RZ/A2M Group

Blinky Sample Program FreeRTOS Application Note

Introduction

This application note covers the operation of the Blinky sample application running under FreeRTOS on the RZ/A2M CPU board.

It provides a comprehensive overview of the software. For further details please refer to the software itself.

The user is assumed to be equipped with an RZ/A2M CPU board.

Target Device

RZ/A2M

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

Contents

List of Abbreviations and Acronyms

Abbreviation	Full Form
API	Application Programming Interface
ARM	Advanced RISC Machines
BSS	Block Started by Symbol
BTAC	Branch Target Address Cache
CPG	Clock Pulse Generator
CPU	Central Processing Unit
DACR	Domain Access Control Register
DMAC	Direct Memory Access Controller
FIFO	First In First Out
FIQ	Fast Interrupt
FPU	Floating Point Unit
GPIO	General Purpose Input Output
INTC	INTerrupt Controller
I/O	Input/Output
IRQ	Interrupt ReQuest
L1	Level 1 (primary cache)
L2	Level 2 (secondary cache)
LED	Light Emitting Diode
LSB	Least Significant Bit
MCU	MicroController Unit
MMU	Memory Management Unit
NMI	Non-Maskable Interrupt
OS	Operating System
OSTM	Operating System Timer Module
PC	Personal Computer
PLL	Phase-Locked Loop
RAM	Random-Access Memory
ROM	Read Only Memory
RTC	Real Time Clock
SC	Smart Configurator
SCIFA	Serial Communications Interface with FIFO
SCTLR	System Control Register
SDRAM	Synchronous Dynamic Random-Access Memory
SPI	Serial Peripheral Interface
SPIBSC	SPI Multi I/O Bus Controller
STB	STandBy module
STDIO	Standard Input/Output

TLB	Translation Lookaside Buffer
TTBCR	Translation Table Base Control Register
TTBR	Translation Table Base Register
USB	Universal Serial Bus
VBAR	Vector Base Address Register
XTAL	Crystal

Table 1-1 List of Abbreviations and Acronyms

1. Specifications

After a device reset, the following devices are initialized by the program located in the serial flash memory assigned in SPI Multi I/O bus space.

- SPI Multi I/O Bus Controller
- Clock Pulse Generator
- Interrupt Controller
- Bus State Controller
- OS Timer
- General I/O Ports
- Memory Management Unit
- Primary Cache
- Secondary Cache

In this application note, the SPI Multi I/O Bus Controller, Clock Pulse Generator, Interrupt Controller, OS Timer, Serial Communication Interface with FIFO, General I/O Ports, Power-down Mode and Memory Management Unit are referred to as SPIBSC, CPG, INTC, OSTM, SCIFA, STB and MMU, respectively.

Table 1.1 lists the peripheral functions supported in this application note. Figure 1.1 shows the operation overview.

Table 1.1 Peripheral devices used

Peripheral device	Usage
SPI Multi I/O Bus Controller (SPIBSC)	Configure SPIBSC as external address space read mode and generate the signal for CPU to read data from serial flash memory assigned in SPI Multi I/O bus space directly
Clock Pulse Generator (CPG)	Generate the operating frequency of RZ/A2M
Interrupt Controller (INTC)	Control the interrupts used by OSTM channel 0 and SCIFA channel 4
OS Timer (OSTM)	Manage the timer for FreeRTOS ticks using OSTM channel 0
Serial Communication Interface with FIFO (SCIFA)	Control the communications between the RZ/A2M and a host PC using SCIFA channel 4
General I/O Ports (GPIO)	Assign the multiplexed pin functions for SCIFA channel 4 Control I/O pins for blinking the LED
Power-down Mode (STB)	Disable the RZ/A2M peripheral I/O module standby Enable writing to the on-chip data retention RAM
Memory Management Unit (MMU), Primary (L1) Cache and Secondary (L2) Cache	Generate MMU translation table for configuring the area where L1 cache is enabled, memory attribute of each region of memory (i.e. Normal, Device, or Strongly-Ordered) Enable L1 and L2 Cache

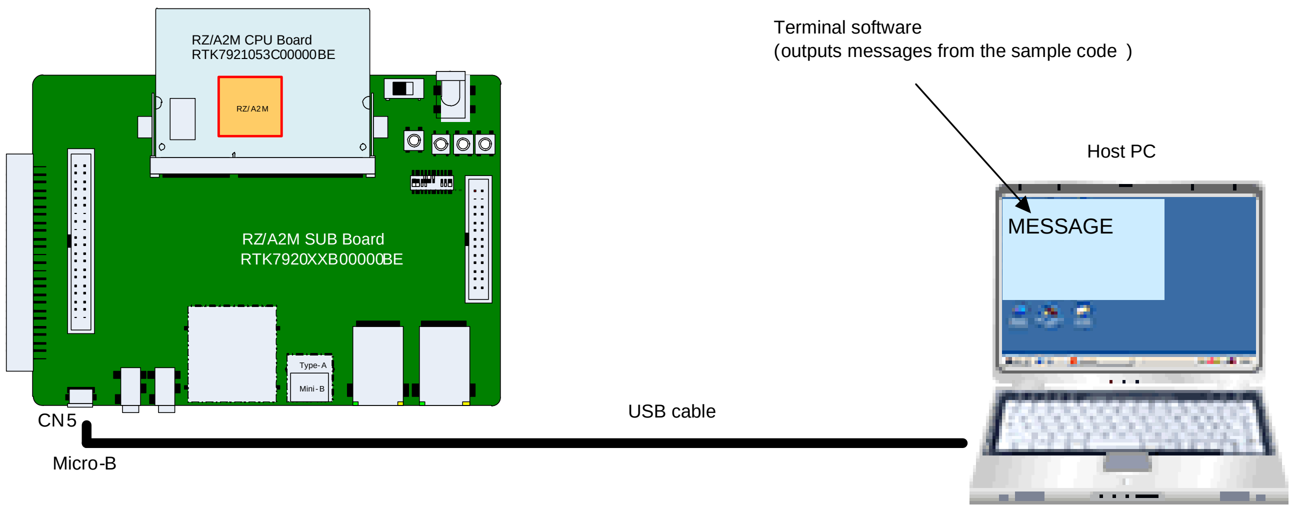


Figure 1.1 Operation check conditions

2. Proven Operating Conditions

The sample program accompanying this application note has been verified and works as expected under the conditions shown in Table 2.1.

Table 2.1 Proven Operating Conditions (1/2)

Item	Description
Microcomputer used	RZ/A2M
Operating frequency (Note)	CPU Clock (Iϕ): 528MHz Image processing clock (Gϕ): 264MHz Internal Bus Clock (Bϕ): 132MHz Peripheral Clock 1 (P1ϕ): 66MHz Peripheral Clock 0 (P0ϕ): 33MHz QSPI0_SPCLK: 66MHz CKIO: 132MHz
Operating voltage	Power supply voltage (I/O): 3.3 V Power supply voltage (either 1.8V or 3.3V I/O (PVcc SPI)): 3.3V Power supply voltage (internal): 1.2 V
Integrated Development Environment	e^2 studio V2020-07
C compiler	"GNU Arm Embedded Tool chain 6-2017-q2-update" compiler options(except directory path) Release: -mcpu=cortex-a9 -march=armv7-a -marm -mthumb-interwork -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -O3 -ffunction-sections -fdata-sections -Wextra -Wmissing-declarations -Wconversion -Wnull-dereference -Wstack-usage=100 -fabi-version=0 Hardware Debug: -mcpu=cortex-a9 -march=armv7-a -marm -mthumb-interwork -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -Og -ffunction-sections -fdata-sections -Wmissing-declarations -Wconversion -Wnull-dereference -g -Wstack-usage=100 -fabi-version=0

Note: The operating frequency used in clock mode 1 (Clock input of 24MHz from EXTAL pin)

Table 2.2 Proven Operating Conditions (2/2)

Item	Description
Operation mode	Boot mode 3 (Serial Flash boot 3.3V)
Terminal software communication settings	• Communication speed: 115200bps • Data length: 8 bits • Parity: None • Stop bits: 1 bit • Flow control: None
Board to be used	RZ/A2M CPU board RTK7921053C00000BE RZ/A2M SUB board RTK79210XXB00000BE
Device (functionality to be used on the board)	• Serial flash memory allocated to SPI multi-I/O bus space - Manufacturer: Macronix Inc. - Model Name: MX25L51245GXD • RL78/G1C (USB-to-serial converter, which is used for communicating with host PC) • LED1

3. Reference Application Notes

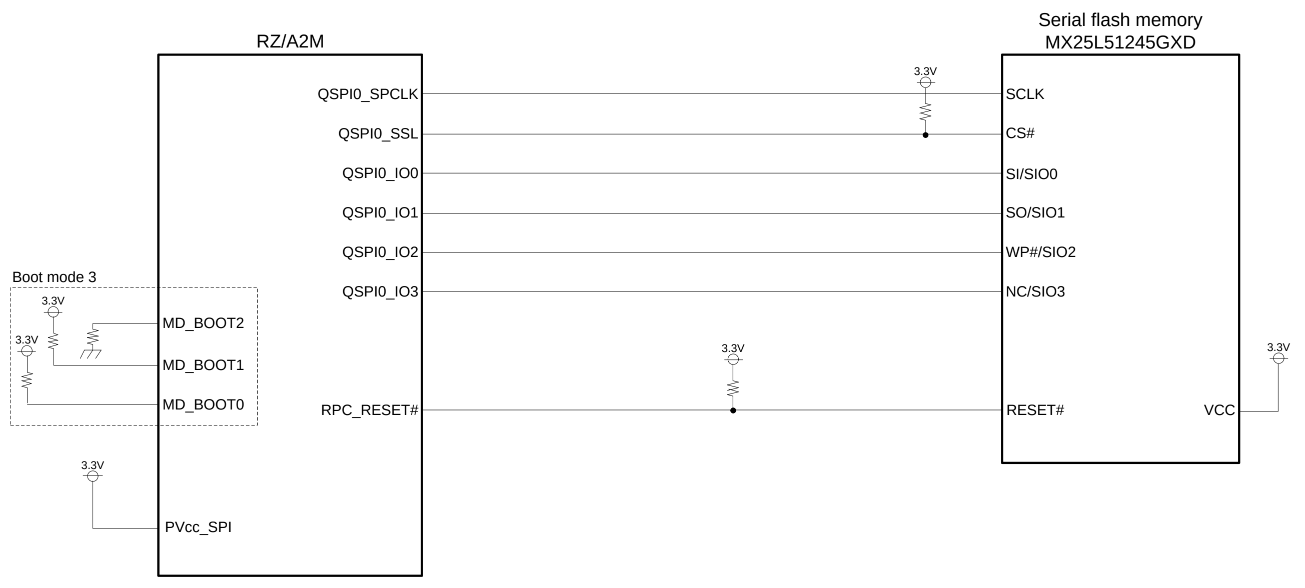
In this chapter, application note referenced in this note are listed:

- RZ/A2M Group Example of booting from serial flash memory (R01AN4333)

4. Hardware

4.1 Hardware Configuration

In the initial setup example described in this application note, boot mode is configured as boot mode 3. Consequently, processing is carried out by the program located in the serial flash memory assigned in the SPI Multi I/O bus space. Figure 4.1 shows the hardware connection diagram when RZ/A2M is booted up from serial flash memory assigned in SPI Multi I/O bus area under boot mode 3.



Note: The "#" symbol indicates negative logic (active low).

Figure 4.1 Connection diagram example under boot mode 3

4.2 Pins used

Table 4.1 lists the pins used in this application note.

Table 4.1 Pins to be used and its function

Pin Name	I/O	Description
MD_BOOT2	Input	Boot mode selection (Select boot mode 3) MD_BOOT2: "L", MD_BOOT1: "H", MD_BOOT0: "H"
MD_BOOT1	Input	
MD_BOOT0	Input	
QSPI0_SCLK	Output	Clock output to serial flash memory
QSPI0_SSL	Output	Slave selection for serial flash memory
QSPI0_IO0	Input/Output	Data 0 pin for serial flash memory
QSPI0_IO1	Input/Output	Data 1 pin for serial flash memory
QSPI0_IO2	Input/Output	Data 2 pin for serial flash memory
QSPI0_IO3	Input/Output	Data 3 pin for serial flash memory
RPC_RESET# (Note)	Output	Reset control signal output to serial flash memory
P6_0	Output	LED blinking
RxD4 (P9_1)	Input	Serial receive data signal
TxD4 (P9_0)	Output	Serial transmit data signal

Note: The "#" symbol denotes negative logic (active low)

5. Software

5.1 Operation Overview

After a device reset, the ROM program for boot-up (known as the boot program) stored in RZ/A2M internal ROM (address: H'FFFF 0000) is invoked. Firstly, the boot program sets up the configuration for accessing serial flash memory under boot mode 3, and then jumps to the address H'2000 0000 which is the starting address of the SPI Multi I/O bus space.

This initial setup program consists of the loader program placed at the address H’2000 0000, and application program placed at the address H’2004 0000.

We now describe how the initial setup is performed in the sample program.

Firstly, the loader program sets up the appropriate configuration for accessing serial flash memory and jumps to the start-up processing implemented in the application program.

In the start-up processing, the following initialization is carried out and execution then jumps to the resetprg() function:

- Initialization of stack pointer
- Cache setup
- MMU setup
- FPU setup
- Initialization of memory section

In the resetprg() function, the L1 and L2 cache are enabled, and initialization of INTC is carried out. Then, an address area in the on-chip large-capacity RAM is assigned for VBAR in order to accelerate interrupt processing. Finally, IRQ and FIQ interrupts are enabled followed by a call to the main() function.

In the main() function, the following initialization is carried out:

- CPG initialization by calling cpg_open() and cpg_close() via direct_open() and direct_close() respectively.
- The FreeRTOS is then started by a call to R_OS_KernelInit().
- This initialises the memory manager, creates a new thread; main_task(), and then starts the FreeRTOS kernel with a call to R_OS_KernelStart().

The main_task() thread calls initialise_monitor_handles() to hook up stdin, stdout, and stderr to SCIFA channel 4, so that strings output on the RZ/A2M CPU board serial interface will appear on the console running on the host PC.

This will result in a call to scifa_hld_open() which will perform the following SCIFA channel 4 related setup:

- Initialization
- Enable the interrupt
- Configure the interrupt priority level
- Register the interrupt handler

The main_task() function then jumps to the function os_main_task_t() which becomes the main application thread, and does not return.

os_main_task_t() does the following:

- Opens the GPIO driver and then calls direct_control() to configure port 6 pin 0 for LED blinking
- Creates a new task using the function os_console_task_t() to handle console input/output.
- Enters an infinite loop which toggles the state of the LED every 500 milliseconds. The 500 millisecond delay is provided by OS abstraction wrapper function R_OS_TaskSleep(500) and this results in the red LED flashing at 1Hz.

For details on the processing implemented in the loader program, please refer to the application note "Example of booting from serial flash memory".

Figure 5.1 shows the sequence diagram until the function main() is invoked.

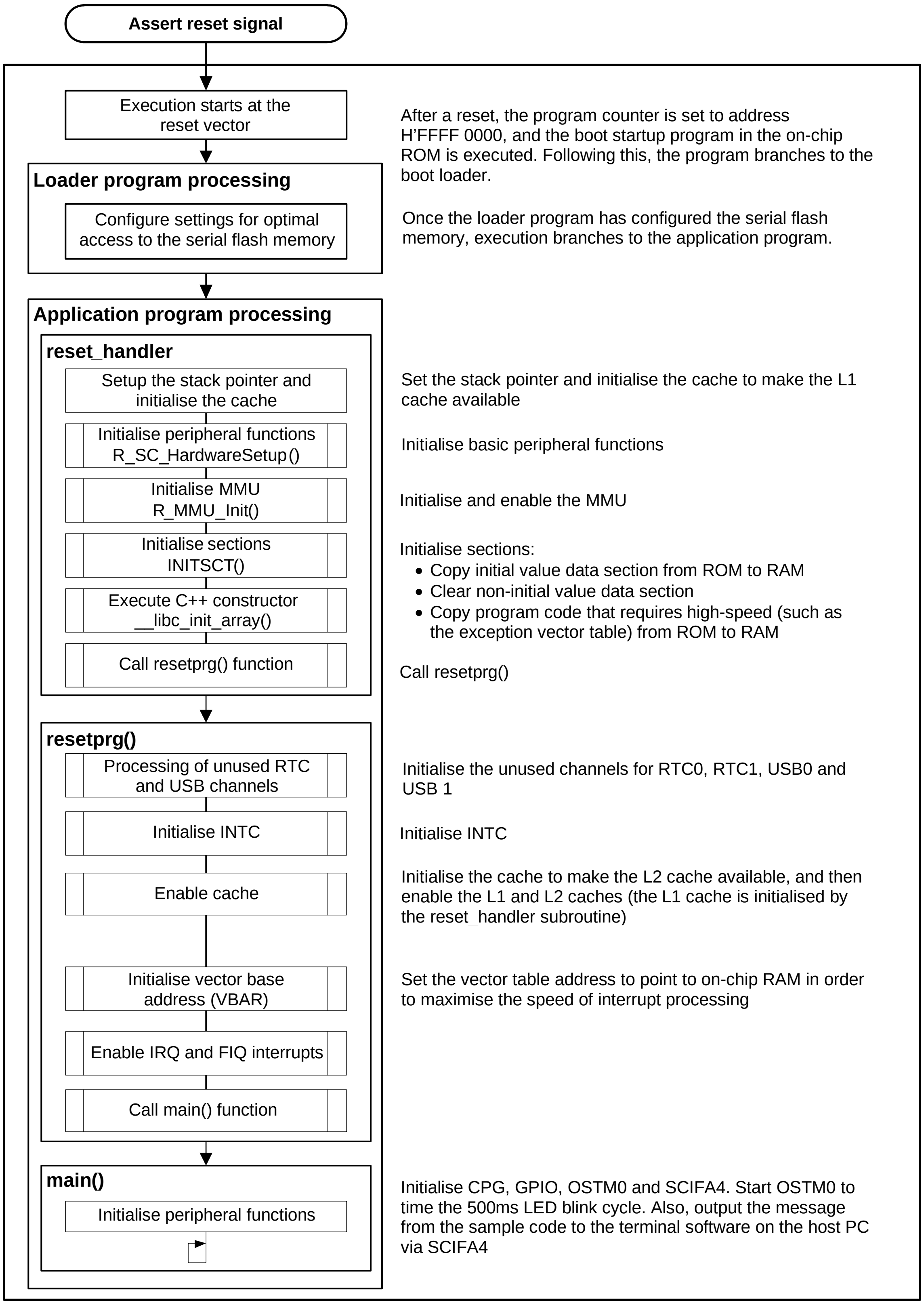

Assert reset signal
Execution starts at the reset vector
Loader program processing
Configure settings for optimal access to the serial flash memory
Application program processing
reset_handler
Setup the stack pointer and initialise the cache
Initialise peripheral functions R_SC_HardwareSetup()
Initialise MMU R_MMU_Init()
Initialise sections INITSCT()
Execute C++ constructor __libc_init_array()
Call resetprg() function
resetprg()
Processing of unused RTC and USB channels
Initialise INTC
Enable cache
Initialise vector base address (VBAR)
Enable IRQ and FIQ interrupts
Call main() function
main()
Initialise peripheral functions
After a reset, the program counter is set to address H'FFFF 0000, and the boot startup program in the on-chip ROM is executed. Following this, the program branches to the boot loader.
Once the loader program has configured the serial flash memory, execution branches to the application program.
Set the stack pointer and initialise the cache to make the L1 cache available
Initialise basic peripheral functions
Initialise and enable the MMU
Initialise sections:
• Copy initial value data section from ROM to RAM
• Clear non-initial value data section
• Copy program code that requires high-speed (such as the exception vector table) from ROM to RAM
Call resetprg()
Initialise the unused channels for RTC0, RTC1, USB0 and USB 1
Initialise INTC
Initialise the cache to make the L2 cache available, and then enable the L1 and L2 caches (the L1 cache is initialised by the reset_handler subroutine)
Set the vector table address to point to on-chip RAM in order to maximise the speed of interrupt processing
Initialise CPG, GPIO, OSTM0 and SCIFA4. Start OSTM0 to time the 500ms LED blink cycle. Also, output the message from the sample code to the terminal software on the host PC via SCIFA4

Figure 5.1 Sequence diagram of initial setup before main() function is invoked

5.2 Peripheral Setting and Memory Address Map in Sample Program

5.2.1 Peripheral setting

Table 5.1 shows the peripheral setting and memory configuration in the sample program.

Table 5.1 Peripheral setting in sample program

Peripheral	Setting
CPG	CPU clock: ½ of the PLL circuit frequency Internal bus clock: ⅛ of the PLL circuit frequency Peripheral clock 1: 1/16 of the PLL circuit frequency Each clock takes the following value under Clock Mode 1 (the frequency division ratio by divider 1 is ½, and the input clock frequency is multiplied by 88) when the input clock frequency is 24MHz: • CPU clock (Iϕ): 528MHz • Image processing clock (Gϕ): 264MHz • Internal bus clock (Bϕ): 132MHz • Peripheral clock 1 (P1ϕ): 66MHz • Peripheral clock 0 (P0ϕ): 33MHz • QSPI0 SPCLK: 66MHz (when selecting Bφ as output clock) • CKIO clock: 132MHz (when selecting Bφ as output clock)
STB	Make on-chip Data Retention RAM writable and enable clocks to the peripheral modules
GPIO	Configure the multiplexed pin function of PORT6 and PORT9 as follows: • P6_0: LED blinking • P9_1: Serial communications - RxD4, P9_0: TxD4
OSTM	Configure the channel 0 as interval timer mode: Counter is configured so that OSTM fires an interrupt every 1ms (when P1φ is specified as 66MHz) to provide the FreeRTOS tick interrupt
INTC	Initial setup of INTC Interrupt hander registration and invocation: • Registration of OSTM channel 0 interrupt handler (Interrupt ID: 88) • Registration of SCIFA channel 4 interrupt handler and its invocation (Interrupt ID: 321, 322 and 323)
SCIFA	Configure SCIFA channel 4 as asynchronous communication mode Serial communication setting: • Data length: 8-bits • Stop bits: 1 bit • Parity: None • LSB-first transfer SCIFA is configured as follows (when P1φ is 66MHz): • Clock source: No frequency division • Baud rate generator: double-speed mode • Operating on the base clock with 8 times the bit rate • Configures the appropriate bit rate for a baud rate of 115200bps Thus, the bit rate should become -0.53%

5.2.2 Memory Address Map

Figure 5.2 shows RZ/A2M group address space and memory address map of the RZ/A2M CPU board.

In the sample program, code and data destined for ROM is placed in the serial flash memory connected to SPI Multi I/O bus space, while code and data destined for RAM is placed in the on-chip large capacity internal RAM.

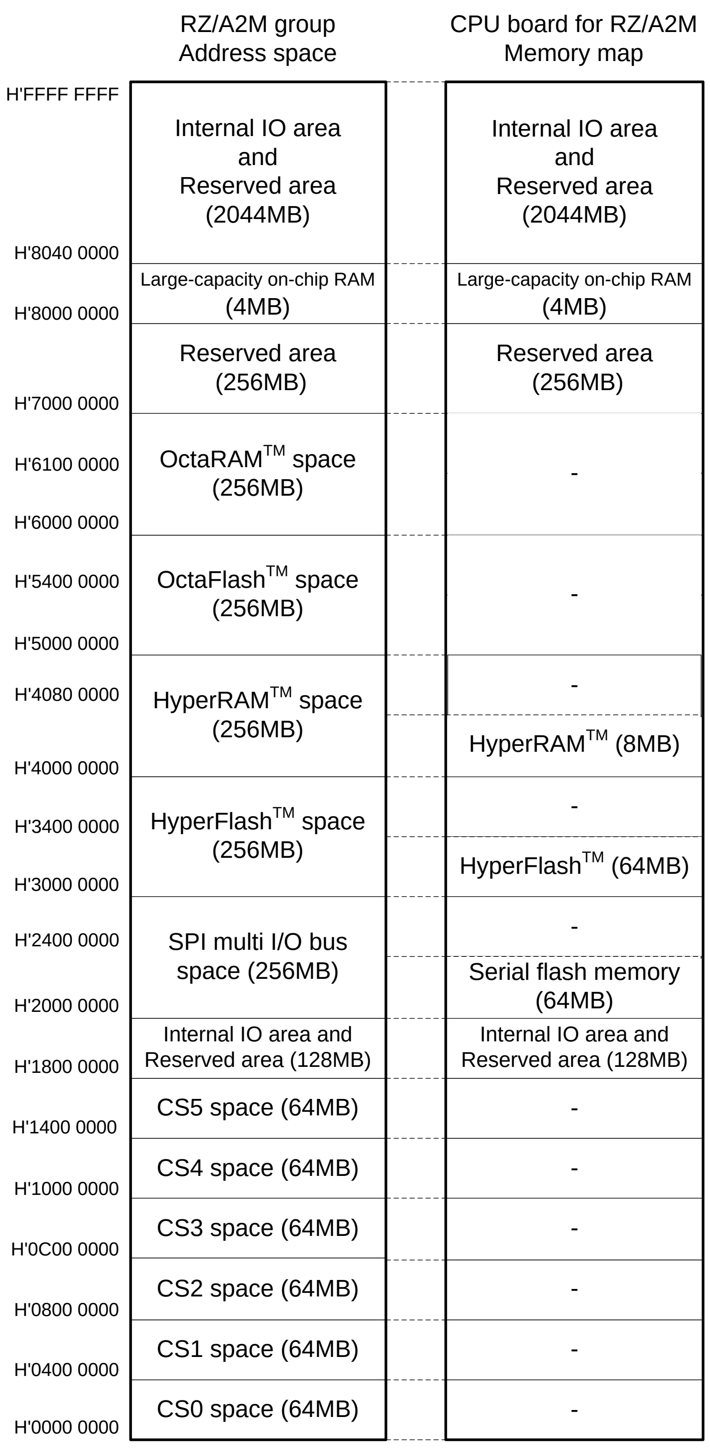


Figure 5.2 Memory Address Map

5.2.3 MMU Configuration

In the sample program, the 4GB sized area whose starting address is H’0000 0000 is managed by using a “section” descriptor classified with the first level descriptor according to the memory address map of the RZ/A2M CPU board. Please note that the MMU configuration table (MMU_SC_TABLE[ ]) is defined in the file r_mmu_drv_sc_cfg.h. When customizing the MMU configuration for your system, please note that sizes are specified in units of 1MB.

The translation table for the first level descriptor is configured by the Translation Table Base Control Register (TTBCR) and the Translation Table Base Register 0 or 1 (TTBR0 or TTBR1). After all the translation tables are configured, the MMU can be enabled by configuring the M bit in System Control Register (SCTLR). For full details on the MMU, please refer to “ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition Issue C”.

- TTBCR
    - This register determines the base register (either TTBR0 or TTBR1) used as the base address for the translation table. Also, it determines the size of translation table specified by TTBR0. In the sample program, b’000 is specified for the N[2:0] bits so that TTBR0 can be always used and the size of translation table can become 16KB (i.e. 4096 entries).
    - As shown in Figure 5.3, the first level translation table and section descriptor format are used as translation table and first level descriptor respectively. When using section descriptor format, each entry of translation table becomes 1MB sized memory block, and the conversion from virtual address to physical address using translation table is carried out against 4GB (4096 entries x 1MB) sized address space.
- TTBR
    - This register specifies the base address of translation table, cacheable attributes for the region where translation table is located, etc. In the sample code, the base address is “__mmu_page_table_base” defined in linker_script.ld, which is the starting address of section area of translation table located in the on-chip large capacity internal RAM.

Figure 5.3, Tables 5.2 and 5.3 show the overview of first level descriptor and its configuration in the sample program respectively. Tables 5.5 and 5.6 show the MMU setting in the sample program.

b31 20	19	18	17	16	15	14 12	11 10	9	8 5	4	3	2	1	0
Base address of section PA[31:20]	NS	0	nG	S	AP[2]	TEX [2:0]	AP [1:0]	IMP	Domain	XN	C	B	1	0

Figure 5.3 First level descriptor format when specifying Section for descriptor type

Table 5.2 Field of 1st level descriptor

Field	Description
TEX[2:0], C, B	Memory region attribute bits Please refer to Table 5.3 for the configuration in the sample program
XN	Execute-never bit When setting this bit to 1, the instruction shouldn't be fetched from the memory area located in the domain specified as client. In the sample program, Normal region is configured as executable (i.e. XN = 1), while Strongly-ordered region is configured as non-executable (i.e. XN = 0).
Domain	2 types of domain access are supported. One is client mode and the other is manager mode. Domain access type is specified by Domain Access Control Register (DACR). In the sample program, D15 field of DACR is configured as b'01 (i.e. client access mode) and b'1111 (D15 field) is specified for all the spaces.
IMP	This is NOT implemented in RZ/A2M and therefore, the configuration of this bit is ignored
AP[2], AP[1:0]	These bits and the domain determine if access is enabled. In the sample program, access to the reserved area is disabled (i.e. AP[2:0] = b'000), while access to all other areas is enabled (i.e. AP[2:0] = b'011).
S	Shareable bit that can determines if the memory area is shareable. In the sample program, All Normal regions are configured as non-shareable area (i.e. S=0).
nG	The non-Global bit In the sample program, all regions are configured as global (i.e. nG = 0)
NS	The Non-Secure bit In the sample program, on-chip large capacity RAM and external address space are configured as Non-Secure space (i.e. NS=1), while internal peripheral module space is configured as Secure space (i.e. NS=0).
PA[31:20]	Upper 12 bits of the physical address converted from virtual address

Note: For RZ/A2M, please configure CPU security state as the "Secure" (i.e. NS bit of Secure Configuration Register = 0). By getting CPU to configure "Secure", CPU can access all the area where its physical address is converted into virtual address using MMU regardless of NS bit setting.
In the sample code, the CPU accesses the on-chip large capacity RAM and external address space by setting NS bit to 1 on the basis that AXI bus related registers MSTACCCTL0, MSTACCCTL1, MSTACCCTL2, MSTACCCTL3 and MSTACCCTL4 and for on-chip peripheral modules are set as 1 (i.e. Non Secure Access). By default, MSTACCCTL0-4 registers are configured as 1.

Table 5.3 Memory attribute setting in the sample program

TEX[2:0]	C	B	Memory Type	L1 Cache	L2 Cache
b'000	0	0	Strongly-ordered	-	-
b'000	0	1	Device (shareable)	-	-
b'001	1	1	Normal memory	Enabled	Enabled
b'100	0	1	Normal memory	Enabled	Disabled
b'100	0	0	Normal memory	Disabled	Disabled

Table 5.4 MMU settings and its description

Name of MMU settings	Memory type	Cache	NS	AP[2:0] (Access Permission)	XN
Unused	Strongly-ordered	-	1	Read/Write (b'011)	1
Strongly-ordered (Secure, Never-execute)	Strongly-ordered	-	0	Read/Write (b'011)	1
Strongly-ordered (Non-secure, Never-execute)	Strongly-ordered	-	1	Read/Write (b'011)	1
Strongly-ordered (Non-secure, Executable)	Strongly-ordered	-	1	Read/Write (b'011)	0
Shareable Device (Secure)	Device	-	0	Read/Write (b'011)	1
Shareable Device (Non-secure)	Device	-	1	Read/Write (b'011)	1
Normal (Non-secure, L1 cacheable)	Normal	Only L1 cache is enabled	1	Read/Write (b'011)	0
Normal (Non-secure, L2 cacheable)	Normal	Only L2 cache is enabled	1	Read/Write (b'011)	0
Normal (Non-secure, L1/L2 cacheable)	Normal	Both L1 and L2 are enabled	1	Read/Write (b'011)	0
Normal (Non-secure, Non-cacheable)	Normal	Neither L1 nor L2 are enabled	1	Read/Write (b'011)	0
Reserved	Strongly-ordered	-	1	Access Inhibit (b'000)	1

Table 5.5 MMU settings (1/2)

Virtual address	Physical address	Size	Address area	Name of MMU settings (Note)
H'0000 0000 ~ H'03FF FFFF	H'0000 0000 ~ H'03FF FFFF	64MB	CS0 area (Unused)	Unused
H'0400 0000 ~ H'07FF FFFF	H'0400 0000 ~ H'07FF FFFF	64MB	CS1 area (Unused)	Unused
H'0800 0000 ~ H'0BFF FFFF	H'0800 0000 ~ H'0BFF FFFF	64MB	CS2 area (Unused)	Unused
H'0C00 0000 ~ H'0FFF FFFF	H'0C00 0000 ~ H'0FFF FFFF	64MB	CS3 area Cacheable	Normal (Non-secure, L1/L2 cacheable)
H'1000 0000 ~ H'13FF FFFF	H'1000 0000 ~ H'13FF FFFF	64MB	CS4 area (Unused)	Unused
H'1400 0000 ~ H'17FF FFFF	H'1400 0000 ~ H'17FF FFFF	64MB	CS5 area (Unused)	Unused
H'1800 0000 ~ H'1EFF FFFF	H'1800 0000 ~ H'1EFF FFFF	112MB	Reserved	Reserved
H'1F00 0000 ~ H'1FFF FFFF	H'1F00 0000 ~ H'1FFF FFFF	16MB	Internal I/O area	Strongly-ordered (Secure, Never-execute)
H'2000 0000 ~ H'2FFF FFFF	H'2000 0000 ~ H'2FFF FFFF	256MB	SPI Multi I/O bus area Cacheable	Normal (Non-secure, L1/L2 cacheable)
H'3000 0000 ~ H'3FFF FFFF	H'3000 000 ~ H'3FFF FFFF	256MB	HyperFlash Cacheable	Normal (Non-secure, L1/L2 cacheable)
H'4000 0000 ~ H'4FFF FFFF	H'4000 0000 ~ H'4FFF FFFF	256MB	HyperRAM Cacheable	Normal (Non-secure, L1/L2 cacheable)
H'5000 0000 ~ H'5FFF FFFF	H'5000 0000 ~ H'5FFF FFFF	256MB	OctaFlash Cacheable	Normal (Non-secure, L1/L2 cacheable)
H'6000 0000 ~ H'6FFF FFFF	H'6000 0000 ~ H'6FFF FFFF	256MB	OctaRAM Cacheable	Normal (Non-secure, L1/L2 cacheable)
H'7000 0000 ~ H'7FFF FFFF	H'7000 0000 ~ H'7FFF FFFF	256MB	Reserved	Unused
H'8000 0000 ~ H'803F FFFF	H'8000 0000 ~ H'803F FFFF	4MB	On-chip large capacity RAM Cacheable	Normal (Non-secure, L1/L2 cacheable)
H'8040 0000 ~ H'81FF FFFF	H'8040 0000 ~ H'81FF FFFF	28MB	Reserved	Unused
H'8200 0000 ~ H'823F FFFF	H'8000 0000 ~ H'803F FFFF	4MB	On-chip large capacity RAM Non-cacheable	Normal (Non-secure, Non-cacheable)
H'8240 0000 ~ H'87FF FFFF	H'8240 0000 ~ H'87FF FFFF	92MB	Reserved	Unused
H'8800 0000 ~ H'8BFF FFFF	H'0000 0000 ~ H'03FF FFFF	64MB	CS0 area (Unused)	Unused
H'8C00 0000 ~ H'8FFF FFFF	H'0400 0000 ~ H'07FF FFFF	64MB	CS1 area (Unused)	Unused
H'9000 0000 ~ H'93FF FFFF	H'0800 0000 ~ H'0BFF FFFF	64MB	CS2 area (Unused)	Unused
H'9400 0000 ~ H'97FF FFFF	H'0C00 0000 ~ H'0FFF FFFF	64MB	CS3 area Non-cacheable	Normal (Non-secure, Non-cacheable)
H'9800 0000 ~ H'9BFF FFFF	H'1000 0000 ~ H'13FF FFFF	64MB	CS4 area (Unused)	Unused

Table 5.6 MMU settings (2/2)

Virtual address	Physical address	Size	Address area	Name of MMU settings (Note)
H’9C00 0000 ~ H’9FFF FFFF	H’1400 0000 ~ H’17FF FFFF	64MB	CS5 area (Unused)	Unused
H’A000 0000 ~ H’AFFF FFFF	H’3000 0000 ~ H’3FFF FFFF	256MB	HyperFlash Non-cacheable	Strongly-ordered (Non-secure, Executable)
H’B000 0000 ~ H’BFFF FFFF	0x4000 0000 ~ H’4FFF FFFF	256MB	HyperRAM Non-cacheable	Normal (Non-secure, Non-cacheable)
H’C000 0000 ~ H’CFFF FFFF	H’5000 0000 ~ H’5FFF FFFFH’	256MB	OctaFlash Non-cacheable	Strongly-ordered (Non-secure, Non-cacheable)
H’D000 0000 ~ H’DFFF FFFF	H’6000 0000 ~ H’6FFF FFFF	256MB	OctaRAM Non-cacheable	Normal (Non-secure, Non-cacheable)
H’E000 0000 ~ H’E7FF FFFF	H’E000 0000 ~ H’E7FF FFFF	128MB	Reserved	Reserved
H’E800 0000 ~ H’FFFF FFFF	H’E800 0000 ~ H’FFFF FFFF	384MB	Internal I/O area	Strongly-ordered (Secure, Never-execute)

Notes: 1. For the relationship between MMU settings and the corresponding name, please refer to Tables 5.3 and 5.4.
2. The physical address is shown in red when the virtual address is different from the corresponding physical address.

5.2.4 Virtual Address Space in Sample Program

Figures 5.4 and 5.5 show the virtual address space which is configured using the MMU and accessible from CPU.

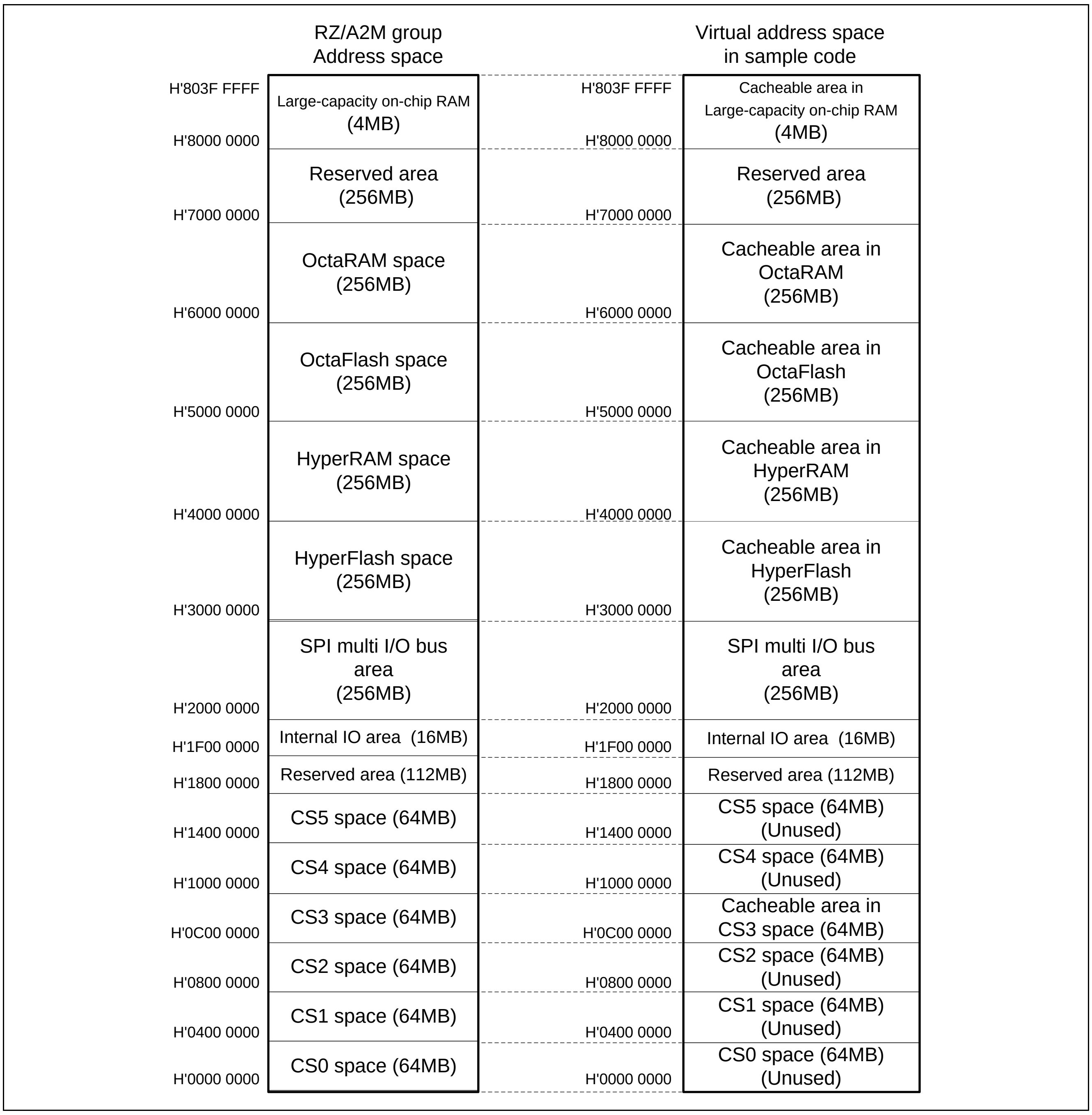
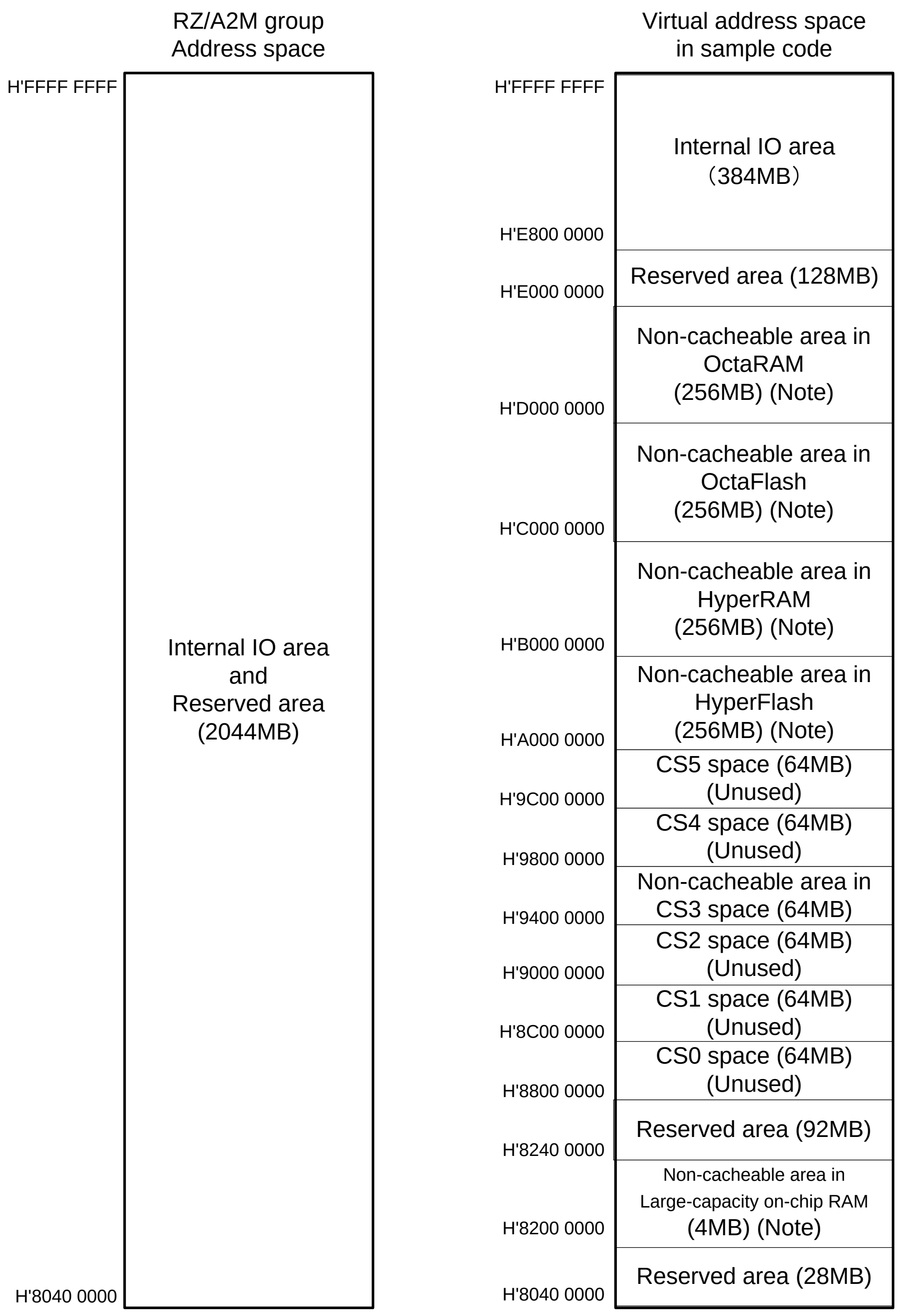

RZ/A2M group Address space
Virtual address space in sample code
H'803F FFFF
H'8000 0000
Large-capacity on-chip RAM (4MB)
Cacheable area in Large-capacity on-chip RAM (4MB)
Reserved area (256MB)
H'7000 0000
OctaRAM space (256MB)
Cacheable area in OctaRAM (256MB)
H'6000 0000
OctaFlash space (256MB)
Cacheable area in OctaFlash (256MB)
H'5000 0000
HyperRAM space (256MB)
Cacheable area in HyperRAM (256MB)
H'4000 0000
HyperFlash space (256MB)
Cacheable area in HyperFlash (256MB)
H'3000 0000
SPI multi I/O bus area (256MB)
H'2000 0000
Internal IO area (16MB)
H'1F00 0000
Reserved area (112MB)
H'1800 0000
CS5 space (64MB)
CS5 space (64MB) (Unused)
H'1400 0000
CS4 space (64MB)
CS4 space (64MB) (Unused)
H'1000 0000
CS3 space (64MB)
Cacheable area in CS3 space (64MB)
H'0C00 0000
CS2 space (64MB)
CS2 space (64MB) (Unused)
H'0800 0000
CS1 space (64MB)
CS1 space (64MB) (Unused)
H'0400 0000
CS0 space (64MB)
CS0 space (64MB) (Unused)
H'0000 0000

Figure 5.4 Virtual address space in the sample program (1/2)



Note: For each space of large capacity internal RAM, HyperFlash, HyperRAM, OctaFlash, and OctaRAM, it is prepared the area is able to use as the non-cacheable area. The non-cache area of HyperFlash and OctaFlash sets MMU to strongly-odered memory attribute.

Figure 5.5 Virtual address space in the sample program (2/2)

5.2.5 Section setup in the sample program

In this sample code, the exception processing vector table and the IRQ interrupt handler are placed in the large-capacity on-chip RAM, and they are executed in this RAM to maximise interrupt processing speed. The transfer of execution from the serial flash memory area where the program code for the exception processing vector table and the IRQ interrupt handler resides to the large-capacity on-chip RAM area, the clear to zero processing for the data section without initial data, and the initialisation for the data section with initial data are executed by using INITSCT function. The INITSCT function refers to the table data for section initialisation defined in section.c and initialise each section. The arrangement of program data is described in the linker script (linker_script.ld).

Table 5.7 Memory Area Used

Output section Name	Input section Name Input Object Name	Description	Loading Area	Execution Area
LOAD_MODULE1	VECTOR_TABLE	Exception processing vector table	FLASH	FLASH
LOAD_MODULE2	*/r_cpg/*.o (.text .rodata .data)	Program code area for CPG	FLASH	LRAM
LOAD_MODULE3	*/r_cpg/*.o (.bss)	Data area with initial value for CPG	-	LRAM
LOAD_MODULE4	RESET_HANDLER	Program code area of reset handler processing	FLASH	FLASH
	INIT_SECTION */sections.o	Program code area of section initialization processing		
.data	VECTOR_MIRROR_TABLE	Exception processing vector table (on-chip RAM)	FLASH	LRAM
	*/r_intc_*.o (.text .rodata .data)	Program code area for INTC Driver		
	IRQ_FIQ_HANDLER	IRQ/FIQ handler processing		
	*/rza_io_regrw.o (.text .rodata .data)	Program code area for function of I/O register access		
.bss	*/rza_io_regrw.o (.bss)	Data area without initial value for function of I/O register access	-	LRAM
.uncached_RAM	*/r_cache_*.o (.bss) UNCACHED_BSS	Data area without initial value for setting the L1 and L2 caches (see Note 2)	-	LRAM
.uncached_RAM2	*/r_cache_*.o (.text .rodata .data)	Program code area for setting the L1 and L2 caches (see Note 2)	FLASH	LRAM
.mmu_page_table	none	MMU translation table area	-	LRAM
.stack	none	System mode stack area	-	LRAM
		IRQ mode stack area		
		FIQ mode stack area		
		Supervisor (SVC) mode stack area		
		Abort (ABT) mode stack area		
.text2	* (.text .text.*)	Program code area for defaults	FLASH	LRAM
	* (.rodata .rodata.*)	Constant data area for defaults		
.data2	* (.data .data.*)	Data area with initial value for defaults	FLASH	LRAM
.bss2	* (.bss .bss.*) * (COMMON)	Data area without initial value for defaults	-	LRAM
.heap	none	Heap area	-	LRAM
.freertos_heap	none	FreeRTOS heap area	-	LRAM

Notes: 1. "FLASH" and "LRAM" shown in Loading Area and Execution Area indicate the serial flash memory area and the large-capacity on-chip RAM area respectively.
2. This section should be placed in the cache-disable area.

Table 5.7 lists the sections used in the sample code. Figure 5.6 shows the section assignment for the initial condition of the sample code and the condition after using INITSCT function.

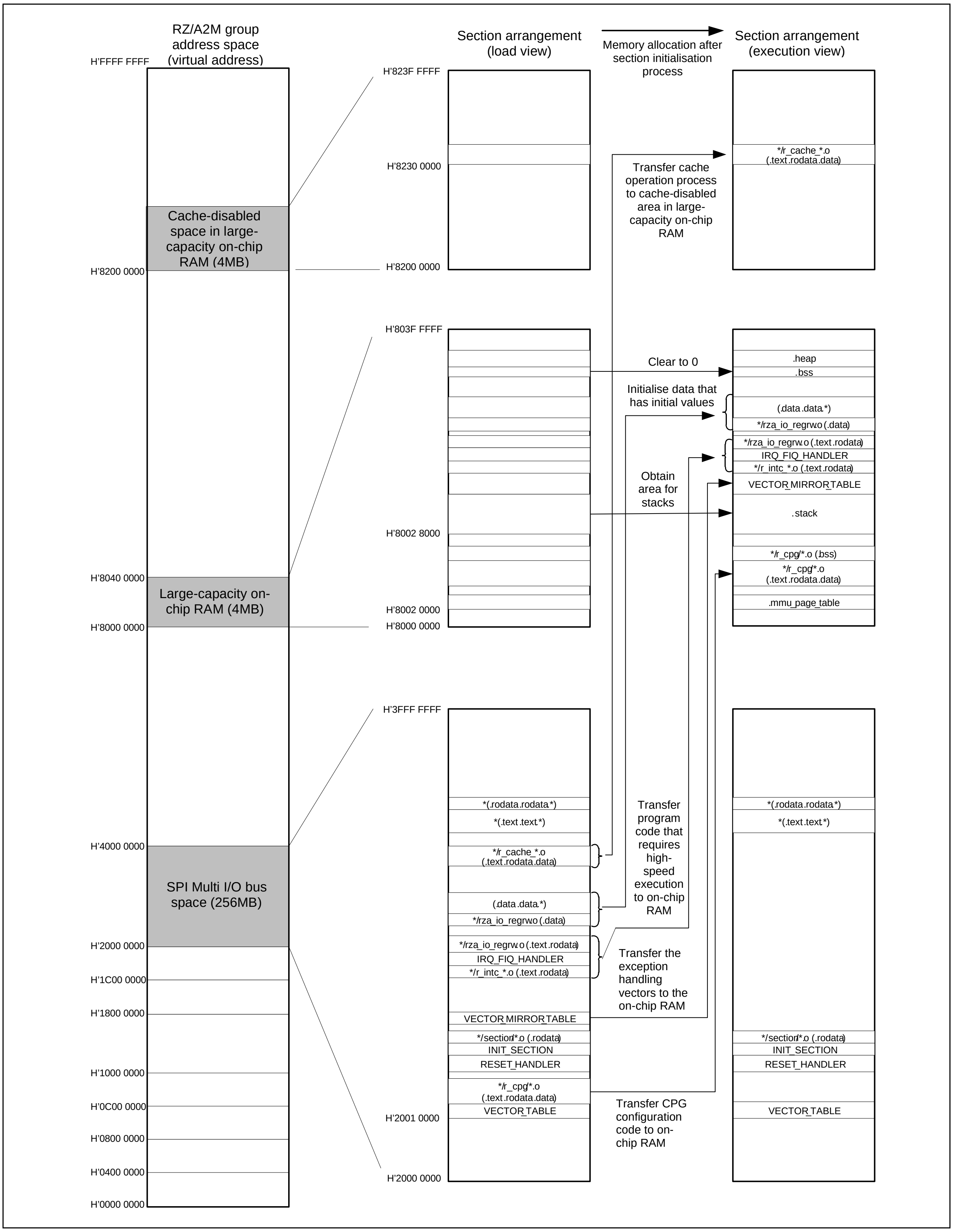

RZ/A2M group address space (virtual address)
H'FFFF FFFF
Cache-disabled space in large-capacity on-chip RAM (4MB)
H'8200 0000
H'8040 0000
Large-capacity on-chip RAM (4MB)
H'8000 0000
H'4000 0000
SPI Multi I/O bus space (256MB)
H'2000 0000
H'1C00 0000
H'1800 0000
H'1000 0000
H'0C00 0000
H'0800 0000
H'0400 0000
H'0000 0000
Section arrangement (load view)
Memory allocation after section initialisation process
Section arrangement (execution view)
H'823F FFFF
H'8230 0000
H'8200 0000
*/r_cache_*.o (.text.rodata.data)
Transfer cache operation process to cache-disabled area in large-capacity on-chip RAM
H'803F FFFF
Clear to 0
.heap
.bss
Initialise data that has initial values
(.data.data.*)
*/rza_io_regrw.o (.data)
*/rza_io_regrw.o (.text.rodata)
IRQ_FIQ_HANDLER
*/r_intc_*.o (.text.rodata)
VECTOR_MIRROR_TABLE
Obtain area for stacks
.stack
H'8002 8000
/r_cpg/.o (.bss)
/r_cpg/.o (.text.rodata.data)
.mmu_page_table
H'8002 0000
H'8000 0000
H'3FFF FFFF
(.rodata.rodata.)
(.text.text.)
*/r_cache_*.o (.text.rodata.data)
(.data.data.*)
*/rza_io_regrw.o (.data)
*/rza_io_regrw.o (.text.rodata)
IRQ_FIQ_HANDLER
*/r_intc_*.o (.text.rodata)
VECTOR_MIRROR_TABLE
/section/.o (.rodata)
INIT_SECTION
RESET_HANDLER
/r_cpg/.o (.text.rodata.data)
VECTOR_TABLE
H'2001 0000
H'2000 0000
Transfer program code that requires high-speed execution to on-chip RAM
Transfer the exception handling vectors to the on-chip RAM
Transfer CPG configuration code to on-chip RAM

Figure 5.6 Section Assignment

5.2.6 L1/L2 Cache Settings

The RZ/A2M has two types of cache, L1 cache and L2 cache. L1 cache consists of 32K byte instruction cache and 32K byte data cache, while L2 cache consists of 128K byte cache which is commonly used for instruction and data.

Figure 5.7 shows the block diagram for L1 and L2 cache in memory hierarchy.

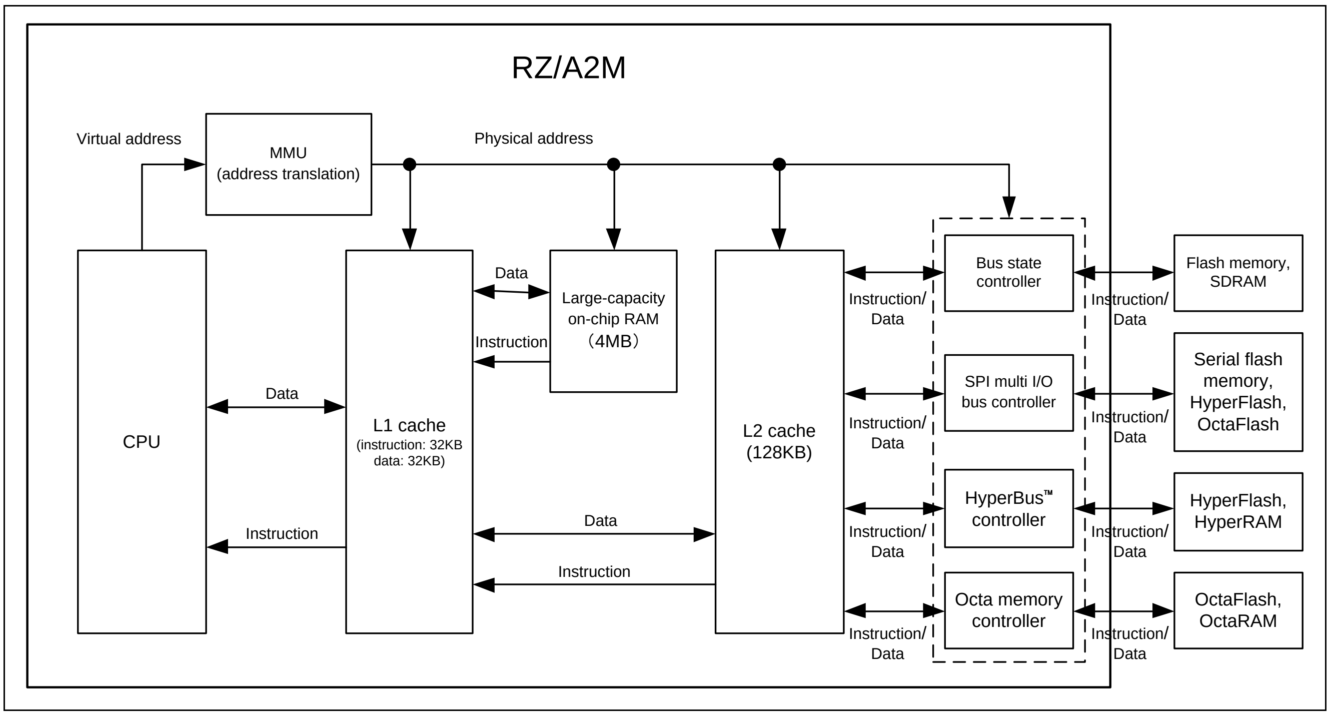


Figure 5.7 Block diagram of L1/L2 cache in memory hierarchy

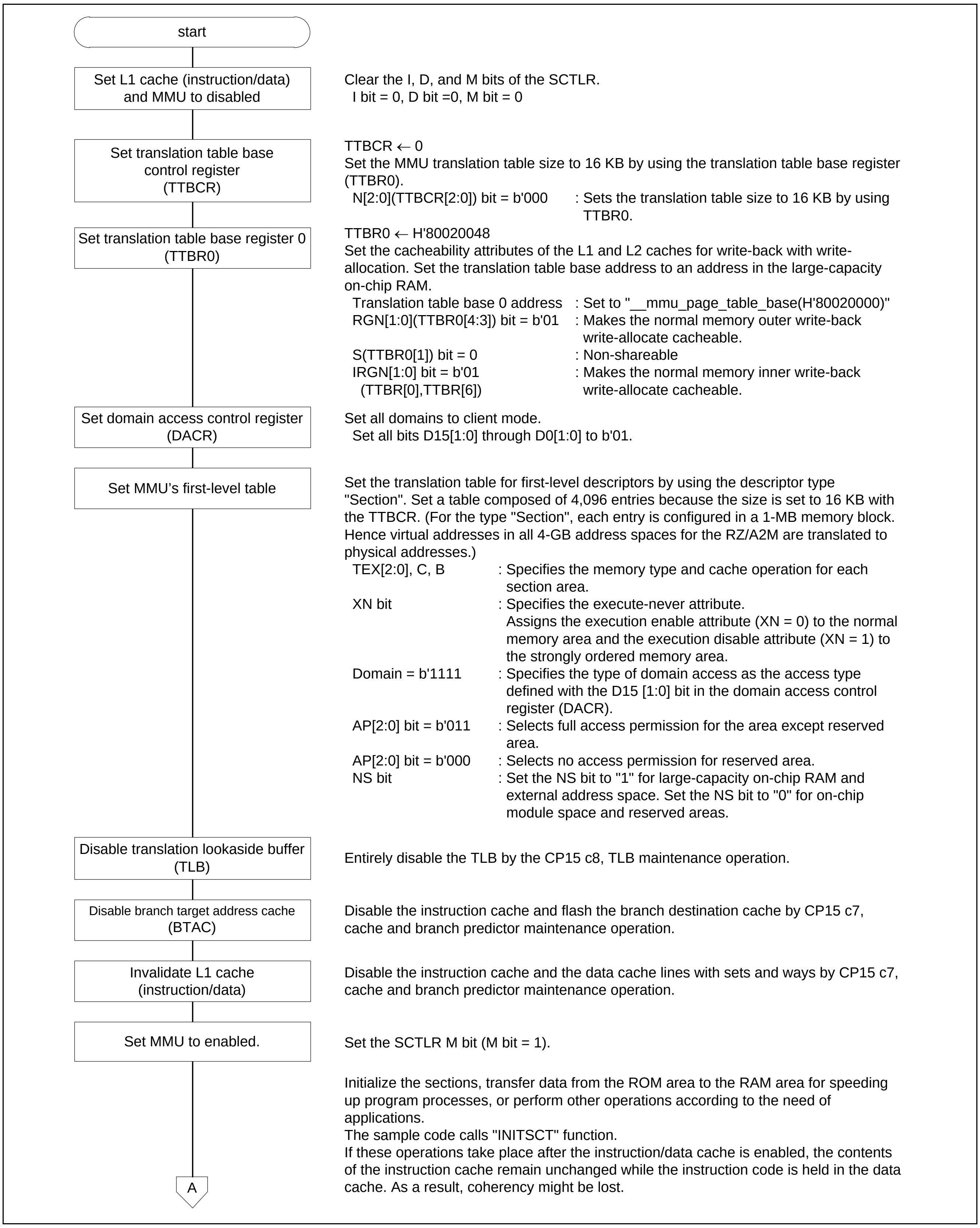
When accessing external memory space or on-chip large capacity RAM from the CPU, the virtual address is converted into a physical address using the MMU translation table. As shown in Figure 5.3, the cache behavior of the memory area whose base address and size are determined by the first level descriptor should also be specified by the first level descriptor. Please note that the size of area should be 1MB when “Section” is specified as first level descriptor type. For RZ/A2M, cache behavior of both external memory spaces (CS0 to CS5) and on-chip memory space (such as on-chip peripheral module, SPI Multi I/O Bus space, HyperFlash/HyperRAM space, OctaFlash/OctaRAM space on-chip large capacity RAM, on-chip data retention RAM and reserved area) should be set up by MMU translation table.

Enabling/Disabling cache is controlled by the registers stated below:

- CP15 System Control Register (SCTLR) : I bit (b12), C bit (b2)
  I bit “0” denotes instruction cache is disabled, while “1” denotes instruction cache is enabled
  C bit “0” denotes data cache is disabled, while “1” denotes data cache is enabled
  Please note that SCTLR has M bit (b0) which enables/disables MMU (“1” denotes MMU is enabled)
- Control Register (reg1_control) : L2 Cache enable bit (b0)
  L2 Cache enable bit “0” denotes L2 Cache is disabled, while “1” denotes L2 Cache is enabled
  Please note that reg1_control is the register implemented in CoreLink level2 cache controller (L2C-310).

Note: The Direct Memory Access Controller (DMAC) can’t access memory via the L1 cache, while the L2 cache is enabled. So, if both the CPU and the DMAC access a memory area where the L1 cache is enabled, the user application should invoke the appropriate cache operation to ensure cache coherency before starting the DMA transfer.

Figure 5.8 and 5.9 shows how L1 and L2 cache are initialized respectively.


start
Set L1 cache (instruction/data) and MMU to disabled
Set translation table base control register (TTBCR)
Set translation table base register 0 (TTBR0)
Set domain access control register (DACR)
Set MMU's first-level table
Disable translation lookaside buffer (TLB)
Disable branch target address cache (BTAC)
Invalidate L1 cache (instruction/data)
Set MMU to enabled.
A

Clear the I, D, and M bits of the SCTLR.
I bit = 0, D bit =0, M bit = 0

TTBCR ← 0
Set the MMU translation table size to 16 KB by using the translation table base register (TTBR0).

N[2:0](TTBCR[2:0]) bit = b'000	: Sets the translation table size to 16 KB by using TTBR0.

TTBR0 ← H'80020048
Set the cacheability attributes of the L1 and L2 caches for write-back with write-allocation. Set the translation table base address to an address in the large-capacity on-chip RAM.

Translation table base 0 address	: Set to "__mmu_page_table_base(H'80020000)"
RGN[1:0](TTBR0[4:3]) bit = b'01	: Makes the normal memory outer write-back write-allocate cacheable.
S(TTBR0[1]) bit = 0	: Non-shareable
IRGN[1:0] bit = b'01 (TTBR[0],TTBR[6])	: Makes the normal memory inner write-back write-allocate cacheable.

Set all domains to client mode.
Set all bits D15[1:0] through D0[1:0] to b'01.

Set the translation table for first-level descriptors by using the descriptor type "Section". Set a table composed of 4,096 entries because the size is set to 16 KB with the TTBCR. (For the type "Section", each entry is configured in a 1-MB memory block. Hence virtual addresses in all 4-GB address spaces for the RZ/A2M are translated to physical addresses.)

TEX[2:0], C, B	: Specifies the memory type and cache operation for each section area.
XN bit	: Specifies the execute-never attribute. Assigns the execution enable attribute (XN = 0) to the normal memory area and the execution disable attribute (XN = 1) to the strongly ordered memory area.
Domain = b'1111	: Specifies the type of domain access as the access type defined with the D15 [1:0] bit in the domain access control register (DACR).
AP[2:0] bit = b'011	: Selects full access permission for the area except reserved area.
AP[2:0] bit = b'000	: Selects no access permission for reserved area.
NS bit	: Set the NS bit to "1" for large-capacity on-chip RAM and external address space. Set the NS bit to "0" for on-chip module space and reserved areas.

Entirely disable the TLB by the CP15 c8, TLB maintenance operation.

Disable the instruction cache and flash the branch destination cache by CP15 c7, cache and branch predictor maintenance operation.

Disable the instruction cache and the data cache lines with sets and ways by CP15 c7, cache and branch predictor maintenance operation.

Set the SCTLR M bit (M bit = 1).

Initialize the sections, transfer data from the ROM area to the RAM area for speeding up program processes, or perform other operations according to the need of applications.
The sample code calls "INITSCT" function.
If these operations take place after the instruction/data cache is enabled, the contents of the instruction cache remain unchanged while the instruction code is held in the data cache. As a result, coherency might be lost.

Figure 5.8 Flow of L1 and L2 Cache Initial Setup (1/2)

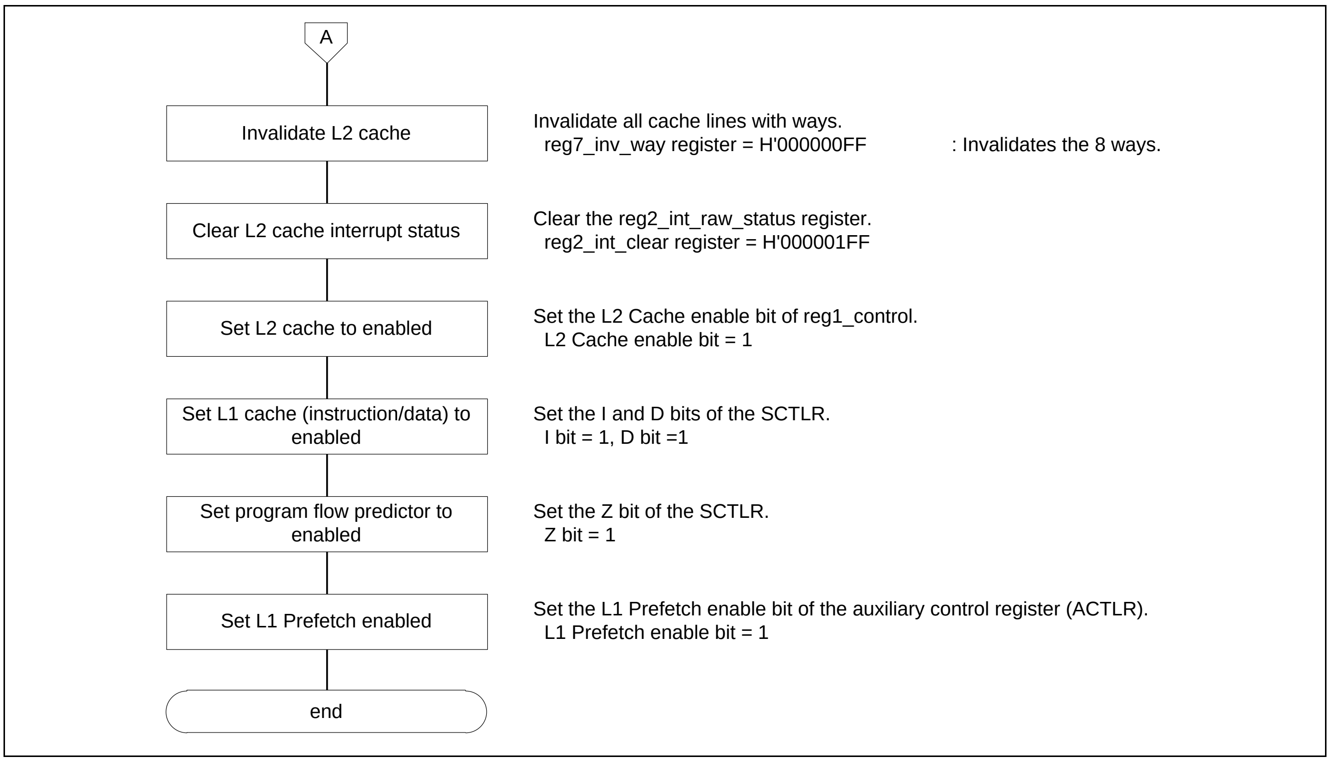

A
Invalidate L2 cache
Invalidate all cache lines with ways.
reg7_inv_way register = H'000000FF : Invalidates the 8 ways.
Clear L2 cache interrupt status
Clear the reg2_int_raw_status register.
reg2_int_clear register = H'000001FF
Set L2 cache to enabled
Set the L2 Cache enable bit of reg1_control.
L2 Cache enable bit = 1
Set L1 cache (instruction/data) to enabled
Set the I and D bits of the SCTLR.
I bit = 1, D bit =1
Set program flow predictor to enabled
Set the Z bit of the SCTLR.
Z bit = 1
Set L1 Prefetch enabled
Set the L1 Prefetch enable bit of the auxiliary control register (ACTLR).
L1 Prefetch enable bit = 1
end

Figure 5.9 Flow of L1 and L2 Cache Initial Setup (2/2)

5.2.7 Exception Vector Table

On RZ/A2M, the 7 types of exception (i.e. reset exception, undefined instruction exception, software interrupt exception, prefetch abort exception, data abort exception, IRQ exception and FIQ exception) might occur. When the boot mode is configured as Boot Mode 3, exception vector table is placed in the area whose starting address and size are H'2000 0000 and 32-bytes respectively. That means the exception table is placed from the address H'2000 0000 to H'2000 001F. And the branch instruction to each exception should be described in the exception vector table.

Figure 5.10 shows the exception vector table implemented in the sample program for your reference.

```
vector_table
    LDR pc, =R_OS_ABSTRACTION_CFG_PRV_RESET_HANDLER
    LDR pc, =R_OS_ABSTRACTION_CFG_PRV_UNDEFINED_HANDLER
    LDR pc, =R_OS_ABSTRACTION_CFG_PRV_SVC_HANDLER
    LDR pc, =R_OS_ABSTRACTION_CFG_PRV_PREFETCH_HANDLER
    LDR pc, =R_OS_ABSTRACTION_CFG_PRV_ABORT_HANDLER
    LDR pc, =R_OS_ABSTRACTION_CFG_PRV_RESERVED_HANDLER
    LDR pc, =R_OS_ABSTRACTION_CFG_PRV_IRQ_HANDLER
    LDR pc, =R_OS_ABSTRACTION_CFG_PRV_FIQ_HANDLER
```

Figure 5.10 Example of Exception Vector Table Implementation

5.2.8 Processing for RTC and USB unused channel

In the sample program, the processing for RTC and USB unused channel is implemented at the top of resetprg function to reduce power consumption. The macro definition listed in Table 5.8 determines if the processing for unused processing is carried out.

Table 5.8 Macro Definition for selecting RTC and USB channel to be used

Macro definition	Settings	Description
STARTUP_CFG_DISABLE_RTC0	0	RTC channel 0 is used
	1 (Default)	RTC channel 0 is NOT used
STARTUP_CFG_DISABLE_RTC1	0	RTC channel 1 is used
	1 (Default)	RTC channel 1 is NOT used
STARTUP_CFG_DISABLE_USB0	0	USB channel 0 is used
	1 (Default)	USB channel 0 is NOT used
STARTUP_CFG_DISABLE_USB1	0	USB channel 1 is used
	1 (Default)	USB channel 1 is NOT used

Figure 5.11 shows the processing flow for RTC unused channel.

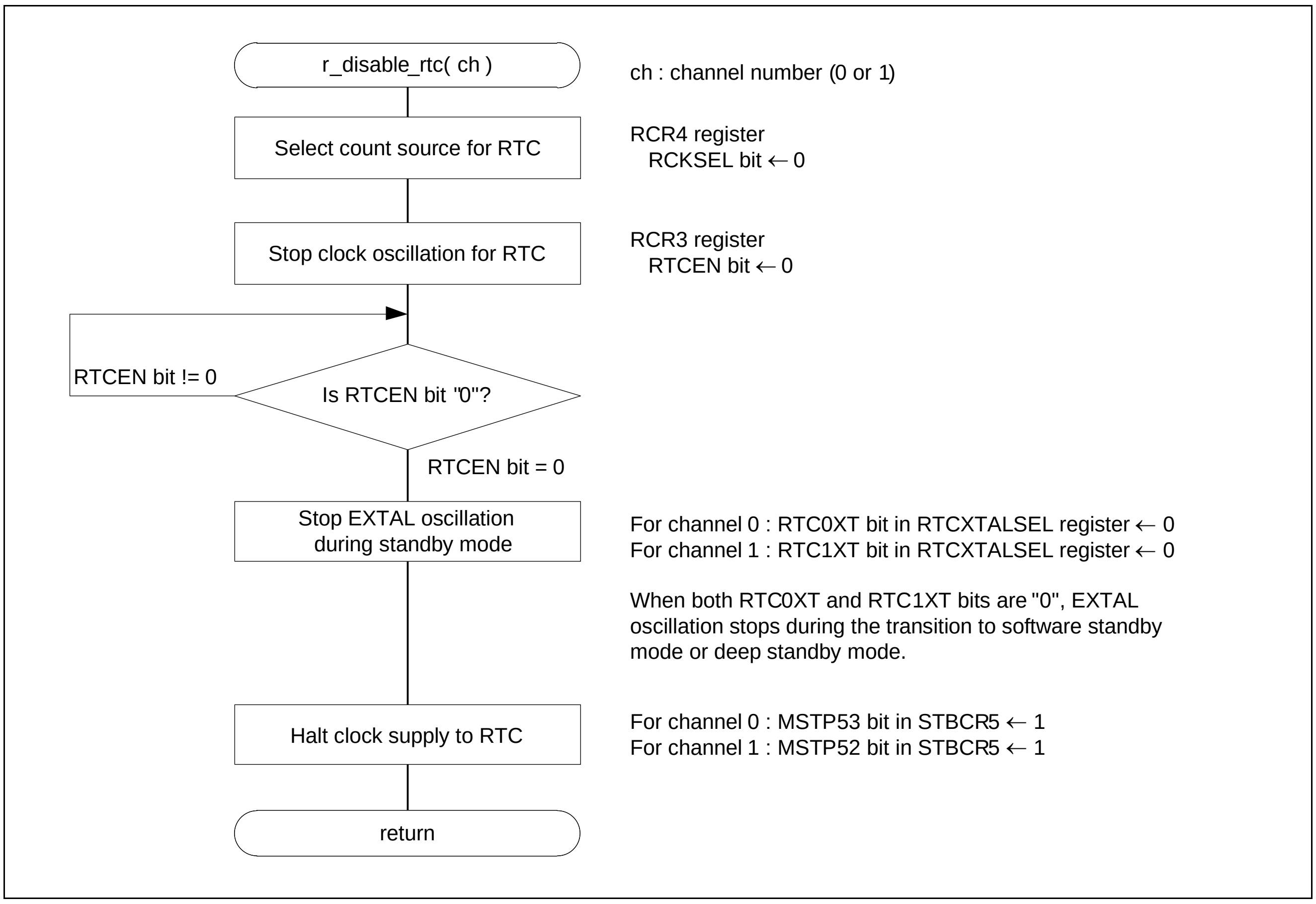

r_disable_rtc(ch)
ch : channel number (0 or 1)
Select count source for RTC
RCR4 register
RCKSEL bit ← 0
Stop clock oscillation for RTC
RCR3 register
RTCEN bit ← 0
RTCEN bit != 0
Is RTCEN bit "0"?
RTCEN bit = 0
Stop EXTAL oscillation during standby mode
For channel 0 : RTC0XT bit in RTCXTALSEL register ← 0
For channel 1 : RTC1XT bit in RTCXTALSEL register ← 0
When both RTC0XT and RTC1XT bits are "0", EXTAL oscillation stops during the transition to software standby mode or deep standby mode.
Halt clock supply to RTC
For channel 0 : MSTP53 bit in STBCR5 ← 1
For channel 1 : MSTP52 bit in STBCR5 ← 1
return

Figure 5.11 Processing flow for RTC unused channel

Figure 5.12 shows the processing flow for USB unused channel.

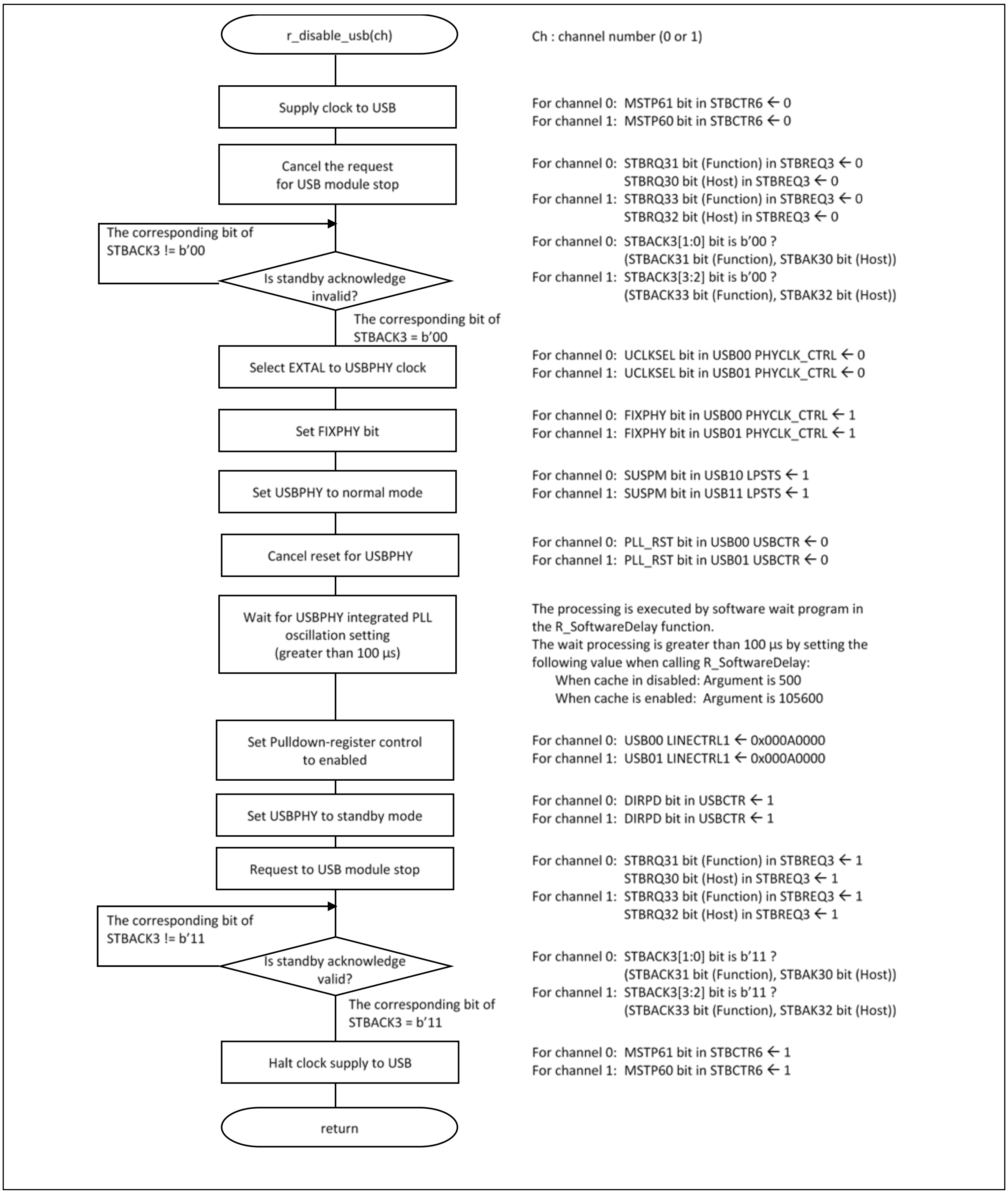

r_disable_usb(ch)
Ch : channel number (0 or 1)
Supply clock to USB
For channel 0: MSTP61 bit in STBCTR6 ← 0
For channel 1: MSTP60 bit in STBCTR6 ← 0
Cancel the request for USB module stop
For channel 0: STBRQ31 bit (Function) in STBREQ3 ← 0
STBRQ30 bit (Host) in STBREQ3 ← 0
For channel 1: STBRQ33 bit (Function) in STBREQ3 ← 0
STBRQ32 bit (Host) in STBREQ3 ← 0
The corresponding bit of STBACK3 != b'00
Is standby acknowledge invalid?
The corresponding bit of STBACK3 = b'00
For channel 0: STBACK3[1:0] bit is b'00 ?
(STBACK31 bit (Function), STBAK30 bit (Host))
For channel 1: STBACK3[3:2] bit is b'00 ?
(STBACK33 bit (Function), STBAK32 bit (Host))
Select EXTAL to USBPHY clock
For channel 0: UCLKSEL bit in USB00 PHYCLK_CTRL ← 0
For channel 1: UCLKSEL bit in USB01 PHYCLK_CTRL ← 0
Set FIXPHY bit
For channel 0: FIXPHY bit in USB00 PHYCLK_CTRL ← 1
For channel 1: FIXPHY bit in USB01 PHYCLK_CTRL ← 1
Set USBPHY to normal mode
For channel 0: SUSPM bit in USB10 LPSTS ← 1
For channel 1: SUSPM bit in USB11 LPSTS ← 1
Cancel reset for USBPHY
For channel 0: PLL_RST bit in USB00 USBCTR ← 0
For channel 1: PLL_RST bit in USB01 USBCTR ← 0
Wait for USBPHY integrated PLL oscillation setting (greater than 100 µs)
The processing is executed by software wait program in the R_SoftwareDelay function.
The wait processing is greater than 100 µs by setting the following value when calling R_SoftwareDelay:
When cache in disabled: Argument is 500
When cache is enabled: Argument is 105600
Set Pulldown-register control to enabled
For channel 0: USB00 LINECTRL1 ← 0x000A0000
For channel 1: USB01 LINECTRL1 ← 0x000A0000
Set USBPHY to standby mode
For channel 0: DIRPD bit in USBCTR ← 1
For channel 1: DIRPD bit in USBCTR ← 1
Request to USB module stop
For channel 0: STBRQ31 bit (Function) in STBREQ3 ← 1
STBRQ30 bit (Host) in STBREQ3 ← 1
For channel 1: STBRQ33 bit (Function) in STBREQ3 ← 1
STBRQ32 bit (Host) in STBREQ3 ← 1
The corresponding bit of STBACK3 != b'11
Is standby acknowledge valid?
The corresponding bit of STBACK3 = b'11
For channel 0: STBACK3[1:0] bit is b'11 ?
(STBACK31 bit (Function), STBAK30 bit (Host))
For channel 1: STBACK3[3:2] bit is b'11 ?
(STBACK33 bit (Function), STBAK32 bit (Host))
Halt clock supply to USB
For channel 0: MSTP61 bit in STBCTR6 ← 1
For channel 1: MSTP60 bit in STBCTR6 ← 1
return

Figure 5.12 Processing flow for USB unused channel

5.3 Interrupts to be used

Table 5.9 shows the interrupts used in the sample program.

Table 5.9 Interrupts to be used in the sample program

Interrupt Source (Interrupt ID)	Priority	Overview of Interrupt Processing
OSTM0 (88)	3	Fire interrupt every 500ms
ERI4BRI4 (321)	9	Fire SCIFA's ERI4/BRI4 interrupt
RXI4 (322)	10	Fire SCIFA's RXI4 interrupt
TXI4 (323)	10	Fire SCIFA's TXI4 interrupt

5.4 Fixed-width types

Table 5.10 shows the fixed-width types used in the sample program.

Table 5.10 Fixed-width types used in the sample program

Symbol	Description
char_t	8-bits character
bool_t	Boolean type. Allowable values are true (1) or false (0)
int_t	High-speed signed integer. In the sample code its width is 32 bits
int8_t	Signed 8-bit integer (declared in the standard library stdint.h)
int16_t	Signed 16-bit integer (declared in the standard library stdint.h)
int32_t	Signed 32-bit integer (declared in the standard library stdint.h)
int64_t	Signed 64-bit integer (declared in the standard library stdint.h)
uint8_t	Unsigned 8-bit integer (declared in the standard library stdint.h)
uint16_t	Unsigned 16-bit integer (declared in the standard library stdint.h)
uint32_t	Unsigned 32-bit integer (declared in the standard library stdint.h)
uint64_t	Unsigned 64-bit integer (declared in the standard library stdint.h)
float_t	Single precision floating point number (declared in the standard library math.h)
double_t	Double precision floating point number (declared in the standard library math.h)
float64_t	Double precision floating point number

5.5 Constants

Table 5.11 shows the constants used in the sample program.

Table 5.11 Constants used in the sample program

Constants	Value	Description
MAIN_PRV_LED_ON	(1)	Denotes LED is ON
MAIN_PRV_LED_OFF	(0)	Denotes LED is OFF

5.5.1 Direct related constants

Table 5.12 shows the device driver definition data (const st_r_driver_t) in the sample program. Any function whose name starts with the peripheral name (e.g. cpg_open) is implemented. Any function whose name starts with "no_dev_" (i.e. no_dev_****) returns with no processing.

Table 5.12 Device Driver Definition Data (const st_r_driver_t)

Constants	Value	Description
g_cpg_driver	In the sample program, the following values are implemented:	CPG driver definition data:
	"Clock Pulse Generator Driver",	Driver name
	cpg_open,	Pointer to the open function
	cpg_close,	Pointer to the close function
	no_dev_read,	Pointer to the driver read function
	no_dev_write,	Pointer to the driver write function
	cpg_control,	Pointer to the driver control function
	cpg_get_version	Pointer to the function to get version info
g_gpio_driver	In the sample program, the following values are implemented:	GPIO driver definition data:
	"GPIO Driver",	Driver name
	gpio_open,	Pointer to the open function
	gpio_close,	Pointer to the close function
	no_dev_read,	Pointer to the driver read function
	no_dev_write,	Pointer to the driver write function
	gpio_control,	Pointer to the driver control function
	gpio_get_version	Pointer to the function to get version info
g_ostm_driver	In the sample program, the following values are implemented:	OSTM driver definition data:
	"OSTM Driver",	Driver name
	ostm_open,	Pointer to the open function
	ostm_close,	Pointer to the close function
	ostm_read,	Pointer to the driver read function
	no_dev_write,	Pointer to the driver write function
	ostm_control,	Pointer to the driver control function
	ostm_get_version	Pointer to the function to get version info
g_scifa_driver	In the sample program, the following values are implemented:	SCIFA driver definition data:
	"SCIFA Driver",	Driver name
	scifa_open,	Pointer to the open function
	scifa_close,	Pointer to the close function
	scifa_read,	Pointer to the driver read function
	scifa_write,	Pointer to the driver write function
	ostm_control,	Pointer to the driver control function
	ostm_get_version	Pointer to the function to get version info

Table 5.13 shows the direct related macro definition.

Table 5.13 The direct related Macro Definition

Macro Definition	Value	Description
DRV_SUCCESS	(0)	Successfully terminated
DRV_ERROR	(-1)	Abnormal termination

5.5.2 CPG related constants

Table 5.14 shows the direct related macro definition.

Table 5.14 CPG Configuration Data

Constants	Value	Description
const float64_t s_sc_cpg_xtal_frequency_khz_config	24000.0	XTAL frequency (kHz)
static const st_r_drv_cpg_set_main_t s_sc_cpg_main_clock_config[]	{ {CPG_SUB_CLOCK_ICLK, 528000.0}, {CPG_SUB_CLOCK_BCKL, 132000.0}, {CPG_SUB_CLOCK_P1CLK, 66000.0} };	Frequency dividing configuration for ICLK, BCLK and P1CKL
static const st_r_drv_cpg_set_src_t s_sc_cpg_sub_clock_src_config[]	{ {CPG_SUB_CLOCK_CKIO, CPG_SUB_CLOCK_BCLK_IN}, {CPG_SUB_CLOCK_OCTAMEM, CPG_SUB_CLOCK_GCLK_IN}, {CPG_SUB_CLOCK_HYPERBUS, CPG_SUB_CLOCK_GCLK_IN}, {CPG_SUB_CLOCK_SPICLK, CPG_SUB_CLOCK_BCLK_IN} };	Frequency dividing configuration for CKIO and SCLK
static const st_r_drv_cpg_ext_clk_t s_sc_cpg_ext_clk_config	{CPG_CKIO_INVALID_UNSTABLE-_OFF_HIZ}	CKIO pin state configuration

5.5.3 MMU related constants

Table 5.15, Table 5.16 and Table 5.17 show the MMU related macro definition.

Table 5.15 MMU configuration data (const static st_r_mmu_sc_config_t) (1/3)

Constants	Value	Description
MMU_SC_TABLE[]		st_r_mmu_sc_config_t has the following members: virtual address: virtual address physical address: physical address page_count: Number of page page_attribute: Attribute of page
	In the sample program, the following values are specified: {	
	{0x00000000uL, 0x00000000uL, 4096, MMU_ATTR_UNUSED | MMU_ATTR_DMAIN(15) },	default setting
	{0x00000000uL, 0x00000000uL, 64, MMU_ATTR_STRONGLY_NS | MMU_ATTR_DMAIN(15) },	CS0 space: Strongly-ordered (NS)
	{0x04000000uL, 0x04000000uL, 64, MMU_ATTR_STRONGLY_NS | MMU_ATTR_DMAIN(15) },	CS1 space: Strongly-ordered (NS)
	{0x08000000uL, 0x08000000uL, 64, MMU_ATTR_STRONGLY_NS | MMU_ATTR_DOMAIN(15)},	CS2 space: Strongly-ordered (NS)
	{0x0C000000uL, 0x0C000000uL, 64, MMU_ATTR_NORMAL_L1L2CACHE | MMU_ATTR_DOMAIN(15)},	CS3 (SDRAM): Normal, L1/L2 Cacheable
	{0x10000000uL, 0x10000000uL, 64, MMU_ATTR_STRONGLY_NS | MMU_ATTR_DOMAIN(15)},	CS4 space: Strongly-ordered (NS)
	{0x14000000uL, 0x14000000uL, 64, MMU_ATTR_STRONGLY_NS | MMU_ATTR_DOMAIN(15)},	CS5 space: Strongly-ordered (NS)
	{0x18000000uL, 0x18000000uL, 112, MMU_ATTR_RESERVED | MMU_ATTR_DOMAIN(15)},	Reserved
	{0x1F000000uL, 0x1F000000uL, 16, MMU_ATTR_STRONGLY | MMU_ATTR_DOMAIN(15)},	Peripheral I/O: Strongly-ordered (NS)
	{0x20000000uL, 0x20000000uL, 256, MMU_ATTR_NORMAL_L1L2CACHE | MMU_ATTR_DOMAIN(15)},	SPI Multi I/O bus area: Normal, L1/L2 Cacheable
	{0x30000000uL, 0x30000000uL, 256, MMU_ATTR_NORMAL_L1L2CACHE | MMU_ATTR_DOMAIN(15)},	Hyper Flash area: Normal, L1/L2 Cacheable
	{0x40000000uL, 0x40000000uL, 256, MMU_ATTR_NORMAL_L1L2CACHE | MMU_ATTR_DOMAIN(15)},	Hyper RAM area: Normal, L1/L2 Cacheable
	{0x50000000uL, 0x50000000uL, 256, MMU_ATTR_NORMAL_L1L2CACHE | MMU_ATTR_DOMAIN(15)},	Octa Flash area: Normal L1/L2 Cacheable

Table 5.16 MMU configuration data (const static st_r_mmu_sc_config_t) (2/3)

Constants	Value	Description
MMU_SC_TABLE[]		st_r_mmu_sc_config_t has the following members: virtual address: virtual address physical address: physical address page_count: Number of page page_attribute: Attribute of page
	In the sample program, the following values are specified: {	
	0x60000000uL, 0x60000000uL, 256, MMU_ATTR_NORMAL_L1L2CACHE | MMU_ATTR_DOMAIN(15)},	Octa RAM area: Normal, L1/L2 Cacheable
	{0x70000000uL, 0x20000000uL, 256, MMU_ATTR_STRONGLY_NS_EXECUTABLE | MMU_ATTR_DOMAIN(15)},	SPI Multi I/O area: Normal, Non-cacheable, Strongly-ordered (NS)
	{0x80000000uL, 0x80000000uL, 4, MMU_ATTR_NORMAL_L1CACHE | MMU_ATTR_DOMAIN(15)},	Internal RAM area: Normal, L1 Cacheable
	{0x80400000uL, 0x80400000uL, 28, MMU_ATTR_RESERVED | MMU_ATTR_DOMAIN(15)},	Reserved
	{0x82000000uL, 0x80000000uL, 4, MMU_ATTR_NORMAL | MMU_ATTR_DOMAIN(15)},	Internal RAM area: Normal, Non-cacheable
	{0x82400000uL, 0x82400000uL, 92, MMU_ATTR_RESERVED | MMU_ATTR_DOMAIN(15)},	Reserved
	{0x88000000uL, 0x00000000uL, 64, MMU_ATTR_STRONGLY_NS | MMU_ATTR_DOMAIN(15)},	CS0 space: Strongly-ordered (NS)
	{0x8C000000uL, 0x04000000uL, 64, MMU_ATTR_STRONGLY_NS | MMU_ATTR_DOMAIN(15)},	CS1 space: Strongly-ordered (NS)
	{0x90000000uL, 0x08000000uL, 64, MMU_ATTR_STRONGLY_NS | MMU_ATTR_DOMAIN(15)},	CS2 space: Strongly-ordered (NS)
	{0x94000000uL, 0x0C000000uL, 64, MMU_ATTR_STRONGLY_NS | MMU_ATTR_DOMAIN(15)},	CS3 (SDRAM): Strongly-ordered (NS)
	{0x98000000uL, 0x10000000uL, 64, MMU_ATTR_STRONGLY_NS | MMU_ATTR_DOMAIN(15)},	CS4 space: Strongly-ordered (NS)
	{0x9C000000uL, 0x14000000uL, 64, MMU_ATTR_STRONGLY_NS | MMU_ATTR_DOMAIN(15)},	CS5 space: Strongly-ordered (NS)

Table 5.17 MMU configuration data (const static st_r_mmu_sc_config_t) (3/3)

Constants	Value	Description
MMU_SC_TABLE[]		st_r_mmu_sc_config_t has the following member: virtual address: virtual address physical address: physical address page_count: Number of page page_attribute: Attribute of page
	In the sample program, the following values are specified:	
	{0xA0000000uL, 0x30000000uL, 256, MMU_ATTR_STRONGLY_NS_EXECUTABLE | MMU_ATTR_DOMAIN(15)},	Hyper Flash area: Non-cacheable, Strongly-ordered (NS)
	{0xB0000000uL, 0x40000000uL, 256, MMU_ATTR_STRONGLY_NS_EXECUTABLE | MMU_ATTR_DOMAIN(15)},	Hyper RAM area: Non-cacheable, Normal
	{0xC0000000uL, 0x50000000uL, 256, MMU_ATTR_STRONGLY_NS_EXECUTABLE | MMU_ATTR_DOMAIN(15)},	Octa Flash area: Non-cacheable, Strongly-ordered (NS)
	{0xD0000000uL, 0x60000000uL, 256, MMU_ATTR_STRONGLY_NS_EXECUTABLE | MMU_ATTR_DOMAIN(15)},	Octa RAM area: Non-cacheable, Normal
	{0xE0000000uL, 0xE0000000uL, 128, MMU_ATTR_RESERVED | MMU_ATTR_DOMAIN(15)},	Reserved

	{0xE8000000uL, 0xE8000000uL, 384, MMU_ATTR_STRONGLY | MMU_ATTR_DOMAIN(15)}
	}
		Peripheral I/O: Strongly-ordered

5.5.4 INTC related constants

Table 5.18 shows the macro definition denoting interrupt ID.

Table 5.18 Macro definition for interrupt ID

Definition	Value	Description
INTC_ID_OSTM_OSTMI0	((intc_intid_t) 88)	OSTMI0 interrupt
INTC_ID_SCIFA_ERI4BRI4	((intc_intid_t) 321)	SCIFA4 ERI4/BRI4 interrupt
INTC_ID_SCIFA_RXI4	((intc_intid_t) 322)	SCIFA4 RXI4 interrupt
INTC_ID_SCIFA_TXI4	((intc_intid_t) 323)	SCIFA4 TXI4 interrupt

5.5.5 OSTM related constants

Table 5.19 shows the constants in OSTM configuration data (static const st_r_drv_ostm_sc_config_t), which is used in the sample program.

Table 5.19 OSTM Configuration Data (static const st_r_drv_ostm_sc_config_t)

Constants	Value	Description
OSTM_SC_TABLE[0]	0,	Channel number
	{	
	OSTM_MODE_INTERVAL,	Timer mode
	OSTM_TIME_MS,	Unit of counter value to be specified
	500,	Value for comparison
	OSTM_START_INTERRUPT_OFF,	Disable interrupt when booting timer
	INTC_ENABLE,	Enable interrupt
	3,	Interrupt priority level
	Sample_LED_Blink	Interrupt handler
	}	

5.5.6 SCIFA related constants

Table 5.20 shows the constants in SCIFA configuration data (static const st_r_drv_scifa_sc_config_t), which is used in the sample program.

Table 5.20 SCIFA Configuration Data (static st_r_drv_scifa_sc_config_t)

Constants	Value	Description
SCIFA_SC_TABLE[]	4,	Channel number
	{	
	SCIFA_MODE_ASYNC,	Asynchronous mode
	115200,	Baud Rate
	SCIFA_CLK_SRC_INT_SCK_IN,	Internal Clock, SCK port is input
	SCIFA_CLK_16X,	Not Used
	SCIFA_DATA_8BIT,	Data Length 8 bits
	SCIFA_PARITY_OFF,	No Parity check
	SCIFA_EVEN_PARITY,	Even Parity (if enabled)
	SCIFA_STOPBITS_1,	1 stop bit
	SCIFA_NOISE_CANCEL_DISABLE,	Noise filter disabled
	SCIFA_LSB_FIRST,	Least Significant Bit first
	SCIFA_LOOPBACK_DISABLE,	Not in Loopback mode
	SCIFA_MODEM_CONTROL_DISABLE,	Modem control disabled
	SCIFA_RTS_ACTIVE_TRIGGER_15,	RTS trigger level
	15,	Transmit FIFO Data trigger number
	1,	Receive FIFO Data trigger number
	SCIFA_SPTR_INIT_HIGH,	Serial port break data select
	SCIFA_SPTR_INIT_HIGH,	SCK pin data select
	SCIFA_SPTR_INIT_HIGH,	CTS pin data select
	SCIFA_SPTR_INIT_HIGH,	RTS pin data select
	SCIFA_TX_INTERRUPT_MODE,	Interrupt driven transmission
	SCIFA_RX_INTERRUPT_MODE,	Interrupt driven reception
	0,	Break interrupt priority
	30,	Rx interrupt priority
	30,	Tx interrupt priority
	0,	Tx end interrupt priority
	NULL,	DMA write callback function pointer
	NULL,	DMA read callback function pointer
	NULL,	DMA module identifier for transmit
	NULL,	DMA module identifier for receive
	},	
	{	
	&GPIO_SC_TABLE_scifa4[0],	Pointer to Pin definition table.
	sizeof(GPIO_SC_TABLE_scifa4)/sizeof(st_r_drv_gpio_sc_config_t),	Number of elements in pin definition table.
	}	

5.5.7 GPIO related constants

Table 5.21 shows enumeration value declared as the control code (e_ctrl_code_gpio_t).

Table 5.21 Control code (e_ctrl_code_gpio_t)

Constants	Value	Description
e_ctrl_code_gpio_t	CTL_GPIO_SET_CONFIGURATION	Set pin configuration GPIO control function
	CTL_GPIO_GET_CONFIGURATION	Get pin configuration GPIO control function
	CTL_GPIO_INIT_BY_PIN_LIST	Configure specified pins by list control function
	CTL_GPIO_CLEAR_BY_PIN_LIST	Clear (deallocate) specified pins by list control function
	CTL_GPIO_INIT_BY_TABLE	Configure pins by table control function
	CTL_GPIO_CLEAR_BY_TABLE	Clear (deallocate) pins by table control function
	CTL_GPIO_PORT_WRITE	Write port data control function
	CTL_GPIO_PORT_READ	Read port data control function
	CTL_GPIO_PIN_WRITE	Write pin data control function
	CTL_GPIO_PIN_READ	Read pin data control function

Table 5.22 shows the constants in GPIO (initialization) configuration data (const st_r_drv_gpio_sc_config_t)

Table 5.22 GPIO (initialization) configuration data (const st_r_drv_gpio_sc_config_t)

Constants	Value	Description
GPIO_SC_TABLE_INIT[]	Empty	

Table 5.23 shows the constants in GPIO (MANUAL) configuration data (const st_r_drv_gpio_sc_config_t)

Table 5.23 GPIO (MANUAL) configuration data (const st_r_drv_gpio_sc_config_t)

Constants	Value	Description
GPIO_SC_TABLE_-MANUAL		pin: pin number function: function settings tint: pin interrupts current: pin driving ability settings
	In the sample code, the following value is specified: {	
	{GPIO_PORT_6_PIN_0, {GPIO_FUNC_OUT_LOW, GPIO_TINT_DISABLE, GPIO_CURRENT_4mA},	P6_0 General-purpose output port (default: LOW) Disable pin interrupts Configure 4mA as pin driving ability
	{GPIO_PORT_C_PIN_1, {GPIO_FUNC_OUT_HIGH, GPIO_TINT_DISABLE, GPIO_CURRENT_4mA} }	PC_1 General-purpose output port (default: HIGH) Disable pin interrupts Configure 4mA as pin driving ability

Table 5.24 shows the constants in GPIO (SCIFA) configuration data (const st_r_drv_gpio_sc_config_t)

Table 5.24 GPIO (SCIFA) configuration data (const st_r_drv_gpio_sc_config_t)

Constants	Value	Description
GPIO_SC_TABLE_scifa4		pin: pin number function: function settings tint: pin interrupts current: pin driving ability settings
	In the sample code, the following value is specified: {	
	{GPIO_PORT_9_PIN_0, {GPIO_FUNC_PERIPHERAL4, GPIO_TINT_DISABLE, GPIO_CURRENT_4mA},	P9_0 Peripheral function (TxD4) Disable pin interrupts Configure 4mA as pin driving ability
	{GPIO_PORT_9_PIN_1, {GPIO_FUNC_PERIPHERAL4, GPIO_TINT_DISABLE, GPIO_CURRENT_4mA} }	P9_1 Peripheral function (RxD4) Disable pin interrupts Configure 4mA as pin driving ability

5.6 Variables

5.6.1 direct related variables

Table 5.25 shows the specification of the direct related static filestream variable.

Table 5.25 static filestream variables (direct related)

Type	Variable	Description	Used in
static st_stream_t	gs_filestream [IOSTREAM]	Table storing device name Device name is searched from the tail of this list. This table is generated by SC.	get_first_free_direct_handle get_first_free_call_stdio_handle check_open close_all get_device_stream_from_handle check_vaild_handle direct_open direct_close direct_read direct_write direct_control low_open low_seek R_DEVLINK_Init

Table 5.26 shows the specification of the direct related static mount table variable.

Table 5.26 static mount table variables (direct related)

Type	Variable	Description	Used in
static st_mount_table_t	gs_mount_table[]	Drivers corresponding to the configuration name registered by SC should be listed first. In the sample code, CPG, GPIO, OSTM0~2, SCIFA0~4 devices are listed. { {“cpg”, (st_r_driver_t *) &g_cpg_driver, R_SC0}, {“gpio”, (st_r_driver_t *) &g_gpio_driver, R_SC0}, {“ostm0”, (st_r_driver_t *) &g_ostm_driver, R_SC0}, {“ostm1”, (st_r_driver_t *) &g_ostm_driver, R_SC1}, {“ostm2”, (st_r_driver_t *) &g_ostm_driver, R_SC2}, {“scifa0”, (st_r_driver_t *) &g_scifa_driver, R_SC0}, {“scifa1”, (st_r_driver_t *) &g_scifa_driver, R_SC1}, {“scifa 2”, (st_r_driver_t *) &g_scifa_driver, R_SC2}, {“scifa 3”, (st_r_driver_t *) &g_scifa_driver, R_SC3}, {“scifa 4”, (st_r_driver_t *) &g_scifa_driver, R_SC4}}	dev_get_pointer

5.6.2 CPG related variables

Table 5.27 shows the CPG related static variable.

Table 5.27 static variables (CPG related)

Type	Variable	Description	Used in
static bool_t	gs_drv_cpg_is_initialized	Denotes if driver has opened true: opened, false: unopened	cpg_open

5.6.3 MMU related variables

Table 5.28 shows the MMU related static variable.

Table 5.28 static variables (MMU related)

Type	Variable	Description	Used in
-	__mmu_page_table_base	The starting address of section region specified in translation table which is placed in on-chip large capacity internal RAM	R_MMU_Init

5.6.4 INTC related variables

Table 5.29 shows the INTC related global variable.

Table 5.29 global variables (INTC related)

Type	Variable	Description	Used in
void	(*g_intc_-func_table[]) (uint32_t int_sense)	Interrupt handler registration table: This table is for registering interrupt handler for each interrupt ID by the function R_INTC_RegistIntFunc. When INTC interrupt is fired, the interrupt handler corresponding to the interrupt ID in this table is called by the function INTC_Handler_Interrupt.	R_INTC_RegistIntFunc INTC_Handler_Interrupt

5.6.5 OSTM related variables

Table 5.30 shows the OSTM related static variable.

Table 5.30 static variables (OSTM related)

Type	Variable	Description	Used in
static uint32_t	main_led_flg	The flag for notifying the OSTM interrupt LSB is inverted each time OSTM interrupt is fired. When setting this flag to MAIN_PRV_LED_ON, LED is turned on, while LED is turned off when setting this flag to MAIN_PRV_LED_OFF.	main Sample_LED_Blink

Table 5.31 shows the OSTM related global variable.

Table 5.31 global variables (OSTM related)

Type	Variable	Description	Used in
bool_t	gs_r_drv_-ostm_open	Denotes if driver has opened true: opened, false: unopened	ostm_open ostm_close

5.7 Functions

The sample code consists of the following functions:

- API function for using peripheral functions
- User Defined Function depending on user system for which user needs to write (functions called by an API function)
- Sample Function implemented for getting sample program to be worked as a reference

In the sample code, the IO register-write and register-read function is used when accessing peripheral IO registers in bit unit.

Table 5.32, list Sample Function, API Function and User Defined Function, respectively.

Table 5.32 Sample Function

Function Name	Description
reset_handler	Reset handler (written in assembly language)
resetprg	Initial setup of INTC and L1/L2 Cache
main	Main processing
Sample_main	Invoke the sample code for peripheral functions in accordance with the command input via terminal software
INITSCT	Initial setup of memory section (written in assembly language)
R_SC_HardwareSetup	Initial setup of peripheral functions
__libc_init_array	C++ constructor processing
__set_vbar	Vector Base Address Register (VBAR) setup
irq_handler	IRQ handler processing (written in assembly language)
INTC_Handler_Interrput	INTC interrupt handler processing
fiq_handler	FIQ handler processing (written in assembly language)
Sample_LED_Blink	OSTM0 interrupt processing
direct_open	Open peripheral I/O
direct_write	Write to peripheral I/O
direct_read	Read from peripheral I/O
direct_control	Control peripheral I/O
direct_close	Close peripheral I/O
direct_get_version	Get version number of device driver for peripheral I/O
r_disable_rtc	Processing for unused RTC channel
r_disable_usb	Processing for unused USB channel

Table 5.33 API Function (1/2)

Function Name	Description
R_MMU_Init	Initial setup of MMU
R_MMU_Disable	Disable MMU
R_MMU_WriteTbl	Configure MMU translation table
R_CACHE_L1Init	Initialize L1 Cache
R_CACHE_L1InstEnable	Enable L1 Instruction Cache
R_CACHE_L1InstDisable	Disable L1 Instruction Cache
R_CACHE_L1InstInvalidAll	Invalidate the whole L1 Instruction Cache
R_CACHE_L1DataEnable	Enable L1 Data Cache
R_CACHE_L1DataDisable	Disable L1 Data Cache
R_CACHE_L1DataInvalidAll	Invalidate the whole L1 Data Cache
R_CACHE_L1DataCleanAll	Clean the whole L1 Data Cache
R_CACHE_L1DataCleanInvalidAll	Clean and invalidate the whole L1 Data Cache
R_CACHE_L1DataInvalidLine	Invalidate L1 Data Cache per line (32-bytes unit)
R_CACHE_L1DataCleanLine	Clean L1 Data Cache per line (32-bytes unit)
R_CACHE_L1DataCleanInvalidLine	Clean and invalidate L1 Data Cache per line (32-bytes unit)
R_CACHE_L1BtacEnable	Enable Branch Predictor
R_CACHE_L1BtacDisable	Disable Branch Predictor
R_CACHE_L1BtacInvalidate	Invalidate Branch Predictor
R_CACHE_L1PrefetchEnable	Enable L1 Data Cache prefetching
R_CACHE_L1PrefetchDisable	Disable L1 Data Cache prefetching
R_CACHE_L2Init	Initialize L2 Cache
R_CACHE_L2CacheEnable	Enable L2 Cache
R_CACHE_L2CacheDisable	Disable L2 Cache
R_CACHE_L2PrefetchEnable	Enable L2 Cache prefetching
R_CACHE_L2PrefetchDisable	Disable L2 Cache prefetching
R_CACHE_L2InvalidAll	Invalidate the whole L2 Cache
R_CACHE_L2CleanAll	Clean the whole L2 Cache
R_CACHE_L2CleanInvalidAll	Clean and invalidate the whole L2 Cache
R_INTC_Init	Initialize interrupt controller
R_INTC_Enable	Enable interrupt controller
R_INTC_Disable	Disable interrupt controller
R_INTC_SetPriority	Set up interrupt priority level
R_INTC_SetMaskLevel	Set up interrupt mask level
R_INTC_GetMaskLevel	Get interrupt mask level
R_INTC_RegistIntFunc	Register interrupt handlers
R_OSTM_Init	Initialize OSTM
R_STB_StartModule	Cancel module standby mode
R_STB_StopModule	Translate to module standby mode

Table 5.34 API Function (2/2)

Function Name	Description
R_CPG_SetXtalClock	Calculate Iϕ, Bϕ and Pϕ based on EXTAL frequency
R_CPG_SetSubClockDividers	Set up the division rate of Iϕ, Bϕ and Pϕ
R_CPG_SetSubClockSource	Set up the clock source of CKIO, OM_SCLK, HM_CK/HM_CK#, QSPI0_SPCLK/QSPI1_SPCLK
R_CPG_ConfigExtPinClock	Configure CKIO pin state
R_GPIO_PortWrite	Set up the output value of specified port
R_GPIO_PortRead	Read out the input value of specified port
R_GPIO_PinWrite	Set up the output value of specified pin
R_GPIO_PinRead	Read out the input value of specified pin
R_GPIO_HWInitialise	Initialize all the GPIOs at once
R_GPIO_PinSetConfiguration	Set up the specified pin function
R_GPIO_InitByPinList	Set up GPIOs according to the list of pins created beforehand
R_GPIO_InitByTable	Set up GPIOs according to the pin table
RZA_IO_RegWrite_8	IO register write function (for 8-bit I/O register)
RZA_IO_RegWrite_16	IO register write function (for 16-bit I/O register)
RZA_IO_RegWrite_32	IO register write function (for 32-bit I/O register)
RZA_IO_RegRead_8	IO register read function (for 8-bit I/O register)
RZA_IO_RegRead_16	IO register read function (for 16-bit I/O register)
RZA_IO_RegRead_32	IO register read function (for 32-bit I/O register)

Table 5.35 User Defined Function

Function Name	Description
Userdef_INTC_Pre_Interrupt	Define the processing carried out before the INTC interrupt handler is invoked
Userdef_INTC_Post_Interrupt	Define the processing carried out after the INTC interrupt handler has terminated
Userdef_INTC_UnregisteredID	Define the processing when an interrupt is fired, but its handler hasn’t been registered yet
Userdef_INTC_UndefId	Define the processing when an interrupt is fired, but its ID is undefined
Userdef_INTC_NMI_Handler	Define the NMI interrupt handler processing

5.8 Function Specification

This section describes the specification of function in the sample code.

reset_handler

Overview	Reset handler
Syntax	reset_handler FUNCTION {}
Description	This is the assembler function invoked just after reset has completed. This function carries out the initial setup of the stack pointer, the MMU, and minimal peripheral setup. Finally, the resetprg() function is called.
Parameters	None
Return value	None

INITSCT		
Overview	Initialize memory section	
Syntax	INITSCT FUNCTION{}	
Description	Initialize DATA and BSS section in accordance with the specified initialization table	
Parameters	r0	Pointer to initialization table for DATA section
	R1	Pointer to initialization table for BSS section
Return value	None	

R_SC_HardwareSetup	
Overview	Initialize peripheral functions
Syntax	void R_SC_HardwareSetup(void)
Description	Initialize peripheral functions This function initializes peripheral functions which need to be initialized before initial setup of section initialization. In this sample code, this function is implemented as an empty function.
Parameters	None
Return value	None
Note	DATA and BSS section shouldn't be initialized at the time this function is called. Therefore please don't use the variables assigned to DATA and/or BSS section in this function and/or functions which are called from this function.

resetprg	
Overview	Initialize INTC and L1/L2 cache
Syntax	Void resetprg(void)
Description	Initialize INTC and the L1/L2 cache and then call the main() function.
Parameters	None
Return value	None

main	
Overview	Main processing
Syntax	int_t main(void)
Description	This function displays information from the sample code to the terminal running on the host PC connected to the RZ/A2M CPU board via its serial interface and initializes the GPIO pin driving the on-board LED. Also, OSTM channel 0 is initialized and its counter configured so that the interrupt is fired every 500ms, and then it is enabled.
Parameters	None
Return value	0

Sample_Main	
Overview	Main processing for sample code
Syntax	void Sample_Main(void)
Description	Invoke the sample code of RZ/A2Ms peripheral functions corresponding to the command input from the terminal running on the host PC connected with RZ/A2M CPU board via its serial interface.
Parameters	None
Return value	None

R_MMU_Init	
Overview	Initialize MMU translation table
Syntax	void Sample_Main(void)
Description	Invoke the sample code of RZ/A2Ms peripheral functions corresponding to the command input from the terminal running on the host PC connected with RZ/A2M CPU board via its serial interface.
Parameters	None
Return value	None

R_MMU_Disable	
Overview	Disable MMU
Syntax	void R_MMU_Disable(void)
Description	Disable MMU
Parameters	None
Return value	None

R_MMU_WriteTbl		
Overview	Configure MMU translation table	
Syntax	void R_MMU_WriteTbl (uint32_t vaddress, uint32_t paddress, uint32_t size, uint32_t entry)	
Description	This function assigns physical address to the first level MMU translation table entry for the virtual address space specified by the parameter vaddress, and set up its attribute. Table 5.5 and Table 5.6 show the first level descriptor that can be specified for translation table.	
Parameters	uint32_t vaddress	Starting address of virtual address area to be specified (Only b31-b20 is valid)
	uint32_t paddress	Starting address of physical address area to be assigned (Only b31-b20 is valid)
	uint32_t size	Number of entry to be specified (1~4096)
	uint32_t entry	Attribute to be specified (Only b19-b0 is valid)
Return value	mmu_err_t MMU_Success: Successfully terminated MMU_ERR_OVERFLOW: Size has overflowed	
Notes	This function must be invoked after R_MMU_Init().	

6. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

7. Reference Documents

User's Manual: Hardware

RZ/A2M Group User's Manual: Hardware
The latest version can be downloaded from the Renesas Electronics website.

RTK7921053C00000BE (RZ/A2M CPU board) User's Manual
The latest version can be downloaded from the Renesas Electronics website.

RTK79210XXB00000BE (RZ/A2M SUB board) 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 Cortex^{TM}-A9 Technical Reference Manual Revision: r4p1
The latest version can be downloaded from the ARM website.

ARM Generic Interrupt Controller Architecture Specification - Architecture version 2.0
The latest version can be downloaded from the ARM website.

ARM CoreLink^{TM} Level 2 Cache Controller L2C-310 Technical Reference Manual Revision: r3p3
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

Integrated development environment e^{2} studio User’s Manual can be downloaded from the Renesas Electronics website.
The latest version can be downloaded from the Renesas Electronics website.

Revision History

Rev.	Date	Description	
		Page	**Summary**
1.00	Oct.04.18	—	First edition issued
1.01	Jan.16.19	various	Updated referenced application note, e^2 studio version, tables in section 5
1.10	May.14.19	7	Removed obsolete compiler option from operating conditions.
1.11	May.29.20	27	Reformatting caption for Figure 5.12.
1.12	Sep.30.20	7	Modified Compile Options.