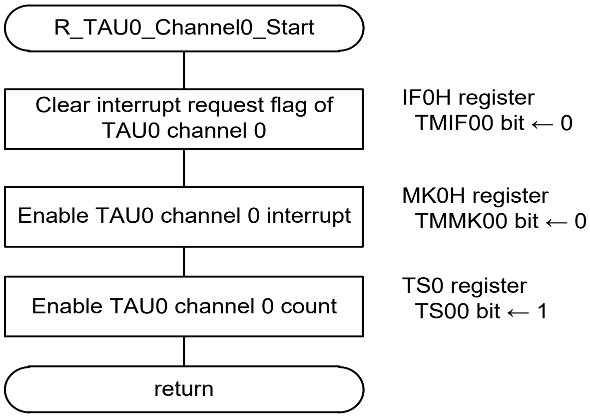


Figure 6.22 Enabling TAU00

6.8.20 TAU00 interrupt handler

Figure 6.23 shows the flowchart of the TAU00 interrupt handler.

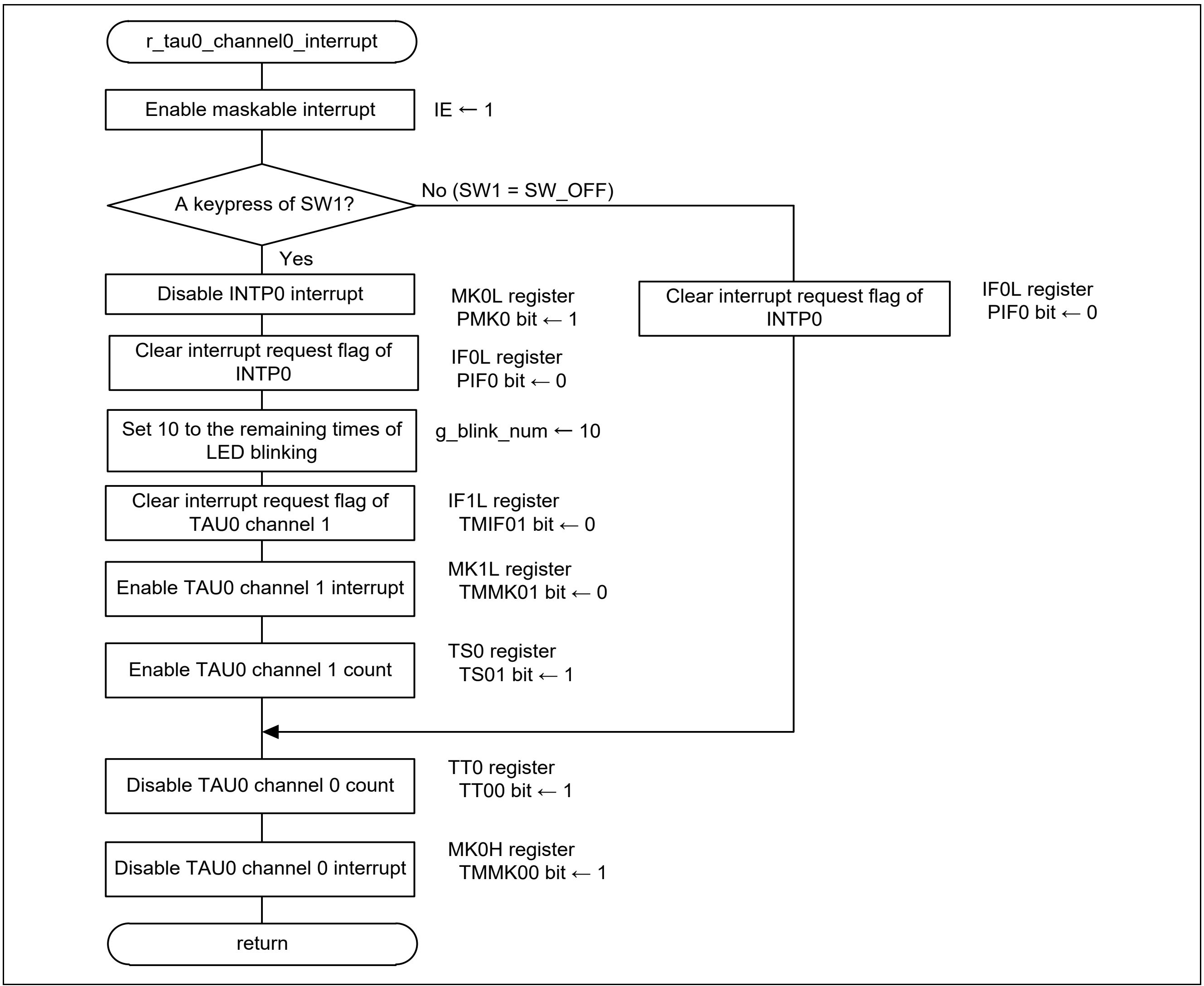


Figure 6.23 TAU00 interrupt handler

6.8.21 Write by EEL

Figure 6.24 shows how to write data by EEL.

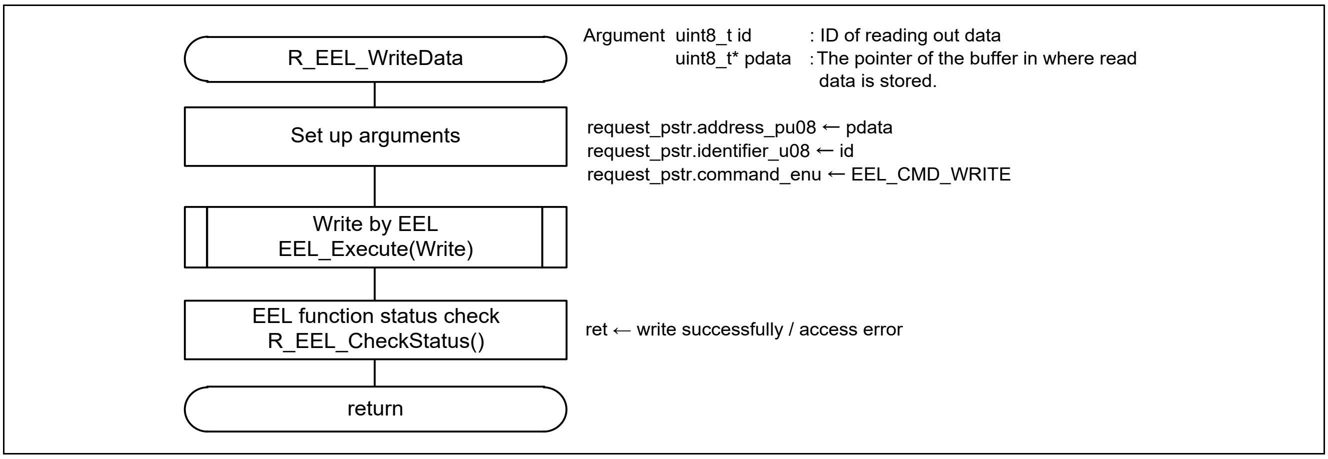


Figure 6.24 Write by EEL

6.8.22 Disabling TAU00

Figure 6.25 shows how to disable TAU00.

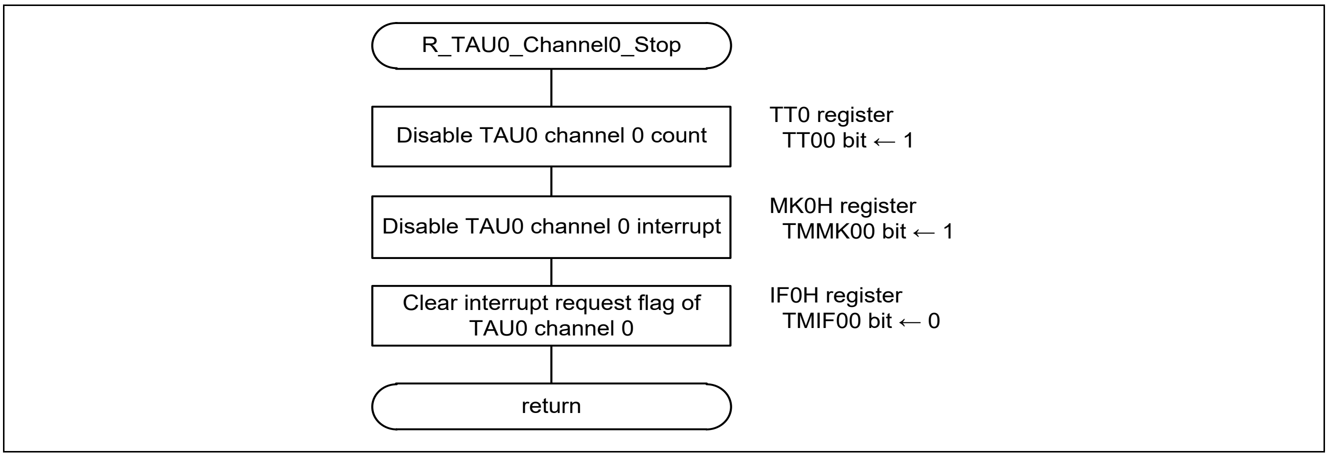


Figure 6.25 Disabling TAU00

6.8.23 Disabling INTP0

Figure 6.26 shows how to disable INTP0.

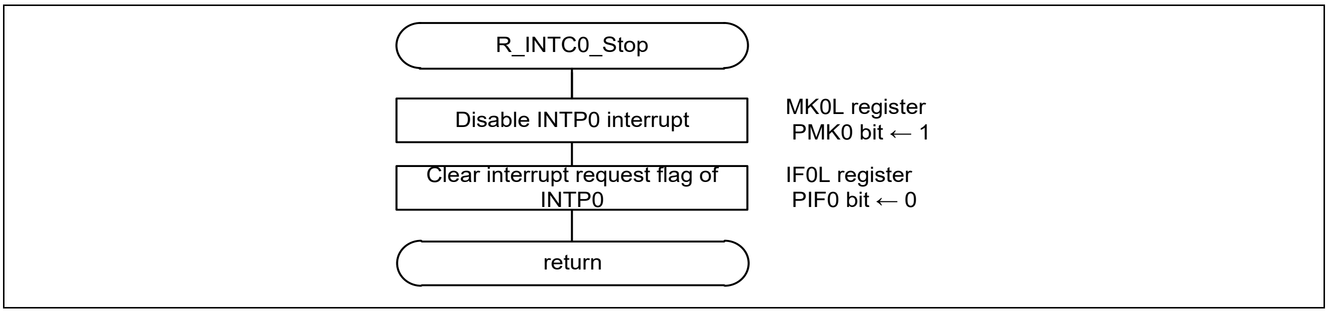


Figure 6.26 Disabling INTP0

6.8.24 Enabling LVD Interrupt

Figure 6.27 shows how to enable LVD interrupt.

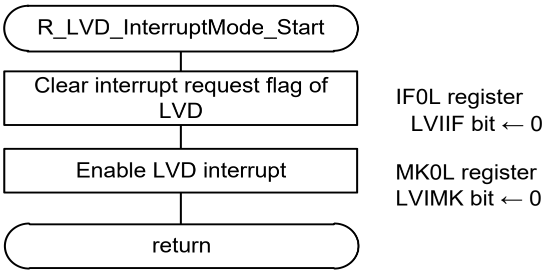


Figure 6.27 Enabling LVD Interrupt

6.8.25 LVD interrupt handler

Figure 6.28 shows the flowchart of the LVD interrupt handler.

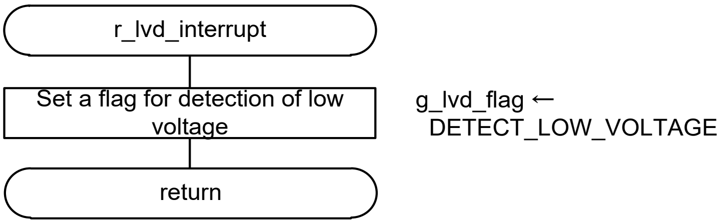


Figure 6.28 LVD interrupt handler

6.9 How to Import EEL into the Software Project

How to import EEL files used by this application into the software project is indicated below.

6.9.1 CubeSuite+ Version

(1) The following files are copied to the root directory of a project.

- fdl.h
- fdl_types.h
- fdl.lib
- eel.h
- eel_types.h
- eel.lib

(2) Right-clicks "File" at the project tree of CubeSuite+ and select the file copied according to the extension (.h, .lib, dr) by clicking "Add" and "Add an existing file".

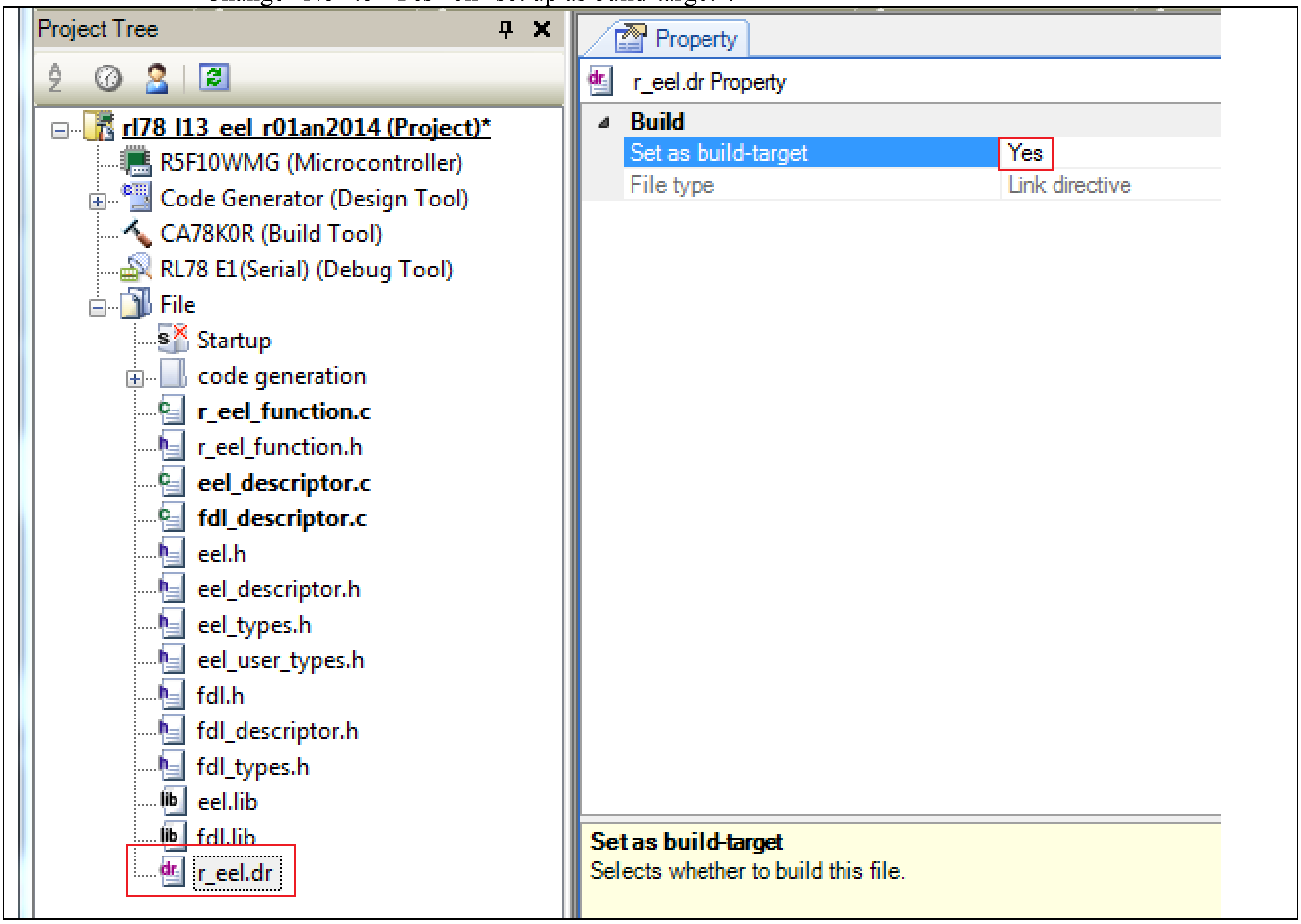
Caution

Please do not import the file in the directory smprl78 included in EEL. Be sure to use the file included in the sample code. (e.g. eel_descriptor.c)
When the file has been overwritten, correct according to "0 EEL Initial Values to be Set by User".

6.10 Modification of the Sample Code

When re-execute code generation, correction of files and projects may be needed as follows.

Environment: CubeSuite+ version
File name: r_eel.dr
Modification required: ■ Set up build-target
Open the property by right-click on r_eel.dr in the project tree of CubeSuite+.
Change “No” to “Yes” on “set up as build-target”.



Environment: CubeSuite+ version
File name: r_cg_port.c
Modification required: ■PFSEG3 register set-up code
Modify "_04_PFDEG_DEFAULT" to "_00_PFDEG_PORT".
■PFSEG6 register set-up codel
Add it before ISCLED register set-up code.

```
/***********************************************************************************************
* Function Name: R_PORT_Create
* Description  : This function initializes the Port I/O.
* Arguments    : None
* Return Value : None
************************************************************************************************
void R_PORT_Create(void)
{
    PFSEG0 = _00_PFSEG07_PORT | _00_PFSEG06_PORT | _00_PFSEG05_PORT | _00_PFSEG04_PORT;
    PFSEG1 = _00_PFSEG15_PORT | _00_PFSEG14_PORT | _00_PFSEG13_PORT | _00_PFSEG12_PORT | _00_PFSEG11_PORT |
             _00_PFSEG10_PORT | _00_PFSEG09_PORT | _00_PFSEG08_PORT;
    PFSEG2 = _00_PFSEG23_PORT | _00_PFSEG22_PORT | _00_PFSEG21_PORT | _00_PFSEG20_PORT | _00_PFSEG19_PORT |
             _00_PFSEG18_PORT | _00_PFSEG17_PORT | _00_PFSEG16_PORT;
    PFSEG3 = _00_PFSEG30_PORT | _00_PFSEG29_PORT | _00_PFSEG28_PORT | _00_PFSEG27_PORT | _00_PFSEG26_PORT |
             _00_PFSEG25_PORT | _00_PFSEG24_PORT | _00_PFDEG_PORT;
    PFSEG4 = _00_PFSEG38_PORT | _00_PFSEG37_PORT | _00_PFSEG36_PORT | _00_PFSEG35_PORT | _00_PFSEG34_PORT |
             _00_PFSEG33_PORT | _00_PFSEG32_PORT | _00_PFSEG31_PORT;
    PFSEG5 = _00_PFSEG46_PORT | _00_PFSEG45_PORT | _00_PFSEG44_PORT | _00_PFSEG43_PORT | _00_PFSEG42_PORT |
             _00_PFSEG41_PORT | _00_PFSEG40_PORT | _00_PFSEG39_PORT;
    PFSEG6 = _00_PFSEG47_PORT | _00_PFSEG48_PORT | _00_PFSEG49_PORT | _00_PFSEG50_PORT;
    ISCLCD = _02_ISCVL3_VALID | _01_ISCCAP_VALID;
```

Environment: CubeSuite+ version
File name: **r_cg_port.h**
Modification required: ■PFDEG set-up macro
Add "_00_PFDEG_PORT" before "_04_PFDEG_DEFAULT" by 0x00U as macro definition.

```
/*
    LCD port function registers 03 (PFSEG03)
*/
/* Port (other than segment output)/segment outputs specification of Pmn pins (PFSEGxx) */
#define _00_PFSEG24_PORT        (0x00U) /* used the P34 pin as port (other than segment output) */
#define _01_PFSEG24_SEG         (0x01U) /* used the P34 pin as segment output */
#define _00_PFSEG25_PORT        (0x00U) /* used the P35 pin as port (other than segment output) */
#define _02_PFSEG25_SEG         (0x02U) /* used the P35 pin as segment output */
#define _00_PFSEG26_PORT        (0x00U) /* used the P46 pin as port (other than segment output) */
#define _08_PFSEG26_SEG         (0x08U) /* used the P46 pin as segment output */
#define _00_PFSEG27_PORT        (0x00U) /* used the P47 pin as port (other than segment output) */
#define _10_PFSEG27_SEG         (0x10U) /* used the P47 pin as segment output */
#define _00_PFSEG28_PORT        (0x00U) /* used the P130 pin as port (other than segment output) */
#define _20_PFSEG28_SEG         (0x20U) /* used the P130 pin as segment output */
#define _00_PFSEG29_PORT        (0x00U) /* used the P22 pin as port (other than segment output) */
#define _40_PFSEG29_SEG         (0x40U) /* used the P22 pin as segment output */
#define _00_PFSEG30_PORT        (0x00U) /* used the P23 pin as port (other than segment output) */
#define _80_PFSEG30_SEG         (0x80U) /* used the P23 pin as segment output */
#define _00_PFDEG_PORT          (0x00U) /* used the P45 pin as port (other than segment output) */
#define _04_PFDEG_DEFAULT       (0x04U) /* PFDEG default value */
```

7. Sample Code

The sample code is available on the Renesas Electronics Website.

8. Documents for Reference

RL78/L13 User's Manual: Hardware
RL78 Family User's Manual: Software
(The latest versions of the documents are available on the Renesas Electronics Website.)

Technical Updates/Technical Brochures
(The latest versions of the documents are available on the Renesas Electronics Website.)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	2014.6.25	-	First edition issued
1.10	2022.9.9	1	Table Download link for CubeSuite+ version was changed and IAR version was deleted
		23	Table 3.1 IAR Ver. Development was deleted.
		29	Table 6.1 IAR version was deleted.
		61	6.9.2 IAR version was deleted.
		63	IAR version was deleted from Environment.

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

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

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Contact information

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