V850E2/ML4

Updating Program Code by Using Flash Self Programming with CAN Controller

R01AN1343EJ0100
Rev.1.00
Mar. 01, 2013

Abstract

This document describes an example to update program code by reprogramming on-chip flash memory in V850E2/ML4 using flash self programming with CAN communication.

The features of the example to update program code in this Application note are described below.

- Reprograms a program code in the flash memory area using update program file with Intel expanded hex format received through the CAN communication.
- For the procedure in case of reprogram failure such as reprogram processing is aborted without intention, an error control register by checksum is included.

Products

V850E2/ML4

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Contents

1. Specifications 4

2. Operation Confirmation Conditions 5

3. Reference Application Notes 6

4. Peripheral Functions 6
4.1 Terms for Flash Self Programming 6
4.2 Notes for Flash Self Programming 7
4.2.1 Setting for Link Directive File 8
4.2.2 Setting for Non-use of Prologue/Epilogue Library 10
4.2.3 Setting for ROMization of Section in RAM 11
4.2.4 Setting for Far Jump Function 12
4.2.5 Setting for Startup Routine 14
4.2.6 Precautions for Interrupt Generated During Use of FSL 16

5. Hardware 17
5.1 Pins Used 17

6. Software 18
6.1 Operation Overview 18
6.1.1 Setting for Section Assignment 18
6.1.2 Overview of Reprogramming Flash Memory 19
6.1.3 Process from Startup to Normal Operation 20
6.1.4 Flash Reprogram Processing after Inputting INTP1 Interrupt 20
6.1.5 Data Receive Processing 20
6.1.6 Processing after Data Reception/Reprogram 21
6.1.7 Communication Control Sequence 22
6.2 File Composition 23
6.3 Constants 24
6.4 Variables 26
6.5 Functions 27
6.6 Function Specifications 28
6.7 Flowcharts 37
6.7.1 Startup Routine Processing 37
6.7.2 Main Processing 38
6.7.3 Switching Processing for Exception Handler Address 39
6.7.4 Checksum Judgment for Reprogram Area 40
6.7.5 Initialization of INTP1 Interrupt 41
6.7.6 INTP1 Interrupt Processing 42
6.7.7 Flash Reprogram Processing 43
6.7.8 Initialization of Flash Environment 45
6.7.9 Start Processing for Flash Environment 46
6.7.10 Checking Processing for FLMD0 Pin using FSL 47
6.7.11 Erase Processing for Specified Block 48
6.7.12 Write Processing from Specified Address 49

6.7.13 Internal Verification of Specified Block 50
6.7.14 Termination Processing for Flash Environment 51
6.7.15 Setting for FLMD0 Pin Level 52
6.7.16 Store Processing for Receive Data 53
6.7.17 Text Binary Conversion Processing 55
6.7.18 TAUA0 Initialization for LED Flash with Fixed-Cycle (Sample Function in Reprogram Area and in Spare Area) 56
6.7.19 TAUA0 Interval Timer Interrupt Processing 57
6.7.20 Initialization of CAN Controller Channel 0 (FCN0) 58
6.7.21 Initialization of FCN0 Port 59
6.7.22 Initialization of FCN0 Message Buffer 60
6.7.23 FCN0 Message Transmit Processing 62
6.7.24 FCN0 Transmit Processing 63
6.7.25 FCN0 Receive Processing 64
6.7.26 Interrupt Processing for FCN0 Receive Processing Completion 65
6.7.27 FCN0 Error Interrupt Processing 66

7. Operation Overview 67

8. Sample Code 69

9. Reference Documents 69

1. Specifications

In this Application note, a program code update is performed by reprogramming on-chip flash memory using flash self programming.

The CAN communication with an arbitrary device enables to receive a program file data for update with Intel expanded hex format type and reprogram a program code in the on-chip flash memory area.

Table 1.1 lists the Peripheral Functions and Their Applications and Figure 1.1 shows the System Configuration.

Table 1.1 Peripheral Functions and Their Applications

Peripheral Function	Application
Flash memory (on-chip flash memory)	Program store area
Flash macro service	Reprogram flash memory
CAN controller (FCN)	Reprogram data/message communication

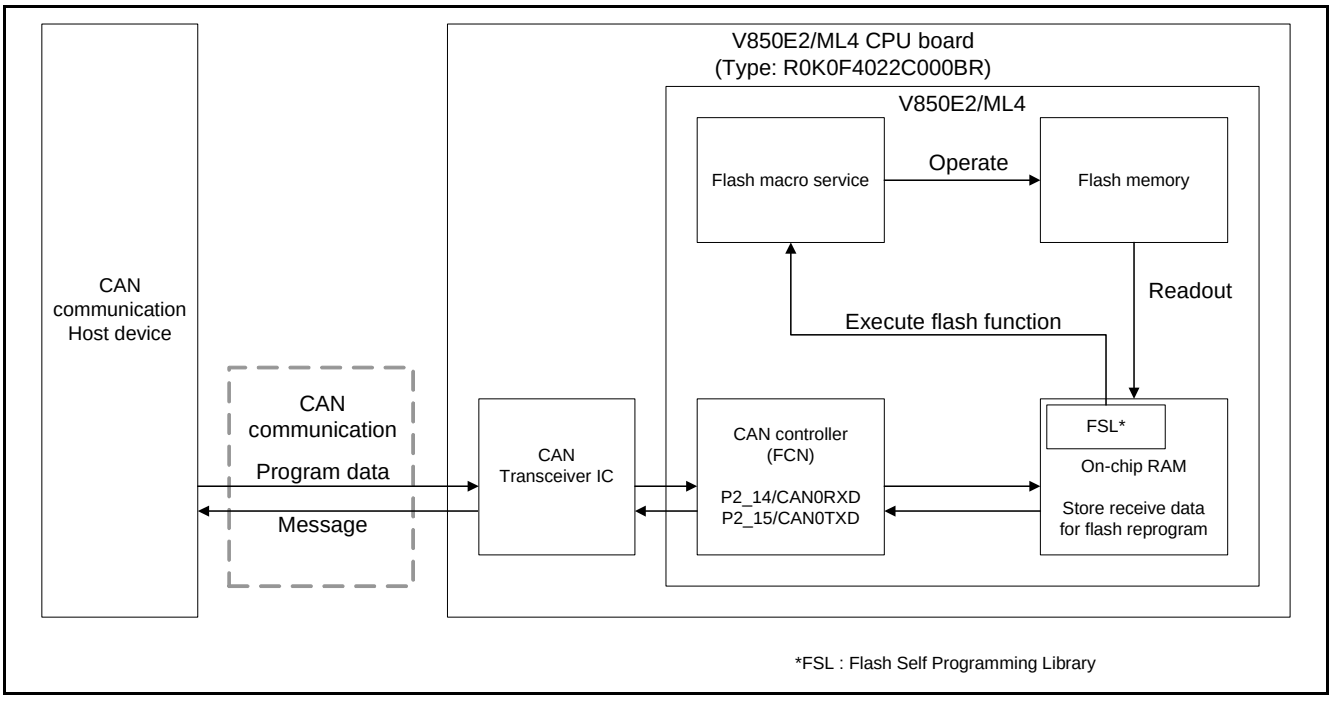


Figure 1.1 System Configuration

2. Operation Confirmation Conditions

The sample code accompanying this application note has been run and confirmed under the conditions below.

Table 2.1 Operation Confirmation Conditions

Item	Contents
MCU used	V850E2/ML4
Operating frequency	Internal system clock (f_{CLK}) : 200MHz P bus clock (f_{PCLK}) : 66.667MHz
Operating voltage	Positive power supply for external pins (EV_{DD}) : 3.3V Positive power supply for internal units (IV_{DD}) : 1.2V
Integrated development environment	Renesas Electronics Corporation CubeSuite+ Ver.1.02.01
C compiler	Renesas Electronics Corporation CX compiler package Ver.1.21
	Compile options -Cf4022 -oDefaultBuild\v850e2ml4_flash_update_can.lmf -Xobj_path=DefaultBuild -g -Xpro_epi_runtime=off -IC:\WorkSpace\v850e2ml4_flash_update_can\inc -IC:\WorkSpace\v850e2ml4_flash_update_can\FSL -Xdef_var -Xfar_jump=v850e2ml4_flash_update_can.fjp -Xlink_directive=v850e2ml4_flash_update_can.dir -Xstartup=DefaultBuild\cstart.obj +Xide -Xmap=DefaultBuild\v850e2ml4_flash_update_can.map -Xsymbol_dump -lFSL_T05_REC_R32 -LC:\WorkSpace\v850e2ml4_flash_update_can\FSL\lib -Xrompsec_text=FSL_CODE.text -Xrompsec_text=FSL_CODE_ROMRAM.text -Xrompsec_text=FSL_CODE_RAM.text -Xrompsec_text=FSL_CODE_RAM_USRINT.text -Xrompsec_text=FSL_CODE_RAM_USR.text -Xrompsec_text=FSL_CODE_RAM_EX_PROT.text -Xrompsec_text=INTP1RAM.text -Xrompsec_text=INTTAUA0I0RAM.text -Xrompsec_text=INTFCN0IERRRAM.text -Xrompsec_text=INTFCN0IRECRAM.text -Xhex=DefaultBuild\v850e2ml4_flash_update_can.hex
Operating mode	Normal operating mode (will be changed to flash memory programming mode at the time of reprogram)
Sample code version	1.00
Board used	R0K0F4022C000BR
Device used	CAN communication host device

3. Reference Application Notes

For additional information associated with this document, refer to the following application notes.

- V850 Microcontrollers Flash Self Programming Library Type05 (R01AN0661EJ)

4. Peripheral Functions

This chapter provides supplementary information on the flash self programming library which is required to reprogram the flash memory using the software operated on the V850E2/ML4. Refer to the "V850E2/ML4 User's Manual (Hardware)" and the "V850 Microcontroller Flash Self Programming Library Type05" for basic information.

4.1 Terms for Flash Self Programming

The terms for flash self programming used in this Application note are described as follows.

- Flash macro service
  This refers to functions for manipulating the flash memory in devices.

- Flash environment
  This refers to the state in which the code flash can be operated by using the flash macro service. There are special restrictions different from execution of normal programs. A transition to other environment cannot occur unless the flash environment is ended.

- Flash function
  This refers to the individual functions comprising the self-library. They can be used with the C language.

- Internal verification
  This refers to the action of internally checking the signal level and verifying that the signal can be read normally following write to flash memory.

4.2 Notes for Flash Self Programming

The V850E2/ML4 has the flash macro service which operates the flash memory. This sample program describes how to reprogram a program code using the flash self programing library (FSL) which enables to use the flash macro service with C language. The following notes are provided to use this library.

- The program allocation in RAM executed during the flash environment (including runtime library)
  - — Setting for a section to allocate the program in RAM
    Creation and setting for the link directive file is required to set a section. Refer to "4.2.1 Setting for Link Directive File" for more details.
  - — Setting for non-use or allocation in RAM for the functional prologue/epilogue runtime library
    This sample program runs the non-use setting of the prologue/epilogue runtime library. Refer to "4.2.2 Setting for Non-use of Prologue/Epilogue Library" for more details.
  - — Setting for the exception handler address switching function when using interrupts
    The setting for the exception handler address switching function is executed by the software. Refer to "6.7.3 Switching Processing for Exception Handler Address" for more details.
  - — Initialization of the program area in the RAM allocation destination
    When allocating a program to RAM on the V850E2/ML4, the 16-byte boundary area (H'xxxx_xxx0 to H'xxxx_xxxF) including the program area in the allocation destination is required to be initialized (cleared to zero). In this sample program, the initialization is executed during the startup routine. Refer to "4.2.5 Setting for Startup Routine" for its change, and "6.7.1 Startup Routine Processing" for its details.
  - — Setting for ROMization of the section to expand the program in RAM
    Regarding to the setting for ROMizaton on the CubeSuite+, refer to "4.2.3 Setting for ROMization of Section in RAM".

- The execution of the flash functions are disabled in the interrupt handler

- The far jump specification for the CX compiler when calling function allocated to the address separated more than 2 MB
  In this sample program, the far jump option is specified to the function allocated in RAM which is called from the flash memory. Refer to "4.2.4 Setting for Far Jump Function" for more details.

- Saving, setting and restoring the gp register and the ep register when accessing to the global variables with C language in the interrupt handler
  The above mentioned operations might be required when accessing to the data section in the interrupt handler. Refer to "4.2.6 Precautions for Interrupt Generated During Use of FSL" for more details.

In regard to the function specification and the system configuration of the FSL, refer to reference application note, "V850 Microcontroller Flash Self Programming Library Type05".

In regard to section specification to the CX compiler, allocation address setting, ROMization, and far jump option specification on the CubeSuite+, refer to "CubeSuite+ V1.03.00 Integrated Development Environment User's manual: Build (CX compiler)".

In regard to switching the exception handler address, refer to "V850E2 User's manual: Architecture".

4.2.1 Setting for Link Directive File

The link directive file creation and the CubeSuite+ setting are required to change the section assignment. When creating the link directive file using text editor without the CubeSuite+ menu, the Cube Suite+ setting is required. Drag the link directive file from explore, and drop it in blank area, the bottom part of the Project Tree. In the CubeSuite+, the file which has extension of "dir" or "dr" is considered as the link directive file. Select "CX (Build Tool)" under the Project Tree, and click "Link Options" tab in the Property. Open "Input File" to check "Using link directive file". Refer to "CubeSuite+ V1.03.00 Integrated Development Environment User's manual: Coding (CX compiler)" for more details.

When creating the link directive file, in this sample program,the reprogram area section (MasterPRG.text), the spare area section (SparePRG.text), and the FSL area (FSL.CONST) should be created in the flash memory other than the default area. In addition, the FSL use area and user program area sections (FSL_DATA.bss, FSL_CODE.text, FSL_CODE_ROMRAM.text, FSL_CODE_RAM.text, FSL_CODE_RAM_USRINT.text, FSL_CODE_RAM_USR.text, and FSL_CODE_RAM_EX_PROT.text), and exception handler address sections (INTP1RAM.text, INTTAUA0I0RAM.text, INTFCN0IERRRAM.text, and INTFCN0IRECRAM.text) should be created in RAM.

In this sample program, the start address of the MasterPRG.text section is assumed to be H'0000 8000. Also the start address of the exception handler address section is assumed to be the address that adds the respective interrupt handler address to the transfer destination base address H'FEDF E000

Figure 4.1 shows the Location of Link Directive File.

Figure 4.2 shows the Example of Creation and Section Setting for Link Directive File.

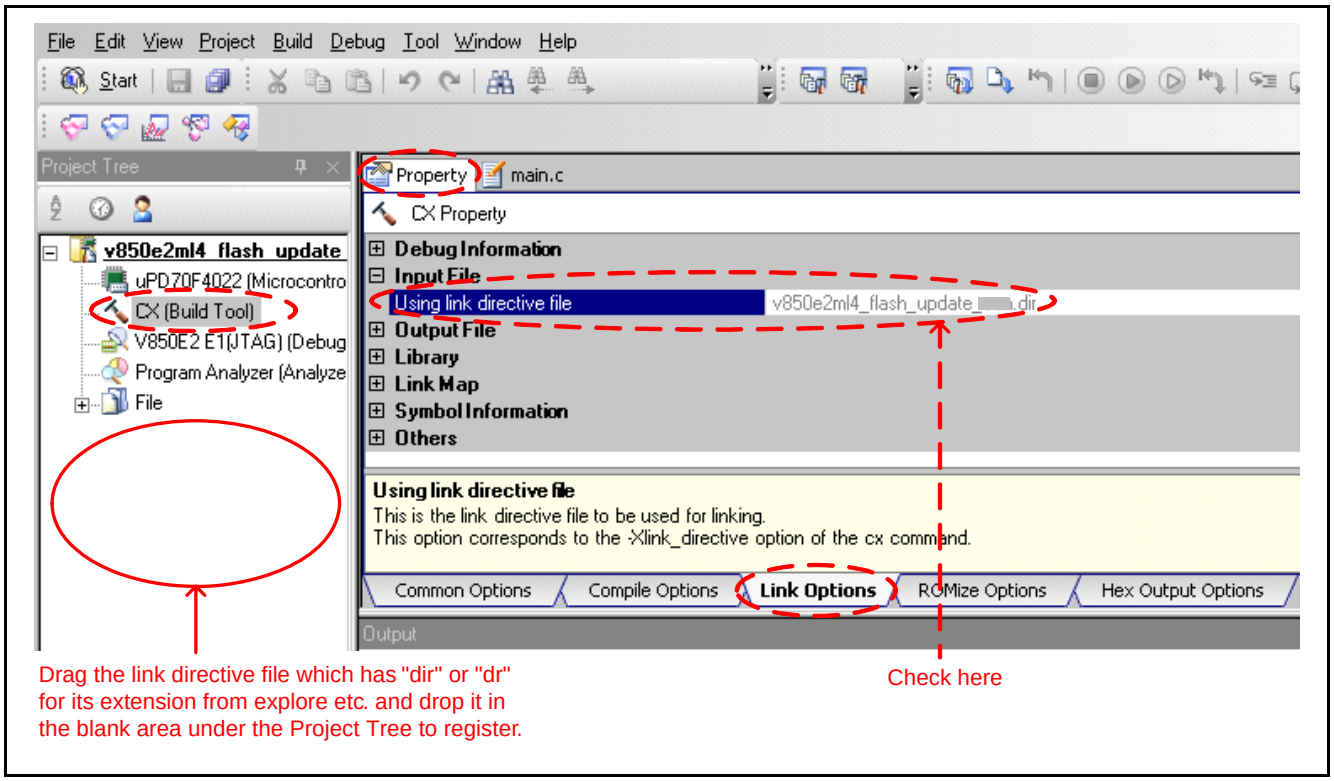


Figure 4.1 Location of Link Directive File

```
SCONST:!LOAD ?R {
.sconst = $PROGBITS ?A .sconst ;

};
CONST:!LOAD ?R V0x00001100 {
.const = $PROGBITS ?A .const ;
FSL_CONST.const = $PROGBITS ?A FSL_CONST.const ;    # FSL use area

};
TEXT:!LOAD ?RX {
.pro_epi_runtime = $PROGBITS ?AX .pro_epi_runtime ;
.text = $PROGBITS ?AX .text ;

};

# Spare area
SparePRG:!LOAD ?RX V0x00006000 {
SparePRG.text = $PROGBITS ?AX V0x00006000 SparePRG.text ;

};

# Reprogram area
MasterPRG:!LOAD ?RX V0x00008000 {
MasterPRG.text = $PROGBITS ?AX V0x00008000 MasterPRG.text ;

};

DATA:!LOAD ?RW V0xfedf0000 {
.data = $PROGBITS ?AW .data ;
.sdata = $PROGBITS ?AWG .sdata ;
.sbss = $NOBITS ?AWG .sbss ;
FSL_DATA.bss = $NOBITS ?AW FSL_DATA.bss ;    # FSL use area
.bss = $NOBITS ?AW .bss ;

};
SEDATA:!LOAD ?RW {
.sedata = $PROGBITS ?AW .sedata ;
.sebss = $NOBITS ?AW .sebss ;

};
SIDATA:!LOAD ?RW {
.tidata.byte = $PROGBITS ?AW .tidata.byte ;
.tibss.byte = $NOBITS ?AW .tibss.byte ;
.tidata.word = $PROGBITS ?AW .tidata.word ;
.tibss.word = $NOBITS ?AW .tibss.word ;
.tidata = $PROGBITS ?AW .tidata ;
.tibss = $NOBITS ?AW .tibss ;
.sidata = $PROGBITS ?AW .sidata ;
.sibss = $NOBITS ?AW .sibss ;

};

# Program area allocated in RAM
RAM_PROG:!LOAD ?RX V0xfedfc000 {
FSL_CODE.text = $PROGBITS ?AX FSL_CODE.text ;
FSL_CODE_ROMRAM.text = $PROGBITS ?AX FSL_CODE_ROMRAM.text ;
FSL_CODE_RAM.text = $PROGBITS ?AX FSL_CODE_RAM.text ;
FSL_CODE_RAM_USRINT.text = $PROGBITS ?AX FSL_CODE_RAM_USRINT.text ;
FSL_CODE_RAM_USR.text = $PROGBITS ?AX FSL_CODE_RAM_USR.text ;
FSL_CODE_RAM_EX_PROT.text = $PROGBITS ?AX FSL_CODE_RAM_EX_PROT.text ;

};

# Exception handler area allocated in RAM
INTRAM:!LOAD ?RX V0xfedfe000 L0x00001080 {
INTP1RAM.text = $PROGBITS ?AX V0xfedfe170 H0x0000000a INTP1RAM.text ;
INTTAUA0I0RAM.text = $PROGBITS ?AX V0xfedfe3b0 H0x0000000a INTTAUA0I0RAM.text ;
INTFCN0IERRRAM.text = $PROGBITS ?AX V0xfedfeb30 H0x0000000a INTFCN0IERRRAM.text ;
INTFCN0IRECRAM.text = $PROGBITS ?AX V0xfedfeb40 H0x0000000a INTFCN0IRECRAM.text ;

};

__tp_TEXT@ %TP_SYMBOL ;
__gp_DATA@ %GP_SYMBOL &__tp_TEXT { DATA  } ;
__ep_DATA@ %EP_SYMBOL ;
```

← Create section for FSL use area on the ROM

← Create segment and section for spare area on the ROM

← Create segment and section for reprogram area on the ROM

← Create section for FSL use area on the RAM

← Create segment and section for FSL area and user program area on the RAM

← Create segment and section for exception handler on the RAM

Figure 4.2 Example of Creation and Section Setting for Link Directive File

4.2.2 Setting for Non-use of Prologue/Epilogue Library

The CubeSuite+ executes setting for non-use of the prologue/epilogue library. Select "CX (Build Tool)" under the Project Tree, and click "Compile Options" tab in the Property. Select "No (-Xpro_epi_runtime=off)" for "Use prologue/epilogue library" in "Optimization (Details)".

Figure 4.3 shows the Location of Setting Non-Use of Prologue/Epilogue Library.

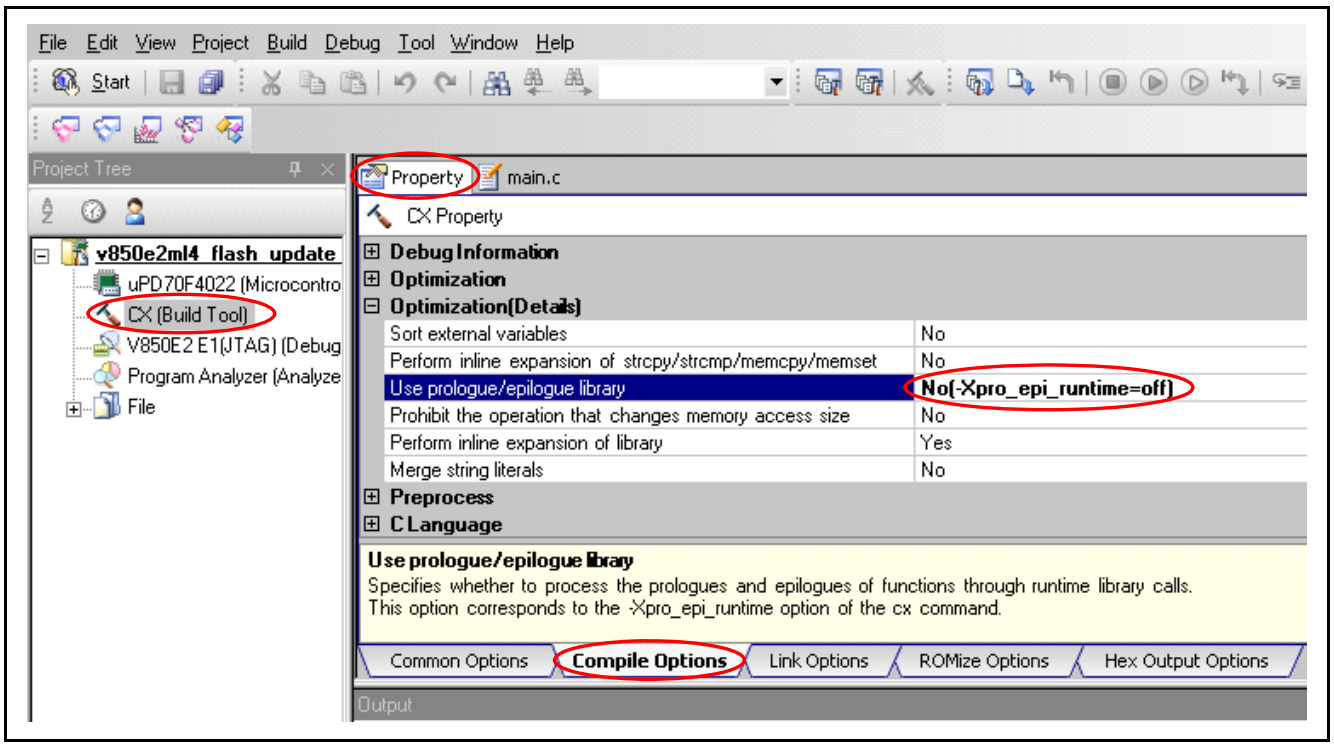


Figure 4.3 Location of Setting Non-Use of Prologue/Epilogue Library

4.2.3 Setting for ROMization of Section in RAM

The setting for CubeSuite+ is required for ROMization to expand the section in RAM. Select "CX (Built Tool)" under the Project Tree, and click "ROMize Options" tab in the "Property". From "Text sections included rompsec section", specify the section required for ROMization out of the sections to be assigned in RAM. Write the section names (one section per line) in the "Text Edit" window shown by clicking the "…" button on the right.

Figure 4.4 shows the Setting for Romization of Section in RAM.

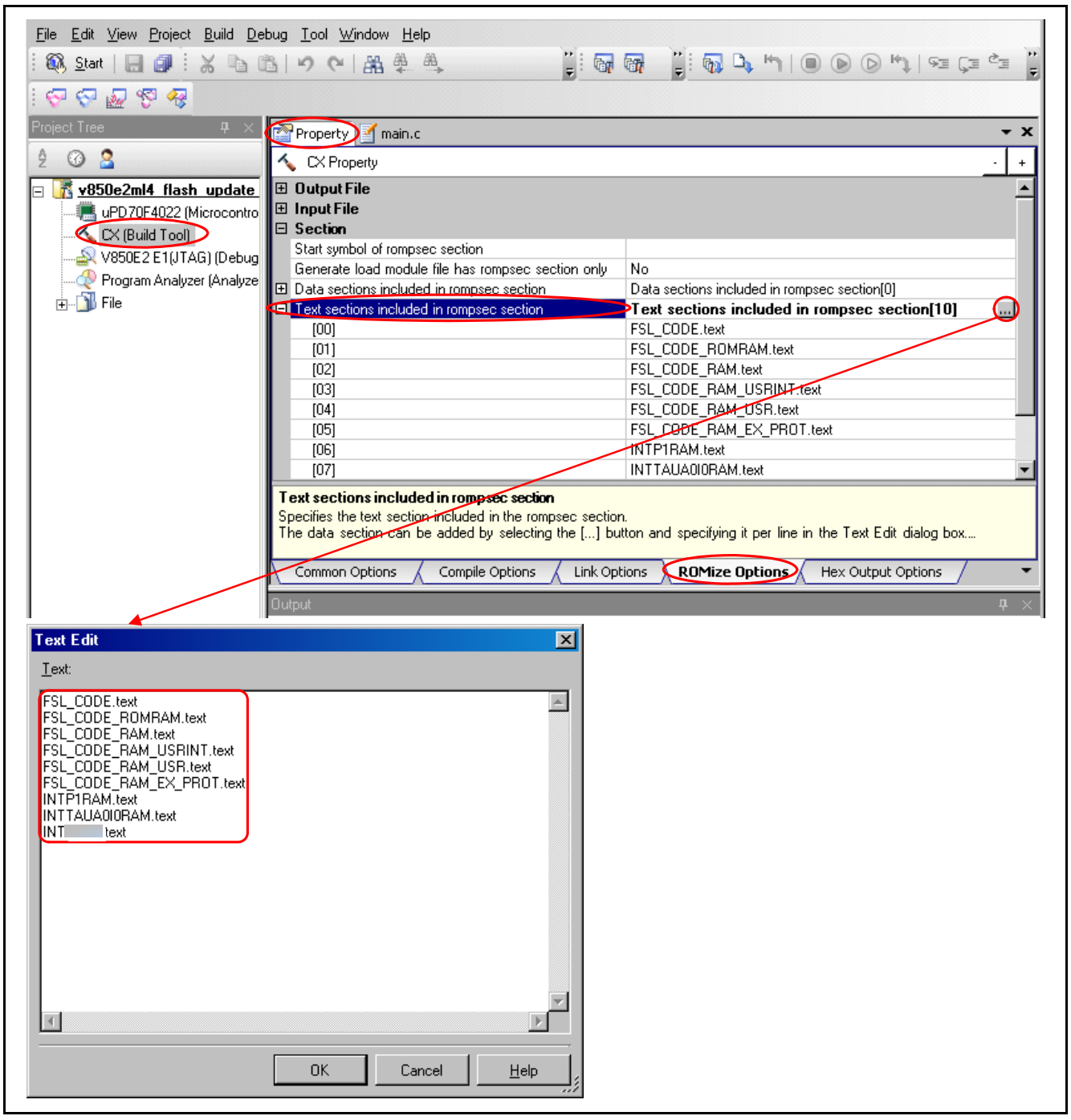


Figure 4.4 Setting for Romization of Section in RAM

4.2.4 Setting for Far Jump Function

In the V850E2/ML4, the end address of the flash memory and the start address of the on-chip RAM are separated more than 2MB. In the CX compiler, when jumping to the area more than ±2MBs away at the time of function call, the far jump option should be specified to the call destination function. In this sample code, the far jump option is specified to the functions called from the ones on the flash memory out of the functions allocated in the on-chip RAM and all interrupt handlers to be used.

To specify the far jump option, create the file which lists the functions to be specified (far jump calling function list file) and specify the file name in the compile option "-Xfar_jump". To set in the CubeSuite+, select "CX (Built Tool)" under the Project Tree, and click "Compile Options" tab in the Property. Click "…" button shown on the right side of "Far Jump file names" in "Output Code", and write the path of the created far jump calling function list file. (Note that ".fjp" is recommended for the extension of the far jump calling function list file.)

In the far jump calling function list file, write one function name per line. The function name should have "_ (underscore)" at the beginning of the function name with C language. Note that if "{all_interrupt}" is written, all interrupt handler functions are subject for the far jump calling functions. For creation of far jump calling function file, refer to "3.3.3 far jump function" in "CubeSuite+ V1.03.00 Integrated Development Environment User's manual: Coding (CX compiler)"

Figure 4.5 shows the Location of Far Jump Calling Function File.

Figure 4.6 shows the Example of Creation of Far Jump Calling Function File.

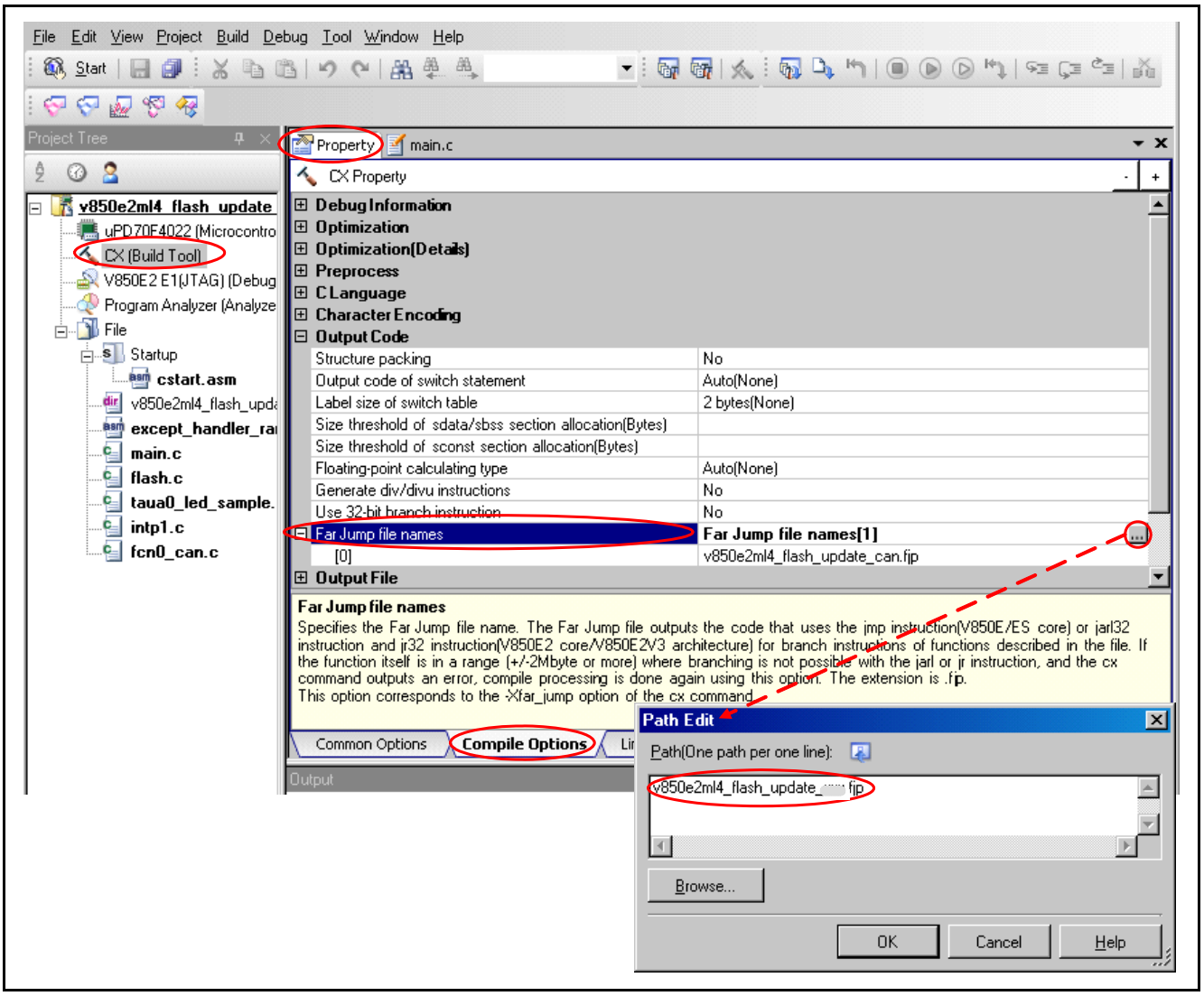


Figure 4.5 Location of Far Jump Calling Function File

```
_fcn0_can_tx_msg
```
Because the fcn0_tx_msg function on RAM is also called from the main function on ROM, far jump specification is required.

```
_flash_reprogram
```
Because the flash_reprogram function on RAM is also called from the main function on ROM, far jump specification is required.

```
{all_interrupt}
```
All interrupt handler functions are subject for far jump specification.
The interrupt hander is allocated on RAM, but far jump specification is required because the exception vector table is allocated on RAM with default (pre-change base address)

Figure 4.6 Example of Creation of Far Jump Calling Function File

4.2.5 Setting for Startup Routine

The stack used in this sample program requires larger area than the stack size (512 bytes) which is set in the standard startup routine. In the standard startup routine, the function "_rcopy" (ROMize processing) is executed to develop the data with initial value and the program allocated in RAM. However, when executing the ROMize processing for the program area, initialize (clear to 0) the 16-byte boundary area of program destination before executing "_rcopy". In this sample program, the initialization processing for the stack size change and the 16-byte boundary area of program destination is added for the assembler source file "cstart.asm" in which the standard startup routine is written.

When switching the standard startup routine, create the user-created assembler source file in which the startup routine is written to register on the CubeSuite+ project. Right click "startup" in "file" under the Project tree, then the menu will appear to add the startup routine source file.

Figure 4.7 shows the Location of Startup Routine.

Figure 4.8 shows the Example of Startup Routine Preparation (Excerpt from cstart.asm) .

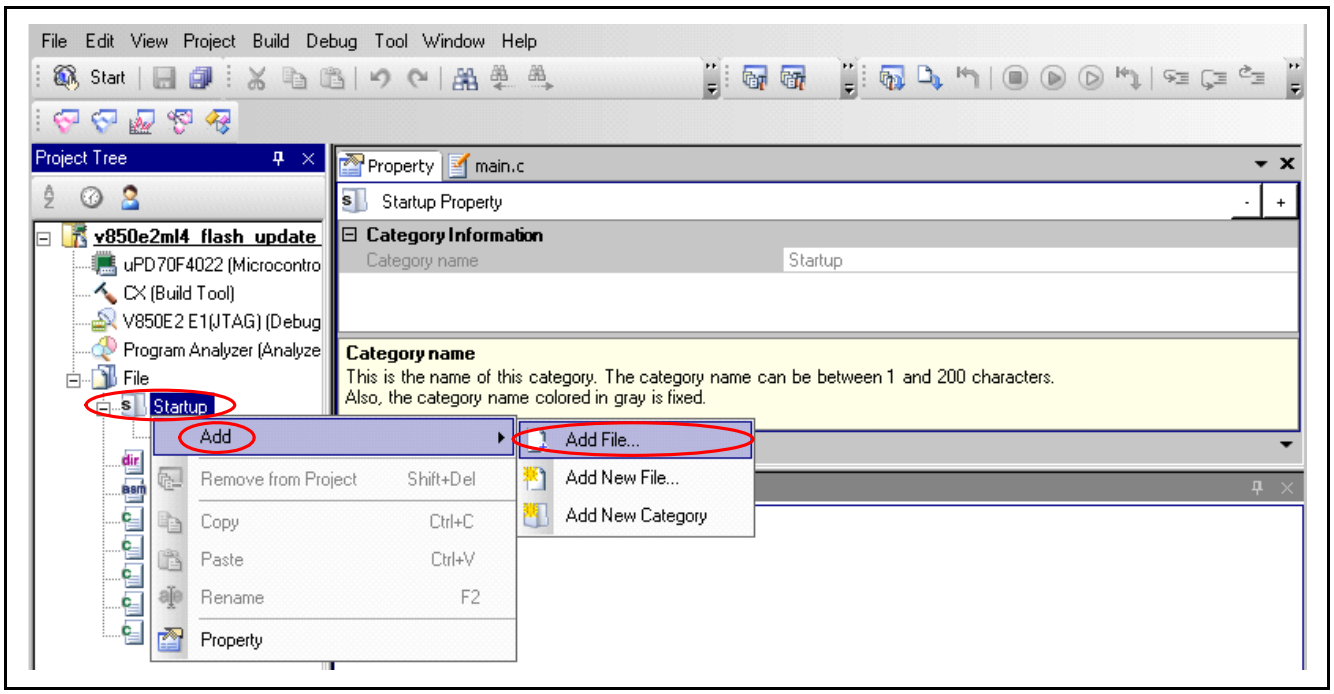


Figure 4.7 Location of Startup Routine

```
                                        :
                                        : (Excerpt from cstart.asm)
                                        :
#-------------------------------------------------------------------------------
#       system stack
#-------------------------------------------------------------------------------
STACKSIZE       .set    0x500
        .dseg   bss
        .align 4
__stack:        .ds     (STACKSIZE)


#-------------------------------------------------------------------------------
#       RESET vector
#-------------------------------------------------------------------------------

RESET   .cseg   text
        jr      __start

        .cseg   text
        .align 4
__start:
        mov32   #__tp_TEXT, tp          ;  set tp register
        mov32   #__gp_DATA, gp          ;  set gp register offset
        add     tp, gp                  ;  set gp register
        mov32   #__stack+STACKSIZE, sp      ;  set sp register
        mov32   #__ep_DATA, ep          ;  set ep register

        mov32   #___PROLOG_TABLE, r12   ;  for prologue/epilogue runtime
        ldsr    r12, 20                 ;  set CTBP (CALLT base pointer)

        jarl    _hdwinit, lp            ;  initialize hardware

        mov32   #__ssbss, r6            ;  clear sbss section
        mov32   #__esbss, r7
        jarl    __zeroclrw, lp

        mov32   #__sbss, r6             ;  clear bss section
        mov32   #__ebss, r7
        jarl    __zeroclrw, lp

        mov32   0xfedfc000, r6          ;  clear ram_prog section for e2core prefetch processing
        mov32   0xfedfffff, r7
        jarl    __zeroclrw, lp

                                        :
                                        : (Continue)
                                        :
```

Change the stack size to execute FSL and user program

Clear periphery of the area to be used as a program on RAM to zero before executing "_rcopy"

Figure 4.8 Example of Startup Routine Preparation (Excerpt from cstart.asm)

4.2.6 Precautions for Interrupt Generated During Use of FSL

When accessing to the data using the gp register or the ep register in the interrupt processing generated during the use of the FSL, set appropriate values to the gp register or the ep register before accessing to the data. The saving process for the gp register or the ep register is required before setting the appropriate values to the registers. Furthermore, the restoring process for the gp register or the ep register is required before returning from the interrupt processing. If the said measures are not executed, the data access using the gp register or the ep register cannot be operated properly.

- Sections when accessing to the gp register as a base address:
  (The created global variables without section specification will be allocated to .sdata or .sbss.)
  — .data
  — .bss
  — .sdata
  — Sbss

- Sections when accessing to the ep register as a base address:
  — .sedata
  — .sebss
  — .sidata
  — .sibss
  — .tidata.byte
  — .tibss.byte
  — .tidata.word
  — .tibss.word

This sample program does not use a section which accesses to the ep register as a base address and therefore the saving, setting, and restoring processes for the ep register are not executed in the interrupt processing. The V850E2/ML4 does not require the saving, setting, and restoring of the gp register when using the FSL.

When changing the microcomputer or using the above sections, the saving, setting, and restoring of the gp register or ep register may be necessary in the interrupt processing. Cautions are required when applying.

5. Hardware

5.1 Pins Used

Table 5.1 lists the Pins Used and Their Functions.

Table 5.1 Pins Used and Their Functions

Pin Name	I/O	Function
P2_14/CAN0RXD	**Input**	CAN0 receive data input
P2_15/CAN0TXD	**Output**	CAN0 transmit data output
P2_3/INTP1	**Input**	INTP1 interrupt

6. Software

6.1 Operation Overview

This sample program receives a program file data for update with Intel expanded hex format using the CAN communication, and reprograms the program in the flash memory area. This section describes its operation overview.

6.1.1 Setting for Section Assignment

The access to the flash memory is prohibited while the flash memory is reprogrammed. All programs that are used during the reprogram of flash memory should be transferred to the area except flash memory. This sample program sets section assignment to transfer all the sections used during the reprogram to the on-chip RAM.

Table 6.1 lists the Sections Used During Flash Memory Reprogram.

Table 6.1 Sections Used During Flash Memory Reprogram

Section name	Program detail	Function name
FSL_CODE_ROMRAM.text, FSL_CODE_RAM.text, FSL_CODE_RAM_EX_PROT.text	FSL area	Flash function
FSL_CODE_RAM_USRINT.text	User program interrupt section for RAM	fcn0_can_rx_isr, fcn0_can_error_isr, flash_store_can_data, hex2bin, intp1_isr, taua0_ch0_interval_timer_isr
FSL_CODE_RAM_USR.text	User program section for RAM	fcn0_can_tx_msg, fcn0_can_tx, fcn0_can_rx, flash_reprogram, flash_init, flash_activate, flash_modecheck, flash_erase, flash_write, flash_iverify, flash_end, flash_set_flmd0
INTP1RAM.text, INTTAUA0I0RAM.text, INTFCN0ITRXRAM.text, INTFCN0IRECRAM.text	Jump instruction to interrupt handler function	None

This sample program additionally assigns a section area to store a spare program as a solution when the flash memory reprogram processing failed to reprogram properly such as abort without any intention. For the reprogram area and the spare area before receiving data (initial state), the programs which have the same processing are stored in respective area.

Table 6.2 lists the Functions and Sections Specifying Addresses on Flash Memory.

Table 6.2 Functions and Sections Specifying Addresses on Flash Memory

Item	Start address (block number)	Store function name	ROM section name
Reprogram area	H'0000 8000 (8)	taua0_led_sample	MasterPRG.text
Spare area	H'0000 6000 (6)	taua0_led_spare	SparePRG.text

6.1.2 Overview of Reprogramming Flash Memory

Figure 6.1 shows the Overview of Reprogramming Flash Memory.

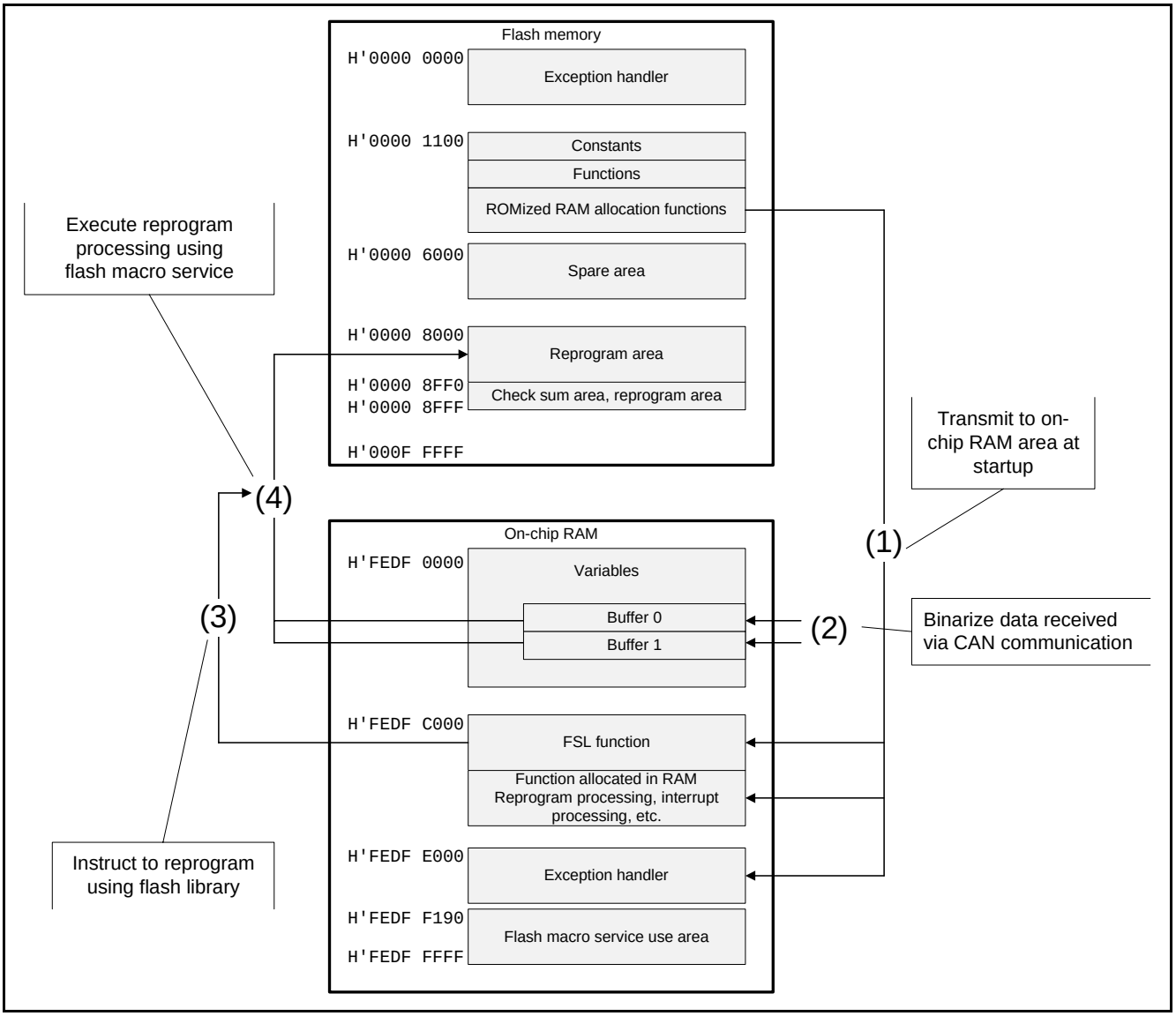


Figure 6.1 Overview of Reprogramming Flash Memory

1. After cancelling the reset, the __S_romp (ROMized section group) is copied to the on-chip RAM during the cstart.asm processing before starting the main function.
2. The Intel Extend Hex format data received via CAN communication is stored to the on-chip RAM with the state of binary data which executes writing.
3. The operation for the flash macro service is executed by the flash library function which is assigned to the on-chip RAM.
4. The flash macro service executes the reprogram processing of on-chip flash memory.

6.1.3 Process from Startup to Normal Operation

After the system activation, execute initializations in the main processing, and transmit a message "Generate INTP1 interrupt for transition to flash programming event." to the host. Then call the checksum judgment function to judge the program code in the reprogram area.

The checksum of this sample program uses "Program code size" and "Checksum data" that a program was added to one byte at a time. The checksum judgment function adds a program one byte at a time with the start address (H'0000 8000) in the reprogram area for the number of program data size. The calculation result is compared with the checksum judgment data calculated when received a data (Stored in the last 16-byte area of MasterPRG.text. Refer to 6.1.6 for the details). The program in the reprogram area will be executed when the calculation result matches the said data, and the one in the spare area will be executed if there is a difference.

6.1.4 Flash Reprogram Processing after Inputting INTP1 Interrupt

When the INTP1 interrupt (rising edge detection/ INTP1 switch push down on the board) is generated, moves to flash reprogram processing.

In the flash reprogram processing, the message "--> INTP1 detected!" is transmitted to the host to erase the reprogram area. Then the message "Send subroutine code to update program in Intel expanded hex format." is transmitted to the host to enter wait state for data reception from the host.

In the wait state for data reception, flag variables are used by polling to detect if the flash write is enabled or disabled. When receiving program file data for update with Intel expanded hex format from the host, the data receive processing (later described) is executed, and the data is stored into the write data store buffer (write buffer). When the write buffer becomes full, the buffer data will be written to the flash memory.

This sample program provides a double structured write buffer. Regarding "Storing write data during data receive processing" and "Writing to the flash memory", each processing should be executed by switching the write buffer to be used.

6.1.5 Data Receive Processing

After the V850E2/ML4 entered the wait state for data reception, CAN controller channel 0 (FCN0) receive processing completion interrupt is generated every time it receives communication data from the host. When the said interrupt is generated, the received data will be stored into the CAN receive data store buffer (receive buffer) in the order received. When the V850E2/ML4 receives the line feed code, it judges the data that has been stored in the receive buffer as a record data for a line and executes data receive processing described as follows to extract write data necessary for updating.

The data receive processing is described as follows referring to Figure 6.2 "Example of Data with Intel Expanded Hex Format". (The data shown in Figure 6.2 is color coded depending on its function.)

```
:04000005000013C81C
:02000040000FA
:20800000E0570584CA5EEFFF605F0484E0670583CC6EEFFF606F0483407640FF2E7F054609
:20802000CF86EFFF408E40FF71870546E0970580929E1000609F0480405681FF6A070082E5
:208040002B06FAFF0000406681FF6C5F4082206EFF3F606F00C44076FFFF0E7F66608F86C8
:1A8060000F00408EFFFF518766604096FFFFD2BF6660019A609FC4C57F00C0
:00000001FF
```

Figure 6.2 Example of Data with Intel Expanded Hex Format

- For the processing of each line, determine if the 1st character of the data in the receive buffer shows ":". If it shows ":", judge the 8th and 9th characters (red) as Intel expanded hex format. If the 1st character does not show ":", the record data become invalid, and returns to wait state for receive data. When the 8th character does not show "0", the record data also become invalid, and returns to wait state.

- The 8th and 9th characters (red) of the first line show "05". This "05" indicates the start linear address record which does not have a program data. When received the start linear address record, return to the wait state for receive data until the next entire record (line data) will be displayed.

- The 8th and 9th characters (red) of the second line show "04". This "04" indicates the extended linear address record which does not have a program data. When received the extended linear address record, return to the wait state for receive data until the next entire record will be displayed.

- When the entire 3rd line of the record is displayed, the line is determined as a "data record" because the 8th and 9th characters (red) show "00". The type of the record can be determined by the numbers from the start to 9th of each record with Intel expanded hex format.

- The 2th and 3rd characters (blue) of the record indicate the hex for 1 byte of the record size. The four characters from 4th to 8th (green) indicate the lower 2 bytes of the start data store address of the record.

- Regarding the 10th and later characters (orange) of the record, each two characters indicates 1 byte. In the data receive processing, the 10th and later characters (orange) is converted into binary data every 2 characters (call "text binary conversion processing"), store the 1 byte data after the conversion into the write buffer in the order converted. Add the one byte data for the checksum judgment (checksum data), and count the amounts of the data as a program code size. When repeated these processing before the last two characters (black) of the record, return to the wait state for receive data until the next entire record will be displayed.

- When the 8th and 9th (red) characters of the record data show "01", it means "end record" (the bottom line in Figure 6.2). When the end record is shown, terminate the data receive processing without storing receive data. However, if the data size in the write buffer is less than 16 bytes (unit of flash write) at this point, add H'FF to make the buffer size 16 bytes.

This sample program provides a double structured write buffer with 16-byte size. Every time the store data in a write buffer becomes full at 16 bytes, the store destination is switched to another write buffer during the data receive processing. When the said buffer becomes full, the buffer data is written to the flash memory during flash reprogram event processing. Writing to the flash memory is executed by polling waiting for receive data, not by an interrupt processing. When switching the buffer at full, set flag variables which indicate writability.

6.1.6 Processing after Data Reception/Reprogram

When the end record is determined during data receive processing and the write of flash memory for the receive data is terminated, the V850E2/ML4 leaves from the wait state for data reception in the flash reprogram event processing, and writes the data for checksum judgment calculated at the time of data reception (program code size and checksum data/ 2 bytes for each) the flash memory. In this sample program, the data for checksum judgment is stored the last 4 bytes of the reprogram area H'0000 8FF0 to H'0000 8FF3 (H'0000 8FF0 to H'0000 8FF1 for the program code size and H'0000 8FF2 to H'0000 8FF3 for the checksum data).

After writing the data for checksum judgment, a message is transmitted to the host and the V850E2/ML4 enters wait state for reset.

6.1.7 Communication Control Sequence

Figure 6.3 shows the Communication Control Sequence.

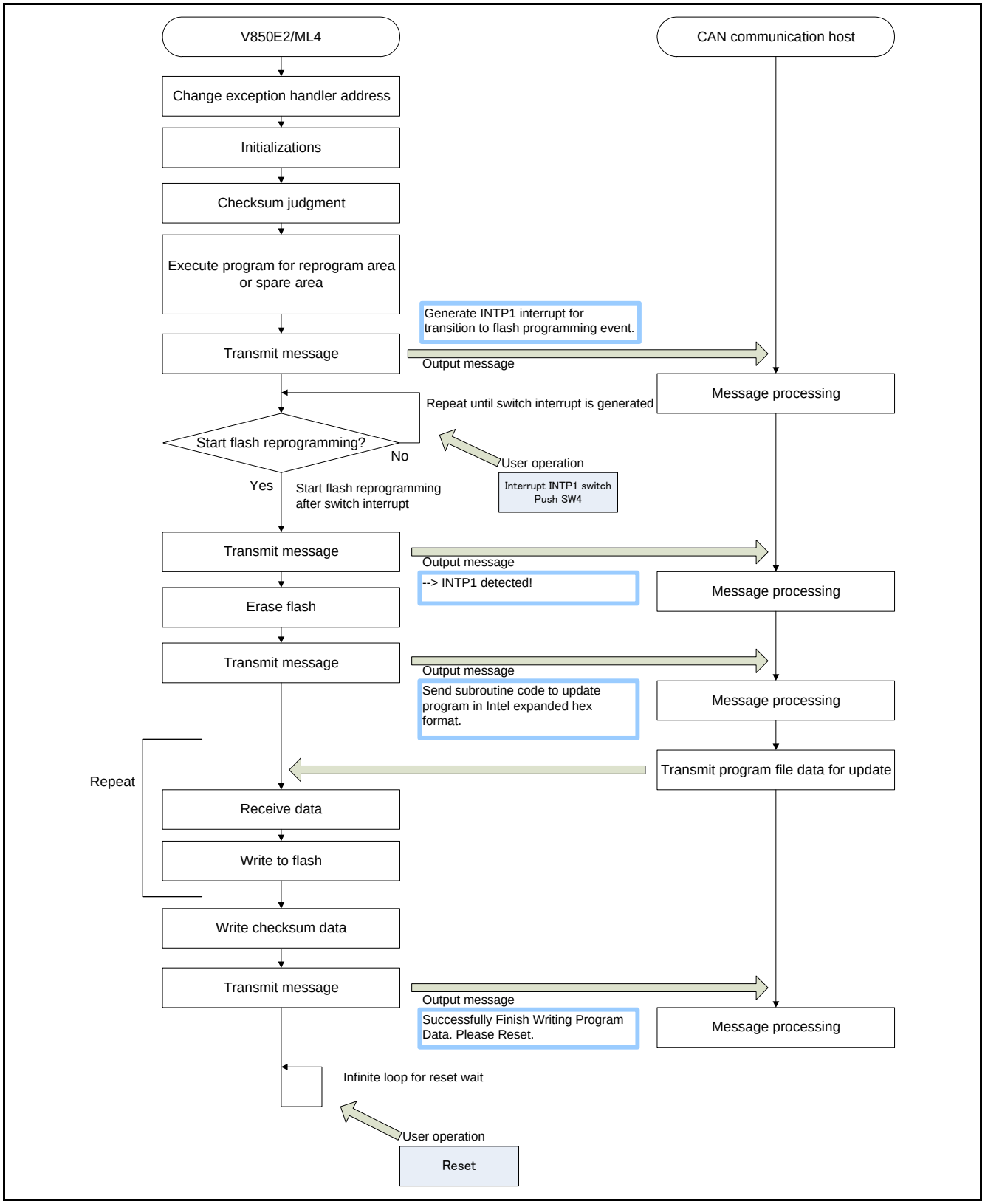


Figure 6.3 Communication Control Sequence

6.2 File Composition

Table 6.3 lists the Files Used in the Sample Code. Files generated by the integrated development environment are not included in this table.

Table 6.3 Files Used in the Sample Code

File Name	Outline	Remarks
main.c	Main processing	
intp1.c	INTP1 interrupt processing	
flash.c	Processing related to flash reprogram	
fcn0_can.c	Processing related to CAN controller	
taua0_led_sample.c	Sample program for update and LED flash port processing	
flash.h	Common header for flash memory reprogram processing	
r_typedefs.h	Fixed length integer type header	
FSL.h	FSL header file	
except_handler_ram.asm	Exception handler in RAM*	Jump to interrupt processing function from RAM
cstart.asm	Startup routine	Change stack size from standard startup routine, and add initialization of program area in RAM
libFSL_T05_REC_R32.lib	FSL library (32 register mode)	
v850e2ml4_flash_update_can.dir	Link directive setting file	
v850e2ml4_flash_update_can.fjp	Far jump calling function file	

[Note] * Defines the jump instruction from the interrupt handler address to the interrupt handler function to be allocated on the exception handler.

6.3 Constants

Table 6.4 and Table 6.5list the Constants Used in the Sample Code.

Table 6.4 Constants Used in the Sample Code

Constant Name	Setting Value	Description
RET_OK	0	Normal end
RET_ERR	-1	Error end
RET_ERR_FLASH_ACTIVATE	-1	Failure to start flash environment
RET_ERR_FLASH_MODECHECK	-2	Failure to check FLMD0 pin
RET_ERR_FLASH_ERASE	-3	Failure of erase processing
RET_ERR_FLASH_WRITE	-4	Failure of write processing
RET_ERR_FLASH_IVERIFY	-5	Failure of internal verification
RET_ERR_FLASH_DEACTIVATE	-6	Failure to terminate flash environment
RET_ERR_FLASH_FLMD0_HIGH	-7	Failure to set High level for FLMD0 pin
RET_ERR_FLASH_FLMD0_LOW	-8	Failure to set Low level for FLMD0 pin
RET_ERR_FLASH_CAN_RX_NUM	-9	Abnormal receive data size of CAN communication
RET_ERR_FLASH_HEX_LINESIZE	-10	Abnormal number of hex file line data
RET_ERR_FLASH_HEX_DATA	-11	Abnormal hex file program data
BLOCK_MASTER_PRG	8	Reprogram area block number
TOP_ADDR_MASTER_PRG	H'00008000	Reprogram area start address
SIZE_MASTER_PRG	H'1000	Reprogram area size (4KB)
SIZE_WRITE	16	Write specification size
TOP_ADDR_MASTER_PRG_CHKSUM	TOP_ADDR_MASTER_PRG + SIZE_MASTER_PRG - SIZE_WRITE	Start address of checksum area (H'00008FF0)
TOP_ADDR_EXT_HANDLER	H'FEDF E000	Transfer destination exception handler start address

Table 6.5 Constants Used in the Sample Code

Constant Name	Setting Value	Contents
FLASH_STATUS_FLMD0_HIGH	H'01	FLMD0 High setting completion state (pull-up valid)
FLASH_STATUS_FSL_ACTIVATE	H'02	FSL start status
HEXDATA_POS_RECMARK	0	Record mark position of hex data
HEXDATA_POS_BYTE_NUM	1	Position for number of bytes of hex data
HEXDATA_POS_RECTYPE_UPPER	7	Upper digit position of hex data record type
HEXDATA_POS_RECTYPE_LOWER	8	Lower digit position of hex data record type
HEXDATA_POS_CODE_TOP	9	Start position of hex data code
SIZE_BUF_RX_DATA	525	Buffer size to store receive data (total of the followings) — Record mark: 1 character — Number of bytes: 2 characters — Location address: 4 characters — Record type: 2 characters — Code: 512 characters (max) — Checksum: 2 characters — Return (\r) + New line (\n): 2 characters
FCN0_CAN_SIZE_DATA_MAX	8	Maximum transmit/receive data length
FCN0_CAN_NUM_MB	64	Number of message buffers
FCN0_CAN_GAP_MB_ADDR	H'40	Address interval of message buffers
FCN0_CAN_ADDR_M0_STRB	H'FF481024	FCN0M0STRB register address
FCN0_CAN_ADDR_M0_CTL	H'FF489038	FCN0M0CTL register address
FCN0_CAN_ADDR_M0_DAT	H'FF481000	FCN0M0DAT register address
FCN0_CAN_ID_SEND_MSG	0	Message ID used in the fcn0_can_tx_msg function
PORT_BIT_P1_4	H'0010	Bit position of port function setting P1_4
PORT_BIT_P2_3	H'0008	Bit position of port function setting P2_3
PORT_BIT_P2_14	H'4000	Bit position of port function setting P2_14
PORT_BIT_P2_15	H'8000	Bit position of port function setting P2_15

6.4 Variables

Table 6.6 lists the Global Variables.

Table 6.6 Global Variables

Type	Variable Name	Contents	Function Used
uint8_t	g_flag_start_flash_reprog	Flash reprogram start flag	main, intp1_isr
fsl_status_t	g_error_fsl_status	FSL error save	main, flash_activate, flash_modecheck, flash_erase, flash_write, flash_iverify
uint32_t	g_addr_write_error	Write error address	main, flash_write
uint8_t	g_flag_w_data_buf0_full	Write buffer 0 full flag	flash_reprogram, flash_store_can_data
uint8_t	g_flag_w_data_buf1_full	Write buffer 1 full flag	flash_reprogram, flash_store_can_data
uint8_t	g_status_end_record	End record receive flag	flash_reprogram, flash_store_can_data
uint16_t	g_chksm_size	Program code size of write data	flash_reprogram, flash_store_can_data
uint16_t	g_chksm_data	Checksum data of write data	flash_reprogram, flash_store_can_data
uint8_t	g_buf_write_data0 [SIZE_WRITE]	Write data store buffer 0	flash_reprogram, flash_store_can_data
uint32_t	g_cnt_store_buf_w_data0	Data counts of write data store buffer 0	flash_reprogram, flash_store_can_data
uint8_t	g_buf_write_data1 [SIZE_WRITE]	Write data store buffer 1	flash_reprogram, flash_store_can_data
uint32_t	g_cnt_store_buf_w_data1	Data counts of write data store buffer 1	flash_reprogram, flash_store_can_data
uint32_t	g_index_rx_data	Receive data store location index	flash_reprogram, flash_store_can_data
uint8_t	g_buf_rx_data [SIZE_BUF_RX_DATA]	Receive data store buffer	flash_store_can_data
int8_t	g_status_store_error	Error flag	flash_reprogram, flash_store_can_data
uint8_t	g_flag_flash_status	Flash environment status	flash_init, flash_activate, flash_end
char	g_msg_sendcode[]	Program transmission request message	flash_reprogram

6.5 Functions

Table 6.7 lists the Functions.

Table 6.7 Functions

Function Name	Outline
main	Main processing
except_handler_addr_set	Setting for exception handler base address
check_sum_check	Checksum judgment for reprogram area
intp1_init	Initialization of INTP1 interrupt
intp1_isr	Interrupt processing for INTP1
flash_reprogram	Flash reprogram processing
flash_init	Initialization of flash environment
flash_activate	Start processing for flash environment
flash_modecheck	Checking processing for FLMD0 pin using FSL
flash_erase	Erase processing for specified block
flash_write	Write processing from specified address
flash_iverify	Internal verification of specified block
flash_end	Termination processing for flash environment
flash_set_flmd0	Setting for LFMD0 pin level
flash_store_can_data	Store processing for receive data conversion
hex2bin	Text binary conversion processing
taua0_led_sample	TAUA0 initialization for LED flash with constant period (sample function in reprogram area)
taua0_led_spare	TAUA0 initialization for LED flash with constant period (sample function in spare area)
taua0_i0_interval_timer_isr *	Interrupt processing for TAUA0 interval timer
fcn0_can_init	Initialization of CAN controller channel 0 (FCN0)
fcn0_can_port_init	Initialization of FCN0 port
fcn0_can_mb_init	Initialization of FCN0 message buffer
fcn0_can_tx_msg	FCN0 message transmit processing
fcn0_can_tx	FCN0 transmit processing
fcn0_can_rx	FCN0 receive processing
fcn0_can_rx_isr	Interrupt processing for FCN0 reception completion
fcn0_can_error_isr	Exception handling for FCN0 error interrupt

[Notes] * To set the store processing for received program data by CAN communication above the LED flash processing, the interrupt handler function taua0_ch0_interval_timer_isr enables multiple interrupts. TAUA0 interval timer interrupt is set to a lower priority than FCN0 reception completion interrupt.

6.6 Function Specifications

The following tables list the sample code function specifications.

main	
Outline	Main processing
Header	
Declaration	void main (void)
Description	After initializing the variables, exception handler address, INTP1 interrupt, and CAN controller, executes the program located in the reprogram area or spare area according to the result of checksum judgment. After enabling interrupt and outputting INTP1 interrupt request message, executes flash reprogram processing when the INTP1 interrupt is generated. Outputs a reset request message if the reprogram is successful or an error message if it is failed, and enters into the infinite loop.
Arguments	None
Return Value	None

except_handler_addr_set	
Outline	Switching of exception handler base address
Header	
Declaration	except_handler_addr_set (uint32_t base_addr)
Description	After setting the specified values for the argument to the SW_BASE register, sets 1 to the SET bit for SW_CTL register, and transmits the contents of the SW_BASE register to the exception handler base address register (EH_BASE).
Arguments	uint32_t base_addr : Setting value of exception handler base address (The low 12-bit should be 0.)
Return Value	0 (RET_OK) : Normal end -1 (RET_ERR) : Argument error (The low 12-bit is not 0.)

check_sum_check	
Outline	Checksum judgment for reprogram area
Header	
Declaration	int32_t check_sum_check (void)
Description	Based on the program code size or checksum data stored in the last 4 bytes (H'0000 8FF0 to H'0000 8FF3) of reprogram area, calculates sum value from the start address (H'0000 8000) of reprogram area to judge the consistency with the checksum data.
Arguments	None
Return Value	0 (RET_OK) : Checksum matched -1 (RET_ERR) : Checksum unmatched

intp1_init	
Outline	Initialization of INTP1 interrupt
Header	
Declaration	void intp1_init (void)
Description	Initializes INTP1 interrupt. After setting P2_3 pin function to INTP1 input, sets the interrupt request to be detected at the falling edge for input using interrupt controller. Then sets INTP1 interrupt priority level.
Arguments	None
Return Value	None

intp1_isr	
Outline	INTP1 interrupt processing
Header	
Declaration	void intp1_isr (void)
Description	Sets a flag indicating that INTP1 interrupt was generated.
Arguments	None
Return Value	None

flash_reprogram	
Outline	Flash reprogram processing
Header	flash.h
Declaration	int32_t flash_reprogram (void)
Description	First, executes flash environment initialization, flash environment start processing, FLMD0 pin checking processing, and reprogram block erase processing. Next, transmits a program transmit request message and enters into the loop to execute program receive wait and flash write. When the program is received to the last and the writing is terminated, executes flash reprogram termination processing by writing checksum data.
Arguments	None
Return Value	0 (RET_OK) : Normal end -1 (RET_ERR_FLASH_ACTIVATE) : Failure to start flash environment. (RET_ERR_FLASH_MODECHECK) : Failure to check FLMD0 pin -3 (RET_ERR_FLASH_ERASE) : Failure of erase processing -4 (RET_ERR_FLASH_WRITE) : Failure of write processing -5 (RET_ERR_FLASH_IVERIFY) : Failure of internal verification -6 (RET_ERR_FLASH_DEACTIVATE) : Failure to terminate flash environment -7 (RET_ERR_FLASH_FLMD0_HIGH) : Failure to set FLMD0 pin to High level -8 (RET_ERR_FLASH_FLMD0_LOW) : Failure to set FLMD0 pin to Low level -9 (RET_ERR_FLASH_CAN_RX_NUM) : Abnormal receive data size in CAN communication -10 (RET_ERR_FLASH_HEX_LINESIZE) : Abnormal number of hex file line data -11 (RET_ERR_FLASH_HEX_DATA) : Abnormal hex file program data

flash_init		
Outline	Initialization of flash environment	
Header		
Declaration	int32_t flash_init (void)	
Description	After executing FLMD0 pin level setting function and setting FLMD0 pin to High level, the FSL_Init function is executed to initialize the self library. When the flash_set_flmd0 function becomes an error, returns RET_ERR_FLASH_FLMD0_HIGH.	
Arguments	None	
Return Value	0 (RET_OK)	: Normal end
	-7 (RET_ERR_FLASH_FLMD0_HIGH)	: Failure to set FLMD0 pin to High level

flash_activate		
Outline	Start processing for flash environment	
Header		
Declaration	int32_t flash_activate (void)	
Description	Starts flash environment by calling the FSL_FlashEnv_Activate function. In case of normal end, sets a bit to show the global variable g_flag_flash_status that the flash environment has been started and returns RET_OK to terminate. If the FSL_FlashEnv_Activate function returns a value other than FSL_OK, stores the return value in the global variable g_error_fsl_status and returns RET_ERR_FLASH_ACTIVATE to terminate.	
Arguments	None	
Return Value	0 (RET_OK)	: Normal end
	-1 (RET_ERR_FLASH_ACTIVATE)	: Failure to start flash environment

flash_modecheck		
Outline	Checking processing for FLMD0 pin using FSL	
Header		
Declaration	int32_t flash_modecheck (void)	
Description	Calls the FSL_ModeCheck function to check the FLMD0 pin. In case of normal end, returns RET_OK to terminate. When the FSL_ModeChec function returns other than FSL_OK, stores the return value in the global variable g_error_fsl_status and returns RET_ERR_FLASH_MODECHECK to terminate.	
Arguments	None	
Return Value	0 (RET_OK)	: Normal end
	-2 (RET_ERR_FLASH_MODECHECK)	: Failure to check FLMD0 pin

flash_erase	
Outline	Erase processing for specified block
Header	
Declaration	int32_t flash_erase (uint32_t start_block, uint32_t end_block)
Description	Erases the block by calling the FSL_Erase function according to the specified arguments. After executed the FSL_Erase function, calls the FSL_StatusCheck function and waits until the erase processing is completed. If the FSL_Erase function or the FSL_StatusCheck function returns an error value, stores the return value in the global variable g_error_fsl_status and returns RET_ERR_FLASH_ERASE to terminate.
Arguments	uint32_t start_block : Start block number of the range to be erased uint32_t end_block : End block number of the range to be erased
Return Value	0 (RET_OK) : Normal end -3 (RET_ERR_FLASH_ERASE) : Failure of erase processing

flash_write	
Outline	Write processing from specified address
Header	
Declaration	int32_t flash_write (uint8_t * src_data_addr, uint32_t dst_write_addr, uint32_t length)
Description	Writes to the flash by calling the FSL_Write function according to the specified arguments. After executed the FSL_Write function, calls the FSL_StatusCheck function and waits until the write processing is completed. When the FSL_Write function or the FSL_StatusCheck function returns an error value, stores the return value in the global variable g_error_fsl_status and returns RET_ERR_FLASH_WRITE to terminate.
Arguments	uint8_t * src_data_addr : Start address of write data (outside the on-chip ROM) uint32_t dst_write_addr : Store destination address of write data (4-word boundary) uint32_t length : Write data length
Return Value	0 (RET_OK) : Normal end -4 (RET_ERR_FLASH_WRITE) : Failure to write

flash_iverify	
Outline	Internal verification of specified block
Header	
Declaration	int32_t flash_iverify (uint32_t start_block, uint32_t end_block)
Description	Executes the internal verification of the specified block by calling the FSL_IVerify function according to the arguments. After executed the FSL_IVerify function, calls the FSL_StatusCheck function and waits until the internal verification is completed. When the FSL_IVerify function or the FSL_StatusCheck function returns an error value, stores the return value in the global variable g_error_fsl_status and returns RET_ERR_FLASH_IVERIFY to terminate.
Arguments	uint32_t start_block : Start block number of the range subject for verification uint32_t end_block : End block number of the range subject for verification
Return Value	0 (RET_OK) : Normal end -5 (RET_ERR_FLASH_IVERIFY) : Failure of internal verification

flash_end	
Outline	Termination processing for flash reprogram
Header	
Declaration	int32_t flash_end (void)
Description	After terminated the flash environment by calling the FSL_FlashEnv_Deactivate function, calls the flash_set_flmd0 function and sets the FLMD0 pin to Low level. When the FSL_FlashEnv_Deactivate function returns an error value, RET_ERR_FLASH_DEACTIVATE is returned. When the flash_set_flmd0 function returns a value other than 0, RET_ERR_FLASH_FLMD0_LOW is returned to terminate.
Arguments	None
Return Value	0 (RET_OK) : Normal end -6 (RET_ERR_FLASH_DEACTIVATE) : Failure to terminate flash environment -8 (RET_ERR_FLASH_FLMD0_LOW) : Failure to set FLMD0 pin to High level

flash_set_flmd0	
Outline	Setting for FLMD0 pin level
Header	
Declaration	int32_t flash_set_flmd0 (uint8_t level)
Description	Sets FLMD control register to switch the FLMD0 pull-up/pull-down control register. According to the reprogram sequence to protect register, after substituted H'A5 in the FLMD protect command register, substitutes the value specified by the argument and the inverted value in the FLMD control register. Then substitutes again the value specified by the argument in the FLMD control register. Ensures that the register values have been changed to terminate.
Arguments	uint8_t level uint8_t level
Return Value	0 (RET_OK) : Normal end -1 (RET_ERR) : Error in write operation to FLMDCNT register

flash_store_can_data	
Outline	Store processing for receive data conversion
Header	flash.h
Declaration	void flash_store_can_data (void)
Description	Converts the hex data to binary data with each line and stores it in the buffer. If the hex data for a line is a data record, converts the data into binary form and stores it until the buffer becomes full. If the hex data for a line is an end record, sets the flag indicating that the reception has been terminated by filling the remaining of the buffer with H'FF.
Arguments	None
Return Value	None

hex2bin	
Outline	Text binary conversion processing
Header	
Declaration	int32_t hex2bin(uint8_t upper, uint8_t lower)
Description	Converts the text data (2 characters) to binary data with 1 byte. When the data given to the arguments is the text data with "0 to 9" or "A to F", the data will be considered as valid data and converted to binary data with H'0 to H'F. After shifting the conversion result of the first argument (the upper) to left by 4 bits and implementing the OR operation with the conversion result of the second argument (the lower), returns the data as binary data with 1 byte.
Arguments	uint8_t upper : Text data for upper 4-bit uint8_t lower : Text data for lower 4-bit
Return Value	0 to 255 : Binary data with 1 byte -1 (RET_ERR) : Input data error

taua0_led_sample	
Outline	TAUA0 initialization for LED flash with constant period (sample function in reprogram area)
Header	
Declaration	void taua0_led_sample (void)
Description	Sets the ports connected to the LEDs to output in order to flash the LEDs. Sets TAUA0 to the interval timer which generates interrupts with constant period.
Arguments	None
Return Value	None

taua0_led_spare	
Outline	TAUA0 initialization for LED flash with constant period (sample function in spare area)
Header	
Declaration	void taua0_led_spare (void)
Description	Sets the ports connected to the LEDs to output in order to flash the LEDs Sets TAUA0 to the interval timer which generates interrupts with constant period.
Arguments	None
Return Value	None

taua0_i0_interval_timer_isr	
Outline	Interrupt processing for TAUA0 interval timer
Header	
Declaration	void taua0_i0_interval_timer_isr (void)
Description	Inverts the output of P1_4 to flash LEDs.
Arguments	None
Return Value	None

fcn0_init	
Outline	Initialization of CAN controller channel 0 (FCN0)
Header	
Declaration	int32_t fcn0_init (void)
Description	After initializing FCN0 port (the fcn0_port_init function), sets FCN0 system clock and communication baud rate to enable module operation for related interrupt enable. After initializing the message buffer (the fcn0_mb_init function), transfers FCN0 to normal operating mode.
Arguments	None
Return Value	0 (RET_OK) : Normal end -1 (RET_ERR) : Register error

fcn0_port_init	
Outline	Initialization of FCN0 port
Header	
Declaration	void fcn0_port_init (void)
Description	Sets the port to use P2_14 for reception and P2_15 for transmission in CAN communication.
Arguments	None
Return Value	None

fcn0_mb_init	
Outline	Initialization of FCN0 message buffer
Header	
Declaration	void fcn0_mb_init (void)
Description	After executing minimum initialization for all of the message buffers, sets the message buffer 0 for reception and the message buffer 1 for transmission.
Arguments	None
Return Value	None

fcn0_can_tx_msg	
Outline	FCN0 message transmit processing
Header	
Declaration	void fcn0_can_tx_msg (char * msg)
Description	Executes FCN0 transmit processing (the fcn0_can_tx function) for the number of times required and transmits the message specified by the argument. The message including end character is transmitted until "\0" appears at the end of the character string. The transmit data ID is set to 0 for transmit processing.
Arguments	char * msg : Transmit message character string
Return Value	None

fcn0_can_tx	
Outline	FCN0 transmit processing
Header	
Declaration	int32_t fcn0_can_tx (int8_t length, uint8_t * send_data, uint32_t id)
Description	Transmits the data specified by the argument send_data for the bytes specified by the argument length as CAN communication data using FCN0 message buffer1. Waits until the FCN0M1TRQF and the FCN0M1RDYF become transmittable. Sets the transmit data to the message buffer with specified size and transmits the ID specified by the argument id.
Arguments	int8_t length : Transmit data length uint8_t * send_data : Start address of transmit data uint32_t id : ID set to transmit data
Return Value	0 (RET_OK) : Normal end -1 (RET_ERR) : Argument error

fcn0_can_receive	
Outline	FCN0 receive processing
Header	
Declaration	int32_t fcn0_can_receive (int8_t * length, uint8_t * recv_data, uint32_t * id)
Description	While FCN0 message buffer 0 receives data, the receive data is stored in the address area specified by the argument recv_data. When storing the data in the message buffer, the receive data length and the ID will be stored in the argument length and the argument id respectively. After the data is stored, if the store or update bit for message buffer is set, starts over the data store processing.
Arguments	int8_t * length : Receive data length (When an error in receive data length occurs, it will be 8 which indicates the data counts actually stored.) uint8_t * recv_data : Start address of receive data store area uint32_t * id : ID set to the receive data
Return Value	0 to 8 (0 to FCN_MAX_DATA_LENGTH) : Normal receive data length Above 9 (>FCN_MAX_DATA_LENGTH) : Error in receive data length

fcn0_rx_isr	
Outline	FCN0 reception completion interrupt processing
Header	
Declaration	void fcn0_rx_isr (void)
Description	Executes processing to store the receive data conversion (flash_store_can_data function).
Arguments	None
Return Value	None

fcn0_error_isr	
Outline	FCN0 error interrupt processing
Header	
Declaration	void fcn0_error_isr (void)
Description	Execute force recovery in case of bus off.
Arguments	None
Return Value	None

6.7 Flowcharts

6.7.1 Startup Routine Processing

Figure 6.4 shows the Startup Routine Processing.

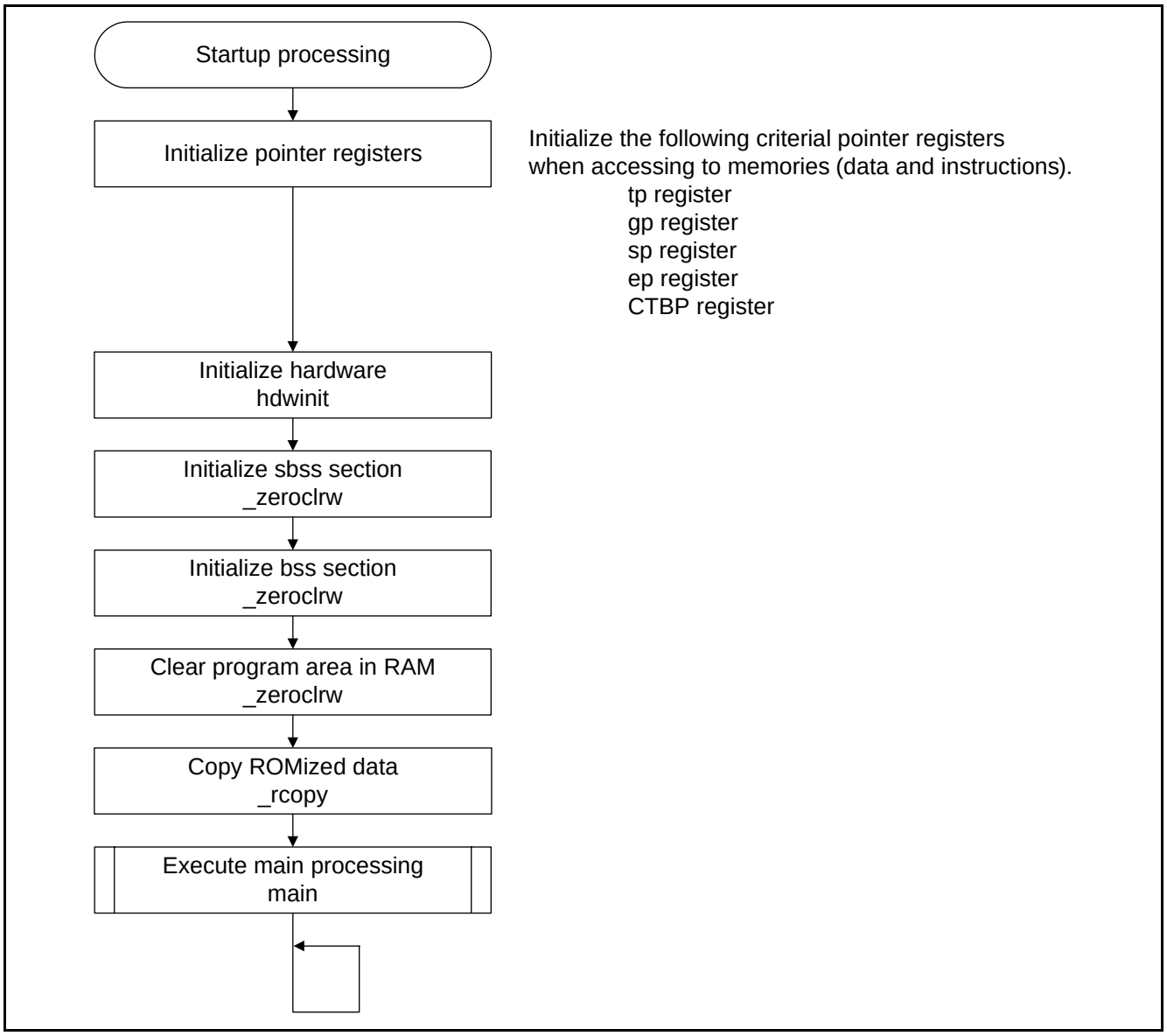


Figure 6.4 Startup Routine Processing

6.7.2 Main Processing

Figure 6.5 shows the Main Processing.

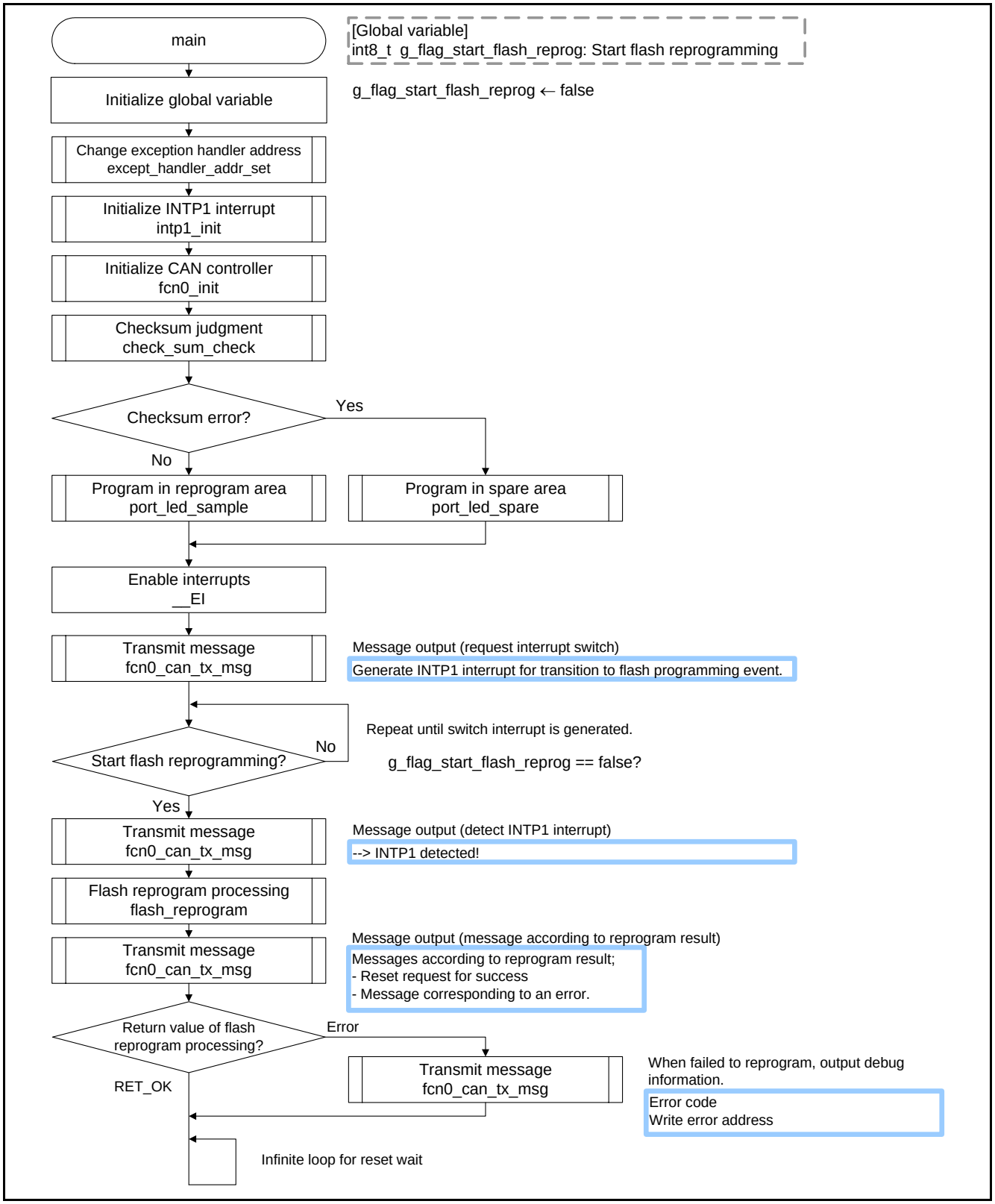
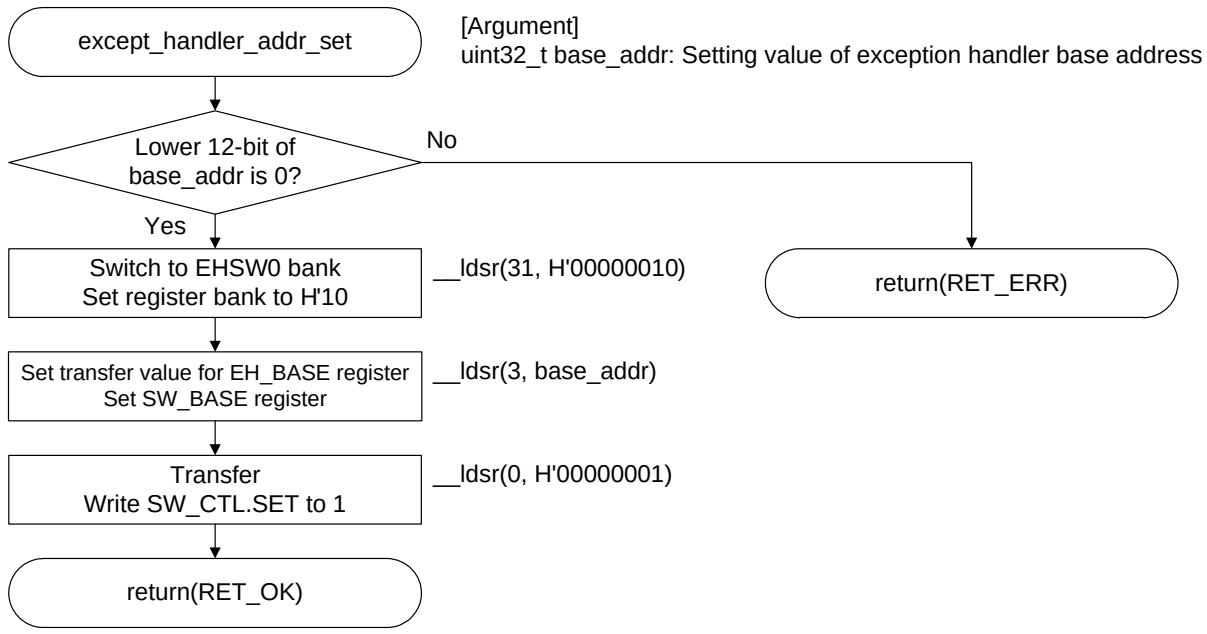


Figure 6.5 Main Processing

6.7.3 Switching Processing for Exception Handler Address

Figure 6.6 shows the Switching Processing for Exception Handler Address.



[Note] When switching the exception handler address, the period from the startup of switching procedure to the termination thereof must be free from exceptions, or any problem in case that an exception was generated. This sample program has a program that operates properly for any exception handler address of before/after switching.
If the program cannot be located at the time of application, settings such as interrupt disable will be required.

Figure 6.6 Switching Processing for Exception Handler Address

6.7.4 Checksum Judgment for Reprogram Area

Figure 6.7 shows the Checksum Judgment for Reprogram Area.

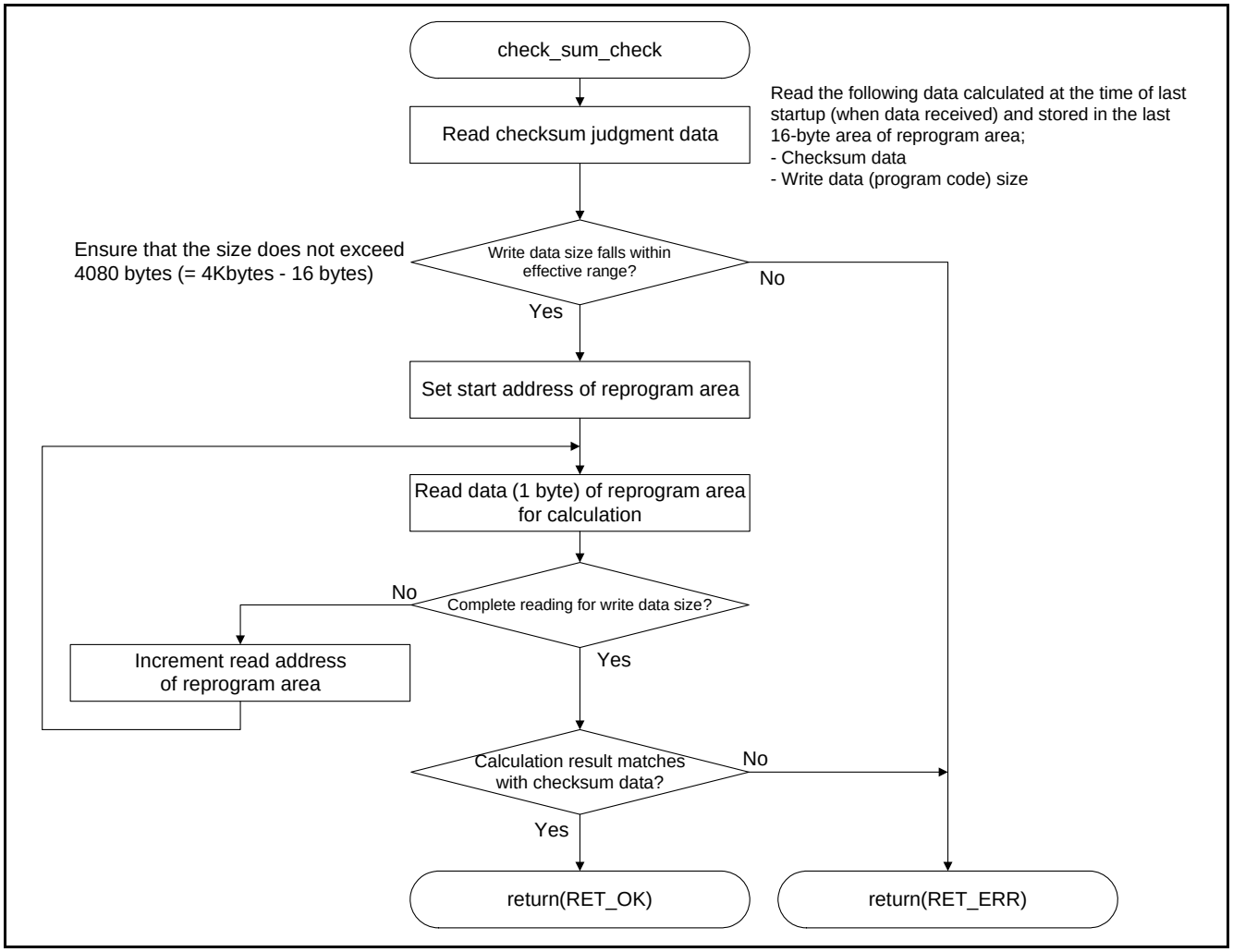


Figure 6.7 Checksum Judgment for Reprogram Area

6.7.5 Initialization of INTP1 Interrupt

Figure 6.8 shows the Initialization of INTP1 Interrupt.

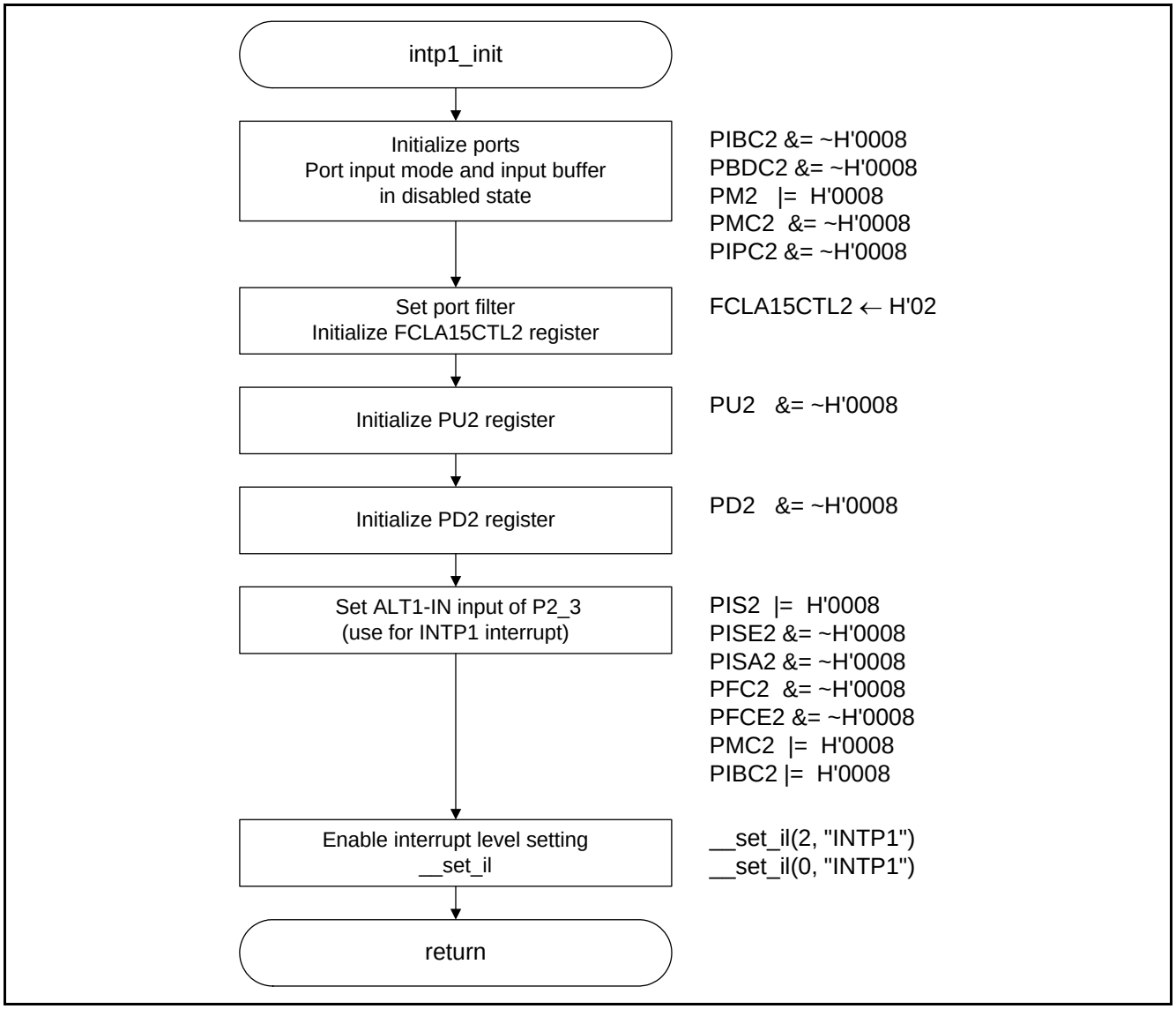
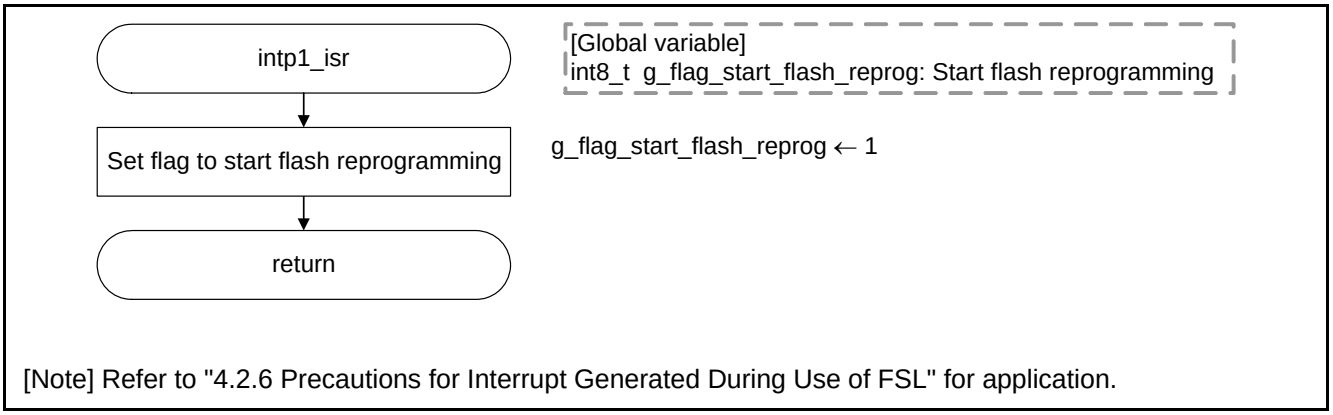


Figure 6.8 Initialization of INTP1 Interrupt

6.7.6 INTP1 Interrupt Processing

Figure 6.9 shows the INTP1 Interrupt Processing.



[Note] Refer to "4.2.6 Precautions for Interrupt Generated During Use of FSL" for application.

Figure 6.9 INTP1 Interrupt Processing

6.7.7 Flash Reprogram Processing

Figure 6.10 and Figure 6.11 show the Flash Reprogram Processing.

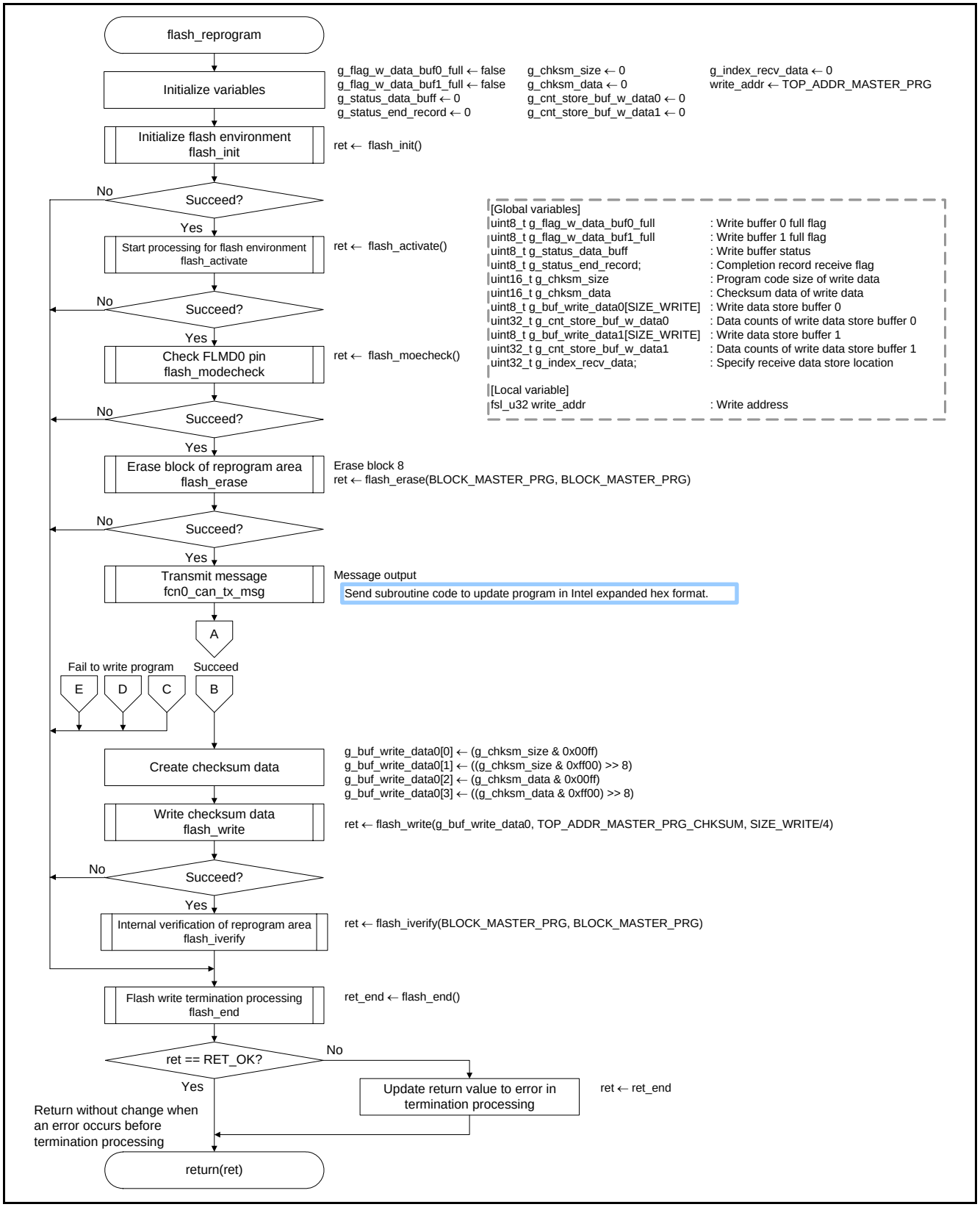


Figure 6.10 Flash Reprogram Processing (1/2)

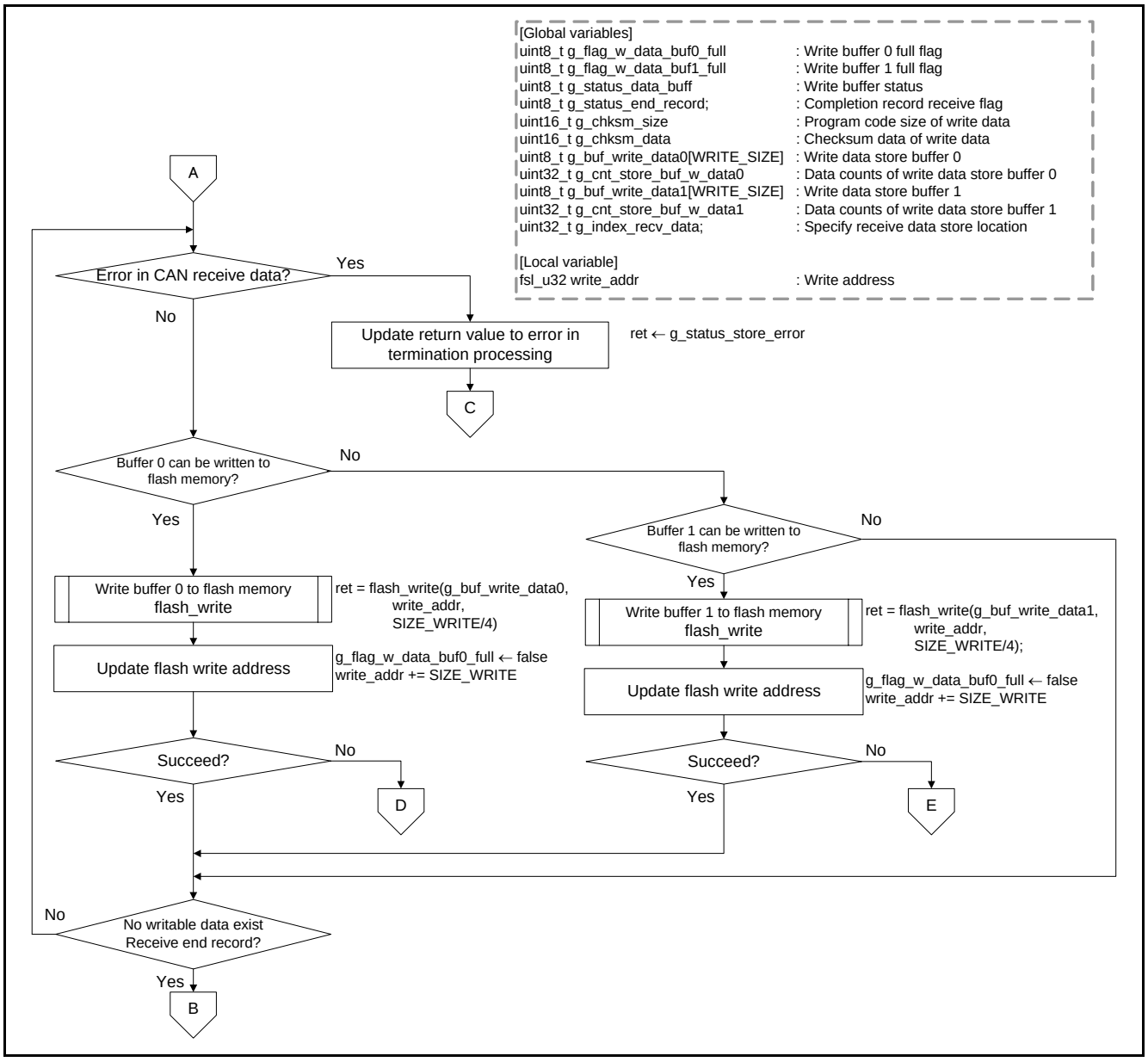


Figure 6.11 Flash Reprogram Processing (2/2)

6.7.8 Initialization of Flash Environment

Figure 6.12 shows the Initialization of Flash Environment.

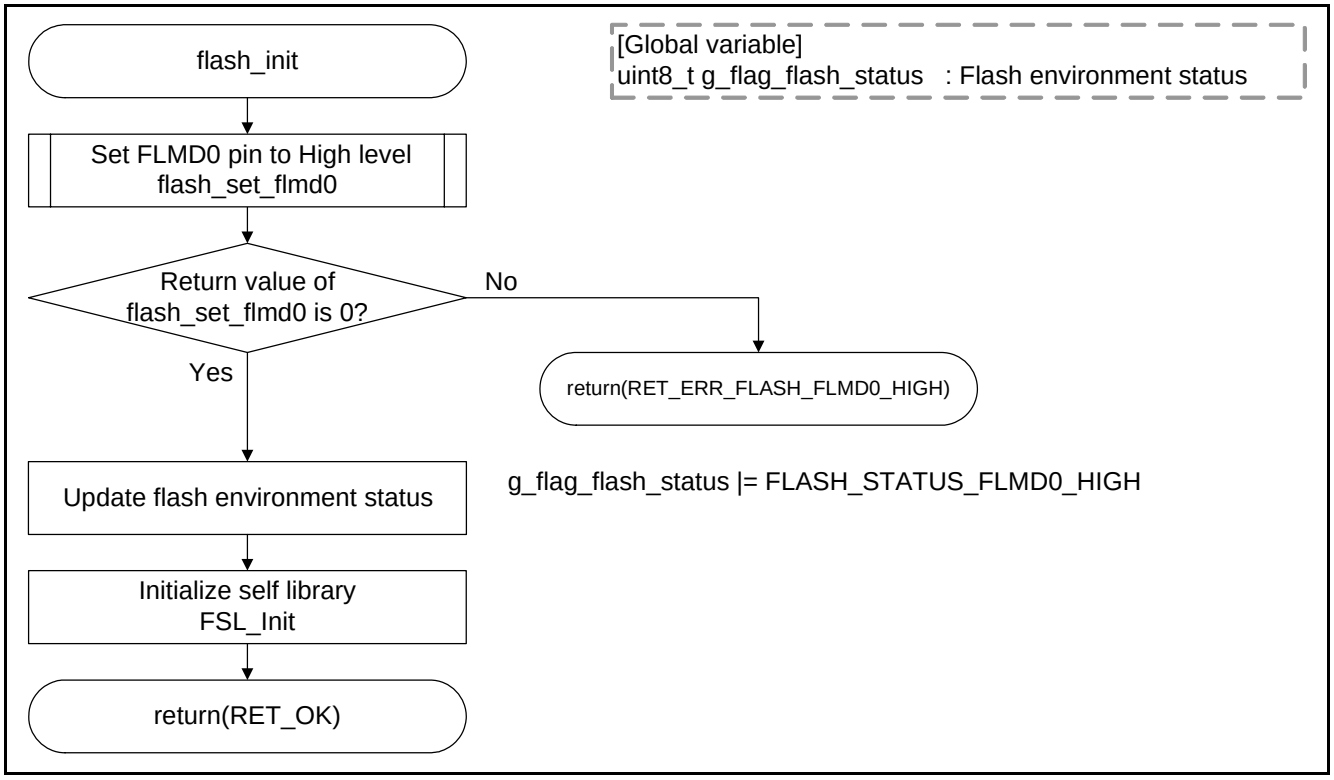


Figure 6.12 Initialization of Flash Environment

6.7.9 Start Processing for Flash Environment

Figure 6.13 Start Processing for Flash Environment

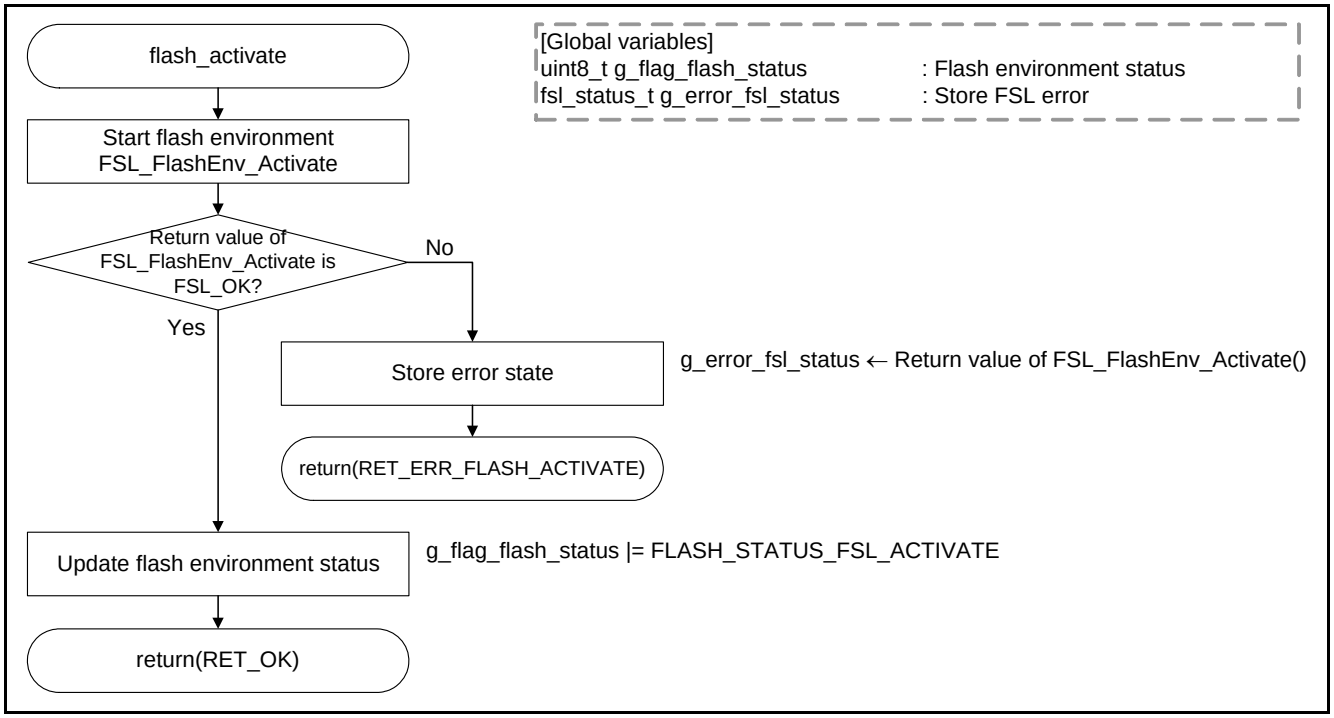


Figure 6.13 Start Processing for Flash Environment

6.7.10 Checking Processing for FLMD0 Pin using FSL

Figure 6.14 Checking Processing for FLMD0 Pin using FSL

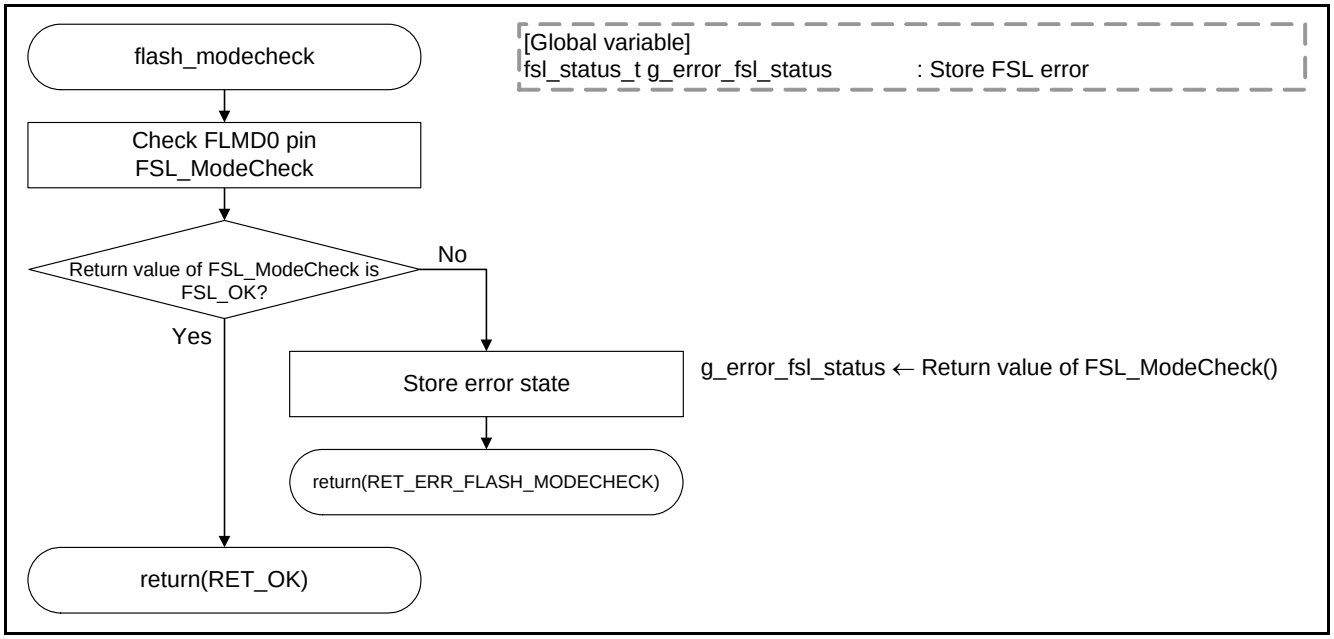


Figure 6.14 Checking Processing for FLMD0 Pin using FSL

6.7.11 Erase Processing for Specified Block

Figure 6.15 Erase Processing for Specified Block

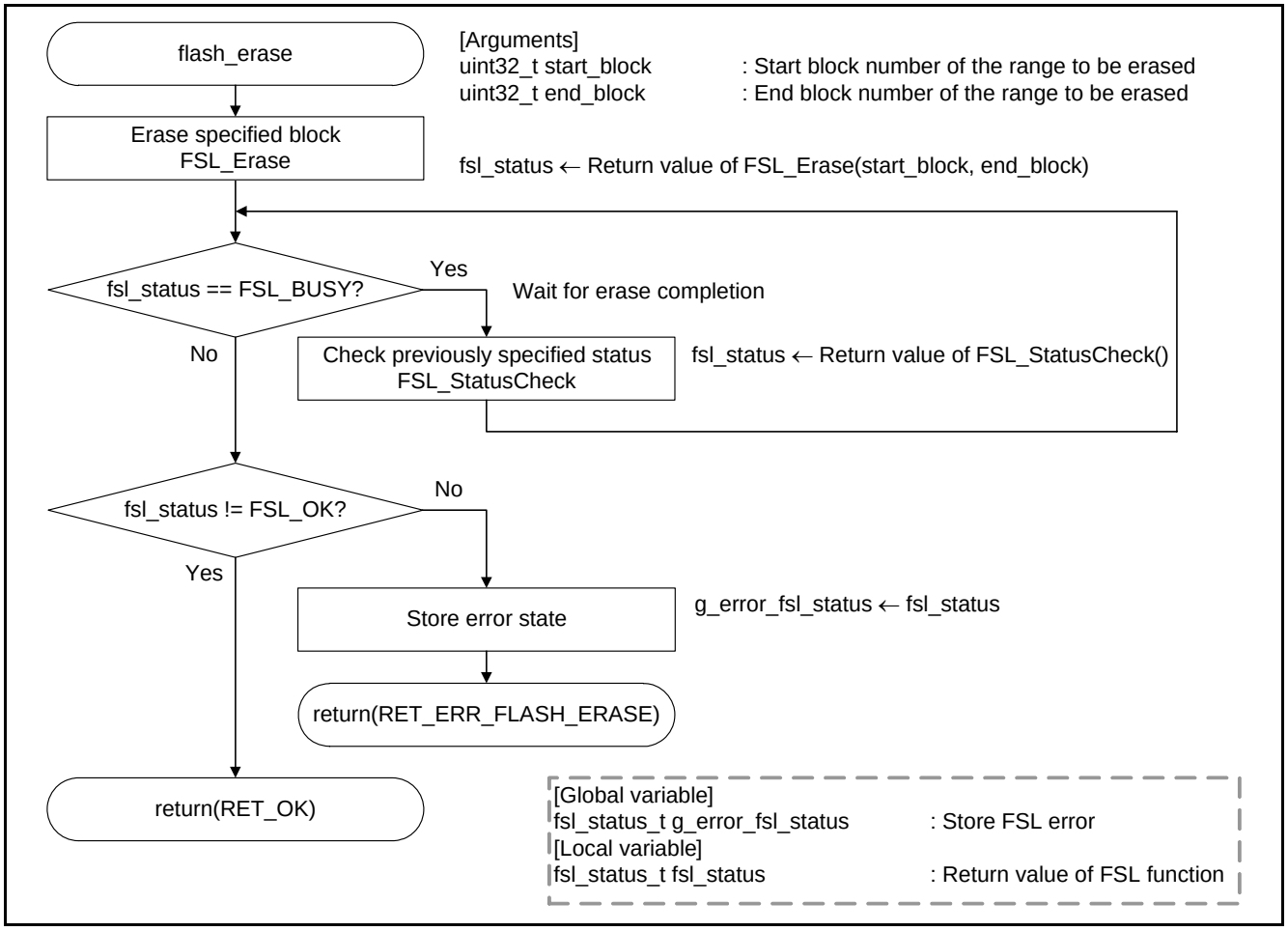


Figure 6.15 Erase Processing for Specified Block

6.7.12 Write Processing from Specified Address

Figure 6.16 Write Processing from Specified Address

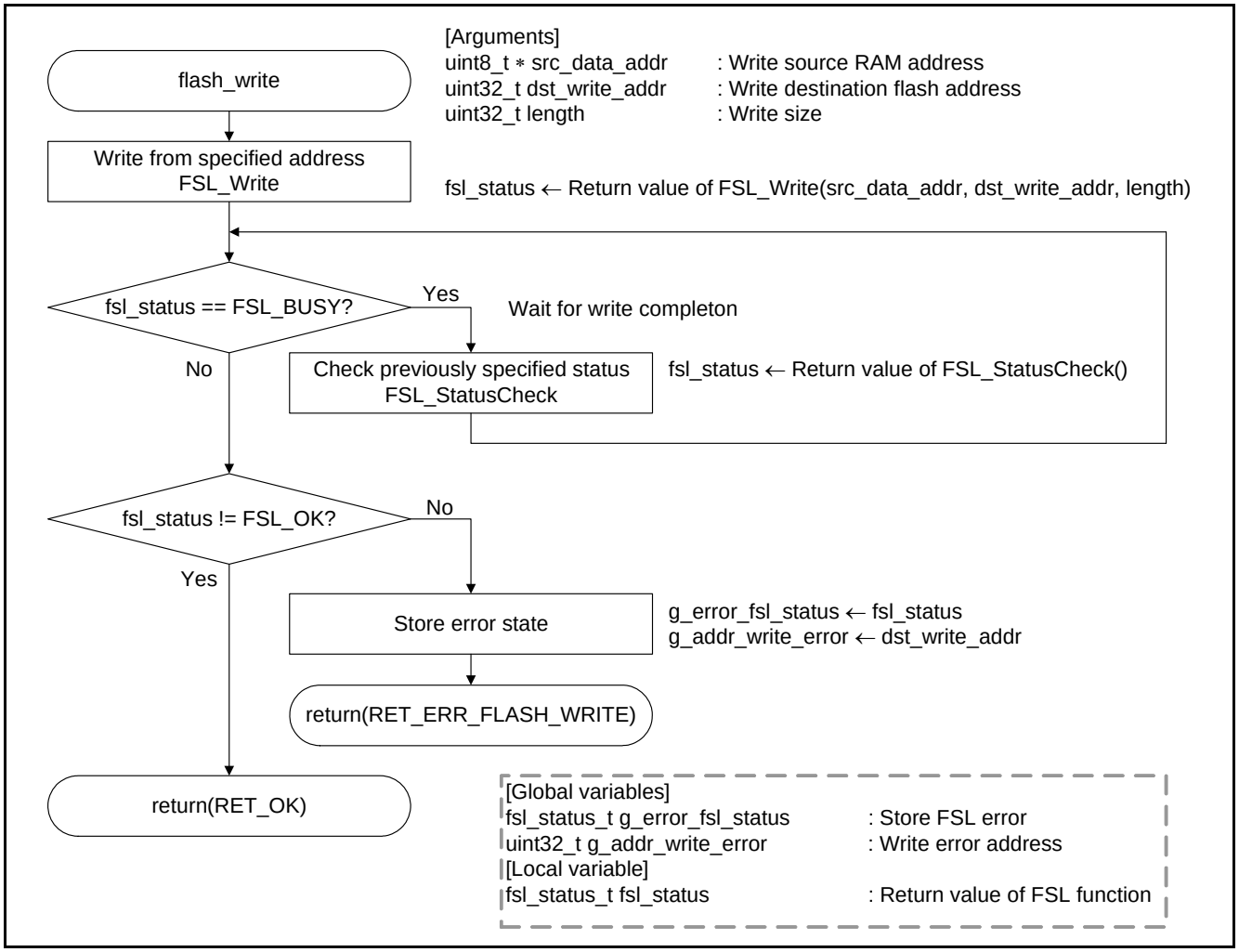


Figure 6.16 Write Processing from Specified Address

6.7.13 Internal Verification of Specified Block

Figure 6.17 shows the Internal Verification of Specified Block.

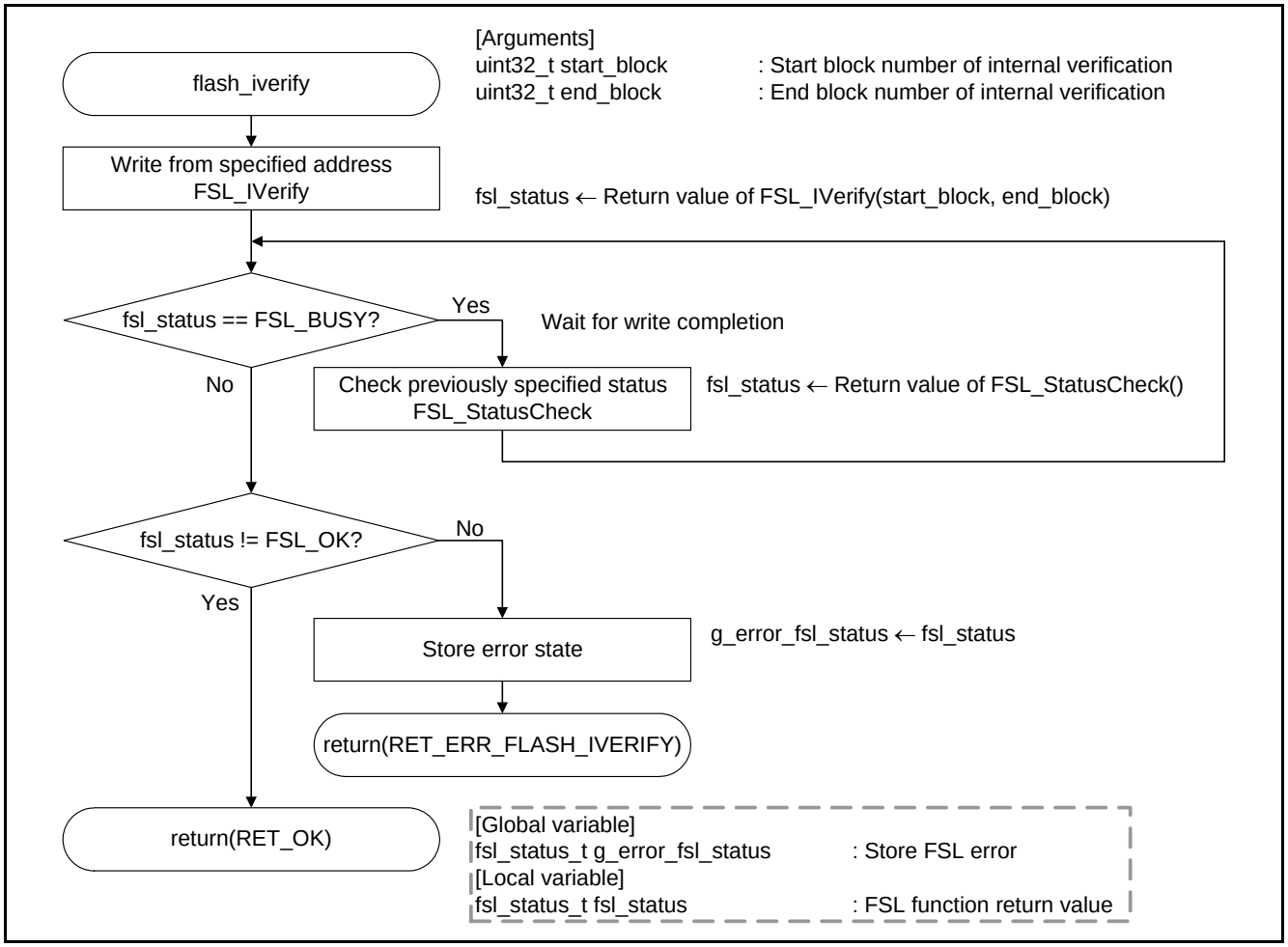


Figure 6.17 Internal Verification of Specified Block

6.7.14 Termination Processing for Flash Environment

Figure 6.18 shows the Termination Processing for Flash Environment.

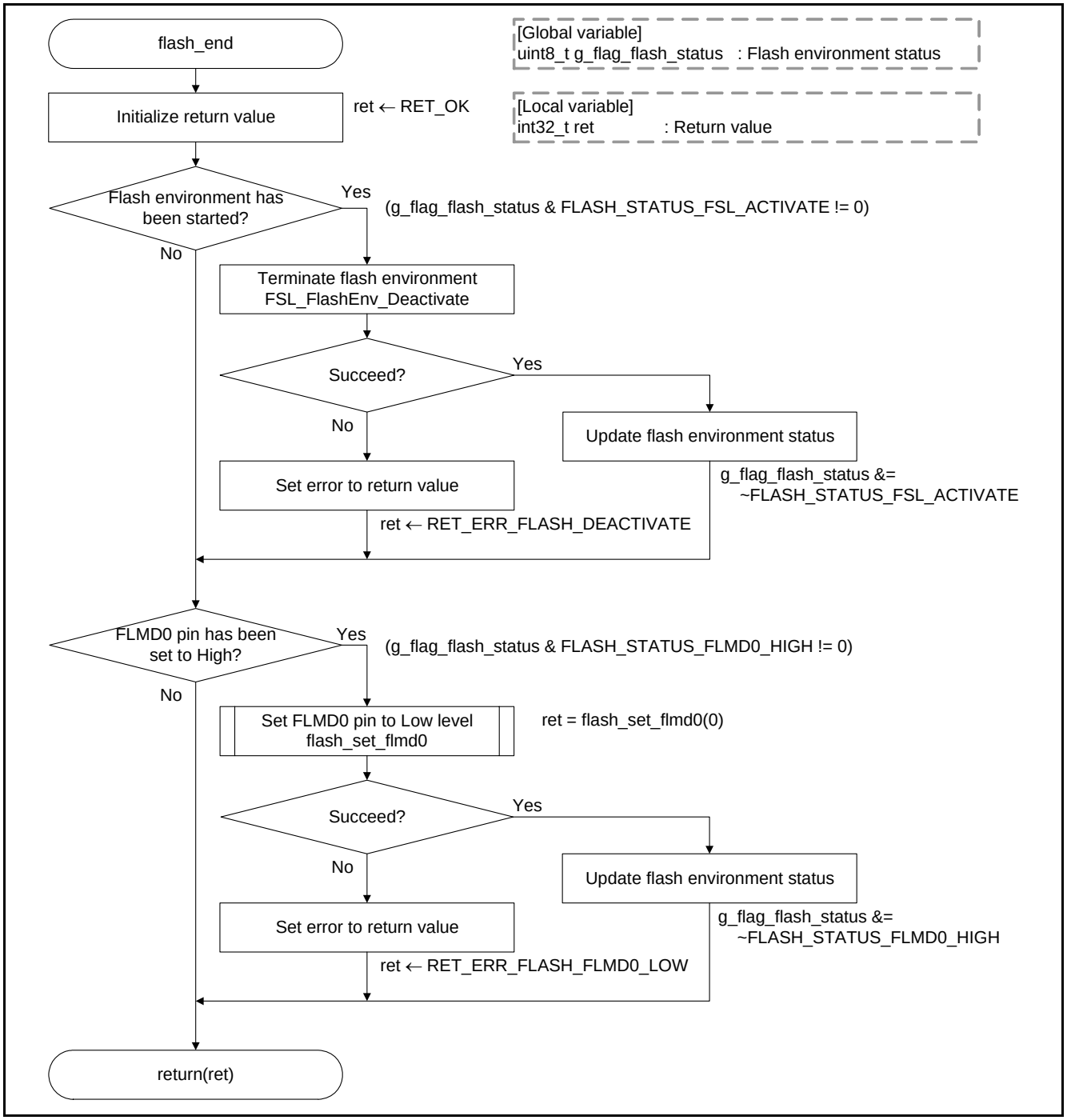


Figure 6.18 Termination Processing for Flash Environment

6.7.15 Setting for FLMD0 Pin Level

Figure 6.19 shows the Setting for FLMD0 Pin Level.

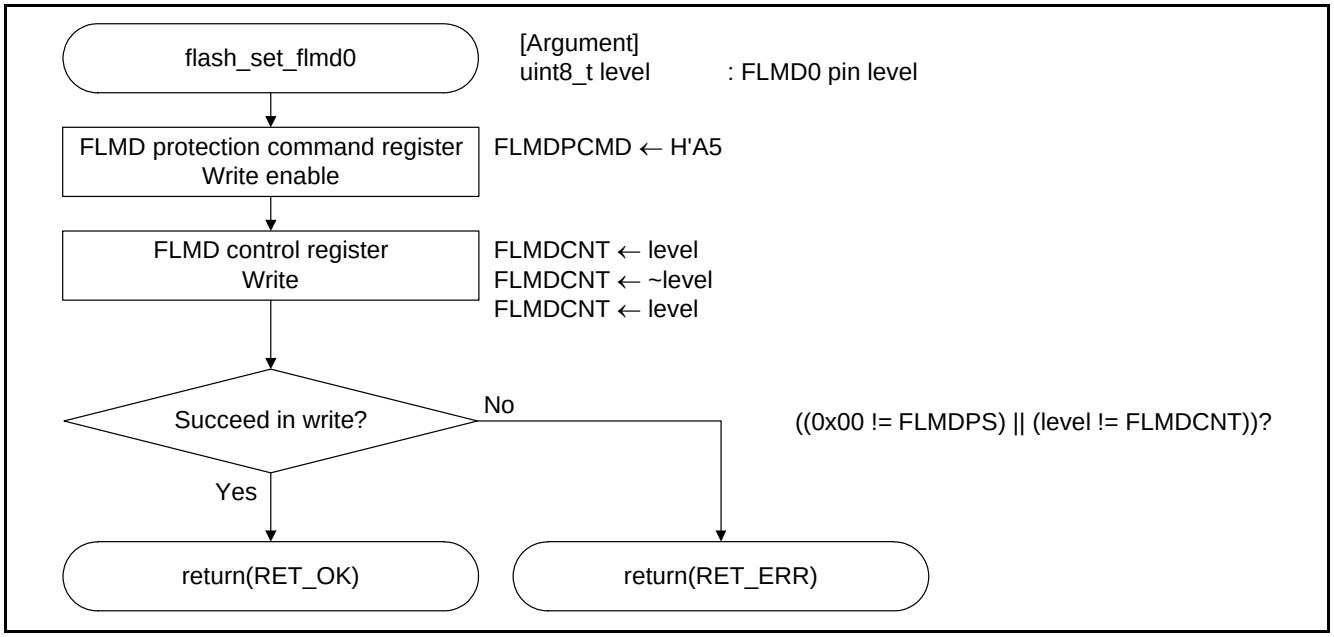


Figure 6.19 Setting for FLMD0 Pin Level

6.7.16 Store Processing for Receive Data

Figure 6.20 and Figure 6.21 show the Store Processing for Receive Data.

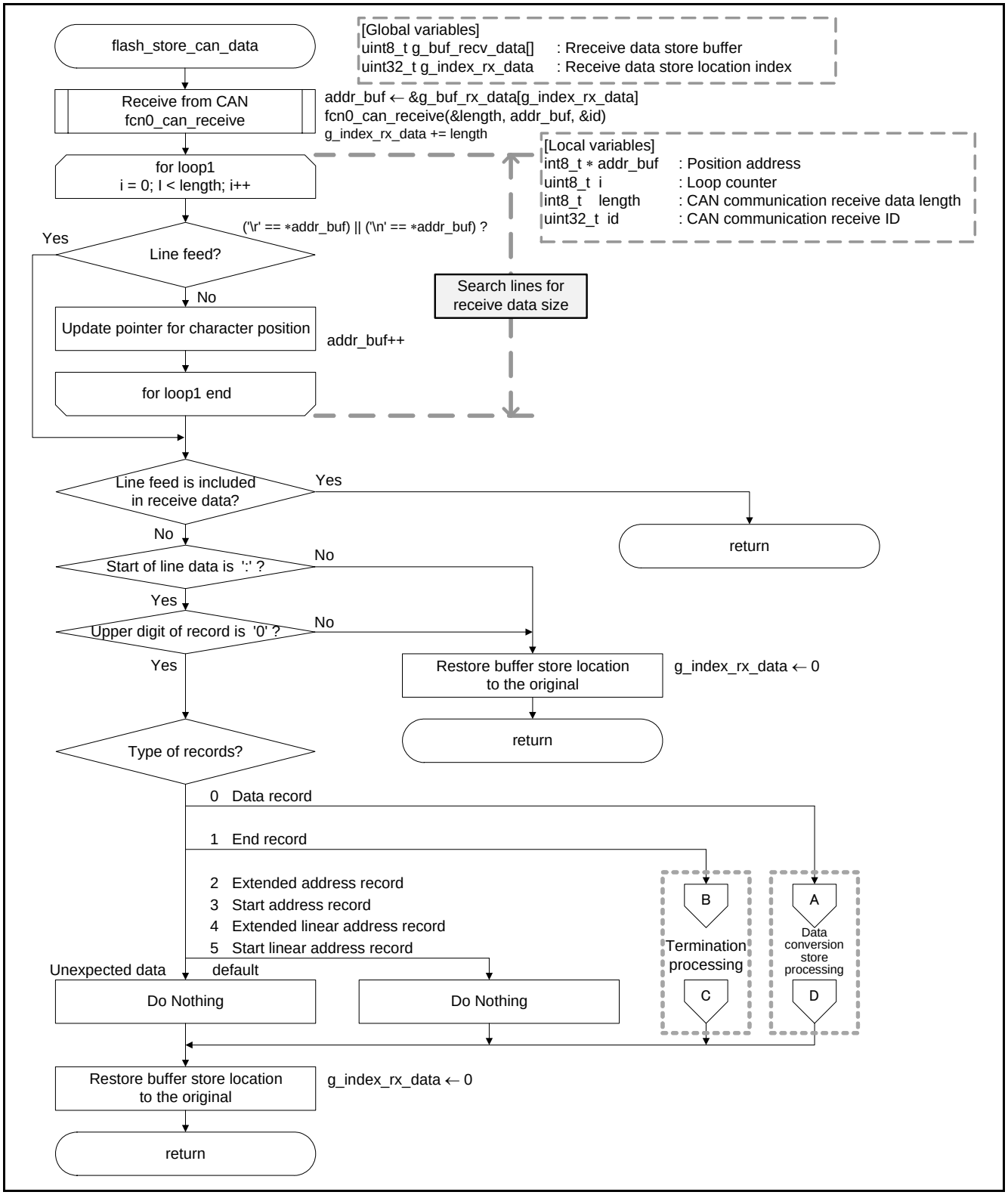


Figure 6.20 Store Processing for Receive Data (1/2)

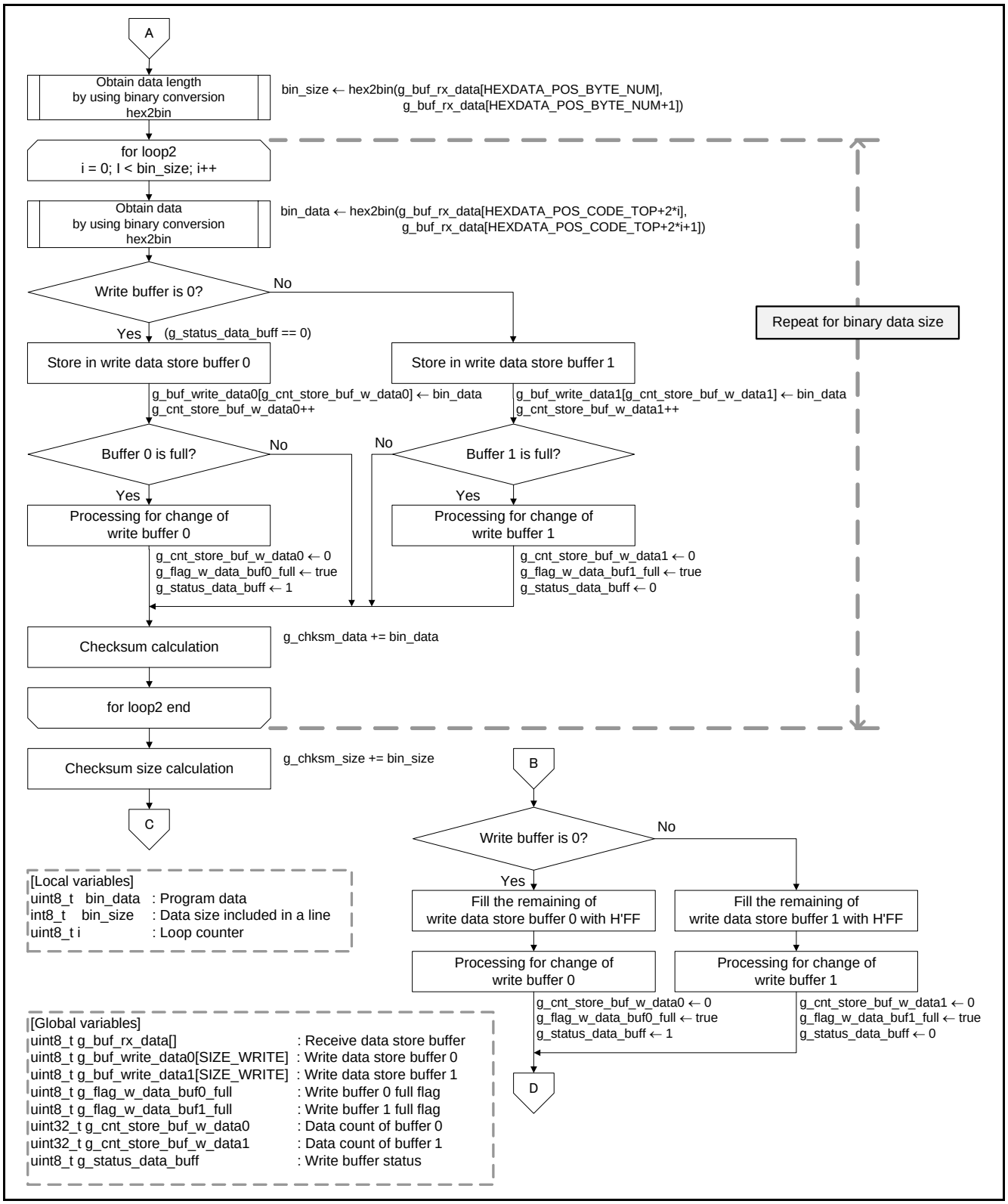


Figure 6.21 Store Processing for Receive Data (2/2)

6.7.17 Text Binary Conversion Processing

Figure 6.22 shows the Text Binary Conversion Processing.

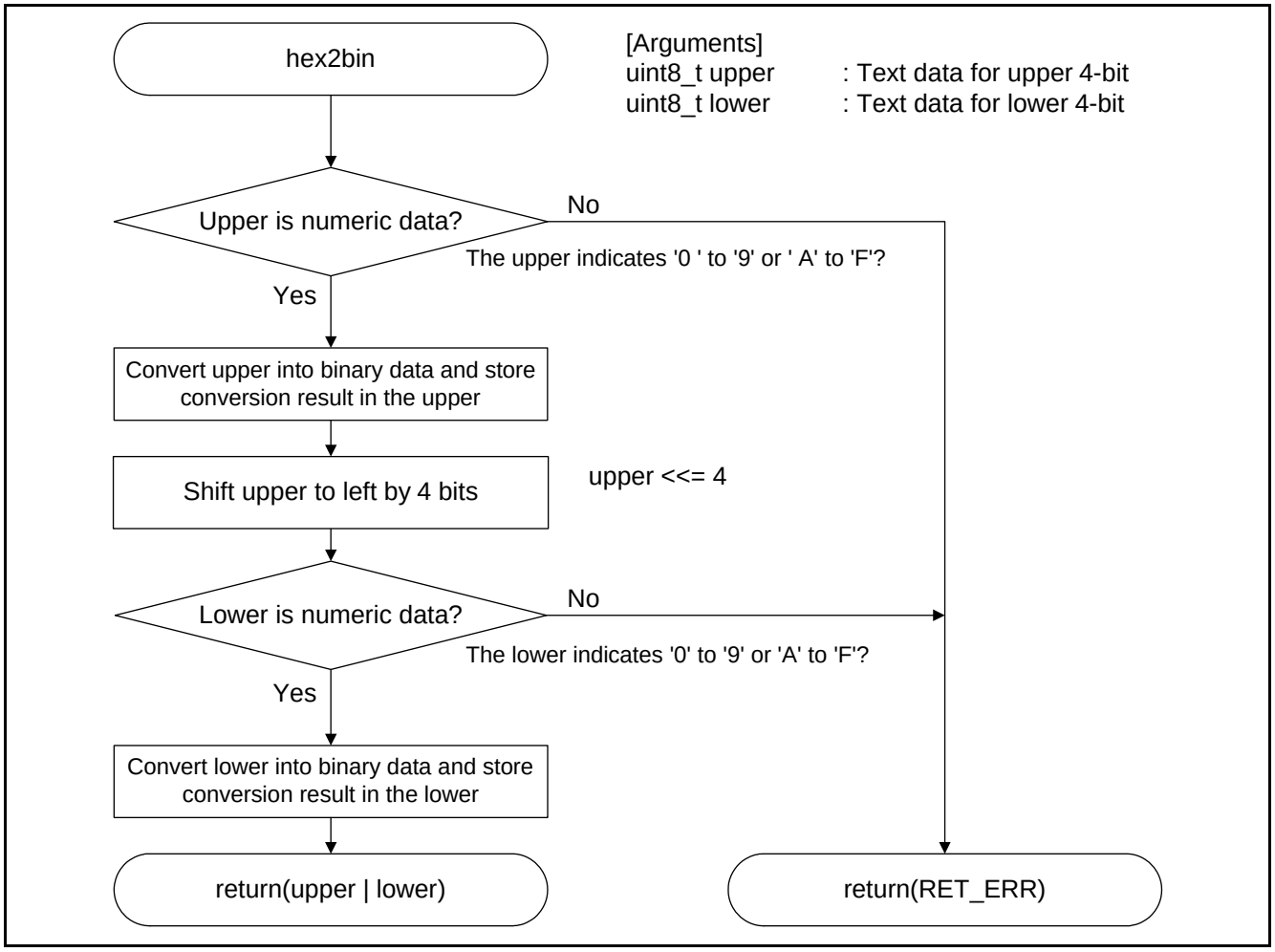


Figure 6.22 Text Binary Conversion Processing

6.7.18 TAUA0 Initialization for LED Flash with Fixed-Cycle (Sample Function in Reprogram Area and in Spare Area)

Figure 6.23 shows the TAUA0 Initialization for LED Flash with Fixed-Cycle.

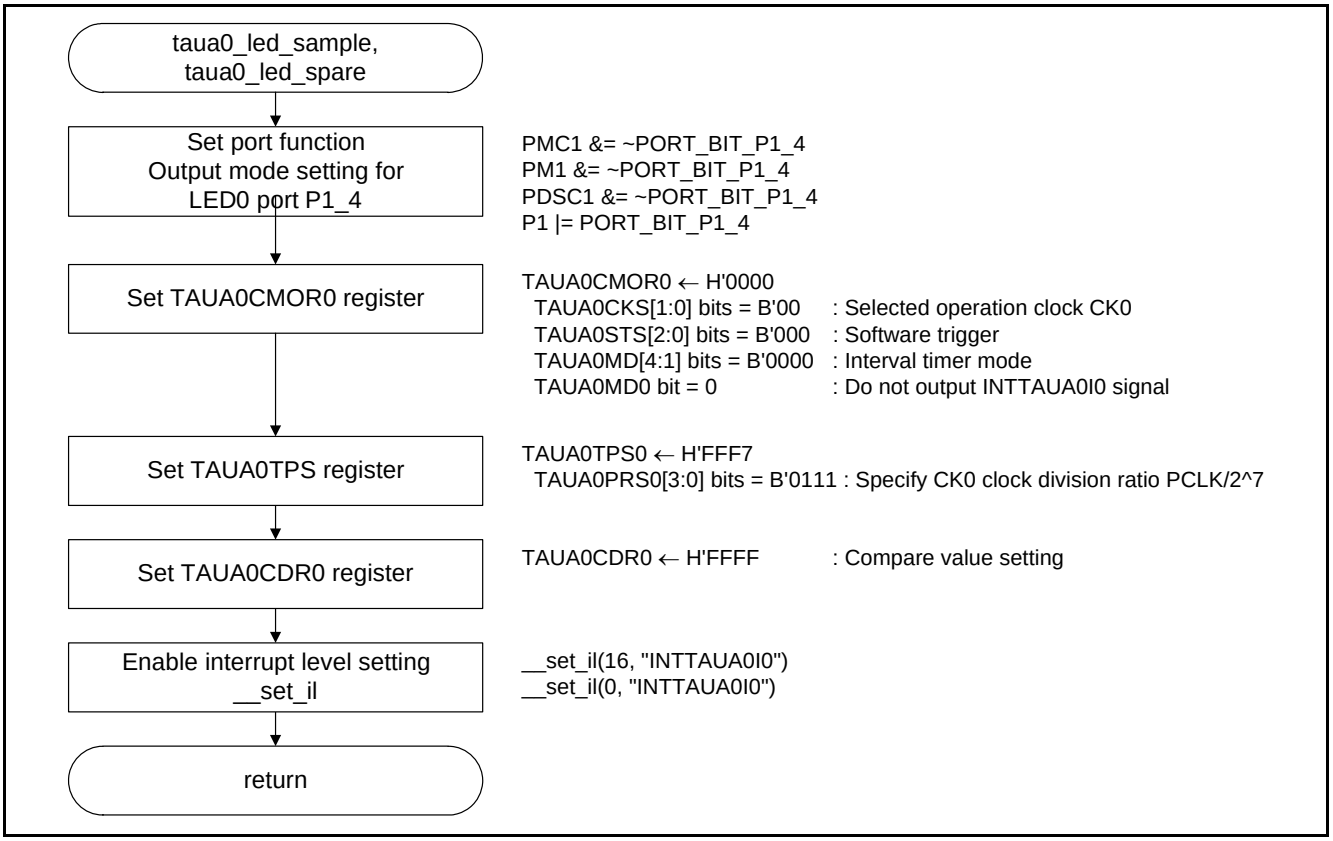
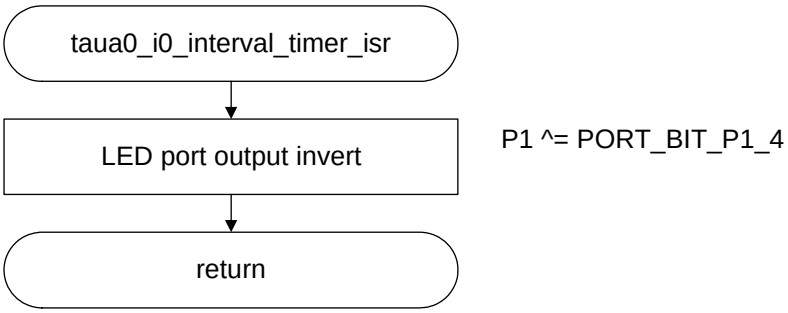


Figure 6.23 TAUA0 Initialization for LED Flash with Fixed-Cycle

6.7.19 TAUA0 Interval Timer Interrupt Processing

Figure 6.24 shows the TAUA0 Interrupt Processing.



[Note] Refer to "4.2.6 Precautions for Interrupt Generated During Use of FSL" for application.

Figure 6.24 TAUA0 Interrupt Processing

6.7.20 Initialization of CAN Controller Channel 0 (FCN0)

Figure 6.25 shows the Initialization of CAN Controller Channel 0 (FCN0).

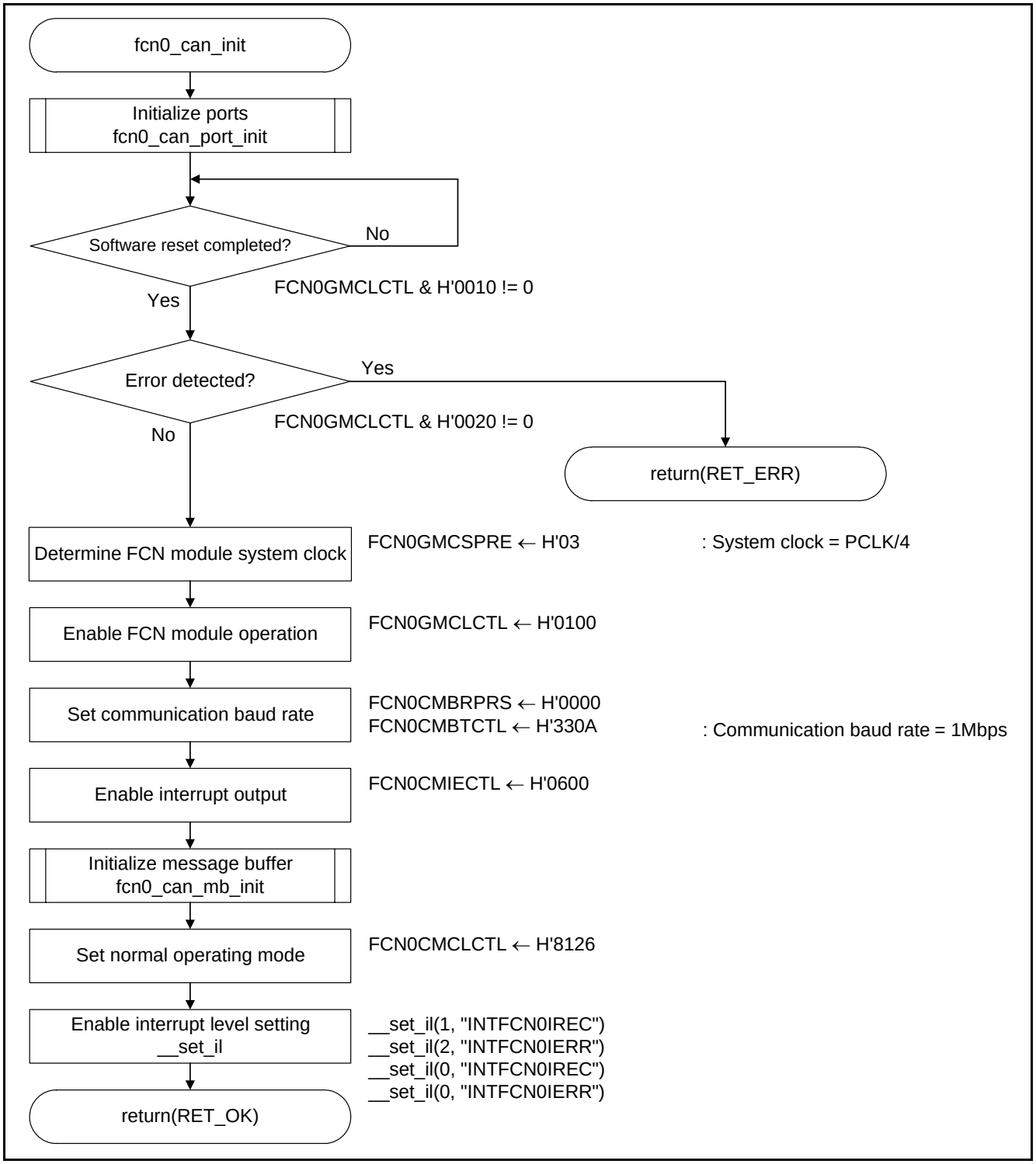


Figure 6.25 Initialization of CAN Controller Channel 0 (FCN0)

6.7.21 Initialization of FCN0 Port

Figure 6.26 shows the Initialization of FCN0 Port.

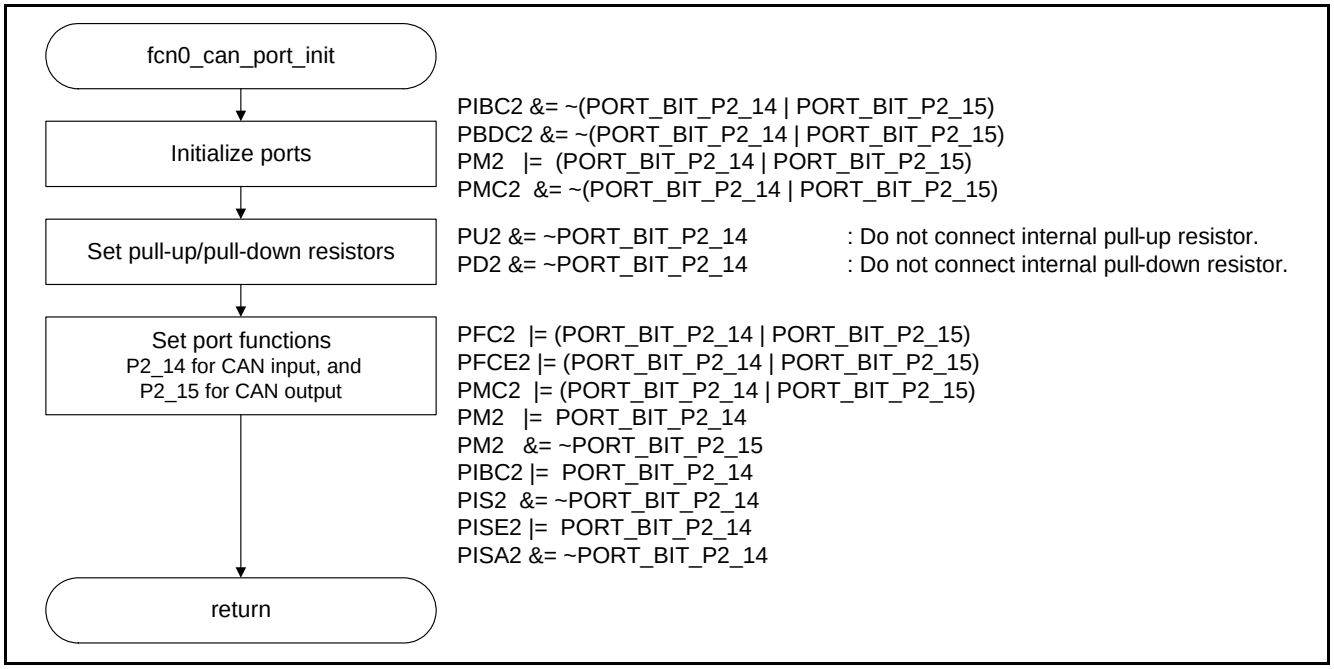


Figure 6.26 Initialization of FCN0 Port

6.7.22 Initialization of FCN0 Message Buffer

Figure 6.27 and Figure 6.28 show the Initialization of FCN0 Message Buffer.

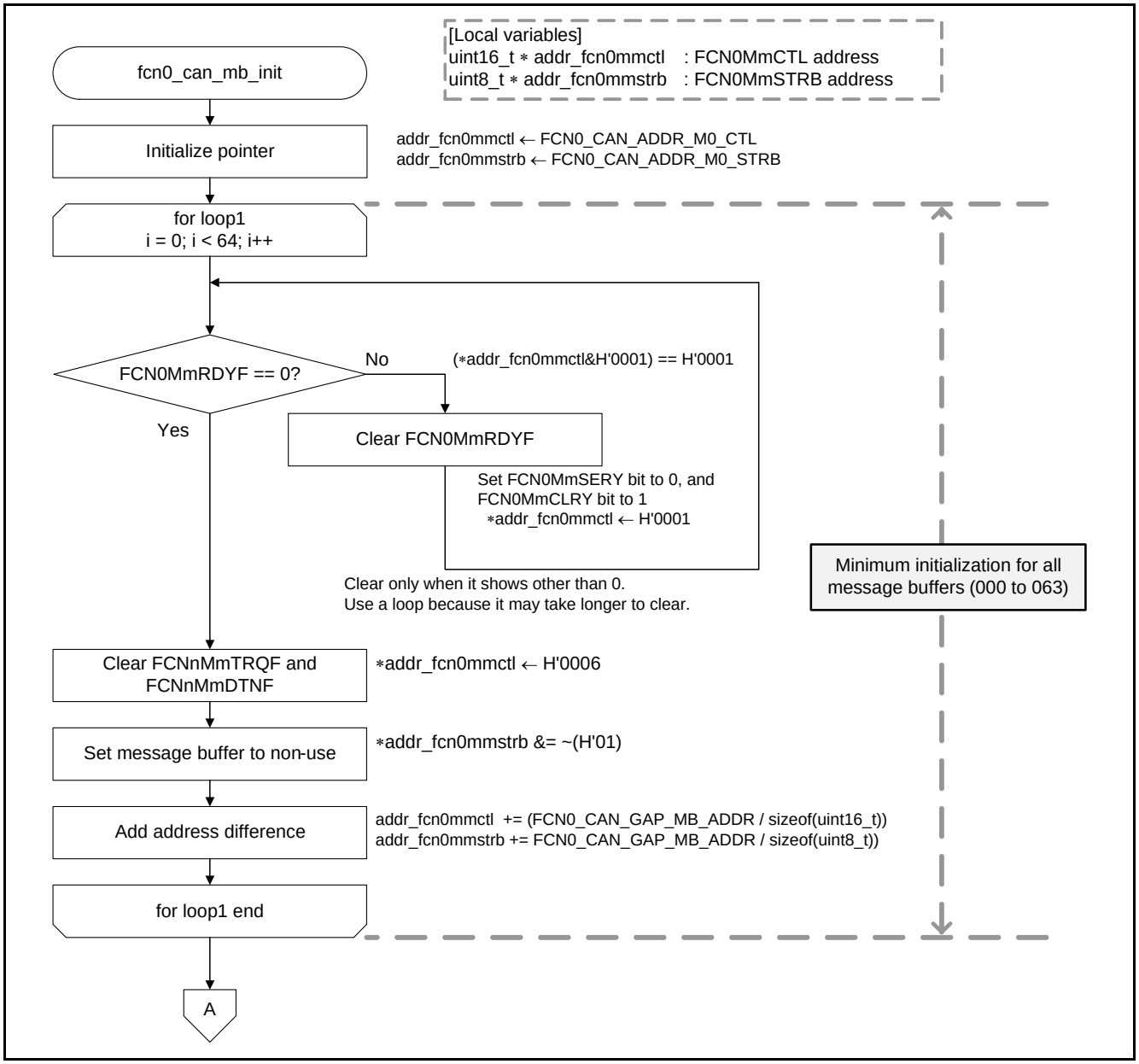


Figure 6.27 Initialization of FCN0 Message Buffer (1/2)

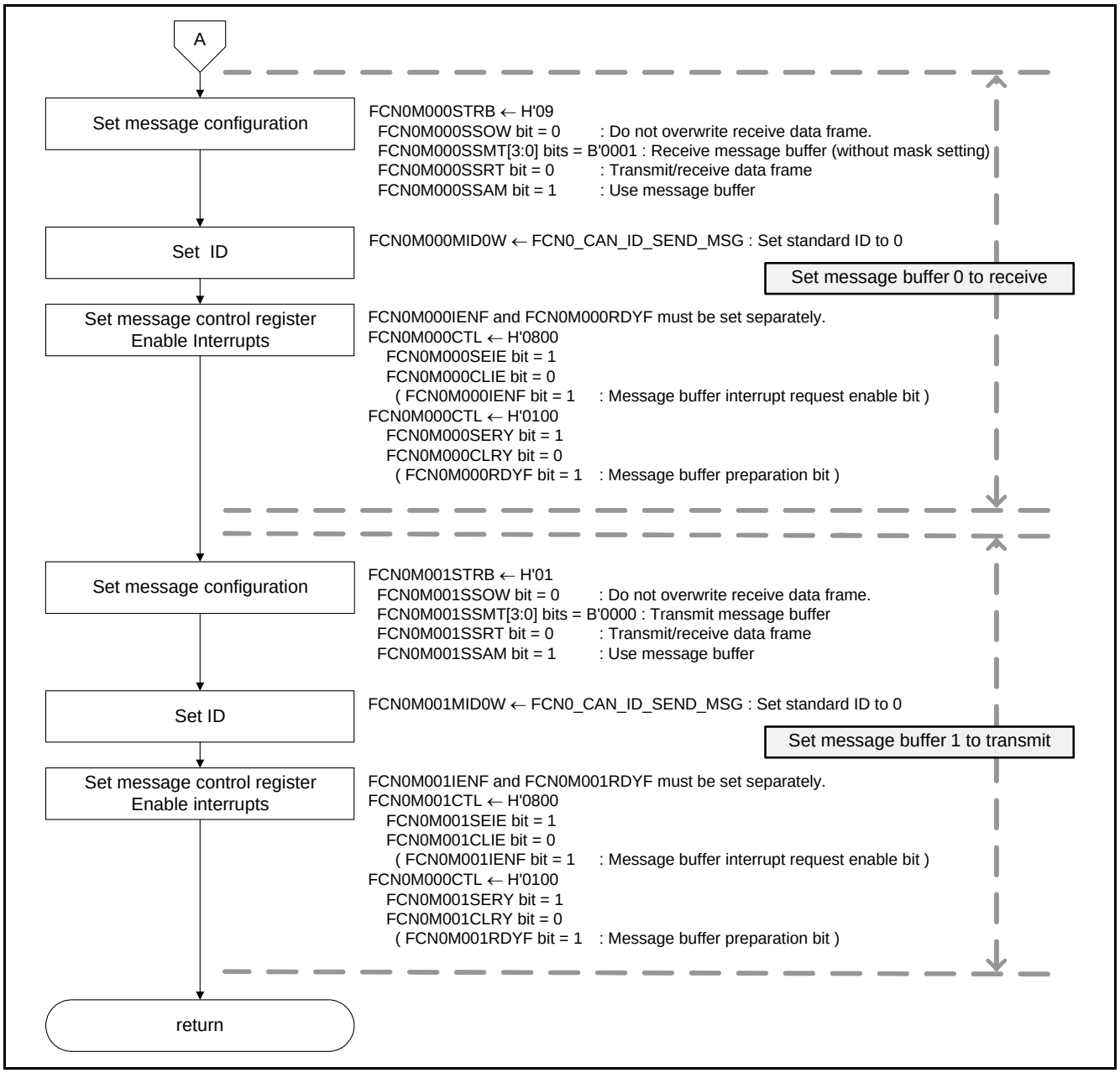


Figure 6.28 Initialization of FCN0 Message Buffer (2/2)

6.7.23 FCN0 Message Transmit Processing

Figure 6.29 shows the FCN0 Message Transmit Processing.

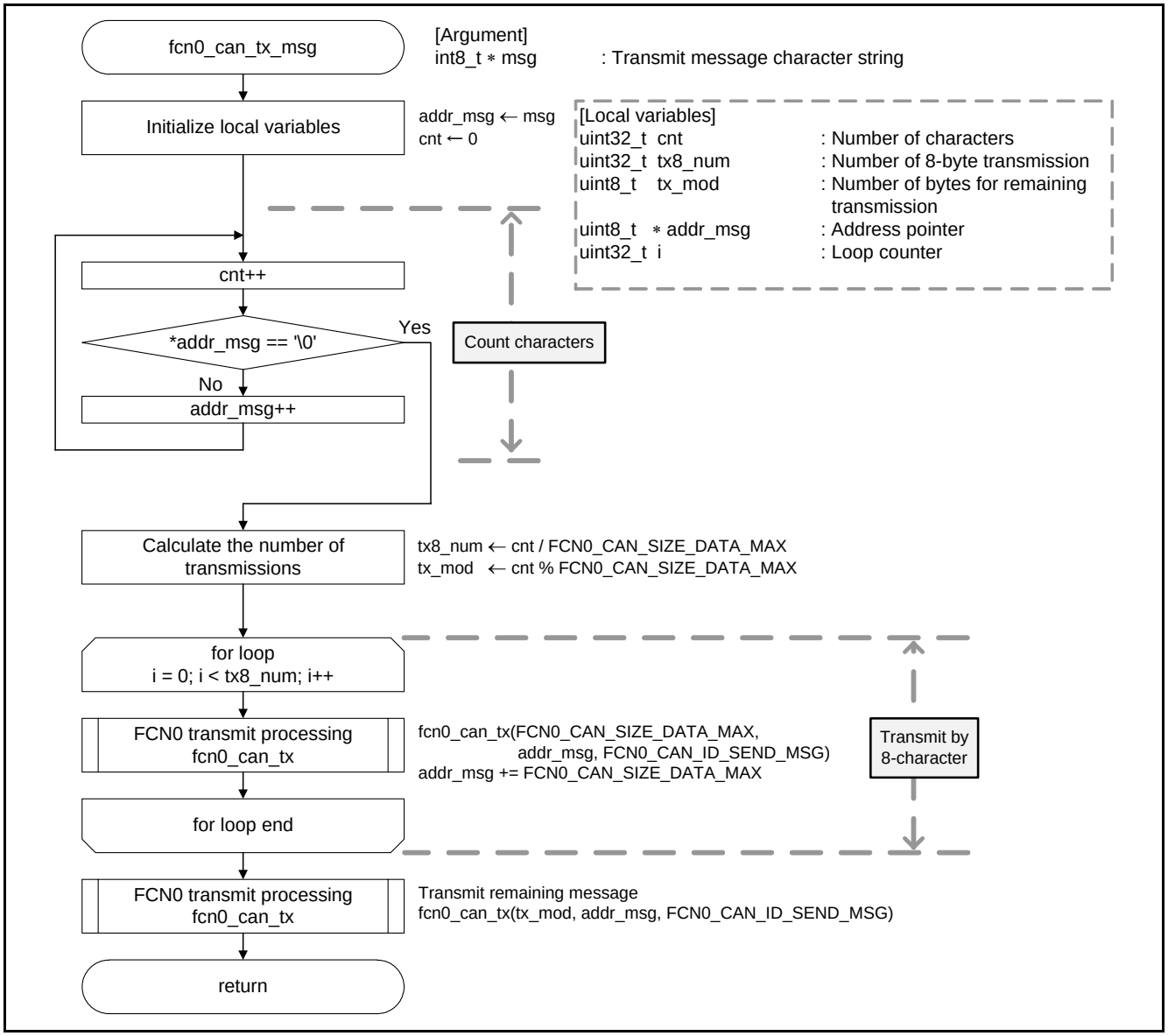


Figure 6.29 FCN0 Message Transmit Processing

6.7.24 FCN0 Transmit Processing

Figure 6.30 shows the FCN0 Transmit Processing.

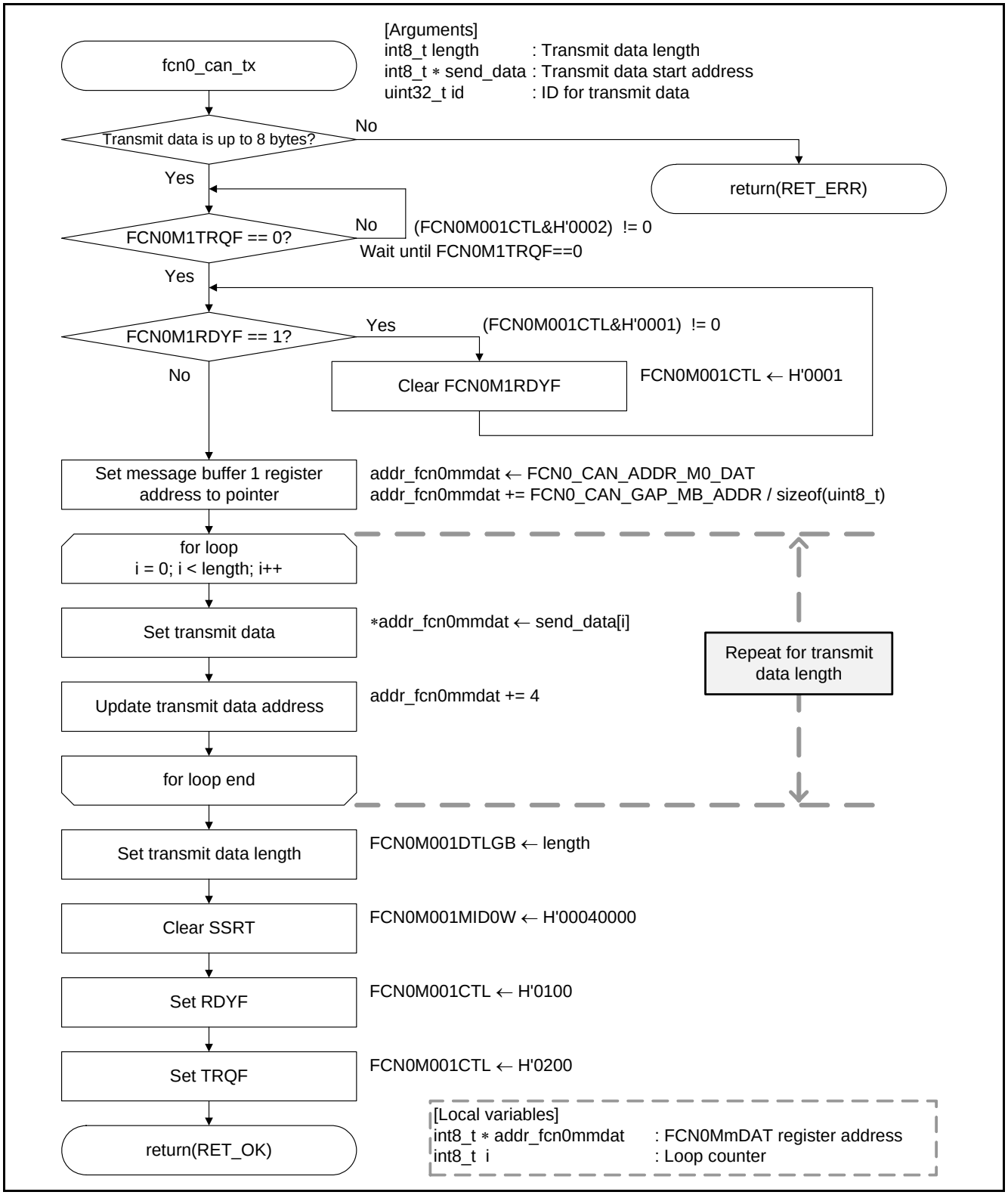


Figure 6.30 FCN0 Transmit Processing

6.7.25 FCN0 Receive Processing

Figure 6.31 shows the FCN0 Receive Processing.

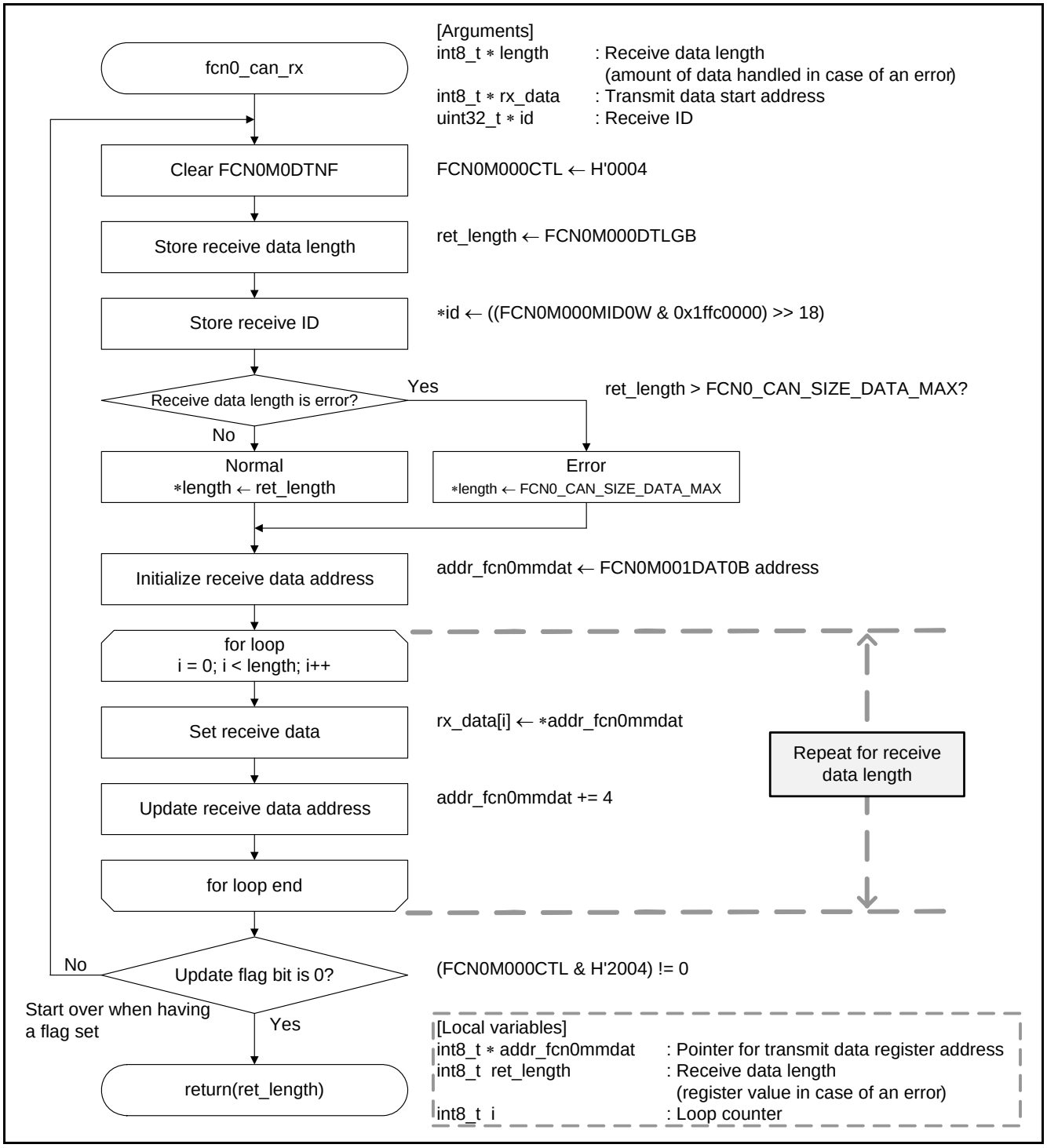
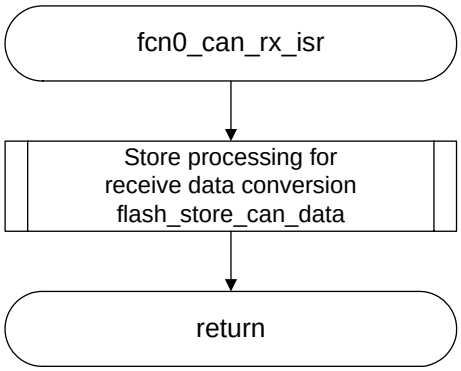


Figure 6.31 FCN0 Receive Processing

6.7.26 Interrupt Processing for FCN0 Receive Processing Completion

Figure 6.32 shows the Interrupt Processing for FCN0 Receive Processing Completion.



[Note] Refer to "4.2.6 Precautions for Interrupt Generated During Use of FSL" for application.

Figure 6.32 Interrupt Processing for FCN0 Receive Processing Completion

6.7.27 FCN0 Error Interrupt Processing

Figure 6.33 shows the FCN0 Error Interrupt Processing.

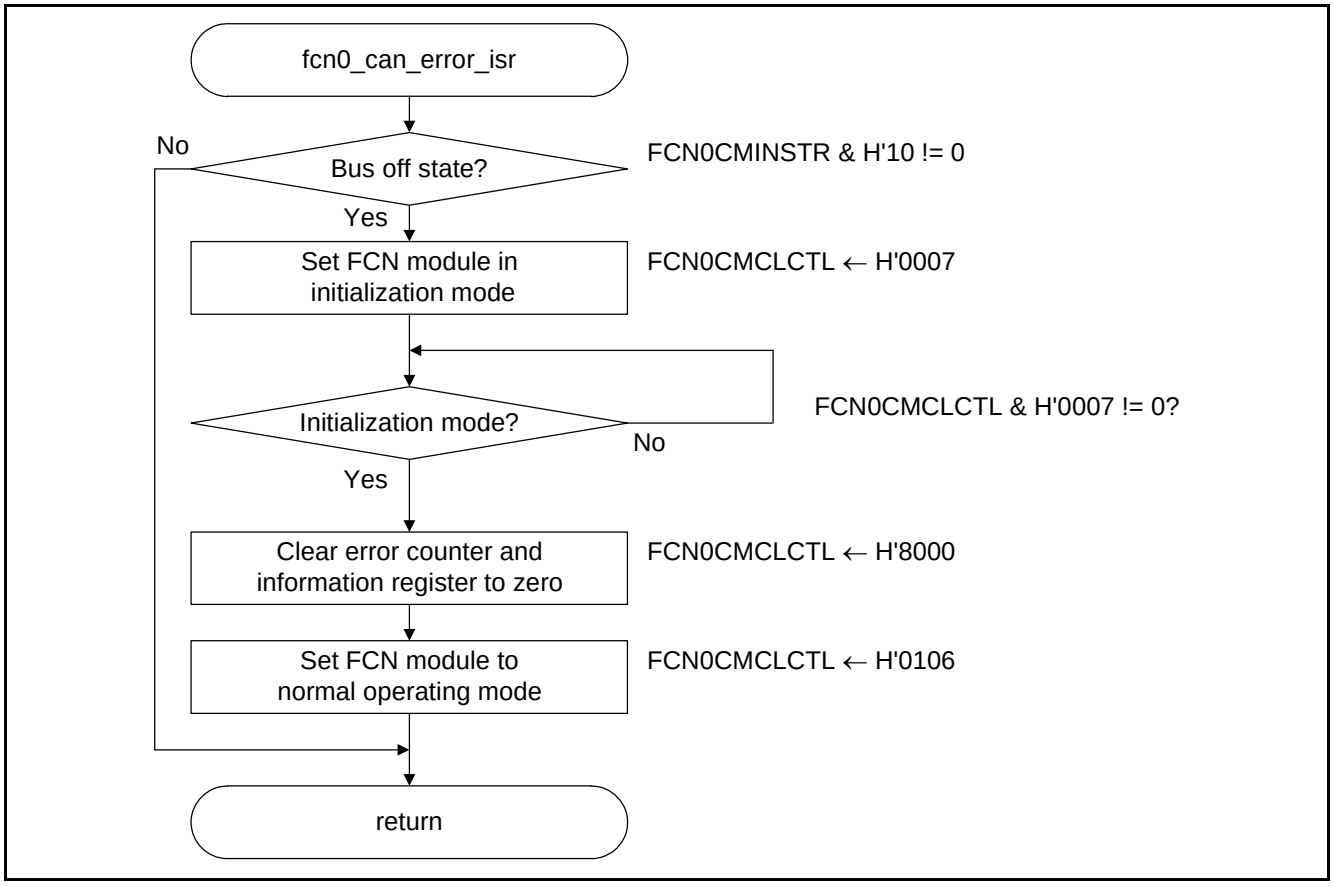


Figure 6.33 FCN0 Error Interrupt Processing

7. Operation Overview

In this sample program, the updating program is transmitted using the CAN communication host device. The V850E2/ML4 CPU board R0K0F4022C000BR is used as the CAN communication host device to describe the control with the host PC.

Figure 7.1 shows the Hardware Configuration Example for Sample Code. To execute the CAN communication with the V850E2/ML4 which is run by this sample program, set another CPU board to pass the data through between the serial communication and the CAN communication. (This sample program includes the load module file, v850e2ml4_serial_can_through.lmf which is a program for data passing.)

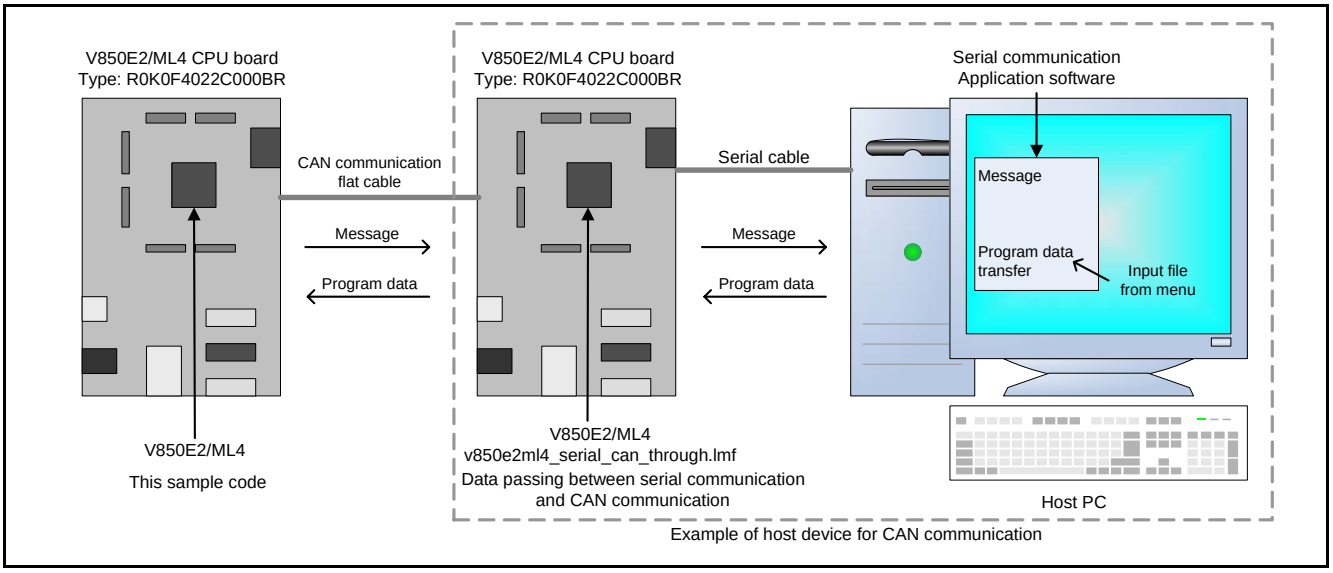


Figure 7.1 Hardware Configuration Example for Sample Code

JP8 and JP10 for signal selection of the two CPU boards should be switched to 2-3 to use the CAN as shown in Table 7.1. JP1 of the CPU which is run by this sample program should be switched to 2-3 to use the INTP1 external interrupt switch (SW4).

The CAN connectors (J4) are connected by the cable for CAN communication between the two CPU boards. The CPU for data through (serial port connector (J5)) and the host PC should be connected by the serial cable.

Refer to "V850E2/ML4 CPU board R0K0F4022C000BR User's Manual" for more details about the CPU board jumper settings and connectors.

Table 7.1 Jumper List

Jumper	1-2 (default)	2-3 (used in this program)
JP1*	VBUS	P2_3
JP8 (P2_14)	SDL1	CAN0RXD
JP10 (P2_15)	SDA1	CAN0TXD

[Note] *: Only for the CPU board which is run by this sample program.

An operation procedure with the VT100 compatible terminal emulator is described as follows. First of all, activate the terminal emulator and set for serial port connection. Select the number connected to the through board for the serial port number of the terminal emulator. The setting values for serial ports are listed in Table 7.2.

Table 7.2 Serial Port Setting

Item	Setting value
Bit/sec	9600bps
Data bit	8bit
Parity	None
Stop bit	1bit
Flow control	None

After the above setting is completed, switch on the through board and the board for this sample program.

When the board for this sample program is activated, the V850E2/ML4 transmits a message "Generate INTP1 interrupt for transition to flash programming event." to the host.

Then the V850E2/ML4 executes the program stored in the reprogram area, and flashes the LEDs on the board with the fixed period.

When the INTP1 switch (SW4) on the board is pushed in this condition, the V850E2/ML4 transmits a message "--> INTP1 detected!" to the host. When the INTP1 interrupt is generate, the V850E2/ML4 enters into flash reprogram processing, and erases the update area. After the erasing is completed, the V850E2/ML4 transmits a message "Send subroutine code to update program in Intel expanded Hex format." to the host, and enters into wait state for data reception from the host.

In case of transmitting a file with Intel expanded hex format as a program data from the host, the terminal emulator transmit function should be used. When transmitting an appropriate file (such as v850e2ml4_sample_host_send.hex), the program data is transmitted to the through board by serial communication, and then the through board transmits the data to the V850E2/ML4 for this sample program by CAN communication.

After the writing is completed, the V850E2/ML4 transmits a message "Successfully Finish Writing Program Data. Please Reset." to the host, it enters into wait state for reset. Reset the board.

When restarting, the LEDs on the board flash with the different period from previous one. If the data reception/flash reprogram (update) prior to restart was failed to execute properly, the V850E2/ML4 finds a checksum error at the time of restarting by reset input. The V850E2/ML4 executes the program in the spare area.

8. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

9. Reference Documents

User's Manual: Hardware

V850E2/ML4 User's Manual: Hardware Rev.2.00 (R01UH0262EJ)
The latest version can be downloaded from the Renesas Electronics website.

Technical Update/Technical News

The latest information can be downloaded from the Renesas Electronics website.

User's Manual: Development Tools

CubeSuite+ V1.03.00 Integrated Development Environment User's Manual: Coding for CX compiler Rev.1.00 (R20UT2139EJ)
CubeSuite+ V1.03.00 Integrated Development Environment User's Manual: Build for CX compiler Rev.1.00 (R20UT2142EJ)
V850E2/ML4 CPU Board R0K0F4022C000BR User's Manual Rev.1.00 (R20UT0778EJ)
The latest version can be downloaded from the Renesas Electronics website.

User's Manual: Software

V850E2M User's Manual: Architecture Rev.1.00 (R01US0001EJ)
The latest version can be downloaded from the Renesas Electronics website.

Website and Support

Renesas Electronics website

http://www.renesas.com

Inquiries

http://www.renesas.com/contact/

REVISION HISTORY	V850E2/ML4 Application Note Updating Program Code by Using Flash Self Programming with CAN Controller

Rev.	Date	Description	
		Page	Summary
1.00	Mar. 01, 2013	-	First edition issued

General Precautions in the Handling of MPU/MCU Products

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

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

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.
   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

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

RENESAS

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

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

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

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

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

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

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

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

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

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

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

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

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

Colophon 2.2