R01AN2013EJ0110
Rev.1.10
2022.9.9

RL78/L12

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

Introduction

Self-programming is a function that the microcontroller to rewrite the internal flash memory by itself. RL78/L12 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 FDL

This application note has a sample code (excluding the FDL). In order to operate this sample code, it is required to download and link FDL to the project. Refer to "6.9 How to import FDL" for details on method of linking FDL to the project.

The FDL has a CubeSuite+ version and a GNU version. However, the version of FDL 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 FDL, and the release note (or README.txt on the download source page) before using the FDL.

Correspondence between Compiler and FDL

	FDL	Download Link
CubeSuite+ Version	Data Flash Library Type04 for the RL78 Family Ver.1.05	https://www.renesas.com/software-tool/data-flash-libraries#overview
	RENESAS_FDL_RL78_T04E_V 1.20	http://www.renesas.eu/updates?oc=EEPROM_EMULATION_RL78
GNU Version	RENESAS_FDL_RL78_T04E_V 1.20	http://www.renesas.eu/updates?oc=EEPROM_EMULATION_RL78

Target Device

RL78/L12

FDL used in this application note supports other devices of RL78.

RL78/D1A, RL78/F12, RL78/F13, RL78/F14, RL78/G12, RL78/G13, RL78/G14, RL78/G1A,

RL78/G1C, RL78/G1E, RL78/I1A, RL78/L13, RL78/L1C

Confirm by the latest user's manual of the FDL about the supported device of FDL.

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 Table **1.1**.

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

Table 1.1 List of Self Programing Library

Name of library	Corresponding 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 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.

1.2 Outline of 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).

1.3 Proper Use of FDL and 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 **Table 1.2**.

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 FDL and 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 **Table 1.3**. 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 Table 1.3 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

(e.g. Operation Frequency 24MHz, Full Speed Mode)

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.

(1) 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 advantages when replacing the function of EEPROM IC with data flash memory by using FDL, 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.
- An optimization would be possible according to the varying conditions by simplifying control area based on the data to save.

1.4.2 Difference from EEPROM IC

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

- The size of the flash memory which can be used by user decreases due to data control areas are required.
- The program which communicates with EEPROM IC is not required. Instead, FDL and a user-created program which can control data are necessary.
- Instead of a communications program with EEPROM IC, FDL and a program for data management are required.

  It is necessary to erase data in units of one block in advance in order to overwrite the same address.

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 FDL, LED3 which shows the completion of data saving will be lit up, and then the mode moves to the STOP mode. 0x00 is added to data as a termination symbol.

Moreover, if an error occurs when accessing data flash memory with FDL 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 . 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 shows that the rest of the LED1's blinking times is 5.

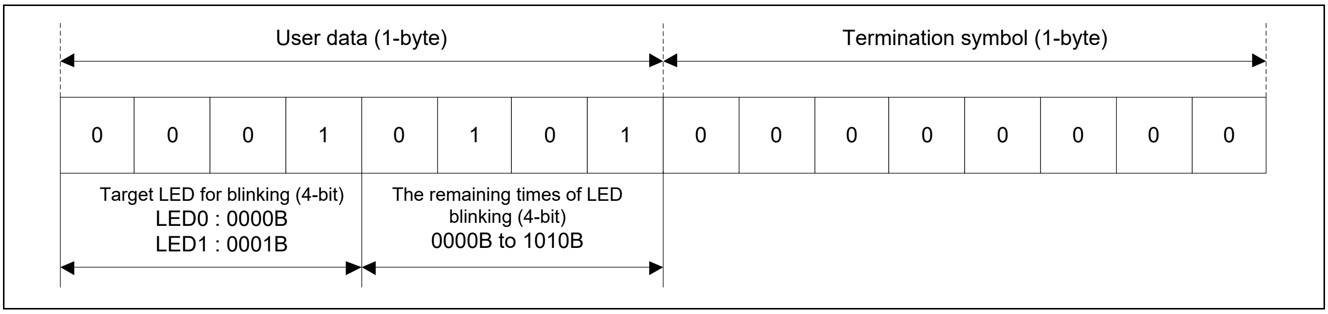


Figure 2.1 Stored Data

Table 2.1 shows the required peripheral functions and their uses. **Figure 2.1** 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
P30	LED lighting control (LED0)
P42	LED lighting control (LED1)
P52	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 blink interval time (500ms)

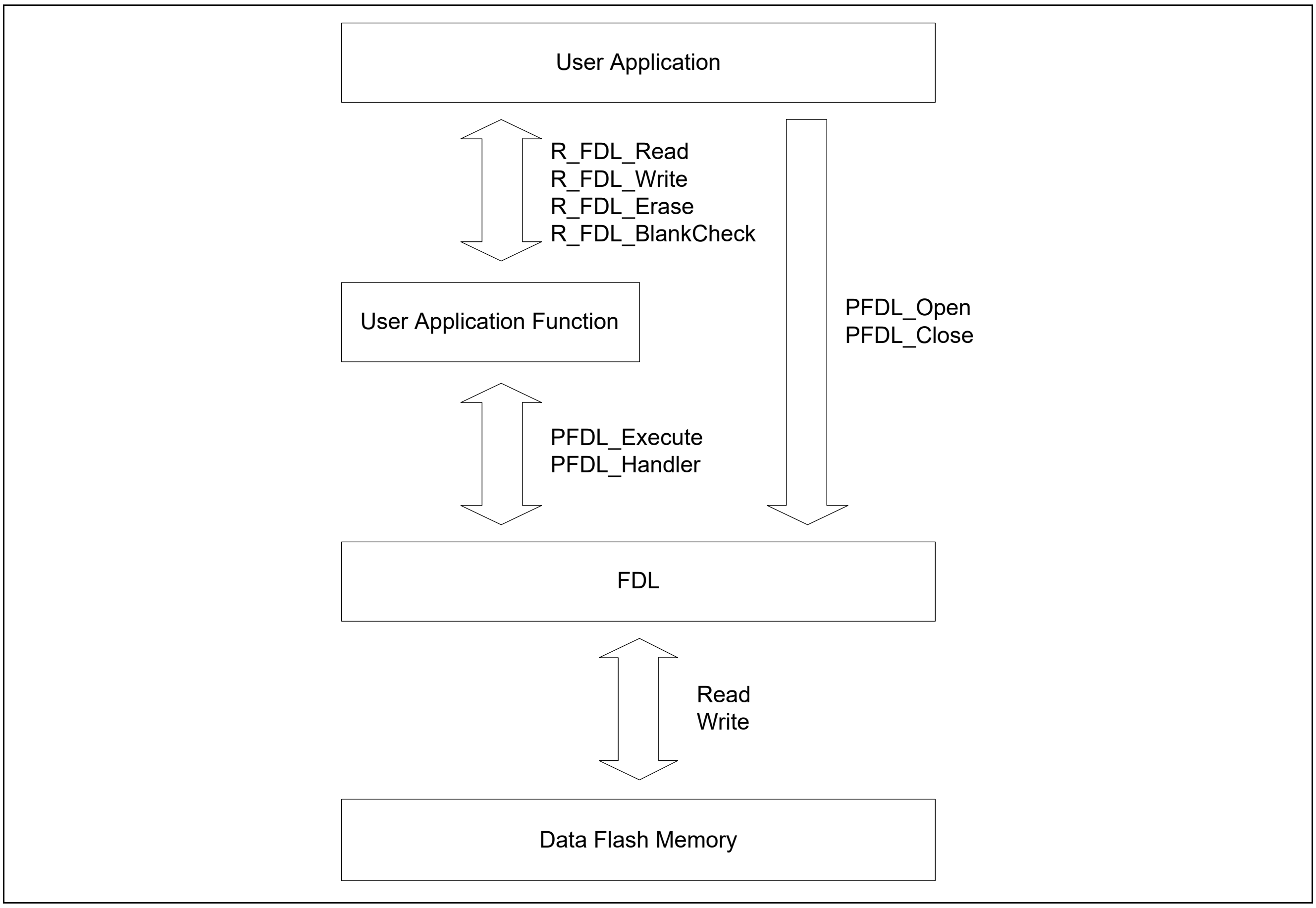


Figure 2.2 Overall Picture of Application

The PFDL_Open and PFDL_Close functions can enable or disable the accessing from user application to data flash memory. The FDL functions which can read/write data flash memory are indirectly executed by calling user application functions in the state that accessing data flash memory is permitted.

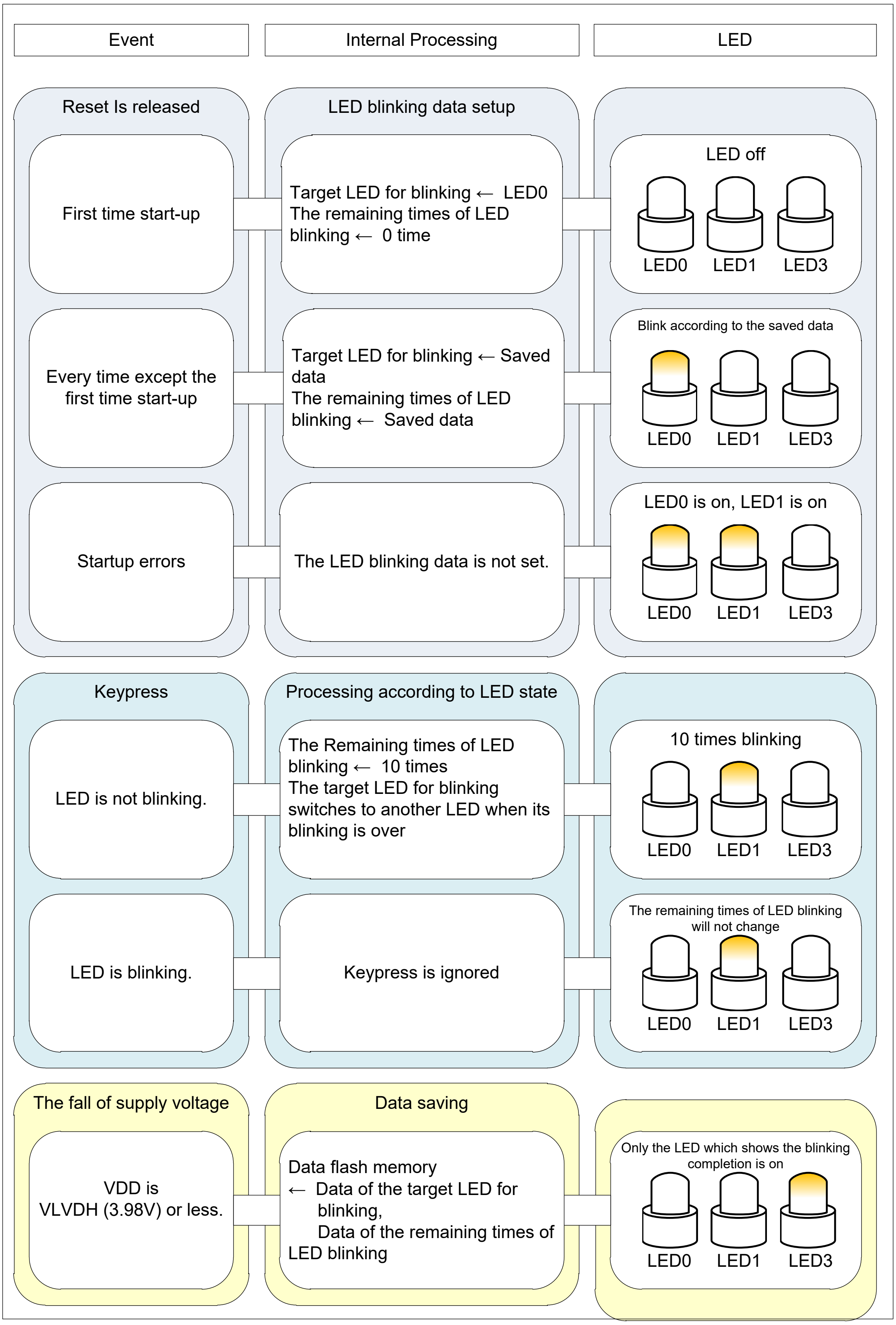


Figure 2.3 Operation Outline

2.1 Shortening of the Write Time of FDL

In order to access data flash memory from user application using FDL, 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. Therefore, FDL realizes the above-mentioned processing by calling a library function PFDL_Open to starting the accessing.

It is necessary to get the address where data to write in this application note, by executing the R_FDL_GetWriteAddr function.

However, executing the function PFDL_Open and obtaining the address by the R_FDL_GetWriteAddr function at a low voltage, it may become power disconnect during 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 processing when it is performed by a two-step processing. The saving phase takes 187.4[µs] in the case of batch processing, and takes 127.6[µs] in the case of two-step processing.

Remark. The measurements described in this application note are the actual measured value calculated by using FDL RL78 Type04 Ver1.05 on the integrated development environment CubeSuite+.

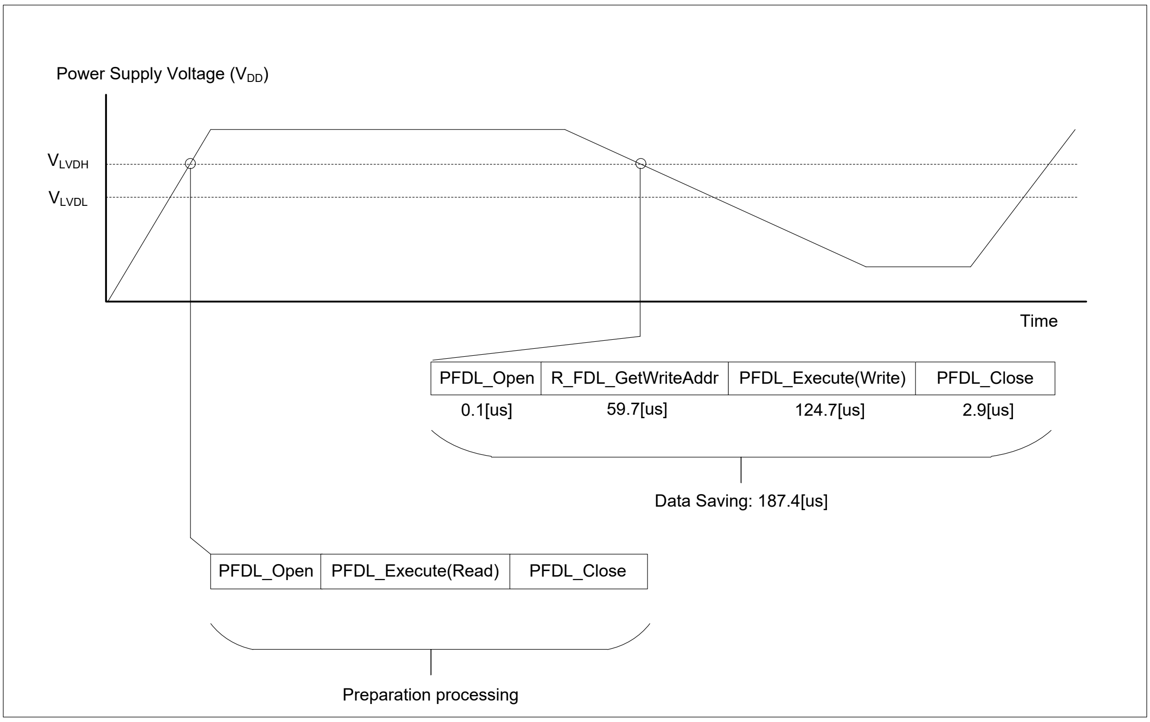


Figure 2.4 Data Saving Processing (by a batch processing)

Remark. FDL functions are used to read data from data flash memory as an example in this application though the reading can be executed by setting the DFLEN bit either.

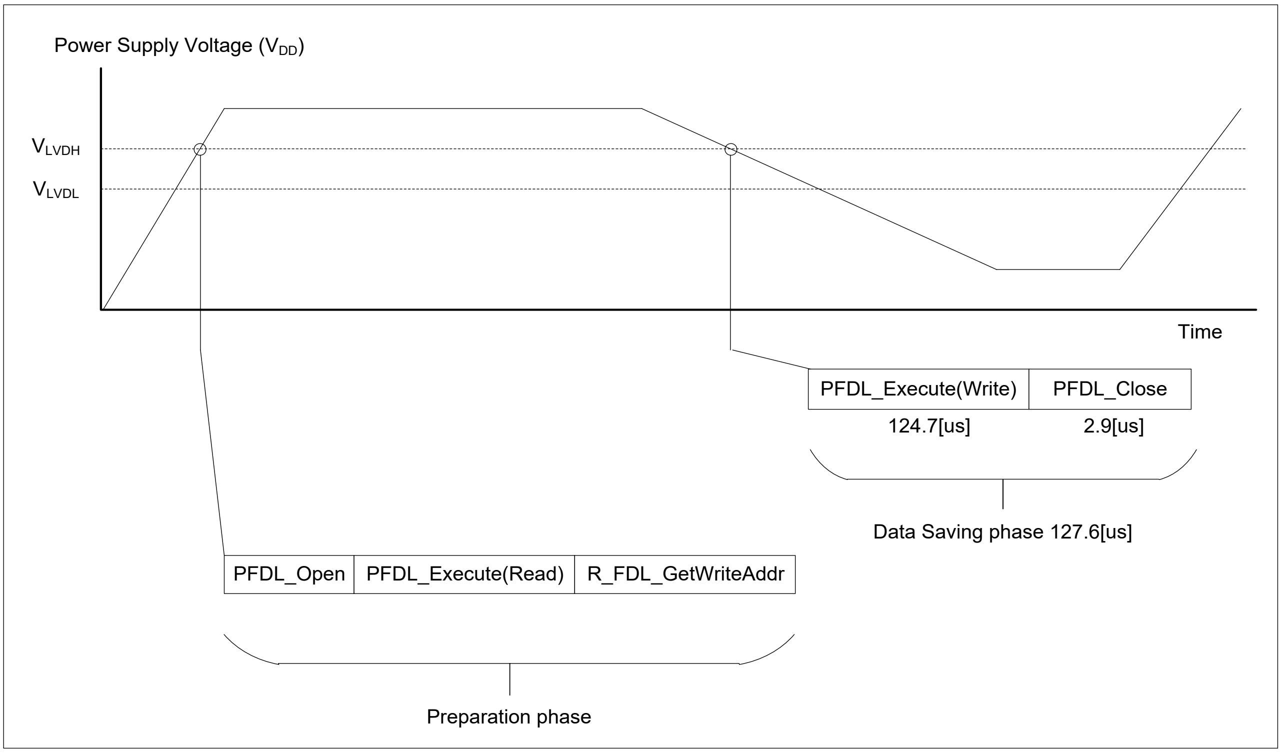


Figure 2.5 Data Saving Processing (by a two-step processing)

2.2 How to Use the Data Flash Memory

In this application note, data flash memory is used as shown in **Figure 2.6**.

Data flash memory is divided into 1-KB blocks. 2 bytes of head of each block is used as a data area for management. The data for management consists of a valid block flag and an invalid block flag. The flags distinguish the state of blocks. Block state is shown in **Table 2.2**.

It means that the block is an “unused” block when both the valid block flag and the invalid block flag are 0xFF, since the read value of blank state (unused state) data flash memory is 0xFF. 0x00 is written to the valid block flag to change the state of this block to "valid" when starting use a block. Write LED blink data in “valid” block and when reach the end of the block, 0x00 would be written into the invalid flag to change the block state to “invalid”. After that, searches “unused” block, and if there are any “unused” blocks, erase an “unused” block to uses it.

At the time of the block erase occurs, preparation processing time is longer for 5.77[ms] in comparison with usual.

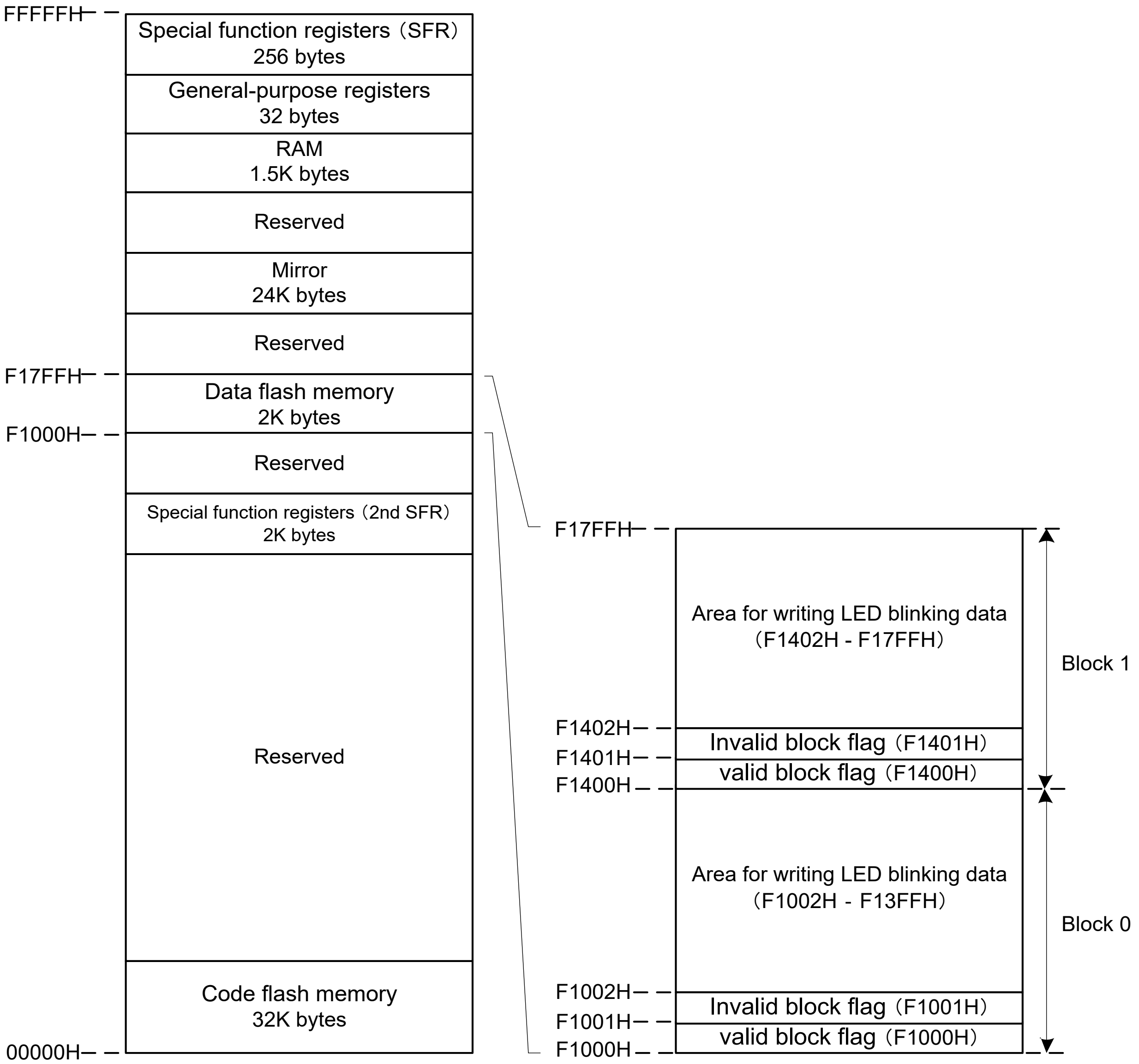


Figure 2.6 Data Flash Memory Block Structure（Example of RL78/L12(R5F10RLC)）

Table 2.2 Block State

	Data area for management	
	Valid block flag (F1000H:Block 0 / F1400H: Block 1)	Invalid block flag (F1001H：Block 0 / F1401H：Block 1)
Unused	FFH	FFH
Valid	00H	FFH
Invalid	00H	00H

2.3 The Algorithm of the Data-addressing in a Block

The algorithm of the data-addressing in this application note is explained.

It starts operating the blank check in units of the data size (2 bytes in this application note) from the end of the area for writing LED blinking data, since the data is stored from the head of the area for LED blinking data. The latest data is stored in an address which is recognized as the first non-blank address (where can't be written with data) by the blank check processing.

If the address where the latest data is written is not the end of block, it is easy to decide the address where data should be written since the next address to the latest data's address is blank (available for data writing). The address for writing data should be set to the leading address of the LED blinking data writing area because a block change would occur if the address to which the newest data is written is the end of block.

Caution: Please execute the blank check processing to judge whether or not data can be written to the area in data flash memory. It does not mean that the memory is in the unused state though the read value of the address is FFH. It is necessary to execute the blank check processing to confirm whether the memory can be written with data or not, since overwriting could damage the data flash memory.

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/L12(R5F10RLC)
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 : Interruption & 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 ・FDL	 CubeSuite+ V2.01.00 from Renesas Electronics Corp. CA78K0R V1.70 from Renesas Electronics Corp. FDLRL78 Type04 Ver1.05 NOTE
GNU Ver. development environment Integrated development environment C compiler ・FDL	 e2studio V2.2.0.13 from Renesas Electronics Corp. KPIT GNURL78-ELF Toolchain V13.02 from Renesas Electronics Corp. FDLRL78 T04E V1.20 NOTE
Board to be used	Renesas Starter Kit for RL78/L12 CPU Board (R0K5010RLC000BR)

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.

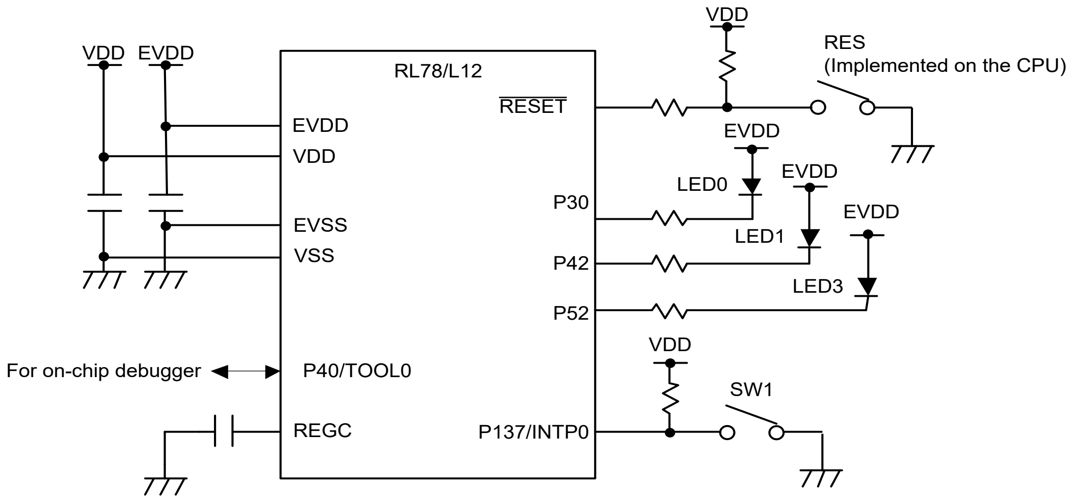
RL78 Family Data Flash Library Type04 (R01AN0608EJ) User Manual

Data Flash Access Library (Type T04 (Pico), European Release) (R01US0055ED0110) 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 VDD or VSS via a resistor).
2. VDD must be held at not lower than the reset release voltage (VLVDH) 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
P30	Output	LED On (LED0) control port
P42	Output	LED On (LED1) control port
P52	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 FDL, LED3 which shows the completion of data saving will be lit up, and then the mode moves to the STOP mode. 0x00 is added to data as a termination symbol.

Moreover, if an error occurs when accessing data flash memory with FDL functions, 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 P30, P42, and P52 as the output ports. (LED0, LED1, and LED3 are off.)
   - Switch input: Configure P137/INTP0 for detecting INTP0 falling edges. (Interrupt servicing disabled)
2. Starts the initialization of RAM which is used by FDL.
   Specifically, PFDL_Open function is called.
3. Searches valid blocks in the data flash memory.
   - The valid block is a block whose management area's read values (2 bytes of head) of each block are 00H and FFH. PFDL_Execute (Read) function is used for reading of data.
   - If there is no valid block, erase a head block of the data flash memory and treats it as a valid block. Uses PFDL_Execute(Erase) function in order to erase block.
4. Reads the lateest LED blinking data to blink the target LED for blinking at intervals of 500 ms according to the read data.
   - Performs the blank check every 2 bytes from the end of a valid block, and writes the latest data in the even addresses of the memory address which is recognized as that is the first non-blank address. PFDL_Execute(Blankcheck) function is used for the blank check and PFDL_Execute (Read) function is used for reading data.
   - 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.
   - 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.
   - Blinking according to the read data is started.
5. Acquires the address for saving data when the voltage is getting low.

・Acquires the address of the next data writing through the read address. Specifically, calls the R_FDL_GetWriteAddr function.

・When the block numbers of the read address and the write address are different, block change occurs.

・After obtaining address for writing, it becomes the keypress waiting state.

6. 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.

7. 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.
PFDL_Execute(Write), PFDL_Close

8. If an error occurs when accessing data flash memory by the FDL, it will go to the stop mode after stopping the FDL and turning on both LED0 and LED1.
Specifically, after the following function is called, STOP command is executed.
PFDL_Close

9. If reset occurs, it will return to1.

Figure 6.1 shows the timing chart.

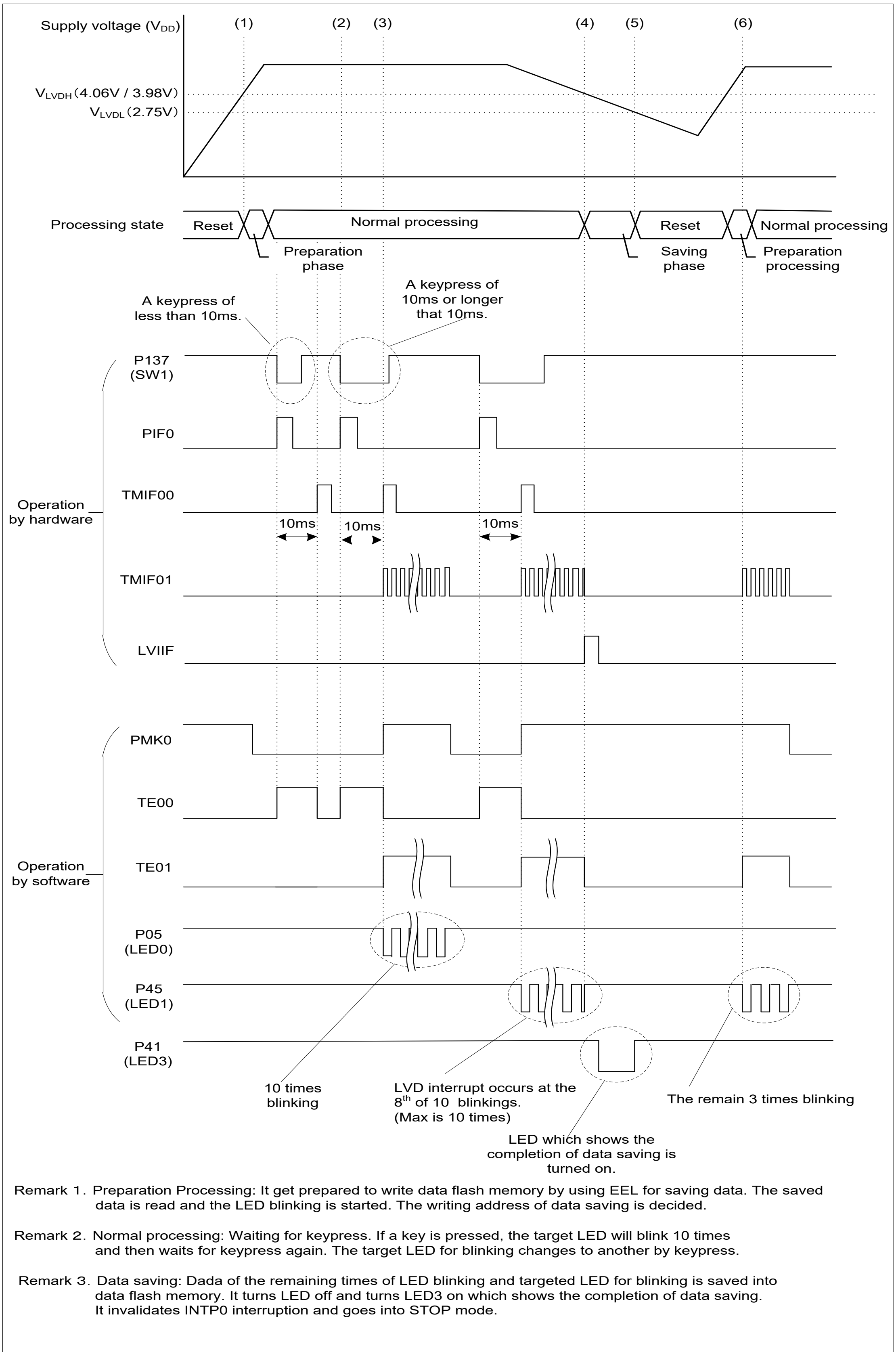


Figure 6.1 Timing Chart

(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. The address for writing the data which should be saved at a low voltage will be obtained to prepare for that time.

(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_fdl_function.c	Source file for the data saving function	Additional functions: R_FDL_Read R_FDL_Write R_FDL_Erase R_FDL_BlankCheck R_FDL_EnableBlock R_FDL_DisableBlock R_FDL_ChangeBlock R_FDL_ReadManageData R_FDL_ReadLedData R_FDL_SearchEnableBlock R_FDL_SearchReadAddr R_FDL_CheckDataRange R_FDL_GetWriteAddr R_FDL_WriteLedData
r_fdl_function.h	Header file for the data saving function	-
pfdl.h [Note 1]	Header file of FDL	Same in both complier
pfdl_types.h [Note 1]	Header file of FDL type definition	Same in both complier
pfdl.lib [Note 1]	FDL	CubeSuite+ version
pfdl.a [Note 1]	FDL	GNU version
r_fdl.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 FDL"** 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 and **Table 6.4** list the constants for the sample program.

Table.6.3 Constants (1/2)

Constant	Setting	Description
BLOCK_NUM	0x0002	The number of the data flash memory blocks
RET_OK	0x00	Normal response
RET_NG	0x01	Abnormal response
RET_NG_DEVICE	0x02	FDL access error
RET_BLOCK_UNUSED	0x03	Unused block
RET_BLOCK_ENABLE	0x04	Valid block
RET_BLOCK_DISABLE	0x05	Invalid block
RET_BLOCK_UNEXPECTED	0x06	Abnormal block
RET_CHECK_BLANK	0x07	Blank state
RET_CHECK_FILL	0x08	Filled with data
STA_ADDR_SEARCH	0x10	Status of searching data storage address
STA_ADDR_FOUND	0x11	Status of data storage address found
STA_ADDR_NOTFOUND	0x12	Status of no available data storage address
STA_ACCESS_ERROR	0x13	FDL access error
SHIFT_NUM	0x04	The number of bits to be shifted for LED blinking data
BLINK_LED_MAX	0x01	The maximum of the blinking LED's number
BLINK_NUM_MAX	0x0A	The maximum of LED blinking times
BLOCK_SIZE	0x0400	The size of 1 block
ENABLE_AREA_SIZE	0x0001	The size of valid block flag
DISABLE_AREA_SIZE	0x0001	The size of invalid block flag
MANAGE_AREA_SIZE	0x0002	The size of management area
BLANK_DATA	0xFF	Blank state data
ENABLE_DATA	0x00	Valid block state data
DISABLE_DATA	0x00	Invalid block state data
LED_DATA_SIZE	0x02	The size of LED blinking data
TERMINAL_SYMBOL	0x00	Termination symbol of LED blinking data
ENABLE_AREA_ADDR	0x0000	Address of valid block confirmation area
DISABLE_AREA_ADDR	0x0001	Address of invalid block confirmation area
FIRST_DATA_ADDR	0x0002	The head address of LED blinking data storage area
LAST_DATA_ADDR	0x03FE	The last address of LED blinking data storage area

Table 6.4 Constants (2/2)

Constant	Setting	Description
BLINK_LED0	0x00	Target LED for blinking: LED0
BLINK_LED1	0x01	Target LED for blinking: LED1
LED0	P3.0	LED0 control port (CubeSuite+ version)
	P3_bit.no0	LED0 control port (GNU version)
LED1	P4.2	LED1 control port (CubeSuite+ version)
	P4_bit.no2	LED1 control port (GNU version)
LED3	P5.2	LED3 control port (CubeSuite+ version)
	P5_bit.no2	LED3 control port (GNU 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 (GNU 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 6.5 lists the global variables.

Table 6.5 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 6.6 gives a list of functions that are used by this sample program.

Table 6.6 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_FDL_SearchEnableBlock	Valid block search
R_FDL_SearchReadAddr	Read address search
R_FDL_EnableBlock	Block validation
R_FDL_ReadLedData	Read LED blinking data
R_FDL_CheckDataRange	Valid range check of LED blinking data
R_TAU0_Channel1_Start	Enabling TAU01
r_tau0_channel1_interrupt	TAU01 interrupt handler
R_TAU0_Channel1_Stop	Disabling TAU01
R_INTC0_Start	Enabling INTP interruption
r_intc0_interrupt	INTP0 interrupt handler
R_TAU0_Channel0_Start	Enabling TAU00
r_tau0_channel0_interrupt	TAU00 interrupt handler
R_FDL_GetWriteAddr	Obtaining write address
R_FDL_WriteLedData	Write LED blinking data
R_INTC0_Stop	Disabling INTP0
R_TAU0_Channel0_Stop	Disabling TAU00
R_LVD_InterruptMode_Start	Enabling LVD interruption
r_lvd_interrupt	LVD interrupt handler
R_FDL_ReadManageData	Read management data
R_FDL_Read	Read by FDL
R_FDL_BlankCheck	Blank check of FDL
R_FDL_Write	Write by FDL
R_FDL_ChangeBlock	Block change
R_FDL_DisableBlock	Block invalidation
R_FDL_Erase	Erase of block
PFDL_Open	Start of FDL
PFDL_Close	Stop FDL
PFDL_Execute	Control of the data flash memory
PFDL_Handler	Confirmation of control state of data flash memory and set-up of continuation run (Status process)

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_FDL_SearchEnableBlock

Synopsis	Valid block search	
Header	r_fdl_function.h	
Declaration	uint8_t R_FDL_SearchEnableBlock(uint16_t* pblock)	
Explanation	Obtains the number of enabled block.	
Arguments	uint16_t* pblock	The pointer of valid block number.
Return value	・Valid block is found: RET_OK ・No Valid block: RET_NG ・FDL access error: RET_NG_DEVICE	

R_FDL_SearchReadAddr

Synopsis	Read address search	
Header	r_fdl_function.h	
Declaration	uint8_t R_FDL_SearchReadAddr(uint16_t block, uint16_t* paddr)	
Explanation	Searches address where data is stored in a specified block. If data is stored, obtains the address. If no data is stored (The whole block is blank.), obtains the head address of data area.	
Arguments	uint16_t block	Valid block number
	uint16_t* paddr	The pointer to the data storage address.
Return value	・An address is obtained: RET_OK ・FDL access error: RET_NG_DEVICE	

R_FDL_EnableBlock

Synopsis	Block validation	
Header	r_fdl_function.h	
Declaration	uint8_t R_FDL_EnableBlock(uint16_t block)	
Explanation	Enables a specified unused block.	
Arguments	uint16_t block	The number of valid block.
Return value	・The block is valid: RET_OK ・FDL access error: RET_NG_DEVICE	

R_FDL_ReadLedData

Synopsis	Read LED blinking data	
Header	r_fdl_function.h	
Declaration	uint8_t R_FDL_ReadLedData(uint16_t addr, uint8_t* pdata)	
Explanation	Reads LED blinking data from specified address.	
Arguments	uint16_t addr	Read address
	uint8_t* pdata	The pointer to a buffer to store read data.
Return value	・Read-out is successful: RET_OK ・FDL access error: RET_NG_DEVICE	

R_FDL_CheckDataRange

Synopsis	Valid range check of LED blinking data	
Header	r_fdl_function.h	
Declaration	uint8_t R_FDL_CheckDataRange(uint8_t* pdata)	
Explanation	Checks whether the LEDblinking blink data is valid or not.	
Arguments	uint8_t* pdata	The pointer to LED blinking data.
Return value	・ Within the range: RET_OK ・ Outside of the range: RET_NG	

R_TAU0_Channel1_Start

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

r_tau0_channel1_interrupt

Synopsis	TAU01 interrupt handler
Header	r_cg_timer.h r_fdl_function.h
Declaration	__interrupt static void r_tau0_channel1_interrupt(void)
Explanation	Switches ON/OFF of LED and decrements the total number of times a LED will blink. Target LED for blinking is changed when blink 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 counting 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	Enabling TAU00
Header	r_cg_timer.h r_fdl_function.h
Declaration	__interrupt static void r_tau0_channel0_interrupt(void)
Explanation	Checks SW1 state and starts LED blinking.
Arguments	None
Return value	None

R_FDL_GetWriteAddr

Synopsis	Obtaining write address	
Header	r_fdl_function.h	
Declaration	uint8_t R_FDL_GetWriteAddr(uint16_t* pblock , uint16_t* paddr)	
Explanation	Calculates the write address through the read address which is given to the argument. When block change occurs during calculation, the block number is updated.	
Arguments	uint16_t* pblock	The pointer of read/write blocks number.
	uint16_t* paddr	The pointer of read/write address.
Return value	・Obtaining address is successful: RET_OK ・FDL access error: RET_NG_DEVICE	

R_FDL_WriteLedData

Synopsis	Write LED blinking data	
Header	r_fdl_function.h	
Declaration	uint8_t R_FDL_WriteLedData(uint16_t addr, uint8_t data)	
Explanation	Writes the LED blinking data to the specified address.	
Arguments	uint16_t addr	Writing destination address
	uint8_t data	Data to be written
Return value	・Data writing is successful: RET_OK ・FDL access error: RET_NG_DEVICE	

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_TAU0_Channel0_Stop

Synopsis	Disabling TAU00
Header	r_cg_timer.h
Declaration	void R_TAU0_Channel0_Stop(void)
Explanation	Stops counting TAU00.
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_fdl_function.h
Declaration	__interrupt static void r_lvd_interrupt(void)
Explanation	Sets the low voltage detection flag
Arguments	None
Return value	None

R_FDL_ReadManageData

Synopsis	Read management data	
Header	r_fdl_function.h	
Declaration	uint8_t R_FDL_ReadManageData(uint16_t block)	
Explanation	Reads data in the specified block (2 byte of head) and obtains the state of block.	
Arguments	uint16_t block	Reading block number.
Return value	・Unused block: RET_BLOCK_UNUSED ・Valid block: RET_BLOCK_ENABLE ・Disabled block: RET_BLOCK_DISABLE ・Abnormal block: RET_BLOCK_UNEXPECTED ・FDL access error: RET_NG_DEVICE	

R_FDL_Read

Synopsis	Read by FDL	
Header	r_fdl_function.h	
Declaration	uint8_t R_FDL_Read(uint16_t addr, uint8_t* pdata, uint16_t size)	
Explanation	Reads data from the data flash memory.	
Arguments	uint16_t addr	Head address to be read
	uint8_t* pdata	The pointer to a buffer in which read data is stored
	uint16_t size	Size of reading data.
Return value	・Read-out is successful: RET_OK ・FDL access error: RET_NG_DEVICE	

R_FDL_BlankCheck

Synopsis	Blank check of FDL	
Header	r_fdl_function.h	
Declaration	uint8_t R_FDL_BlankCheck(uint16_t addr, uint16_t size)	
Explanation	Confirms the specified range whether to be in a blank state.	
Arguments	uint16_t addr	The address where check is started
	uint16_t size	The number of bytes to be checked
Return value	・The state filled with data: RET_CHECK_FILL ・Blank state: RET_CHECK_BLANK ・FDL access error: RET_NG_DEVICE	

R_FDL_Write

Synopsis	Write by FDL	
Header	r_fdl_function.h	
Declaration	uint8_t R_FDL_Write(uint16_t addr, uint8_t* pdata, uint16_t size)	
Explanation	Writes data into specified address.	
Arguments	uint16_t addr	Writing destination address
	uint8_t* pdata	Data to be written
	uint16_t size	The number of data to be written
Return value	・Data writing is successful: RET_OK ・FDL access error: RET_NG_DEVICE	

R_FDL_ChangeBlock

Synopsis	Block change	
Header	r_fdl_function.h	
Declaration	uint8_t R_FDL_ChangeBlock(uint16_t* pblock)	
Explanation	The block given by arguments is invalidated and a new block is validated. After the execution of this function, the block number after changing is set to argument pblock.	
Arguments	uint16_t* pblock	[IN] The block number before changing. [OUT] The block number after changing.
Return value	・The change is successful: RET_OK ・The change is failure: RET_NG ・FDL access error: RET_NG_DEVICE	

R_FDL_DisableBlock

Synopsis	Block invalidation	
Header	r_fdl_function.h	
Declaration	uint8_t R_FDL_DisableBlock(uint16_t block)	
Explanation	Invalidate a specified block.	
Arguments	uint16_t block	The block number to be invalidated
Return value	・Invalidation of block is successful: RET_OK ・ FDL access error: RET_NG_DEVICE	

R_FDL_Erase

Synopsis	Erase of block	
Header	r_fdl_function.h	
Declaration	uint8_t R_FDL_Erase(uint16_t block)	
Explanation	Erase specified block.	
Arguments	uint16_t block	The block to be erased
Return value	・Erase is successful: RET_OK ・FDL access error: RET_NG_DEVICE	

PFDL_Open

Synopsis	Start of FDL	
Header	pfdl.h	
Declaration	pfdl_status_t __far PFDL_Open(__near pfdl_descriptor_t* descriptor_pstr)	
Explanation	Initializes RAM which is used by FDL and starts the FDL (This is a FDL library function.)	
Arguments	__near pfdl_descriptor_t* descriptor_pstr	Initial value of FDL.
Return value	・ Normal termination: PFDL_OK	

PFDL_Close

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

PFDL_Execute

Synopsis	Execution of control to the data flash memory
Header	pfdl.h
Declaration	pfdl_status_t __far PFDL_Execute(__near pfdl_request_t* request_pstr)
Explanation	Control data flash memory according to commands. (This is an FDL library function.)
Arguments	__near pfdl_request_t* request_pstr — Requester: Specifies the controlling content of data flash memory. (commands and values)
Return value	・Normal termination: PFDL_OK ・Erasing error: PFDL_ERR_ERASE ・Blank check error: PFDL_ERR_MARGIN or the Internal verify error. ・Writing error: PFDL_ERR_WRITE ・Start executing a specified command: PFDL_BUSY

PFDL_Handler

Synopsis	The check of the control state to data flash memory, and settings of continuous execution. (Status check processing)
Header	pfdl.h
Declaration	pfdl_status_t __far PFDL_Handler(void)
Explanation	Confirms the control state of the command performed immediately before which specified by the PFDL_Execute function. Do the required settings for continuous execution. (This is an FDL library function.)
Arguments	None
Return value	・Normal termination: PFDL_OK ・Erasing error: PFDL_ERR_ERASE ・Blank check error:: PFDL_ERR_MARGIN or the Internal verify error. ・Writing error: PFDL_ERR_WRITE ・Idle state: PFDL_IDLE ・A command is being executed: PFDL_BUSY

6.8 Flowcharts

6.8.1 Overall Flowchart

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

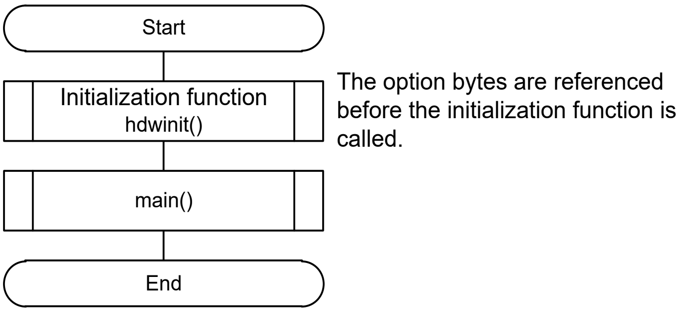


Figure 6.2 Overall Flowchart

6.8.2 Initialization of Peripheral Functions

Figure 6.3 shows the initialization of peripheral function.

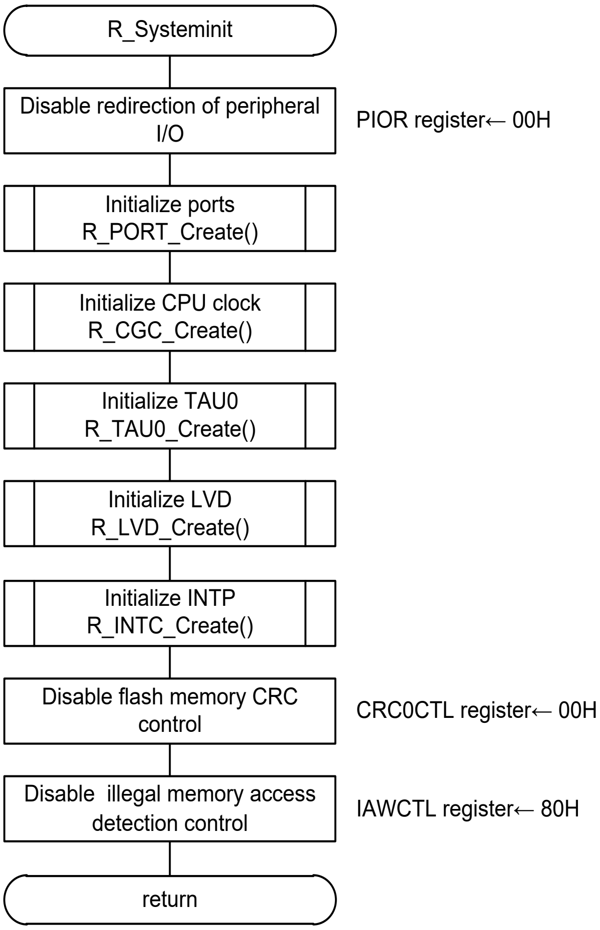


Figure 6.3 Initialization of Peripheral Functions

6.8.3 Initialization of Ports

Figure 6.4 shows the initialization of ports.

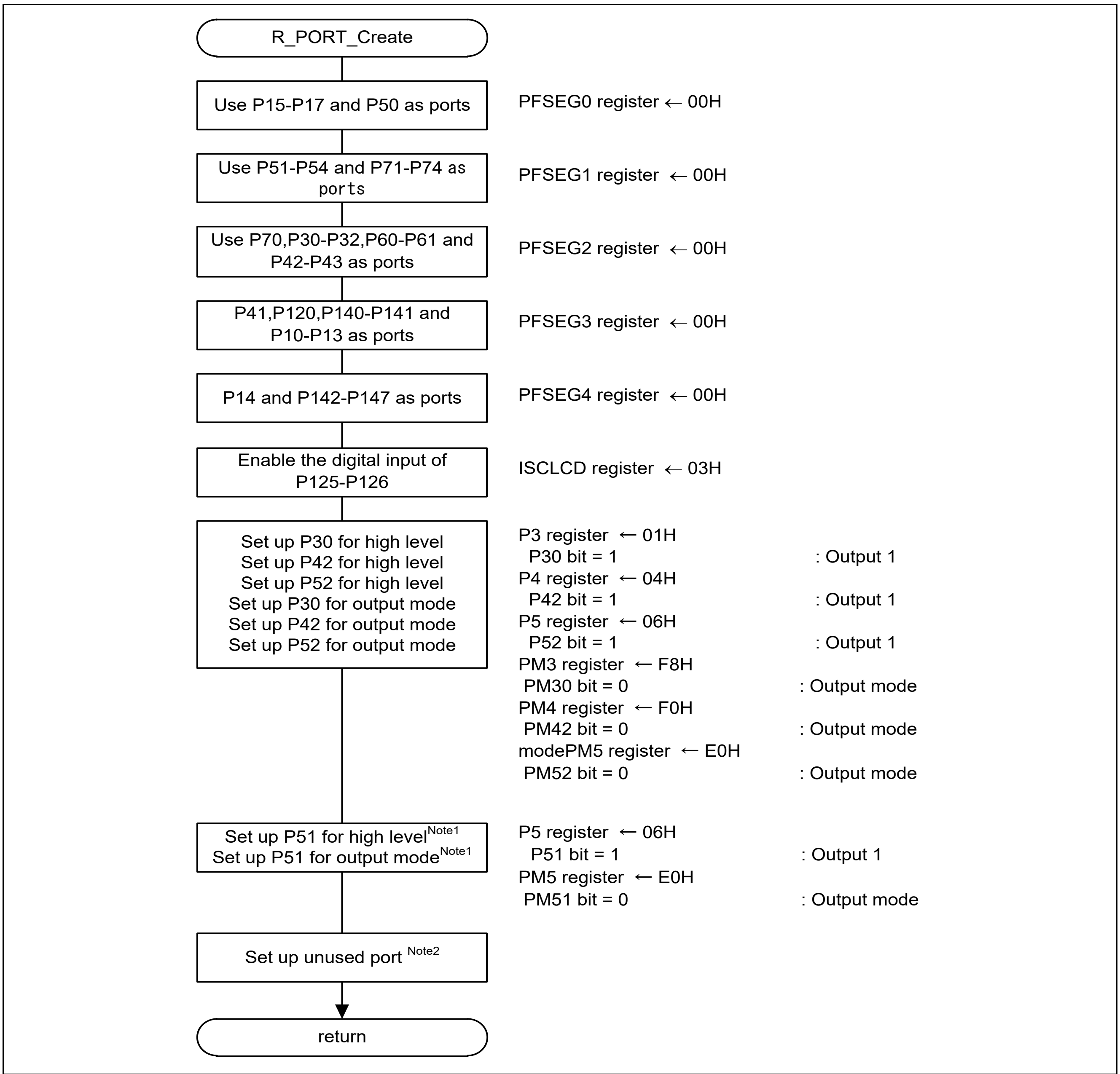


Figure 6.4 Initialization of Ports

Note 1: This is a setup which makes unused LED switch off.

Note 2: Refer to "**RL78/L12 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 V_{DD} or V_{SS} via a separate resistor.

6.8.4 Initialization of CPU Clock

Figure 6.5 shows the initialization of CPU clock.

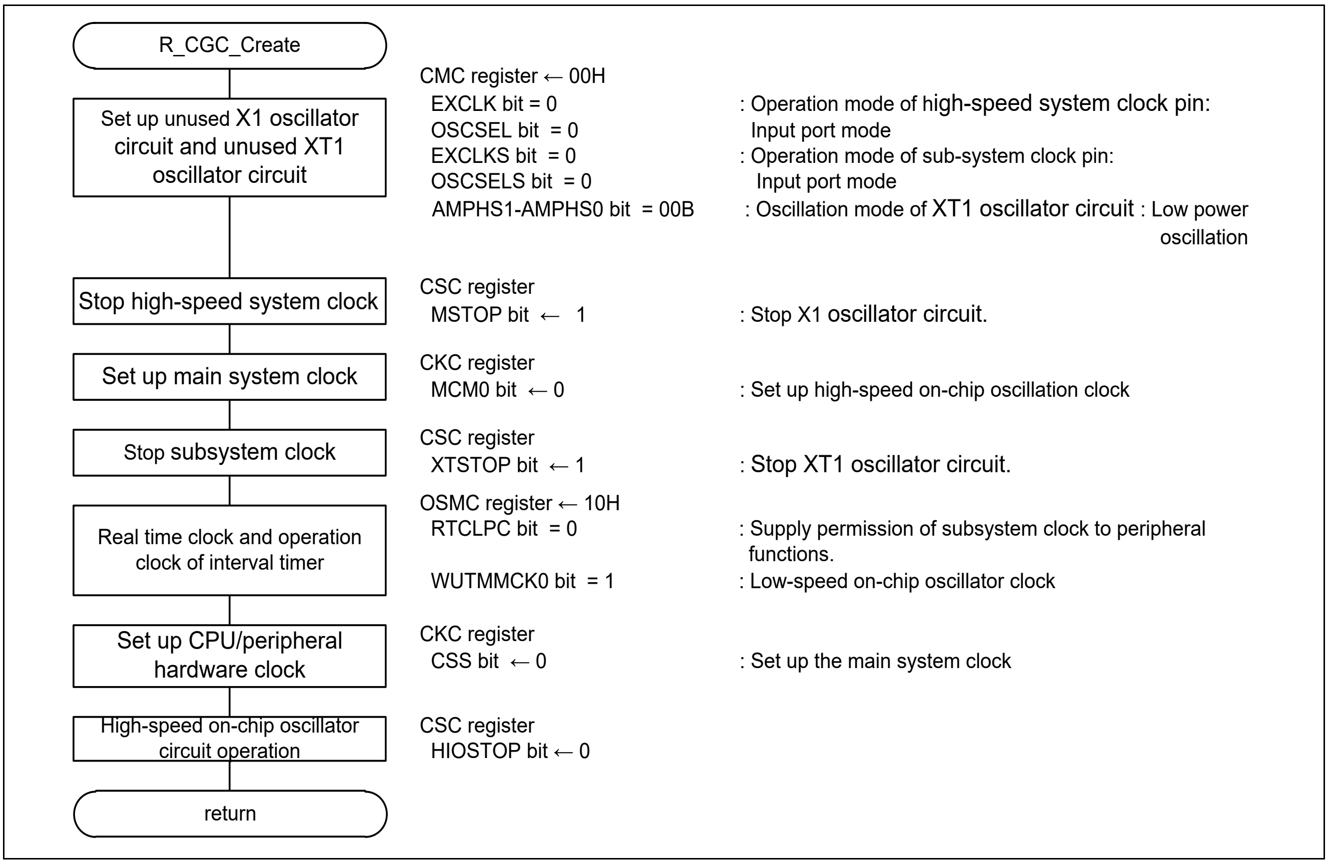


Figure 6.5 Initialization of CPU Clock

6.8.5 Initialization of TAU0

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

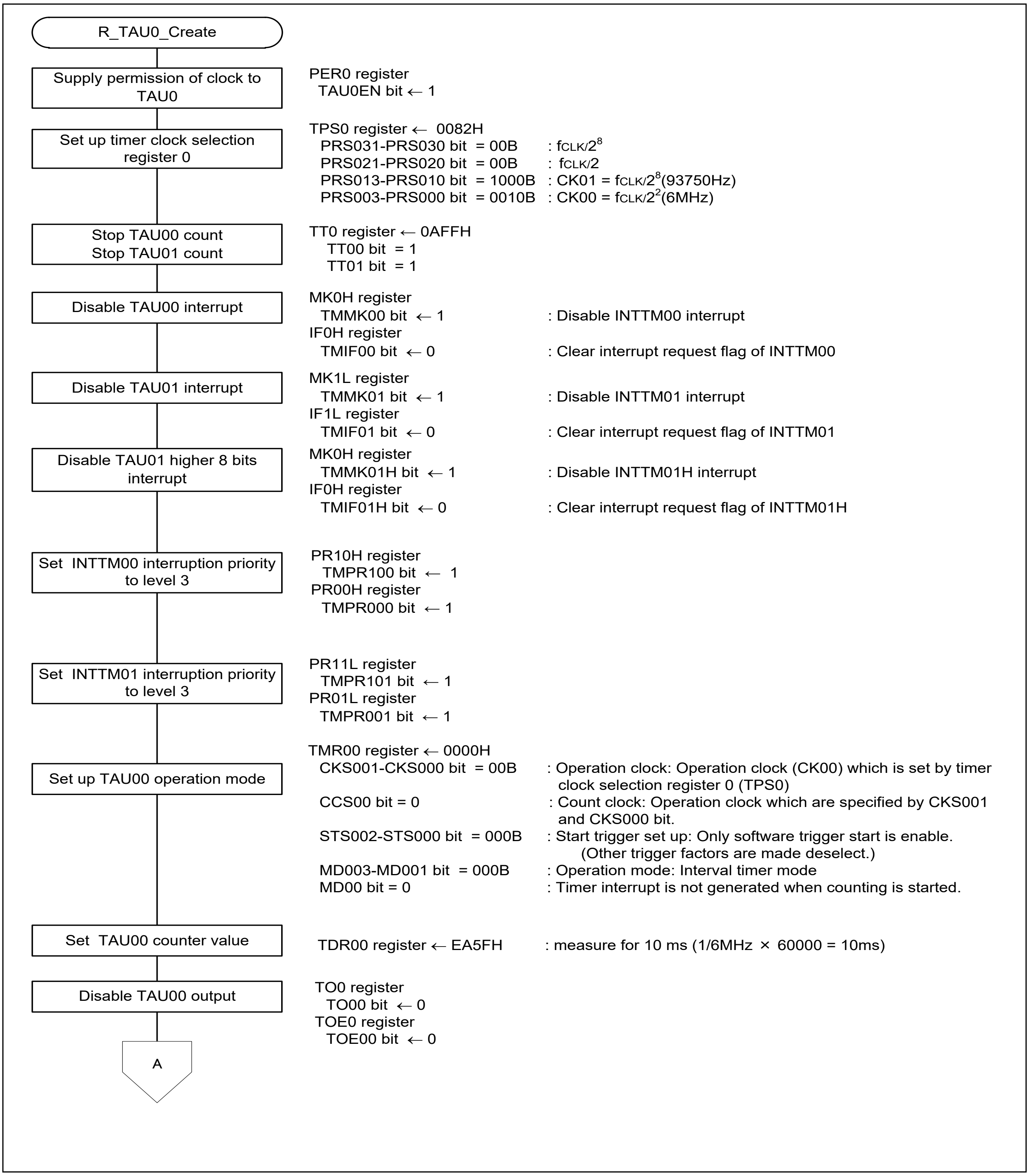


Figure 6.6 Initialization of TAU0 (1/2)

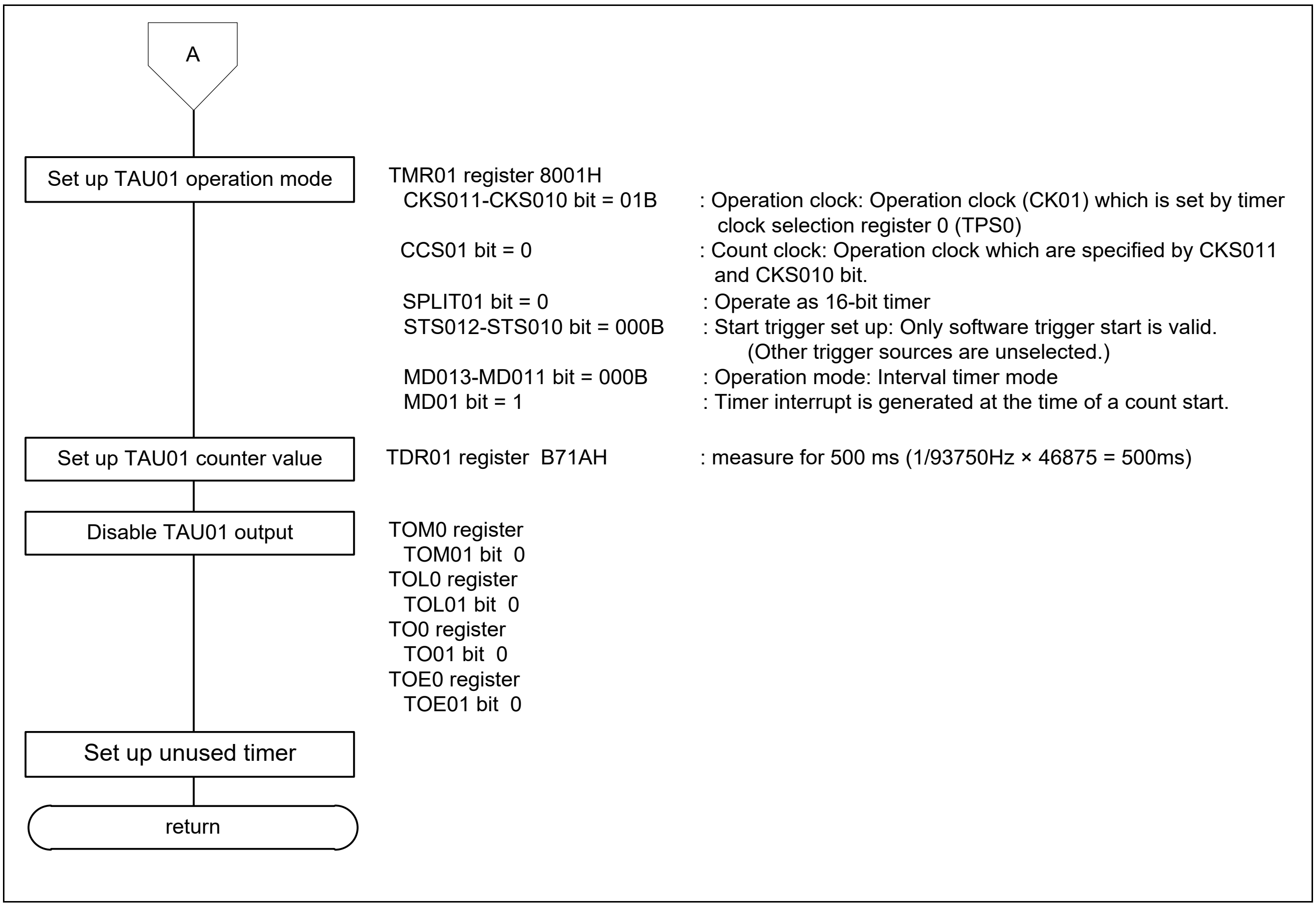


Figure 6.7 Initialization of TAU0 (2/2)

6.8.6 Initialization of INTP

Figure 6.8 shows the initialization of INTP.

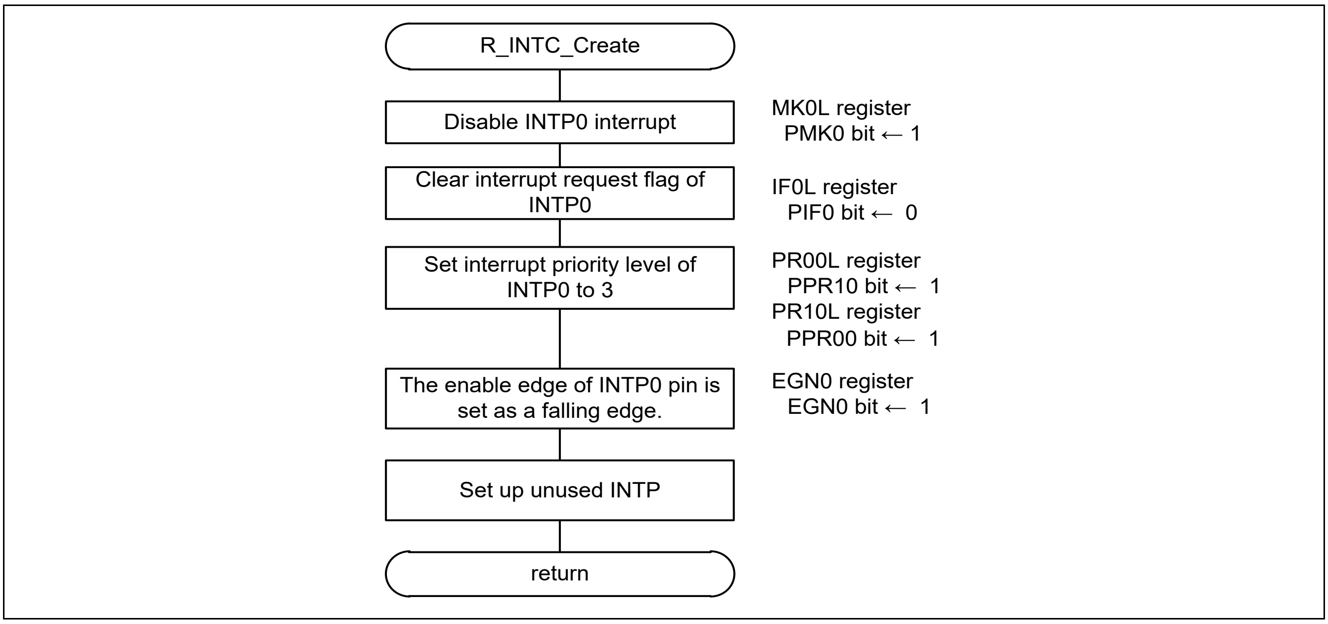


Figure 6.8 Initialization of INTP

6.8.7 Initialization of LVD

Figure 6.9 shows the initialization of LVD.

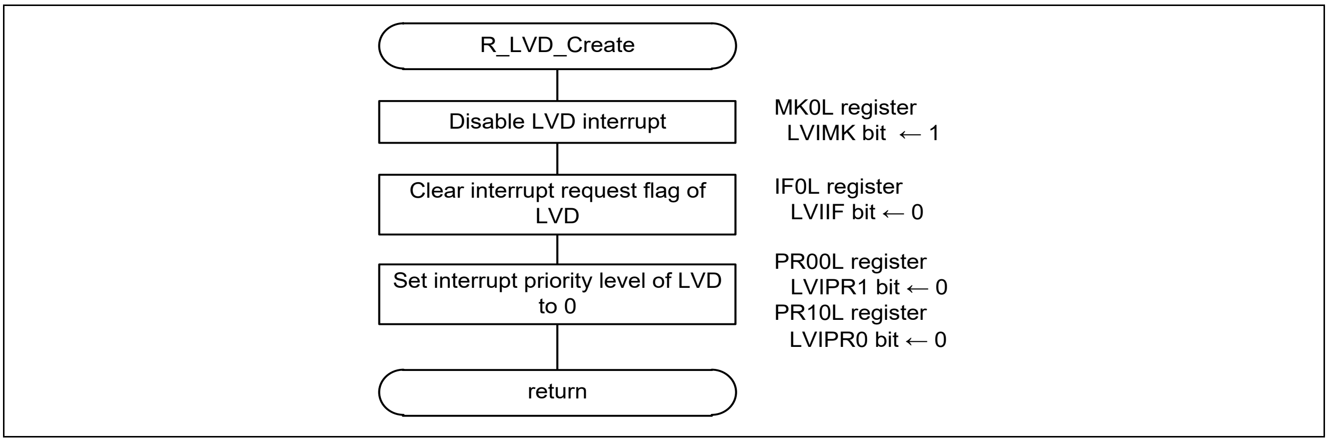


Figure 6.9 Initialization of LVD

6.8.8 Main Processing

Figure 6.10, **Figure 6.11** and **Figure 6.12** show the main processing.

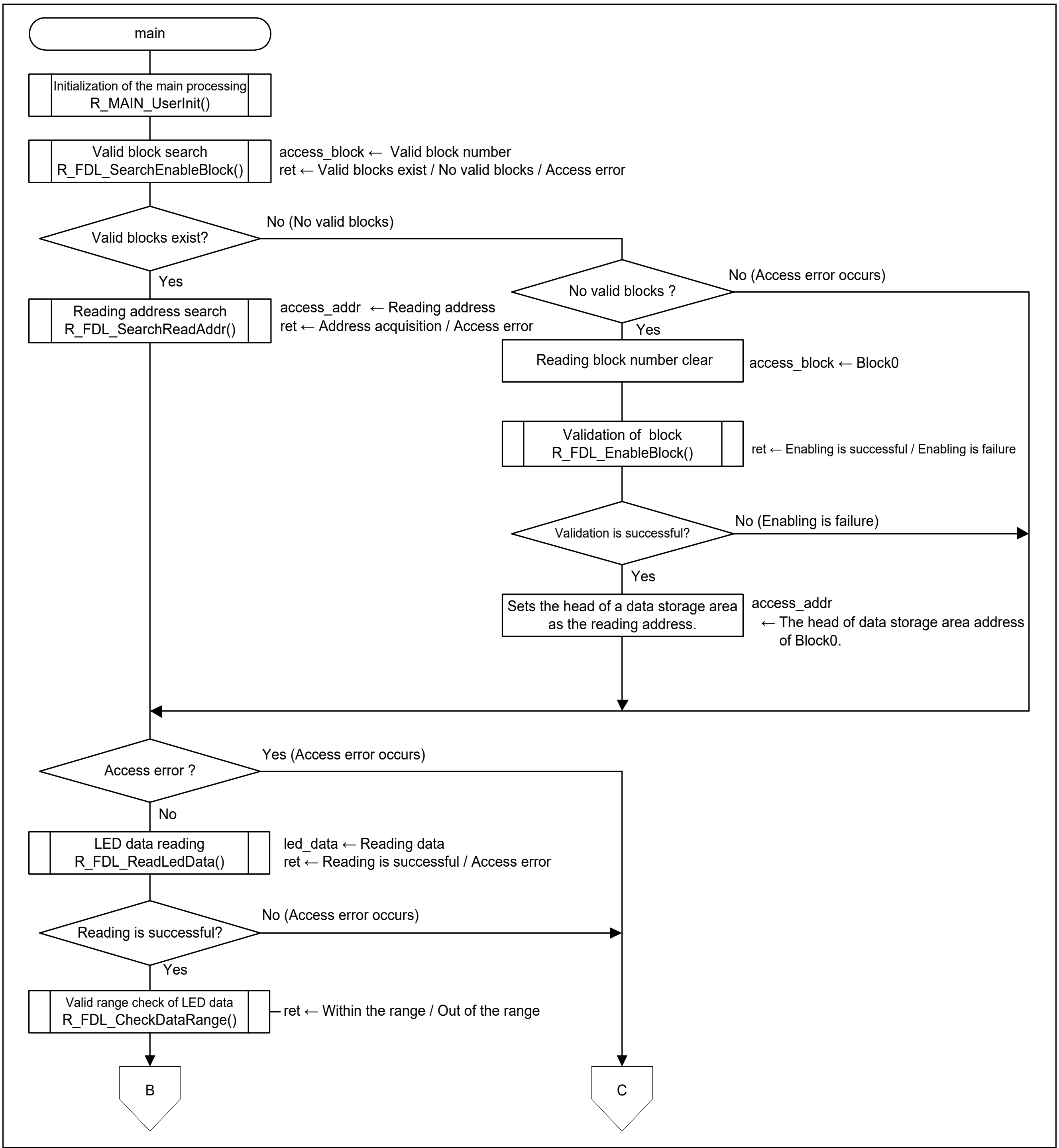


Figure 6.10 Main Processing (1/3)

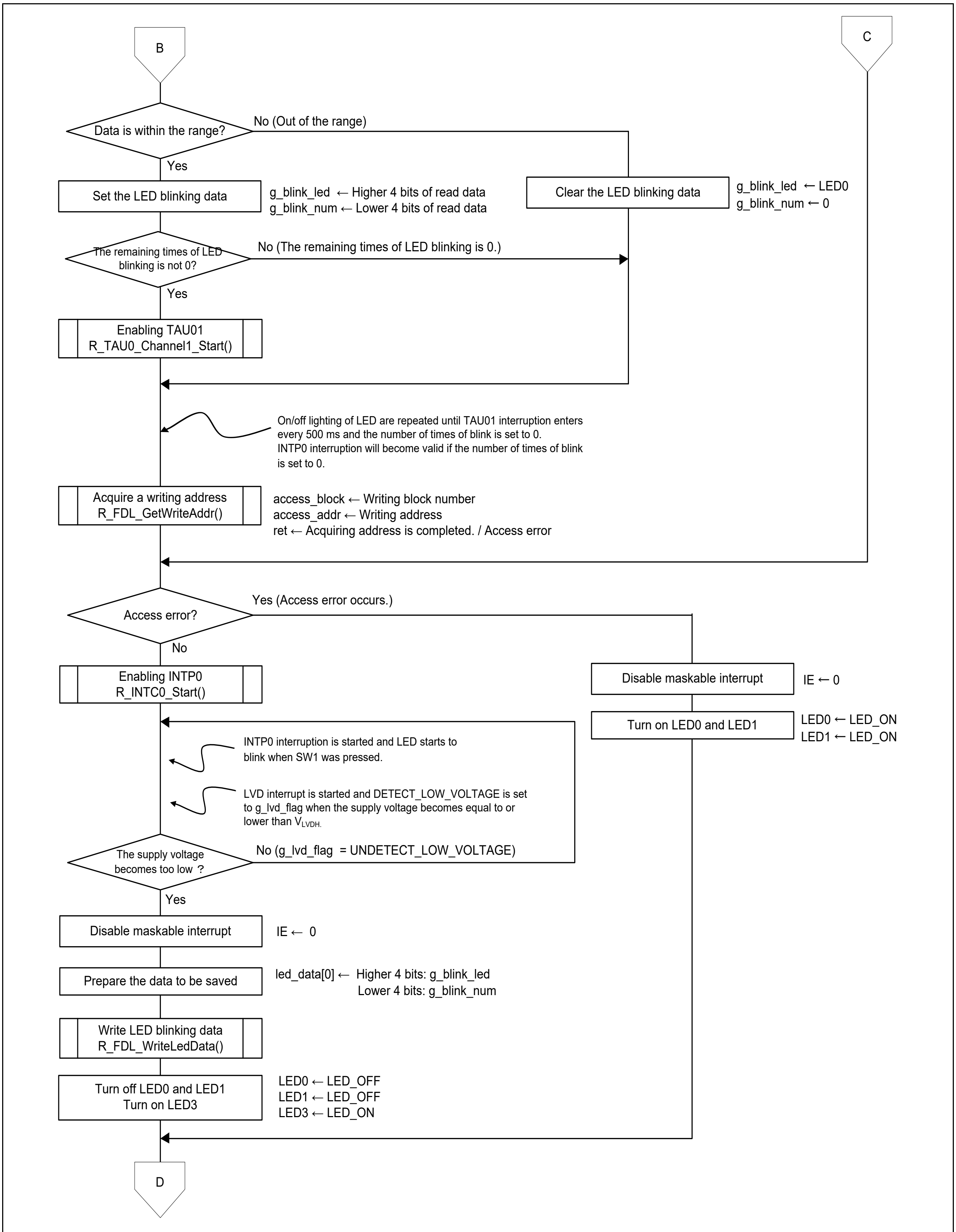


Figure 6.11 Main Processing (2/3)

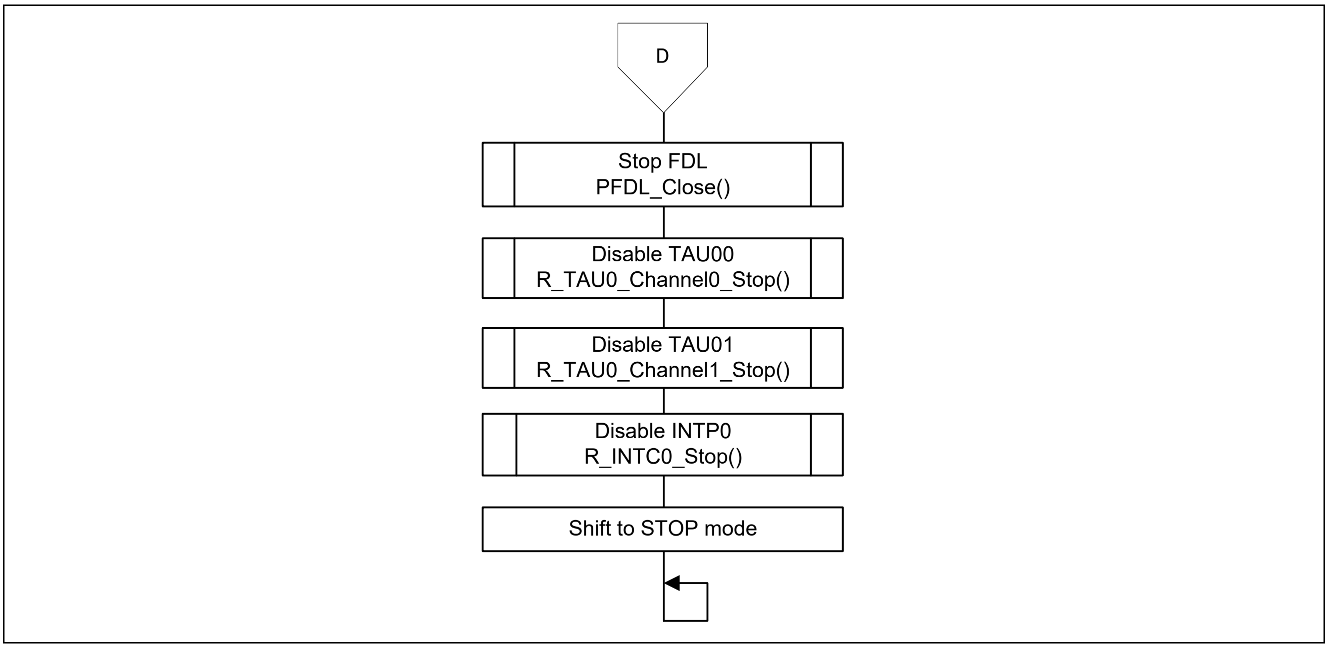


Figure 6.12 Main Processing (3/3)

6.8.9 Initialization of the Main Processing

Figure 6.13 shows the initialization of main processing.

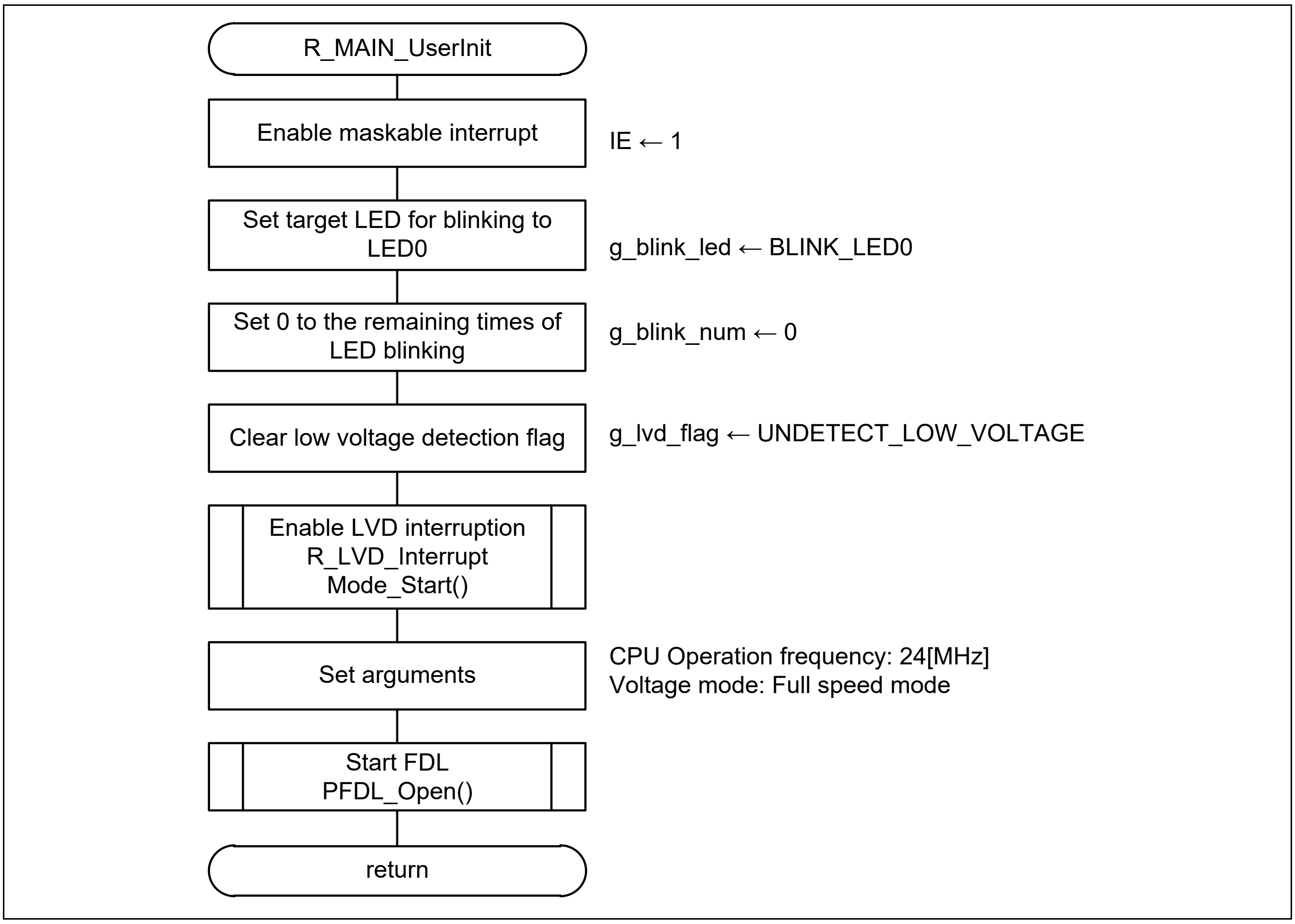


Figure 6.13 Initialization of the Main Processing

6.8.10 Valid Block Search

Figure 6.14 shows the how to search valid blocks.

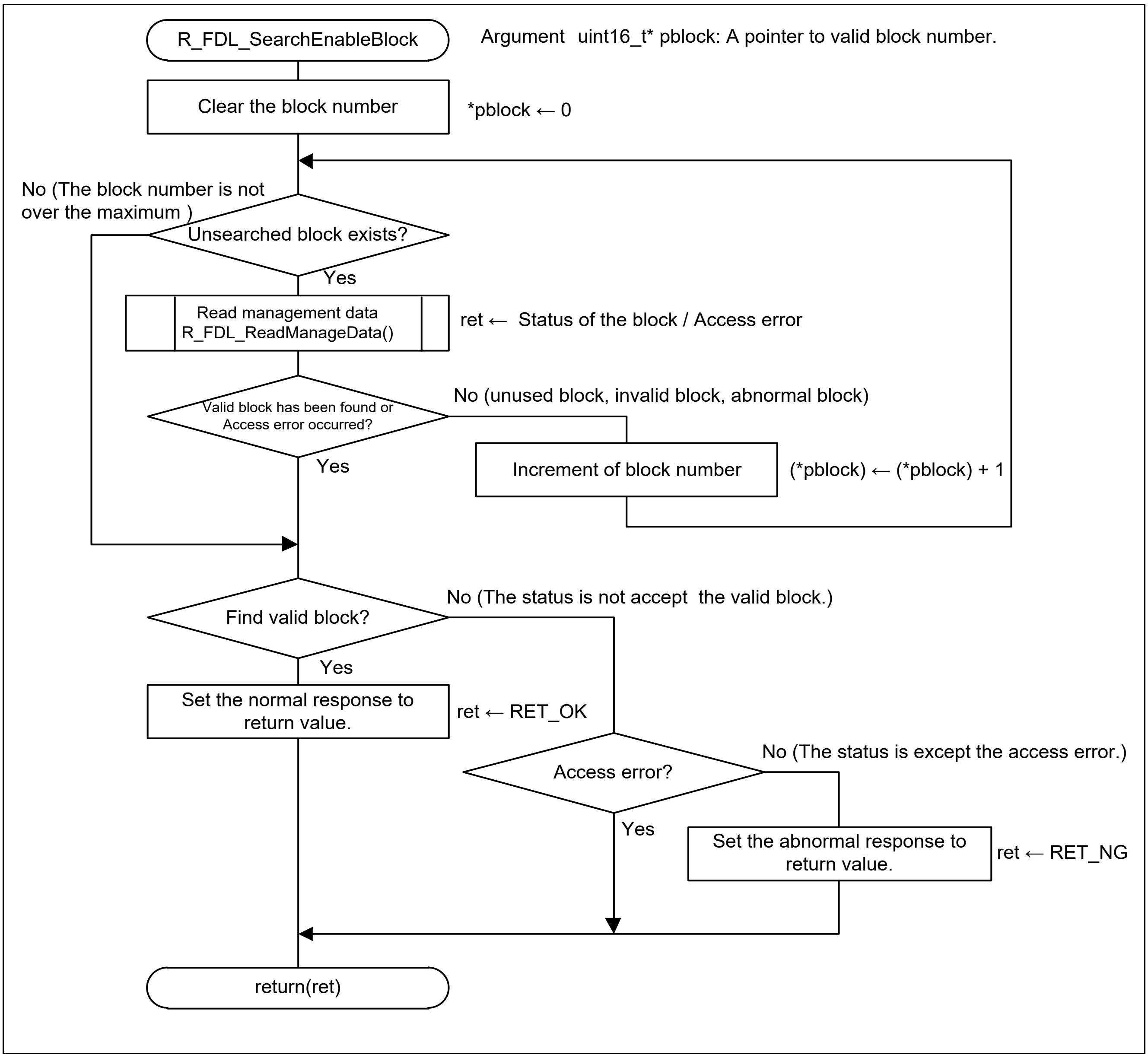


Figure 6.14 Enabled Block Search

6.8.11 Read Address Search

Figure 6.15 shows the how to search the read address.

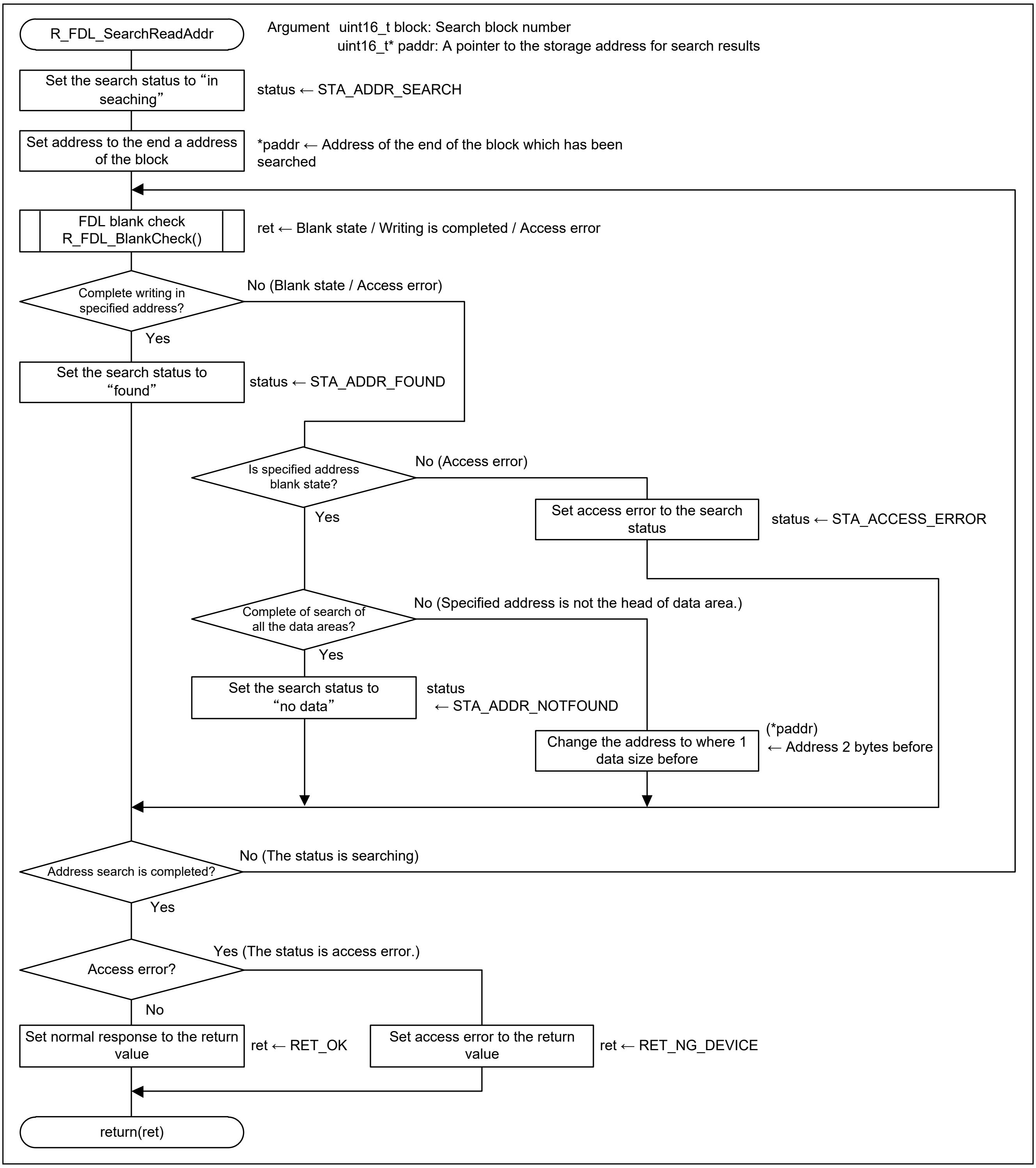


Figure 6.15 Read Address Search

6.8.12 Block validation

Figure 6.16 shows how to validate block.

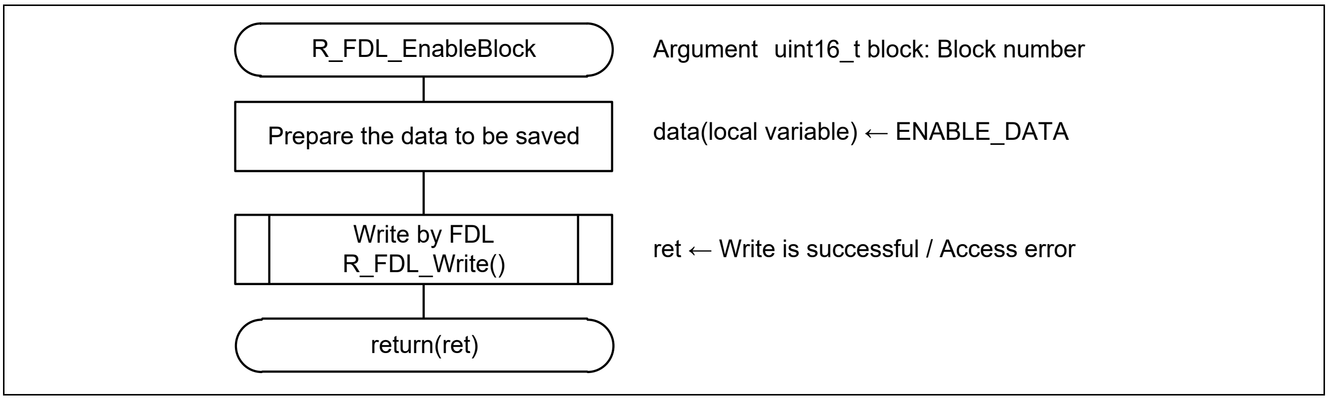


Figure 6.16 Block Enabling

6.8.13 Read LED Blinking Data

Figure 6.17 shows how to read the LED blinking data.

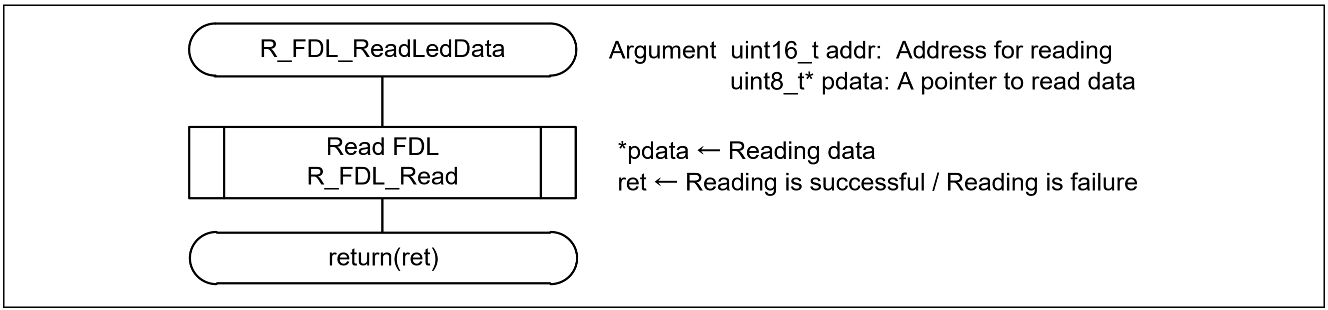


Figure 6.17 Read LED Blinking Data

6.8.14 Valid Range Check of LED Blinking Data

Figure 6.18 shows the valid range check of LED blink data.

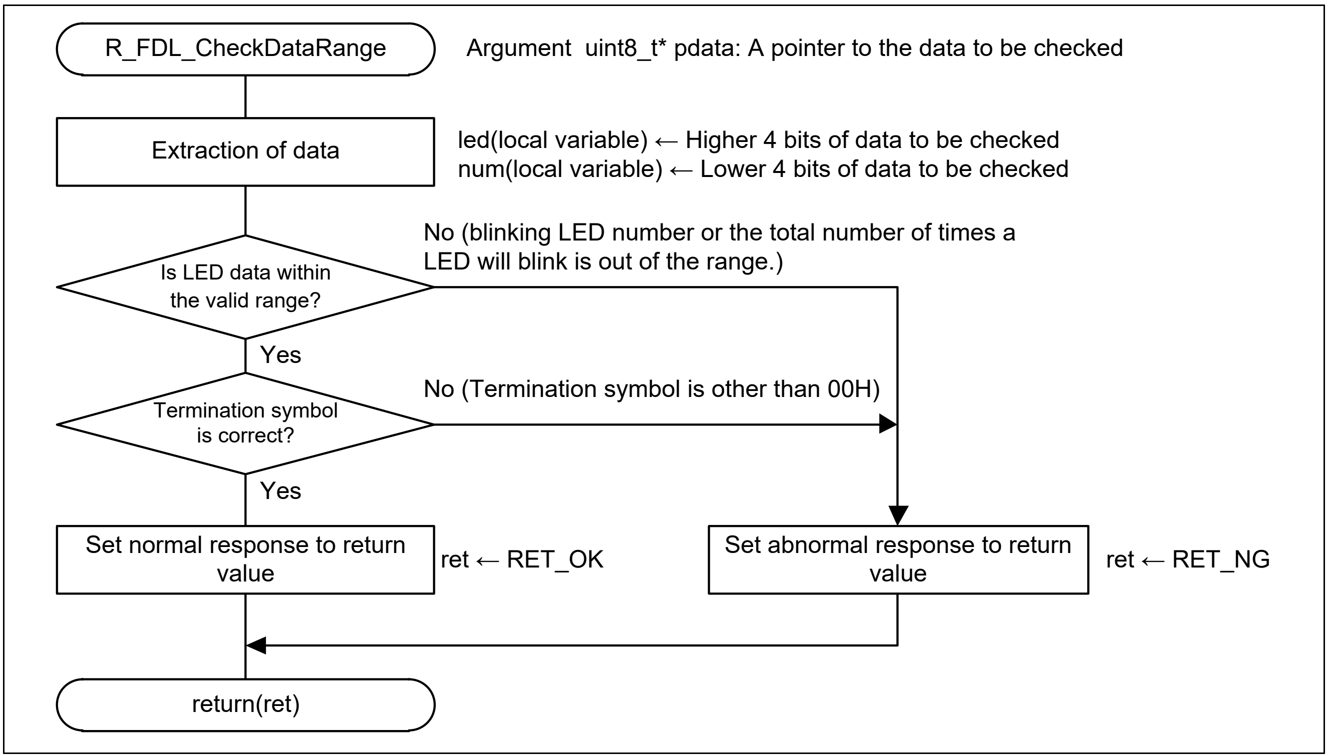


Figure 6.18 Valid Range Check of LED Blinking Data

6.8.15 Enabling TAU01

Figure 6.19 shows how to enable TAU01.

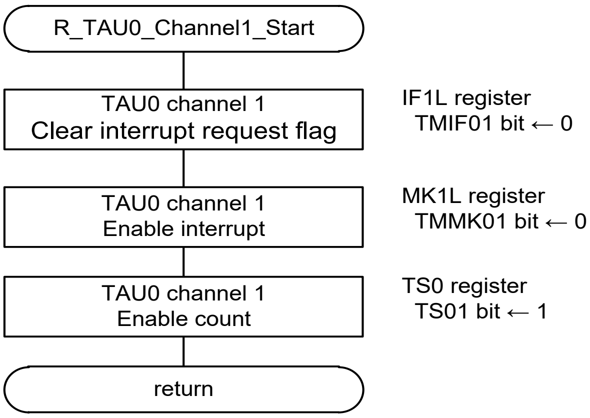


Figure 6.19 Enabling TAU01

6.8.16 TAU01 Interrupt Handler

Figure 6.20 shows the flowchart of the TAU01 interrupt handler.

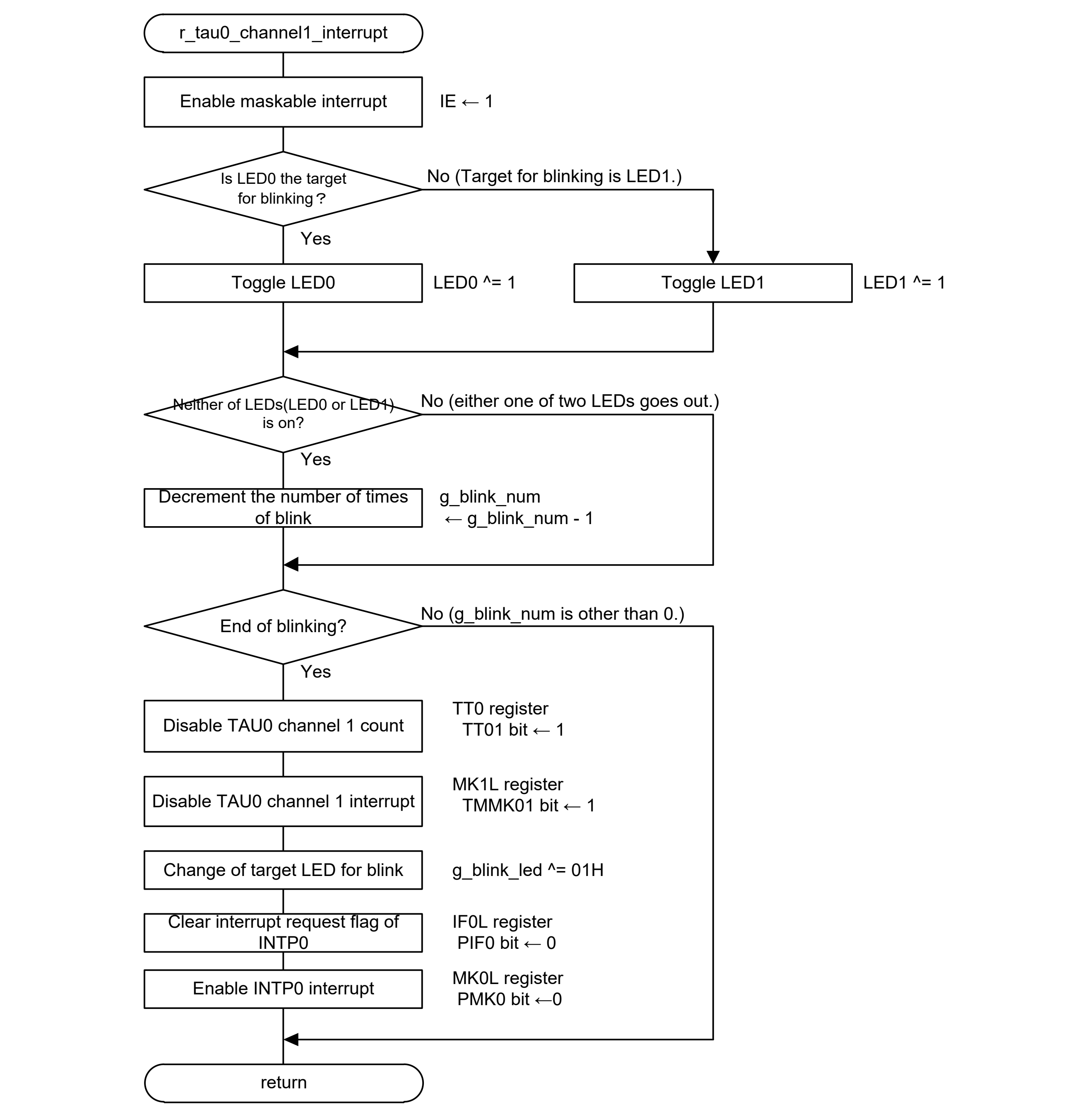


Figure 6.20 Enabling TAU01 Interrupt Handler

6.8.17 Disabling TAU01

Figure 6.21 shows how to disable TAU01.

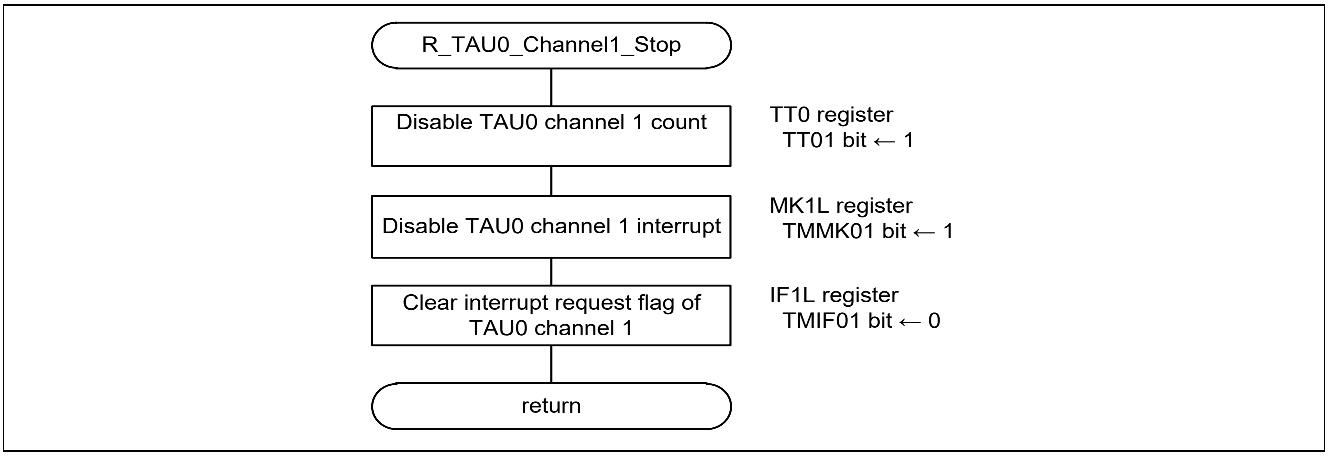


Figure 6.21 Disabling TAU01

6.8.18 Enabling INTP0

Figure 6.22 shows how to enable INTP0.

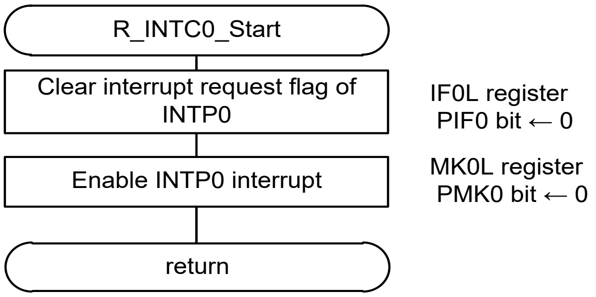


Figure 6.22 Enabling INTP0

6.8.19 INTP0 Interrupt Handler

Figure 6.23 shows the flowchart of the INTP0 interrupt handler.

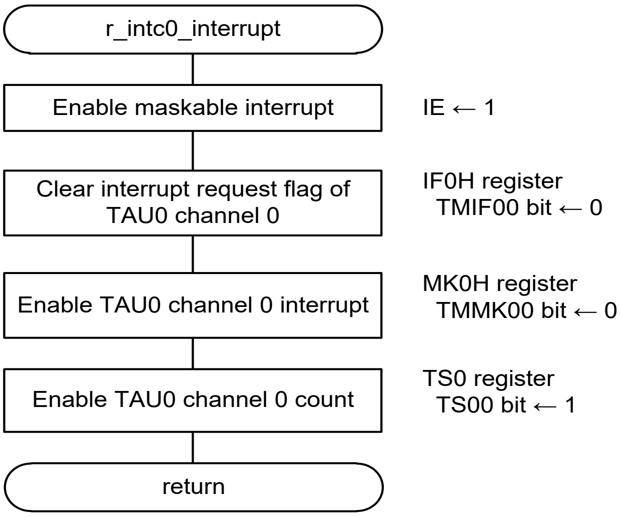


Figure 6.23 INTP0 Interrupt Handler

6.8.20 Enabling TAU00

Figure 6.24 shows how to enable TAU00.

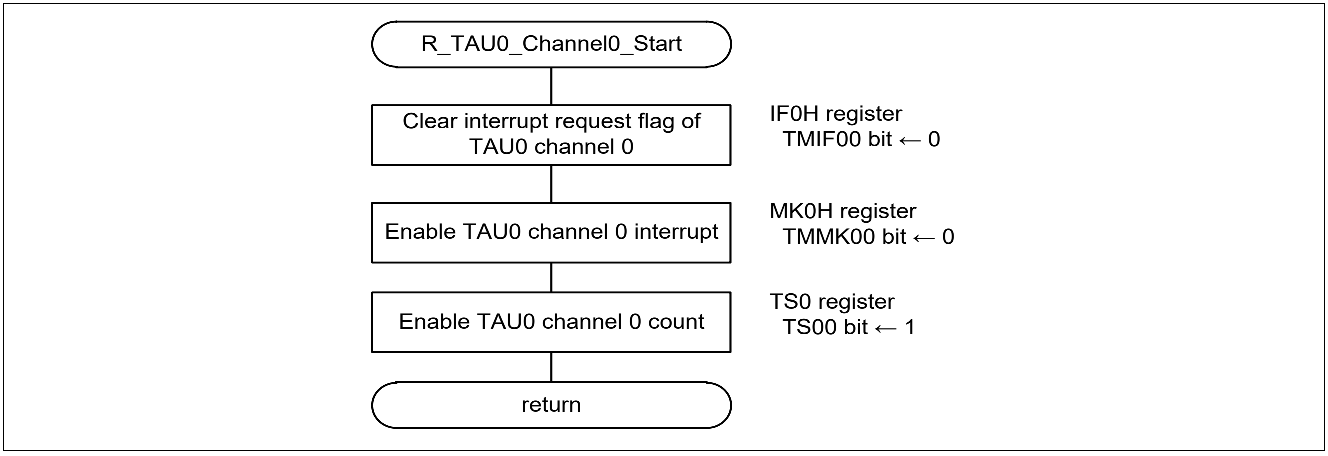


Figure 6.24 Enabling TAU00

6.8.21 TAU00 interrupt handler

Figure 6.25 shows the flowchart of the TAU00 interrupt handler.

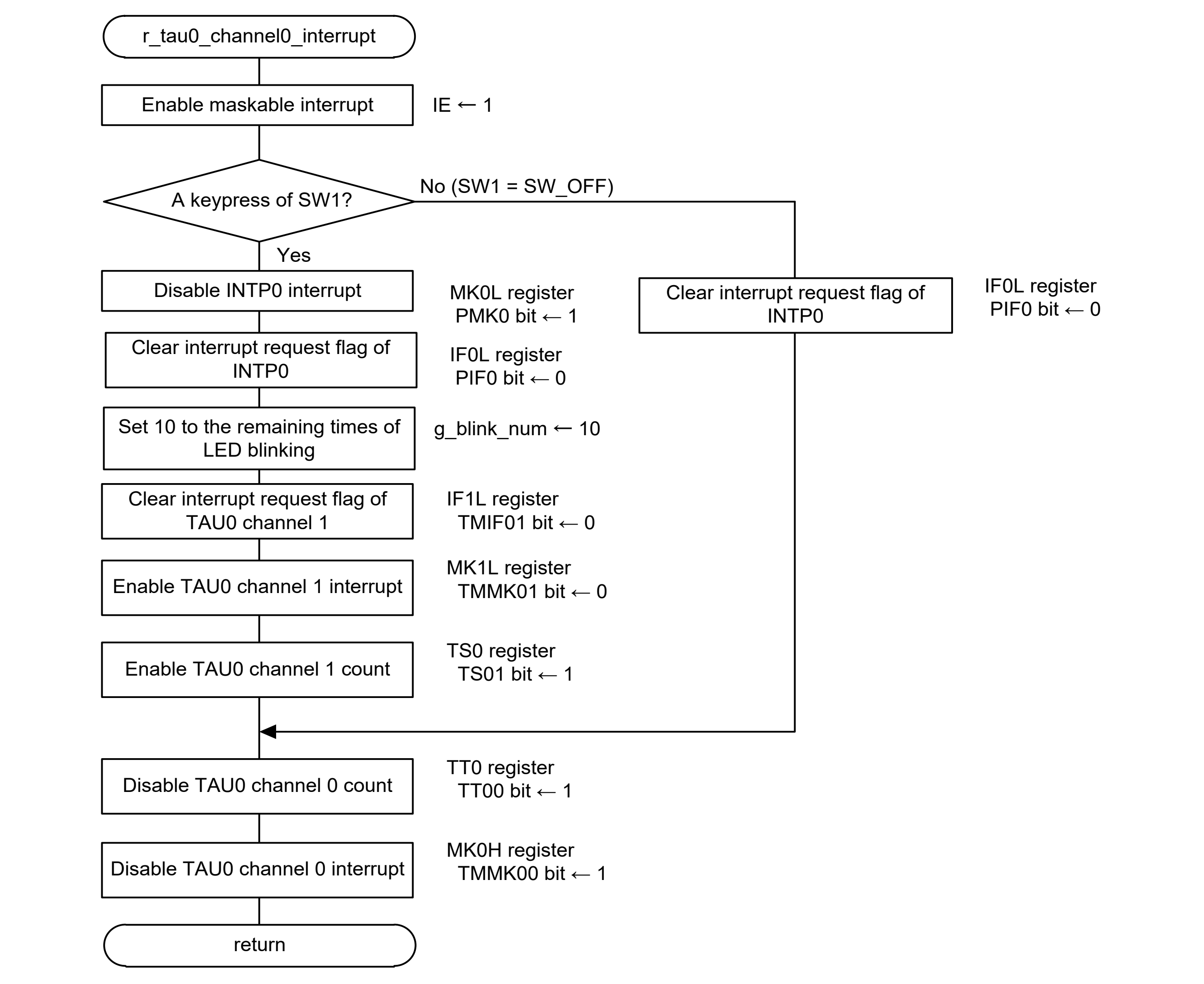


Figure 6.25 TAU00 interrupt handler

6.8.22 Acquisition of Writing Address

Figure 6.26 shows the flowchart of the acquisition of writing address.

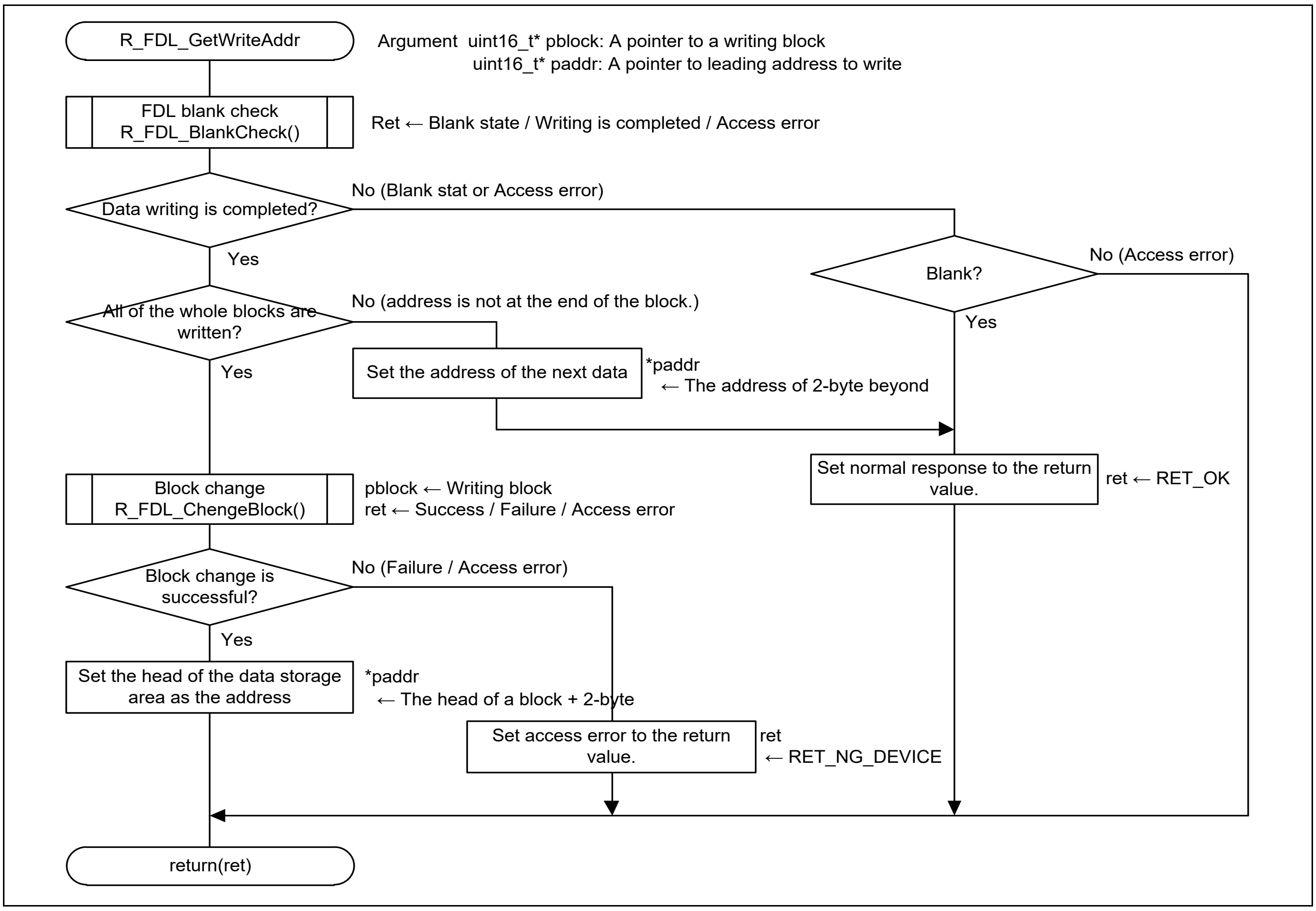


Figure 6.26 Acquisition of Writing Address

6.8.23 Write LED Blinking Data

Figure 6.27 shows how to write LED blinking data.

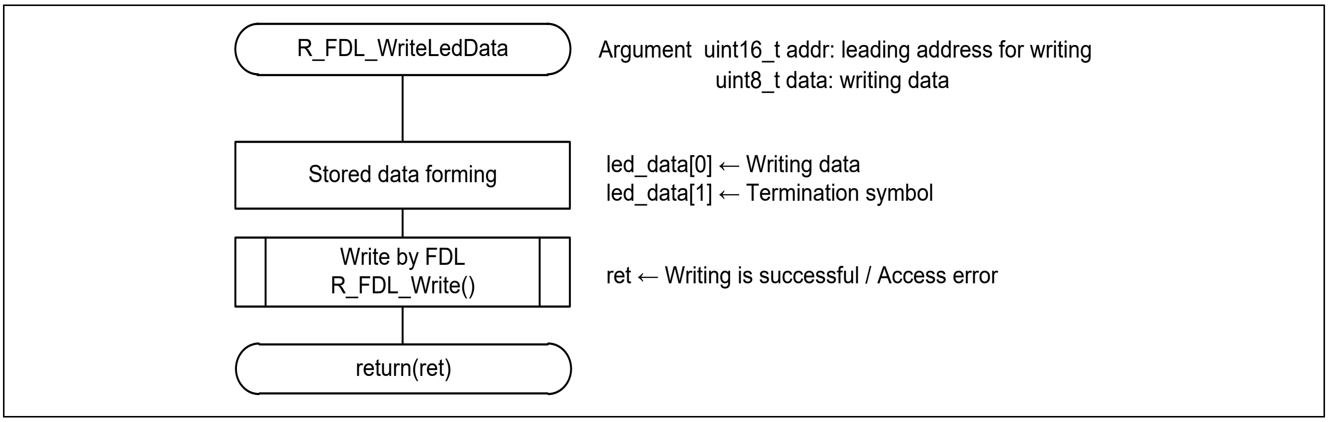


Figure 6.27 Write LED Blinking Data

6.8.24 Disabling INTP0

Figure 6.28 shows how to disable INTP0.

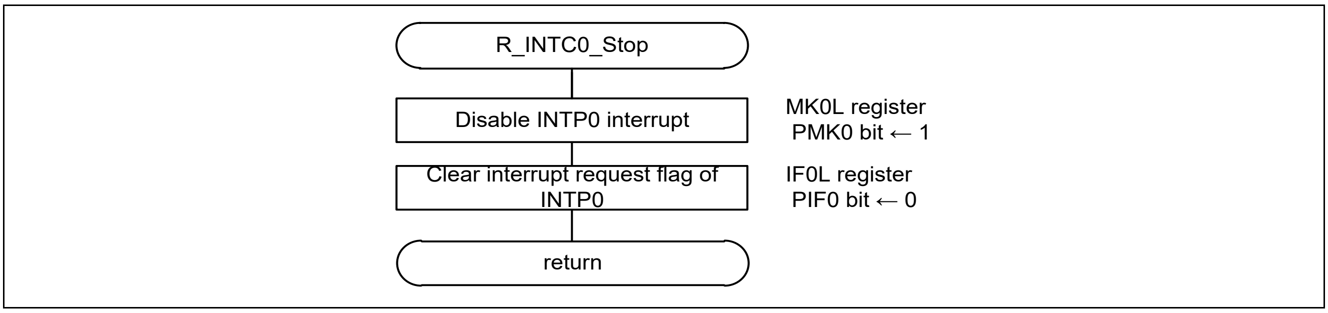


Figure 6.28 Disabling INTP0

6.8.25 Disabling TAU00

Figure 6.29 shows how to disable TAU00.

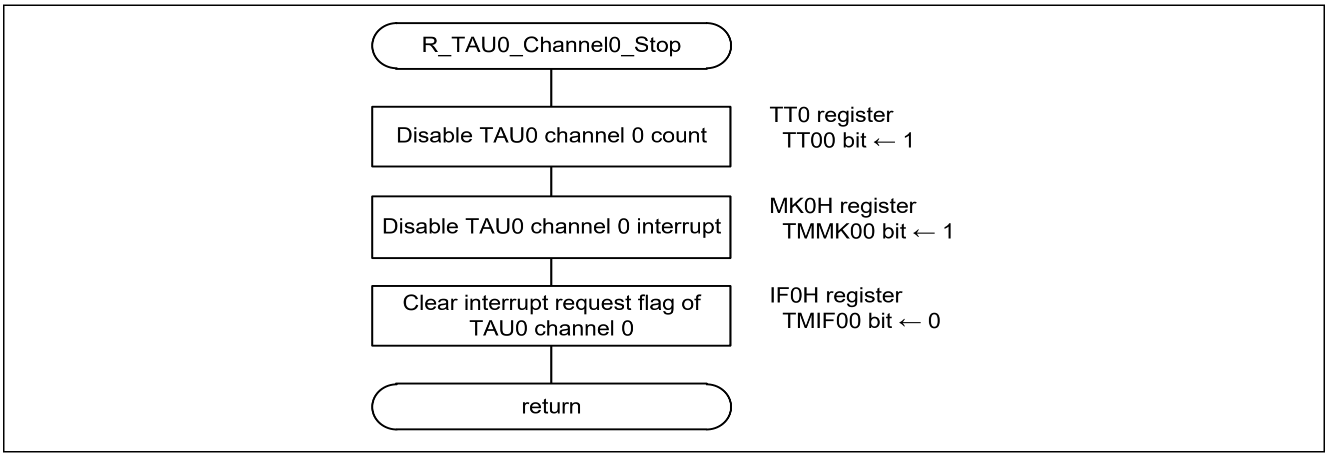


Figure 6.29 Disabling TAU00

6.8.26 Enabling LVD Interrupt

Figure 6.30 shows how to enable LVD interrupt.

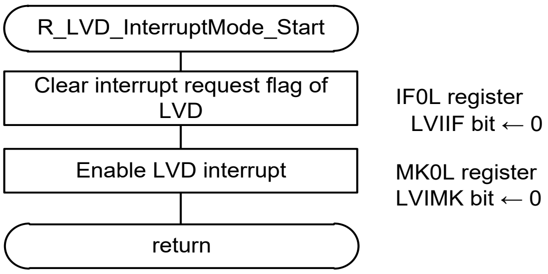


Figure 6.30 Enabling LVD Interrupt

6.8.27 LVD interrupt handler

Figure 6.31 shows the flowchart of the LVD interrupt handler.

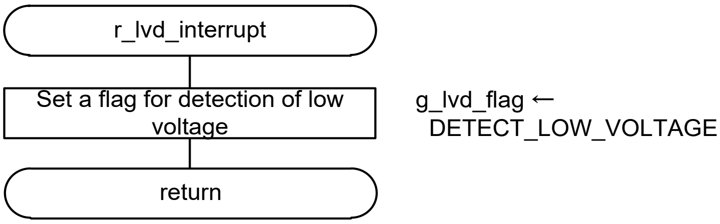


Figure 6.31 LVD interrupt handler

6.8.28 Management Data Read

Figure 6.32 and **Figure 6.33** show the flowchart of the management data read.

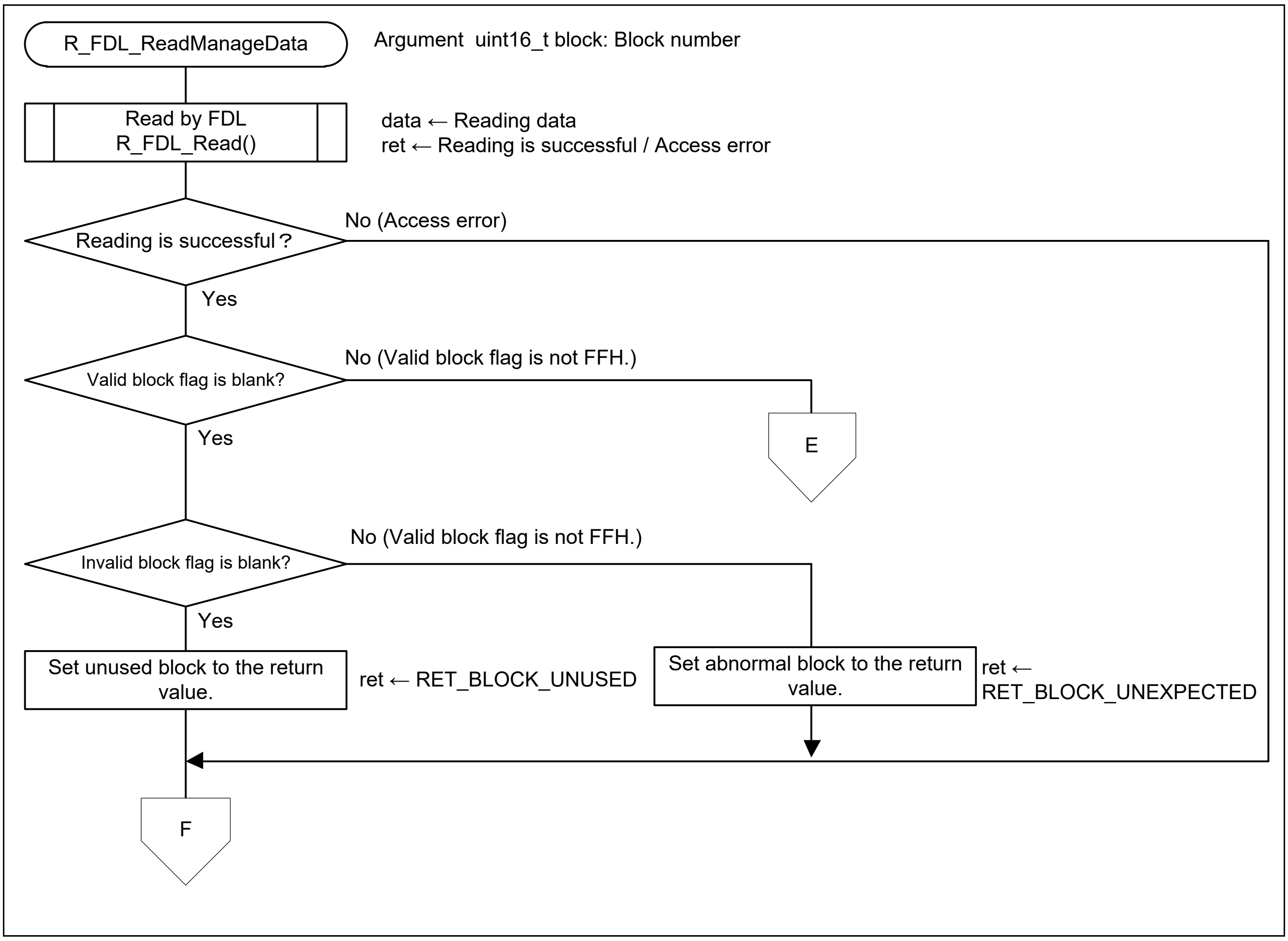


Figure 6.32 Management Data Read (1/2)

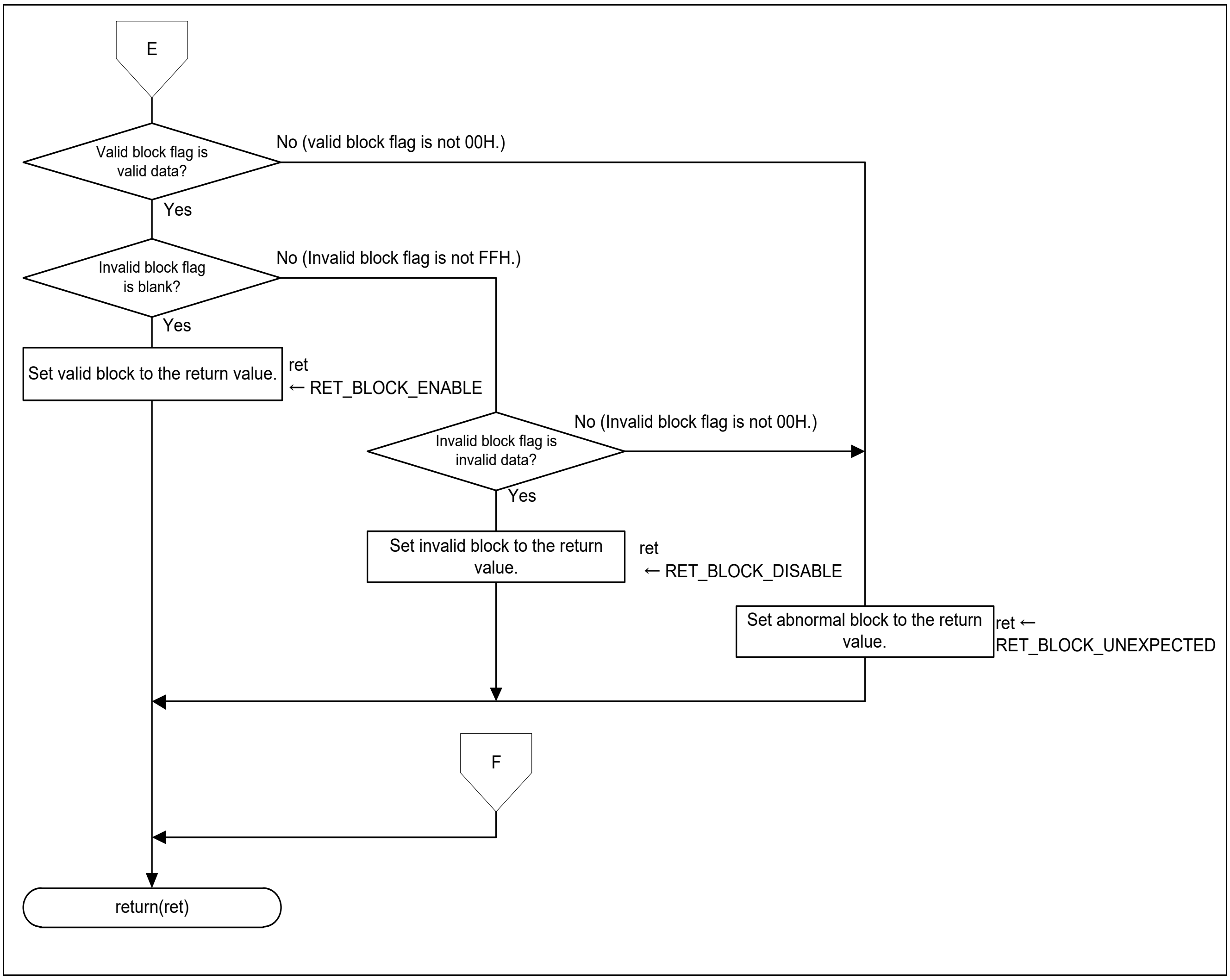


Figure 6.33 Management Data Read (2/2)

6.8.29 Read by FDL

Figure 6.34 shows how to read data by FDL.

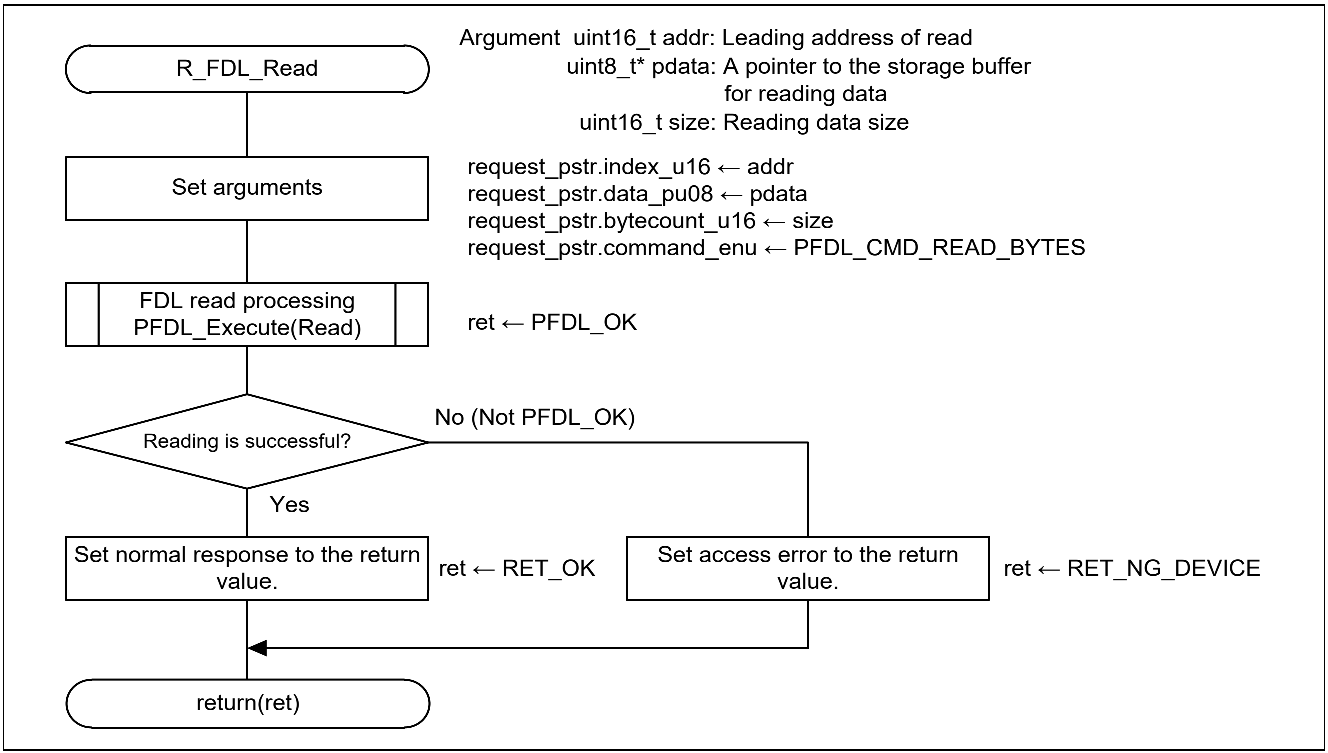


Figure 6.34 Read by FDL

6.8.30 FDL Blank Check

Figure 6.35 shows the flowchart of FDL blank check.

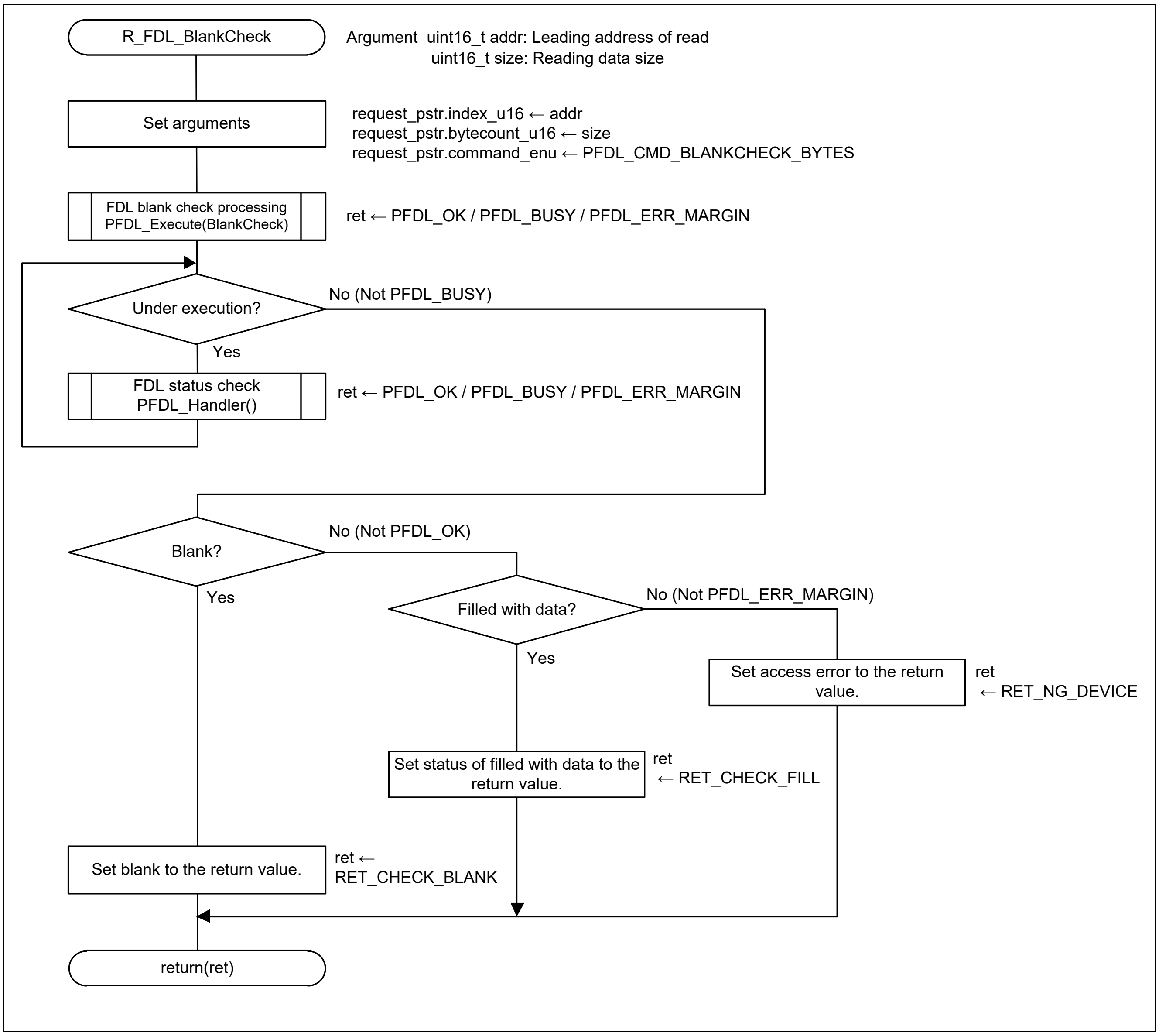


Figure 6.35 FDL Blank Check

6.8.31 Write by FDL

Figure 6.36 shows how to write by FDL.

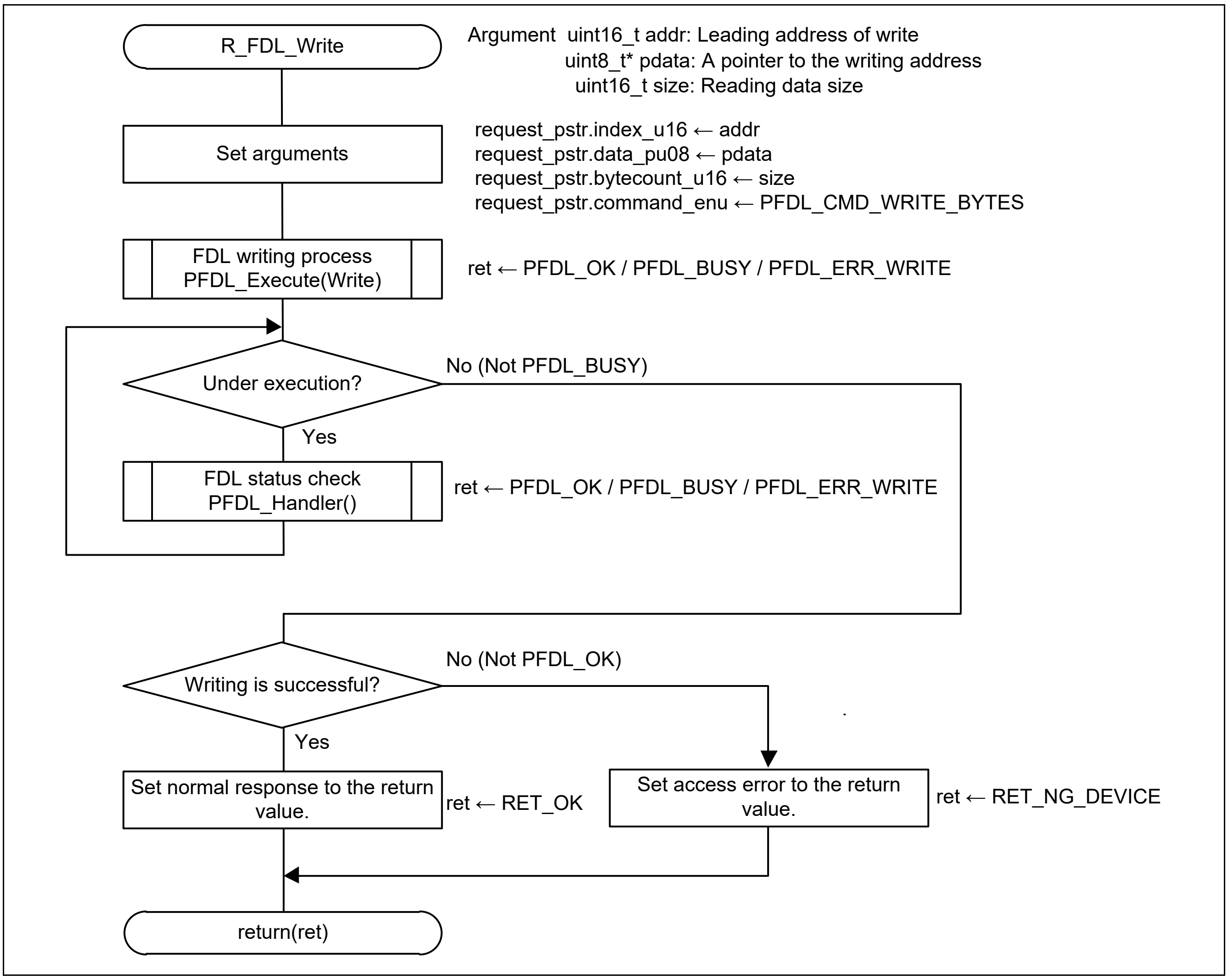


Figure 6.36 Write by FDL

6.8.32 Block Change

Figure 6.37 shows the flowchart of block change.

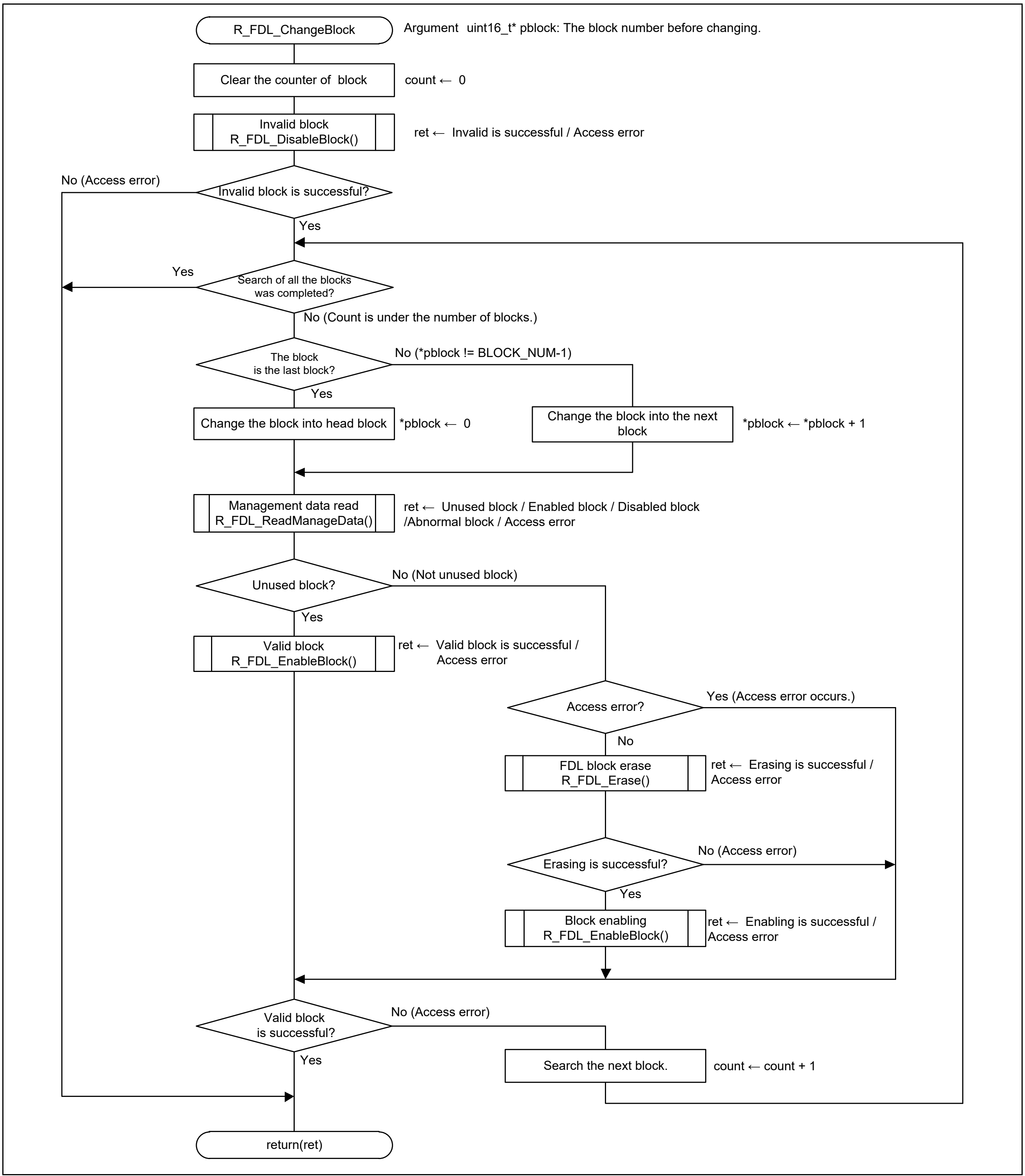


Figure 6.37 Block Change

6.8.33 Block Invalidation

Figure 6.38 shows the flowchart of the block invalidation.

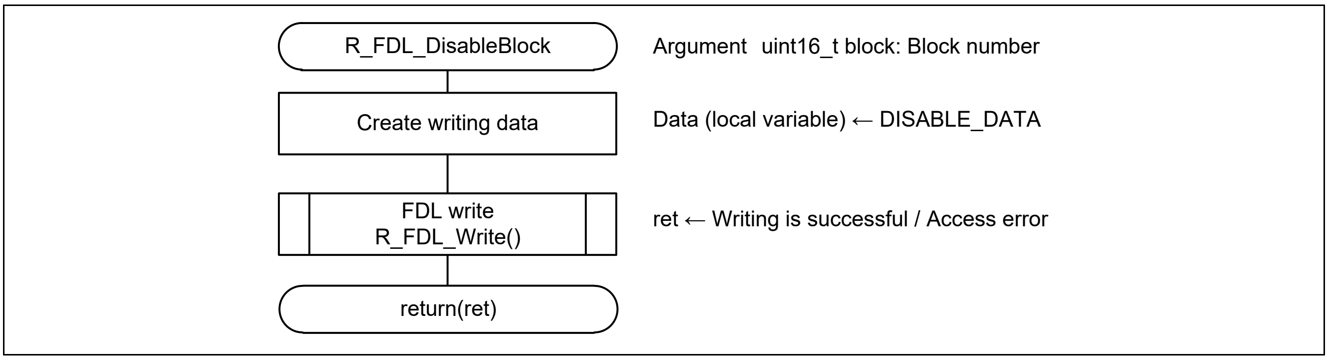


Figure 6.38 Block Invalidation

6.8.34 FDL Block Erasing

Figure 6.39 shows the flowchart of FDL block erasing.

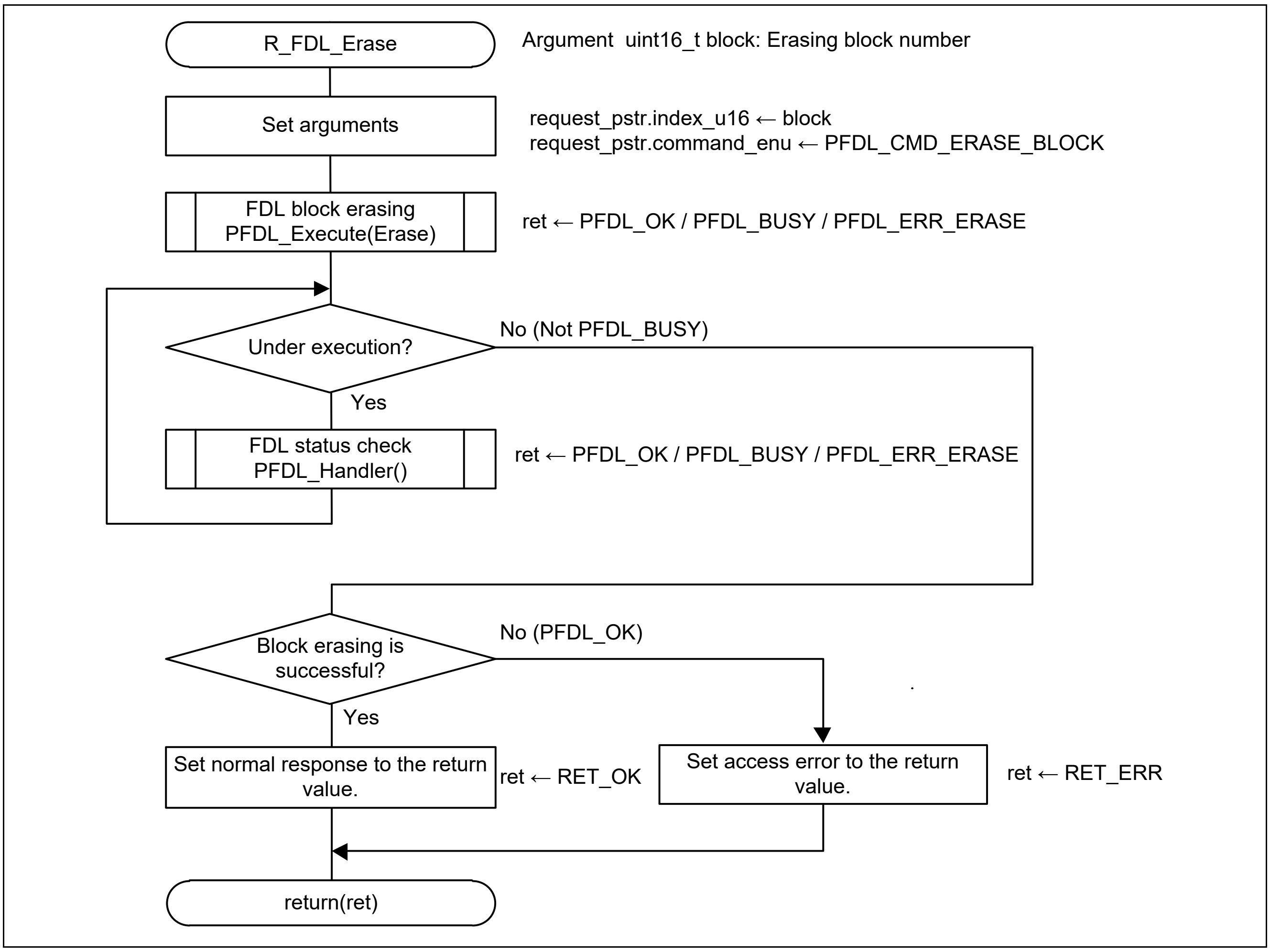


Figure 6.39 FDL Block Erasing

6.9 How to Import FDL into the Software Project

How to import FDL files used by this application into a project is indicated below.

6.9.1 CubeSuite+ Version

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

- pfdl.h
- pfdl_types.h
- pfdl.lib

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”.

6.9.2 GNU Version

(1) The following files are copied to src directory in a project.

- pfdl.h
- pfdl_types.h
- pfdl.a

(2) Select “Update” by right-click project name at project explorer of e2studio.

(3) Select “Properties” by right-click project name at project explorer of e2studio.

(4) In the Property window, select “Tool settings” tab by clicking on “C/C++ build” → “Environment”, and add pfdl.a to the additional input file at “Linker” → “Input” screen.

7. Sample Code

The sample code is available on the Renesas Electronics Website.

8. Documents for Reference

RL78/L12 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	**2014.03.25**	—	First edition issued
1.10	**2022.09.09**	1	Table Download link for CubeSuite+ version was changed and IAR version was deleted
		16	Table 3.1 IAR Ver. Development was deleted.
		23	Table 6.1 IAR version was deleted.
		26	Table 6.4 IAR version was deleted.
		67	6.9.2 IAR version was deleted.

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.

Notice

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

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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