RX62N, RX621 Group

Example of Memory-Protection Unit Settings

R01AN1078EJ0100
Rev.1.00
Jul 01, 2014

Introduction

This application note describes, as example settings for the memory-protection unit (MPU) of the RX62N Group and RX621 Group, the method for specifying memory-protection areas, the area search function, the conditions under which access exceptions are generated, and exception handling when an access exception occurs.

Target Device

RX62N Group, RX621 Group

Contents

1. Specifications

The example program presented in this application note reads, writes, and executes code in a "read-only area," a "write-only area," and an "execute-only area," respectively, set in the MPU.

A memory-protection error occurs when an unauthorized access to one of these areas takes place.

It is also possible to use the area search function to detect the access enabled/disabled status of an area.

The example program presented in this application note uses three switches and four LEDs.

- Pressing switch 1 (SW1) selects the area to be accessed from among the available areas.
- Pressing switch 2 (SW2) toggles the area search function on and off (enabled/disabled).
- Pressing switch 3 (SW3) performs an access, using the access method specified by SW1 and SW2.

The LEDs indicate the operating status of the program.

- LED0 and LED1 indicate the area being accessed.
- LED2 indicates whether the area search function is enabled or disabled.
- LED3 indicates the execution result.

Note that the example program presented in this application note uses conditional compilation to switch between read, write, and execute access. For details, see section 5, Software.

Table 1.1 lists the peripheral functions used by the sample program. Figure 1.1 shows an operation overview.

Table 1.1 Peripheral Functions and Their Applications

Peripheral Function	Application
MPU	Memory protection
Compare match timer channel 0 (CMT0)	System timer (switch chattering elimination, LED blink timer)

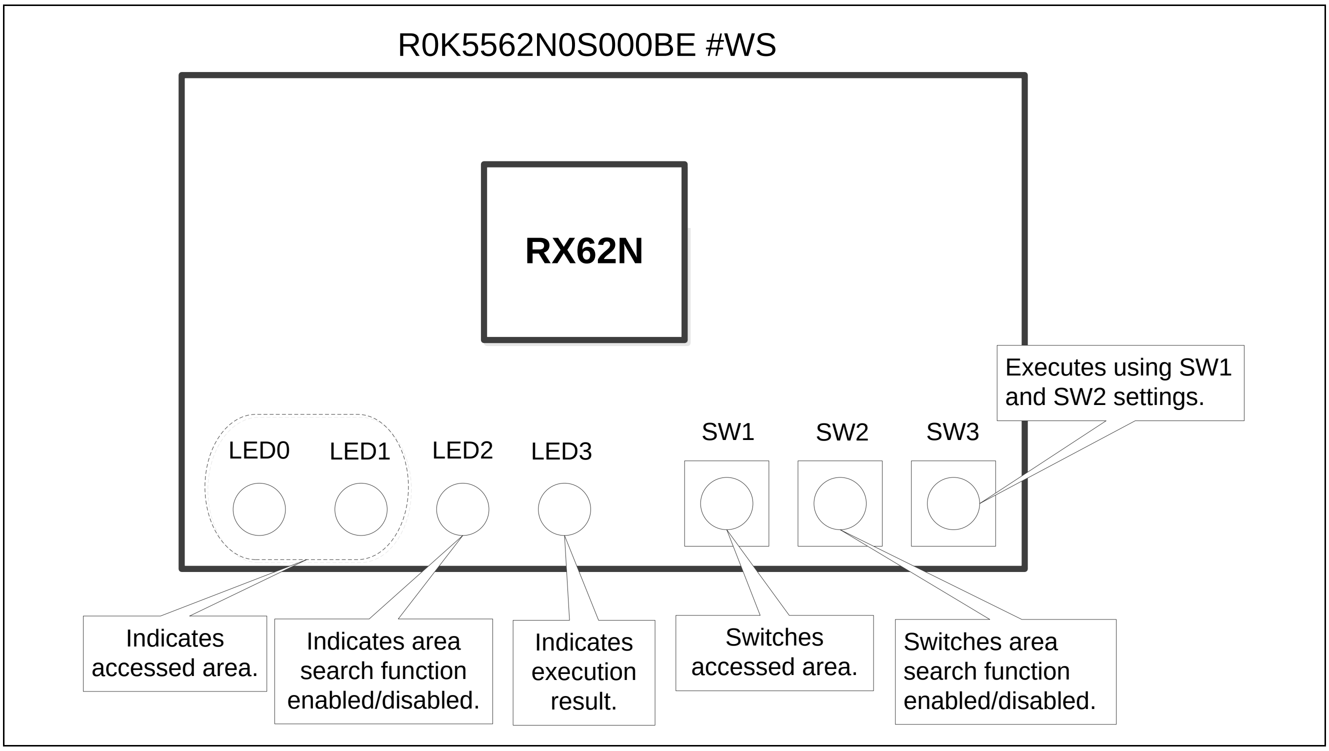

Figure 1.1 Operation Overview

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	RX62N Group R5F562N8BDBG
Operating frequency	• Main clock: 12.0 MHz • ICLK: 96 MHz • PCLK: 48 MHz • BCLK: 24 MHz
Operating voltage	5.0 V (CPU operating voltage: 3.3 V)
Integrated development environment	High-performance Embedded Workshop Version 4.09.00.007
Toolchain	RX Standard Toolchain (V1.2.0.0) RX Family C/C++ Compiler Driver V.1.02.00.000 RX Family C/C++ Compiler V.1.02.00.000 RX Family Assembler V.1.02.00.000 Optimizing Linkage Editor V.10.02.00.000 RX Family C/C++ Standard Library Generator V1.02.00.000
iodefine.h version	V 2.0
Endian mode	Little-endian
Operating mode	Single-chip mode
Sample code version	Version 1.00
Board used	Renesas Starter Kit+ for RX62N (R0K5562N0S000BE #WS)

3. Peripheral Functions

Supplementary information about the MPU is presented below. A basic discussion is provided in RX62N Group, RX621 Group—User's Manual: Hardware.

The RX62N Group and RX621 Group are equipped with an on-chip MPU that performs address checking of CPU accesses within the entire address space (0000 0000h to FFFF FFFFh). Area-specific access-control information can be specified for up to eight areas and the background area. The supported access-control information for the individual areas consists of permission to read, permission to write, and permission to execute.

The access-control information is enabled when the processor mode of the CPU is user mode. Memory protection is not applied when the CPU is in supervisor mode. When the access-control information is enabled and an attempt by the CPU to access an area causes an access-control information violation (a memory-protection error) to be detected, access exception handling starts. Note that it is possible to limit accesses to permitted areas only, and prevent the CPU from generating access-control information violations, by using the area search function to determine the access-control information of areas to be accessed.

Notes:
1. The MPU only monitors memory accesses by the CPU. It has no effect on memory accesses by the DMAC or DTC.
2. Interrupt handling is performed in supervisor mode, so the MPU access-control information has no effect.
3. To enable the MPU access-control information, it is necessary to put the CPU into user mode.
4. The background area refers to the entire address space, except for the areas set as memory-protection areas 0 to 7.
5. Registers related to the memory-protection unit can be accessed only in supervisor mode.

For details on transitioning from user mode to supervisor mode, see 5.7.17, Change from User Mode to Supervisor Mode.

For details on transitioning from supervisor mode to user mode, see 5.7.16, Change from Supervisor Mode to User Mode.

3.1 Memory Protection by the MPU

The example program presented in this application note uses the MPU to specify the memory protection applied to different areas of memory. Figure 3.1 shows the section allocations and memory-protection area settings, and Table 3.1 lists the access-control information for each memory area.

Note that in order to implement execute access to monitor area A and monitor area B in the on-chip RAM area, the program PSAMPLE1 is transferred to PRead_Area and the program PSAMPLE2 is transferred to PWrite_Area.

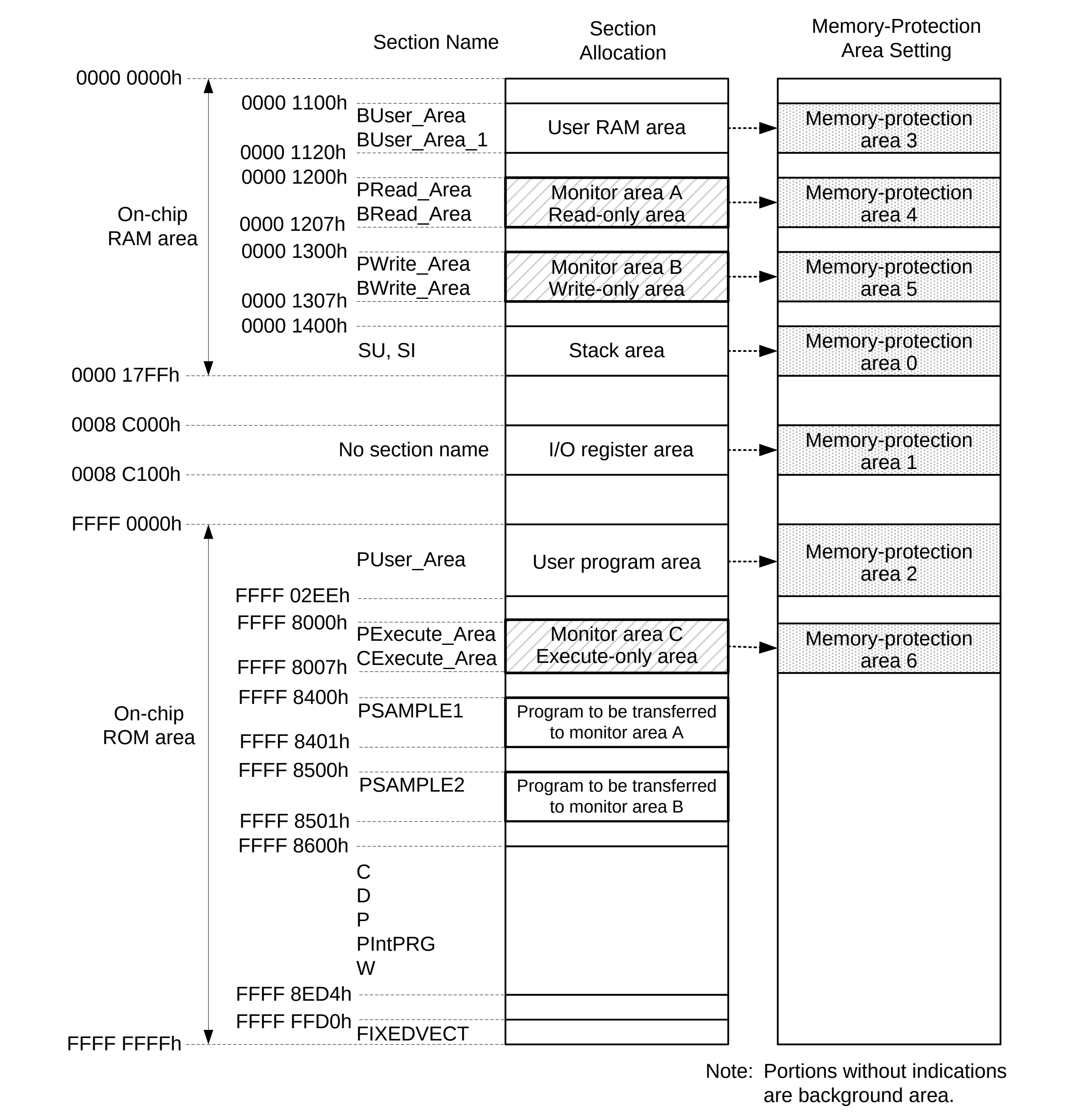

Figure 3.1 Section Allocations and Memory-Protection Area Settings

Table 3.1 Access-Control Information by Memory Area

Area Name and Description	Corresponding Register	Setting Address	Access Control Information: Read	Write	Execute
Stack area specified in sample code	RSPAGE0 (area 0 start)	SU start address			
	REPAGE0 (area 0 end)	SI end address	○	○	×
I/O register area specified in sample code	RSPAGE1 (area 1 start)	0008 C000h			
	REPAGE1 (area 1 end)	0008 C100h	○	○	×
Program area specified in sample code	RSPAGE2 (area 2 start)	PUser_Area start address			
	REPAGE2 (area 2 end)	PUser_Area end address	×	×	○
RAM use area specified in sample code	RSPAGE3 (area 3 start)	BUser_Area start address			
	REPAGE3 (area 3 end)	BUser_Area_1 end address	○	○	×
Monitor area A Read-only area	RSPAGE4 (area 4 start)	PRead_Area start address			
	REPAGE4 (area 4 end)	BRead_Area end address	○	×	×
Monitor area B Write-only area	RSPAGE5 (area 5 start)	PWrite_Area start address			
	REPAGE5 (area 5 end)	BWrite_Area end address	×	○	×
Monitor area C Execute-only area	RSPAGE6 (area 6 start)	PExecute_Area start address			
	REPAGE6 (area 6 end)	CExecute_Area end address	×	×	○
Not used	RSPAGE7 (area 7 start)	—	—	—	—
	REPAGE7 (area 7 end)	—	—	—	—
Background area[1]	MPBAC	Entire memory area other than the above areas	×	×	×

[Legend] ○ indicates enabled; × indicates disabled.

Note: 1. Access information has no effect for areas accessed in supervisor mode, in which the MPU memory-protection function is disabled.

4. Hardware

4.1 Hardware Configuration

In the example presented in this application note, the access method and access execution are manipulated by using switches SW1 to SW3. LEDs are used to show the operating status of the program.

Figure 4.1 shows a connection example for this application note.

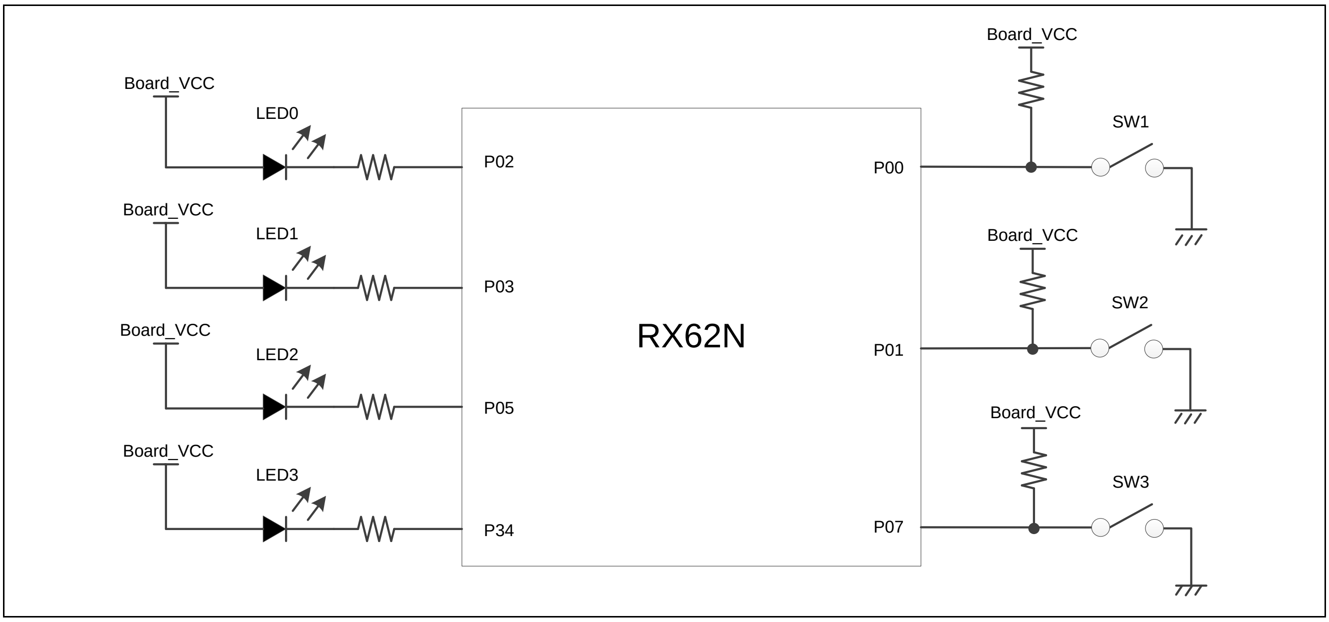

Figure 4.1 Connection Example

4.2 Pins Used

Table 4.1 lists the pins used and their functions.

Table 4.1 Pins Used and Their Functions

Pin Name	I/O	Function
P02	Output	LED0 control
P03	Output	LED1 control
P05	Output	LED2 control
P34	Output	LED3 control
P00	Input	SW1
P01	Input	SW2
P07	Input	SW3

5. Software

In the sample code the type of access by the CPU to the specified area is fixed at read access. To change the type of access by the CPU, it is necessary to change the definition of PRG_TYPE in the header file main.h. Figure 5.1 shows the portion of the header file main.h in which PRG_TYPE is defined. The type of access performed by the CPU can be selected among “read,” “write,” and “execute” by changing the PRG_TYPE definition shown in figure 5.1 and then recompiling sample code. Table 5.1 lists the correspondences between the PRG_TYPE definition and the operation of the sample code.

Figure 5.1 Location for Changing Access Method in Sample Code (Example with Read Access Selected)

```
< Excerpt from file main.h >

#define    READ_ACCESS       (0x00000008)
#define    WRITE_ACCESS      (0x00000004)
#define    EXECUTE_ACCESS    (0x00000002)

#define    PRG_TYPE          READ_ACCESS
```

Figure 5.1 Location for Changing Access Method in Sample Code (Example with Read Access Selected)

Table 5.1 Correspondences between PRG_TYPE Definition and Operation of Sample Code

Definition	Sample Code Operation
READ_ACCESS	Fixed at read access to specified area
WRITE_ACCESS	Fixed at write access to specified area
EXECUTE_ACCESS	Fixed at execute access to specified area

A description of the software is provided below. The only portion of the program that changes when conditional compilation is used to switch between read, write, and execute access is described in 5.7.9, Function for Accessing Specified Access Area. Flowcharts of operation when different conditional compilation options are used are shown in 5.7.9, Function for Accessing Specified Access Area.

5.1 Operation Overview

The sample code accesses the individual memory areas listed in Table 3.1 and makes section allocation and memory-protection area settings as shown in Figure 3.1. Figure 5.2 is a sequence diagram. When the program is run it enters the access method setting mode. The initial settings specify monitor area A as the access area and enable the area search function.

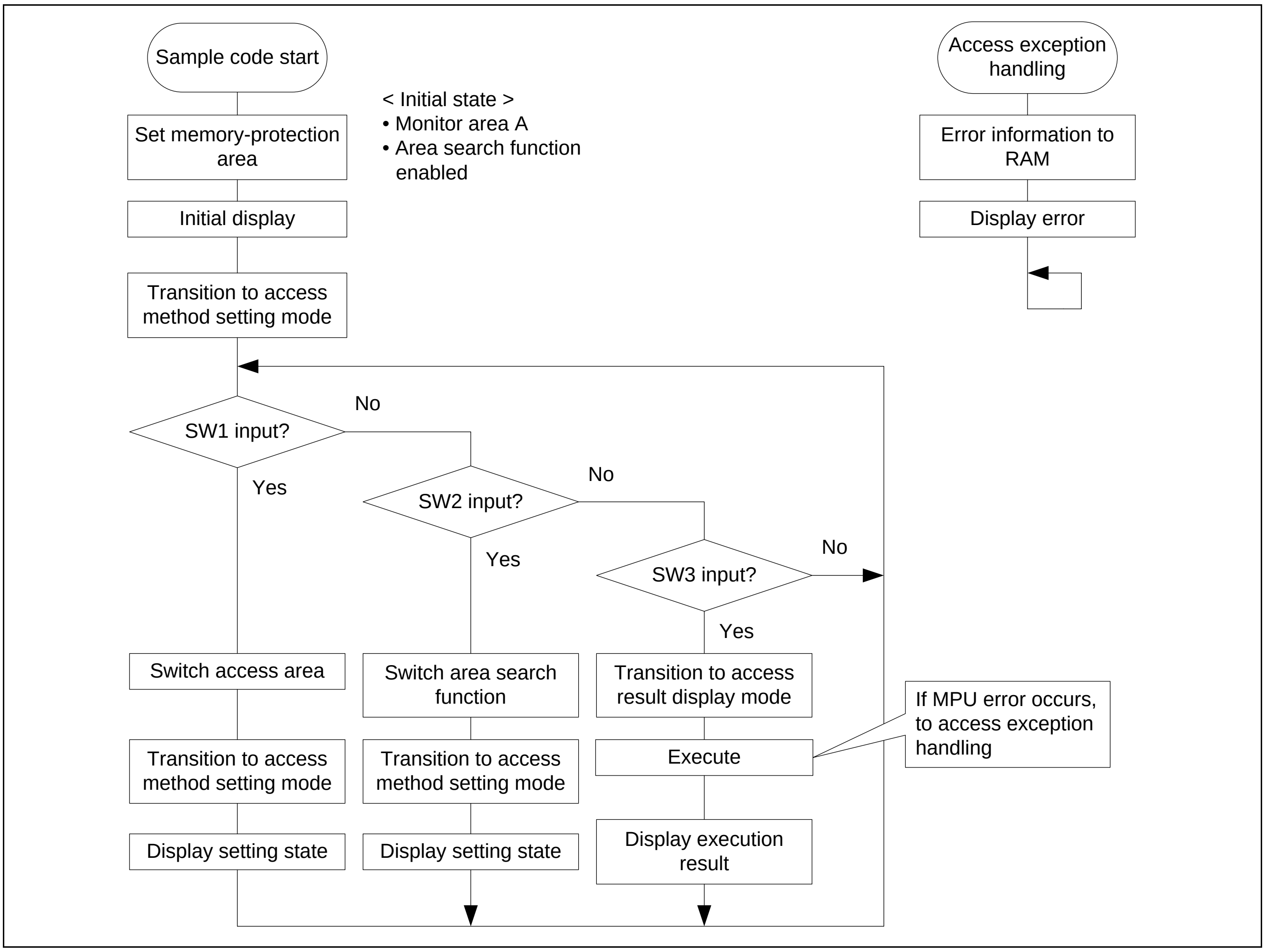

Figure 5.2 Sequence Diagram

5.1.1 Access Method Setting Mode

(1) SW1 Pressed

Each press of SW1 cycles the area to be accessed one step forward in the following sequence: monitor area A (memory-protection area 4) → monitor area B (memory-protection area 5) → monitor area C (memory-protection area 6) → monitor area A (memory-protection area 4). Figure 5.3 illustrates the operation when SW1 is pressed.

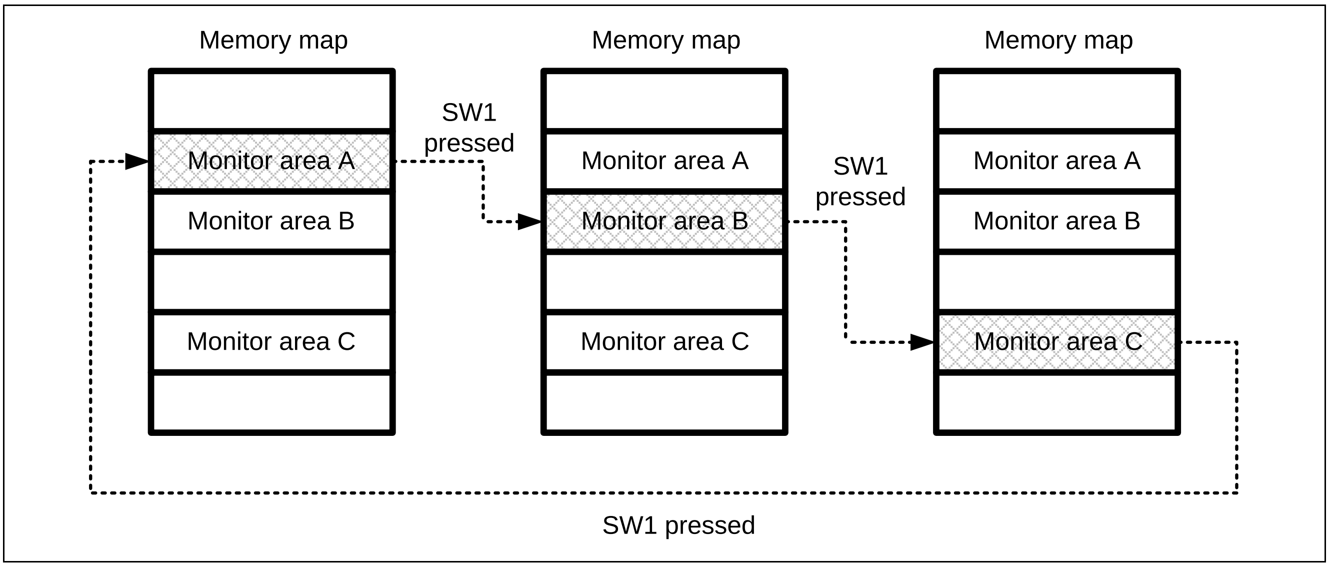

Figure 5.3 SW1 Pressed

(2) SW2 Pressed

Each press of SW2 toggles between the area search function enabled and disabled states. When the area search function is enabled, the area search function is used on the addresses specified beforehand to be accessed. Access control violations can be avoided, and accesses limited to permitted areas only, by checking the access-control information. Figure 5.4 illustrates the operation when SW2 is pressed.

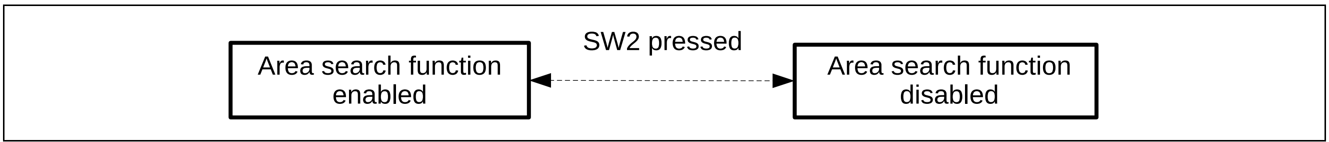

Figure 5.4 SW2 Pressed

(3) Program Status Display

Figure 5.5 shows the program status display indications in the access method setting mode.

Access Area		Area Search Function	Access Result			
LED0	LED1	LED2	LED3	Access Area	Area Search Function	Access Result
Blinking	Blinking	Blinking	Dark	Monitor area A	Enabled	—
Blinking	Blinking	Dark	Dark		Disabled	—
Blinking	Dark	Blinking	Dark	Monitor area B	Enabled	—
Blinking	Dark	Dark	Dark		Disabled	—
Dark	Blinking	Blinking	Dark	Monitor area C	Enabled	—
Dark	Blinking	Dark	Dark		Disabled	—

Lit ○
Dark ●
Blinking

Figure 5.5 Program Status Display Indications in Access Method Setting Mode

5.1.2 Access Result Display Mode

(1) SW3 Pressed

When SW3 is pressed in access method setting mode, the program enters access result display mode, performs an access of the specified type, and displays the access result. Figure 5.6 illustrates the operation when SW3 is pressed.

If no memory-protection error occurs, pressing SW1 or SW2 in access result display mode causes the program to return to access method setting mode. Pressing SW1 switches the area to be accessed, and pressing SW2 toggles the area search function enabled/disabled state, after which the program enters access method setting mode.

If a memory-protection error occurs, pressing SW3 in access result display mode causes the program to retry the same access method, after which the access result is displayed.

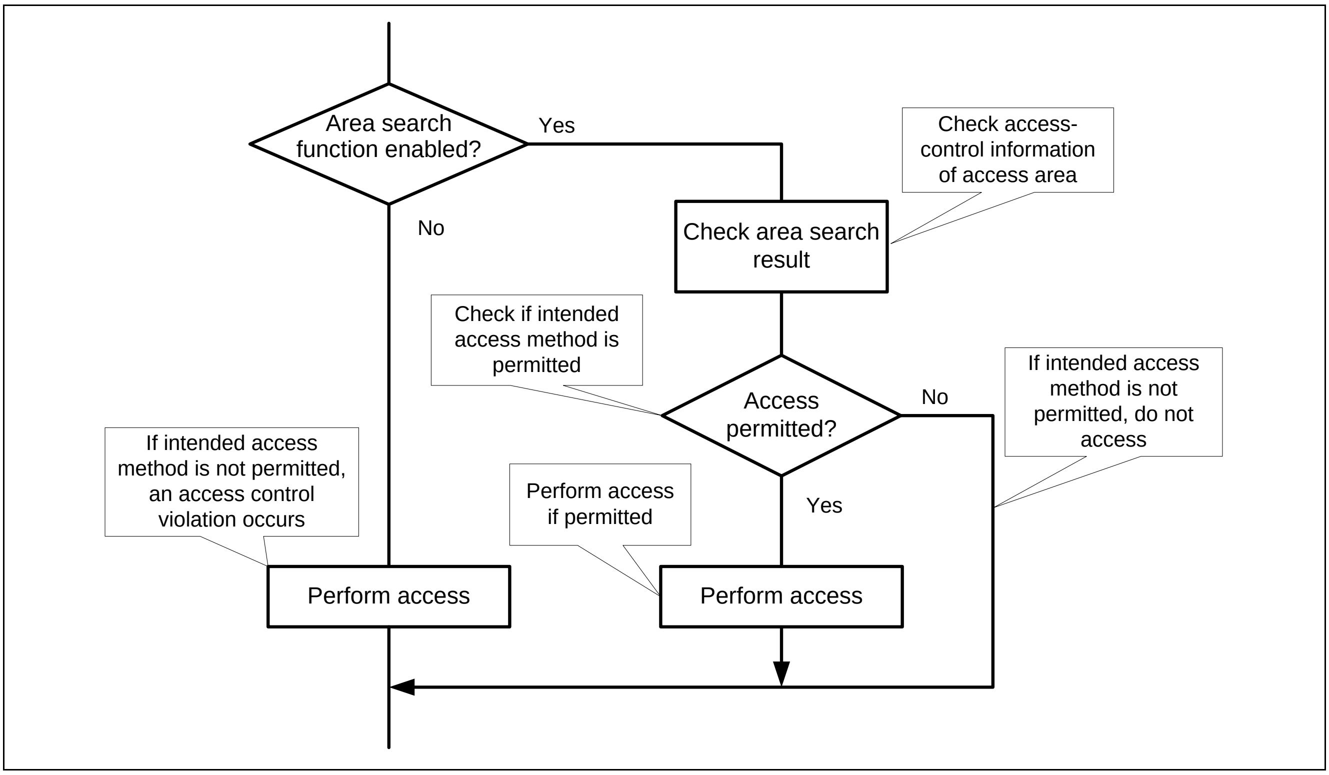

Figure 5.6 SW3 Pressed

(2) Program Status Display

Figure 5.7 shows the program status display indications in access result display mode.

Access Area		Area Search Function	Access Result			
LED0	LED1	LED2	LED3	Access Area	Area Search Function	Access Result
○	○	○	⊗	Monitor area A	Enabled	Prohibited access type detected, avoided.
○	○	○	●			Permitted access type performed.
○	○	●	○		Disabled	Error occurred.
○	○	●	●			No error.
○	●	○	⊗	Monitor area B	Enabled	Prohibited access type detected, avoided.
○	●	○	●			Permitted access type performed.
○	●	●	○		Disabled	Error occurred.
○	●	●	●			No error.
●	○	○	⊗	Monitor area C	Enabled	Prohibited access type detected, avoided.
●	○	○	●			Permitted access type performed.
●	○	●	○		Disabled	Error occurred.
●	○	●	●			No error.

Lit ○

Dark ●

Blinking ⊗

Figure 5.7 Program Status Display Indications in Access Result Display Mode

5.2 File Composition

Table 5.2 lists the files containing the sample code. Note that files that are generated automatically by the integrated development environment and are used without modification are not listed. (Changes resulting from alterations to typedefine.h are also omitted.)

Table 5.2 Files Used in the Sample Code

File Name	Outline	Remarks
main.h	Definition file	
resetprg.c	Initial setting processing after reset	
main.c	Main processing routine	
initialize.c	Initial settings	
vect.h	Interrupt handler declarations	Excep_Memory_ProtectionError function declaration added.
vecttbl.c	Fixed vector interrupt vector table	Excep_Memory_ProtectionError function added to access exception handler.
intprg.c	Vector function definitions • Unconditional trap exception handling • CMT0 compare match exception handling	This file is generated automatically by the integrated development environment and contains the code of various interrupt handlers. In the example program presented in this application note, the Excep_Memory_ProtectionError function is deleted from this file and the functions Excep_BRK and Excep_CMTU0_CMT0 are modified.
Excep_Memory_ProtectionError.src	Access exception handler	Reads the instruction address when a memory-protection error occurs.
MPU_Error.c	Error information storage and error display when memory-protection error occurs	
stacksct.h	Stack size setting	
iodefine_mpu.h	Definition file of registers related to memory-protection function	Added because MPU-related registers are not included in iodefine.h version 2.0.

5.3 Constants

Table 5.3 and Table 5.4 list the constants used in the sample code.

Table 5.3 Constants Used in Sample Code (1)

Constant Name	Setting Value	Contents
READ_ACCESS	00000008h	Sample code for read access
WRITE_ACCESS	00000004h	Sample code for write access
EXECUTE_ACCESS	00000002h	Sample code for execute access
PRG_TYPE	READ_ACCESS	Selects sample code for read access. (See table 5.1.)
SETUP_STATUS	0	Sample code execution is in access method setting mode.
RESULT_STATUS	1	Sample code execution is in access result display mode.
SET	1	Set to 1.
CLEAR	0	Clear to 0.
NO_INPUT	0	No switch input
PUSH_DOWN_SW1	01h	SW1 pressed
PUSH_DOWN_SW2	02h	SW2 pressed
PUSH_DOWN_SW3	80h	SW3 pressed
SW_VALID	1	Indicates valid switch press.
SW_CHAT_TIMES	3	Comparison count to determine valid switch input signal
SW_FIX	0	Switch input final determination
ERROR_STATUS_CLEAR	01h	Clears memory-protection error status register.
NO_ERROR	0	No memory-protection error generated.
MPU_ERROR_MASK	00000007h	Mask for memory-protection error status
READ_ERROR	00000002h	Operand access memory-protection error (read) generated.
WRITE_ERROR	00000006h	Operand access memory-protection error (write) generated.
EXECUTE_ERROR	00000001h	Instruction memory-protection error generated.
RWE_MASK	0000000Eh	Mask for read, write, execute enable/disable
READ_MASK	00000008h	Mask for read enable/disable
WRITE_MASK	00000004h	Mask for write enable/disable
EXECUTE_MASK	00000002h	Mask for execute enable/disable
ALL_ENABLE	0000000Eh	Read, write, execute enabled.
READ_ENABLE	00000008h	Read enabled.
WRITE_ENABLE	00000004h	Write enabled.
EXECUTE_ENABLE	00000002h	Execute enabled.
DISABLE	0	Disabled.
AREA_SEARCH_ENABLE	0001h	Area search function enabled.
AREA_A	0	Monitor area A
AREA_B	AREA_A + 1	Monitor area B
AREA_C	AREA_B + 1	Monitor area C

Table 5.4 Constants Used in Sample Code (2)

Constant Name	Setting Value	Contents
LED0_MASK	04h	Mask for LED0 control
LED1_MASK	08h	Mask for LED1 control
LED2_MASK	20h	Mask for LED2 control
LED3_MASK	10h	Mask for LED3 control
SET_BLINK_CNT	20	LED blink period setting
BLINK_TIMING	0	LED blink timing determination
MPU_REG_MASK	FFFFFFF0h	Mask for MPU register write
SCOPE_ICLK	00000000h	Sets I clock multiplication ratio to ×8.
SCOPE_PCLK	00000100h	Sets P clock multiplication ratio to ×4.
SCOPE_BCLK	00020000h	Sets B clock multiplication ratio to ×2.
LED3_ACTIVE	10h	Sets to output bit corresponding to LED3.
SET_PORT0_DDR	2Ch	Sets to output ports for LED0, LED1, and LED2; sets to input ports for SW1, SW2, and SW3.
TURN_OFF_LED_0_1_2	2Ch	Turns off LED0, LED1, and LED2.
TURN_OFF_LED_3	10h	Turns off LED3.
ENABLE_INPUT_BUF_SW1	01h	Enables input control buffer for SW1.
ENABLE_INPUT_BUF_SW2	02h	Enables input control buffer for SW2.
ENABLE_INPUT_BUF_SW3	80h	Enables input control buffer for SW3.
ACCESS_CONTROL_INF0	0Dh	Memory-protection area 0 access-control information
ACCESS_CONTROL_INF1	0Dh	Memory-protection area 1 access-control information
ACCESS_CONTROL_INF2	03h	Memory-protection area 2 access-control information
ACCESS_CONTROL_INF3	0Dh	Memory-protection area 3 access-control information
ACCESS_CONTROL_INF4	09h	Memory-protection area 4 access-control information
ACCESS_CONTROL_INF5	05h	Memory-protection area 5 access-control information
ACCESS_CONTROL_INF6	03h	Memory-protection area 6 access-control information
IO_DEFINE_STARTADDRESS	0008C000h	I/O register start address in user mode
IO_DEFINE_ENDADDRESS	0008C100h	I/O register end address in user mode
BACKGROUND_ACCESS_CONTROL	0000h	Background area access-control information
MPU_ENABLE	0001h	MPU enabled.
E_OK	0	MPU register successful write
WRITE_FAULT	1	MPU register write failure
SYSTEM_TIME	3A97h	System timer period setting (10 ms)

5.4 Variables

Table 5.5 lists the global variables, Table 5.6 the static variables, and Table 5.7 the const variables.

Table 5.5 Global Variables

Type	Variable Name	Contents	Function Used
uint32_t	g_rd_isp	Stores the PSW save data when changing from user mode to supervisor mode.	Change_PSW_PM_to_SuperVisorMode, Excep_BRK
uint32_t	g_mpu_err_status	Stores the memory-protection error status. (NO_ERROR: no error, READ_ERROR: read error, WRITE_ERROR: write error, EXECUTE_ERROR: execute error)	chip_init, output_resultLED, MPU_Error
uint32_t	g_mpu_err_instruction_adrs	Stores the address of an instruction that generated a memory-protection error.	chip_init, Excep_Memory_ProtectionError
uint32_t	g_mpu_err_operand_adrs	Stores the address of an operand access that generated a memory-protection error.	chip_init, MPU_Error
uint32_t	g_mpu_err_area	Stores the area that generated a memory-protection error and the access-control information of the area.	chip_init, MPU_Error
uint32_t	g_mpu_hit_area	Stores the area where a search hit occurred for an address with the area search function and the access-control information of the area.	chip_init, Memory_Protection, output_resultLED, Area_Access
uint8_t	g_access_area	Stores the area data for an access. (AREA_A: monitor area A, AREA_B: monitor area B, AREA_C: monitor area C)	chip_init, Memory_Protection, output_setupLED, Area_Access_Select
uint8_t	g_area_search_flg	Stores the area search function enabled/disabled status. (0: Disabled, 1: Enabled)	chip_init, Memory_Protection, output_setupLED, output_resultLED, Area_Access
uint8_t	g_system_tmr_flg	Stores the system timer period flag. (0: Waiting for period to elapse, 1: Period elapsed)	chip_init, main, Excep_CMTU0_CMT0

Table 5.6 static Variables

Type	Variable Name	Contents	Function Used
static uint8_t	g_sw_status	Stores finalized switch input data.	main, get_sw, Memory_Protection
static uint8_t	g_last_sw	Stores previous switch input data.	main, get_sw
static uint8_t	g_sw_chat_cnt	Switch chattering elimination counter	main, get_sw
static uint8_t	g_sw_update	Stores switch input data finalization update status. (0: Not updated, 1: Updated)	main, get_sw, Memory_Protection, control_LED
static uint8_t	g_led_cycle_cnt	LED blink counter	main, control_LED
static uint8_t	g_prg_status	Stores program status.	main, Memory_Protection, control_LED
uint32_t	g_Area_A_ operand	Monitor area A operand	areaAccess_Select
uint32_t	g_Area_B_ operand	Monitor area B operand	areaAccess_Select

Table 5.7 const Variable

Type	Variable Name	Contents	Function Used
uint32_t	g_Area_C_ operand	Monitor area C operand	areaAccess_Select

5.5 Functions

Table 5.8 lists the functions.

Table 5.8 Functions

Function Name	Outline
PowerON_Reset_PC	Startup program
Main	Main processing routine
get_sw	Switch input detect
Memory_Protection	Access area and access method setting and access control
control_LED	Valid switch input LED display update and LED blink update
output_setupLED	Access method setting LED control
output_resultLED	Access result display LED control
areaAccess_Select	Specified access area determination
Area_Access	Function for accessing specified access area
areaSearch	Acquisition of area corresponding to address specified by area search function and access-control information of area
operand_Access_Read	Function for reading specified operand
operand_Access_Write	Function for writing to specified operand
Area_A_prg	Function for executing program in monitor area A
Area_B_prg	Function for executing program in monitor area B
Area_C_prg	Function for executing program in monitor area C
Change_PSW_PM_to_UserMode	Change from supervisor mode to user mode
Change_PSW_PM_to_SuperVisorMode	Change from user mode to supervisor mode
init	Call initial settings functions
chip_init	MCU initial settings, operating clock settings, RAM initialization
mpu_init	MPU initialization
verify_mpu_reg	Verify MPU-related registers
cmt0_init	CMT0 settings
Excep_BRK	Unconditional trap exception handler
Excep_Memory_ProtectionError	Memory-protection error access exception handler
MPU_Error	Memory-protection error information storage and error display
Excep_CMTU0_CMT0	CMT0 compare match exception handler

5.6 Function Specifications

The specifications of the sample code functions are listed below.

PowerON_Reset_PC	
Outline	Startup program
Header	None
Declaration	void PowerON_Reset_PC(void)
Description	The PowerON_Reset_PC function is called after a reset. It uses embedded functions and standard library functions to set the CPU registers. Then it calls the init function and main function.
Arguments	None
Return Value	None

main	
Outline	Main processing routine
Header	main.h
Declaration	void main(void)
Description	This function calls the Change_PSW_PM_to_UserMode function, get_sw function, Memory_Protection function, and control_LED function.
Arguments	None
Return Value	None

get_sw		
Outline	Switch input detect	
Header	main.h	
Declaration	static uint8_t get_sw(uint8_t *p_last_sw, uint8_t *p_sw_chat_cnt, uint8_t *p_sw_status)	
Description	This function detects input from the switches.	
Arguments	uint8_t *p_last_sw	: Previous switch input data
	uint8_t *p_sw_chat_cnt	: Switch input signal match counter
	uint8_t *p_sw_status	: Finalized switch input data
Return Value	The switch input valid/invalid state is returned. CLEAR: Switch input invalid SW_VALID: Switch input valid	

Memory_Protection	
Outline	Access area and access method setting and access control
Header	main.h
Declaration	static void Memory_Protection(uint8_t sw_status, uint8_t sw_update, uint8_t *p_prg_status)
Description	Based on the switch input state indicated by the arguments, this function calls functions to switch the access area, switch the area search function enable/disable state, and access the specified area.
Arguments	uint8_t sw_status : Finalized switch input data uint8_t sw_update : Switch input valid/invalid uint8_t *p_prg_status : Program status
Return Value	None

control_LED	
Outline	Valid switch input LED display update and LED blink update
Header	main.h
Declaration	static void control_LED(uint8_t sw_update, uint8_t prg_status, uint8_t *p_led_cycle_cnt)
Description	This function controls the LEDs to display the program status, as indicated by the arguments. It also controls LED blinking.
Arguments	uint8_t sw_update : Switch input valid/invalid uint8_t prg_status : Program status uint8_t *p_led_cycle_cnt : LED blink control timer
Return Value	None

output_setupLED	
Outline	Access method setting LED control
Header	main.h
Declaration	void output_setupLED(void)
Description	This function performs access method setting LED control.
Arguments	None
Return Value	None

output_resultLED	
Outline	Access result display LED control
Header	main.h
Declaration	void output_resultLED (uint8_t prg_status, uint8_t *p_led_cycle_cnt)
Description	This function performs access result display LED control.
Arguments	uint8_t prg_status : Program status uint8_t *p_led_cycle_cnt : LED blink control timer
Return Value	None

areaAccess_Select	
Outline	Specified access area determination
Header	main.h
Declaration	static void areaAccess_Select (void)
Description	This function determines the access area and calls the Area_Access function.
Arguments	None
Return Value	None

areaAccess		
Outline	Function for accessing specified access area	
Header	main.h	
Declaration	static void Area_Access(uint32_t *p_access_operand, void (*prg)())	
Description	This function calls a function to access the area specified by the arguments.	
Arguments	uint32_t *p_access_operand	: Operand address to be accessed
	void (*prg)()	: Start address of execute access
Return Value	None	

areaSearch		
Outline	Acquisition of area corresponding to address specified by area search function and access-control information of area	
Header	None	
Declaration	static uint32_t areaSearch(uint32_t *p_access_address)	
Description	This function calls the Change_PSW_PM_to_SuperVisorMode function. It determines the area corresponding to the address specified by an argument and gets the access-control information of that area. Then it calls the Change_PSW_PM_to_UserMode function.	
Arguments	uint32_t *p_access_address	: Area search function target address
Return Value	Area corresponding to address specified by argument and access-control information of area	

operand_Access_Read		
Outline	Function for reading specified operand	
Header	None	
Declaration	static void operand_Access_Read(uint32_t *p_access_operand)	
Description	This function performs read access to the address specified by an argument.	
Arguments	uint32_t *p_access_operand	: Address of read access
Return Value	None	

operand_Access_Write		
Outline	Function for writing to specified operand	
Header	None	
Declaration	static void operand_Access_Write(uint32_t *p_access_operand)	
Description	This function performs write access to the address specified by an argument.	
Arguments	uint32_t *p_access_operand	: Address of write access
Return Value	None	

Area_A_prg

Outline	Function for executing program in monitor area A
Header	None
Declaration	static void Area_A_prg(void)
Description	This function executes a program in monitor area A.
Arguments	None
Return Value	None

Area_B_prg

Outline	Function for executing program in monitor area B
Header	None
Declaration	static void Area_B_prg(void)
Description	This function executes a program in monitor area B.
Arguments	None
Return Value	None

Area_C_prg

Outline	Function for executing program in monitor area C
Header	None
Declaration	static void Area_C_prg(void)
Description	This function executes a program in monitor area C.
Arguments	None
Return Value	None

Change_PSW_PM_to_UserMode

Outline	Changing from supervisor mode to user mode
Header	None
Declaration	void Change_PSW_PM_to_UserMode(void)
Description	This function switches the processor mode from supervisor mode to user mode.
Arguments	None
Return Value	None

Change_PSW_PM_to_SuperVisorMode

Outline	Changing from user mode to supervisor mode
Header	None
Declaration	static void Change_PSW_PM_to_SuperVisorMode(void)
Description	This function triggers unconditional trap exception handling. After unconditional trap exception handling ends, the processor mode returns to supervisor mode.
Arguments	None
Return Value	None

init	
Outline	Call initial settings functions
Header	None
Declaration	void init(void)
Description	This function initializes peripheral functions and makes memory-protection area settings, etc., by calling the chip_init function, mpu_init function, cmt0_init function, and output_setupLED function.
Arguments	None
Return Value	None

chip_init	
Outline	MCU initial settings, operating clock settings, RAM initialization
Header	main.h
Declaration	static void chip_init(void)
Description	This function makes clock settings, port settings for using the LEDs and switches, initial settings for the RAM to be used, etc.
Arguments	None
Return Value	None

mpu_init	
Outline	MPU initialization
Header	iodefine_mpu.h
Declaration	static void mpu_init(void)
Description	This function makes memory-protection area settings and enables the MPU function.
Arguments	None
Return Value	None

verify_mpu_reg	
Outline	Verify MPU-related registers
Header	iodefine_mpu.h
Declaration	static uint8_t verify_mpu_reg(volatile __evenaccess uint32_t *target, uint32_t expect_val)
Description	The verify_mpu_reg function confirms that the values written to the MPU registers are correct. It has no direct effect on the operation of the program.
Arguments	volatile __evenaccess uint32_t *p_target : Address of verify target register uint32_t expect_val : Expected value of verified register
Return Value	MPU register write result E_OK: Successful write WRITE_FAULT: Write failure

cmt0_init	
Outline	CMT0 settings
Header	None
Declaration	static void cmt0_init(void)
Description	This function sets the system timer period.
Arguments	None
Return Value	None

Excep_BRK	
Outline	Unconditional trap exception handler
Header	vect.h
Declaration	void Excep_BRK(void)
Description	This function is the unconditional trap exception handler. After exception handling completes, return occurs with the CPU in supervisor mode.
Arguments	None
Return Value	None

Excep_Memory_ProtectionError	
Outline	Memory-protection error access exception handler
Header	vect.h
Declaration	void Excep_Memory_ProtectionError(void)
Description	The access exception handler reads the address of the instruction that triggered the memory-protection error and calls the MPU_Error function.
Arguments	None
Return Value	None

MPU_Error	
Outline	Memory-protection error information storage and error display
Header	vect.h
Declaration	void MPU_Error (void)
Description	After storing the memory-protection error information and displaying an error indication, this function enters an infinite loop.
Arguments	None
Return Value	None

Excep_CMTU0_CMT0	
Outline	CMT0 compare match exception handler
Header	vect.h
Declaration	void Excep_CMTU0_CMT0 (void)
Description	This function is the CMT0 compare match exception handler. It sets the system timer elapsed flag.
Arguments	None
Return Value	None

5.7 Flowcharts

5.7.1 Startup Program

Figure 5.8 is a flowchart of the startup program.

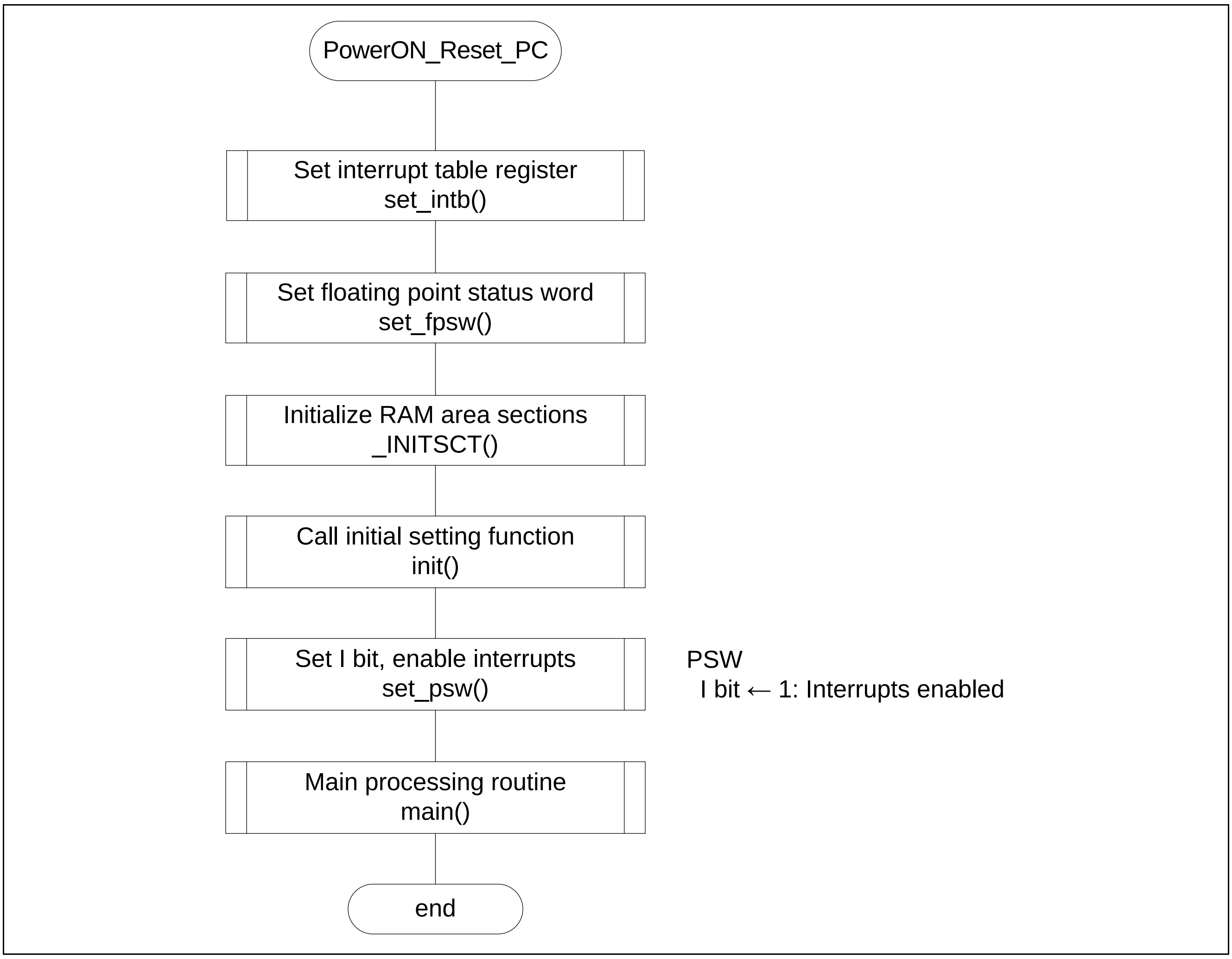

Figure 5.8 Startup Program

5.7.2 Main Processing Routine

Figure 5.9 is a flowchart of the main processing routine.

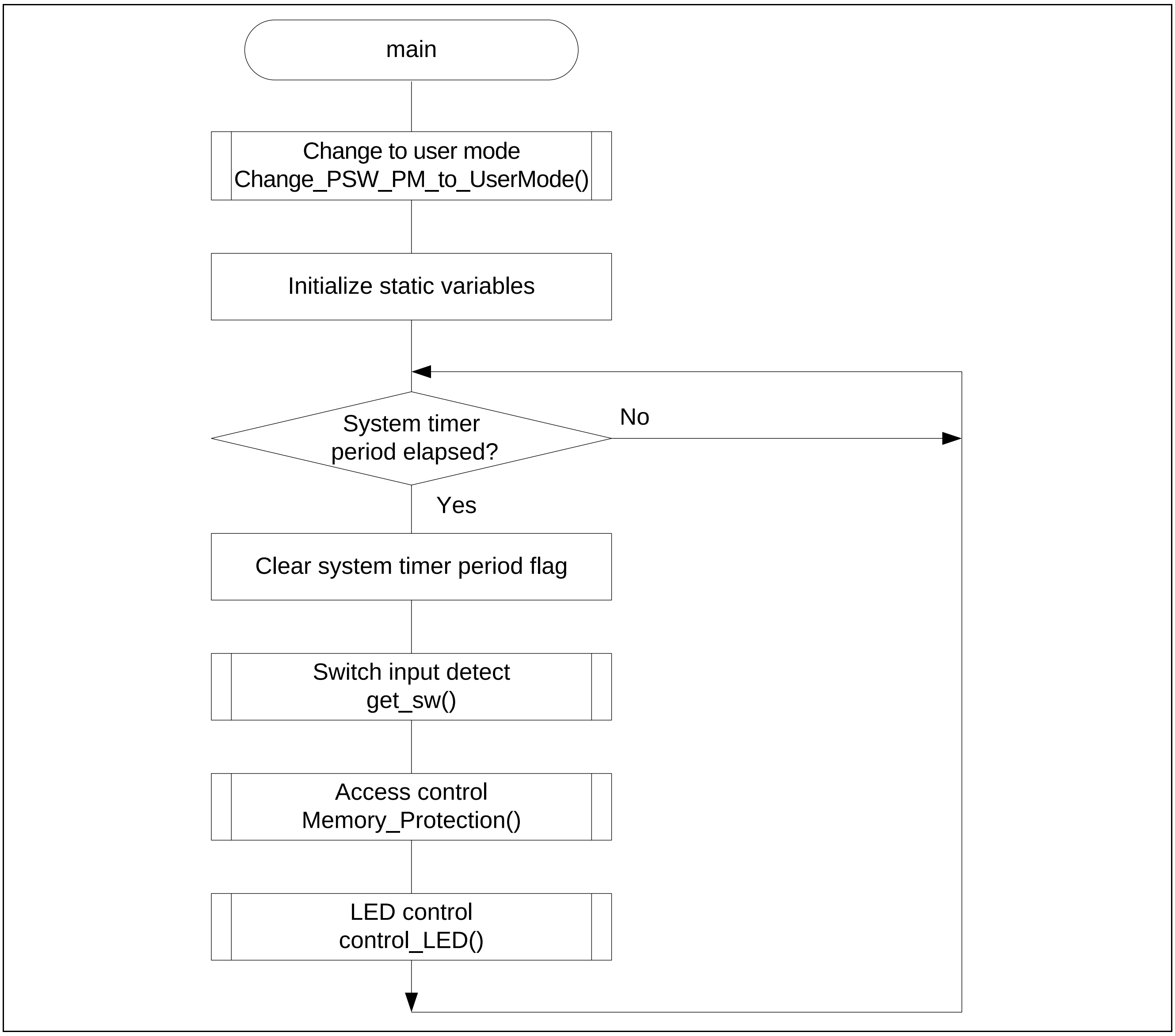

Figure 5.9 Main Processing Routine

5.7.3 Switch Input Detect

Figure 5.10 is a flowchart of the switch input detect function.

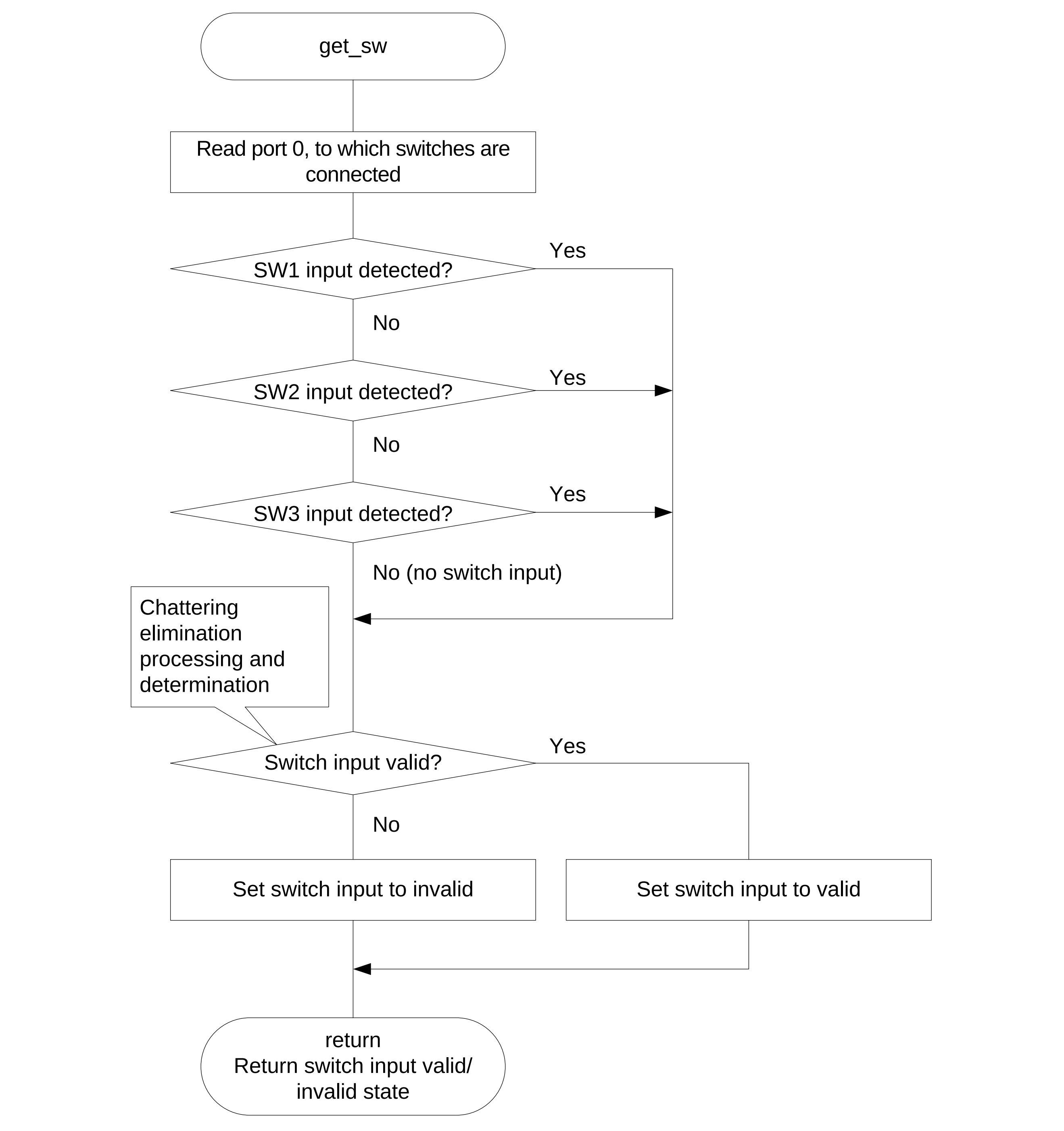

Figure 5.10 Switch Input Detect

5.7.4 Access Area and Access Method Setting and Access Control

Figure 5.11 is a flowchart of the access area and access method setting and access control function.

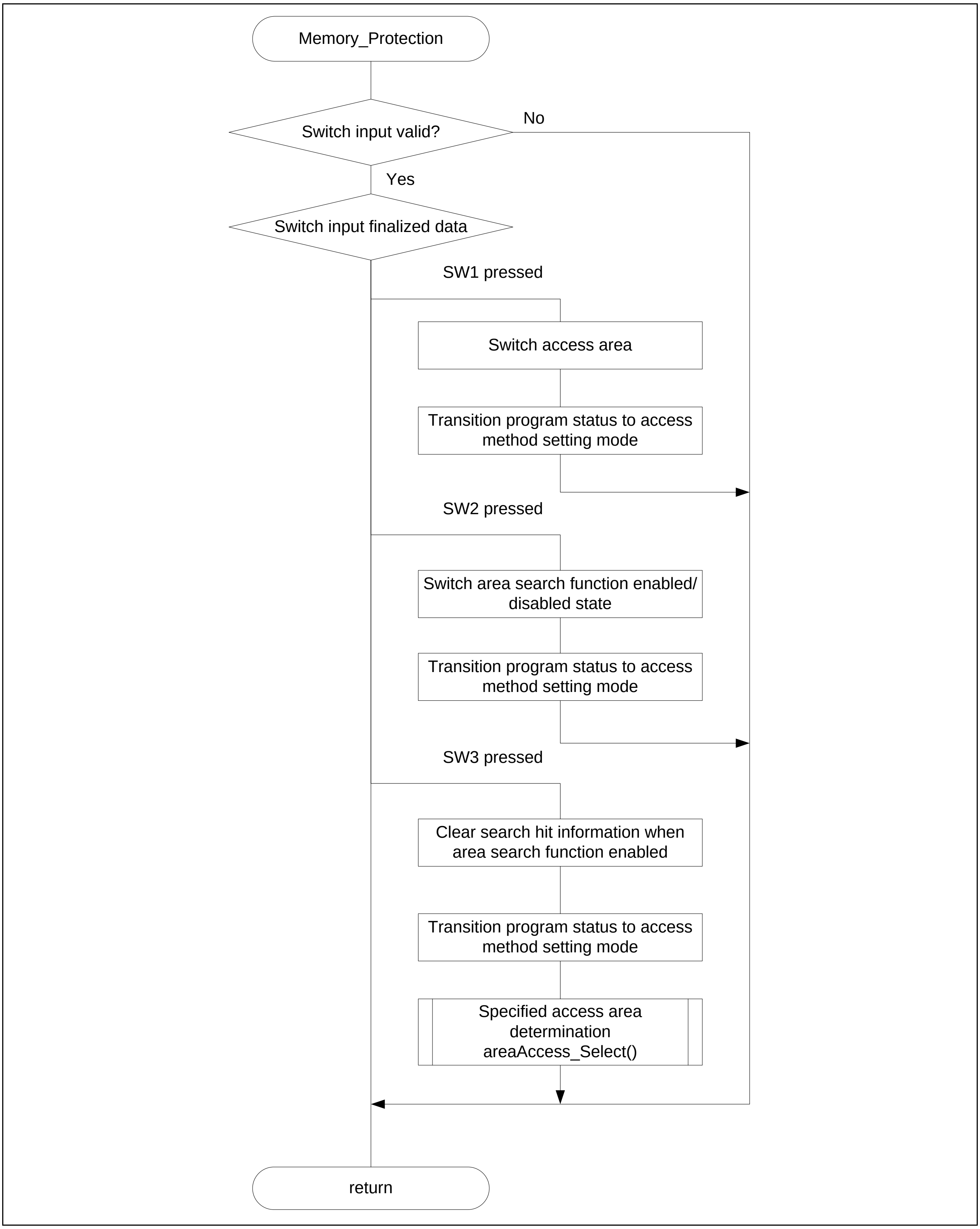

Figure 5.11 Access Area and Access Method Setting and Access Control

5.7.5 Valid Switch Input LED Display Update and LED Blink Update

Figure 5.12 is a flowchart of the valid switch input LED display update and LED blink update function.

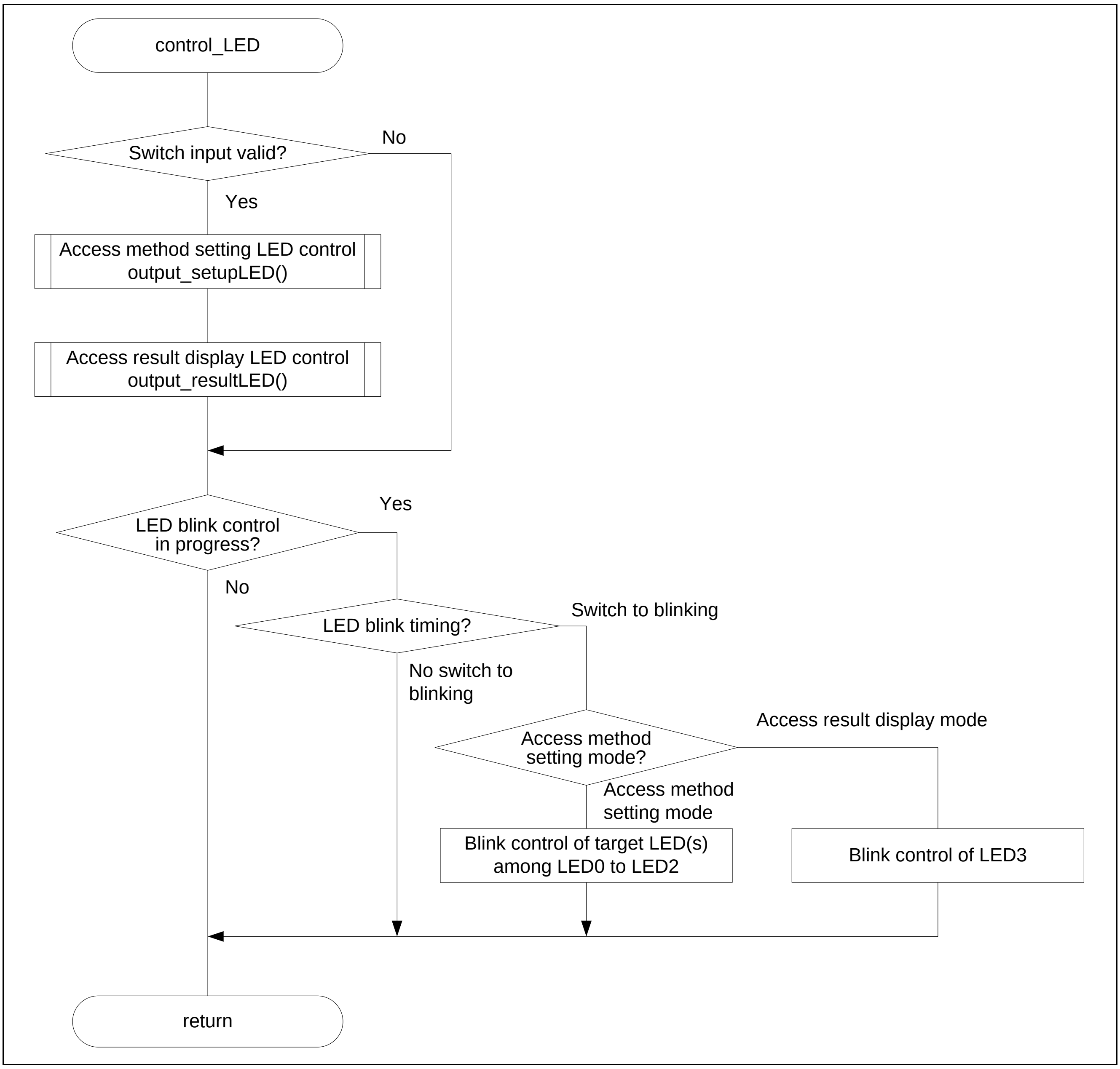

Figure 5.12 Valid Switch Input LED Display Update and LED Blink Update

5.7.6 Access Method Setting LED Control

Figure 5.13 is a flowchart of the access method setting LED control function.

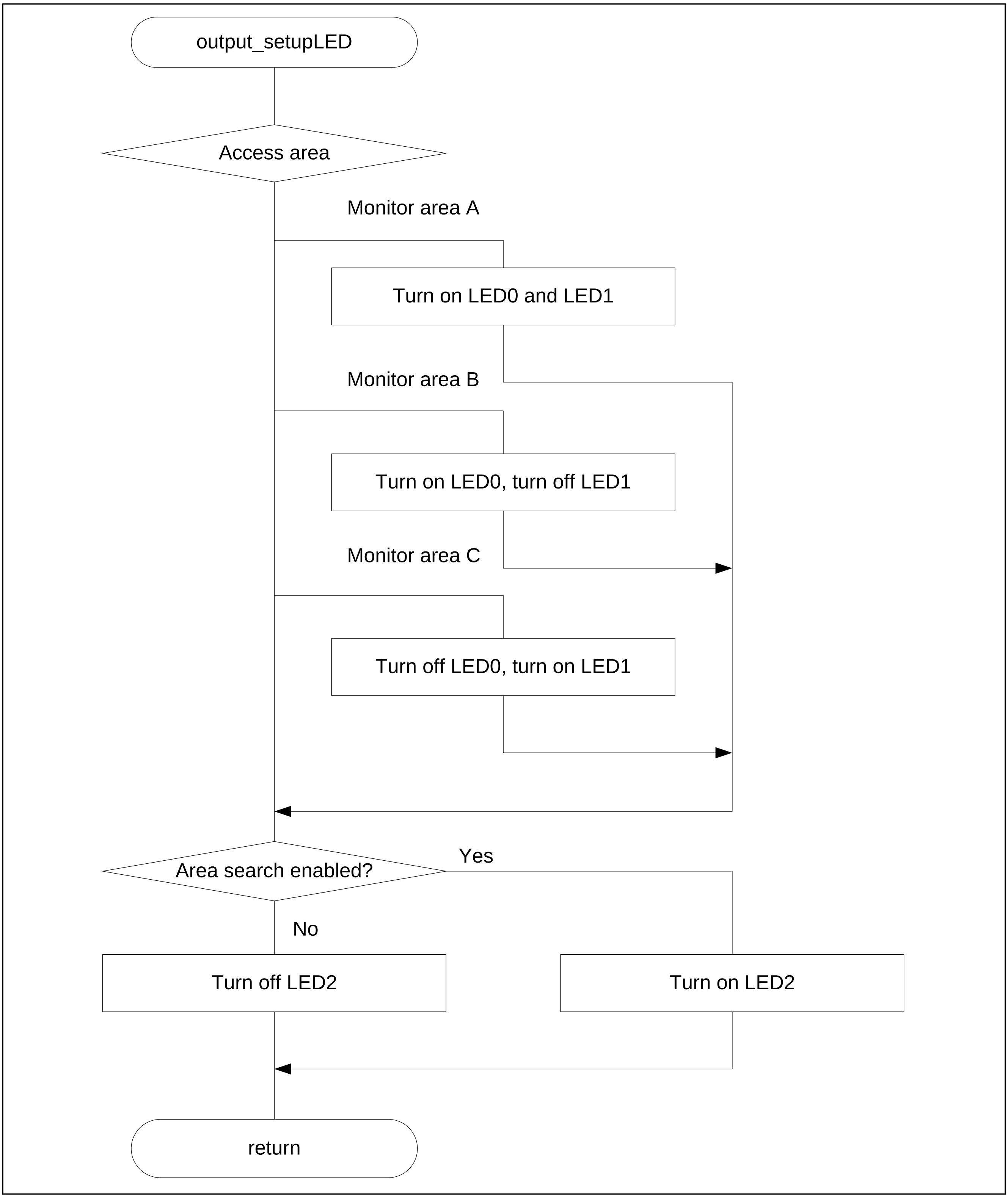

Figure 5.13 Access Method Setting LED Control

5.7.7 Access Result Display LED Control

Figure 5.14 is a flowchart of the access result display LED control function.

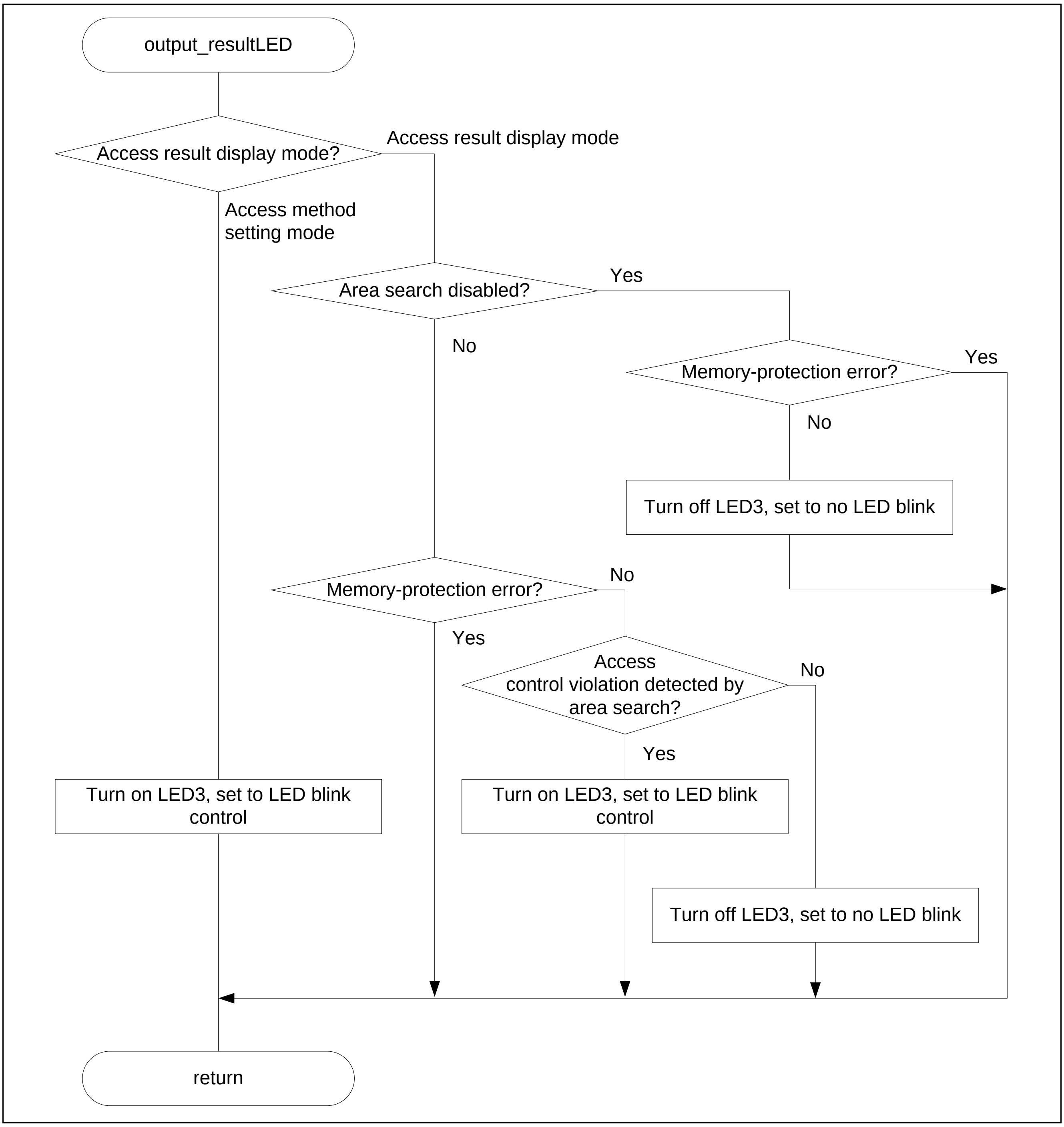

Figure 5.14 Access Result Display LED Control

5.7.8 Specified Access Area Determination

Figure 5.15 is a flowchart of the specified access area determination function.

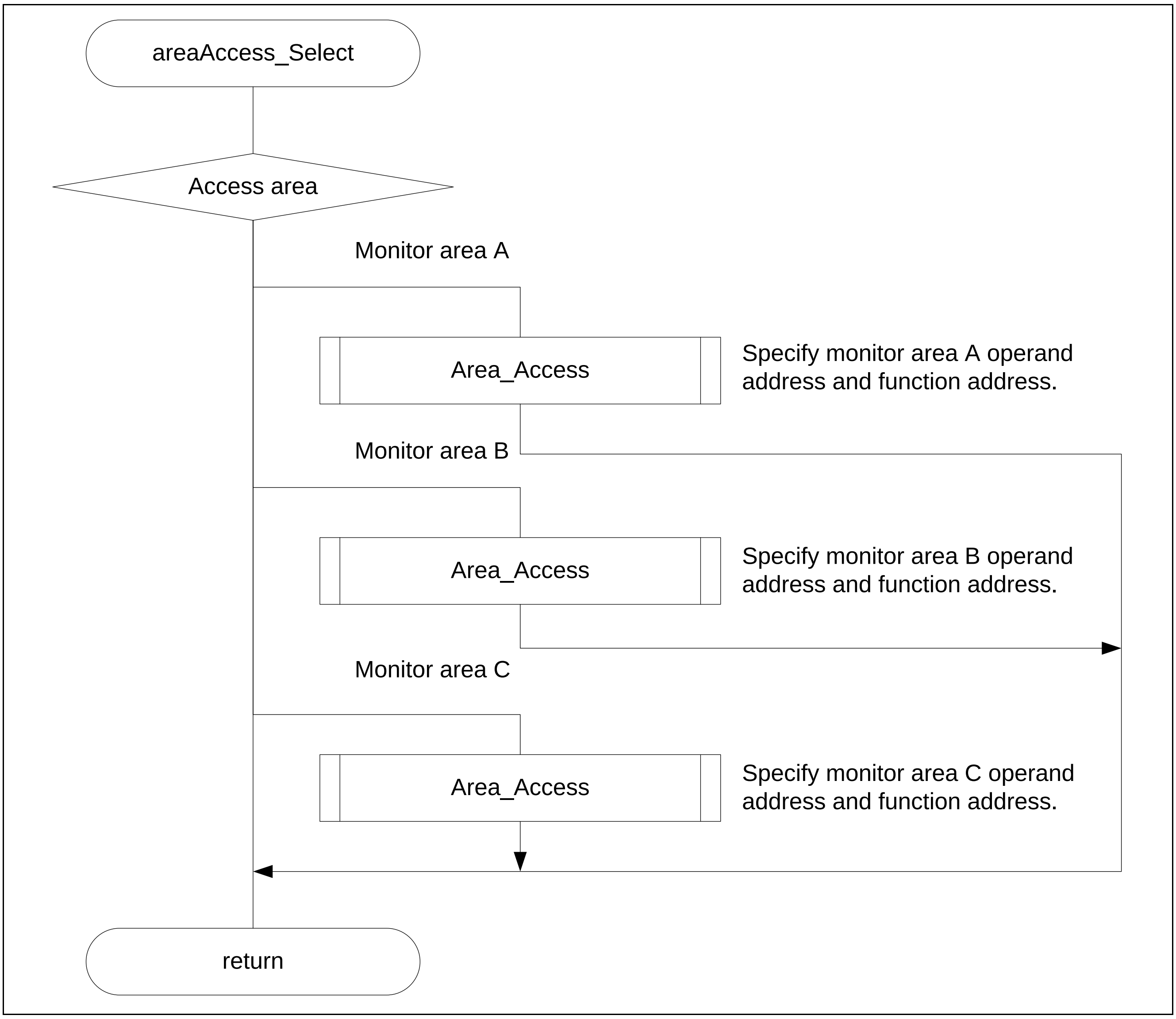

Figure 5.15 Specified Access Area Determination

5.7.9 Function for Accessing Specified Access Area

Conditional compilation is used so that the operation of this function differs according to the correspondence between the PRG_TYPE definition and the resulting sample code operation, as shown in Table 5.1. The conditional compilation conditions are described below.

(1) Read Access to Specified Area

Figure 5.16 is a flowchart of the function for accessing the specified access area (READ_ACCESS), which operates when the selection is fixed at read access to the specified area.

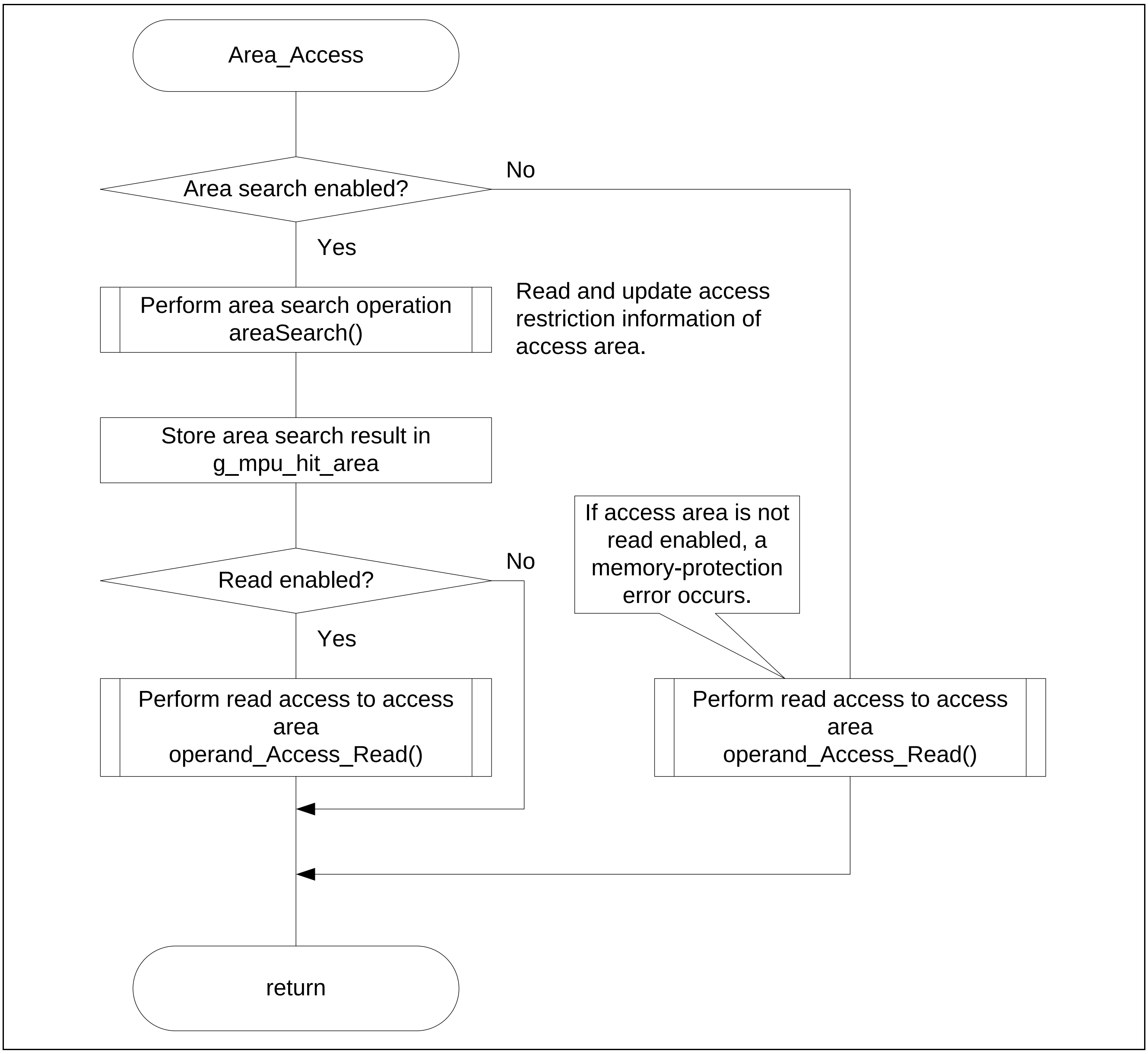

Figure 5.16 Read Access to Specified Area (READ_ACCESS)

(2) Write Access to Specified Area

Figure 5.17 is a flowchart of the function for accessing the specified access area (WRITE_ACCESS), which operates when the selection is fixed at write access to the specified area.

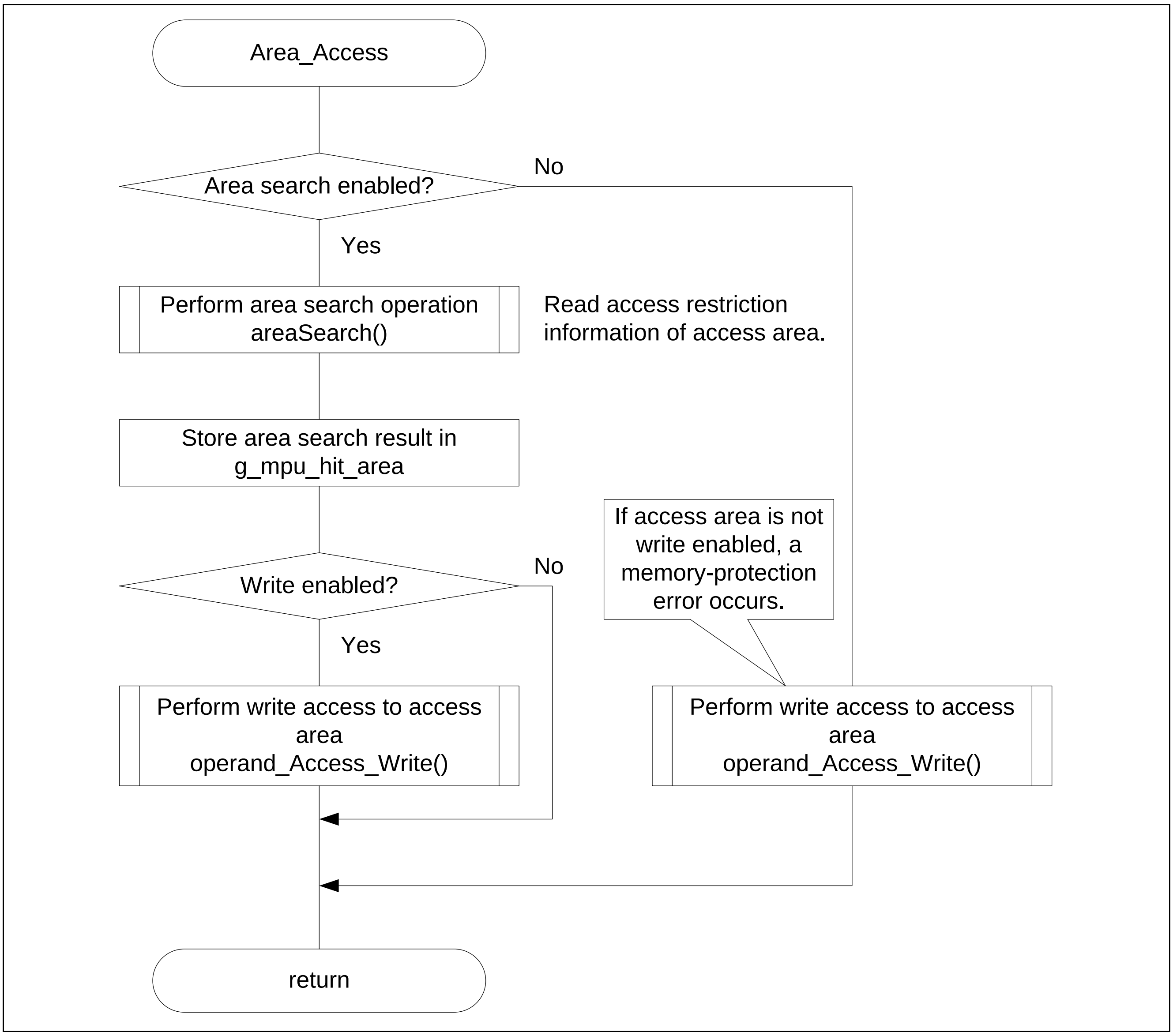

Figure 5.17 Write Access to Specified Area (WRITE_ACCESS)

(3) Execute Access to Specified Area

Figure 5.18 is a flowchart of the function for accessing the specified access area (EXECUTE_ACCESS), which operates when the selection is fixed at execute access to the specified area.

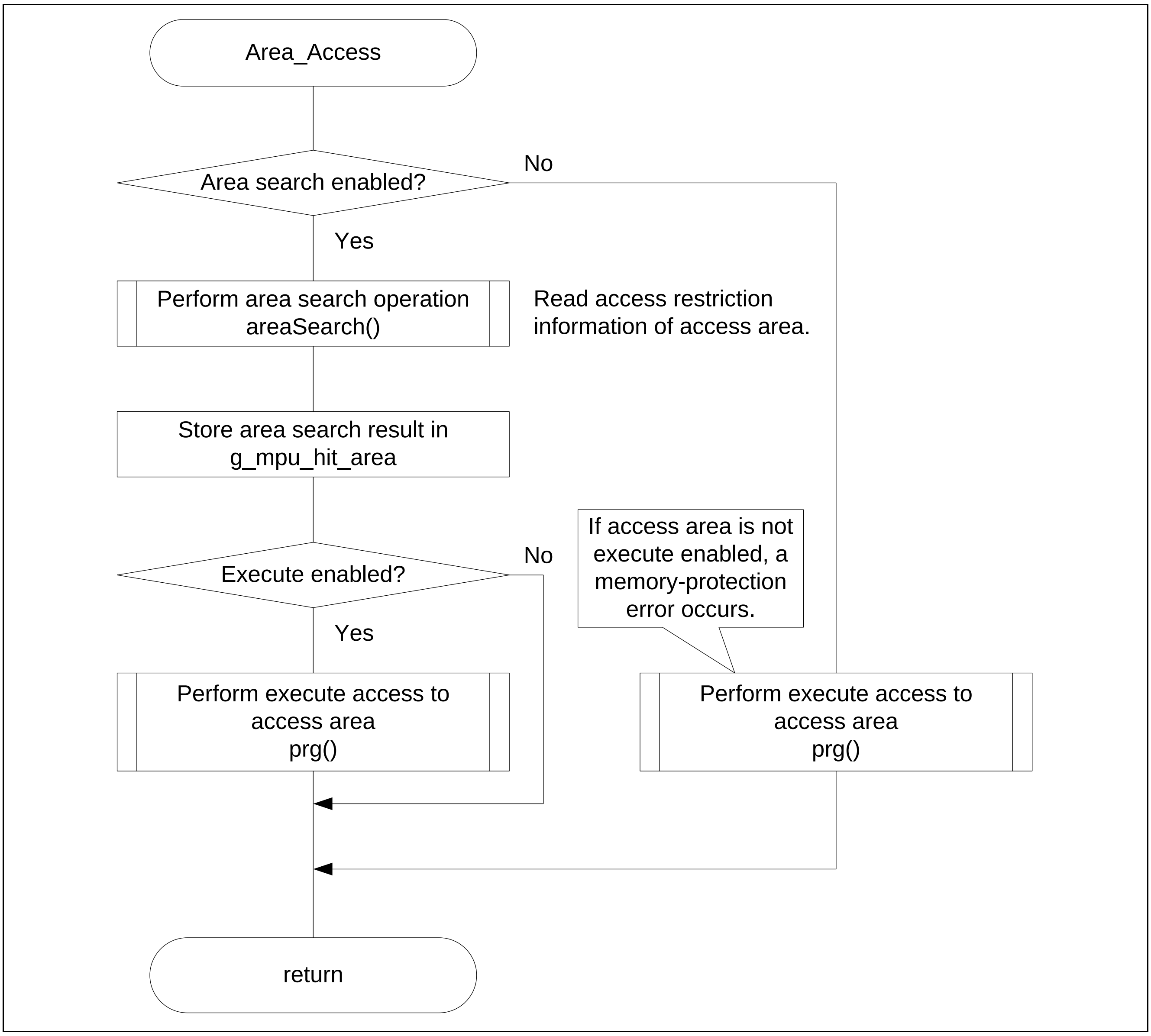

Figure 5.18 Execute Access to Specified Area (EXECUTE_ACCESS)

5.7.10 Acquisition of Area Corresponding to Address Specified by Area Search Function and Access-Control Information of Area

Figure 5.19 is a flowchart of the function for acquisition of the area corresponding to the address specified by the area search function and access-control information of the area.

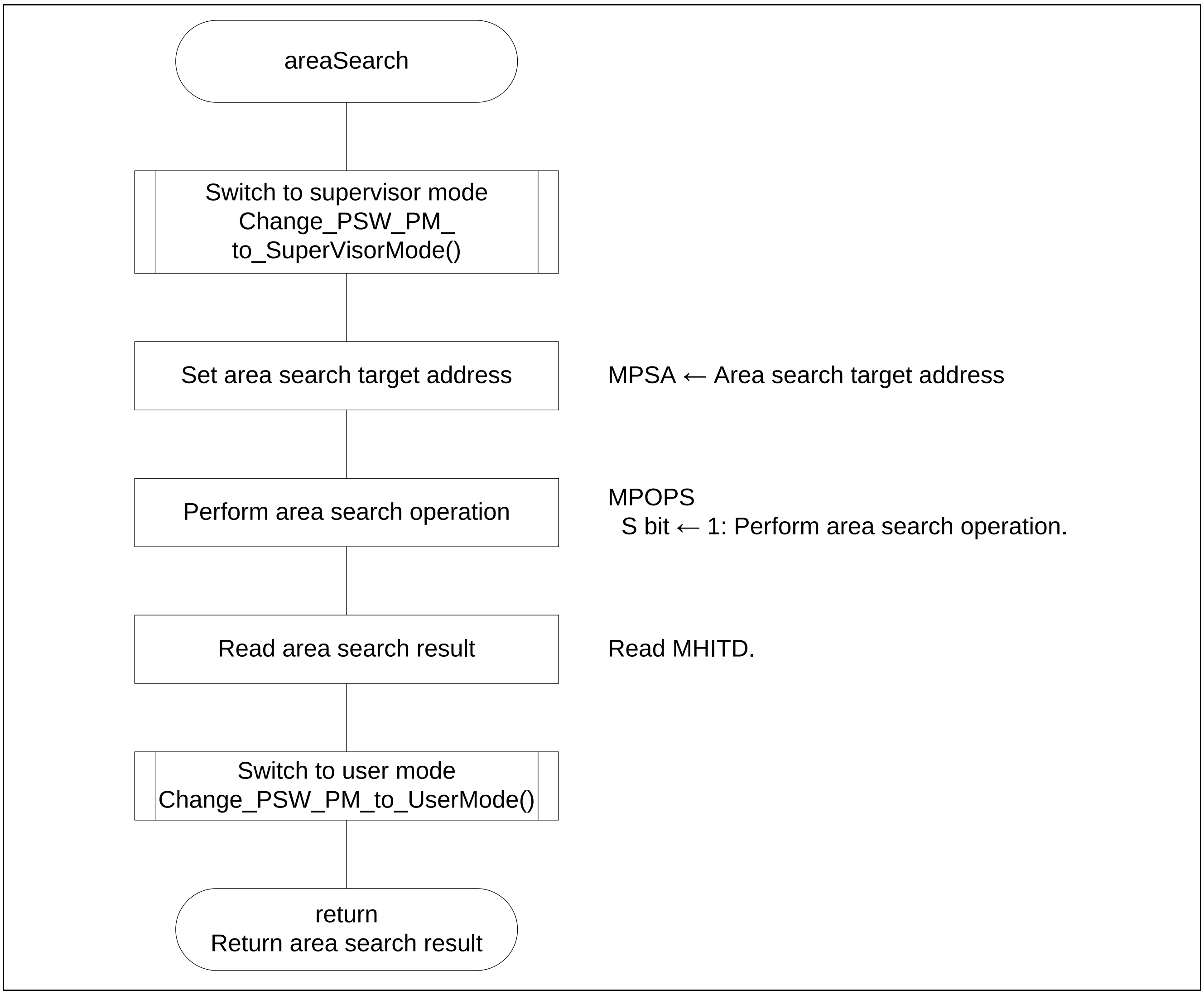

Figure 5.19 Acquisition of Area Corresponding to Address Specified by Area Search Function and Access-Control Information of Area

5.7.11 Function for Reading Specified Operand

Figure 5.20 is a flowchart of the function for reading a specified operand.

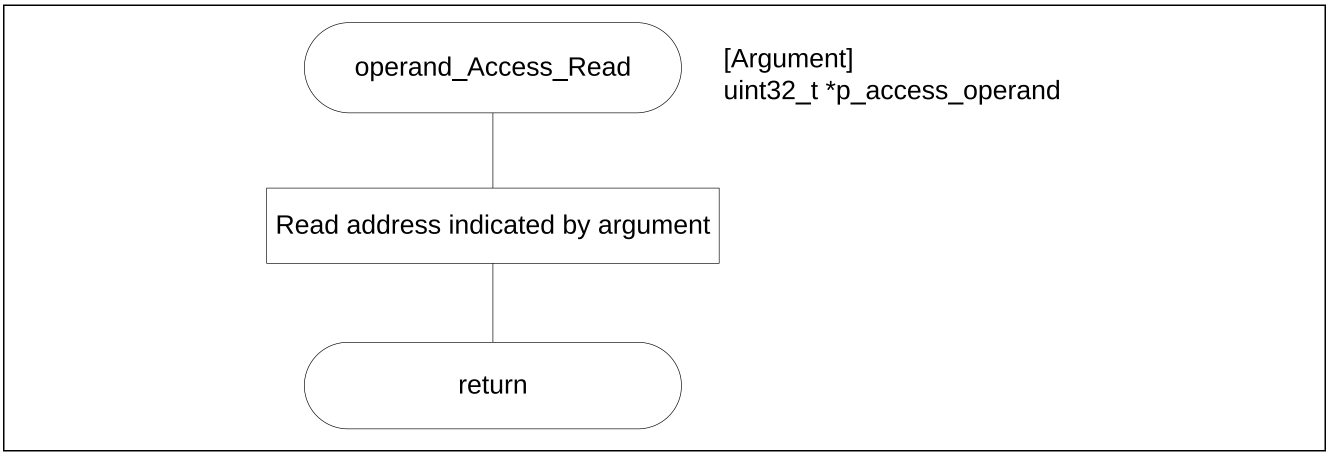

Figure 5.20 Function for Reading Specified Operand

5.7.12 Function for Writing to Specified Operand

Figure 5.21 is a flowchart of the function for writing to a specified operand.

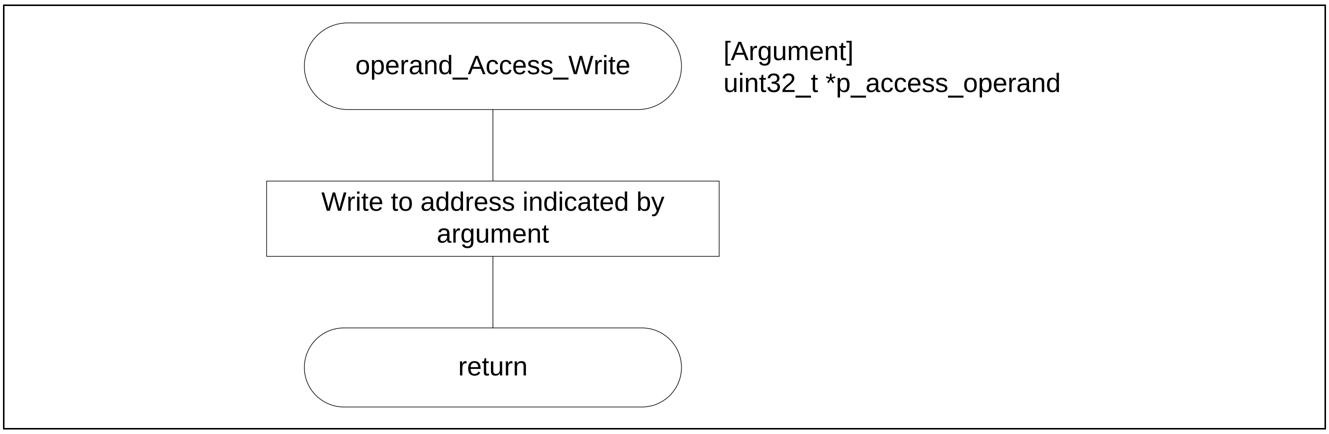

Figure 5.21 Function for Writing to Specified Operand

5.7.13 Function for Executing Program in Monitor Area A

Figure 5.22 is a flowchart of the function for executing a program in monitor area A.

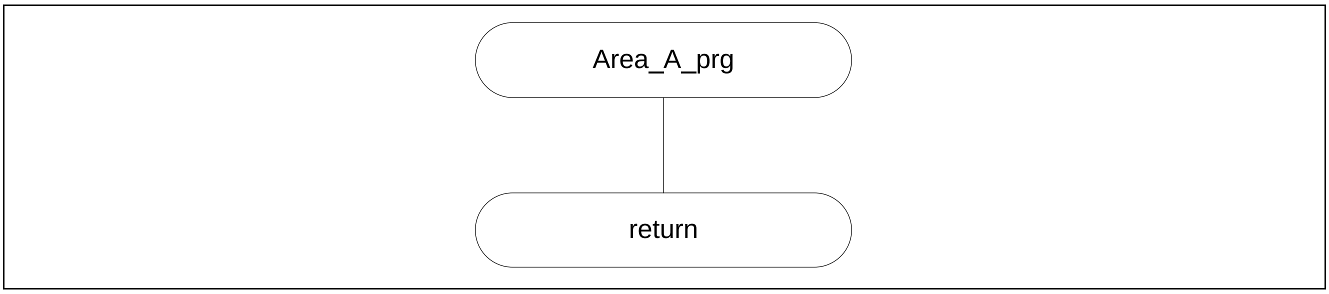

Figure 5.22 Function for Executing Program in Monitor Area A

5.7.14 Function for Executing Program in Monitor Area B

Figure 5.23 is a flowchart of the function for executing a program in monitor area B.

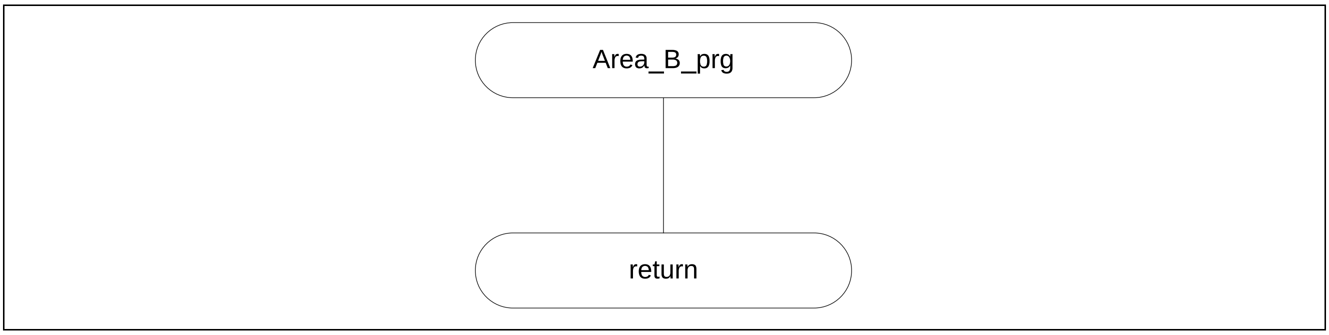

Figure 5.23 Function for Executing Program in Monitor Area B

5.7.15 Function for Executing Program in Monitor Area C

Figure 5.24 is a flowchart of the function for executing a program in monitor area C.

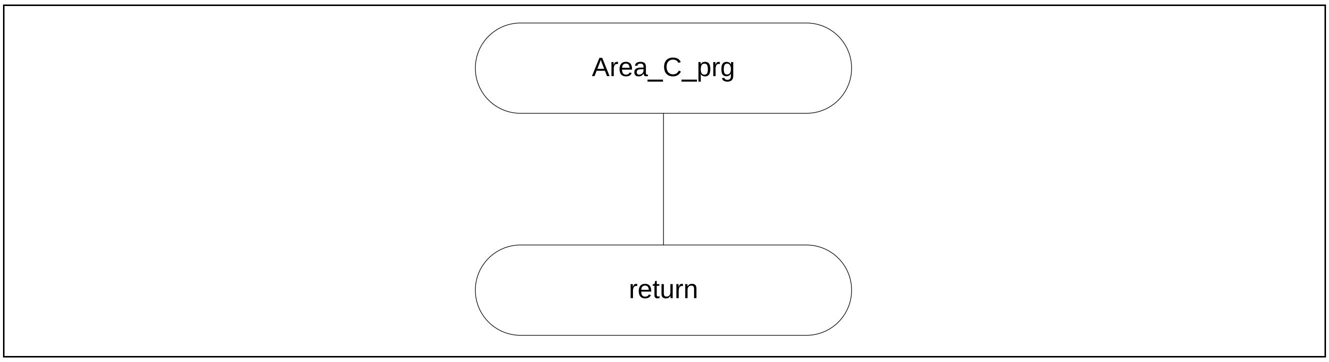

Figure 5.24 Function for Executing Program in Monitor Area C

5.7.16 Change from Supervisor Mode to User Mode

Figure 5.25 is a flowchart of the change from supervisor mode to user mode function.

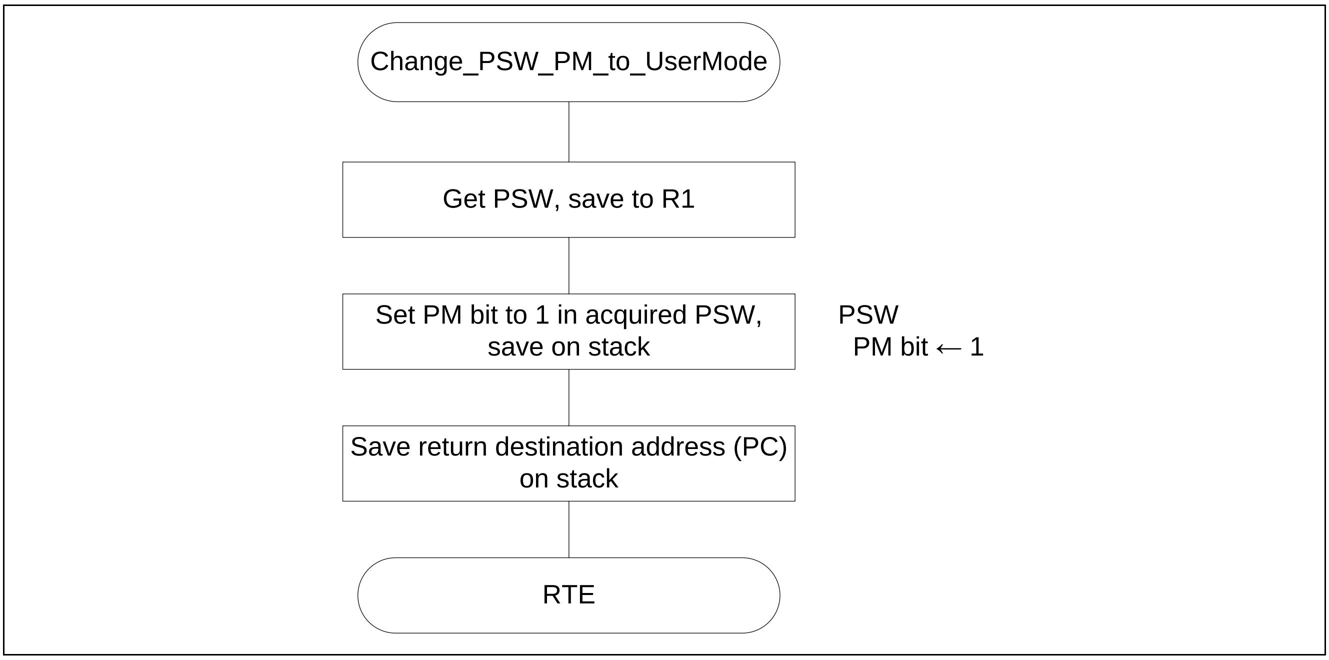

Figure 5.25 Change from Supervisor Mode to User Mode

5.7.17 Change from User Mode to Supervisor Mode

Figure 5.26 is a flowchart of the change from user mode to supervisor mode function.

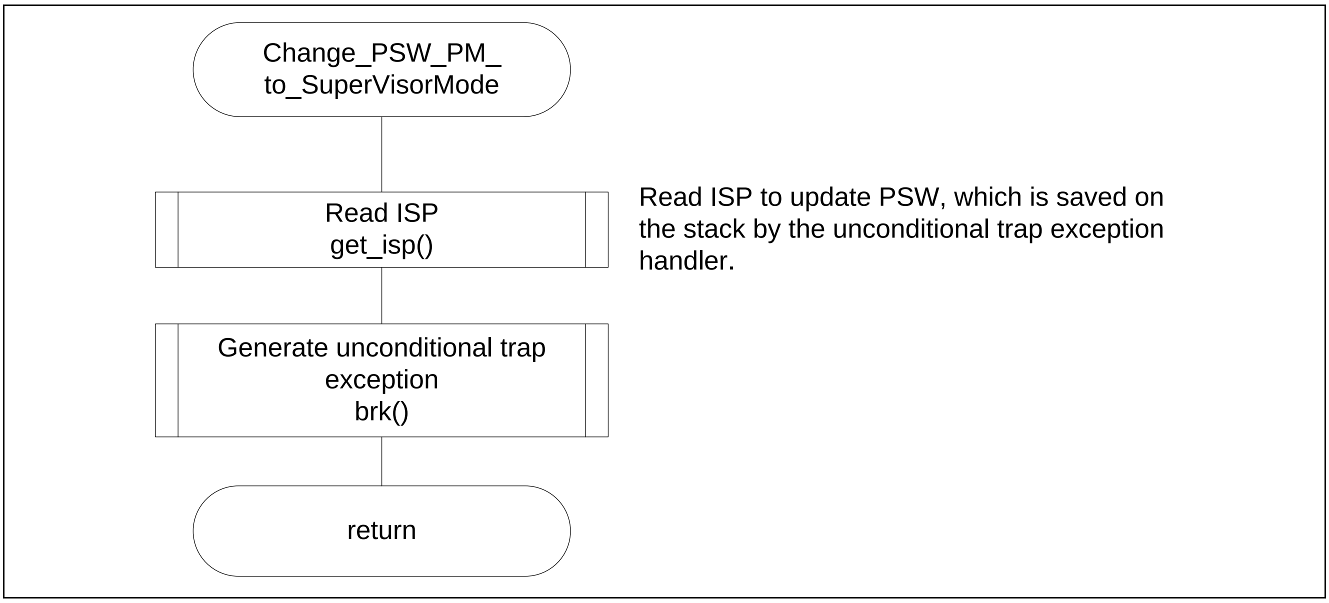

Figure 5.26 Change from User Mode to Supervisor Mode

5.7.18 Call Initial Settings Functions

Figure 5.27 is a flowchart of the function that calls the initial settings functions.

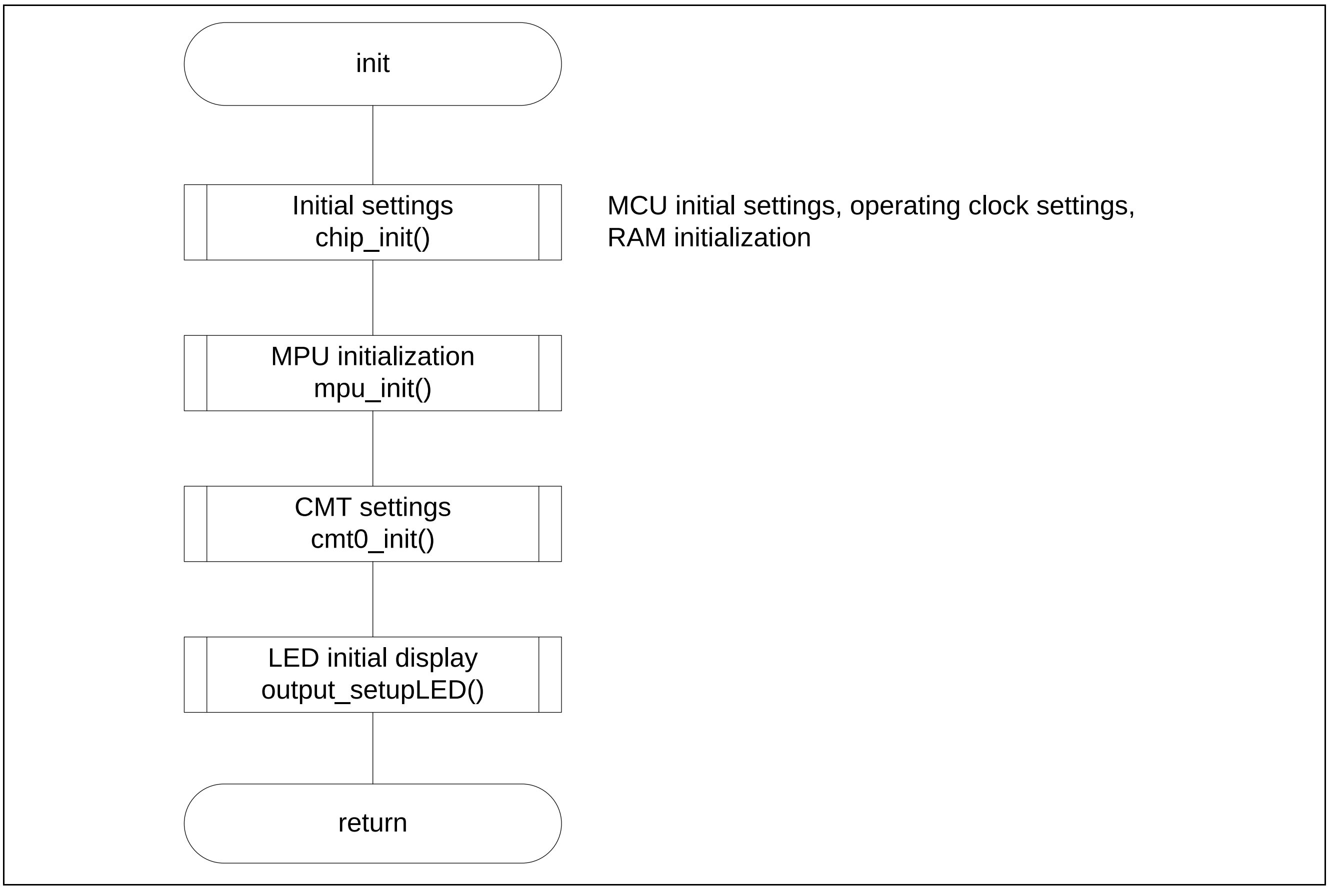

Figure 5.27 Call Initial Settings Functions

5.7.19 MCU Initial Settings, Operating Clock Settings, RAM Initialization

Figure 5.28 is a flowchart of the function that performs MCU initial settings, operating clock settings, and RAM initialization.

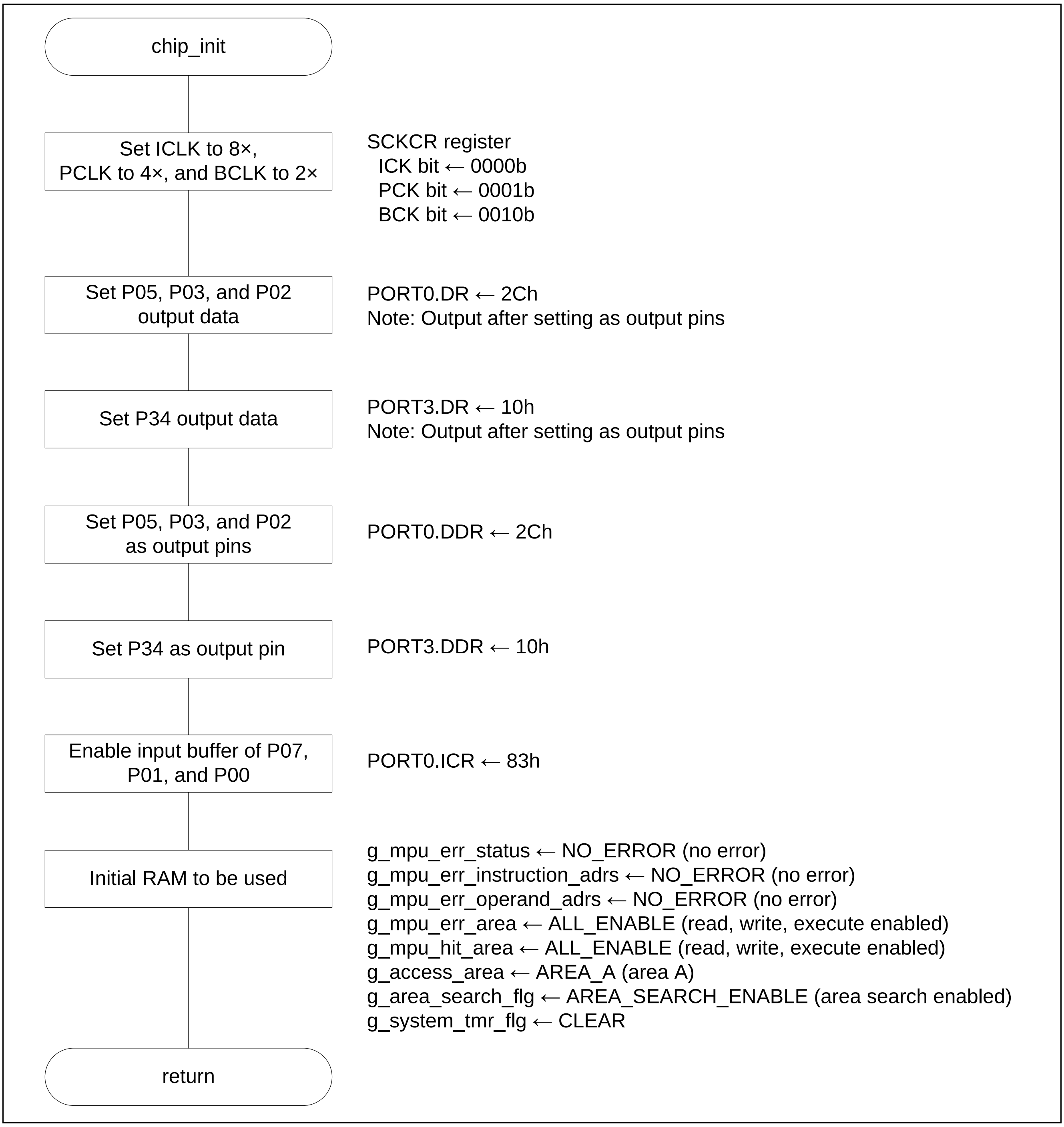

Figure 5.28 MCU Initial Settings, Operating Clock Settings, RAM Initialization

5.7.20 MPU Initialization

Figure 5.29 and Figure 5.30 are flowcharts (parts 1 and 2) of the MPU initialization function.

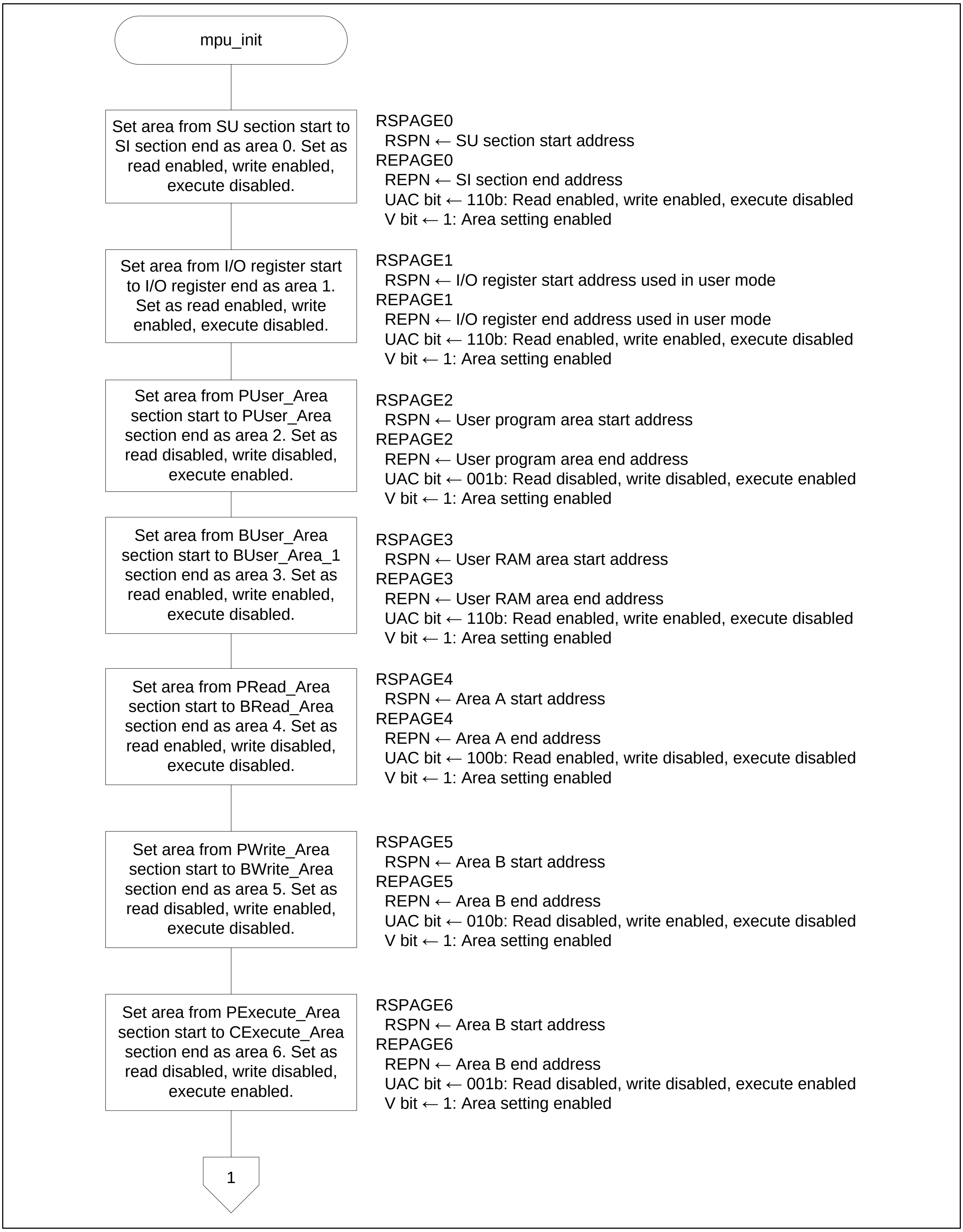

Figure 5.29 MPU Initialization (1)

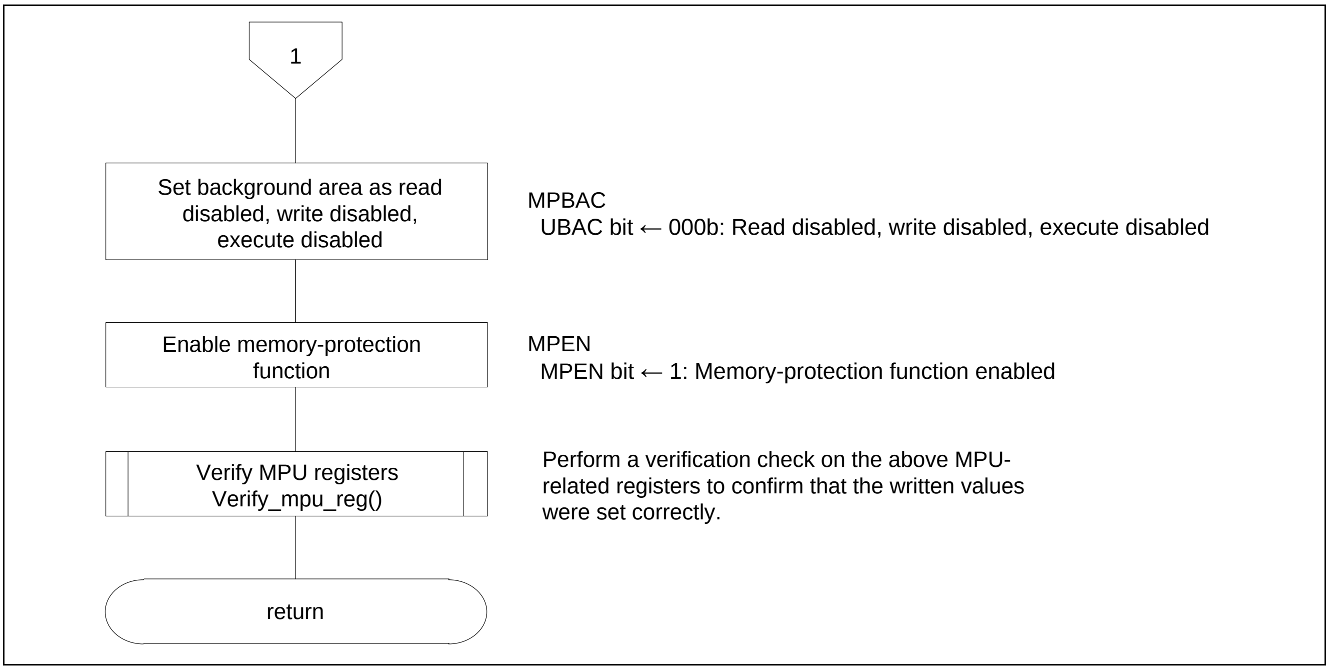

Figure 5.30 MPU Initialization (2)

5.7.21 Verify MPU-Related Registers

Figure 5.31 is a flowchart of the verify MPU-related registers function.

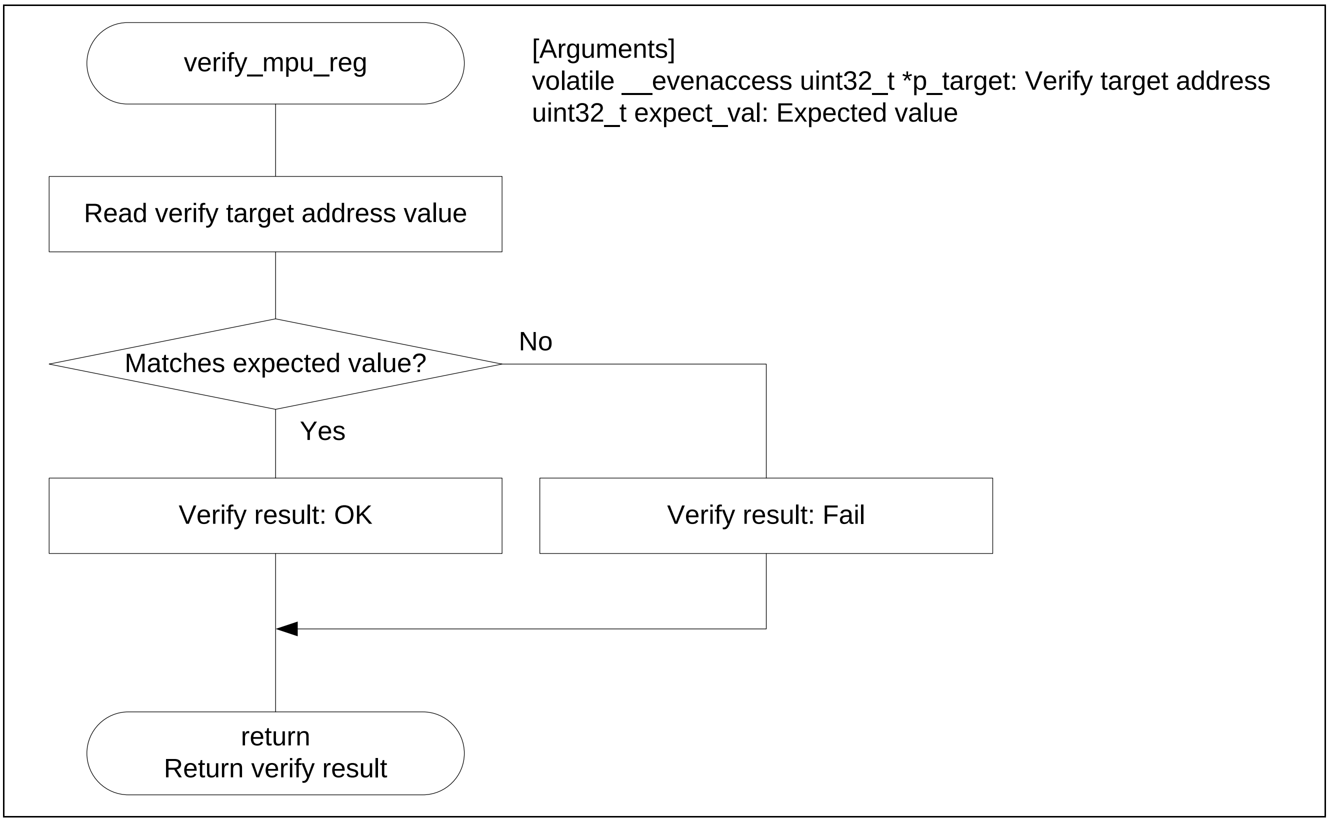

Figure 5.31 Verify MPU-Related Registers

5.7.22 CMT0 Settings

Figure 5.32 is a flowchart of the CMT0 settings function.

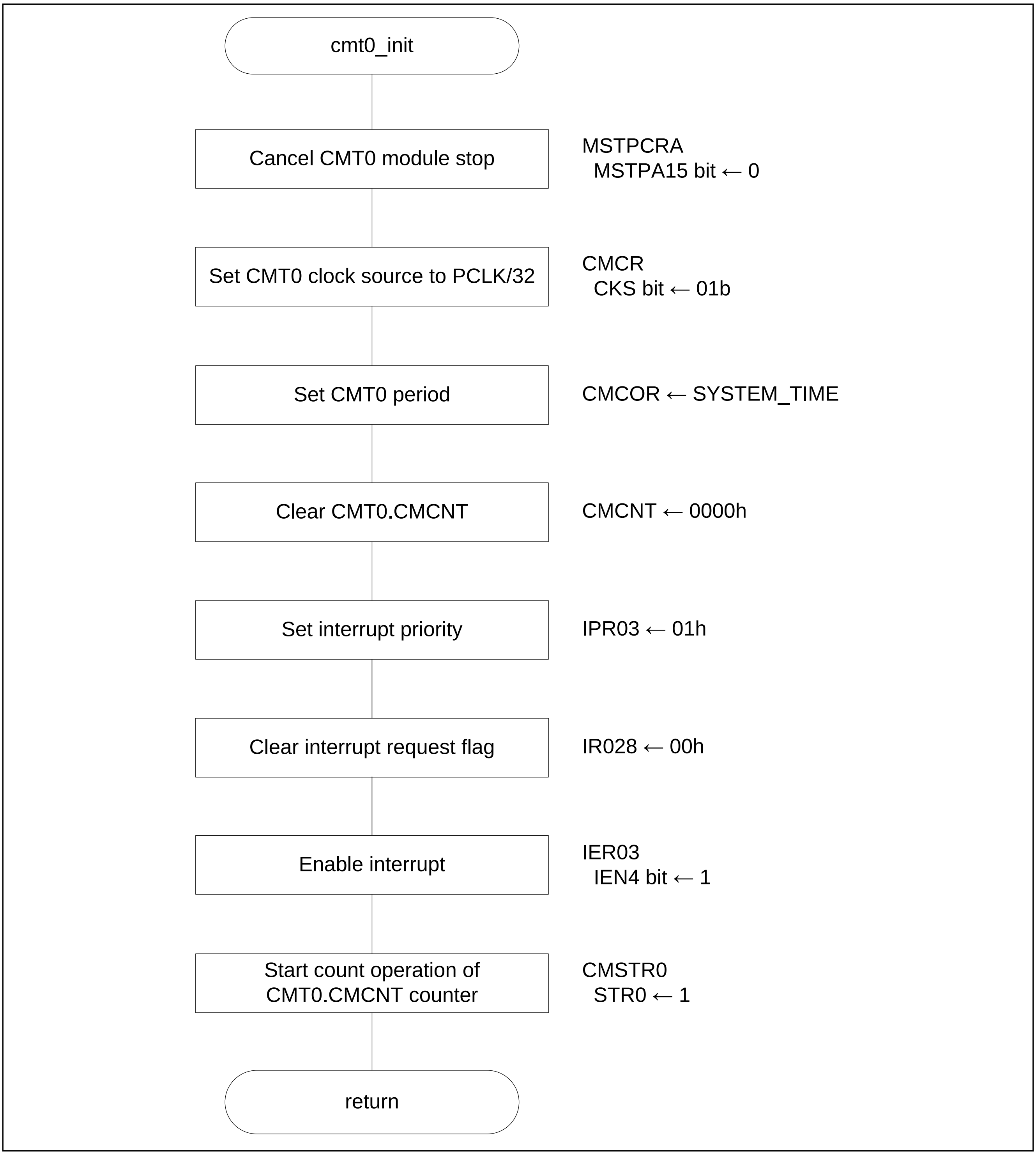

Figure 5.32 CMT0 Settings

5.7.23 Unconditional Trap Exception Handler

Figure 5.33 is a flowchart of the unconditional trap exception handler.

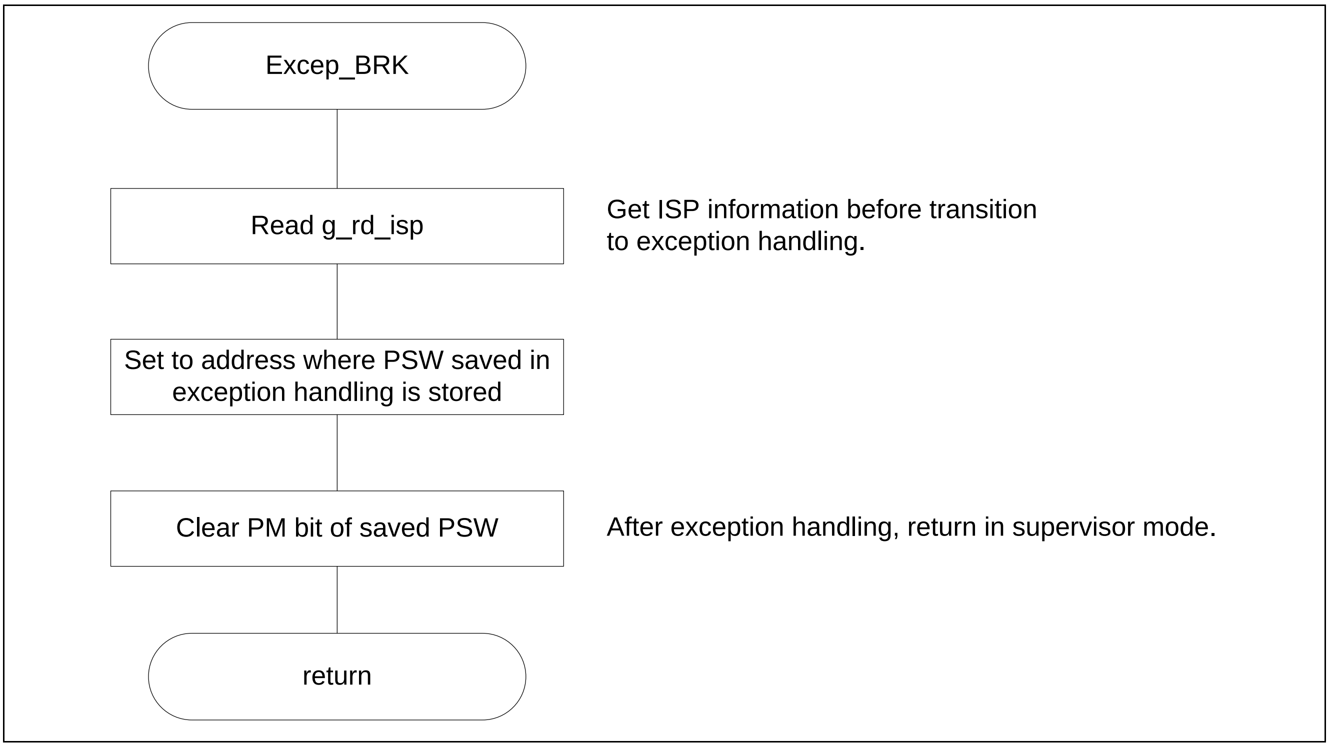

Figure 5.33 Unconditional Trap Exception Handler

5.7.24 Memory-Protection Error Access Exception Handler

Figure 5.34 is a flowchart of the memory-protection error access exception handler.

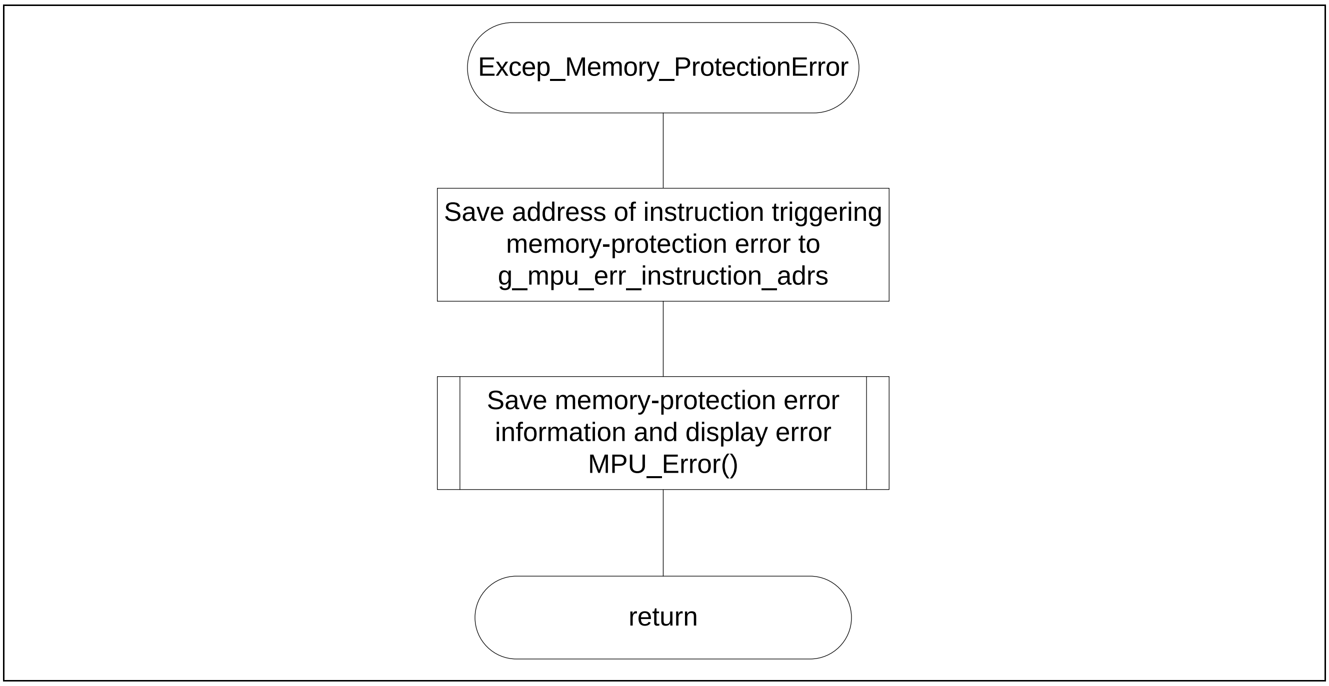

Figure 5.34 Memory-Protection Error Access Exception Handler

5.7.25 Memory-Protection Error Information Storage and Error Display

Figure 5.35 is a flowchart of the function for memory-protection error information storage and error display.

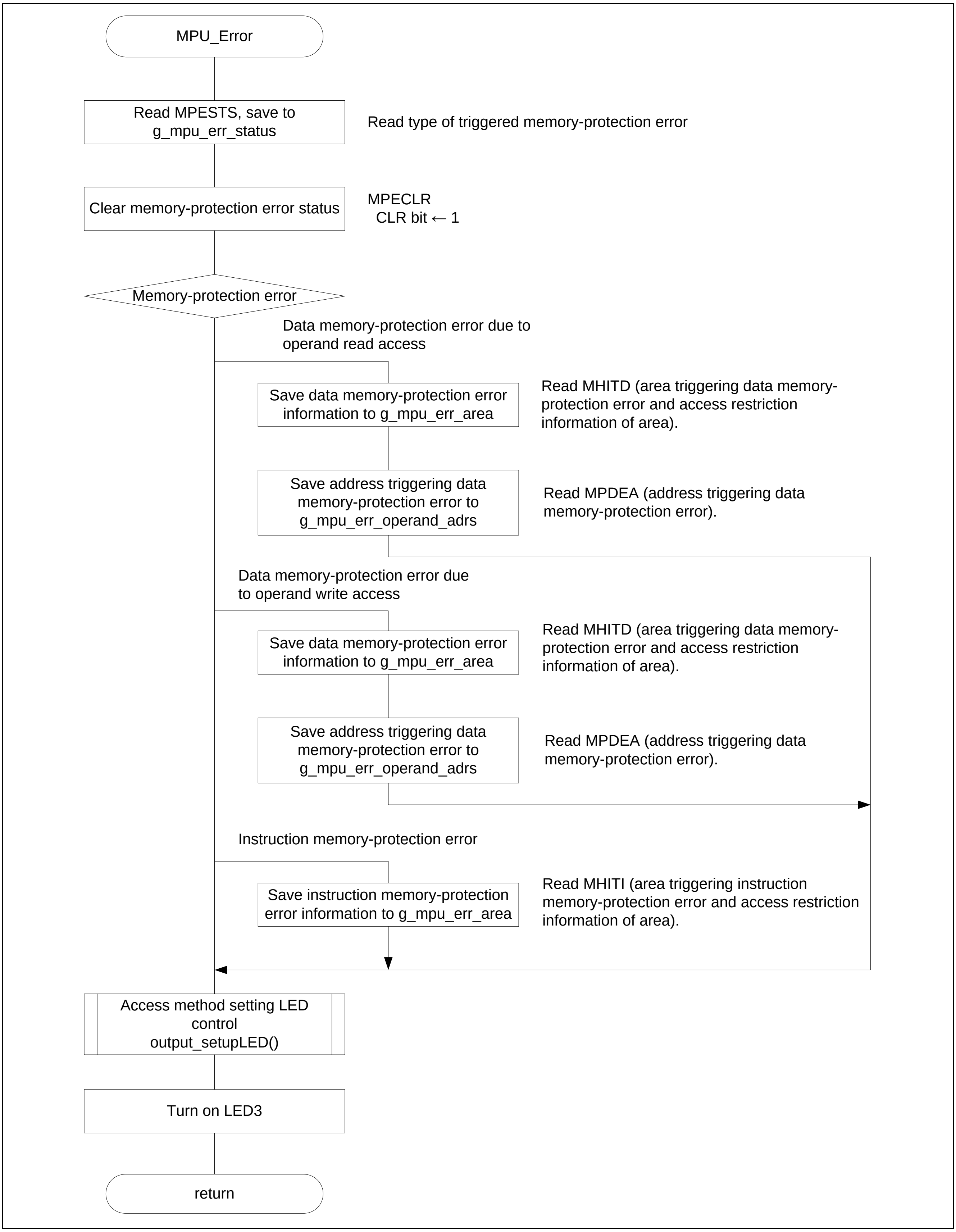

Figure 5.35 Memory-Protection Error Information Storage and Error Display

5.7.26 CMT0 Compare Match Exception Handler

Figure 5.36 is a flowchart of the CMT0 compare match exception handler.

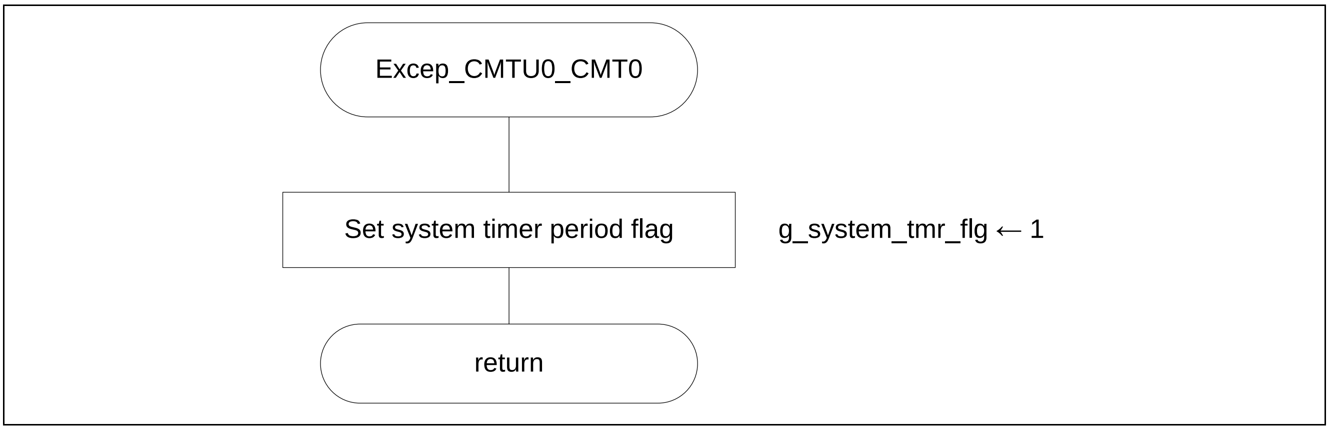

Figure 5.36 CMT0 Compare Match Exception Handler

6. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

7. Reference Documents

User's Manual: Hardware
RX62N, RX621 Group User's Manual: Hardware Rev.1.30
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
RX Family C/C++ Compiler, Assembler, Optimizing Linkage Editor—Compiler Package V. 1.01 User's Manual, Rev. 1.00
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

Rev.	Date	Description	
		Page	Summary
1.00	Jul 01, 2014	—	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 a product with a different type number, confirm that the change will not lead to problems.
   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

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

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

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

Renesas Electronics Canada Limited
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: +44-1628-585-100, Fax: +44-1628-585-900

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

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

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

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

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

Renesas Electronics Malaysia Sdn.Bhd.
Unit 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.
12F., 234 Teheran-ro, Gangnam-Ku, Seoul, 135-920, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 4.0