RZ/A1LU Group

Example of Writing to Serial Flash Memory Using the SPI Multi-I/O Bus Controller

R01AN3201EJ0100
Rev.1.00
May. 17, 2017

Abstract

This application note describes a sample program that performs write accesses to serial flash memory using the RZ/A1LU microcontroller's SPI multi-I/O bus controller (hereinafter called "SPIBSC").

Products

RZ/A1LU

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

After booting in boot mode 1 (serial flash boot) from serial flash memory allocated to the SPI multi-I/O bus space, this sample code switches the SPIBSC to SPI operating mode and issues commands to the serial flash memory to perform the processing for read/write accesses and other operations.

At the same time as performing the SPIBSC setup, this sample code also performs maintenance processing for the memory management unit, primary cache (L1 cache), and secondary cache (L2 cache).

In this application note, "SCIF" refers to the serial communication interface with FIFO, "PORT" refers to the general-purpose I/O ports, "STB" refers to power-down modes, and "MMU" refers to the memory management unit.

Table 1.1 lists the Peripheral Functions Used and Their Application and Figure 1.1 shows the Operating Environment used to run this sample code.

Table 1.1 Peripheral Functions Used and Their Application

Peripheral Function	Application
SPI multi-I/O bus controller (SPIBSC)	Perform the control for switching between external address space read mode and SPI operating mode, and is used for reads and writes to serial flash memory.
Serial communication interface with FIFO (SCIF)	Communicate between SCIF channel 0 and the host PC.
General-purpose I/O ports (PORT)	Switch multiplexed pin functions for SPIBSC and SCIF channel 0.
Power-down modes (STB)	Cancel the module standby state of the SPIBSC. *1
Memory management unit (MMU), L1 cache, L2 cache	Perform MMU translation table operations and L1 and L2 cache maintenance processing when switching the SPIBSC operating mode.

Note: 1. The SPIBSC module standby state is canceled by the startup on-chip ROM program that is executed during the serial flash boot operation.

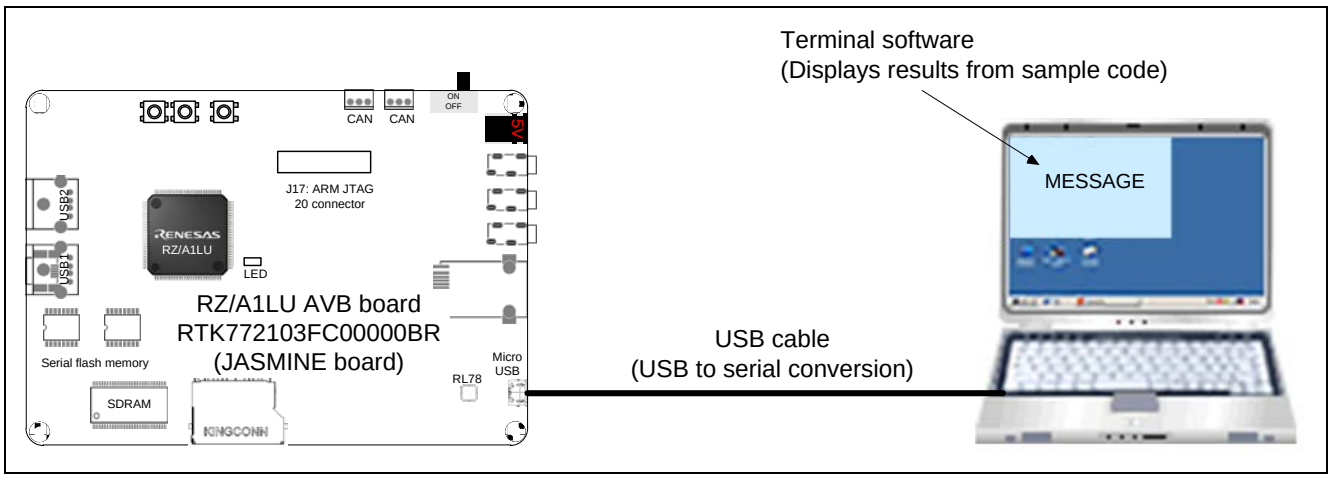


Figure 1.1 Operating Environment

2. Operation Confirmation Conditions

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

Table 2.1 Operation Confirmation Conditions

Item	Contents
Microcontroller used	RZ/A1LU
Operating frequency*	CPU clock (Iϕ): 400 MHz Internal bus clock (Bϕ): 133.33 MHz Peripheral clock (P1ϕ): 66.67 MHz Peripheral clock (P0ϕ): 33.33 MHz
Operating voltage	Power supply voltage (I/O): 3.3 V Power supply voltage (internal): 1.18 V
Integrated development environment	ARM® integrated development environment ARM Development Studio 5 (DS-5™) Version 5.24
C compiler	ARM C/C++ Compiler/Linker/Assembler Ver.5.06 update 2 [Build 183]
	Compiler options: -O3 -Ospace --cpu=Cortex-A9 --littleend --arm --apcs=/interwork --no_unaligned_access --fpu=vfpv3_fp16 -g --asm
Operating mode	Boot mode 1 (Serial flash boot)
Board used	RZ/A1LU AVB board RTK772103FC00000BR (Referred to as the JASMINE board in this document)
Terminal software communication settings	• Communication speed: 115,200 bps • Data length: 8 bits • Parity: None • Stop-bit length: 1 bit • Flow control: None
Device used (functions on board used)	• Serial flash memory (connected to the SPI multi-I/O bus space) — Manufacturer: Macronix International Co., Ltd. — Part No.: MX25L51245G • RL78/G1C (Convert between USB communication and serial communication to communicate with the host PC.) • LED1

Note: * The operating frequency used in clock mode 0 (Clock input of 13.33MHz from EXTAL pin)

3. Reference Application Notes

Documents related to this application note are listed below.

- RZ/A1H Group I/O definition header file iodefine.h (R01AN1860EJ)
- RZ/A1H Group Example of Initialization (R01AN1864EJ)
- RZ/A1LU Group Example of Booting from Serial Flash Memory (R01AN3093EJ)

4. Peripheral Functions

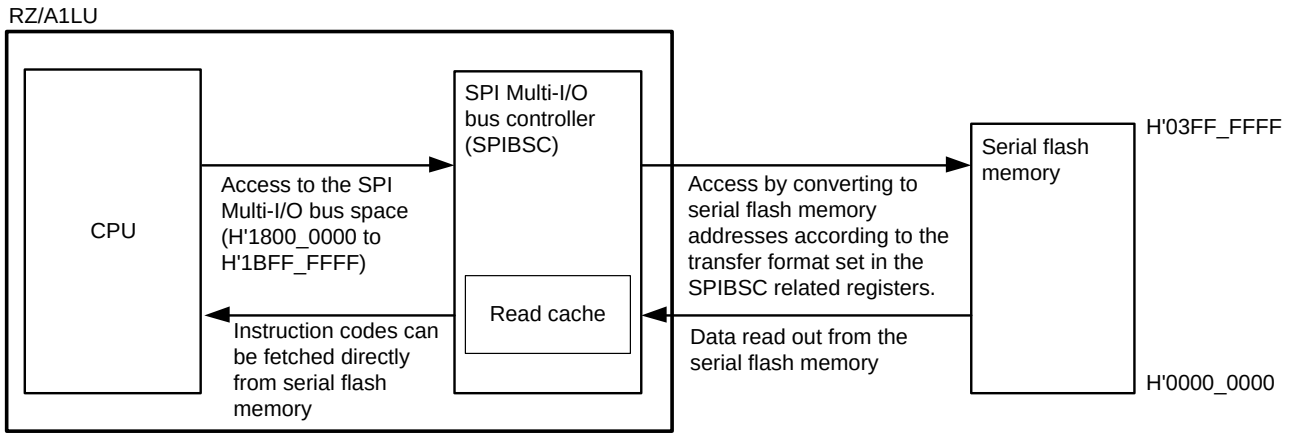
This section describes the operating modes of the RZ/A1LU's SPIBSC. See the "RZ/A1L Group, RZ/A1LU Group, and RZ/A1LC Group Users' Manual: Hardware" for details.

As listed in Table 4.1, the SPIBSC supports two modes: external address space read mode and SPI operating mode. By switching between these two modes and accessing the serial flash memory, the following operations are possible: read access, erase access, write access, and access to the serial flash memory's registers.

Table 4.1 Serial Flash Memory Access Using the SPIBSC

Operating Mode	Description
External address space read mode (CMNCR.MD = 0)	• Used for read access to the SPI multi-I/O bus space (H'1800_0000 to H'1BFF_FFFF) • The bus master can directly access data from serial flash memory (direct instruction code fetches from the CPU are possible) • The range over which direct read access is possible is up to the 64 MB of the SPI multi-I/O bus space (SPIBSC register control is required for accesses in excess of 64 MB)
SPI operating mode (CMNCR.MD = 1)	• Arbitrary commands can be issued to the serial flash memory by setting the SPIBSC related registers in software. • Used for erase access, write access, and access to the serial flash memory registers (status register, configuration register, and others). • Read access to a 4 GB space when 1 serial flash memory chip is connected and to an 8 GB space when 2 chips are connected is possible. • When set to SPI operating mode, access to the SPI multi-I/O bus space is not possible.

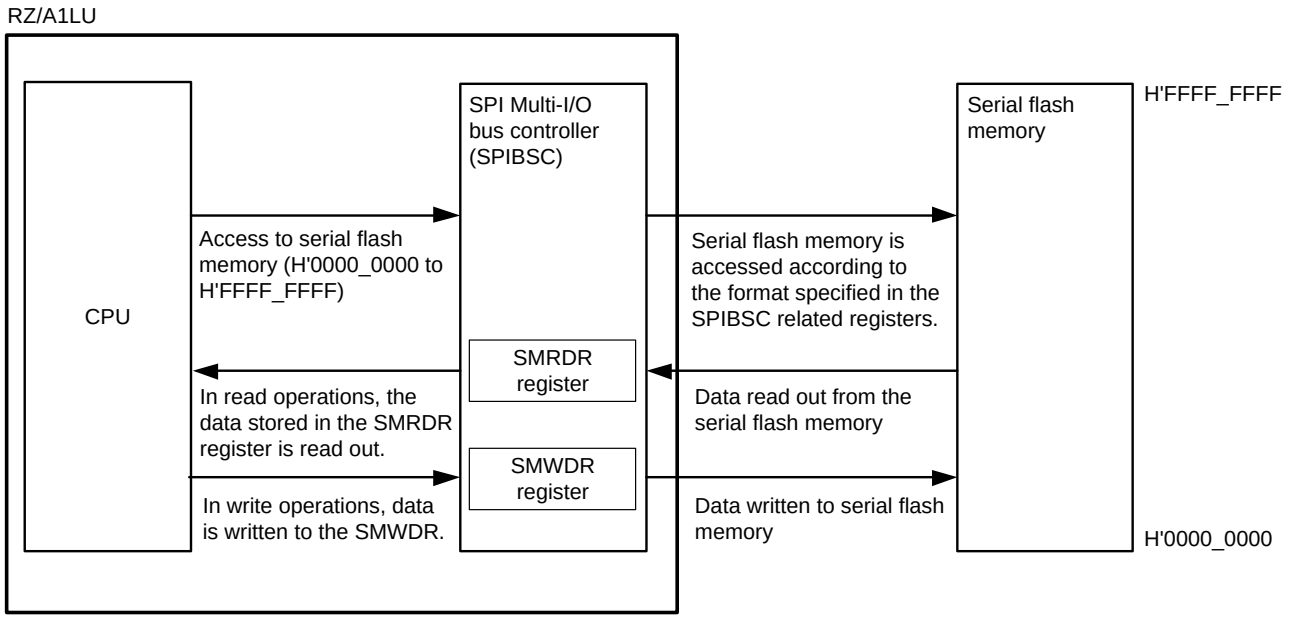
In external address space read mode, instructions stored in serial flash memory can be executed directly by converting read accesses to the SPI multi-I/O bus space to which the serial flash memory is connected to SPI communication (read command issue, address output, and dummy cycle output). In the RZ/A1LU, this is used with serial flash boot mode (boot mode 1), which use serial flash memory as the boot device.



Note: In external address space read mode, the serial flash memory data can be cached in the read cache by setting the RBE bit in the data read control register (DRCR). If the data requested by the CPU is held in the read cache, the access to the external serial flash memory is not performed and the data in the read cache is returned to the CPU.

Figure 4.1 Serial Flash Memory Access in External Address Space Read Mode

SPI operating mode is used for access to the serial flash memory registers (status register, configuration register, and other registers) and for the write accesses (write command issue, address output, and write data output) used when writing data. When serial flash memory is used as the storage device, the serial flash memory is accessed using SPI operating mode.



Note: While an area of 4 GB in serial flash memory (8 GB if two devices are used) can be accessed in SPI operating mode, in external address space read mode, the 64 MB area (H'1800_0000 to H'1BFF_FFFF) allocated as the SPI multi-I/O bus space can be accessed directly.

Figure 4.2 Serial Flash Memory Access in SPI Operating Mode

5. Hardware

5.1 Hardware Configuration

Figure 5.1 shows the Circuit Diagram for the case where the system boots from serial flash memory (MX25L51245G) in boot mode 1 and read and write accesses to the serial flash memory are performed.

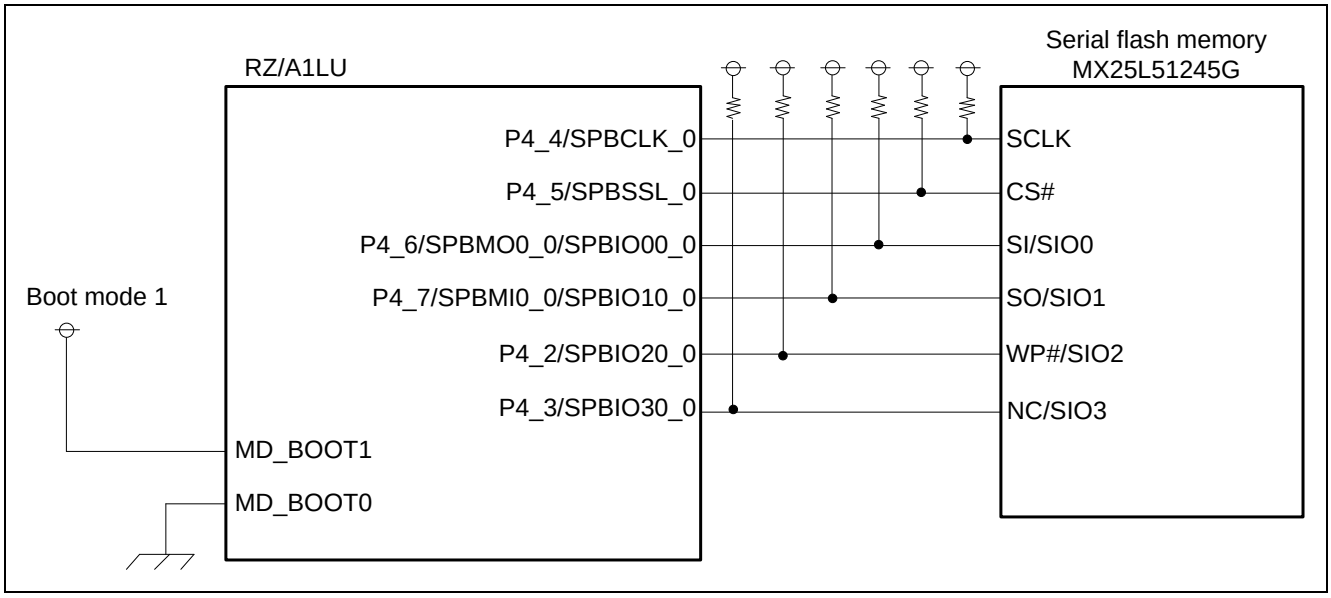


Figure 5.1 Circuit Diagram

5.2 Pins Used

Table 5.1 lists the Pins and Their Functions.

Table 5.1 Pins and Their Functions

Pin Name	I/O	Description
SPBCLK_0	Output	Clock output
SPBSSL_0	Output	Slave select
SPBMO0_0/SPBIO00_0	I/O	Master send data / data 0
SPBMI0_0/SPBIO10_0	I/O	Master input data / data 1
SPBIO20_0	I/O	Data 2
SPBIO30_0	I/O	Data 3
MD_BOOT1	Input	Select boot mode MD_BOOT1: "H", MD_BOOT0: "L" (Set to boot mode 1) After boot-up, MD_BOOT0 is switched to the function of the serial receive data signal as RxD0 (Note).
MD_BOOT0/RxD0	Input	
P8_12	Output	Turns LED1 on and off.
TxD0	Output	Serial transmit data signal

Note: The MD_BOOT0 and RxD0 functions are multiplexed on the P0_0 pin. When the power-on reset is canceled, this pin operates as MD_BOOT0 and is used as a pin that determines the boot mode. The sample code performs input pin selection control with the multiplexer/demultiplexer (SN74CB3Q3257) on the JASMINE board. A boot function selection signal is input from the switch on the board when the power-on reset is canceled and then the pin is pulled up after the power-on reset has canceled to set the pin to operate as the RxD0 function.

6. Software

6.1 Operation Overview

After starting the program in serial flash boot mode using the SPIBSC external address space read mode, this sample code performs serial flash memory erase, write, and read access processing according to command character strings input from the terminal. See section 6.6, Sample Commands, for the commands provided by this sample code.

When started by the serial flash boot operation, this sample code sets the system so that programs stored in serial flash memory connected to the SPI multi-I/O bus space by the SPIBSC to external address space read mode can be executed directly. When an erase or write operation is performed on the serial flash memory, that processing is performed by switching the SPIBSC to SPI operating mode to set the SPIBSC registers and issuing commands to the serial flash memory.

6.2 Accessing Serial Flash Memory Using the SPIBSC

Since the SPI multi-I/O bus space cannot be accessed when the SPIBSC is in SPI operating mode, it is necessary to assure that software does not access code or data allocated to the SPI multi-I/O bus space.

Processing that must be executed after setting the system to SPI operating mode, must be copied to another memory space that is not the SPI multi-I/O bus space. If interrupt handling is used, care is required to allocate the interrupt vector and any interrupt handlers that are used to a memory space other than the SPI multi-I/O bus space.

Also note that to speed up processing, the ARM Cortex-A9 processor supports branch prediction and speculative execution. When there is a conditional branch instruction, before the condition is confirmed, the branch target instruction fetch and/or a data access may occur due to the branch prediction/speculative execution operation, and it is possible that, when the prediction fails, an access that would not be performed is actually performed. When having switched to SPI operating mode to perform processing such as an erase or write access to serial flash memory after executing a program in external address space read mode, it is possible that an access to SPI multi-I/O bus space while in SPI operating mode may occur due to the branch prediction/speculative execution operation and the program may not operate correctly. When switching the operating mode, this sample code sets the "access disabled/enabled" and "execution disabled/enabled" attributes with the SPI multi-I/O bus space MMU translation table so that illegal accesses to the SPI multi-I/O bus space do not occur. This processing is described in detail in section 6.3, External Address Space Read Mode and SPI Operating Mode Switching.

6.3 External Address Space Read Mode and SPI Operating Mode Switching

The SPIBSC can be switched between external address space read mode and SPI operating mode using the following procedures.

Note that since the SPIBSC operating mode switching processing cannot be executed from serial flash memory allocated to the SPI multi-I/O bus space, it must be executed from another memory space. In this sample code, the processing in sections 6.3.1 and 6.3.2 is transferred to on-chip large-capacity RAM and that processing is executed from on-chip large-capacity RAM.

For details on the procedures for modifying the MMU translation table, see the "TLB maintenance operations and the memory order model" section in the "ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition Issue C" manual provided by ARM, Inc.

6.3.1 Switching from External Address Space Read Mode to SPI Operating Mode

The following is the procedure for switching to the SPIBSC SPI operating mode.

1. Modify the MMU translation table for the SPI multi-I/O bus space

   Set the MMU translation table settings for the SPI multi-I/O bus space to the access disabled and execution disabled attributes so that access to the SPI multi-I/O bus space do not occur even if speculative execution is performed by the Cortex-A9.
2. Perform L1 data cache cleaning (write back) and invalidation processing

   L1 data cache cleaning and invalidation is performed to invalidate the SPI multi-I/O bus space data and the contents of the MMU translation table held in the cache.
3. Execute a DSB instruction

   A DSB instruction is executed to assure that the L1 data cache cleaning operation in step 2 above has completed.
4. Perform L2 cache cleaning (write back) and invalidate processing

   L2 cache cleaning and invalidation is performed to invalidate the SPI multi-I/O bus space instruction and data held in the cache.
5. Perform Cache Sync

   This is performed to assure that the L2 cache maintenance processing in step 4 above has completed.
6. Invalidate the TLB

   The MMU translation table settings are changed, so the translation lookaside buffer (TLB) is invalidated.

7. Invalidate the L1 instruction cache

   The MMU translation table settings are changed, so the L1 instruction cache is invalidated.

8. Execute a DSB instruction

   A DSB instruction is executed to assure that the L1 instruction cache invalidation operation in step 7 above has completed.

9. Execute an ISB instruction

   An ISB instruction is executed to assure that instructions executed after this point reference the modified MMU translation table.

10. Switch the SPIBSC from external address space read mode to SPI operating mode

    The TEND bit in the common status register (CMNSR) is read to verify that no access to serial flash memory is occurring. Then, 1 is written to the MD bit in the common control register (CMNCR) and a dummy read of the CMNCR register is performed to switch to SPI operating mode.

Note: If interrupts are enabled, it is necessary to take measures such as copying the interrupt handlers from serial flash memory to another memory space, so that accesses to the SPI multi-I/O bus space do not occur after switching from external address space read mode to SPI operating mode. In this sample code, interrupts are disabled before switching to SPI operating mode and interrupts are enabled before switching to external address space read mode, so that such accesses to serial flash memory do not occur.

6.3.2 Switching from SPI Operating Mode to External Address Space Read Mode

The following is the procedure for switching to the SPIBSC external address space read mode.

1. Switch the SPIBSC from SPI operating mode to external address space read mode.

   Verify that no accesses to serial flash memory have occurred by reading the TEND bit in the common status register (CMNSR). Write a 1 to the RCF bit in the data read control register (DRCR), perform a dummy read of the DRCR register, and clear the SPIBSC read cache. Write a 0 to the MD bit in the common control register (CMNCR), perform a dummy read of the CMNCR register, and switch to external address space read mode.
2. Modify the MMU translation table for the SPI multi-I/O bus space

   Set the MMU translation table settings for the SPI multi-I/O bus space to the access enabled and execution enabled attributes and set the SPI multi-I/O bus space to the access enabled state.
3. Perform L1 data cache cleaning (write back) and invalidation processing

   L1 data cache cleaning and invalidation is performed to invalidate the SPI multi-I/O bus space data and the contents of the MMU translation table held in the cache.
4. Execute a DSB instruction

   A DSB instruction is executed to assure that the L1 data cache cleaning operation in step 3 above has completed.
5. Perform L2 cache cleaning and invalidate processing

   L2 cache cleaning and invalidation is performed to invalidate the SPI multi-I/O bus space instruction and data held in the cache.
6. Perform Cache Sync

   This is performed to assure that the L2 cache maintenance processing in step 5 above has completed.
7. Invalidate the TLB

   The translation lookaside buffer (TLB) is invalidated to change the MMU translation table settings.
8. Invalidate the L1 instruction cache

   The L1 instruction cache is invalidated to change the MMU translation table settings.
9. Execute a DSB instruction

   A DSB instruction is executed to assure that the L1 instruction cache invalidation operation in step 8 above has completed.
10. Execute an ISB instruction

    An ISB instruction is executed to assure that instructions executed after this point reference the modified MMU translation table.

Notes: 1. The processing in steps 5 and 6 above are not required when both the following conditions are met.
If the MMU translation table is allocated not to external memory but to on-chip large-capacity RAM.
If the processing in steps 5 and 6 above were performed during the processing for switching from external address space read mode to SPI operating mode and coherency of the cache and the serial flash memory are assured.

2. If interrupts were disabled in the processing of section 6.3.1, in the processing following the above, the user should enable any interrupts that will be used.

6.4 SPIBSC Operating Mode Switching Processing Implementation

This section shows the implementations of the various processing steps mentioned in section 6.3, External Address Space Read Mode and SPI Operating Mode Switching.

6.4.1 Implementation of Switching Processing from External Address Space Read Mode to SPI Operating Mode

1. MMU translation table modification implementation

This section presents an example of the implementation of the modification processing for the MMU translation table used when switching from external address space read mode to SPI operating mode. This is implemented in the Change_MMU_TTbl_SpibscSpi() function in this sample code, and Figure 6.1 and Figure 6.2 present excerpts from that processing. The start address and end address of the SPI multi-I/O bus space are specified with the variables start_addr and end_addr respectively as the arguments to the Change_MMU_TTbl_SpibscSpi() function.

```
static int32_t Change_MMU_TTbl_SpibscSpi(uint32_t start_addr, uint32_t end_addr)
{
    uint32_t index_start;         /* Start address table index */
    uint32_t index_end;           /* End address table index */
    uint32_t index;               /* Loop variable: table index */
    mmu_ttbl_desc_section_t desc; /* Loop variable: descriptor */

    index_start = MMU_TTbl_GetIndex(start_addr);  /* Get start address table index */
    index_end = MMU_TTbl_GetIndex(end_addr);      /* Get end address table index */

    for(index = index_start; index <= index_end; index++)
    {
        /* Get descriptor from translation table */
        MMU_TTbl_GetDesc(index, &desc);

        /* Modify memory attribute descriptor */
        desc.AP1_0 = 0x0u; /* AP[2:0] = b'000 (No access) */
        desc.AP2 = 0x0u;
        desc.XN = 0x1u;    /* XN = 1 (Execute never)  */

        /* Write descriptor back to translation table */
        MMU_TTbl_SetDesc(index, &desc);
    }

    return 0;
}
```

Figure 6.1 MMU Translation Table Modification Processing when Switching to SPI Operating Mode

```
/* Definition of the short-descriptor translation table first-level descriptor format (section) */
typedef struct {
    uint32_t b0        : 1 ;   /* bit 0       : -        (0)                                  */
    uint32_t b1        : 1 ;   /* bit 1       : -        (1)                                  */
    uint32_t B         : 1 ;   /* bit 2       : B        Memory region attribute bit          */
    uint32_t C         : 1 ;   /* bit 3       : C        Memory region attribute bit          */
    uint32_t XN        : 1 ;   /* bit 4       : XN       Execute-never bit                    */
    uint32_t Domain    : 4 ;   /* bit 8-5     : Domain   Domain field                         */
    uint32_t b9        : 1 ;   /* bit 9       : IMP      IMPLEMENTATION DEFINED               */
    uint32_t AP1_0     : 2 ;   /* bit 11-10   : AP[1:0]  Access permissions bits:bit1-0       */
    uint32_t TEX       : 3 ;   /* bit 14-12   : TEX[2:0] Memory region attribute bits         */
    uint32_t AP2       : 1 ;   /* bit 15      : AP[2]    Access permissions bits:bit2         */
    uint32_t S         : 1 ;   /* bit 16      : S        Shareable bit                        */
    uint32_t nG        : 1 ;   /* bit 17      : nG       Not global bit                       */
    uint32_t b18       : 1 ;   /* bit 18      : -        (0)                                  */
    uint32_t NS        : 1 ;   /* bit 19      : NS       Non-secure bit                       */
    uint32_t base_addr : 12;   /* bit 31-20   : PA[31:20] PA(physical address) bits:bit31-20  */
} mmu_ttbl_desc_section_t;

/* Definition of the base address for the MMU translation table */
#define TTB    ((uint32_t)&Image$$TTB$$ZI$$Base)     /* using linker symbol */

uint32_t MMU_TTbl_GetIndex(uint32_t addr)
{
    uint32_t index;
    index = addr >> 20;
    return index;
}

int32_t MMU_TTbl_GetDesc(uint32_t index, mmu_ttbl_desc_section_t * pdesc)
{
    mmu_ttbl_desc_section_t * table;

    /* ==== Get descriptor value from translation table ==== */
    table = (mmu_ttbl_desc_section_t *)TTB;
    *pdesc = table[index];

    return 0;
}

int32_t MMU_TTbl_SetDesc(uint32_t index, mmu_ttbl_desc_section_t * pdesc)
{
    mmu_ttbl_desc_section_t * table;

    /* ==== Set descriptor value in translation table ==== */
    table = (mmu_ttbl_desc_section_t *)TTB;
    table[index] = *pdesc;

    return 0;
}
```

Figure 6.2 MMU Translation Table Setting Processing

2. L1 data cache cleaning and invalidation processing implementation

Figure 6.3 and Figure 6.4 show the implementation for L1 Data Cache Cleaning and Invalidation Processing. This is implemented in the L1_D_CacheWritebackFlushAll() function in this sample code.

```
void L1_D_CacheWritebackFlushAll(void)
{
    /* ==== Invalidate and clean all D cache by set/way ==== */
    L1_D_CacheOperationAsm(L1CACHE_WB_FLUSH);     /* L1CACHE_WB_FLUSH == 2 */
}


;*****************************************************************************
; Function Name: L1_D_CacheOperationAsm
; Description  : r0 = 0 : DCISW. Invalidate data or unified cache line by set/way.
;              : r0 = 1 : DCCSW. Clean data or unified cache line by set/way.
;              : r0 = 2 : DCCISW. Clean and Invalidate data or unified cache line by set/way.
;*****************************************************************************
L1_D_CacheOperationAsm FUNCTION

    PUSH {r4-r11}

    MRC  p15, 1, r6, c0, c0, 1    ;;; Read CLIDR
    ANDS r3, r6, #0x07000000      ;;; Extract coherency level
    MOV  r3, r3, LSR #23          ;;; Total cache levels << 1
    BEQ  Finished                 ;;; If 0, no need to clean

    MOV  r10, #0                  ;;; R10 holds current cache level << 1
Loop1
    ADD  r2, r10, r10, LSR #1     ;;; R2 holds cache "Set" position
    MOV  r1, r6, LSR r2           ;;; Bottom 3 bits are the Cache-type for this level
    AND  r1, r1, #7               ;;; Isolate those lower 3 bits
    CMP  r1, #2
    BLT  Skip                     ;;; No cache or only instruction cache at this level

    MCR  p15, 2, r10, c0, c0, 0   ;;; Write the Cache Size selection register (CSSELR)
    ISB                           ;;; ISB to sync the change to the CacheSizeID reg
    MRC  p15, 1, r1, c0, c0, 0    ;;; Reads current Cache Size ID register (CCSIDR)
    AND  r2, r1, #7               ;;; Extract the line length field
    ADD  r2, r2, #4               ;;; Add 4 for the line length offset (log2 16 bytes)
    LDR  r4, =0x3FF
    ANDS r4, r4, r1, LSR #3       ;;; R4 is the max number on the way size (right aligned)
    CLZ  r5, r4                   ;;; R5 is the bit position of the way size increment
    LDR  r7, =0x7FFF
    ANDS r7, r7, r1, LSR #13      ;;; R7 is the max number of the index size (right aligned)
Loop2
    MOV  r9, r4                   ;;; R9 working copy of the max way size (right aligned)
```

Figure 6.3 L1 Data Cache Cleaning and Invalidation Processing (1/2)

```
Loop3
    ORR  r11, r10, r9, LSL r5     ;;; Factor in the Way number and cache number into R11
    ORR  r11, r11, r7, LSL r2     ;;; Factor in the Set number
    CMP  r0, #0
    BNE  Dccsw
    MCR  p15, 0, r11, c7, c6, 2   ;;; Invalidate by Set/Way (DCISW)
    B    Count
Dccsw
    CMP  r0, #1
    BNE  Dccisw
    MCR  p15, 0, r11, c7, c10, 2  ;;; Clean by set/way (DCCSW)
    B    Count
Dccisw
    MCR  p15, 0, r11, c7, c14, 2  ;;; Clean and Invalidate by set/way (DCCISW)
Count
    SUBS r9, r9, #1               ;;; Decrement the Way number
    BGE  Loop3
    SUBS r7, r7, #1               ;;; Decrement the Set number
    BGE  Loop2
Skip
    ADD  r10, r10, #2             ;;; increment the cache number
    CMP  r3, r10
    BGT  Loop1

Finished
    DSB
    POP  {r4-r11}
    BX   lr

    ENDFUNC
```

Figure 6.4 L1 Data Cache Cleaning and Invalidation Processing (2/2)

3. DSB instruction implementation

The DSB instruction used to guarantee that the L1 data cache cleaning and invalidation processing has completed is implemented in the L1_D_CacheOperationAsm() function.

4. L2 cache cleaning and invalidation processing implementation

Figure 6.5 shows an example of the implementation of the L2 Cache Cleaning and Invalidation Processing.

```
#define L2CACHE_8WAY   (0x000000FFuL)       /* All entries(8way) in the L2 cache */

void L2CacheWritebackFlushAll(void)
{
    /* ==== Clean and Invalidate all cache by Way ==== */
    L2C.REG7_CLEAN_INV_WAY = L2CACHE_8WAY;  /* Set 1 to Way bits[7:0] of the reg7_clean_inv_way */
    while ((L2C.REG7_CLEAN_INV_WAY &  L2CACHE_8WAY)
                        != 0x00000000uL)     /* Wait until Way bits[7:0] is cleared */
    {
    }

    /* ==== Cache Sync ==== */
    L2C.REG7_CACHE_SYNC = 0x00000000uL;     /* Ensures completion of the operation */
}
```

Figure 6.5 L2 Cache Cleaning and Invalidation Processing

5. Cache sync processing implementation

The cache sync processing used to guarantee that the L2 cache cleaning and invalidation processing is implemented in the L2CacheWritebackFlushAll() function.

6. TLB invalidation processing implementation

Figure 6.6 shows an example of the implementation of the TLB Invalidation Processing. This processing is implemented in the TLB_FlushAll() function in this sample code.

```
TLB_FlushAll    FUNCTION
    MOV  r0,#0
    MCR  p15, 0, r0, c8, c7, 0    ;;; Cortex-A9 I-TLB and D-TLB invalidation (TLBIALL)
    DSB
    ISB

    BX   lr

    ENDFUNC
```

Figure 6.6 TLB Invalidation Processing

7. L1 instruction cache invalidation processing implementation

Figure 6.7 shows an example of the implementation of the L1 Instruction Cache Invalidation Processing. This is implemented in the L1_I_CacheFlushAllAsm() function in this sample code.

```
L1_I_CacheFlushAllAsm FUNCTION

    MOV  r0, #0
    MCR  p15, 0, r0, c7, c5, 0    ;;; ICIALLU
    DSB
    ISB

    BX   lr

    ENDFUNC
```

Figure 6.7 L1 Instruction Cache Invalidation Processing

8. DSB instruction implementation

The DSB instruction used to guarantee that the L1 instruction cache invalidation processing has completed is implemented in the L1_I_CacheFlushAllAsm() function.

9. ISB instruction implementation

The ISB instruction used to guarantee that all following instructions reference the post-modification MMU translation table is implemented in the L1_I_CacheFlushAllAsm() function.

10. Implementation of switching the SPIBSC from external address space read mode to SPI operating mode

The processing that switches to SPI operating mode is performed by writing a 1 to the MD bit in the SPIBSC common control register (CMNCR). This is implemented in the R_SFLASH_Spimode() function in this sample code.

6.4.2 Implementation of Switching Processing from SPI Operating Mode to External Address Space Read Mode

1. Implementation of switching the SPIBSC from SPI operating mode to external address space read mode

   Clear the SPIBSC read cache by setting the RCF bit in the SPIBSC data read control register (DRCR) to 1 and write a 0 to the MD bit in the common control register (CMNCR) to switch to external address space read mode. This is implemented in the R_SFLASH_Exmode() function in this sample code.

2. MMU translation table modification implementation

   This section presents an example of the implementation of the processing for modifying the MMU translation table when switching from SPI operating mode to external address space read mode. This is implemented in the Change_MMU_TTbl_SpibscXip() function in this sample code and an excerpt of this processing is shown in Figure 6.8. The start address and end address of the SPI multi-I/O bus space are specified with the start_addr and end_addr arguments to the Change_MMU_TTbl_SpibscXip() function. The processing to set up the MMU translation table is implemented in the MMU_TTbl_GetIndex(), MMU_TTbl_GetDesc(), and MMU_TTbl_SetDesc() functions. See Figure 6.2 for more information on those functions.

```
static int32_t Change_MMU_TTbl_SpibscXip(uint32_t start_addr, uint32_t end_addr)
{
    uint32_t index_start;         /* Start address table index */
    uint32_t index_end;           /* End address table index */
    uint32_t index;               /* Loop variable: table index */
    mmu_ttbl_desc_section_t desc; /* Loop variable: descriptor */

    index_start = MMU_TTbl_GetIndex(start_addr); /* Get start address table index */
    index_end = MMU_TTbl_GetIndex(end_addr);     /* Get end address table index */

    for(index = index_start; index <= index_end; index++)
    {
        /* Get descriptor from translation table */
        MMU_TTbl_GetDesc(index, &desc);

        /* Modify memory attribute descriptor */
        desc.AP1_0 = 0x3u; /* AP[2:0] = b'011 (Full access) */
        desc.AP2 = 0x0u;
        desc.XN = 0x0u;    /* XN = 0 (Executable)  */

        /* Write descriptor back to translation table */
        MMU_TTbl_SetDesc(index, &desc);
    }

    return 0;
}
```

Figure 6.8 MMU Translation Table Modification Processing when Switching to External Address Space Read Mode

3. L1 data cache cleaning and invalidation processing implementation

   Figure 6.3 and Figure 6.4 show the implementation for L1 Data Cache Cleaning and Invalidation Processing. This is implemented in the L1_D_CacheWritebackFlushAll() function in this sample code.

4. DSB instruction implementation

   An example of the implementation of the L1 data cache cleaning and invalidation processing is shown. This is implemented in the L1_D_CacheWritebackFlushAll() function in this sample code.

5. L2 cache cleaning and invalidation processing implementation

   Figure 6.5 shows an example of the implementation of the L2 Cache Cleaning and Invalidation Processing.

6. Cache sync processing implementation

   An example of the implementation of the L2 cache cleaning and invalidation processing is shown. This is implemented in the L2CacheWritebackFlushAll() function in this sample code.

7. TLB invalidation processing implementation

   Figure 6.6 shows an example of the implementation of the TLB Invalidation Processing. This processing is implemented in the TLB_FlushAll() function in this sample code.

8. L1 instruction cache invalidation processing implementation

   Figure 6.7 shows an example of the implementation of the L1 Instruction Cache Invalidation Processing. This is implemented in the L1_I_CacheFlushAllAsm() function in this sample code.

9. DSB instruction implementation

   An example of the implementation of the L1 instruction cache invalidation processing is shown. This is implemented in the L1_I_CacheFlushAllAsm() function in this sample code.

10. ISB instruction implementation

    An example of the implementation of the L1 instruction cache invalidation processing is shown. This is implemented in the L1_I_CacheFlushAllAsm() function in this sample code.

6.5 The MMU Translation Table

In this sample code, the descriptor type of the first level descriptors in the MMU translation table are set to "section", and the information for 1 MB memory areas is specified with 32-bit descriptor. Also, the system is set up with the translation table base control register (TTBCR) and translation table base register 0 (TTBR0), so that a 16 KB (4096 entry) translation table area is managed by the MMU using TTBR0. The 16 KB area from location H'2003_4000 to location H'2003_7FFF is used as this MMU translation table storage area. Attributes in 1 MB units, such as the cache enabled/disabled state, the memory type attribute, the executable attribute, and the accessible attribute, are specified with the MMU translation table for the 4 GB address space from location H'0000_0000 to location H'FFFF_FFFF.

In this sample code, when switching the SPIBSC operating mode, the information in the MMU translation table for the SPI multi-I/O bus space is modified. Figure 6.9 shows the MMU Translation Table Settings Used in This Sample Code. When switching the SPIBSC to SPI operating mode, the MMU translation table AP[2:0] bits are set to B'000 (no access) and the XN bit is set to 1 (execute never). When switching the SPIBSC to external address space read mode, the MMU translation table AP[2:0] bits are set to B'011 (full access) and the XN bit is set to 0 (executable). Note that for domains set to manager, no attribute access check is performed for the XN bit and no access check for setting the AP[2:0] bits is performed. To protect accesses due to setting the MMU translation table entries, the domain must be set to client.

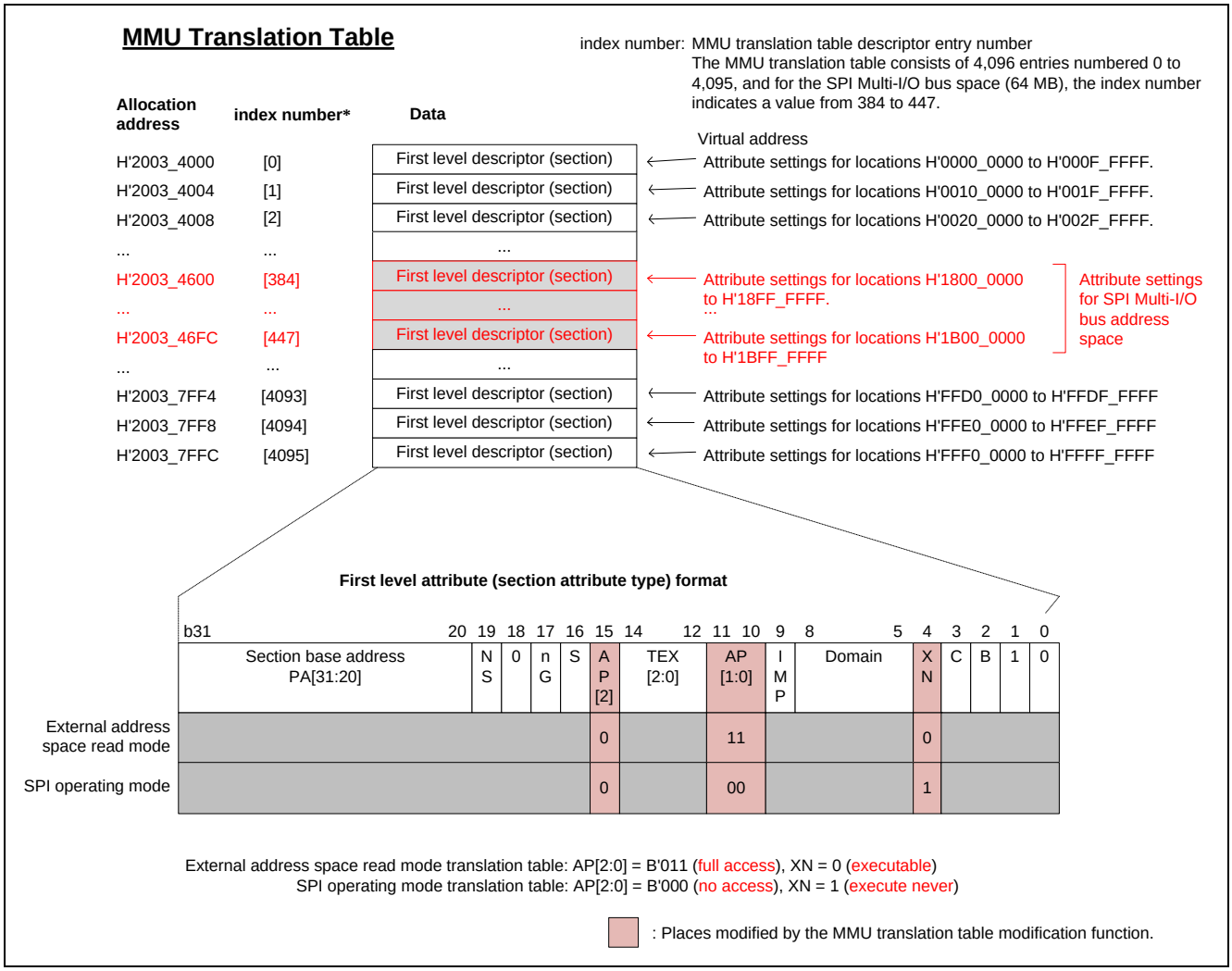


Figure 6.9 MMU Translation Table Settings Used in This Sample Code

6.6 Sample Commands

This sample code provides commands to perform erase, write, and read processing for the serial flash memory. These commands are executed when input from the terminal.

After a reset is canceled, when starting up with the serial flash boot operation, the boot program in internal ROM sets the SPIBSC to external address space read mode and executes this sample code with direct read accesses to the serial flash memory. When using these commands to perform erase or write operations on the serial flash memory, since it is necessary to access the serial flash memory in SPI operating mode, this sample code is written so that the procedures described in section 6.3, External Address Space Read Mode and SPI Operating Mode Switching, are performed when executing these commands. For the processing for reading from serial flash memory, this sample code provides two commands: one that performs the read processing in SPI operating mode and one that performs the read processing in external address space read mode. After executing the processing for each command, this sample code sets the SPIBSC operating mode to external address space read mode. Table 6.1 lists the Processing Performed by this Sample Command.

The basic command input procedures for operating this sample code are listed below.

1. From the terminal, enter "SPIBSC" + <Enter>. This switches to sample code processing.
2. From the terminal, enter "XREAD" + <Enter>. The CPU reads directly from serial flash memory connected to the SPI multi-I/O bus space.
3. From the terminal, enter "ERASE" + <Enter>. This erases the data in the serial flash memory.
4. From the terminal, enter "WRITE" + <Enter>. This writes data to the serial flash memory.
5. From the terminal, enter "SREAD" + <Enter>. This reads data from the serial flash memory.

Table 6.1 Processing Performed by this Sample Command

Sample Processing	Description	Command
Direct read (XIP) processing of the SPI multi-I/O bus space	Directly reads, and displays on the terminal, the data in serial flash memory connected to the SPI multi-I/O bus space in external address space read mode.	XREAD
Erase processing	Erases the serial flash memory. In particular, after switching to SPI operating mode, this function issues an erase command to the serial flash memory using the R_SFLASH_EraseSector() function. After the erase completes, this sample command operates in external address space read mode.	ERASE
Write processing	After switching to SPI operating mode, uses the R_SLASH_ByteProgram() function to issue a program command to the serial flash memory and write the test data using software processing. After the write processing completes, this sample command operates in external address space read mode.	WRITE
Read processing using SPI operating mode	After switching to SPI operating mode, uses the R_SLASH_ByteRead() function to issue a memory read command to the serial flash memory and read data using software processing. The read data is then displayed on the terminal. After the read processing completes, this sample command operates in external address space read mode.	SREAD

6.6.1 XREAD Command

The XREAD command allows an address (H'1800_0000 to H1BFF_FFFF) in the SPI multi-I/O bus space and a byte count to be specified from the terminal. The CPU directly reads the specified number of bytes of data starting at the specified start address and displays the read data on the terminal.

Figure 6.10 shows an example of the Terminal Output Example when XREAD Command Executed.

```
SPIBSC> XREAD                                                    --- [1]
Please input start address (hex) ?
18000000                                                         --- [2]
Please input size ?
256                                                              --- [3]



address      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F    --- [4]
-------------------------------------------------------------
0x18000000 : 18 f0 9f e5 18 f0 9f e5 18 f0 9f e5 18 f0 9f e5
0x18000010 : 18 f0 9f e5 18 f0 9f e5 18 f0 9f e5 18 f0 9f e5
0x18000020 : 00 40 00 18 60 40 00 18 64 40 00 18 68 40 00 18

(Omitted)

0x180000D0 : ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0x180000E0 : ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0x180000F0 : ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

SPIBSC>
```

Figure 6.10 Terminal Output Example when XREAD Command Executed

[1] When "XREAD" + <Enter> are entered, the XREAD command is started.

[2] When the "Please input start address (hex)?" prompt is displayed, enter the read start address in hexadecimal and press <Enter>.
(The allowable range for the address is from H'1800_0000 to H'1BFF_FFFF.)

[3] When the "Please input size?" prompt is displayed, enter the number of bytes to read (as a decimal value) and press <Enter>.
(The byte count values that may be entered are from 1 to 4096. Also, the command terminates with an error if a read address outside the range H'1800_0000 to H1BFF_FFFF is specified.)

[4] The result of the CPU directly reading, in external address space read mode, the area with the specified address and byte count is displayed.

6.6.2 ERASE Command

The ERASE command allows an address (H'0010_0000 to H03FF_FFFF) in serial flash memory and a byte count (1 to 1048576) to be specified from the terminal. The command calculates the block count (the block size is 4 KB) to be erased from the specified byte count and calls the R_SFLASH_EraseSector() function to erase that data in the serial flash memory.

Figure 6.11 shows an example of the Terminal Output Example when ERASE Command Executed.

```
SPIBSC> ERASE                                                    --- [1]
Please input start address (hex) ?
100000                                                           --- [2]
Please input size ?
256                                                              --- [3]

 ERASE command: success                                          --- [4]
 Erased 4096 bytes data from 0x00100000 to 0x00100FFF.

SPIBSC>
```

Figure 6.11 Terminal Output Example when ERASE Command Executed

[1] When "ERASE" + <Enter> are entered, the ERASE command is started.

[2] When the "Please input start address (hex)?" prompt is displayed, enter the erase start address in hexadecimal and press <Enter>.
(The allowable range for the address is from H'0010_0000 to H'03FF_FFFF. The 1 MB area in serial flash memory from H'0000_0000 to H'000F_FFFF holds this sample code and may not be erased.)

[3] When the "Please input size?" prompt is displayed, enter the number of bytes to erase (as a decimal value) and press <Enter>.
(The byte count values that may be entered are from 1 to 1048576. Also, the command terminates with an error if the start address plus the byte count exceeds the range H'0010_0000 to H'03FF_FFFF.)

[4] The command then erases an area with the specified address and byte count. The ERASE command terminates after displaying the result of command execution.

6.6.3 WRITE Command

The WRITE command allows an address (H'0010_0000 to H'03FF_FFFF) in serial flash memory and a byte count (1 to 1048576) to be specified from the terminal and it writes the specified number of bytes of data to serial flash memory starting at the specified address. It also acquires the initial value (0 to 255) for the write data from the terminal, and uses that initial value to generate consecutive data in the amount specified to write. It calls the R_SFLASH_ByteProgram() function to write that data to serial flash memory. It then calls the R_SFLASH_ByteRead() function to read back that data from serial flash memory and verify the written data.

Note that before using the WRITE command, the user must use the ERASE command to erase the area in serial flash memory that is to be written. If the area is not erased, the intended values will not be written to the serial flash memory.

Figure 6.12 shows an example of the Terminal Output Example when WRITE Command Executed.

```
SPIBSC> WRITE                                                    --- [1]
Please input start address (hex) ?
100000                                                           --- [2]
Please input size ?
256                                                              --- [3]
Please input start value to write ?
1                                                                --- [4]

Are you sure data is erased before write operation (Y/N)?
Y                                                                --- [5]

 WRITE command: success                                          --- [6]
 Wrote 256 bytes data from 0x00100000 to 0x001000FF.
 Verify OK.

SPIBSC>
```

Figure 6.12 Terminal Output Example when WRITE Command Executed

[1] When "WRITE" + <Enter> are entered, the WRITE command is started.

[2] When the "Please input start address (hex)?" prompt is displayed, enter the write start address in hexadecimal and press <Enter>.
(The allowable range for the address is from H'0010_0000 to H'03FF_FFFF, and the value must be a multiple of 256 bytes (the page size). The 1 MB area in serial flash memory from H'0000_0000 to H'000F_FFFF holds this sample code and may not be written.)

[3] When the "Please input size?" prompt is displayed, enter the number of bytes to write (as a decimal value) and press <Enter>.
(The byte count values that may be entered are from 1 to 1048576. Also, the command terminates with an error if the start address plus the byte count exceeds the range H'0010_0000 to H'03FF_FFFF.)

[4] When the "Please input start value to write?" prompt is displayed, enter the starting value of the consecutive data to write and press <Enter>.
(The allowable range for this value is 0 to 255. The command generates consecutive data in the range 0 to 255.)

[5] When the "Are you sure data is erased before the write operation (Y/N)?" prompt is displayed, enter "Y" or "N" and then press <Enter>. If an ERASE command was performed before executing this command, enter Y. If N is entered this command will exit without performing the write operation.

[6] The command performs a write operation for the specified address and number of bytes. After displaying the result of executing the command, the WRITE command terminates. The result of the verification following execution of the command is also displayed.

If an error occurs during verification, processing continues until writing of the specified number of bytes of write data completes and on the "Verify NG:" output line, "<number of verify error bytes>/<number of bytes written>" is displayed. Verification is performed in order of increasing addresses, and if one or more verification errors occur, the information for the address at which the first verification error occurs is displayed. This information consists of the write address, written value, and read value.

Figure 6.13 shows an example of Terminal Output during WRITE Command Execution when a Verification Error Occurs.

```
SPIBSC> WRITE
Please input start address (hex) ?
100000
Please input size ?
256
Please input start value to write ?
1

Are you sure data is erased before write operation (Y/N)?
Y

 WRITE command: success
 Wrote 256 bytes data from 0x00100000 to 0x001000FF.
 Verify NG              : 7 / 256
   Verify error data : address=0x00100000, write=0x11, read=0x01

SPIBSC>
```

Figure 6.13 Terminal Output during WRITE Command Execution when a Verification Error Occurs

6.6.4 SREAD Command

The SREAD command allows an address (H'0000_0000 to H'03FF_FFFF) in serial flash memory and a number of bytes to read (1 to 4096) to be specified from the terminal. After switching the SPIBSC to SPI operating mode, it calls the R_SFLASH_ByteRead() function to read from serial flash memory and displays the read data on the terminal.

Figure 6.14 shows an example of the Terminal Output Example when SREAD Command Executed.

```
SPIBSC> SREAD                                                    --- [1]
Please input start address (hex) ?
100000                                                           --- [2]
Please input size ?
256                                                              --- [3]



address      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F     --- [4]
-------------------------------------------------------------
0x00100000 : 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10
0x00100010 : 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20
0x00100020 : 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30

(Omitted)

0x001000D0 : d1 d2 d3 d4 d5 d6 d7 d8 d9 da db dc dd de df e0
0x001000E0 : e1 e2 e3 e4 e5 e6 e7 e8 e9 ea eb ec ed ee ef f0
0x001000F0 : f1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ff 00

SPIBSC>
```

Figure 6.14 Terminal Output Example when SREAD Command Executed

[1] When "SREAD" + <Enter> are entered, the SREAD command is started.
[2] When the "Please input start address (hex)?" prompt is displayed, enter the read start address in hexadecimal and press <Enter>.
(The allowable range for the address is from H'0000_0000 to H'03FF_FFFF.)
[3] When the "Please input size?" prompt is displayed, enter the number of bytes to read (as a decimal value) and press <Enter>.
(The byte count values that may be entered are from 1 to 4096. Also, the command terminates with an error if the start address plus the byte count exceeds the range H'0000_0000 to H'03FF_FFFF.)
[4] The command displays the result of reading in SPI operating mode the specified number of bytes from the specified address.

6.7 Peripheral Functions and Memory Allocation in Sample Code

6.7.1 Setting for Peripheral Functions

Table 6.2 lists the Setting for Peripheral Functions during execution of this sample code.

Table 6.2 Setting for Peripheral Functions

Module	Settings
CPG	CPU clock (Iϕ): 400 MHz Internal bus clock (Bϕ): 133.33 MHz Peripheral clock 1 (P1ϕ): 66.67 MHz Peripheral clock 0 (P0ϕ): 33.33 MHz
SPIBSC	Settings to set the SPIBSC to external address space read mode and settings for signal generation so that the CPU can directly read from the serial flash memory connected to the SPI multi-I/O bus space.
PORT	Settings for the PORT4 multiplexed pin functions: P4_4: SPBCLK_0 P4_5: SPBSSL_0 P4_6: SPBMO0_0/SPBIO00_0 P4_7: SPBMI0_0/SPBIO10_0 P4_2: SPBIO20_0 P4_3: SPBIO30_0
STB	Clock supply to peripheral functions and write enable of the internal RAM used for data retention Provides clock supply with STBCR2 to STBCR12 and provides clock supply for all peripheral functions that support stop control.
SCIF	The SCIF channel 0 is set to asynchronous mode. • Data length: 8 bits • Number of stop bits: 1 • Parity: none When P1ϕ is 66.67 MHz, the clock source is not divided, the bit rate value is set to 17, and other settings are set so that the bit rate becomes 115,200 bps. The error margin is 0.46%.

6.7.2 Memory Mapping

Figure 6.15 shows the RZ/A1LU Group Address Space and JASMINE board Memory Mapping.

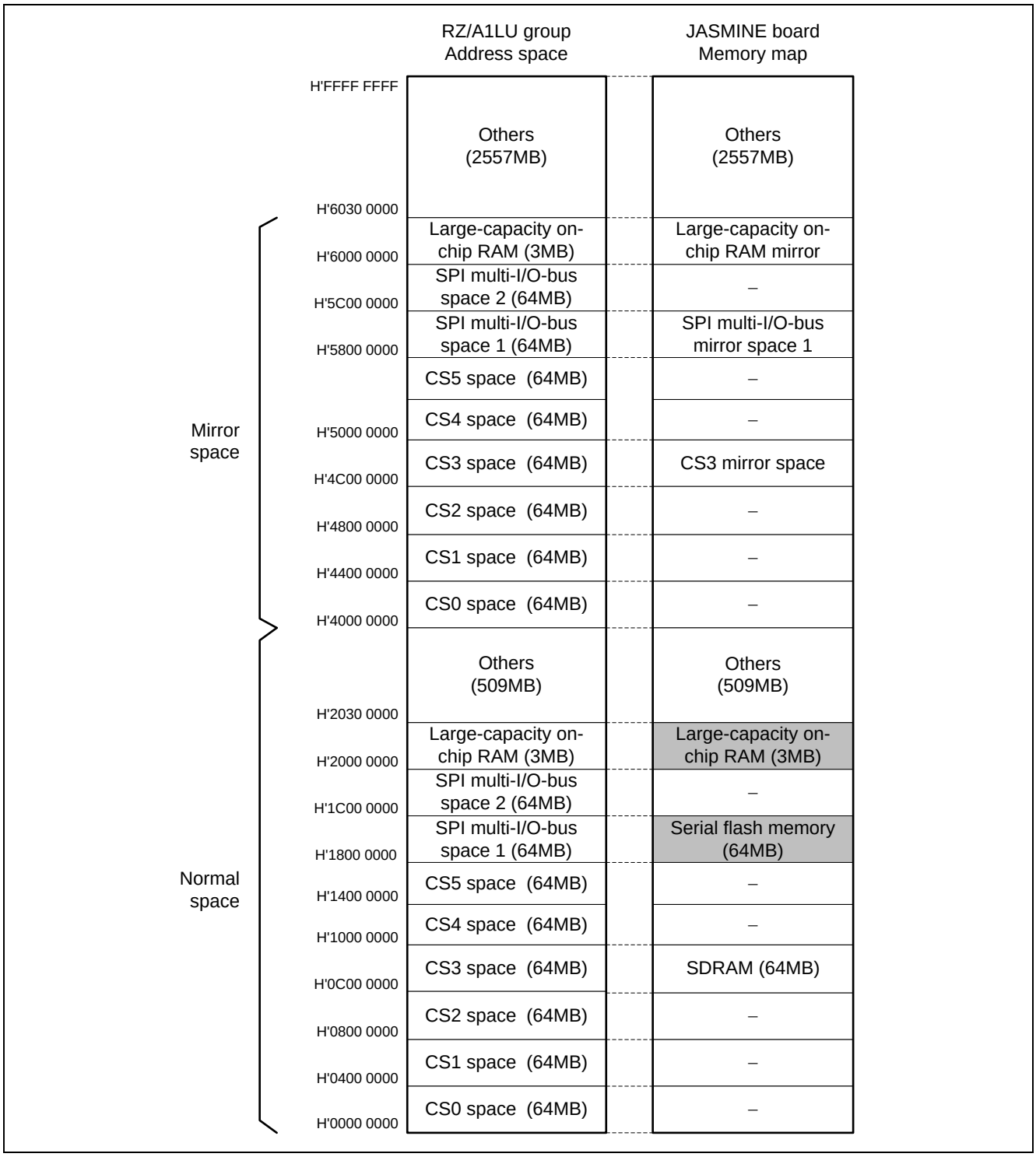


Figure 6.15 RZ/A1LU Group Address Space and JASMINE board Memory Mapping

6.7.3 Section Assignment in Sample Code

Table 6.3 lists the Sections Used by the SPIBSC Initialization Program and Table 6.4 to Table 6.6 list the Sections Used by Application Programs.

Table 6.3 Sections Used by the SPIBSC Initialization Program

Section Name	Description	Type	Loading Area	Execution Area
VECTOR_TABLE	Exception processing vector table	Code	S-FLASH	S-FLASH
CODE_SPIBSC_INIT1	Code area for SPIBSC initial setting program 1	Code	S-FLASH	LRAM
CODE_IO_REGRW	Program code area for read/write functions of I/O register	Code	S-FLASH	LRAM
CODE_SPIBSC_INIT2	Code area for SPIBSC initial setting program 2	Code	S-FLASH	LRAM
DATA_SPIBSC_INIT2	Initialized data area for SPIBSC initial setting program 2	RW Data	S-FLASH	LRAM
BSS_SPIBSC_INIT2	Uninitialized data area for SPIBSC initial setting program 2	ZI Data	—	LRAM
RESET_HANDLER	Program code area of reset handler processing	Code	S-FLASH	S-FLASH
CODE	Default program code area Code type sections for which no section is defined by the C code are all allocated to this area.	Code	S-FLASH	S-FLASH
SVC_STACK	Stack area	ZI Data	—	LRAM

Note: For the loading areas and execution areas in the table, S-FLASH indicates the serial flash memory area and LRAM indicates the on-chip large-capacity RAM area.

Table 6.4 Sections Used by Application Programs (1/3)

Section Name	Description	Type	Loading Area	Execution Area
VECTOR_TABLE	Exception processing vector table	Code	S-FLASH	S-FLASH
RESET_HANDLER	Program code area for reset handlers This area consists of the following sections. • INITCA9CACHE (L1 cache settings) • INIT_TTB (MMU settings) • RESET_HGANDLER (Reset handler)	Code	S-FLASH	S-FLASH
CODE_BASIC_SETUP	Program code area to enable write for the on-chip data retention RAM.	Code	S-FLASH	S-FLASH
InRoot	This area consists of sections located in the root area such as C standard library.	Code or RO Data	S-FLASH	S-FLASH
CODE_FPU_INIT	Program code area for NEON and VFP initialization This area consists of the following sections. • CODE_FPU_INIT • FPU_INIT	Code	S-FLASH	S-FLASH
CODE_RESET	Program code area for hardware initialization This area consists of the following sections. • CODE_RESET (Startup processing) • INIT_VBAR (Vector base initialization)	Code	S-FLASH	S-FLASH
CODE	Default program code area Code type sections for which no section is defined by the C code are all allocated to this area.	Code	S-FLASH	S-FLASH
CONST	Default constant data area RO Data type sections for which no section is defined by the C code are all allocated to this area.	RO Data	S-FLASH	S-FLASH

Table 6.5 Sections Used by Application Programs (2/3)

Section Name	Description	Type	Loading Area	Execution Area
VECTOR_MIRROR_TABLE	Exception handling vector table (Section used for copying to on-chip large-capacity RAM and executing)	Code	S-FLASH	LRAM
CODE_HANDLER_JMPTBL	Program code area for user-defined functions used as IRQ interrupt handlers	Code	S-FLASH	LRAM
CODE_HANDLER	Program code area for IRQ interrupt handlers This area consists of the following sections. • CODE_HANDLER • IRQ_FIQ_HANDLER	Code	S-FLASH	LRAM
CODE_IO_REGRW	Program code area for I/O register read/write functions	Code	S-FLASH	LRAM
CODE_CACHE_OPERATION	Program code area for L1 and L2 cache setup processing*3	Code	S-FLASH	LRAM
DATA_HANDLER_JMPTBL	Registered table data for user-defined functions used as IRQ interrupt handlers	RW Data	S-FLASH	LRAM
ARM_LIB_STACK	Application stack area	ZI Data	—	LRAM
IRQ_STACK	IRQ mode stack area	ZI Data	—	LRAM
FIQ_STACK	FIQ mode stack area	ZI Data	—	LRAM
SVC_STACK	Supervisor mode (SVC) stack area	ZI Data	—	LRAM
ABT_STACK	Abort mode (ABT) stack area	ZI Data	—	LRAM
TTB	MMU translation table area	ZI Data	—	LRAM
ARM_LIB_HEAP	Application heap area	ZI Data	—	LRAM
DATA	Data area with default initialization RW Data type sections for which no section is defined by the C code are all allocated to this area.	RW Data	S-FLASH	LRAM
BSS	Data area without default initialization ZI Data type sections for which no section is defined by the C code are all allocated to this area.	ZI Data	—	LRAM

Table 6.6 Sections Used by Application Programs (3/3)

Section Name	Description	Type	Loading Area	Execution Area
CODE_SPIBSC_ WRITE_OPERATION	Program code for SPIBSC mode switching	Code	S-FLASH	LRAM
CONST_SPIBSC_ WRITE_OPERATION	Constant data area for SPIBSC mode switching	RO Data	S-FLASH	LRAM
DATA_SPIBSC_ WRITE_OPERATION	Initialized data area for SPIBSC mode switching	RW Data	S-FLASH	LRAM
BSS_SPIBSC_ WRITE_OPERATION	Uninitialized data area for SPIBSC mode switching	ZI Data	—	LRAM
CODE_MMU_ OPERATION	Program code for MMU translation table modification processing	Code	S-FLASH	LRAM
CONST_MMU_ OPERATION	Constant data area for MMU translation table modification processing	RO Data	S-FLASH	LRAM
DATA_MMU_ OPERATION	Initialized data area for MMU translation table modification processing	RW Data	S-FLASH	LRAM
BSS_MMU_ OPERATION	Uninitialized data area MMU translation table modification processing	ZI Data	—	LRAM

Notes: 1. For the loading areas and execution areas in the table, S-FLASH indicates the serial flash memory area and LRAM indicates the on-chip large-capacity RAM area.
2. Although as a rule, the section uses the same name as the area, the following areas contain multiple sections: RESET_HANDLER, InRoot, CODE_FPU_INIT, CODE_RESET, CODE, CONST, CODE_HANDLER, DATA, and BSS. See the ARM compiler tool chain manuals for more information on areas and sections.
3. This section must be allocated to a cache disabled area.

6.8 Interrupt Used

Table 6.7 lists the Interrupt Used in This Sample Code.

Table 6.7 Interrupt Used in This Sample Code

Interrupt Source (Interrupt ID)	Priority	Processing Overview
OSTM0 (134)	5	Generates an interrupt every 500 ms.

6.9 Constants

Table 6.8 to Table 6.10 list the Constants Used in This Sample Code.

Table 6.8 Constants Used in This Sample Code (1/3)

Constant Name	Setting Value	Description
SPIBSC_1BIT	0	Sets the bit width used for commands, optional commands, addresses, option data, and transfer data to 1 bit.
SPIBSC_4BIT	2	Sets the bit width used for commands, optional commands, addresses, option data, and transfer data to 4 bits.
SPIBSC_CMNCR_BSZ_SINGLE	0	Sets the number of serial flash memory devices connected to the channel to 1 device.
SPIBSC_CMNCR_BSZ_DUAL	1	Sets the number of serial flash memory devices connected to the channel to 2 devices.
SPIBSC_OUTPUT_DISABLE	0	Specifies that command, optional command, address, and option data are not output.
SPIBSC_OUTPUT_ENABLE	1	Specifies that command, optional command, address, and option data are output.
SPIBSC_OUTPUT_ADDR_24	0x07	Outputs 24-bit addresses.
SPIBSC_OUTPUT_ADDR_32	0x0f	Outputs 32-bit addresses.
SPIBSC_OUTPUT_OPD_3	0x08	Outputs the option data enable OPD3.
SPIBSC_OUTPUT_OPD_32	0x0c	Outputs the option data enable OPD3 and OPD2.
SPIBSC_OUTPUT_OPD_321	0x0e	Outputs the option data enable OPD3, OPD2, and OPD1.
SPIBSC_OUTPUT_OPD_3210	0x0f	Outputs the option data enable OPD3, OPD2, OPD1, and OPD0.
SPIBSC_OUTPUT_SPID_8	0x08	Sets the transfer data enable to 8 (or 16) bits in SPI operating mode.
SPIBSC_OUTPUT_SPID_16	0x0c	Sets the transfer data enable to 16 (or 32) bits in SPI operating mode.
SPIBSC_OUTPUT_SPID_32	0x0f	Sets the transfer data enable to 32 (or 64) bits in SPI operating mode.
SPIBSC_SPISSL_NEGATE	0	Sets the SPBSSL signal status after a transfer complete to negated in SPI operating mode.
SPIBSC_SPISSL_KEEP	1	Sets the SPBSSL signal levels between a transfer complete until the start of the next access to be held in SPI operating mode.
SPIBSC_SPIDATA_DISABLE	0	Sets data read/write operations to disabled in SPI operating mode.
SPIBSC_SPIDATA_ENABLE	1	Sets data read/write operations to enabled in SPI operating mode.

Table 6.9 Constants Used in This Sample Code (2/3)

Constant Name	Setting Value	Description
SPIBSC_DUMMY_CYC_DISABLE	0	Sets dummy cycle insertion to disabled.
SPIBSC_DUMMY_CYC_ENABLE	1	Sets dummy cycle insertion to enabled.
SPIBSC_DUMMY_1CYC	0	Sets the number of dummy cycles output to the serial flash memory in external address space read mode and SPI operating mode to 1.
SPIBSC_DUMMY_2CYC	1	Sets the number of dummy cycles output to the serial flash memory in external address space read mode and SPI operating mode to 2.
SPIBSC_DUMMY_3CYC	2	Sets the number of dummy cycles output to the serial flash memory in external address space read mode and SPI operating mode to 3.
SPIBSC_DUMMY_4CYC	3	Sets the number of dummy cycles output to the serial flash memory in external address space read mode and SPI operating mode to 4.
SPIBSC_DUMMY_5CYC	4	Sets the number of dummy cycles output to the serial flash memory in external address space read mode and SPI operating mode to 5.
SPIBSC_DUMMY_6CYC	5	Sets the number of dummy cycles output to the serial flash memory in external address space read mode and SPI operating mode to 6.
SPIBSC_DUMMY_7CYC	6	Sets the number of dummy cycles output to the serial flash memory in external address space read mode and SPI operating mode to 7.
SPIBSC_DUMMY_8CYC	7	Sets the number of dummy cycles output to the serial flash memory in external address space read mode and SPI operating mode to 8.
SPIBSC_SDR_TRANS	0	Sets the transfer mode in SPI operating mode to SDR transfer.
SPIBSC_DDR_TRANS	1	Sets the transfer mode in SPI operating mode to DDR transfer.
SF_REQ_PROTECT	0	Sets the register setting information in the serial flash memory to protected.
SF_REQ_UNPROTECT	1	Sets the register setting information in the serial flash memory to protection disabled.

Table 6.10 Constants Used in This Sample Code (3/3)

Constant Name	Setting Value	Description
SF_PAGE_SIZE	256	Serial flash memory page size
SF_SECTOR_SIZE	4 * 1024	Serial flash memory sector size
SF_NUM_OF_SECTOR	16384	Serial flash memory sector count
SPIBSC_ADDR_START	0x18000000	SPI multi-I/O bus space start address
SPIBSC_ADDR_END	0x1BFFFFFF	SPI multi-I/O bus space end address
FLASH_ADDR_START	0x00000000	Serial flash memory start address
FLASH_ADDR_END	0x03FFFFFF	Serial flash memory end address
FLASH_PROTECT_SIZE	0x00100000	The size of the erase and write protected area in serial flash memory used in the SPIBSC sample command. (An area of 1 MB is protected so that erase and write operations are not performed on the area in serial flash memory from H'0000_0000 to H'000F_FFFF. This area is protected so that the storage area for this sample code itself is not modified.)
FLASH_SECTOR_BASE_MASK	~(SF_SECTOR_SIZE - 1)	Mask value used to calculate serial flash memory sector start addresses.
FLASH_PAGE_OFFSET_MASK	SF_PAGE_SIZE - 1	Mask value used to calculate serial flash memory offset from page start addresses.
SPIBSC_WRITE_PAGE_BUF_SIZE	SF_SECTOR_SIZE	Size of the write buffers used in the SPIBSC sample command.
SPIBSC_READ_BUF_SIZE	SF_SECTOR_SIZE << 4	Size of the read buffers used in the SPIBSC sample command.
ARGUMENT_BUF_SIZE	128	Size of the buffers used to store text entered from the terminal console in the SPIBSC sample command.

6.10 Structures and Unions

Table 6.11 to Table 6.15 list the structures used in this sample code.

Table 6.16 lists the structure used in MMU translation table first-level descriptor.

Table 6.11 SPIBSC SPI Operating Mode Structure (st_spibsc_spimd_reg_t) (1/5)

Member Name		Description
uint32_t	cdb	Command bit width • Specifies the command bit width in SPI operating mode. • Allowable values: SPIBSC_1BIT: 1 bit SPIBSC_4BIT: 4 bits • The value stored in this member is set into the CDB[1:0] bits in the SPI mode enable setting register (SMENR).
uint32_t	ocdb	Optional command bit width • Specifies the optional command bit width in SPI operating mode. • Allowable values: SPIBSC_1BIT: 1 bit SPIBSC_4BIT: 4 bits • The value stored in this member is set into the OCDB[1:0] bits in the SPI mode enable setting register (SMENR).
uint32_t	adb	Address bit width • Specifies the address bit width in SPI operating mode. • Allowable values: SPIBSC_1BIT: 1 bit SPIBSC_4BIT: 4 bits • The value stored in this member is set into the ADB[1:0] bits in the SPI mode enable setting register (SMENR).
uint32_t	opdb	Option data bit width • Specifies the option data bit width in SPI operating mode. • Allowable values: SPIBSC_1BIT: 1 bit SPIBSC_4BIT: 4 bits • The value stored in this member is set into the OPDB[1:0] bits in the SPI mode enable setting register (SMENR).
uint32_t	spidb	Transfer data bit width • Specifies the transfer data width in SPI operating mode. • Allowable values: SPIBSC_1BIT: 1 bit SPIBSC_4BIT: 4 bits • The value stored in this member is set into the SPIDB[1:0] bits in the SPI mode enable setting register (SMENR).
uint32_t	cde	Sets whether or not commands are output in SPI operating mode. • Allowable values: SPIBSC_OUTPUT_DISABLE: Commands not output SPIBSC_OUTPUT_ENABLE: Commands are output • The value stored in this member is set into the CDE bit in the SPI mode enable setting register (SMENR).

Table 6.12 SPIBSC SPI Operating Mode Structure (st_spibsc_spimd_reg_t) (2/5)

Member	Name	Description
uint32_t	ocde	Optional command enable • Sets whether or not optional commands are output in SPI operating mode. • Allowable values: SPIBSC_OUTPUT_DISABLE: Output disabled. SPIBSC_OUTPUT_ENABLE: Output enabled. • The value stored in this member is set into the OCDE bit in the SPI mode enable setting register (SMENR).
uint32_t	ade	Address enable • Sets whether or not addresses are output in SPI operating mode. • Allowable values: SPIBSC_OUTPUT_DISABLE: Output disabled. SPIBSC_OUTPUT_ADDR_24: The ADR[23:0] bits are output. SPIBSC_OUTPUT_ADDR_32: The ADR[31:0] bits are output. • The value stored in this member is set into the ADE[3:0] bits in the SPI mode enable setting register (SMENR).
uint32_t	opde	Option data enable • Sets whether or not the option data is output in SPI operating mode. • Allowable values: SPIBSC_OUTPUT_DISABLE: Output disabled. SPIBSC_OUTPUT_OPD_3: OPD3 is output. SPIBSC_OUTPUT_OPD_32: OPD3 and OPD2 are output. SPIBSC_OUTPUT_OPD_321: OPD3, OPD2, and OPD1 are output. SPIBSC_OUTPUT_OPD_3210: OPD3, OPD2, OPD1, and OPD0 are output. • The value stored in this member is set into the OPDE[3:0] bits in the SPI mode enable setting register (SMENR).
uint32_t	spide	Transfer data enable • Sets whether or not data transfers are performed in SPI operating mode. • Allowable values: SPIBSC_OUTPUT_DISABLE: Transfers disabled. SPIBSC_OUTPUT_SPID_8: 8 (or 16) bits are transferred. SPIBSC_OUTPUT_SPID_16: 16 (or 32) bits are transferred. SPIBSC_OUTPUT_SPID_32: 32 (or 64) bits are transferred. • The value stored in this member is set into the SPIDE[3:0] bits in the SPI mode enable setting register (SMENR).
uint32_t	sslkp	SPBSSL signal level hold • Sets the SPBSSL signal states after transfer completion in SPI operating mode. • Allowable values: SPIBSC_SPISSL_NEGATE: Signals are negated when a transfer completes. SPIBSC_SPISSL_KEEP: SPBSSL signal levels are retained from transfer completion until the next transfer starts. • The value stored in this member is set into the SSLKP bit in the SPI mode control register (SMCR).

Table 6.13 SPIBSC SPI Operating Mode Structure (st_spibsc_spimd_reg_t) (3/5)

Member	Name	Description
uint32_t	spire	Data read enable • Sets whether or not data is read in SPI operating mode. • Allowable values: SPIBSC_SPIDATA_DISABLE: Data is not read. SPIBSC_SPIDATA_ENABLE: Data is read. • The value stored in this member is set into the SPIRE bit in the SPI mode control register (SMCR).
uint32_t	spiwe	Data write enable • Sets whether or not data is written in SPI operating mode. • Allowable values: SPIBSC_SPIDATA_DISABLE: Data is not written. SPIBSC_SPIDATA_ENABLE: Data is written. • The value stored in this member is set into the SPIWE bit in the SPI mode control register (SMCR).
uint32_t	dme	Dummy cycle enable • Sets whether or not dummy cycles are inserted in SPI operating mode. • Allowable values: SPIBSC_DUMMY_CYC_DISABLE: Dummy cycles not inserted. SPIBSC_DUMMY_CYC_ENABLE: Dummy cycles inserted. • The value stored in this member is set into the DME bit in the SPI mode enable setting register (SMENR).
uint32_t	addre	Address DDR enable • Selects SDR or DDR transfer for address output in SPI operating mode. • Allowable values: SPIBSC_SDR_TRANS: SDR transfer SPIBSC_DDR_TRANS: DDR transfer • The value stored in this member is set into the ADDRE bit in the SPI mode DDR enable register (SMDRENR).
uint32_t	opdre	Option data DDR enable • Selects SDR or DDR transfer for option data output in SPI operating mode. • Allowable values: SPIBSC_SDR_TRANS: SDR transfer SPIBSC_DDR_TRANS: DDR transfer • The value stored in this member is set into the OPDRE bit in the SPI mode DDR enable register (SMDRENR).
uint32_t	spidre	Transfer data DDR enable • Selects SDR or DDR transfer for data transferred in SPI operating mode. • Allowable values: SPIBSC_SDR_TRANS: SDR transfer SPIBSC_DDR_TRANS: DDR transfer • The value stored in this member is set into the SPIDRE bit in the SPI mode DDR enable register (SMDRENR).

Table 6.14 SPIBSC SPI Operating Mode Structure (st_spibsc_spimd_reg_t) (4/5)

Member	Name	Description
uint8_t	dmdb	Dummy cycle bit width • Sets the bit width for dummy cycles in SPI operating mode. • Allowable values: SPIBSC_1BIT: 1 bit SPIBSC_4BIT: 4 bits • The value stored in this member is set into the DMDB[1:0] bits in the SPI mode dummy cycle setting register (SMDMCR).
uint8_t	dmcyc	Dummy cycle count • Sets the number of dummy cycles in SPI operating mode. • Allowable values: SPIBSC_DUMMY_1CYC: 1 cycle SPIBSC_DUMMY_2CYC: 2 cycles SPIBSC_DUMMY_3CYC: 3 cycles SPIBSC_DUMMY_4CYC: 4 cycles SPIBSC_DUMMY_5CYC: 5 cycles SPIBSC_DUMMY_6CYC: 6 cycles SPIBSC_DUMMY_7CYC: 7 cycles SPIBSC_DUMMY_8CYC: 8 cycles • The value stored in this member is set into the DMCYC[2:0] bits in the SPI mode dummy cycle setting register (SMDMCR).
uint8_t	cmd	Command • Sets the command output in SPI operating mode. • The value stored in this member is set into the CMD[7:0] bits in the SPI mode command setting register (SMCMR).
uint8_t	ocmd	Optional command • Sets the optional command output in SPI operating mode. • The value stored in this member is set into the OCMD[7:0] bits in the SPI mode command setting register (SMCMR).
uint32_t	addr	Address • Sets the address output in SPI operating mode. • The value stored in this member is set into the ADR[31:0] bits in the SPI mode address setting register (SMADR).
uint8_t	opd[4]	Option data • Sets the option data output in SPI operating mode. • The value stored in this member is set into the OPDn[7:0] bits in the SPI mode option setting register (SMOPR). OPD3[7:0] ← opd[0] OPD2[7:0] ← opd[1] OPD1[7:0] ← opd[2] OPD0[7:0] ← opd[3]

Table 6.15 SPIBSC SPI Operating Mode Structure (st_spibsc_spimd_reg_t) (5/5)

Member Name		Description
uint32_t	smrdr[2]	Read data storage buffer • Stores the data (SPI mode read data register n (SMRDRn)) read in SPI operating mode as shown below. SMRDR0 → smrdr[0] SMRDR1 → smrdr[1]
uint32_t	smwdr[2]	Write data storage buffer • Stores the data (SPI mode write data register n (SMWDRn)) to be written in SPI operating mode as shown below. SMWDR0 ← smwdr[0] SMWDR1 ← smwdr[1]

Table 6.16 MMU Translation Table First-Level Descriptor (Section) Descriptor Structure (mmu_ttbl_desc_section_t)

Member Name	Description
uint32_t b0 : 1	First level descriptor (section) bit 0
uint32_t b1 : 1	First level descriptor (section) bit 1
uint32_t B : 1	First level descriptor (section) B bit
uint32_t C : 1	First level descriptor (section) C bit
uint32_t XN: 1	First level descriptor (section) XN bit
uint32_t Domain : 4	First level descriptor (section) Domain bit
uint32_t b9 : 1	First level descriptor (section) bit 9
uint32_t AP1_0 : 2	First level descriptor (section) AP[1:0] bits
uint32_t TEX : 3	First level descriptor (section) TEX[2:0] bits
uint32_t AP2 : 1	First level descriptor (section) AP[2] bit
uint32_t S : 1	First level descriptor (section) S bit
uint32_t nG : 1	First level descriptor (section) nG bit
uint32_t b18: 1	First level descriptor (section) bit 18
uint32_t NS : 1	First level descriptor (section) NS bit
uint32_t base_addr : 12	First level descriptor (section) PA[31:20] bits

6.11 Variables

Table 6.17 lists the Global Variables.

Table 6.17 Global Variables

Type	Variable Name	Description
st_spibsc_spimd_reg_t	g_spibsc_spimd_reg	SPIBSC SPI operating mode settings storage variable • Stores the SPIBSC settings used in SPI operating mode. In the sample code, these settings are also used as arguments when running serial flash control functions within API functions and user-defined functions.
uint8_t	write_page_buf[]	Write buffer used by the SPIBSC sample commands.
uint8_t	read_buf[]	Read buffer used by the SPIBSC sample commands.
char_t	arg_buf[]	Buffer that stores the input character strings from the console in SPIBSC sample commands.

6.12 Functions

This sample code consists of the following functions: interface functions (API functions) required to use the peripheral functions, user-defined functions (functions called from API functions) that the user must provided to match the actual user system, and the sample functions required to run the sample code.

Table 6.18 lists the Sample Function, Table 6.19 lists the API Functions, and Table 6.20 lists the User-Defined Functions.

Note that since the functions called when running the sample commands that perform erase, write, and read operations for the serial flash memory cannot be executed from serial flash memory, they are allocated to on-chip large-capacity RAM and run from on-chip large-capacity RAM. The cache maintenance functions, MMU translation table descriptor acquisition and setting functions are transferred to on-chip large-capacity RAM by scatter loading processing, and the SPIBSC control functions, including the SPIBSC operating mode switching functions, the sample command processing functions, and the user-defined functions, are transferred to on-chip large-capacity RAM by the Sample_SPIBSC_WriteSectionInit() function.

Table 6.18 Sample Function

Function Name	Description
Sample_SPIBSC_Main	SPIBSC sample code main function
Sample_SPIBSC_Xread	XREAD command processing (read in external address space read mode)
Sample_SPIBSC_Erase	ERASE command processing
Sample_SPIBSC_Write	WRITE command processing
Sample_SPIBSC_Sread	SREAD command processing (read in SPI operating mode)
Sample_SPIBSC_WriteSectionInit	Section initialization processing for the SPIBSC sample commands
Sample_SPIBSC_ChangeModeSpi	Processing for switching to SPI operating mode
Sample_SPIBSC_ChangeModeXip	Processing for switching to external address space read mode
Change_MMU_TTbl_SpibscSpi	MMU translation table modification processing for the SPI multi-I/O bus space used in SPI operating mode.
Change_MMU_TTbl_SpibscXip	MMU translation table modification processing for the SPI multi-I/O bus space used in external address space read mode
MMU_TTbl_GetIndex	Processing for getting MMU translation table descriptor index number
MMU_TTbl_GetDesc	Processing for getting MMU translation table descriptor value
MMU_TTbl_SetDesc	Processing for setting MMU translation table descriptor value
L1_I_CacheFlushAll	Flush all of the L1 instruction cache
L1_D_CacheWritebackFlushAll	Writeback and flush all of the L1 data cache
L2CacheWritebackFlushAll	Writeback and flush all of the L2 cache
TLB_FlushAll	Flush all of the TLB

Table 6.19 API Functions

Function Name	Description
R_SFLASH_SpibscStop	SPIBSC stop function Negates SSL and stops access to the serial flash memory
R_SFLASH_Exmode	SPIBSC external address space read mode switching function Switches the SPIBSC from SPI operating mode to external address space read mode.
R_SFLASH_Spimode	SPIBSC SPI operating mode switching function Switches the SPIBSC from external address space read mode to SPI operating mode.
R_SFLASH_Spimode_Init	SPIBSC SPI operating mode initialization function Performs the initializations required to use the SPIBSC in SPI operating mode. After initialization, the mode is set to SPI operating mode.
R_SFLASH_EraseSector	Serial flash memory sector erase function Issues a block erase command to the serial flash memory and performs the erase processing. Set the SPIBSC to SPI operating mode before using this function.
R_SFLASH_ByteProgram	Serial flash memory write function Issues a byte program command to the serial flash memory and performs the write processing. Set the SPIBSC to SPI operating mode before using this function.
R_SFLASH_ByteRead	Serial flash memory read function Issues a read command to the serial flash memory and performs the read processing. Set the SPIBSC to SPI operating mode before using this function.
R_SFLASH_Ctrl_Protect	Serial flash memory protection control function Sets the serial flash memory registers and sets or clears the protection. Set the SPIBSC to SPI operating mode before using this function.
R_SFLASH_Spibsc_Transfer	Serial flash memory command issuing function Issues commands to the serial flash memory according to the content of the arguments. Set the SPIBSC to SPI operating mode before using this function.

Table 6.20 User-Defined Functions

Function Name	Description
Userdef_SFLASH_Write_Enable	Serial flash memory write enable function This function should implement processing that enables writing to the serial flash memory registers according to the serial flash memory actually used. In this sample code, this function issues a "Write Status Register (WRSR)" command for the Macronix serial flash memory (MX25L51245G).
Userdef_SFLASH_Busy_Wait	Serial flash memory wait for write completion function This function should implement processing that waits for completion of a write to the serial flash memory according to the serial flash memory actually used. In this sample code, this function issues a "Read Status Register (RDSR)" command for the Macronix serial flash memory (MX25L51245G), and references the contents of the status register to implement the wait for write completion operation.
Userdef_SFLASH_Ctrl_Protect	Serial flash memory clear protection function This function should implement processing that clears the protection of registers in the serial flash memory according to the serial flash memory actually used. In this sample code, this function clears the protection for the Macronix serial flash memory (MX25L51245G).

6.13 Function Specifications

The sample program specifications are listed below.

Sample_SPIBSC_Main		
Outline	SPIBSC sample code main function	
Declaration	int32_t Sample_SPIBSC_Main(int32_t argc, char_t **argv)	
Description	Displays the SPIBSC sample code information on a terminal program running on the host PC connected by a serial interface to the JASMINE board. Before performing the wait for sample command input, this function calls the Sample_SPIBSC_WriteSectionInit() function to transfer the sample command processing to on-chip large-capacity RAM. When the following commands are entered, it performs the corresponding sample command processing. Input of "XREAD" + <Enter> : Executes the XREAD command. Input of "ERASE" + <Enter> : Executes the ERASE command. Input of "WRITE" + <Enter> : Executes the WRITE command. Input of "SREAD" + <Enter> : Executes the SREAD command.	
Arguments	int32_t argc	Number of command arguments entered from the terminal This argument is not used in this function.
	char_t **argv	Pointer to the command entered from the terminal This argument is not used in this function.
Return Value	COMMAND_EXIT	: The SPIBSC sample code processing has terminated.

Sample_SPIBSC_Xread		
Outline	XREAD command processing	
Declaration	int32_t Sample_SPIBSC_Xread(int32_t argc, char_t **argv)	
Description	This function directly reads with the CPU the area in the SPI multi-I/O bus space with the start address and byte count entered from the terminal and displays the read data on the terminal. The address and byte count for any area in the range H'1800_0000 to H'1BFF_FFFF may be specified. Up to 4 KB may be specified with the byte count. When this function is running, the SPIBSC operates in external address space read mode. This function is called when, after the SPIBSC sample code is started, "XREAD" + <Enter> is entered from the terminal.	
Arguments	int32_t argc	Number of command arguments entered from the terminal This argument is not used in this function.
	char_t **argv	Pointer to the command entered from the terminal This argument is not used in this function.
Return Value	0 : Normal end -1 : Error end	

Sample_SPIBSC_Erase		
Outline	ERASE command processing	
Declaration	int32_t Sample_SPIBSC_Erase(int32_t argc, char_t **argv)	
Description	After switching the SPIBSC to SPI operating mode, this function performs processing that erases serial flash memory for the number of bytes and the start address input from the terminal. The address and byte count for any area in the range H'0010_0000 to H'03FF_FFFF may be specified. Up to 1 MB may be specified with the byte count. This sample code calculates the number of blocks to erase based on the Macronix MX25L51245G serial flash memory, and calls the function R_SFLASH_EraseSector() function for each sector size block to perform this erase processing. After erase processing completes, the SPIBSC is switched to external address space read mode and terminates. After the SPIBSC sample code has been started, this function is called when "ERASE" + <Enter> is entered from the keyboard.	
Arguments	int32_t argc	Number of command arguments entered from the terminal This argument is not used in this function.
	char_t **argv	Pointer to the command entered from the terminal This argument is not used in this function.
Return Value	0 : Normal end -1 : Error end	

Sample_SPIBSC_Write	
Outline	WRITE command processing
Declaration	int32_t Sample_SPIBSC_Write(int32_t argc, char_t **argv)
Description	After switching the SPIBSC to SPI operating mode, this function performs processing to write the input data to serial flash memory, in particular, to the area specified by the number of bytes and the start address input from the terminal. Any area in the range H'0010_0000 to H'03FF_FFFF may be specified with the entered address and byte count. The address must specify the start address (a multiple of 256 bytes) of a page in the serial flash memory, and the byte count must not exceed 1 MB. The entered data item specifies the starting value for the consecutive data to be written and this function generates consecutive data that is incremented over the range 0 to 255 starting with the specified data. This sample code calls the R_SFLASH_ByteProgram() function for each page size increment (256 bytes) in the Macronix MX25L51245G serial flash memory to perform the write processing. After performing this write processing, this function switches the SPIBSC to external address space read mode and terminates. After the SPIBSC sample code has been started, this function is called when "WRITE" + <Enter> is entered from the keyboard. The user must erase the area in serial flash memory to be written before calling this function.
Arguments	int32_t argc — Number of command arguments entered from the terminal This argument is not used in this function. char_t **argv — Pointer to the command entered from the terminal This argument is not used in this function.
Return Value	0 : Normal end -1 : Error end

Sample_SPIBSC_Sread	
Outline	SREAD command processing
Declaration	int32_t Sample_SPIBSC_Sread(int32_t argc, char_t **argv)
Description	After switching the SPIBSC to SPI operating mode, this function performs processing to read data from the area in serial flash memory with the start address and byte count entered from the terminal and display that data on the terminal. Any area in the range H'0000_0000 to H'03FF_FFFF may be specified with the address and byte count entered from the terminal. A byte count of up to 4 KB may be specified. After performing this read processing, this function switches the SPIBSC to external address space read mode and terminates. After the SPIBSC sample code has been started, this function is called when "SREAD" + <Enter> is entered from the keyboard.
Arguments	int32_t argc — Number of command arguments entered from the terminal This argument is not used in this function. char_t **argv — Pointer to the command entered from the terminal This argument is not used in this function.
Return Value	0 : Normal end -1 : Error end

Sample_SPIBSC_WriteSectionInit	
Outline	Section initialization processing for the SPIBSC sample commands
Declaration	void Sample_SPIBSC_WriteSectionInit(void)
Description	Transfers processing to be executed by this sample code to on-chip large-capacity RAM. After switching the SPIBSC to SPI operating mode, this function sets up the system so that the sample commands can be executed from the on-chip large-capacity RAM area.
Arguments	None
Return Value	None

Sample_SPIBSC_ChangeModeSpi	
Outline	Processing for switching to SPI operating mode
Declaration	int32_t Sample_SPIBSC_ChangeModeSpi(void)
Description	Performs processing to switch the SPIBSC from external address space read mode to SPI operating mode. This function calls the Change_MMU_TTbl_SpibscSpi() function and sets the SPI multi-I/O bus space MMU translation table to have the access disabled and execution disabled attributes so that accesses to the SPI multi-I/O bus space are not generated. Also, after setting the MMU translation table, it updates the cache contents using cache maintenance. After that, it calls the R_SFLASH_Spimode() function to set the SPIBSC CMNCR register MD bit to 1, performs a dummy read of the CMNCR register, and switches to SPI operating mode.
Arguments	None
Return Value	0 : Normal end -1 : Error end

Sample_SPIBSC_ChangeModeXip	
Outline	Processing for switching to external address space read mode
Declaration	int32_t Sample_SPIBSC_ChangeModeXip(void)
Description	Performs processing to switch the SPIBSC from SPI operating mode to external address space read mode. First it verifies that all accesses to the serial flash memory have completed by calling the R_SFLASH_Exmode() function and reading the TEND bit. It writes a 1 to the DRCR register RCF bit, performs a dummy read of the DRCR register, and clears the read cache. It sets the CMNCR register MD bit to 0, performs a dummy read of the CMNCR register, and switches to external address space read mode. After that, it calls the Change_MMU_TTbl_SpibscXip() function, sets the SPI multi-I/O bus space MMU translation table to have the access enabled and execution enabled attributes so that access to the SPI multi-I/O bus space is possible. Also, after setting the MMU translation table, it updates the cache contents using cache maintenance.
Arguments	None
Return Value	0 : Normal end -1 : Error end

Change_MMU_TTbl_SpibscSpi		
Outline	MMU translation table modification processing for the SPI multi-I/O bus space used in SPI operating mode.	
Declaration	static int32_t Change_MMU_TTbl_SpibscSpi(uint32_t start_addr, uint32_t end_addr)	
Description	Sets the AP[2:0] bits and the XN bit as shown below so that the SPI multi-I/O bus space (H'1800_0000 to H'1BFF_FFFF) MMU translation table used in SPI operating mode has the access disabled and execution disabled attributes. AP[2:0] bits : B'000 (No access) XN bit : 1 (Execute never)	
Arguments	uint32_t start_addr	Start address of the address range that is the target of the MMU translation table modification processing
	uint32_t end_addr	End address of the address range that is the target of the MMU translation table modification processing
Return Value	0 : Normal end -1 : Error end	

Change_MMU_TTbl_SpibscXip		
Outline	MMU translation table modification processing for the SPI multi-I/O bus space used in external address space read mode	
Declaration	static int32_t Change_MMU_TTbl_SpibscXip(uint32_t start_addr, uint32_t end_addr)	
Description	Sets the AP[2:0] bits and the XN bit as shown below so that the SPI multi-I/O bus space (H'1800_0000 to H'1BFF_FFFF) MMU translation table used in external address space read mode has the access enabled and execution enabled attributes. AP[2:0] bits : B'011 (Full access) XN bit : 0 (Executable)	
Arguments	uint32_t start_addr	Start address of the address range that is the target of the MMU translation table modification processing
	uint32_t end_addr	End address of the address range that is the target of the MMU translation table modification processing
Return Value	0 : Normal end -1 : Error end	

MMU_TTbl_GetIndex		
Outline	Processing for getting MMU translation table descriptor index number	
Declaration	uint32_t MMU_TTbl_GetIndex(uint32_t addr)	
Description	Gets the index number of the section type descriptor for the MMU translation table corresponding to the addr argument.	
Arguments	uint32_t addr	Virtual address (H'0000_0000 to H'FFFF_FFFF) with which to modify the descriptor contents.
Return Value	Index number (0 to 4095) of the descriptor corresponding to the addr argument.	

MMU_TTbl_GetDesc		
Outline	Processing for getting MMU translation table descriptor value	
Declaration	int32_t MMU_TTbl_GetDesc(uint32_t index, mmu_ttbl_desc_section_t *pdesc)	
Description	Gets the value of the descriptor for the section type for the MMU translation table corresponding to the index argument, and sets the value in *pdesc.	
Arguments	uint32_t index	Index number (0 to 4095) for the MMU translation table
	mmu_ttbl_desc_section_t *pdesc	Pointer to a variable to store the acquired descriptor
Return Value	0 : Normal end	
	-1 : Error end	

MMU_TTbl_SetDesc		
Outline	Processing for setting MMU translation table descriptor value	
Declaration	int32_t MMU_TTbl_SetDesc(uint32_t index, mmu_ttbl_desc_section_t *pdesc)	
Description	Sets the descriptor for the section type for the MMU translation table corresponding to the index argument to the value stored in *pdesc.	
Arguments	uint32_t index	Index number (0 to 4095) for the MMU translation table
	mmu_ttbl_desc_section_t *pdesc	Pointer to a variable that holds the value to which the descriptor is to be set.
Return Value	0 : Normal end	
	-1 : Error end	

L1_I_CacheFlushAll	
Outline	Flush all of the L1 instruction cache
Declaration	void L1_I_CacheFlushAll(void)
Description	Performs a flush of the whole L1 instruction cache.
Arguments	None
Return Value	None

L1_D_CacheWritebackFlushAll	
Outline	Writeback and flush all of the L1 data cache
Declaration	void L1_D_CacheWritebackFlushAll(void)
Description	Performs a writeback and flush of the whole L1 data cache. In this sample code, all data cache lines are cleaned and invalidated by set/way.
Arguments	None
Return Value	None

L2CacheWritebackFlushAll	
Outline	Writeback and flush all of the L2 cache
Declaration	void L2CacheWritebackFlushAll(void)
Description	Performs a writeback and flush of the whole L2 cache. In this sample code, all cache lines are cleaned and invalidated by way.
Arguments	None
Return Value	None

TLB_FlushAll	
Outline	Flush all of the TLB
Declaration	void TLB_FlushAll(void)
Description	Executes a CP15 TLBIALL to flush the whole TLB.
Arguments	None
Return Value	None

R_SFLASH_SpibscStop	
Outline	SPIBSC stop function
Declaration	int32_t R_SFLASH_SpibscStop(uint32_t ch_no)
Description	When this function is called with SPIBSC set to external address space read mode, this function sets the data read control register (DRCR) SSLN bit to 1 to nagate SSL and stops access to the serial flash memory.
Arguments	uint32_t ch_no SPIBSC channel number (only 0 may be specified)
Return Value	0 : Normal end -1 : Error end

R_SFLASH_Exmode	
Outline	SPIBSC external address space read mode switching function
Declaration	int32_t R_SFLASH_Exmode(uint32_t ch_no)
Description	Switches the SPIBSC to external address space read mode. After switching to external address space read mode, before reading the SPI multi-I/O bus space, all the entries in the read cache are cleared.
Arguments	uint32_t ch_no SPIBSC channel number (only 0 may be specified)
Return Value	0 : Normal end -1 : Error end

R_SFLASH_Spimode	
Outline	SPIBSC SPI operating mode switching function
Declaration	int32_t R_SFLASH_Spimode(uint32_t ch_no)
Description	Switches the SPIBSC to SPI operating mode. After switching to SPI operating mode, all the entries in the read cache are cleared.
Arguments	uint32_t ch_no SPIBSC channel number (only 0 may be specified)
Return Value	0 : Normal end -1 : Error end

R_SFLASH_Spimode_Init		
Outline	SPIBSC SPI operating mode initialization function	
Declaration	int32_t R_SFLASH_Spimode_Init(uint32_t ch_no, uint32_t dual, uint8_t data_width, uint8_t spbr, uint8_t brdv, uint8_t addr_mode)	
Description	Initializes the SPIBSC so that it operates in SPI operating mode.	
Arguments	uint32_t ch_no	SPIBSC channel number (only 0 may be specified)
	uint32_t dual	Number of serial flash memory chips connected to the channel SPIBSC_CMNCR_BSZ_SINGLE: One SPIBSC_CMNCR_BSZ_DUAL: Two
	uint8_t data_width	Data transfer bus width between the SPIBSC and the serial flash memory SPIBSC_1BIT: 1 bit SPIBSC_4BIT: 4 bits
	uint8_t spbr	Bit rate setting Specify the setting value for the SPBR[7:0] bits in the bit rate register (SPBCR). The bit rate is determined by a combination of this setting with the bit rate frequency division (brdv) setting.
	uint8_t brdv	Bit rate divisor setting Specify the setting value for the BRDV[1:0] bits in the bit rate register (SPBCR). The serial clock (SPBCLK) bit rate is set by this value in conjunction with the bit rate setting (spbr).
	uint8_t addr_mode	Bit width of addresses issued to the serial flash memory Specifies the bit width of addresses output when commands are output. SPIBSC_OUTPUT_ADDR_24: 24-bit addresses output SPIBSC_OUTPUT_ADDR_32: 32-bit addresses output
Return Value	0 : Normal end -1 : Error end	

Table 6.21 Sample Settings for the spbr and brdv Arguments

spbr setting value (n)	brdv setting value (N)	Divisor	Bit Rate		
			Bϕ = 100 MHz	**Bϕ = 128 MHz**	**Bϕ = 133.33 MHz**
0	0	1	Setting prohibited		
1	0	2	50 Mbps	64 Mbps	66.67 Mbps
2	0	4	25 Mbps	32 Mbps	33.33 Mbps
3	0	6	16.67 Mbps	21.33 Mbps	22.22 Mbps
4	0	8	12.5 Mpbs	16 Mpbs	16.67 Mpbs

R_SFLASH_EraseSector

Outline	Serial flash memory sector erase function	
Declaration	int32_t R_SFLASH_EraseSector(uint32_t addr, uint32_t ch_no, uint32_t dual uint8_t data_width, uint8_t addr_mode)	
Description	Erases data in sector units associated with the serial flash memory address specified in the argument. This function must be called in sector size units for the serial flash memory actually used. This function must be called after switching the SPIBSC to SPI operating mode.	
Arguments	uint32_t addr	Start address to be erased (H'0000_0000 to H'03FF_FFFF)
	uint32_t ch_no	SPIBSC channel number (only 0 may be specified)
	uint32_t dual	Number of serial flash memory chips connected to the channel SPIBSC_CMNCR_BSZ_SINGLE: One SPIBSC_CMNCR_BSZ_DUAL: Two
	uint8_t data_width	Data transfer bus width between the SPIBSC and the serial flash memory SPIBSC_1BIT: 1 bit SPIBSC_4BIT: 4 bits
	uint8_t addr_mode	Bit width of addresses issued to the serial flash memory Specifies the bit width of addresses output when commands are output. SPIBSC_OUTPUT_ADDR_24: 24-bit addresses output SPIBSC_OUTPUT_ADDR_32: 32-bit addresses output
Return Value	0 : Normal end -1 : Error end	

R_SFLASH_ByteProgram

Outline	Serial flash memory write function	
Declaration	int32_t R_SFLASH_ByteProgram(uint32_t addr, uint8_t *buf, int32_t size, uint32_t ch_no, uint32_t dual, uint8_t data_width, uint8_t addr_mode)	
Description	Writes to serial flash memory according to the parameters specified in the arguments. This function writes to serial flash memory in page units. This function must be called so that the value set in the size argument is less than the serial flash memory page size. This function must be called after switching the SPIBSC to SPI operating mode.	
Arguments	uint32_t addr	Write start address (H'0000_0000 to H'03FF_FFFF)
	uint8_t *buf	Buffer that holds the write data
	int32_t size	Number of bytes to write
	uint32_t ch_no	SPIBSC channel number (only 0 may be specified)
	uint32_t dual	Number of serial flash memory chips connected to the channel SPIBSC_CMNCR_BSZ_SINGLE: One SPIBSC_CMNCR_BSZ_DUAL: Two
	uint8_t data_width	Data transfer bus width between the SPIBSC and the serial flash memory SPIBSC_1BIT: 1 bit SPIBSC_4BIT: 4 bits
	uint8_t addr_mode	Bit width of addresses issued to the serial flash memory Specifies the bit width of addresses output when commands are output. SPIBSC_OUTPUT_ADDR_24: 24-bit addresses output SPIBSC_OUTPUT_ADDR_32: 32-bit addresses output
Return Value	0 : Normal end	
	-1 : Error end	

R_SFLASH_ByteRead

Outline	Serial flash memory read function	
Declaration	int32_t R_SFLASH_ByteRead(uint32_t addr, uint8_t *buf, int32_t size, uint32_t ch_no, uint32_t dual, uint8_t data_width, uint8_t addr_mode)	
Description	Reads data from serial flash memory according to the parameters specified in the arguments. This function must be called after switching the SPIBSC to SPI operating mode.	
Arguments	uint32_t addr	Start address for reading (H'0000_0000 to H'03FF_FFFF)
	uint8_t *buf	Buffer that holds the read data
	int32_t size	Number of bytes to read
	uint32_t ch_no	SPIBSC channel number (only 0 may be specified)
	uint32_t dual	Number of serial flash memory chips connected to the channel SPIBSC_CMNCR_BSZ_SINGLE: One SPIBSC_CMNCR_BSZ_DUAL: Two
	uint8_t data_width	Data transfer bus width between the SPIBSC and the serial flash memory SPIBSC_1BIT: 1 bit SPIBSC_4BIT: 4 bits
	uint8_t addr_mode	Bit width of addresses issued to the serial flash memory Specifies the bit width of addresses output when commands are output. SPIBSC_OUTPUT_ADDR_24: 24-bit addresses output SPIBSC_OUTPUT_ADDR_32: 32-bit addresses output
Return Value	0 : Normal end -1 : Error end	

R_SFLASH_Ctrl_Protect		
Outline	Serial flash memory protection control function	
Declaration	int32_t R_SFLASH_Ctrl_Protect(en_sf_req_t req, uint32_t ch_no, uint32_t dual, uint8_t data_width)	
Description	Sets or cancels serial flash memory protection by setting the device's status register according to the parameters specified in the arguments. This function must be called after switching the SPIBSC to SPI operating mode.	
Arguments	en_sf_req_t req	Register setting information SF_REQ_PROTECT: Protected state SF_REQ_UNPROTECT: Unprotected state
	uint32_t ch_no	SPIBSC channel number (only 0 may be specified)
	uint32_t dual	Number of serial flash memory chips connected to the channel SPIBSC_CMNCR_BSZ_SINGLE: One SPIBSC_CMNCR_BSZ_DUAL: Two
	uint8_t data_width	Data transfer bus width between the SPIBSC and the serial flash memory SPIBSC_1BIT: 1 bit SPIBSC_4BIT: 4 bits
Return Value	0 : Normal end -1 : Error end	

R_SFLASH_Spibsc_Transfer		
Outline	Serial flash memory command issuing function	
Declaration	int32_t R_SFLASH_Spibsc_Transfer(uint32_t ch_no, st_spibsc_spimd_reg_t * regset)	
Description	Issues commands to the serial flash memory in SPI operating mode according to the content set in the regset argument.	
Arguments	uint32_t ch_no	SPIBSC channel number (only 0 may be specified)
	st_spibsc_spimd_reg_t * regset	SPIBSC SPI mode setting See Table 6.11 to Table 6.15 for the contents set.
Return Value	0 : Normal end -1 : Error end	
Notes	If this function is called in external address space read mode, it switches to SPI operating mode and issues commands to the serial flash memory.	

Userdef_SFLASH_Write_Enable		
Outline	Serial flash memory write enable function	
Declaration	int32_t Userdef_SFLASH_Write_Enable(uint32_t ch_no)	
Description	This function should implement processing that enables writing to the serial flash memory registers according to the serial flash memory actually used. In this sample code, this function issues a "Write Status Register (WRSR)" command for the Macronix MX25L51245G serial flash memory.	
Arguments	uint32_t ch_no	SPIBSC channel number (only 0 may be specified)
Return Value	0 : Normal end	
	-1 : Error end	

Userdef_SFLASH_Busy_Wait		
Outline	Serial flash memory wait for write completion function	
Declaration	int32_t Userdef_SFLASH_Busy_Wait(uint32_t ch_no, uint32_t dual, uint8_t data_width)	
Description	This function should implement processing that reads out the serial flash memory registers and waits for completion of a write to the serial flash memory according to the serial flash memory actually used. In this sample code, this function issues a "Read Status Register (RDSR)" command for the Macronix MX25L51245G serial flash memory, and references the contents of the status register to implement the wait for write completion operation.	
Arguments	uint32_t ch_no	SPIBSC channel number (only 0 may be specified)
	uint32_t dual	Number of serial flash memory chips connected to the channel SPIBSC_CMNCR_BSZ_SINGLE: One SPIBSC_CMNCR_BSZ_DUAL: Two
	uint8_t data_width	Data transfer bus width between the SPIBSC and the serial flash memory SPIBSC_1BIT: 1 bit SPIBSC_4BIT: 4 bits
Return Value	0 : Normal end	

Userdef_SFLASH_Ctrl_Protect		
Outline	Serial flash memory clear protection function	
Declaration	int32_t Userdef_SFLASH_Ctrl_Protect(en_sf_req_t req, uint32_t ch_no, uint32_t dual, uint8_t data_width)	
Description	This function should implement processing that cancels the protection of the serial flash memory according to the serial flash memory actually used. In this sample code, this function issues a "Write Status Register (WRSR)" command for the Macronix MX25L51245G serial flash memory to cancels the protection.	
Arguments	en_sf_req_t req	Register setting information SF_REQ_PROTECT: Protected state SF_REQ_UNPROTECT: Unprotected state
	uint32_t ch_no	SPIBSC channel number (only 0 may be specified)
	uint32_t dual	Number of serial flash memory chips connected to the channel SPIBSC_CMNCR_BSZ_SINGLE: One SPIBSC_CMNCR_BSZ_DUAL: Two
	uint8_t data_width	Data transfer bus width between the SPIBSC and the serial flash memory SPIBSC_1BIT: 1 bit SPIBSC_4BIT: 4 bits
Return Value	0 : Normal end -1 : Error end	

6.14 Flowcharts

6.14.1 SPIBSC Sample Code Main Processing

Figure 6.16 shows the flowchart for the SPIBSC Sample Code Main Processing. This processing waits for input from the terminal on the host PC and branches to the appropriate processing in the SPIBSC sample code according to the command entered.

Since the functions called when executing the sample code that performs the erase, write, and read processing for the serial flash memory cannot be executed from serial flash memory, they are allocated to on-chip large-capacity RAM, and this processing is performed from on-chip large-capacity RAM. In the Sample_SPIBSC_WriteSectionInit() function called from the Sample_SPIBSC_Main() function, the SPIBSC control functions, including the SPIBSC operating mode switching functions, the sample command processing functions, and the user-defined functions are transferred to on-chip large-capacity RAM.

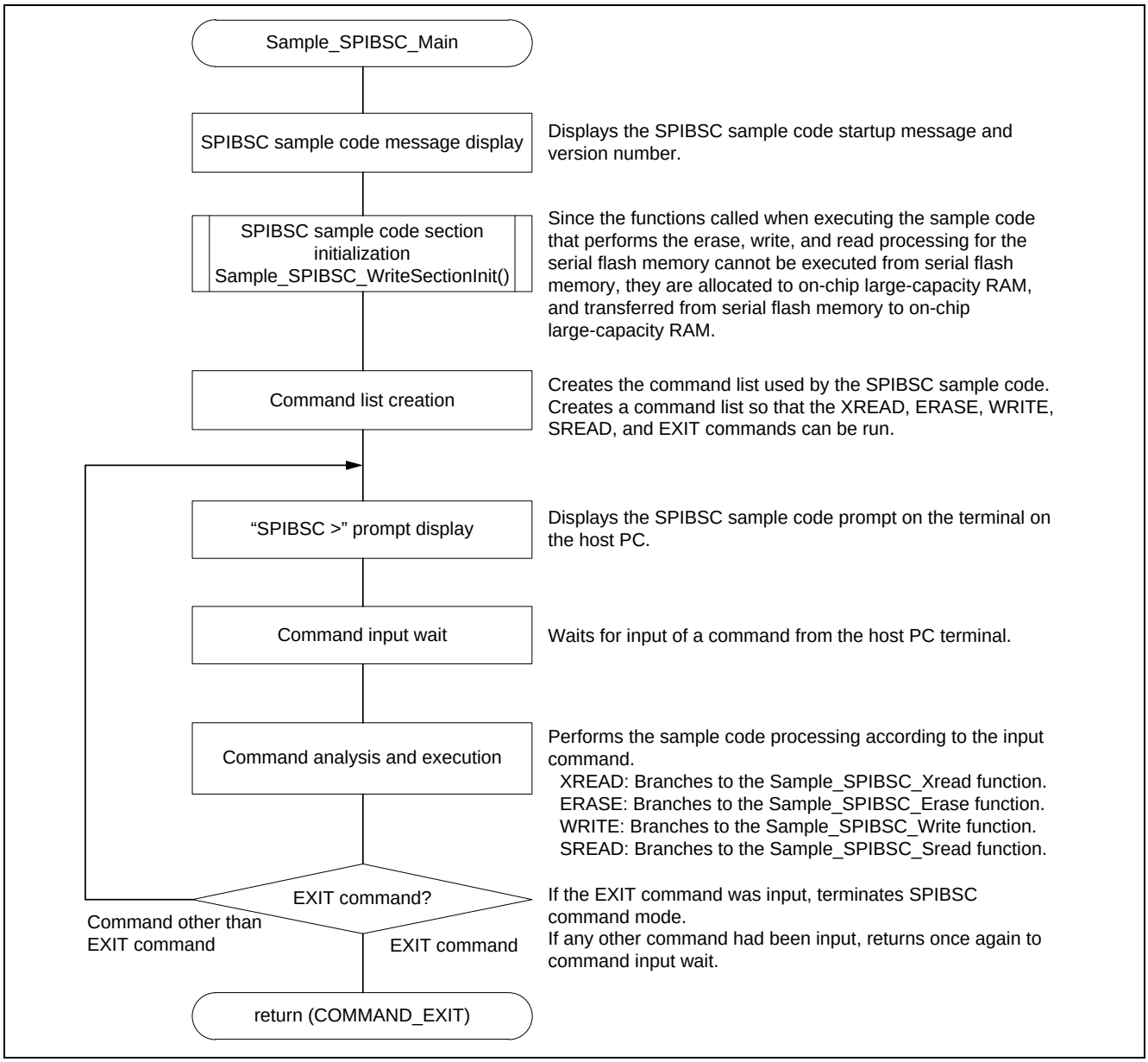


Figure 6.16 SPIBSC Sample Code Main Processing

6.14.2 Processing for Switching to SPI Operating Mode

Figure 6.17 shows the Processing for Switching to SPI Operating Mode. This performs the SPIBSC mode switching described in section 6.3.1, Switching from External Address Space Read Mode to SPI Operating Mode.

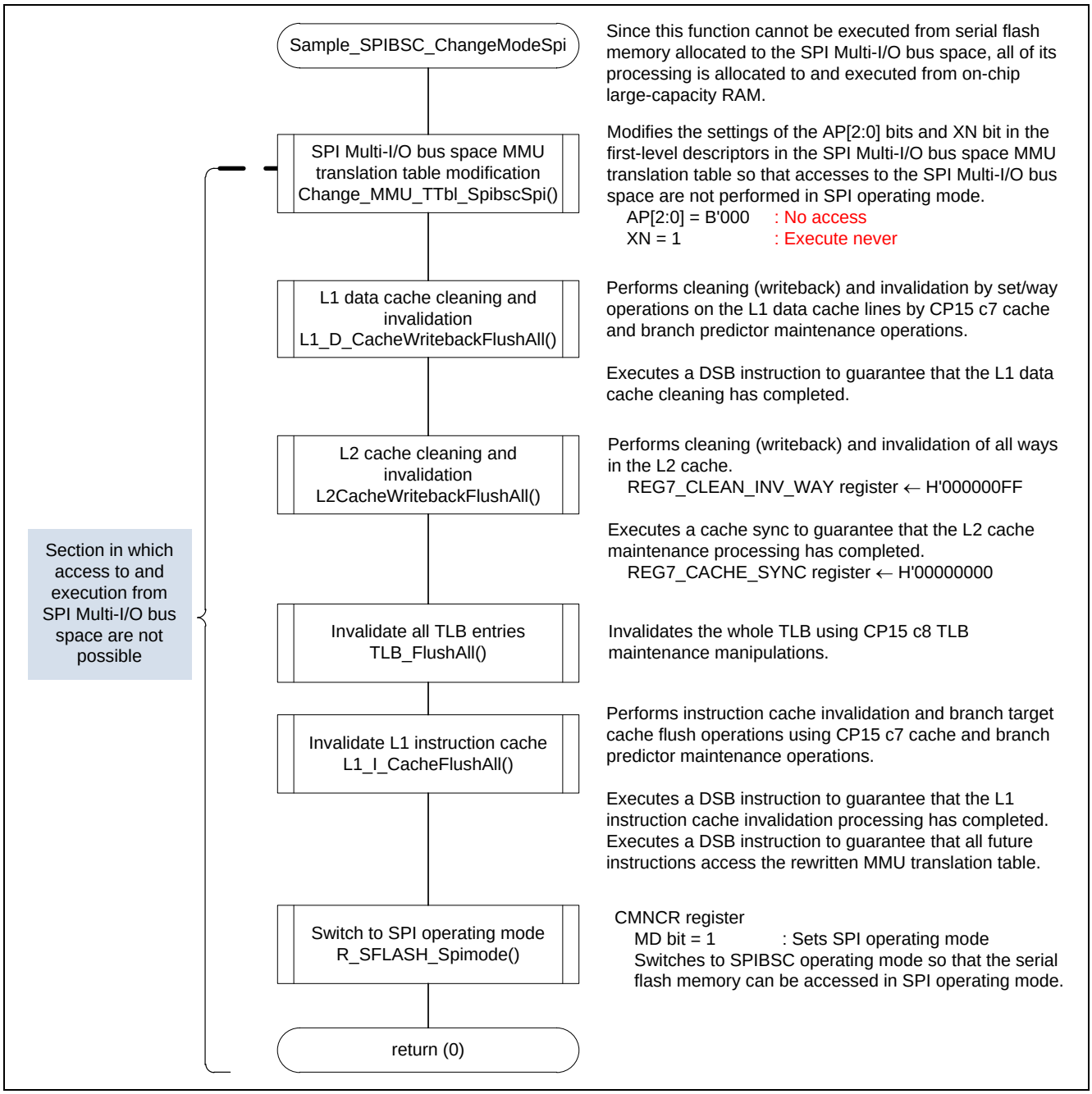


Figure 6.17 Processing for Switching to SPI Operating Mode

6.14.3 Processing for Switching to External Address Space Read Mode

Figure 6.18 shows the Processing for Switching to External Address Space Read Mode. This performs the SPIBSC mode switching described in section 6.3.2, Switching from SPI Operating Mode to External Address Space Read Mode.

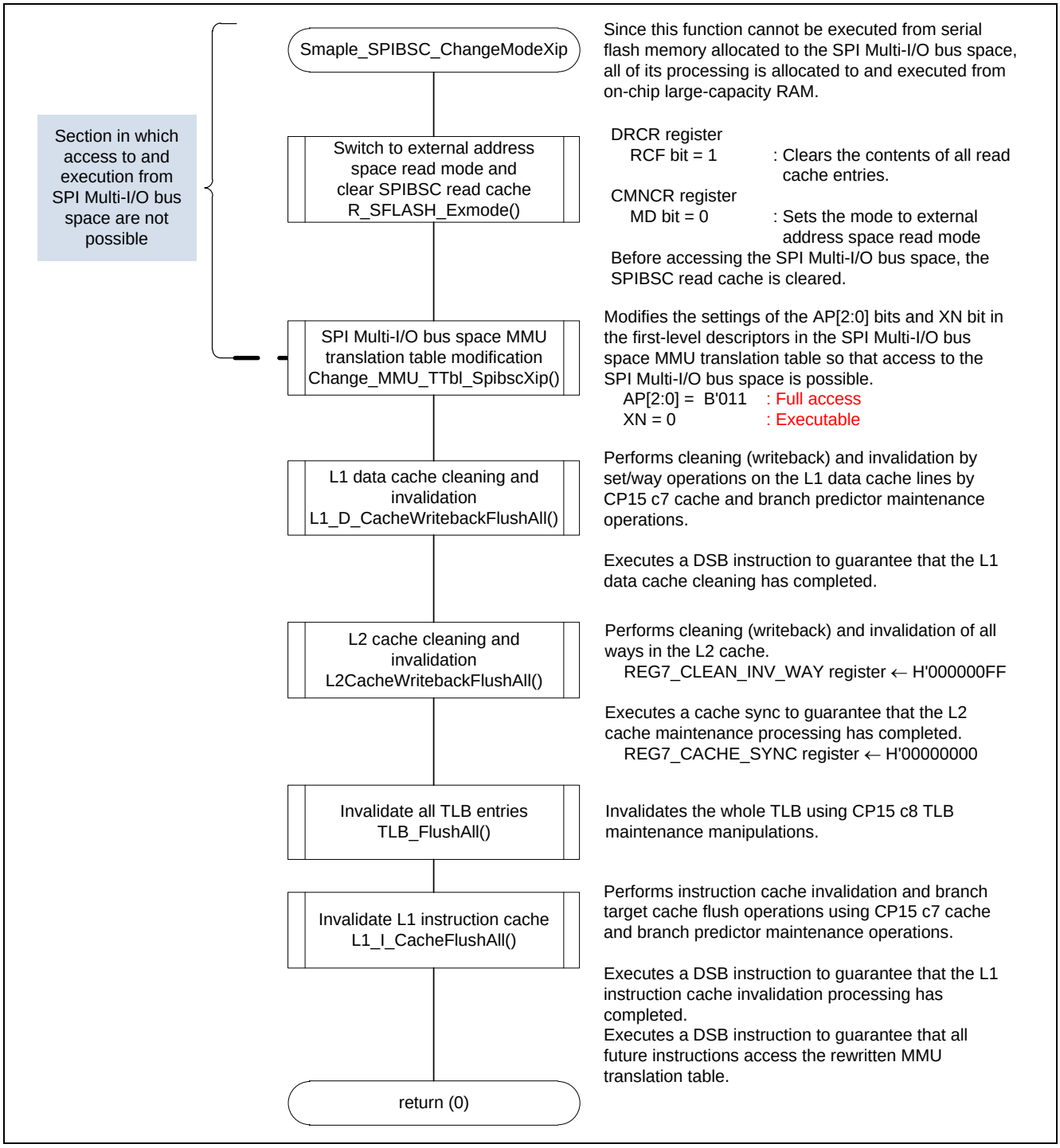


Figure 6.18 Processing for Switching to External Address Space Read Mode

6.14.4 XREAD Command Processing

Figure 6.19 shows the XREAD Command Processing.

The XREAD command processing does not change the SPIBSC mode, but rather operates with the mode remaining unchanged in external address space read mode. In the XREAD command, the CPU reads the SPI multi-I/O bus space to get the serial flash memory data and displays the acquired data on the terminal.

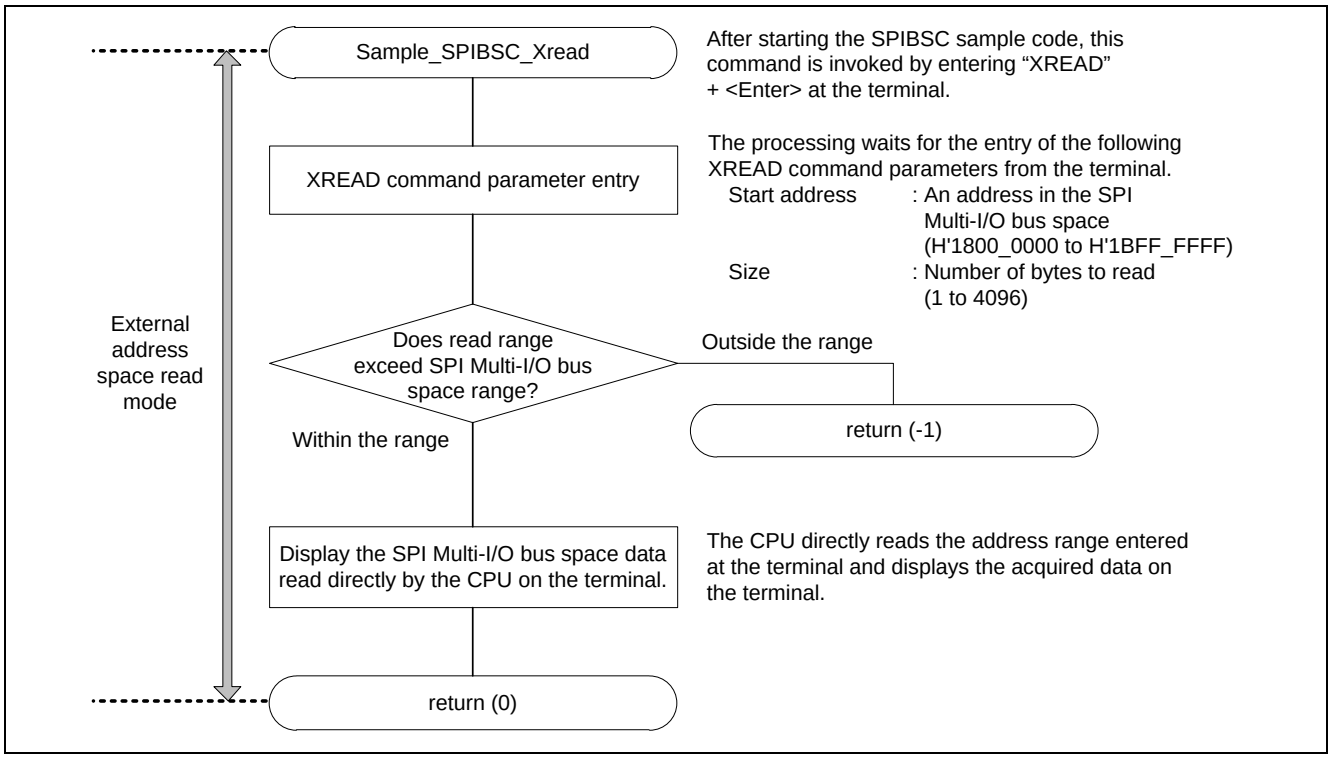


Figure 6.19 XREAD Command Processing

6.14.5 ERASE Command Processing

Figure 6.20 and Figure 6.21 show the ERASE Command Processing.

The ERASE command erases serial flash memory by calling the Sample_SPIBSC_ChangeModeSpi() function to change the SPIBSC mode to SPI operating mode and then calling the R_SFLASH_EraseSector() function to erase data in serial flash memory. After the erase processing completes, this command calls the Sample_SPIBSC_ChangeModeXip() function to perform the processing to change the SPIBSC mode to external address space read mode to set the system to be able to read instructions and data allocated to the SPI multi-I/O bus space (serial flash memory).

In this erase processing, the SPIBSC is set to SPI operating mode, and programs allocated to serial flash memory cannot be executed. Therefore, the ERASE command processing is allocated to and executed from on-chip large-capacity RAM. In SPI operating mode, the SPI multi-I/O bus space cannot be accessed. Therefore, IRQ interrupts are set to the disabled state during erase processing so that branches to programs allocated to serial flash memory do not occur.

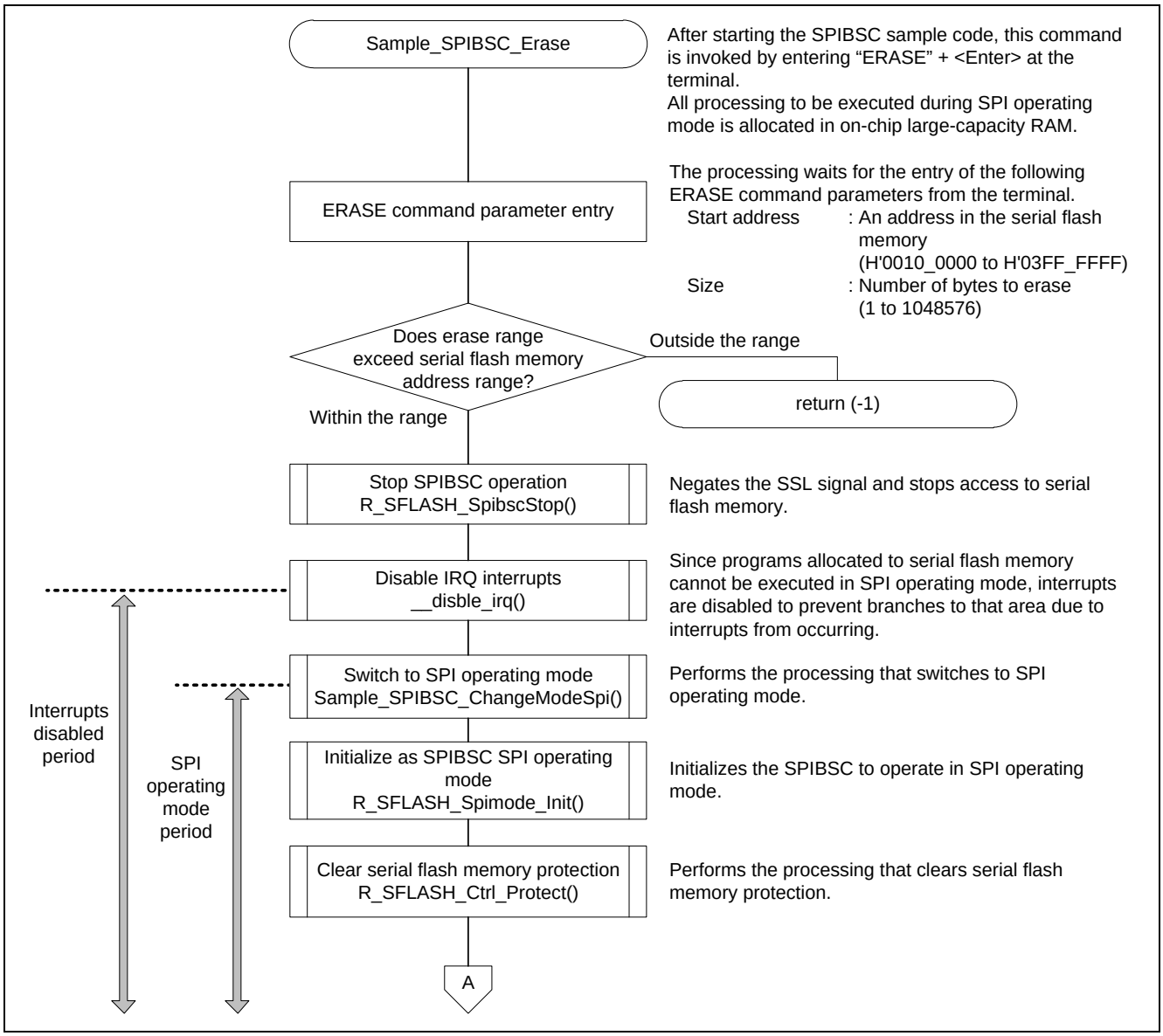


Figure 6.20 ERASE Command Processing (1/2)

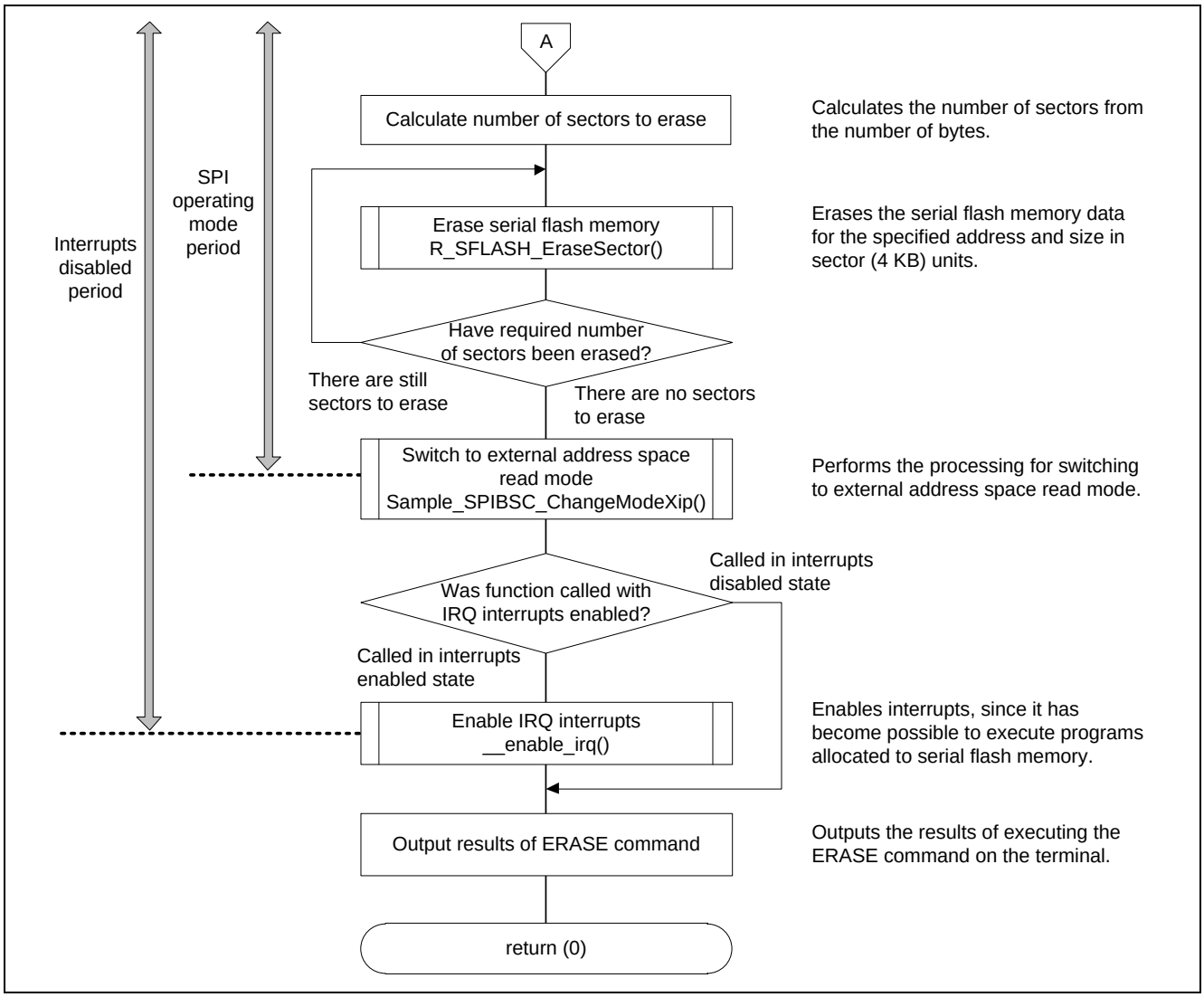


Figure 6.21 ERASE Command Processing (2/2)

6.14.6 WRITE Command Processing

Figure 6.22 and Figure 6.23 show the WRITE Command Processing.

The WRITE command writes data to serial flash memory by calling the Sample_SPIBSC_ChangeModeSpi() function to change the SPIBSC mode to SPI operating mode and then calling the R_SFLASH_ByteProgram() function to write the data to the serial flash memory. After the write processing completes, this command calls the Sample_SPIBSC_ChangeModeXip() function to perform the processing to change the SPIBSC mode to external address space read mode to set the system to be able to read instructions and data allocated to the SPI multi-I/O bus space (serial flash memory).

In this write processing, as in erase processing, the command processing is allocated to and executed from on-chip large-capacity RAM, and IRQ interrupts are disabled during the write processing.

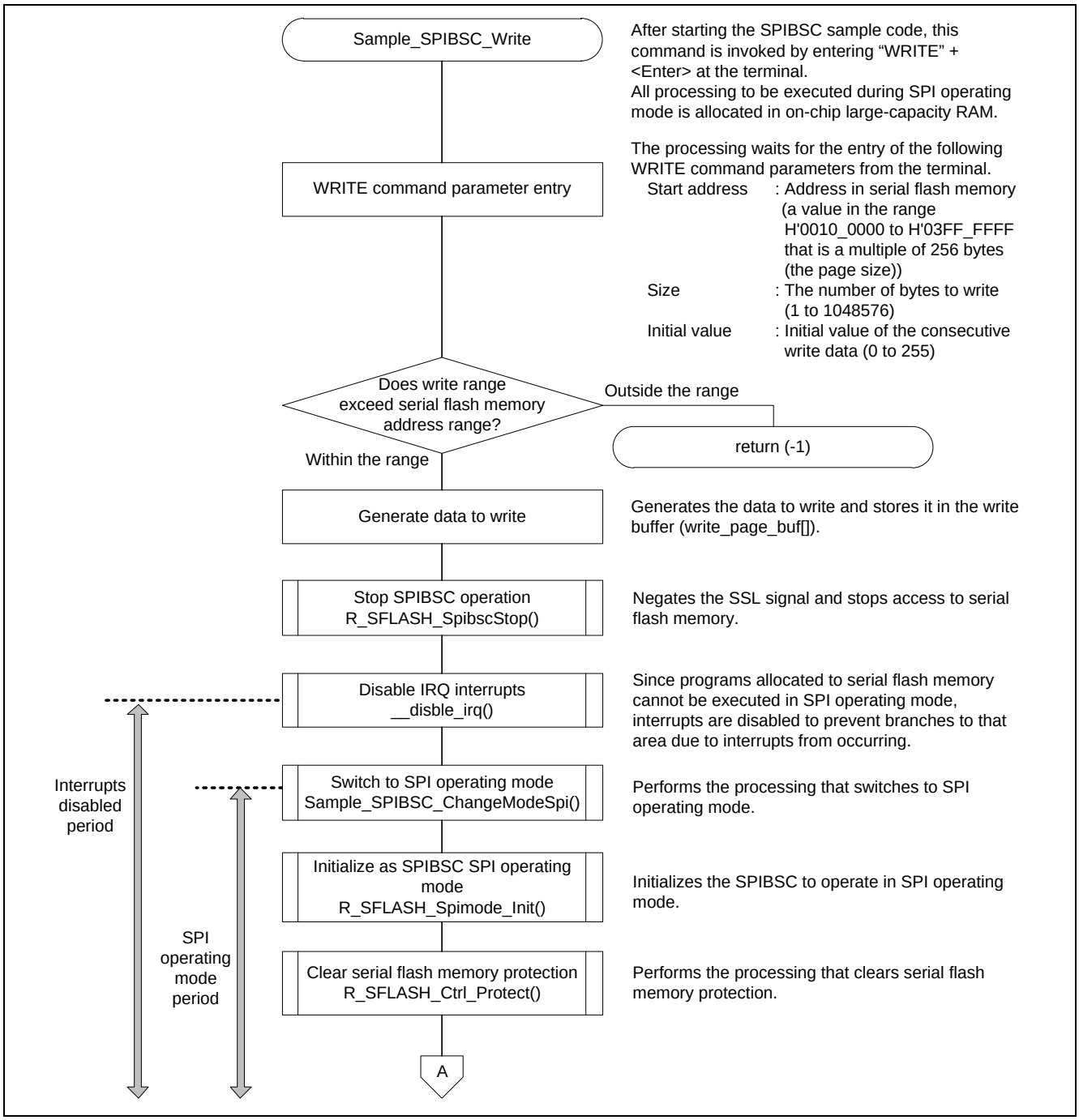


Figure 6.22 WRITE Command Processing (1/2)

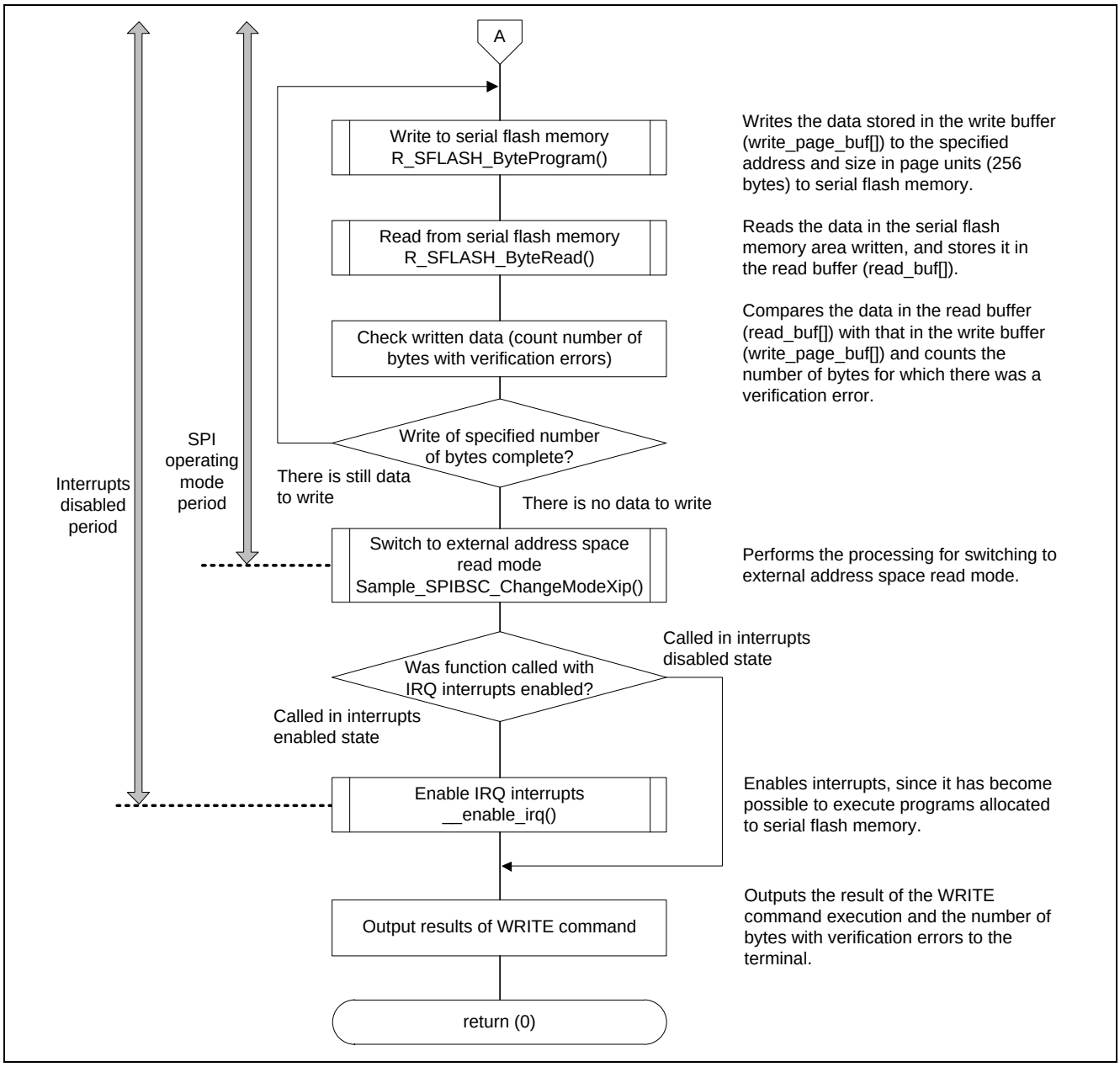


Figure 6.23 WRITE Command Processing (2/2)

6.14.7 SREAD Command Processing

Figure 6.24 and Figure 6.25 show the SREAD Command Processing.

The SREAD command reads data from serial flash memory by calling the Sample_SPIBSC_ChangeModeSpi() function to change the SPIBSC mode to SPI operating mode and then calling the R_SFLASH_ByteRead() function to read data from the serial flash memory. After the read processing completes, this command calls the Sample_SPIBSC_ChangeModeXip() function to perform the processing to change the SPIBSC mode to external address space read mode to set the system to be able to read instructions and data allocated to the SPI multi-I/O bus space (serial flash memory).

In this read processing that uses SPI operating mode, as in erase processing, the command processing is allocated to and executed from on-chip large-capacity RAM, and IRQ interrupts are disabled during the read processing.

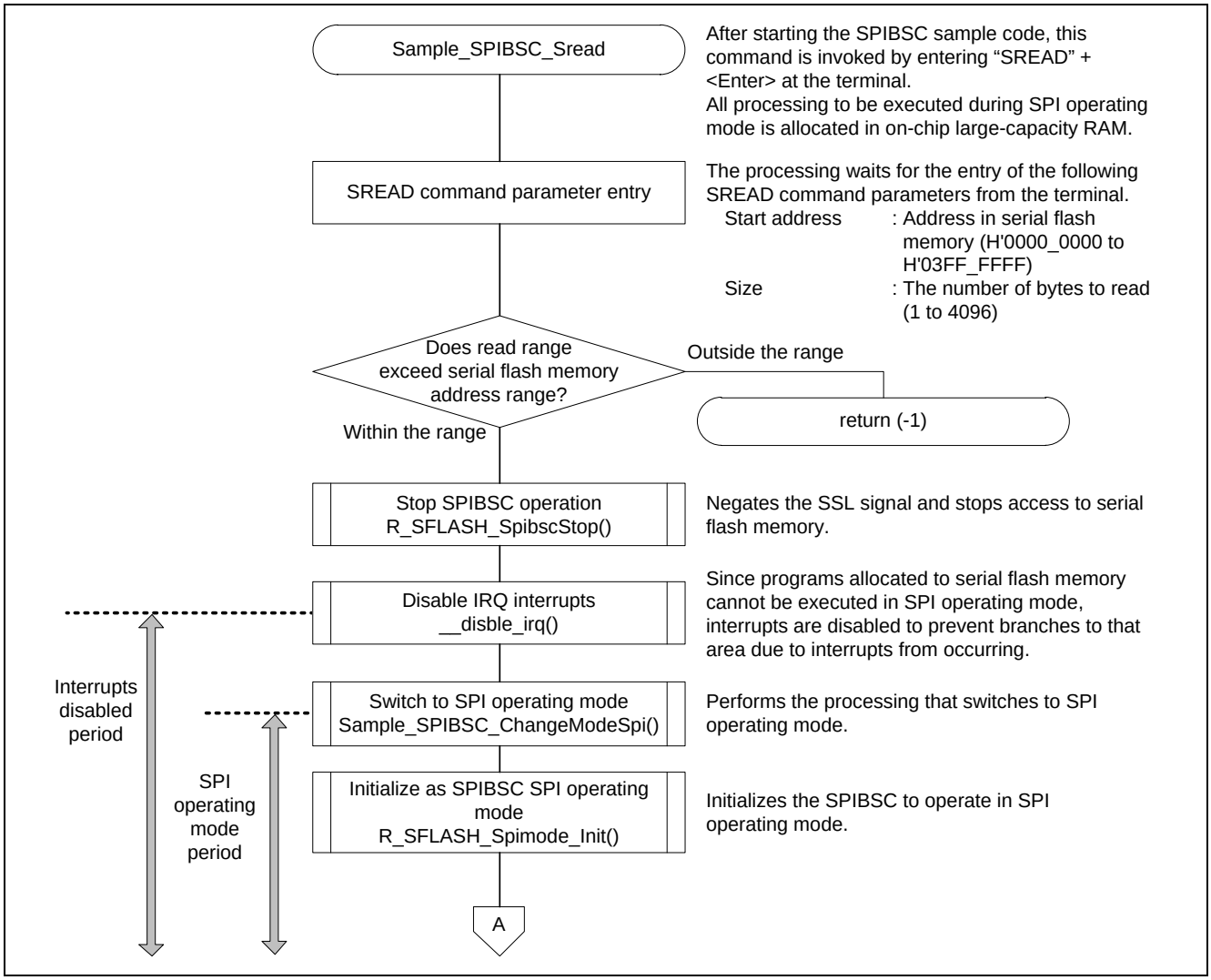


Figure 6.24 SREAD Command Processing (1/2)

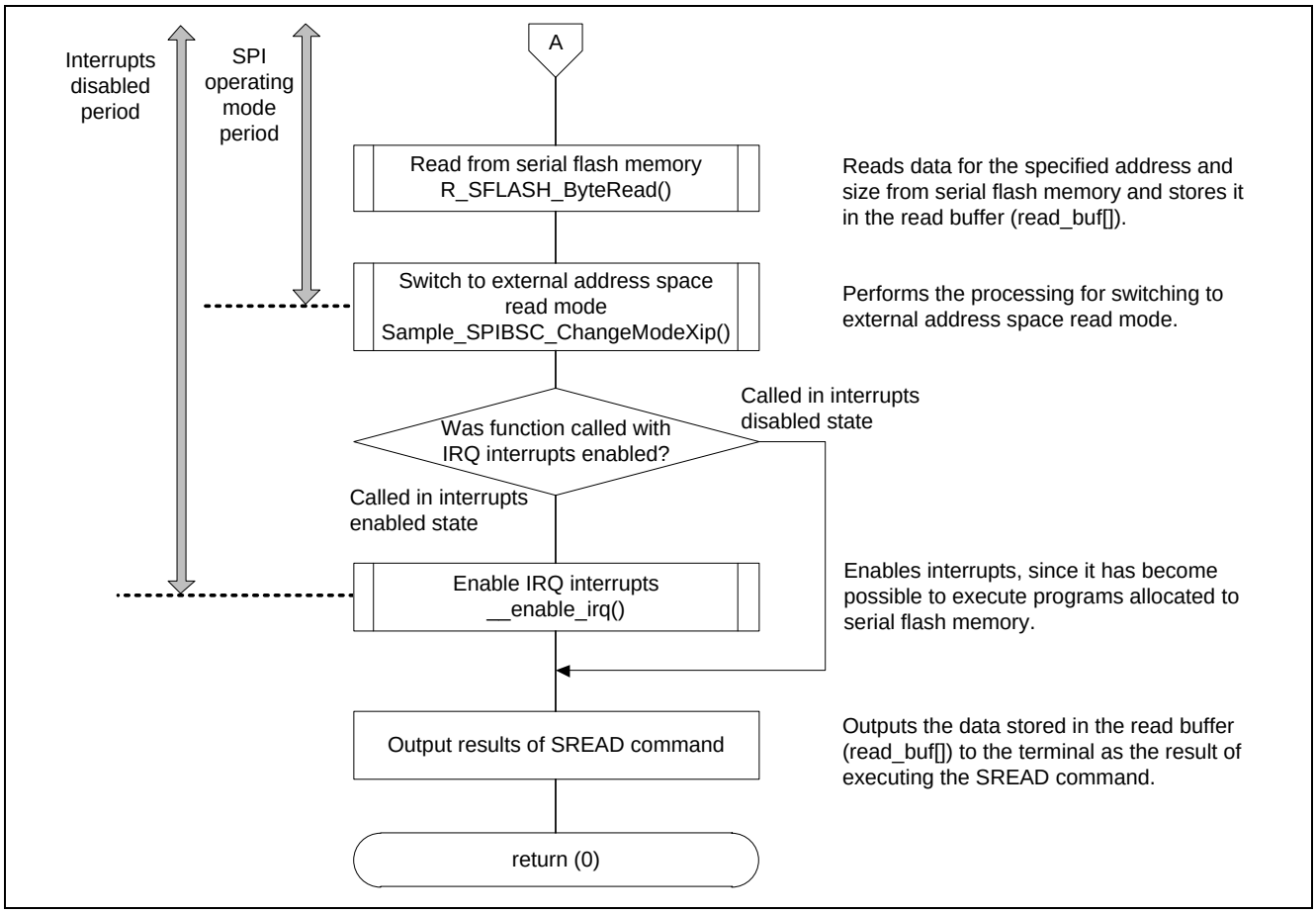


Figure 6.25 SREAD Command Processing (2/2)

6.14.8 Modification of the MMU Translation Table Used in SPI Operating Mode

Figure 6.26 shows the Modification of the MMU Translation Table Used in SPI Operating Mode. This processing modifies the AP[2:0] and XN bits for the MMU translation table for the SPI multi-I/O bus space (locations H'1800_0000 to H'1BFF_FFFF) so that access to the SPI multi-I/O bus space is not possible while in SPI operating mode.

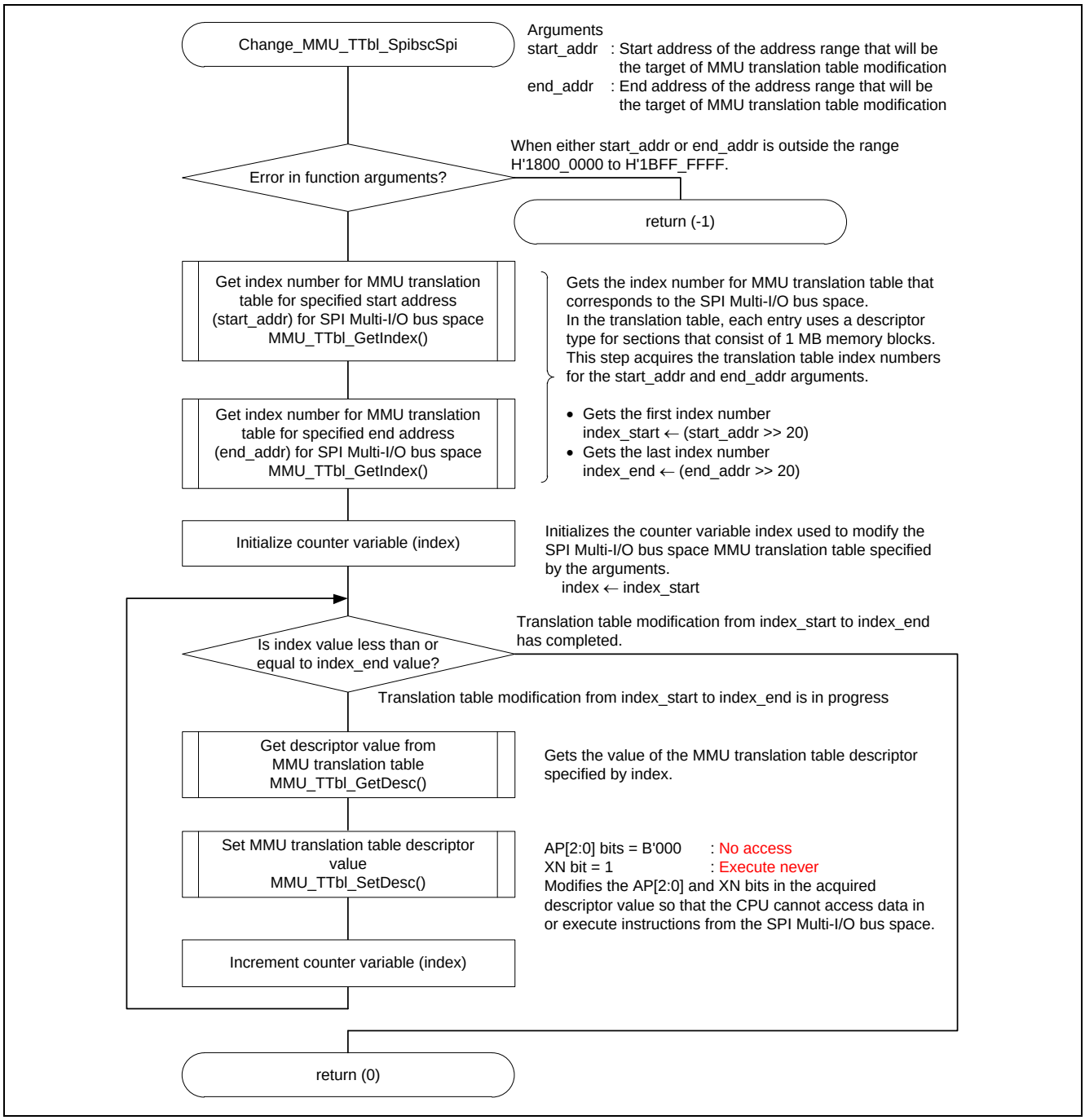


Figure 6.26 Modification of the MMU Translation Table Used in SPI Operating Mode

6.14.9 Modification of the MMU Translation Table Used in External Address Space Read Mode

Figure 6.27 shows the Modification of the MMU Translation Table Used in External Address Space Read Mode. This processing modifies the AP[2:0] and XN bits for the MMU translation table for the SPI multi-I/O bus space (locations H'1800_0000 to H'1BFF_FFFF) so that access to the SPI multi-I/O bus space is possible while in external address space read mode.

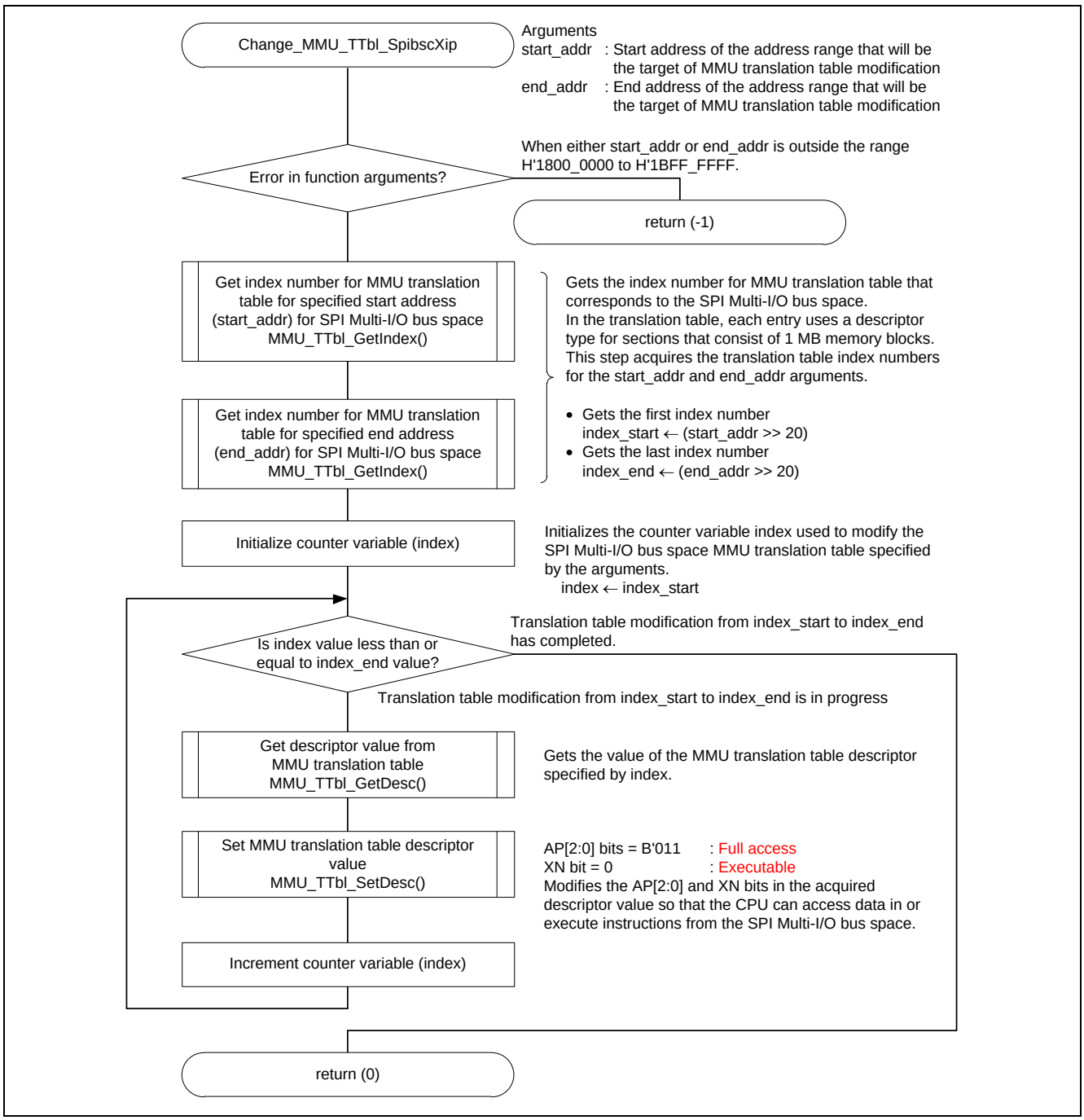


Figure 6.27 Modification of the MMU Translation Table Used in External Address Space Read Mode

6.14.10 Serial Flash Memory Write Enable

To write to the serial flash memory registers (the status register and configuration register), it is necessary to write enable the serial flash memory in advance.

The user must implement the Userdef_SFLASH_Write_Enable() function according to the specifications of the serial flash memory actually used to write enable that serial flash memory.

In the sample code, the serial flash memory command issuing function (R_RFLASH_Spibsc_Transfer()) is used to issue a write enable command (H'06) to perform the processing that changes the serial flash memory state to write enabled (that is, that sets the status register WEL bit to 1).

Figure 6.28 shows the Userdef_SFLASH_Write_Enable() Function Flowchart.

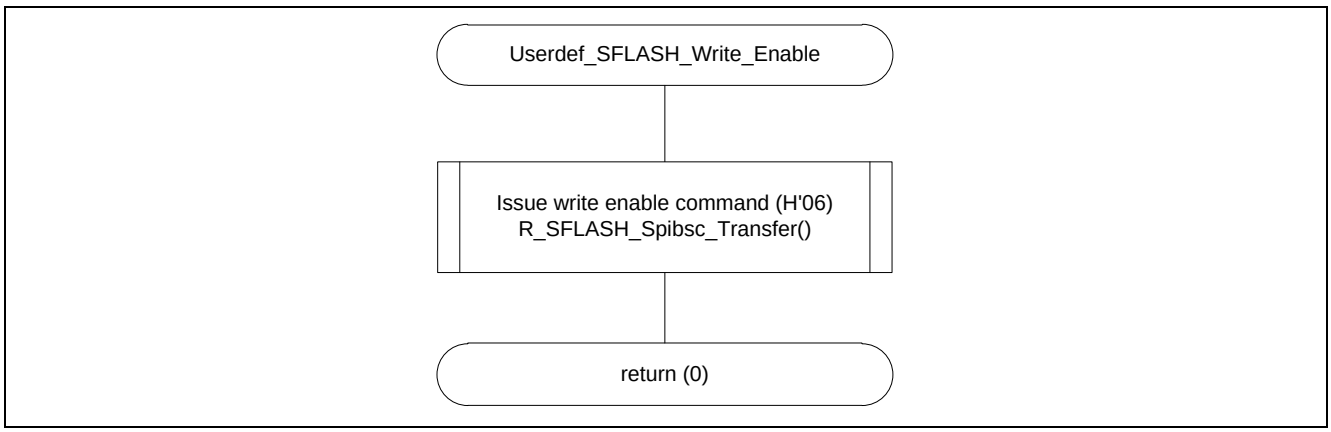


Figure 6.28 Userdef_SFLASH_Write_Enable() Function Flowchart

6.14.11 Serial Flash Memory Wait for Write Completion

When a write operation is performed to a serial flash memory register (the status register or configuration register), the serial flash memory transitions to the busy state. Applications must wait after the busy state until the written data values are reflected in the registers.

The user must implement the Userdef_SFLASH_Busy_Wait() function according to the specifications of the serial flash memory actually used so that it waits until the write operation to the serial flash memory has completed.

This sample code reads the WIP bit in the status register to implement the wait until the write operation has completed.

Figure 6.29 shows the Userdef_SFLASH_Busy_Wait() Function Flowchart.

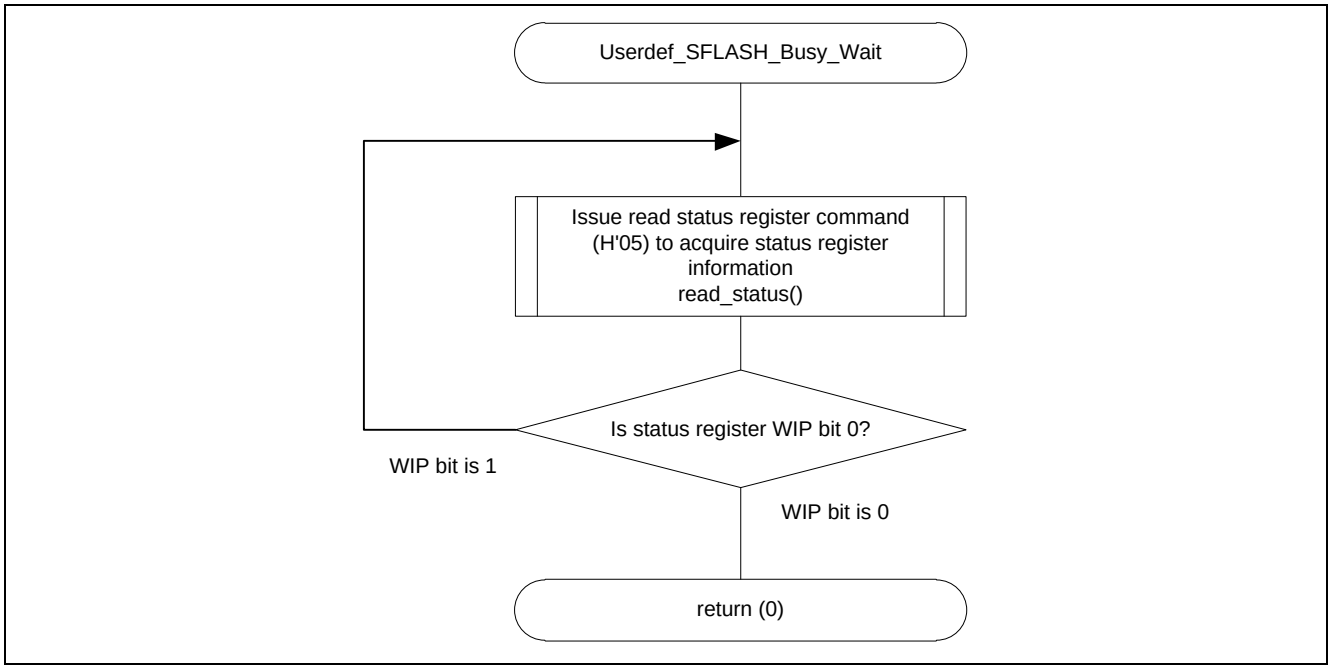


Figure 6.29 Userdef_SFLASH_Busy_Wait() Function Flowchart

6.14.12 Clear Serial Flash Memory Protection

To erase or write serial flash memory, the protection state must be cleared.

The user must implement the Userdef_SFLASH_Ctrl_Protect() function according to the specifications of the serial flash memory actually used to clear the serial flash memory write protection.

This sample code uses the serial flash memory command issuing function (R_SLFASH_Spibsc_Transfer()) which is called from the write_status() function, to set the status register block protect bits (BP3, BP2, BP1, and BP0) to 0 and clear the write protection for all blocks.

Figure 6.30 shows the Userdef_SFLASH_Ctrl_Protect() Function Flowchart.

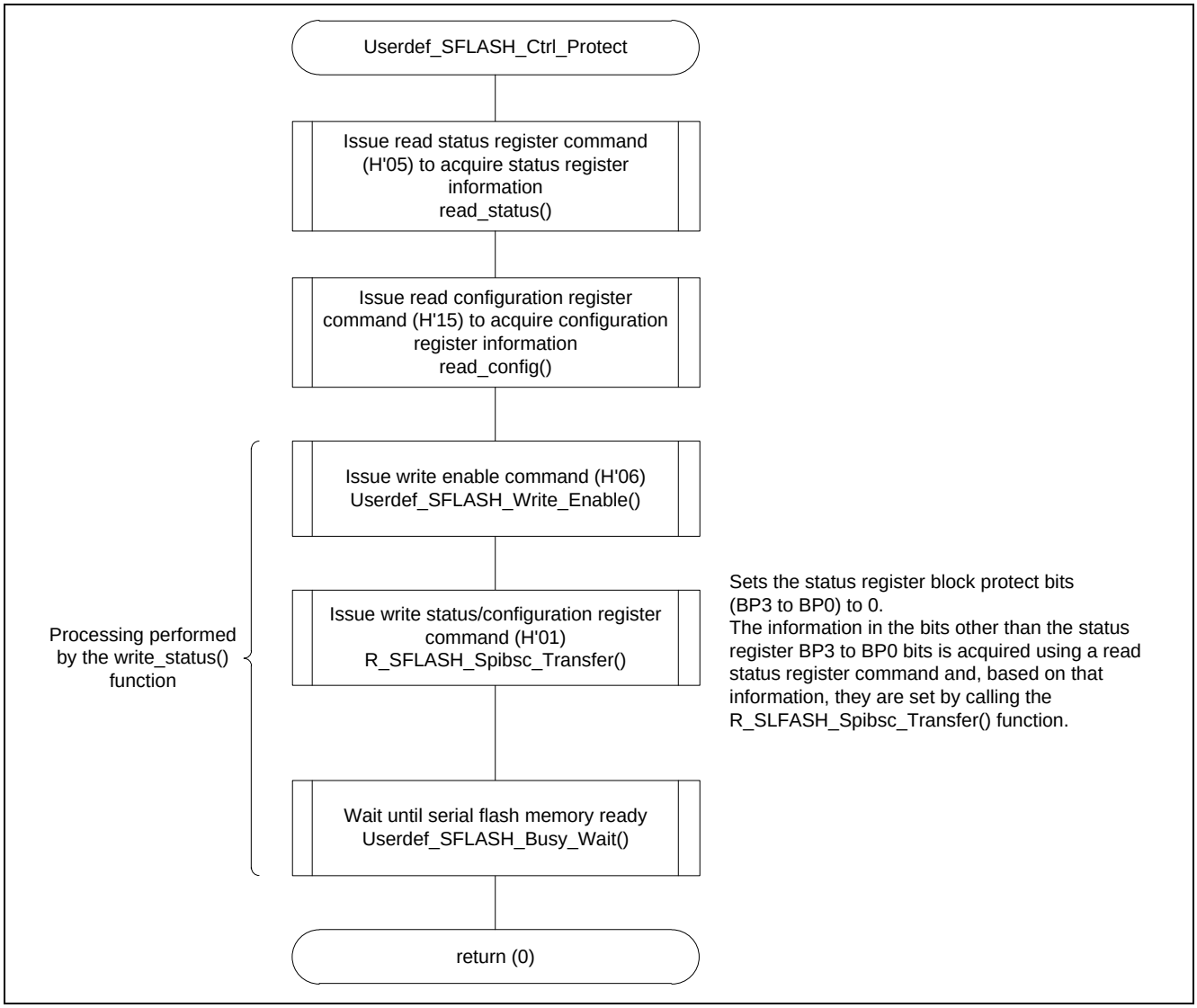


Figure 6.30 Userdef_SFLASH_Ctrl_Protect() Function Flowchart

7. Using this Sample Code

7.1 Starting the Sample Code

This sample code operates by entering commands to a terminal program on a host PC connected to the JASMINE board by a serial interface.

After power is applied to the JASMINE board, the message shown in Figure 7.1[1] is output. When the SPIBSC sample code starts, after the "SAMPLE>" prompt is displayed, enter "SPIBSC" + <Enter>. Then the message shown in Figure 7.1[2] is output. The SPIBSC sample code is operated by entering the following commands after the "SPIBSC>" prompt.

1. When "XREAD" + <Enter> is input, the sample code that performs a dump operation on the SPI multi-I/O bus space is started.
2. When "ERASE" + <Enter> is input, the sample code that performs erase processing on the serial flash memory is started.
3. When "WRITE" + <Enter> is input, the sample code that performs write processing on the serial flash memory is started.
4. When "SREAD" + <Enter> is input, the sample code that performs read processing on the serial flash memory is started.

Display messages

```
RZ/A1LU AVB Board S-Flash Boot Sample Program. Ver.X.XX
Copyright (C) 2017 Renesas Electronics Corporation. All rights reserved.   [1]

select sample program.

SAMPLE>
```

```
RZ/A1LU SPIBSC Sample Program. Ver.Y.YY
Copyright (C) 2017 Renesas Electronics Corporation. All rights reserved.   [2]

select sample program.

SPIBSC >
```

```
SPIBSC > help

  XREAD: Read data from the serial flash memory                            [3]
         (by the external address space read mode)
  ERASE: Erase sector in the serial flash memory
  WRITE: Write data to the serial flash memory
  SREAD: Read data from the serial flash memory
         (by the SPI operating mode)
  EXIT:  Exit from SPIBSC Sample Program

SPIBSC >
```

Figure 7.1 Examples of Terminal Display when the SPIBSC Sample Code Runs

When "HELP" + <Enter> is input, the sample code information shown in Figure 7.1[3] is displayed. When "EXIT" + <Enter> is input, the SPIBSC sample code operation terminates.

The "Ver.X.XX" shown in Figure 7.1 indicates the version number of the main processing in this sample code, and the "Ver.Y.YY" is the SPIBSC sample code version number.

8. Application Example

8.1 Modifying the Sample Code when the Serial Flash Memory Used is Changed

If the serial flash memory is changed, the sample code must be modified according to the specifications of the serial flash memory to be used. This section describes the changes to the sample code required to match the command specifications of a different serial flash memory device.

8.1.1 Changes to the R_SFLASH_EraseSector() Sector Erase Function

Change the contents of the R_SFLASH_EraseSector() function for the items listed in Table 8.1 to match the sector erase command specifications of the serial flash memory used.

Table 8.1 Required Changes to the R_SFLASH_EraseSector() Function

Item	Description
Command	g_spibsc_spimd_reg.cmd = Erase command
Address length transferred	• For a 24-bit address length g_spibsc_spimd_reg.ade = SPIBSC_OUTPUT_24 • For a 32-bit address length g_spibsc_spimd_reg.ade = SPIBSC_OUTPUT_32
Bus width during address transfer	• For a 1-bit bus width g_spibsc_spimd_reg.adb = SPIBSC_1BIT • For a 4-bit bus width g_spibsc_spimd_reg.adb = SPIBSC_4BIT
Data transfer method during address transfer	• For SDR transfer g_spibsc_spimd_reg.addre = SPIBSC_SDR_TRANS • For DDR transfer g_spibsc_spimd_reg.addre = SPIBSC_DDR_TRANS
Sector size (block size)	Change the SF_SECTOR_SIZE macro definition to match the sector size of the serial flash memory used. (Defined in sflash.h)
Total sector count	Change the SF_NUM_OF_SECTOR macro definition to match the total number of sectors in the serial flash memory used. (Defined in sflash.h)

8.1.2 Changes to the R_SFLASH_ByteProgram() Write Function

Change the contents of the R_SFLASH_ByteProgram() function for the items listed in Table 8.2 to match the page program (write) command specifications of the serial flash memory used.

Table 8.2 Required Changes to the R_SFLASH_ByteProgram() Function

Item	Description
Command	g_spibsc_spimd_reg.cmd = Page program command
Address length transferred	• For a 24-bit address length g_spibsc_spimd_reg.ade = SPIBSC_OUTPUT_24 • For a 32-bit address length g_spibsc_spimd_reg.ade = SPIBSC_OUTPUT_32
Bus width during address transfer	• For a 1-bit bus width g_spibsc_spimd_reg.adb = SPIBSC_1BIT • For a 4-bit bus width g_spibsc_spimd_reg.adb = SPIBSC_4BIT
Data transfer method during address transfer	• For SDR transfer g_spibsc_spimd_reg.addre = SPIBSC_SDR_TRANS • For DDR transfer g_spibsc_spimd_reg.addre = SPIBSC_DDR_TRANS
Bus width during data transfer	• For a 1-bit bus width g_spibsc_spimd_reg.spidb = SPIBSC_1BIT • For a 4-bit bus width g_spibsc_spimd_reg.spidb = SPIBSC_4BIT
Data transfer method during data transfer	• For SDR transfer g_spibsc_spimd_reg.spidre= SPIBSC_SDR_TRANS • For DDR transfer g_spibsc_spimd_reg.spidre= SPIBSC_DDR_TRANS
Page size	Change the SF_PAGE_SIZE macro definition to match the page size of the serial flash memory used. (Defined in sflash.h)

8.1.3 Changes to the R_SFLASH_ByteRead() Read Function

Change the contents of the R_SFLASH_ByteRead() function for the items listed in Table 8.3 and Table 8.4 to match the read command specifications of the serial flash memory used.

Table 8.3 Required Changes to the R_SFLASH_ByteRead() Function (1/2)

Item	Description
Comamnd	g_spibsc_spimd_reg.cmd = Read command
Address length transferred	• For a 24-bit address length g_spibsc_spimd_reg.ade = SPIBSC_OUTPUT_24 • For a 32-bit address length g_spibsc_spimd_reg.ade = SPIBSC_OUTPUT_32
Bus width during address transfer	• For a 1-bit bus width g_spibsc_spimd_reg.adb = SPIBSC_1BIT • For a 4-bit bus width g_spibsc_spimd_reg.adb = SPIBSC_4BIT
Data transfer method during address transfer	• For SDR transfer g_spibsc_spimd_reg.addre = SPIBSC_SDR_TRANS • For DDR transfer g_spibsc_spimd_reg.addre = SPIBSC_DDR_TRANS
Whether or not option data is output	• When option data is not output g_spibsc_spimd_reg.opde = SPIBSC_OUTPUT_DISABLE • When the OPD3 data is output g_spibsc_spimd_reg.opde = SPIBSC_OUTPUT_OPD_3 • When the OPD3 and OPD2 data is output g_spibsc_spimd_reg.opde = SPIBSC_OUTPUT_OPD_32 • When the OPD3, OPD2, and OPD1 data is output g_spibsc_spimd_reg.opde = SPIBSC_OUTPUT_OPD_321 • When the OPD3, OPD2, OPD1, and OPD0 data is output g_spibsc_spimd_reg.opde = SPIBSC_OUTPUT_OPD_3210
Bus width used when option data is output	• For SDR transfer g_spibsc_spimd_reg.opdb = SPIBSC_1BIT • For DDR transfer g_spibsc_spimd_reg.opdb = SPIBSC_4BIT
Data transfer method used when option data is output	• For SDR transfer g_spibsc_spimd_reg.opdre = SPIBSC_SDR_TRANS • For DDR transfer g_spibsc_spimd_reg.opdre = SPIBSC_DDR_TRANS

Table 8.4 Required Changes to the R_SFLASH_ByteRead() Function (2/2)

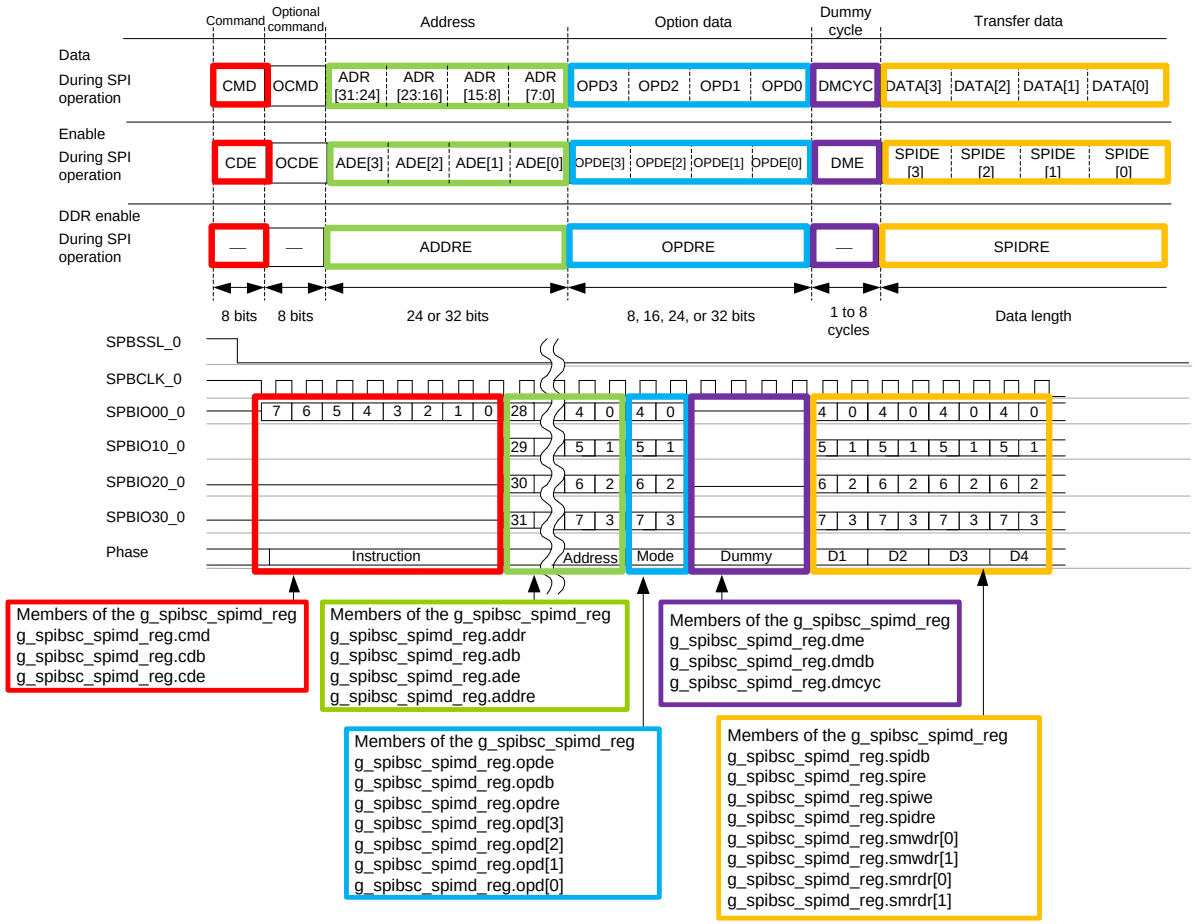
Item	Description
Option data specification	• OPD3 output value g_spibsc_spimd_reg.opd[0] = Output value • OPD2 output value g_spibsc_spimd_reg.opd[1] = Output value • OPD1 output value g_spibsc_spimd_reg.opd[2] = Output value • OPD0 output value g_spibsc_spimd_reg.opd[3] = Output value
Whether or not dummy cycles are output	• When dummy cycle is not output g_spibsc_spimd_reg.dme = SPIBSC_DUMMY_CYC_DISABLE • When dummy cycle is output g_spibsc_spimd_reg.dme = SPIBSC_DUMMY_CYC_ENABLE
Bus width when dummy cycles are output	• For a 1-bit bus width g_spibsc_spimd_reg.dmdb = SPIBSC_1BIT • For a 4-bit bus width g_spibsc_spimd_reg.dmdb = SPIBSC_4BIT
Number of cycles when dummy cycles are output	g_spibsc_spimd_reg.dmcyc = SPIBSC_DUMMY_nCYC*[1]
Bus width during data reception	• For a 1-bit bus width g_spibsc_spimd_reg.spidb = SPIBSC_1BIT • For a 4-bit bus width g_spibsc_spimd_reg.spidb = SPIBSC_4BIT
Transfer method during data reception	• For SDR transfer g_spibsc_spimd_reg.spidre = SPIBSC_SDR_TRANS • For DDR transfer g_spibsc_spimd_reg.spidre = SPIBSC_DDR_TRANS

Note 1. The "n" indicates the number (1 to 8) of dummy cycles.

8.2 Output Signals During Command Issue to Serial Flash Memory

In SPI operating mode, when issuing commands, signals to match the specifications of the serial flash memory used can be output for the read, erase, write, and other commands by setting the SPIBC related registers.

In the sample code, it is possible to change the signals output to the serial flash memory by changing the values set in the SPIBSC registers, that is, the values set in the members of the structures used in each of the sector erase, page program, and read functions shown in Table 8.1 to Table 8.4. Figure 8.1 shows the Relationship Between SPIBSC Register Settings and Waveforms Output to the Serial Flash Memory. The values of the members of the structures used in each function should be set to match the commands for the serial flash memory used.



When a program command (PP4B) is issued to the Macronix MX25L51245G serial flash memory the members of the g_spibsc_spimd_reg structure in the R_SFLASH_ByteProgram() function are set as shown below and the command issue operation can be implemented by calling the R_SFLASH_Spibsc_Transfer() function.

Command	g_spibsc_spimd_reg.cmd	(Command)	: H'12 (PP4B)
	g_spibsc_spimd_reg.cdb	(Bit width during command transfer)	: SPIBSC_1BIT
	g_spibsc_spimd_reg.cde	(Command output enable/disable)	: SPIBSC_OUTPUT_ENABLE
Optional command	g_spibsc_spimd_reg.ocmd	(Option command)	: H'00
	g_spibsc_spimd_reg.ocdb	(Bit width when option command bits are used)	: SPIBSC_1BIT
	g_spibsc_spimd_reg.ocde	(Option command output enable/disable)	: SPIBSC_OUTPUT_DISABLE
Address	g_spibsc_spimd_reg.addr	(Address)	: Serial flash memory address
	g_spibsc_spimd_reg.adb	(Bit width during address transfer)	: SPIBSC_4BIT
	g_spibsc_spimd_reg.ade	(Address length to transfer)	: SPIBSC_OUTPUT_ADDR_32
	g_spibsc_spimd_reg.addre	(Address transfer format)	: SPIBSC_SDR_TRANS
Option data	g_spibsc_spimd_reg.opde	(Option data output enable/disable)	: SPIBSC_OUTPUT_DISABLE
	g_spibsc_spimd_reg.opdb	(Bit width during option data transfer)	: SPIBSC_1BIT
	g_spibsc_spimd_reg.opdre	(Option data transfer format)	: SPIBSC_SDR_TRANS
	g_spibsc_spimd_reg.opd[3]	(Option data OPD0)	: H'00
	g_spibsc_spimd_reg.opd[2]	(Option data OPD1)	: H'00
	g_spibsc_spimd_reg.opd[1]	(Option data OPD2)	: H'00
	g_spibsc_spimd_reg.opd[0]	(Option data OPD3)	: H'00
Dummy cycle	g_spibsc_spimd_reg.dme	(Dummy cycle output enable/disable)	: SPIBSC_DUMMY_CYC_DISABLE
	g_spibsc_spimd_reg.dmdb	(Bit width during dummy cycle transfer)	: SPIBSC_1BIT
	g_spibsc_spimd_reg.dmcyc	(Number of dummy cycles)	: SPIBSC_DUMMY_1CYC
Data	g_spibsc_spimd_reg.spidb	(Transfer data bit width)	: SPIBSC_4BIT
	g_spibsc_spimd_reg.spire	(Data read enable)	: SPIBSC_SPIDATA_DISABLE (Read disable)
	g_spibsc_spimd_reg.spiwe	(Data write enable)	: SPIBSC_SPIDATA_ENABLE (Write enable)
	g_spibsc_spimd_reg.spidre	(Transfer data transfer format)	: SPIBSC_SDR_TRANS
	g_spibsc_spimd_reg.smwdr[0]	(Data written to SMWDR0)	: Write data
	g_spibsc_spimd_reg.smwdr[1]	(Data written to SMWDR1)	: Unused when only 1 serial flash memory device is connected.
SSL settings	g_spibsc_spimd_reg.smrdr[0]	(Data read from SMRDR0)	: Read data
	g_spibsc_spimd_reg.smrdr[1]	(Data read from SMRDR1)	: Unused when only 1 serial flash memory device is connected.
	g_spibsc_spimd_reg.sslkp	(SSL signal keep setting)	: SPIBSC_SPISSL_NEGATE

Figure 8.1 Relationship Between SPIBSC Register Settings and Waveforms Output to the Serial Flash Memory

9. Notes

9.1 Regarding Interrupts that Occur During Sample Command Execution

Since IRQ interrupt are disabled during sample command execution, no interrupt handling is performed during sample command execution. After sample command execution completes and interrupts are enabled, any pending interrupt handling is performed.

When it is desirable to perform interrupt handling during sample command execution, it is possible to perform interrupt handling even during sample command execution by enabling IRQ interrupts. However, it is possible for the SPIBSC to be set to SPI operating mode during sample command execution. Instructions allocated to the SPI multi-I/O bus space (serial flash memory) cannot be executed in SPI operating mode. Therefore, to execute interrupt handling or exception handling during sample command execution (during SPI operating mode), all IRQ interrupt handling, including the interrupt vector, must be allocated to on-chip large-capacity RAM.

Note that if the FIQ interrupts are set to enabled, it is not possible to disable them. Therefore, if it is possible that a FIQ interrupt could occur, all FIQ interrupt handling, including the FIQ exception handling vector, must be allocated to on-chip large-capacity RAM.

10. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

11. Reference Documents

User's Manual: Hardware

RZ/A1L Group, RZ/A1LU Group, RZ/A1LC Group User's Manual: Hardware
The latest version can be downloaded from the Renesas Electronics website.

RZ/A1LU AVB board RTK772103FC00000BR (JASMINE) User's Manual
The latest version can be downloaded from the Renesas Electronics website.

ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition Issue C
The latest version can be downloaded from the ARM website.

ARM Generic Interrupt Controller Architecture Specification Architecture version 1.0
The latest version can be downloaded from the ARM website.

ARM CortexTM-A9 (Revision: r3p0) Technical Reference Manual
The latest version can be downloaded from the ARM website.

ARM CoreLinkTM Level 2 Cache Controller L2C-310 (Revision: r3p2) Technical Reference Manual
The latest version can be downloaded from the ARM website.

Technical Update/Technical News

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

User's Manual: Development Tools

ARM Software Development Tools (ARM Compiler toolchain, ARM DS-5 etc) can be downloaded from the ARM website.
The latest version can be downloaded from the ARM website.

Website and Support

Renesas Electronics Website
http://www.renesas.com/

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

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
Rev.1.00	May. 17, 2017	—	First edition issued

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

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

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

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

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other disputes involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawing, chart, program, algorithm, application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics products.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   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 (space and undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. When using the Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat radiation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions or failure or accident 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 Renesas Electronics 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 ensure to implement safety measures to guard them against the possibility of bodily injury, injury or damage caused by fire, and social damage in the event of failure or malfunction of Renesas Electronics products, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures by your own responsibility as warranty for your products/system. Because the evaluation of microcomputer software alone is very difficult and not practical, 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 investigate applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive carefully and sufficiently and use Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall not use Renesas Electronics products or technologies for (1) any purpose relating to the development, design, manufacture, use, stockpiling, etc., of weapons of mass destruction, such as nuclear weapons, chemical weapons, or biological weapons, or missiles (including unmanned aerial vehicles (UAVs)) for delivering such weapons, (2) any purpose relating to the development, design, manufacture, or use of conventional weapons, or (3) any other purpose of disturbing international peace and security, and you shall not sell, export, lease, transfer, or release Renesas Electronics products or technologies to any third party whether directly or indirectly with knowledge or reason to know that the third party or any other party will engage in the activities described above. When exporting, selling, transferring, etc., Renesas Electronics products or technologies, you shall comply with any applicable export control laws and regulations promulgated and administered by the governments of the countries asserting jurisdiction over the parties or transactions.
10. Please acknowledge and agree that you shall bear all the losses and damages which are incurred from the misuse or violation of the terms and conditions described in this document, including this notice, and hold Renesas Electronics harmless, if such misuse or violation results from your resale or making Renesas Electronics products available any third party.
11. This document shall not be reprinted, 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.

(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.

(Rev.3.0-1 November 2016)

RENESAS

Renesas Electronics Corporation

http://www.renesas.com

SALES OFFICES

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

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

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

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

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

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

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

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

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

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

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

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

Colophon 6.0