RZ/T2M Group

Example of separating loader program and application program projects

Introduction

This application note explains a sample application separating the application into a loader program and an application program.

The major features of the sample program are listed below.

- The program supports two operating modes of the device: xSPI0 boot mode (x1 boot serial flash) version and 16-bit bus boot mode (NOR flash) version.
- The sample application consists of two separated projects, the loader program and the application program.
- The loader program is a program for copying the application program from external flash to internal RAM or external RAM. This is done according to the loader table information (source address, destination address, size) defined in the loader program.
- The application program is ,copied and started by the loader program. It performs initial settings and let the LEDs blink.

Target Devices

RZ/T2M Group

When applying the sample program covered in this application note to another microcomputer, modify the program according to the specifications for the target microcomputer and conduct an extensive evaluation and testing of the modified program.

Contents

1. Specifications

1.1 Operating Environment

The sample program covered in this application note is for the environment below.

Table 1.1 Operating Environment

Item	Description
Microcomputer	RZ/T2M Group (R9A07G075M28GBG)
Operating Frequency	CPU core0: 800MHz (Arm® Cortex®-R52) CPU core1: 800MHz (Arm® Cortex®-R52)*1
Operating Voltage	3.3V / 1.8V / 1.1V
Integrated Development Environment	• Embedded Workbench® for Arm Version 9.50.1 from IAR systems • e^2 studio 2024-01.1 (24.1.1) (R20240125-1623) from Renesas
Operating mode	• xSPI0 boot mode (x1 serial flash) • 16-bit bus boot mode (NOR flash)
Board	Renesas Starter Kit+ for RZ/T2M
Flexible Software Package (FSP)	Version 2.0.0 (RZ/T2 FSP)

Note 1. When using CPU core1, refer to "5.3 How to Debug CPU1 Program".

1.2 File Structure

The details of the file structure and contents of this package are show below.

```
RZT2M_loader_application
├─r01an6729jj0120-rzt2m.pdf
├─r01an6729ej0120-rzt2m.pdf
├─iccarm: for EWARM
│  ├─16bitbusboot: sample program for NOR flash
│  │  └─Loader_application_projects
│  │     ├─application: project for application program
│  │     ├─loader: project for loader program
│  │     ├─cpu1: project for CPU1 program
│  │     └─RZT2M_bsp_16bitbusboot_app_loader.eww: EWARM workspace
│  └─xspi0bootx1: sample program for SPI flash
│     └─Loader_application_projects
│        ├─application: project for application program
│        ├─loader: project for loader program
│        ├─cpu1: project for CPU1 program
│        └─RZT2M_bsp_xspi0bootx1_app_loader.eww: EWARM workspace
└─gcc : for e2 studio
   ├─16bitbusboot: sample program for NOR flash
   │  └─Loader_application_projects.zip
   │     ├─RZT2M_bsp_16bitbusboot_app: project for application program
   │     ├─RZT2M_bsp_16bitbusboot_loader: project for loader program
   │     └─RZT2M_cpu1: project for CPU1 program
   └─xspi0bootx1: sample program for SPI flash
      └─Loader_application_projects.zip
         ├─RZT2M_bsp_xspi0bootx1_app: project for application program
         ├─RZT2M_bsp_xspi0bootx1_loader: project for loader program
         └─RZT2M_cpu1: project for CPU1 program
```

The files of the package are separated to EWARM and e2 studio environment at first level, and to NOR flash and SPI flash at second level.

Each of the six resulting sample application consists of two projects – one project for the loader program , one project for the application program and one project for the CPU1 program.

For the usage procedures of sample program in each development environments, see Appendix Supplementary Notes on Development Environments.

1.3 Switch and Jumper Settings

The switch and jumper settings required to run the sample program are shown below. For details on each setting, see the Renesas Starter Kit+ for RZ/T2M User's Manual.

Table 1.2 Switch settings

Project	SW4-1	SW4-2	SW4-3	SW4-4	SW4-5	SW6-1
16-bit bus boot mode	ON	OFF	ON	ON	OFF	ON
xSPI0 boot mode	ON	ON	ON	ON	OFF	-

Table 1.3 Jumper settings

Project	CN8	CN17
16-bit bus boot mode	-	Short 1-2
xSPI0 boot mode	Short 2-3	-

2. Hardware

2.1 Peripheral Functions

Table 2.1 lists the peripheral functions to be used and their applications.

Table 2.1 Peripheral functions and applications

Peripheral function	Application
Clock generation circuit (CGC)	Used as a CPU clock and each peripheral module clock
Interrupt controller (ICU)	Used for software interrupts (INTCPU0)
Bus state controller (BSC)	Used to attach NOR flash memory to CS0 space and SDRAM to CS3 space
Expanded serial peripheral interface (xSPI)	Used to attach Serial flash memory to external address space xSPI0
General purpose I/O ports	Used to control pins to light LEDs on and off

See the RZ/T2M Group User’s Manual: Hardware for basic descriptions.

2.2 Pins

Table 2.2 lists pins to be used and their functions.

Table 2.2 Pins and Functions

Pin Name	Input/Output	Function
A1 to A25	Output	Address signal output to NOR flash memory and SDRAM
D0 to D15	Input/Output	Data signal input and output to NOR flash memory and SDRAM
CS0#	Output	Device selection signal output to NOR flash memory attached to CS0 space
CS3#	Output	Device selection signal output to SDRAM attached to CS3 space
RAS#	Output	RAS# control signal output to SDRAM
CAS#	Output	CAS# control signal output to SDRAM
RD/WR#	Output	Read control signal or write control signal output to SDRAM
CKE	Output	Clock enabling control signal output to SDRAM
RD#	Output	Strobe signal output indicating reading
WE0#/DQMLL	Output	Write strobe signal output to D15 to D8
WE1#/DQMLU	Output	Write strobe signal output to D7 to D0
XSPI0_CKP	Output	Clock output
XSPI0_CS0	Output	Device selection signal output to QSPI flash memory attached to CS0 space
XSPI0_RESET0	Output	Master reset status output
XSPI0_IO0 ~ XSPI0_IO3	Input/Output	Data input / output
MD0	Input	Operating mode selection: • MD0 = “L”, MD1 = “L”, MD2 = “L” (xSPI0 boot mode) • MD0 = “L”, MD1 = “H”, MD2 = “L” (16-bit bus boot mode)
MD1	Input	
MD2	Input	
P19_6	Output	Lighting LED0 on and off
P19_4	Output	Lighting LED1 on and off
P20_0	Output	Lighting LED2 on and off
P23_4	Output	Lighting LED3 on and off

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

3. Software

This section explains the case of EWARM (from IAR systems) unless otherwise stated.

In this document, the program included in the loader project is called loader program, and the program included in the application project is called application program. Loader program and application program each have startup processing section and main processing section.

3.1 Operation Overview

After the reset is released, the loader program for each operating mode (16-bit bus boot/xSPI0 boot) stored on the external flash memory (NOR flash/Serial flash) is copied to the internal RAM (BTCM).

After boot processing, the loader program is executed. The loader program copies the application program from external flash memory (NOR flash/Serial flash) to RAM (System SRAM). As final step of the loader program the entry point of the copied application program is called. After executing the loader program, the execution of the application program starts.

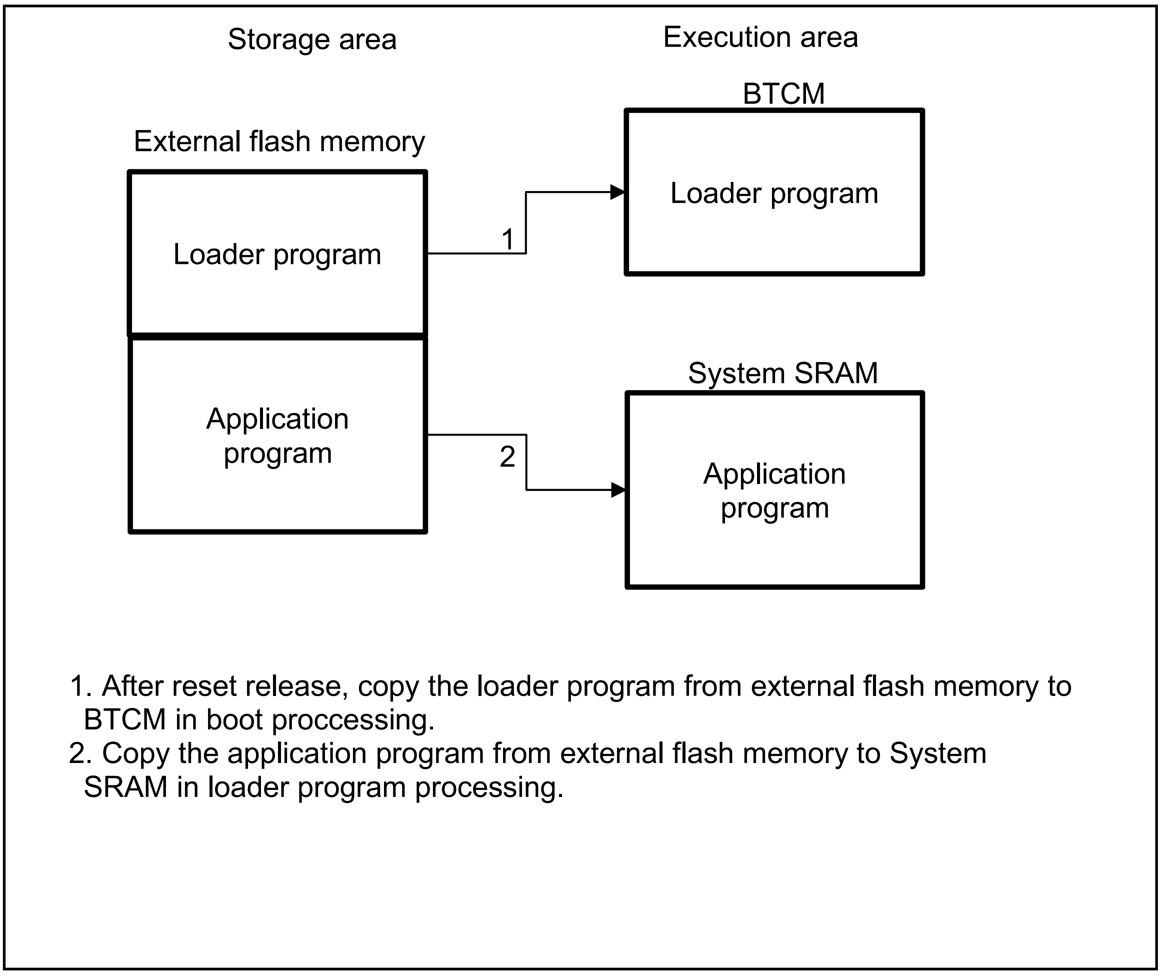


Figure 3.1 Operation overview

3.1.1 Loader Program

The loader program performs initial settings such as changing the exception level and setting the clock as startup processing. Then the main processing is executed. In the main processing, the application program stored in external flash (NOR flash/Serial flash) memory is copied to RAM (System SRAM) according to parameters of loader table. The loader table is a table that the loader program references when copying the application program. For details on the loader table, see 3.2 Loader Table.

In addition, LED0 turns on to signal the start of copy processing, and LED3 turns on to signal the end of copy processing. After copy process is complete, the application program is executed.

Figure 3.2 shows operation overview of loader program.

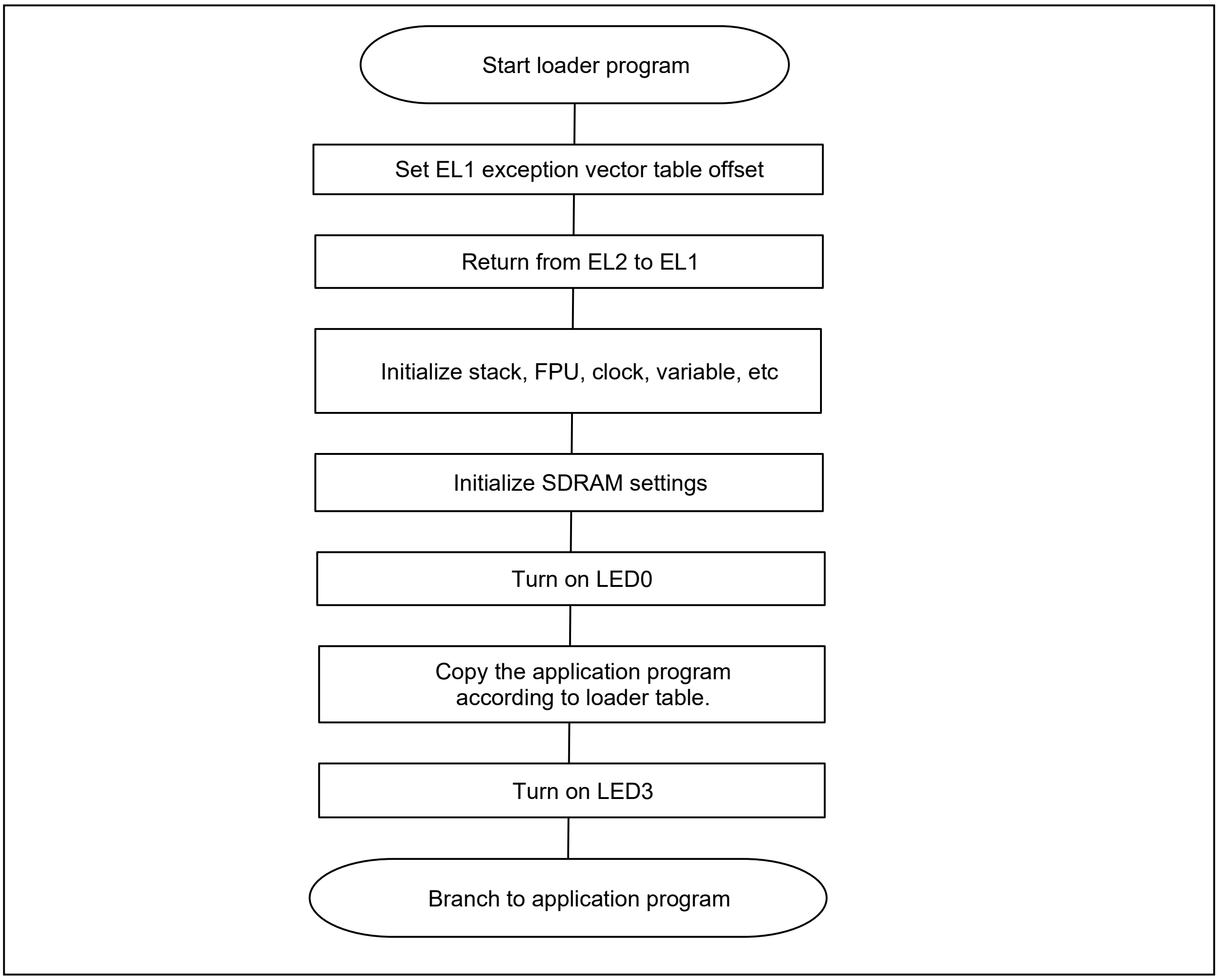


Figure 3.2 Operation overview of loader program

3.1.2 Application Program

The application program performs initial settings such as clock settings, port initialization, and interrupt settings as startup processing. LED0 and LED3, which turned on during loader program processing, turn off in port initialization. Then the main processing is executed.

The main processing executed on System SRAM let the LEDs blink.

The LED blinking process is executed by software interrupt (INTCPU0), and LED0 and LED1 blink.

Figure 3.3 shows operation overview of application program.

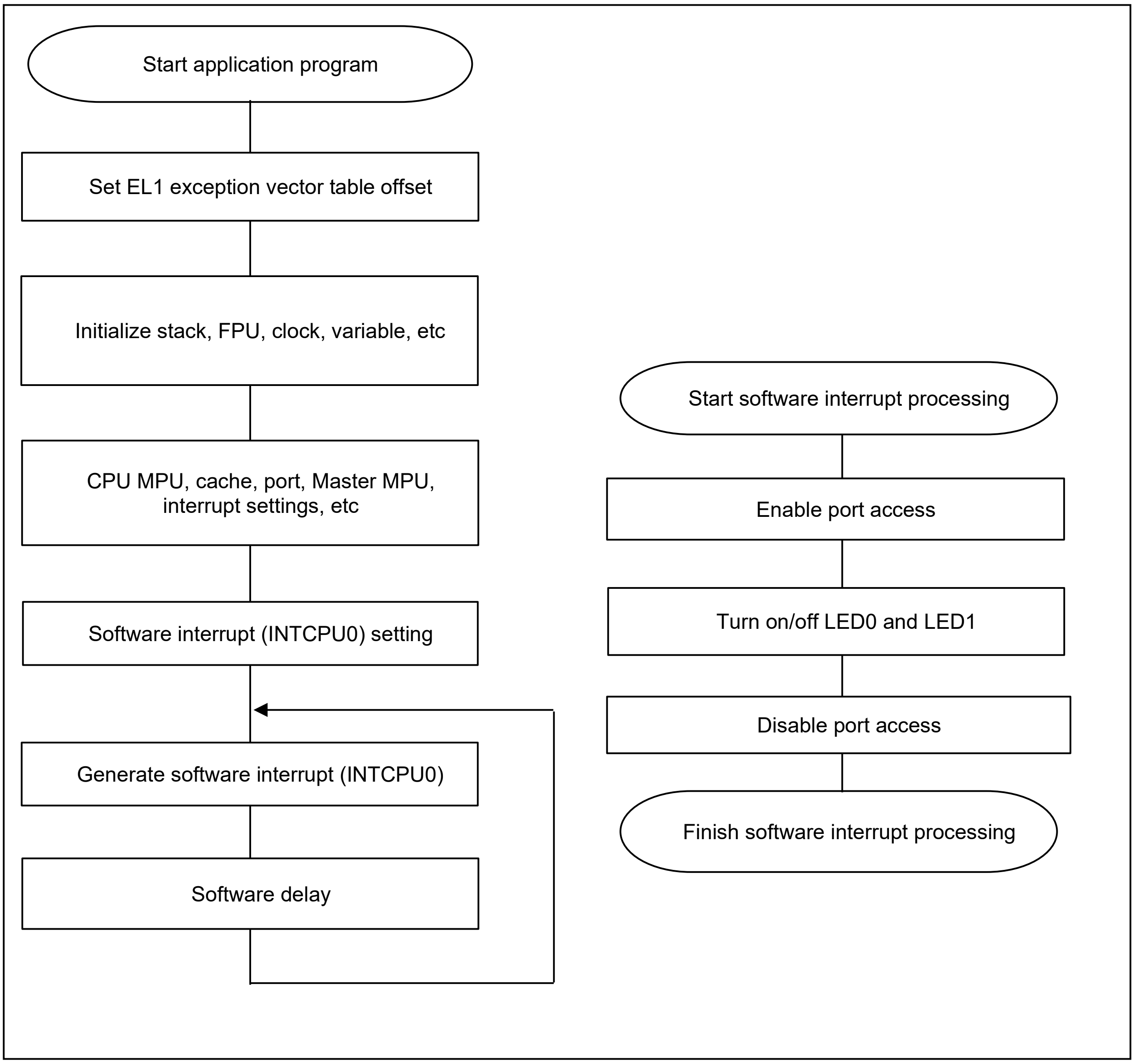


Figure 3.3 Operation overview of application program

3.2 Loader Table

Loader table is a table that the loader program references when copying the application program. The loader table defines the parameters required for program copy, and the loader program performs copy processing according to the table parameters. Multiple loader table entries can be prepared as required, and parameters can be stored in each table entry.

The loader table has four parameters: copy source address, copy destination address, copy size, and table enable/disable flag. Table 3.1 shows the details of the loader table parameters.

In this sample program, four loader tables are prepared in loader_table.c of the loader program. The copy source address depends on the boot operating mode. Tables 3.2 and 3.3 show the loader table parameters in this sample program.

Table 3.1 Loader table parameters

Argument	Parameter	Description
1	Src	Source address of the program to be copied.
2	Dst	Destination address of the program to be copied.
3	Size	Size of the program to be copied.
4	Enable flag	Flag that determines whether the table is enabled/disabled. If this flag is disabled, copy processing will not be performed even if other parameters are set. 0: Disable 1: Enable

Table 3.2 Loader table parameters in this sample program (xSPI0 boot mode)

Table	Src	Dst	Size	Enable flag
0	0x6010_0000	0x1008_0000	0x0000_22AC	0x1
1[*1*2]	0xFFFF_FFFF or 0x6020_0000	0xFFFF_FFFF or 0x1000_0000	0xFFFF_FFFF or 0x0000_1014	0x0 or 0x1
2[*1]	0xFFFF_FFFF	0xFFFF_FFFF	0xFFFF_FFFF	0x0
3[*1]	0xFFFF_FFFF	0xFFFF_FFFF	0xFFFF_FFFF	0x0

Note 1. Table 1,2, and 3 are invalid in this sample program.

2. When CPU1 program is enabled, Table1 hold the parameters for CPU1 program. For details, see”5.3 How to Debug CPU1 Program”.

Table 3.3 Loader table parameters in this sample program (16-bit bus boot mode)

Table	Src	Dst	Size	Enable flag
0	0x7010_0000	0x1008_0000	0x0000_22EC	0x1
1	0xFFFF_FFFF or 0x7018_0000	0xFFFF_FFFF or 0x1000_0000	0xFFFF_FFFF or 0x0000_0EBC	0x0 or 0x1
2	0xFFFF_FFFF	0xFFFF_FFFF	0xFFFF_FFFF	0x0
3[*]	0xFFFF_FFFF	0xFFFF_FFFF	0xFFFF_FFFF	0x0

Note 1. Table 1,2, and 3 are invalid in this sample program.

2. When CPU1 program is enabled, Table1 hold the parameters for CPU1 program. For details, see”5.3 How to Debug CPU1 Program”.

3.3 Memory Map

3.3.1 Program Placement in Flash Memory

Tables 3.4 and 3.5 show the program placed in the flash memory of this sample program. Flash memory address depends on the operating mode. At the start of debugging, the program is downloaded to flash memory. Each program is expanded to the load destination address by boot processing and loader program processing and executed on RAM.

Table 3.4 Program placement in flash memory and load destination address (xSPI0 boot mode)

Flash memory address	Contents	Load destination address
0x6000_0000	Parameters for the loader	-
0x6000_004C	Loader program	0x0010_2000 (BTCM)
0x6008_0000	Loader table	-
0x6010_0000	Application program	0x1008_0000 (System SRAM)
0x6020_0000*1	CPU1 program	0x1000_0000 (System SRAM)

Note 1. CPU1 program is disabled by default. To enable it, see “5.3 How to Debug CPU1 Program”.

Table 3.5 Program placement in flash memory and load destination address (16-bit bus boot mode)

Flash memory address	Contents	Load destination address
0x7000_0000	Parameters for the loader	-
0x7000_004C	Loader program	0x0010_2000 (BTCM)
0x7008_0000	Loader table	-
0x7010_0000	Application program	0x1008_0000 (System SRAM)
0x7018_0000	CPU1 program	0x1000_0000 (System SRAM)

Note 1. CPU1 program is disabled by default. To enable it, see “5.3 How to Debug CPU1 Program”.

3.3.2 Section Assignment in Sample Program

Table 3.6 shows the memory sections used by the loader program, and Table 3.7 shows the sections used by the application program. These sections are defined in the linker script.

Table 3.6 Sections used by loader program

Area Name	Description	Storing/Execution Area[*1]
LOADER_PARAM_BLOCK	Parameters for the loader	Flash
PRG_RBLOCK	Code area (for storing)	Flash
DATA_RBLOCK	Variable area (for storing)	Flash
PRG_WBLOCK	Code area (for execution)	BTCM
DATA_WBLOCK	Variable with initial value area (for execution)	BTCM
DATA_ZBLOCK	Variable without initial value area (for execution)	BTCM
APPLICATION_PRG_RBLOCK	Application program area (for storing)	Flash
APPLICATION_PRG_WBLOCK	Application program area (for executing)	System SRAM
CPU1_PRG_RBLOCK[*2]	CPU1 program area (for storing)	Flash
CPU1_PRG_WBLOCK[*2]	CPU1 program area (for execution)	System SRAM

Note 1. In xSPI0 bus boot, serial flash memory is storing area. In 16-bit bus boot mode, NOR flash memory is storing area.

2. CPU1 program is disabled by default. To enable it, see “5.3 How to Debug CPU1 Program”.

Table 3.7 Sections used by application program

Area name	Description	Storing/Execution Area[*1]
PRG_RBLOCK	Code area (for storing)	Flash
DATA_RBLOCK	Variable area (for storing)	Flash
PRG_WBLOCK	Code area (for execution)	System SRAM
DATA_WBLOCK	Variable with initial value area (for execution)	System SRAM
DATA_ZBLOCK	Variable without initial value area (for execution)	System SRAM

Note 1. In xSPI0 bus boot, serial flash memory is storing area. In 16-bit bus boot mode, NOR flash memory is storing area.

3.3.3 CPU MPU Settings

Table 3.8 shows the CPU MPU settings for areas accessed by CPU in this sample program. These setting are applied during startup processing of the application program.

Table 3.8 CPU MPU Settings

Contents	Address	Memory type
System SRAM	0x1000_0000 to 0x1017_FFFF	Area 2 Normal, cache enabled, non-shared
System SRAM (mirror area)	0x3000_0000 to 0x3017_FFFF	Area 4 Normal, cache disabled, shared
Extended address space (mirror area) xSPI0, xSPI1 CS0, CS2, CS3, CS5	0x4000_0000 to 0x5FFF_FFFF	Area 5 Normal, cache disabled, shared
Extended address space xSPI0, xSPI1 CS0, CS2, CS3, CS5	0x6000_0000 to 0x7FFF_FFFF	Area 6 Normal, cache enabled, non-shared
Non-safety peripheral modules	0x8000_0000 to 0x80FF_FFFF	Area 7 Device (nGnRE) , instruction fetch disabled
Safety peripheral modules	0x8100_0000 to 0x81FF_FFFF	Area 8 Device (nGnRE) , instruction fetch disabled

3.3.4 Exception Processing Vector Table

Exception level 1 of RZ/T2M has 7 types of exception processing (reset, undefined instruction, SVC, prefetch abort, Data abort, IRQ and FIQ exceptions) that are allocated to the 32-byte area starting from specified offset address. Specify a branch instruction to each exception processing in the exception processing vector table.

Table 3.9 lists the contents of exceptional processing vector table for this sample program. Modify the setting to suit your needs.

Table 3.9 Exception Processing Vector Table

Exception	Handler Address*1	Remark*2
RESET	Offset	Branches to startup program
Undefined instruction	Offset + 0x0000 00004	Branches Defaul_Handler
SVC	Offset + 0x0000 00008	Branches Defaul_Handler
Prefetch abort	Offset + 0x0000 0000C	Branches Defaul_Handler
Data abort	Offset + 0x0000 00010	Branches Defaul_Handler
Reserved	Offset + 0x0000 00014	Branches Defaul_Handler
IRQ	Offset + 0x0000 00018	Branches IRQ_Handler (Used for interrupt)
FIQ	Offset + 0x0000 0001C	Branches Defaul_Handler

Note
1. The offset is defined as following.
   Loader program : 0x0010_2000
   Application program : 0x1008_0000
   CPU1 program : 0x1000_0000
2. Software break instruction is executed in Default_Handler.

3.4 Function Specifications

This section describes the function specifications.

3.4.1 system_init

system_init	
Overview	System initialization 1.
Declaration	void system_init (void)
Description	Executes system initialization such as setting the exception handling vector table offset and changing Exception Level to 1 from 2. After that, branches to stack_init.
Arguments	None
Return value	None
Remarks	After boot processing, this function runs as startup process.

3.4.2 stack_init

stack_init	
Overview	System initialization 2.
Declaration	void stack_init (void)
Description	Executes system initialization such as initializing the stacks, FPU, clock, variables for startup process, CPU MPU, cache, and ports. After that, branches to the main process.
Arguments	None
Return value	None
Remarks	None

3.4.3 hal_entry

hal_entry	
Overview	Main process.
Declaration	void hal_entry (void)
Description	• Loader program: Copies the application program to internal RAM. Turns on LED0 before copy processing and turns on LED3 after copy processig is complete. • Application program: Calls System SRAM program and SDRAM program.Then Blinks LED0 and LED1 with software interrupt (INTCPU0).
Arguments	None
Return value	None
Remarks	None

3.4.4 bsp_sdram_init

bsp_sdram_init	
Overview	SDRAM initialization
Declaration	void bsp_sdram_init (void)
Description	Initializes registers related to SDRAM.
Arguments	None
Return value	None
Remarks	None

3.4.5 bsp_copy_multibyte

bsp_copy_multibyte	
Overview	Copy function.
Declaration	void bsp_copy_multibyte (uintptr_t *src, uintptr_t *dst, uintptr_t bytesize)
Description	Copies data for the size specified by the argument.
Arguments	• uintptr_t *src: Copy source address. • uintptr_t *dst: Copy destination address. • uintptr_t bytesize: Copy data size.
Return value	None
Remarks	None

3.5 Flowchart

3.5.1 Loader Program

3.5.1.1 system_init

Figure 3.4 shows flowchart of system_init in the loader program.

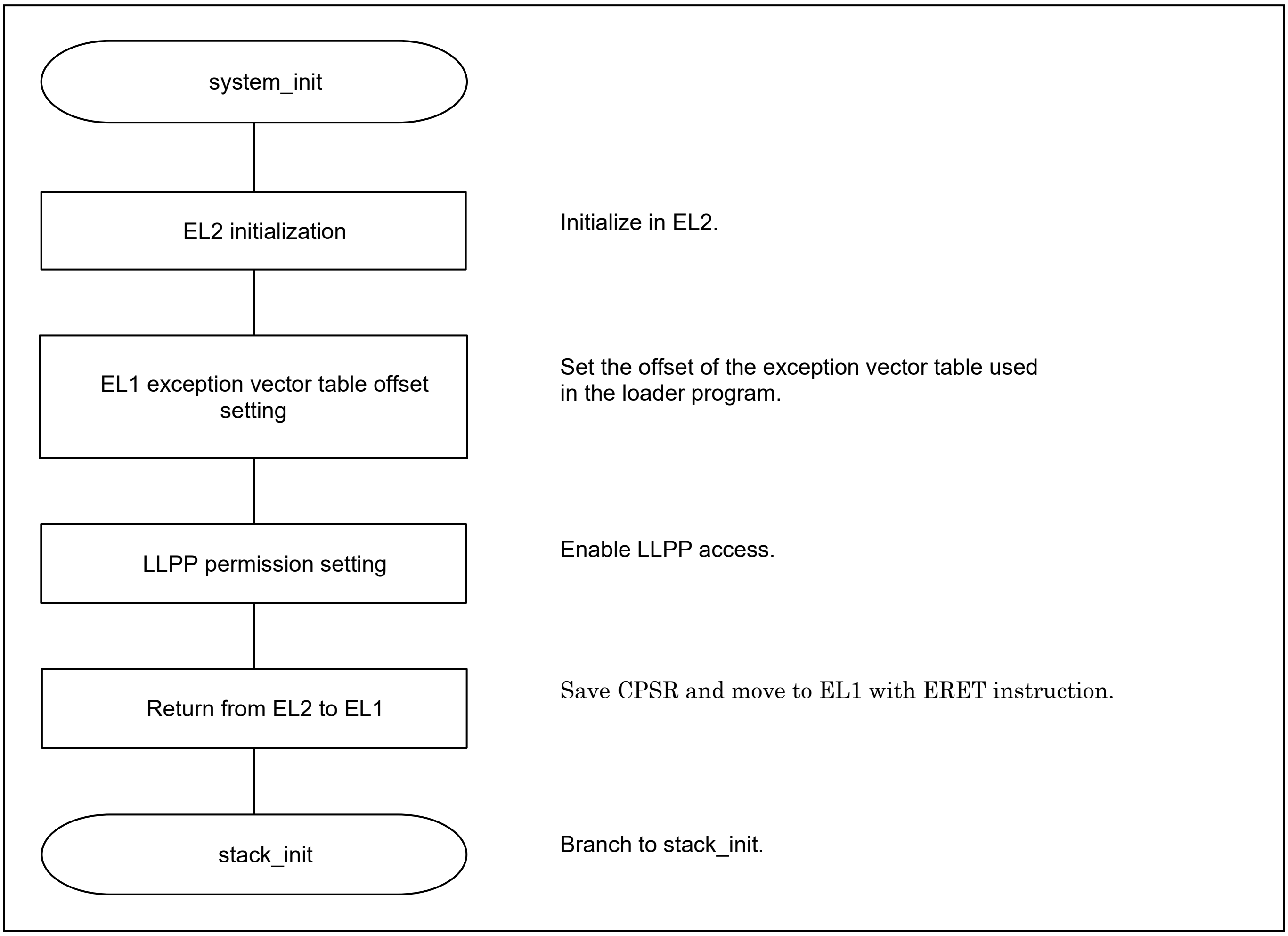


Figure 3.4 system_init processing (loader program)

3.5.1.2 stack_init

Figure 3.5 shows flowchart of stack_init in the loader program.

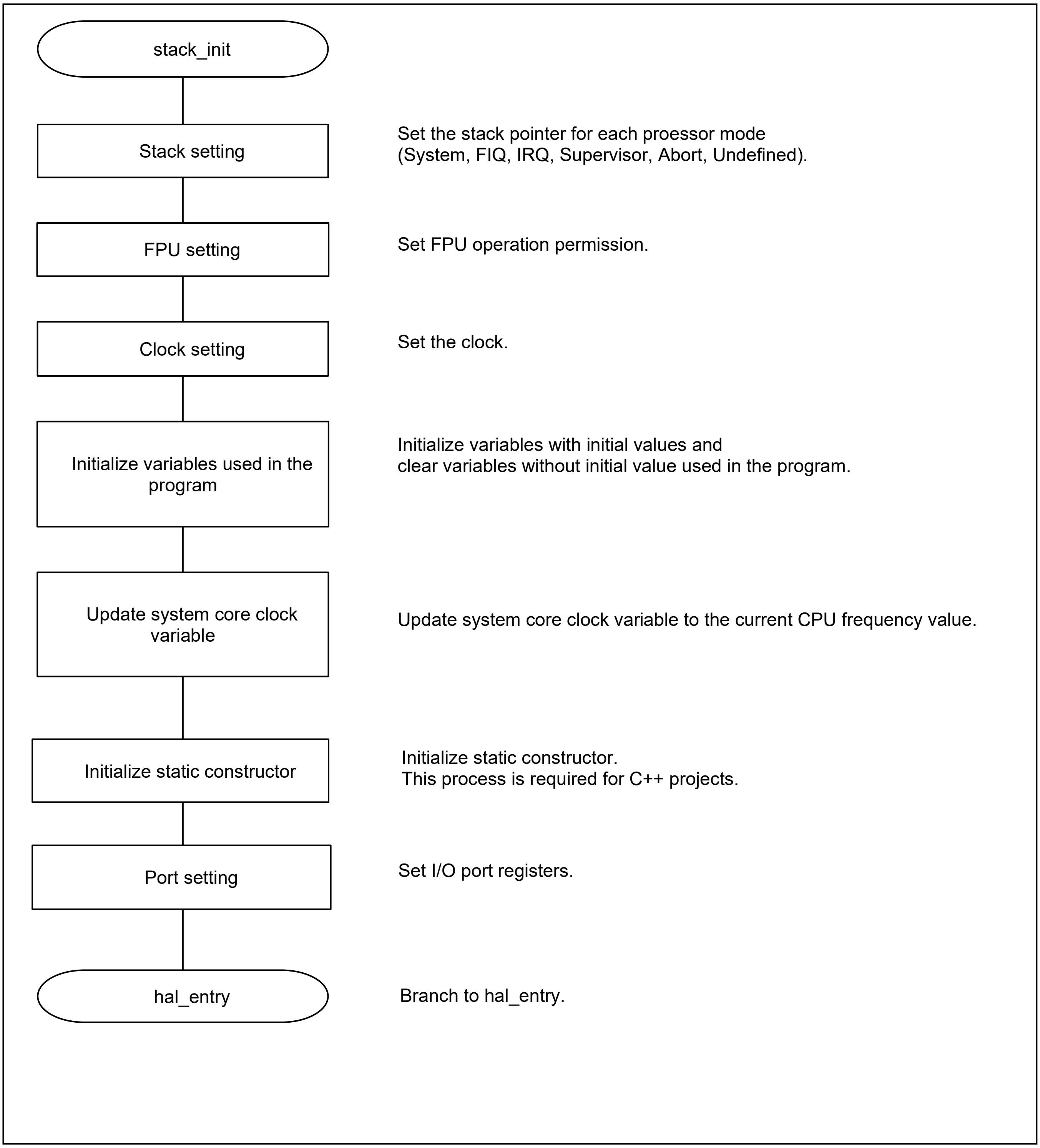


Figure 3.5 stack_init processing (loader program)

3.5.1.3 hal_entry

Figure 3.7 shows flowchart of hal_entry in the loader program.

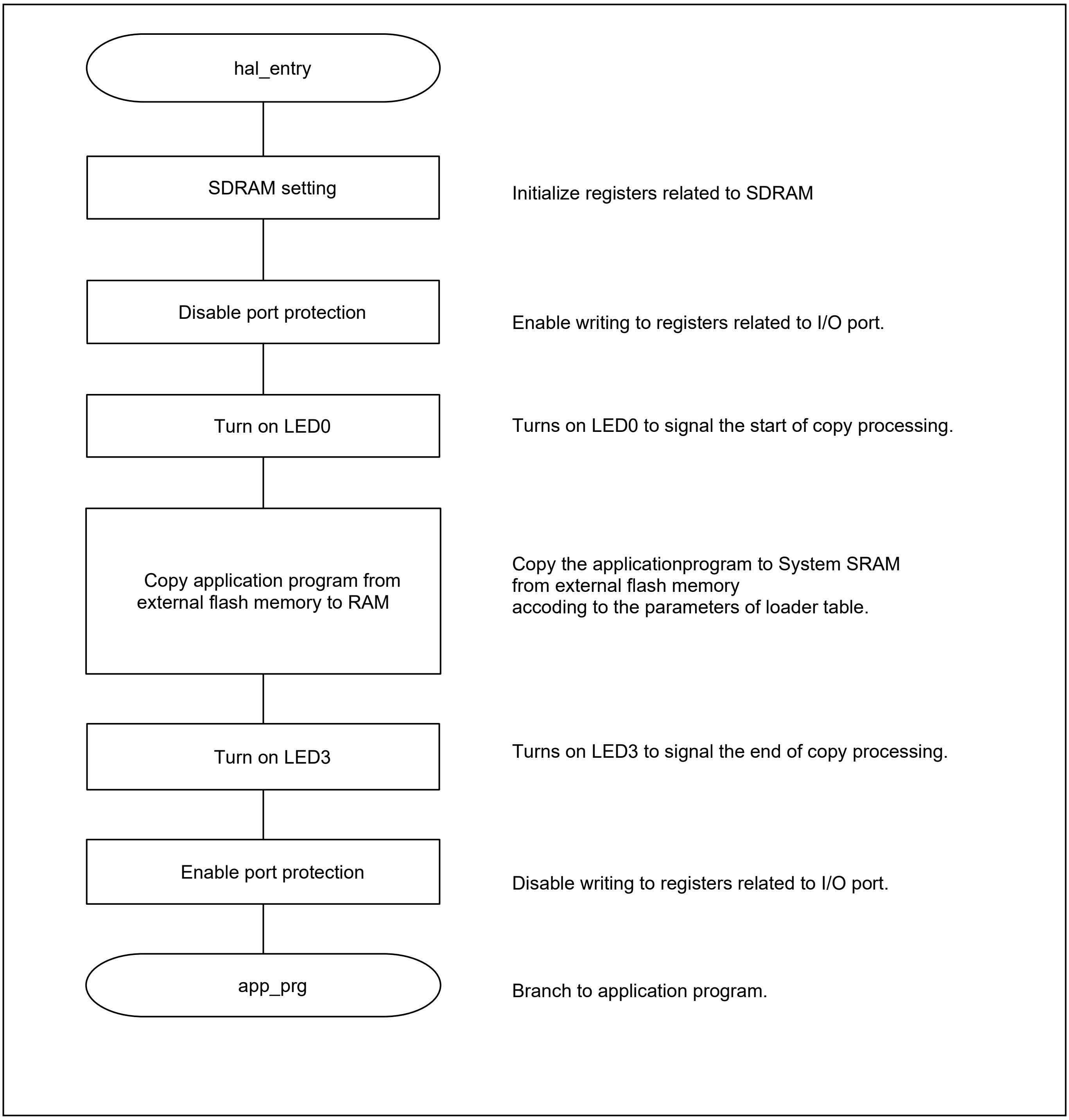


Figure 3.6 hal_entry processing (loader program)

3.5.2 Application Program

3.5.2.1 system_init

Figure 3.8 shows flowchart of system_init in the application program.

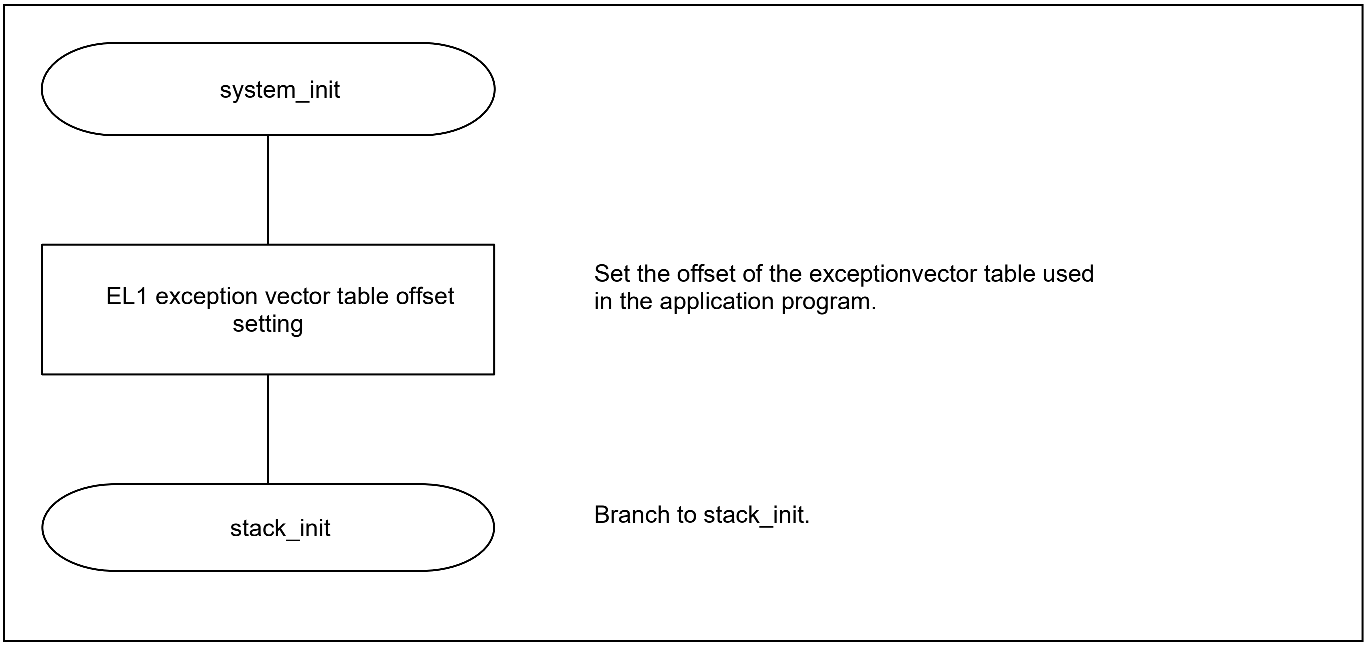


Figure 3.7 system_init processing (application program)

3.5.2.2 stack_init

Figure 3.9 shows flowchart of stack_init in the application program.

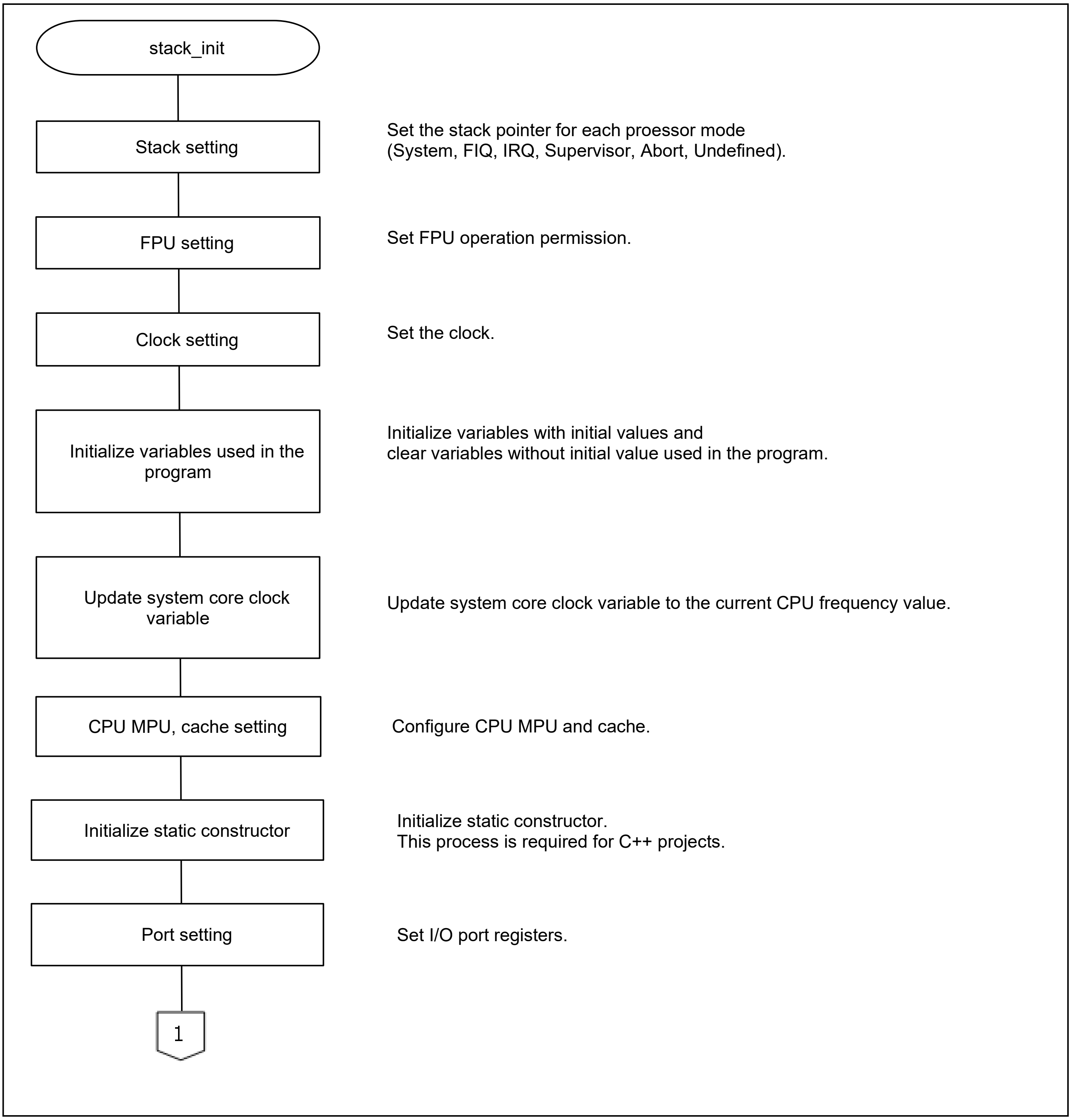


Figure 3.8 stack_init processing (1/2)(application program)

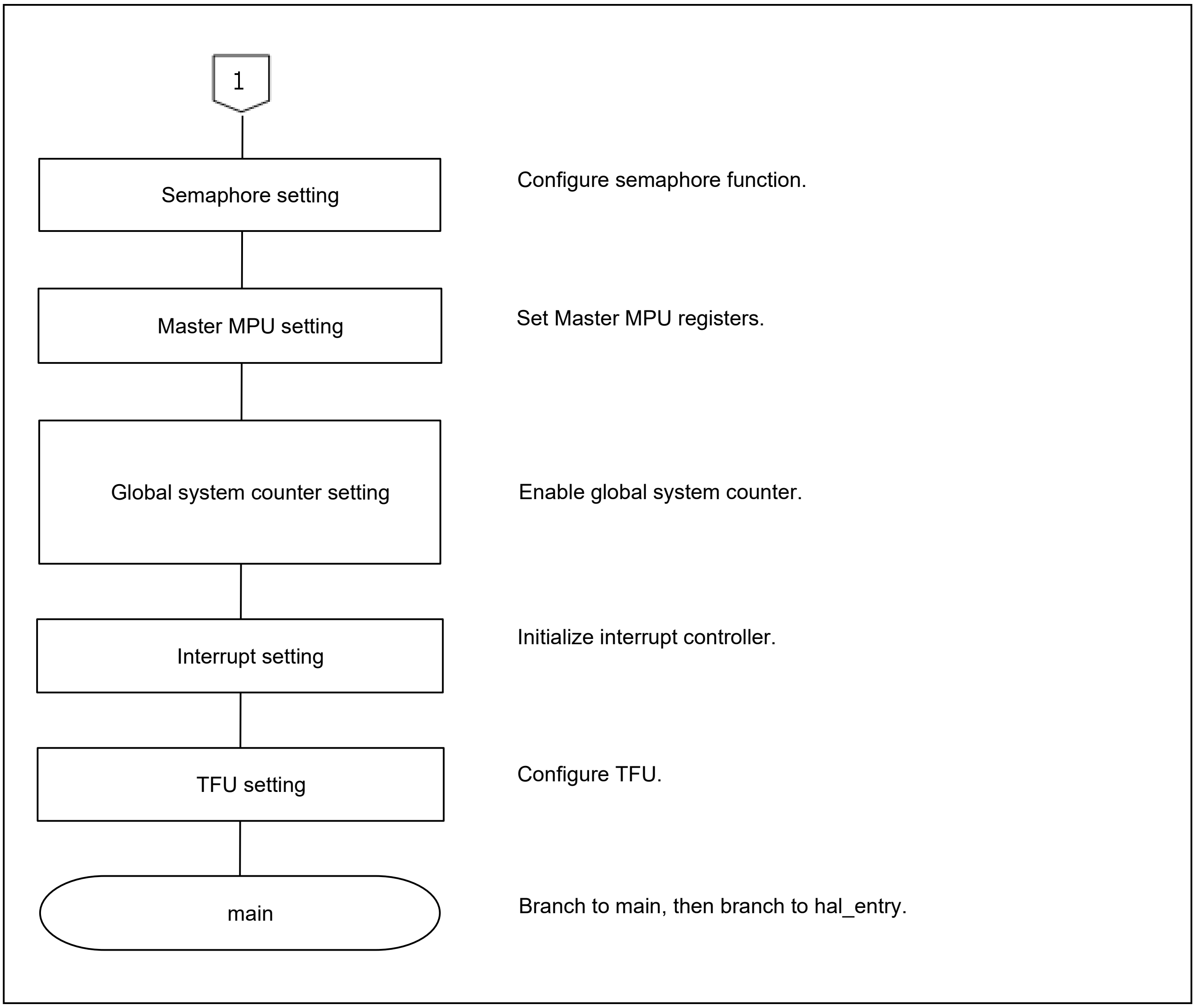


Figure 3.9 stack_init processing (2/2)(application program)

3.5.2.3 hal_entry

Figure 3.11 shows flowchart of hal_entry in the application program.

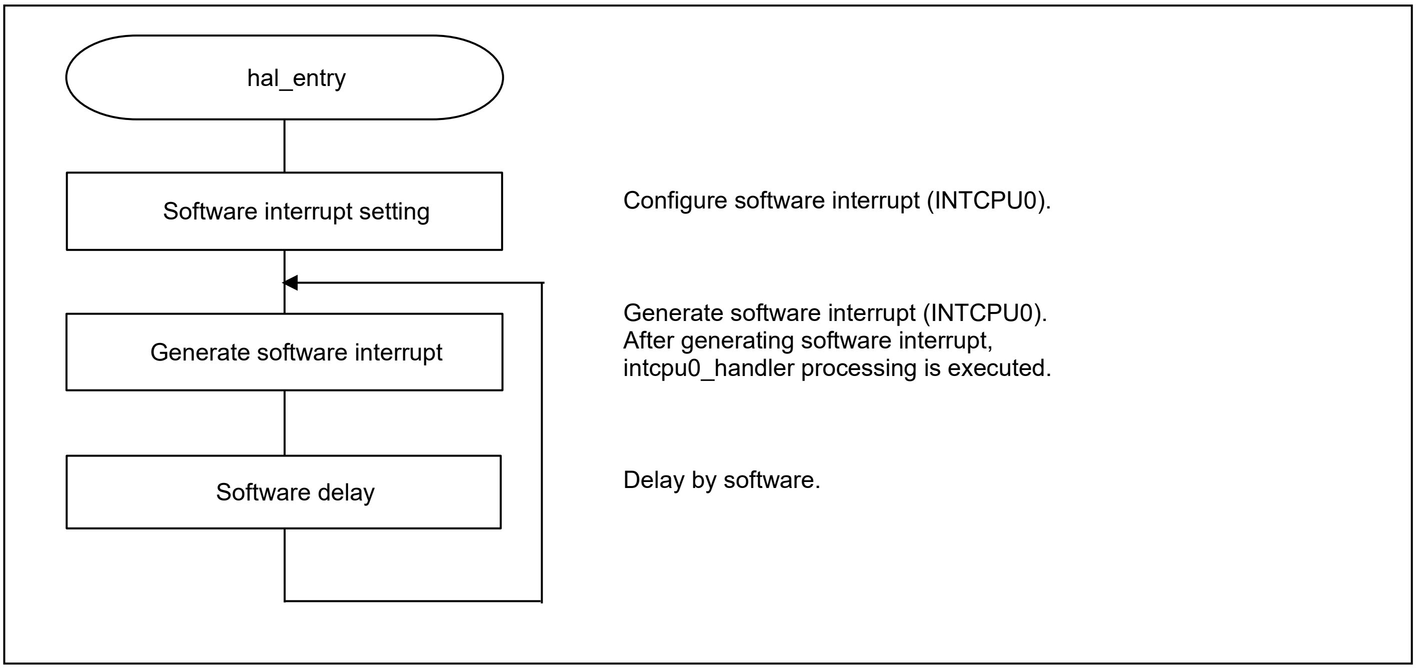


Figure 3.10 hal_entry processing (application program)

Figure 3.12 shows flowchart of interrupt processing in the application program.

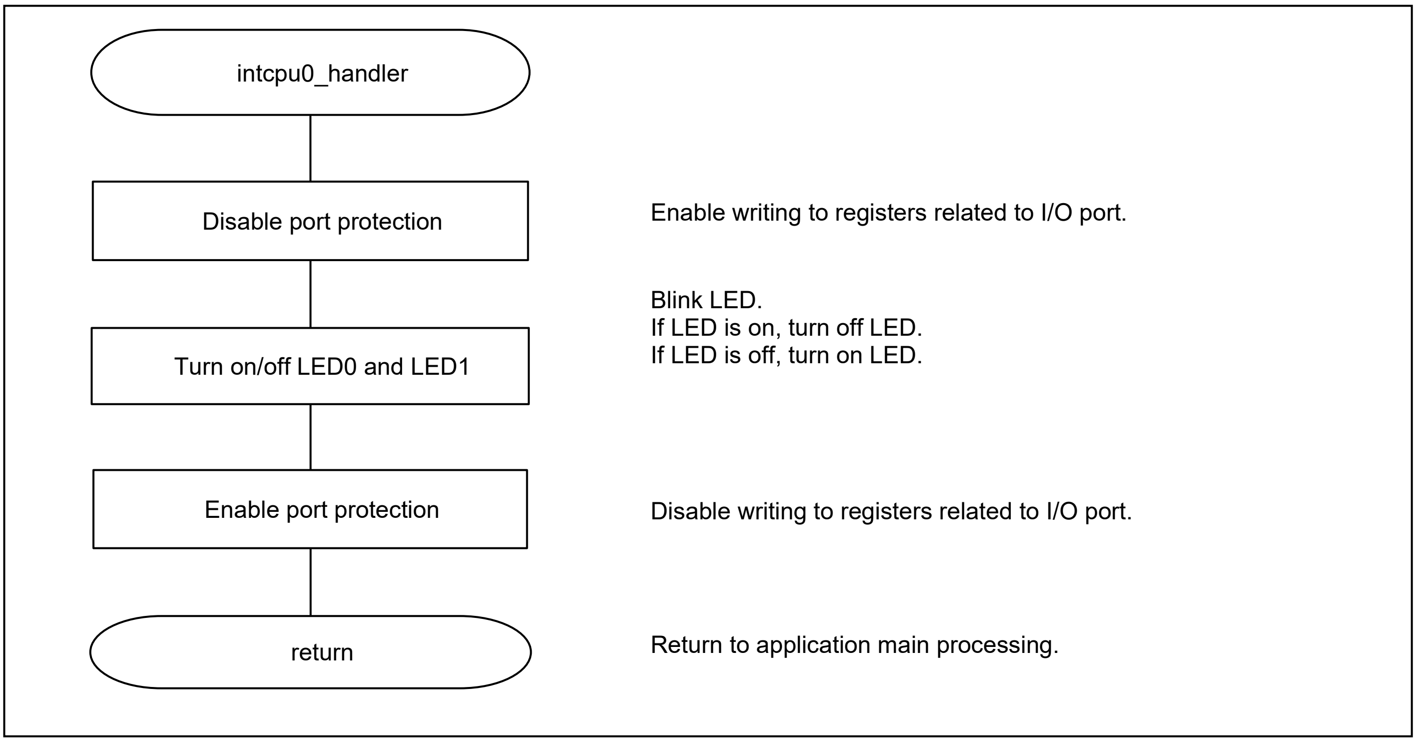


Figure 3.11 interrupt processing (application program)

4. Related Documents

- User’s Manual: Hardware
  RZ/T2M Group User's Manual: Hardware
  Download the latest version from the Renesas Electronics website.

  Renesas Starter Kit+ for RZ/T2M
  Download the latest version from the Renesas Electronics website.

- Technical Update/Technical News
  Download the latest version from the Renesas Electronics website.

- User's Manual: Development Environment
  The latest version for the IAR integrated development environment (IAR Embedded Workbench® for Arm) is available from the IAR Systems website.
  The latest version for the Renesas Electronics integrated development environment (e2studio) is available from the Renesas Electronics website.

5. Appendix Supplementary Notes on Development Environments

This section shows the steps up to the start of debugging of the sample program in each of the available development environments.

5.1 Debug procedure for this sample program.

5.1.1 EWARM from IAR systems

1. Launch EWARM and open "RZT2M_bsp_xpi0bootx1_app_loader.eww" with following procedure. "[File] -> [Open Workspace] -> select Loader_application_projects\RZT2M_bsp_xpi0bootx1_app_loader.eww"
2. Select "RZ/T2M_bsp_xspi0boot1_app" project in Workspace box as Figure 6.1. And run build with "[Project] -> [Rebuild All]"
3. Then, select "RZ/T2M_bsp_xspi0boot1_loader" project in Workspace box. And run build with "[Project] -> [Rebuild All]"
4. Make sure that your PC and RZ/T2M evaluation board are connected with debugger. Then, start debugging with "[Project] -> [Download and debug]"
5. After emulator connecting, both loader program and application program are downloaded to external serial flash memory by Flash Downloader. After downloading is complete, the debugging is started (Program starts running).

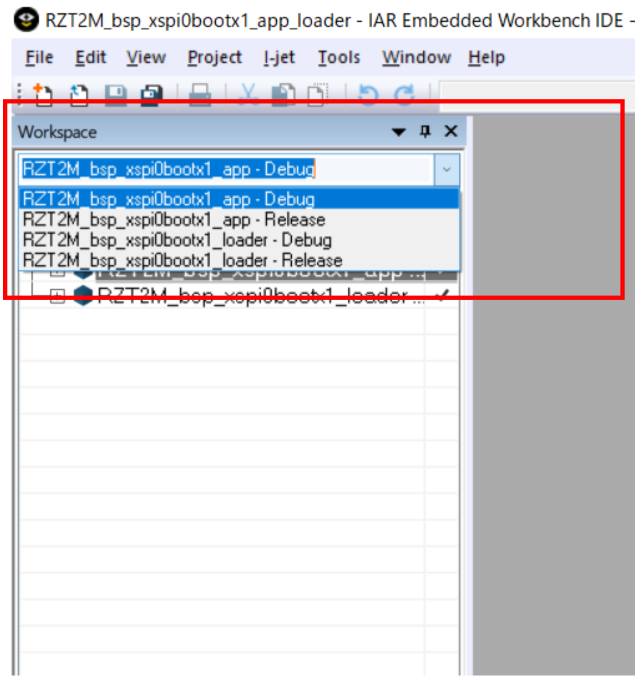


Figure 6.1 Project selection

NOTE: Select loader project when you start debugging as Figure 6.2.
Application program is already specified as extra image in loader project option.
"Right click loader project -> [Options...] -> [Debugger] -> [Images] -> [Download extra image]"

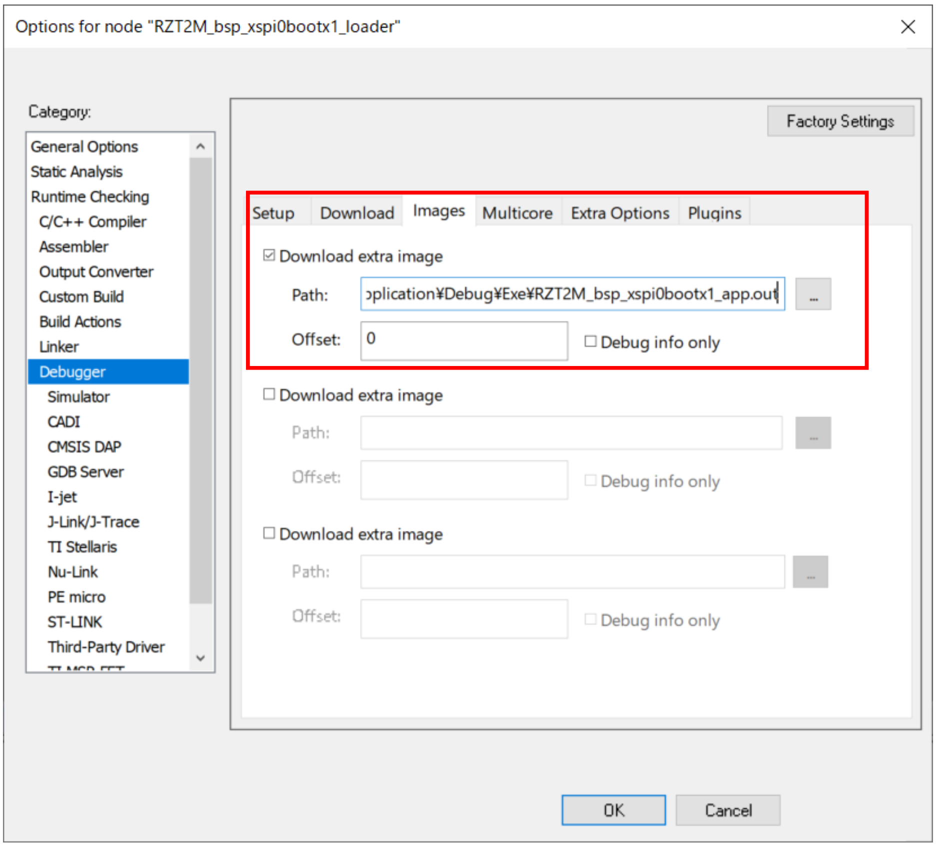


Figure 6.2 EWARM option setting.

5.1.2 e2studio from Renesas

1. Launch e2studio with your workspace. Then, click as following.
   "[File] -> [Import...] -> [General] -> [Existing Projects into Workspace] -> [Next >]"

2. Select "[Select archive file]" and Browse
   "Loader_application_projects\RZT2M_bsp_xpi0bootx1_app_loader.zip".
   Then, click "[Finish]"

3. Run build with "[Project] -> [Build All]"

4. Make sure that your PC and RZ/T2M evaluation board are connected with debugger. Then, select "RZ/T2M_bsp_xspi0bootx1_loader" in connection setting and start debugging with "[Debug]"

5. After emulator connecting, both loader program and application program are downloaded to external serial flash memory by Flash Downloader. After downloading is complete, the debugging is started (Program starts running).

NOTE: Select the loader project when you start debugging. With the following debug configuration, the loader program and application program are written to the external serial flash memory at the same time when the loader project is connected for debug.

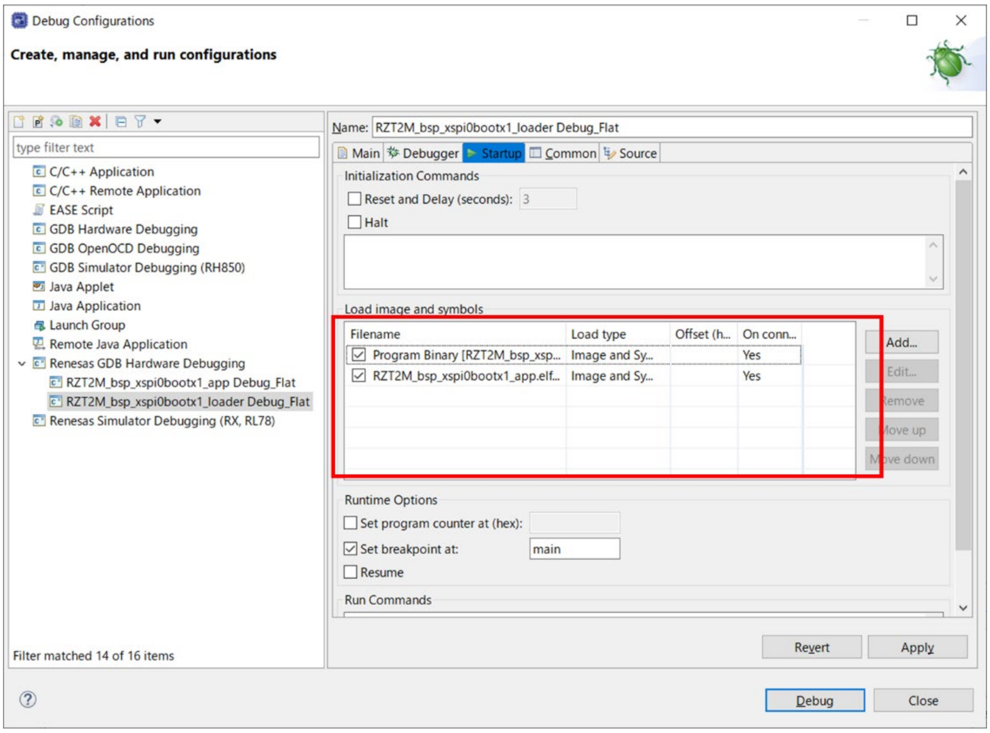


Figure 6.3 Debug configurations in e² studio

5.2 Example of changing RAM placement in application program

The sample program copies the application program from the source address to the destination address specified in the loader table. The user can change the placement of the application program by rewriting the source and destination addresses as necessary.

5.2.1 EWARM from IAR systems

Below is an example (xspi0boot project) of changing the placement for application program from System SRAM to ATCM.

fsp_xspi0_boot_loader.icf (Linker script for loader program)

```
Default
~~~
/* Internal memory */
define region ATCM_region      =   mem:[from 0x00000000 size  128K];
define region BTCM_region      =   mem:[from 0x00100000 size  128K];
define region BTCM_LDR_region  =   mem:[from 0x00102000 size  56K];
define region APPLICATION_RAM_region  =  mem:[from 0x10080000 size 64K];

/* Flash memory */
define region LOADER_TABLE_region  =  mem:[from 0x60080000 size 64K];
define region APPLICATION_ROM_region  =  mem:[from 0x60100000 size 64K];
~~~

↓

After changing
~~~
/* Internal memory */
define region ATCM_region      =   mem:[from 0x00000000 size  128K];
define region BTCM_region      =   mem:[from 0x00100000 size  128K];
define region BTCM_LDR_region  =   mem:[from 0x00102000 size  56K];
define region APPLICATION_RAM_region  =  mem:[from 0x00000000 size 64K]; /* Change copy destination
address to ATCM */

/* Flash memory */
define region LOADER_TABLE_region  =  mem:[from 0x60080000 size 64K];
define region APPLICATION_ROM_region  =  mem:[from 0x60100000 size 64K];
~~~
```

fsp_xspi0_boot_app.icf (Linker script for application program)

```
Default
~~~
place at start of SYSTEM_RAM_PRG_region { block PRG_WBLOCK };
place in SYSTEM_RAM_PRG_region          { block DATA_WBLOCK };
place in SYSTEM_RAM_PRG_region          { block DATA_ZBLOCK };
place in SYSTEM_RAM_PRG_region          { rw data,
                                          rw section .sys_stack,
                                          rw section .svc_stack,
                                          rw section .irq_stack,
                                          rw section .fiq_stack,
                                          rw section .und_stack,
                                          rw section .abt_stack };
place in SYSTEM_RAM_PRG_region          { rw section HEAP };~~~
~~~

↓

After changing
~~~
place at start of ATCM_region  { block PRG_WBLOCK };     /* Change code area to ATCM */
place in ATCM_region           { block DATA_WBLOCK };    /* Change data area to ATCM */
place in ATCM_region           { block DATA_ZBLOCK };    /* Change bss area to ATCM */
place in ATCM_region           { rw data,                /* Change stack area to ATCM */
                                 rw section .sys_stack,
                                 rw section .svc_stack,
                                 rw section .irq_stack,
                                 rw section .fiq_stack,
```

```
                                    rw section .und_stack,
                                    rw section .abt_stack };
place in ATCM_region              { rw section HEAP };      /* Change HEAP area to ATCM */
~~~
```

For example, if you place the application program code in ATCM, the data with and without initial values and the stack in BTCM, and heap area in System SRAM, write as follows.

fsp_xspi0_boot_app.icf (Linker script for application program)

```
~~~
place at start of ATCM_region  { block PRG_WBLOCK };     /* Change code area to ATCM */
place in BTCM_region           { block DATA_WBLOCK };    /* Change data area to BTCM */
place in BTCM_region           { block DATA_ZBLOCK };    /* Change bss area to BTCM */
place in BTCM_region           { rw data,                /* Change stack area to BTCM */
                                 rw section .sys_stack,
                                 rw section .svc_stack,
                                 rw section .irq_stack,
                                 rw section .fiq_stack,
                                 rw section .und_stack,
                                 rw section .abt_stack };
place in SYSTEM_RAM_region           { rw section HEAP }; /* Change HEAP area to SYSTEM SRAM */
~~~
```

5.2.2 e2 studio from Renesas

Below is an example (xspi0boot project) of changing the placement for application program from System SRAM to ATCM.

fsp_xspi0_boot_loader.ld (Linker script for loader program)

```
Default
~~~
.IMAGE_APP_RAM 0x10080000 : AT (0x10080000)
{
      IMAGE_APP_RAM_start = .;
      KEEP(*(APP_IMAGE_RAM))
}
.IMAGE_APP_FLASH_section 0x60100000 : AT (0x60100000)
{
      IMAGE_APP_FLASH_section_start = .;
      KEEP(./src/Flash_section.o(.IMAGE_APP_FLASH_section))
      IMAGE_APP_FLASH_section_end = .;
}
~~~

↓

After changing
~~~
.IMAGE_APP_RAM 0x00000000 : AT (0x00000000) /* Change copy destination address to ATCM */
{
      IMAGE_APP_RAM_start = .;
      KEEP(*(APP_IMAGE_RAM))
}
.IMAGE_APP_FLASH_section 0x60100000 : AT (0x60100000)
{
      IMAGE_APP_FLASH_section_start = .;
      KEEP(./src/Flash_section.o(.IMAGE_APP_FLASH_section))
      IMAGE_APP_FLASH_section_end = .;
}
~~~
```

fsp_xspi0_boot_app.ld (Linker script for application program)

```
Default
~~~
.text 0x10080000 : AT (_mtext)
{
   ~~~
} > SYSTEM_RAM
.rvectors :
{
   ~~~
} > SYSTEM_RAM
.ARM.extab :
{
   ~~~
} > SYSTEM_RAM
.ARM.exidx :
{
   ~~~
} > SYSTEM_RAM
.got :
{
   ~~~
} > SYSTEM_RAM
.data : AT (_mdata)
{
   ~~~
} > SYSTEM_RAM
.bss :
{
   ~~~
} > SYSTEM_RAM
.heap (NOLOAD) :
```

```
{
    ~~~
} > SYSTEM_RAM
.thread_stack (NOLOAD):
{
    ~~~
} > SYSTEM_RAM
.sys_stack (NOLOAD) :
{
    ~~~
} > SYSTEM_RAM
.svc_stack (NOLOAD) :
{
    ~~~
} > SYSTEM_RAM
.irq_stack (NOLOAD) :
{
    ~~~
} > SYSTEM_RAM
.fiq_stack (NOLOAD) :
{
    ~~~
} > SYSTEM_RAM
.und_stack (NOLOAD) :
{
    ~~~
} > SYSTEM_RAM
.abt_stack (NOLOAD) :
{
    ~~~
} > SYSTEM_RAM

~~~

↓

After Changing
~~~
.text 0x00000000 : AT (_mtext) /* Change code area to ATCM */
{
    ~~~
} > ATCM
.rvectors :
{
    ~~~
} > ATCM
.ARM.extab :
{
    ~~~
} > ATCM
.ARM.exidx :
{
    ~~~
} > ATCM
.got :
{
    ~~~
} > ATCM
.data : AT (_mdata) /* Change data area to ATCM */
{
    ~~~
} > ATCM
.bss : /* Change bss area to ATCM */
{
    ~~~
} > ATCM
.heap (NOLOAD) : /* Change heap area to ATCM */
{
    ~~~
} > ATCM
.thread_stack (NOLOAD):
{
    ~~~
} > ATCM
.sys_stack (NOLOAD) : /* Change stack area to ATCM */
{
```

```
    ~~~
} > ATCM
.svc_stack (NOLOAD) :
{
    ~~~
} > ATCM
.irq_stack (NOLOAD) :
{
    ~~~
} > ATCM
.fiq_stack (NOLOAD) :
{
    ~~~
} > ATCM
.und_stack (NOLOAD) :
{
    ~~~
} > ATCM
.abt_stack (NOLOAD) :
{
    ~~~
} > ATCM
~~~
```

For example, if you place the application program code in ATCM, the data with and without initial values and the stack in BTCM, and heap area in System SRAM, write as follows.

fsp_xspi0_boot_app.ld (Linker script for application program)

```
~~~
.text 0x00000000 : AT (_mtext) /* Change code area to ATCM */
{
    ~~~
} > ATCM
.rvectors :
{
    ~~~
} > ATCM
.ARM.extab :
{
    ~~~
} > ATCM
.ARM.exidx :
{
    ~~~
} > ATCM
.got :
{
    ~~~
} > ATCM
.data : AT (_mdata) /* Change data area to BTCM */
{
    ~~~
} > BTCM
.bss : /* Change bss area to BTCM */
{
    ~~~
} > BTCM
.heap (NOLOAD) : /* Change heap area to SYSTEM SRAM */
{
    ~~~
} > SYSTEM_RAM
.thread_stack (NOLOAD):
{
    ~~~
} > BTCM
.sys_stack (NOLOAD) : /* Change stack area to BTCM */
{
    ~~~
} > BTCM
```

```
.svc_stack (NOLOAD) :
{
    ~~~
} > BTCM
.irq_stack (NOLOAD) :
{
    ~~~
} > BTCM
.fiq_stack (NOLOAD) :
{
    ~~~
} > BTCM
.und_stack (NOLOAD) :
{
    ~~~
} > BTCM
.abt_stack (NOLOAD) :
{
    ~~~
} > BTCM
~~~
```

5.3 How to Debug CPU1 Program

The sample program is CPU0 single core operation with default configuration. The definition "USE_CPU1" is added to the project options, and changing its value enables the program required to run CPU1.

When CPU1 configuration is enabled, the Table 1 parameters in the loader table are replaced with the information for copying the CPU1 program. The loader program refers to the parameters and copies the CPU1 program in addition to the application program.

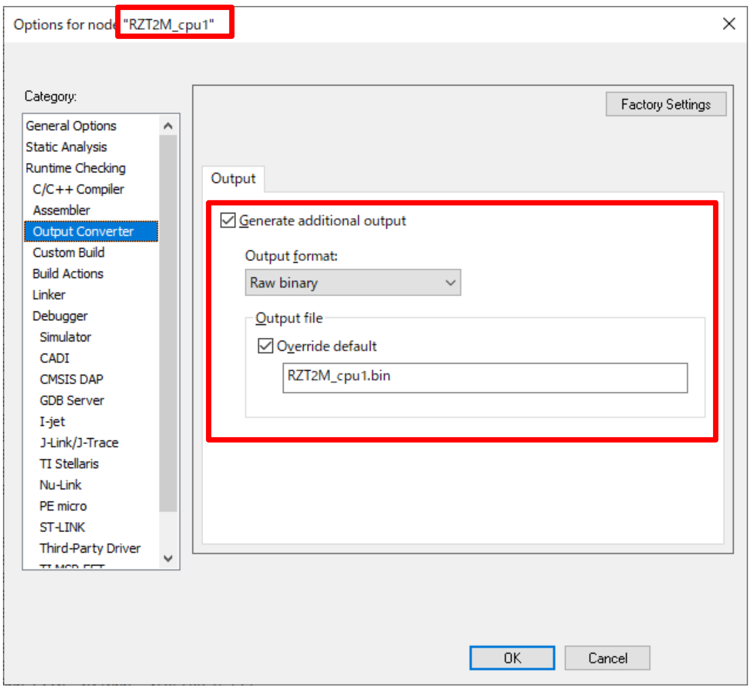
In addition, CPU1 reset release process is added to the application program. After the reset release process is executed, CPU1 program runs from the beginning of System SRAM (0x1000_0000).

Detailed procedures for debugging CPU1 programs in each development environment are shown on the following pages.

5.3.1 EWARM from IAR systems

1. Build CPU1 program

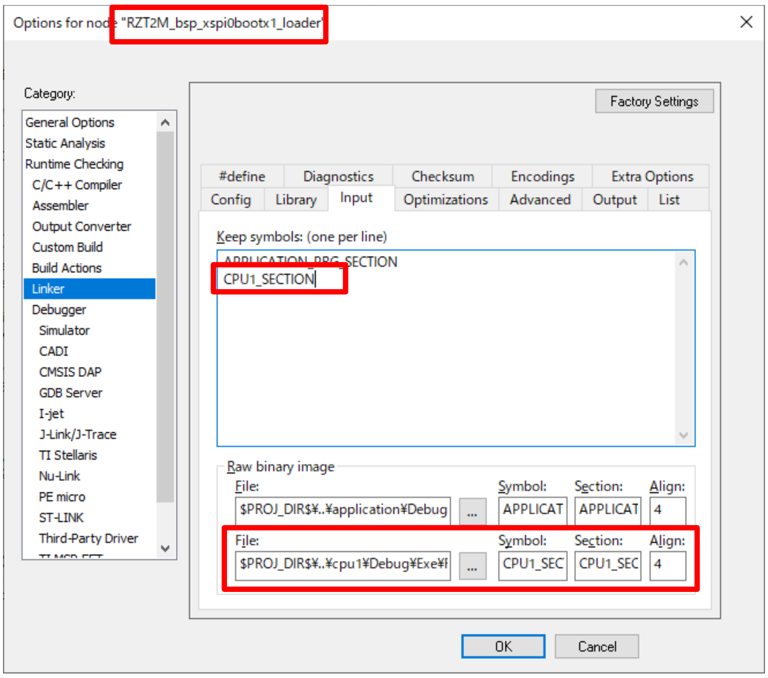
Open the file “**Loader_application_projects\cpu1\RZT2M_cpu1.eww”** and build the **CPU1 program.**
The following project option settings will output build artifacts in raw binary.
After building the CPU1 program, close the RZ/T2M_cpu1.eww workspace.



2. Link CPU1 program to CPU0 loader program

Add the following project option settings to link the CPU1 binary to the CPU0 loader program.

Project for the loader program : [Options] -> [Linker] -> [Input]
Keep symbols : CPU1_SECTION
File : $PROJ_DIR$\..\cpu1\Debug\Exe\RZT2M_cpu1.bin
Symbol : CPU1_SECTION
Section : CPU1_SECTION
Align : 4

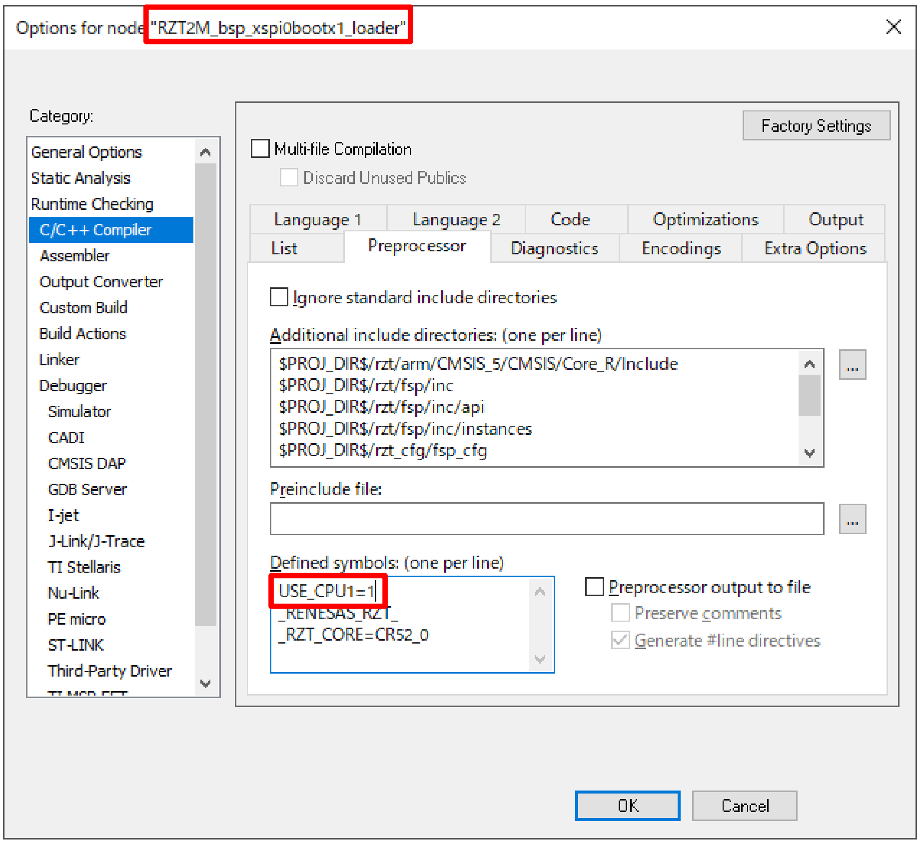


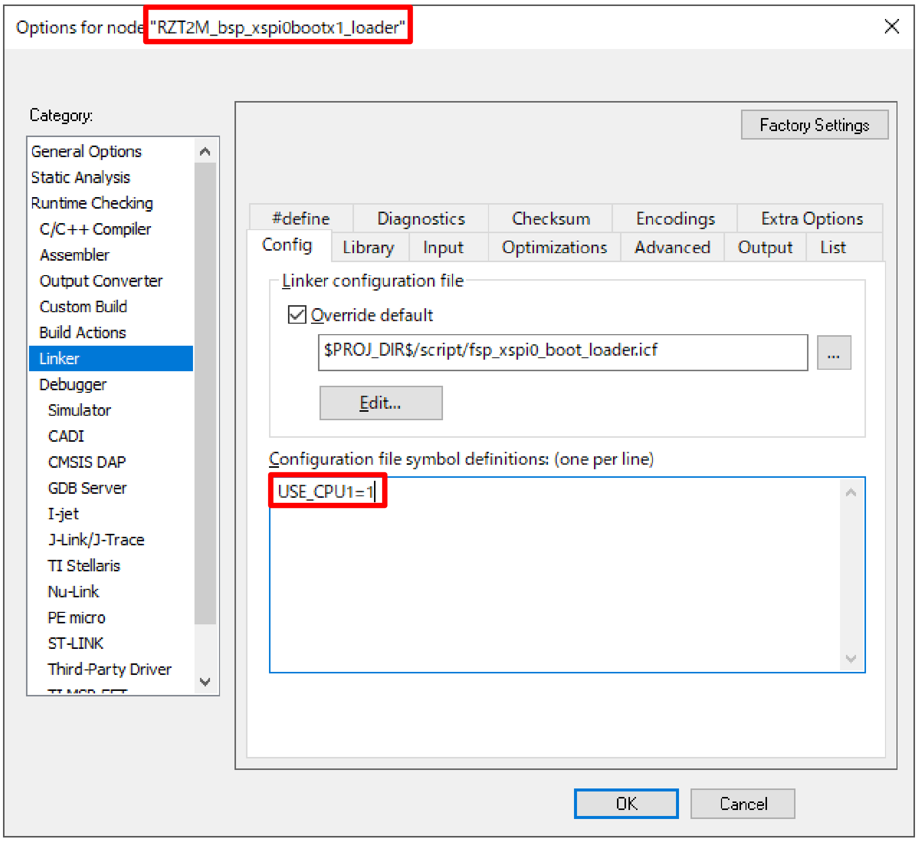
3. Enable USE_CPU1 definition

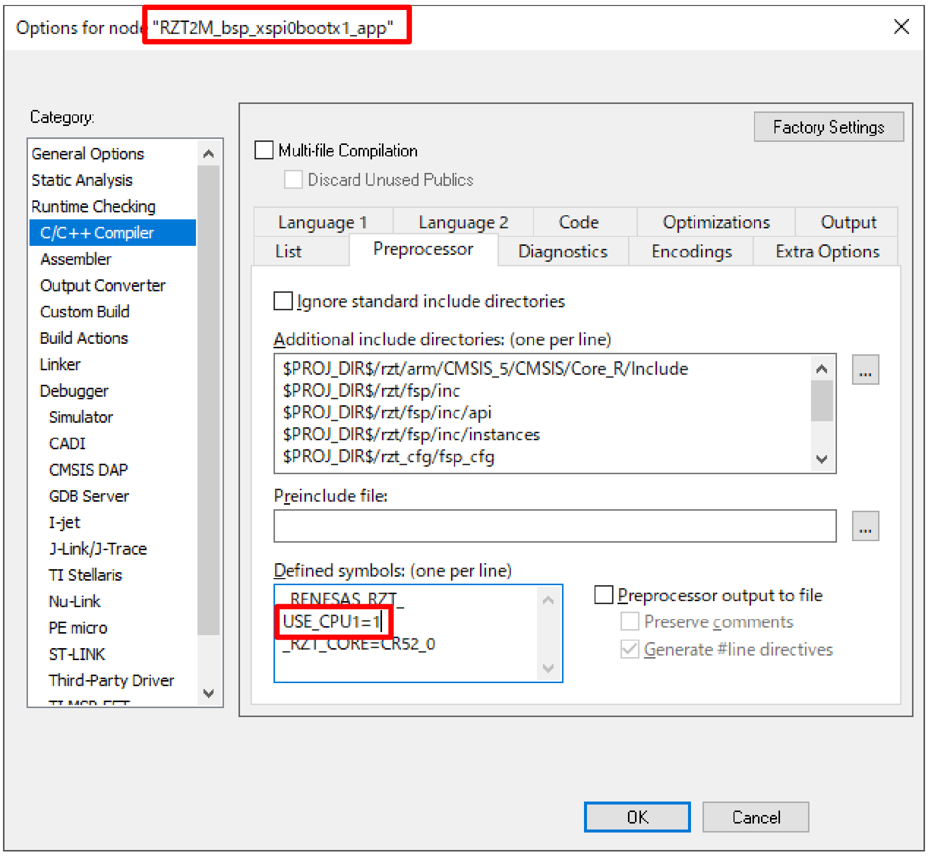
Activate the definition to run the CPU1 program. Change the value of "USE_CPU1" defined in the project options of the loader program and the application program from 0 to 1.

- **Project for the loader program**
  [Options] -> [C/C++ Compiler] -> [Preprocessor]: USE_CPU1=1
  [Options] -> [Linker] -> [Config]: USE_CPU1=1

- **Project for the Application program**
  [Options] -> [C/C++ Compiler] -> [Preprocessor]: USE_CPU1=1


Options for node "RZT2M_bsp_xspi0bootx1_loader"
Category:
C/C++ Compiler
Preprocessor
Additional include directories: (one per line)
$PROJ_DIR$/rzt/arm/CMSIS_5/CMSIS/Core_R/Include
$PROJ_DIR$/rzt/fsp/inc
$PROJ_DIR$/rzt/fsp/inc/api
$PROJ_DIR$/rzt/fsp/inc/instances
$PROJ_DIR$/rzt_cfg/fsp_cfg
Defined symbols: (one per line)
USE_CPU1=1
_RENESAS_RZT_
_RZT_CORE=CR52_0
OK
Cancel


Options for node "RZT2M_bsp_xspi0bootx1_loader"
Category:
Linker
Config
Linker configuration file
Override default
$PROJ_DIR$/script/fsp_xspi0_boot_loader.icf
Edit...
Configuration file symbol definitions: (one per line)
USE_CPU1=1
OK
Cancel

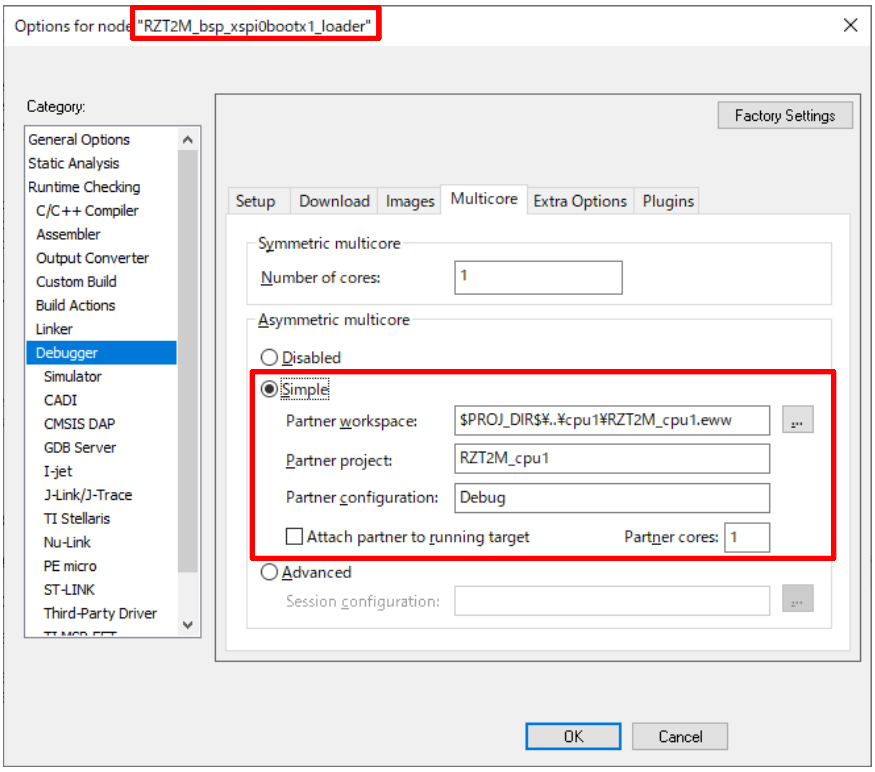

Options for node "RZT2M_bsp_xspi0bootx1_app"
Category:
General Options
Static Analysis
Runtime Checking
C/C++ Compiler
Assembler
Output Converter
Custom Build
Build Actions
Linker
Debugger
Simulator
CADI
CMSIS DAP
GDB Server
I-jet
J-Link/J-Trace
TI Stellaris
Nu-Link
PE micro
ST-LINK
Third-Party Driver
Factory Settings
Multi-file Compilation
Discard Unused Publics
Language 1
Language 2
Code
Optimizations
Output
List
Preprocessor
Diagnostics
Encodings
Extra Options
Ignore standard include directories
Additional include directories: (one per line)
$PROJ_DIR$/rzt/arm/CMSIS_5/CMSIS/Core_R/Include
$PROJ_DIR$/rzt/fsp/inc
$PROJ_DIR$/rzt/fsp/inc/api
$PROJ_DIR$/rzt/fsp/inc/instances
$PROJ_DIR$/rzt_cfg/fsp_cfg
Preinclude file:
Defined symbols: (one per line)
_RENESAS_RZT_
USE_CPU1=1
_RZT_CORE=CR52_0
Preprocessor output to file
Preserve comments
Generate #line directives
OK
Cancel

4. Settings for debugging CPU1 project

To debug the CPU1 program, use EWARM's multicore debugging function. The following additional project option settings enable debugging of CPU1 projects.

Project for the loader program: [Options] -> [Debugger] -> [Multicore]

Asymmetric multicore	: Enable “Simple”.
Partner workspace	: $PROJ_DIR$\..\cpu1\RZT2M_cpu1.eww
Partner project	: RZT2M_cpu1
Partner configuration	: Debug



5. Rebuild and run the project

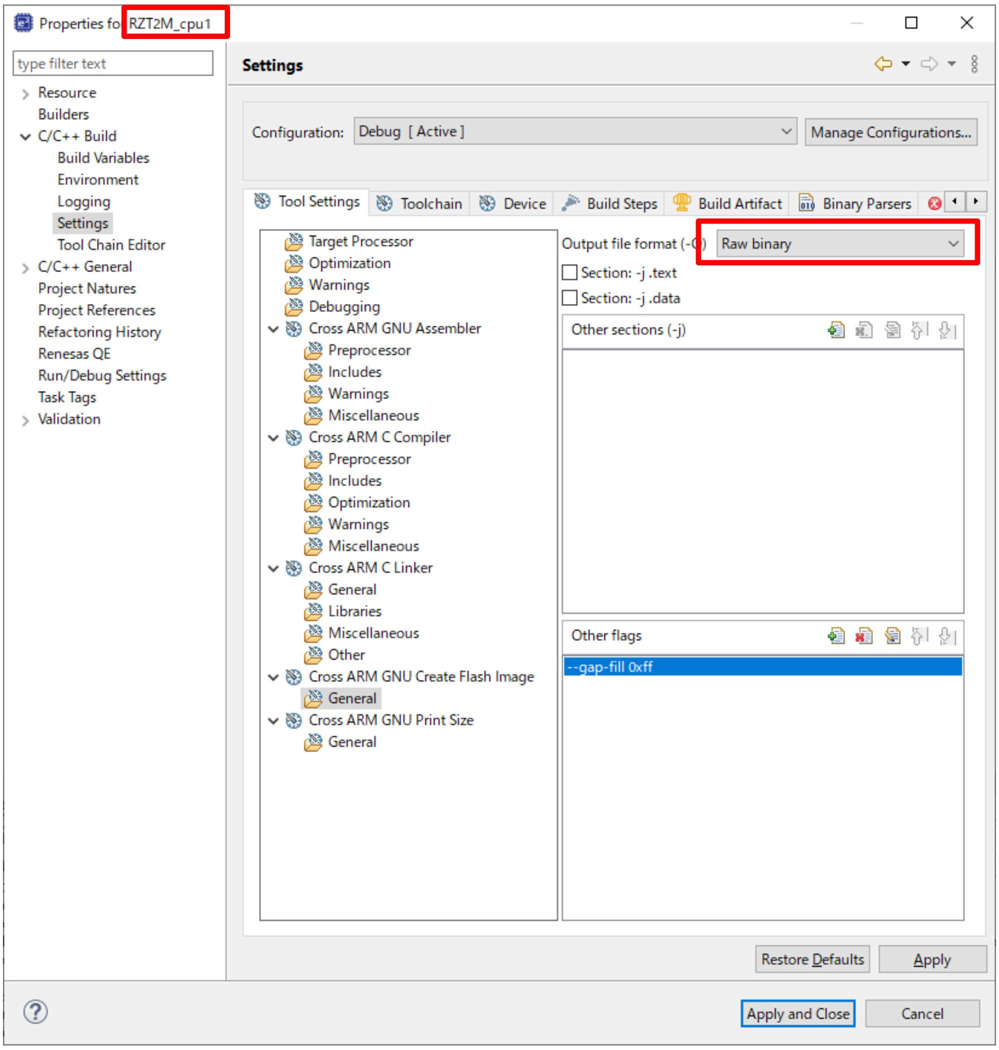
After rebuilding the loader program project, select "Download and Debug" to start debugging the loader program.

During the debugging connection, workspace for the CPU1 project automatically opens as well.
Thereafter, CPU0 and CPU1 projects can be debugged.
When the CPU1 program is executed, LED2 and LED3 blink.

5.3.2 e2studio from Renesas

1. Build CPU1 program

   Build the “**RZT2M_cpu1**” program. The following project option settings will output build artifacts in raw binary.



2. Link CPU1 program to CPU0 loader program

Add a section definition to the linker script file to link the CPU1 binary to the CPU0 loader program.

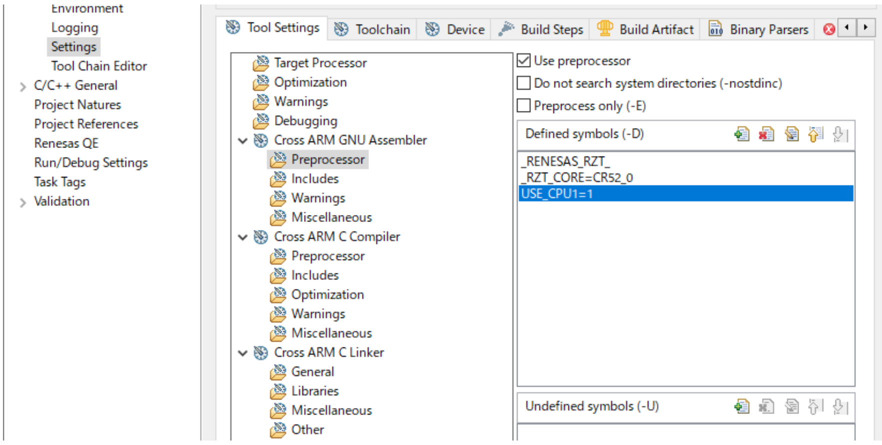
fsp_xspi0_boot_loader.ld (Linker script for the loader program)

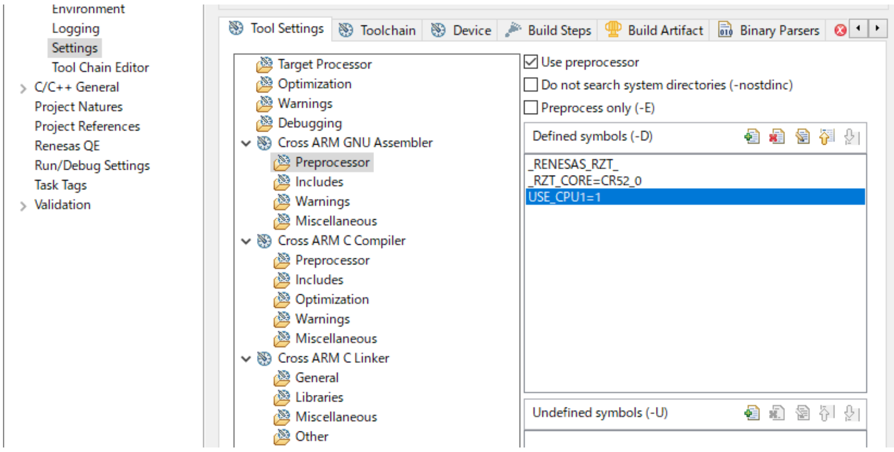
```
SECTIONS
{
    .IMAGE_APP_RAM 0x10080000 : AT (0x10080000)
    {
        IMAGE_APP_RAM_start = .;
        KEEP(*(APP_IMAGE_RAM))
    }
    .IMAGE_APP_FLASH_section 0x60100000 : AT (0x60100000)
    {
        IMAGE_APP_FLASH_section_start = .;
        KEEP(./src/Flash_section.o(.IMAGE_APP_FLASH_section))
        IMAGE_APP_FLASH_section_end = .;
    }
    .IMAGE_CPU1_RAM 0x10000000 : AT (0x10000000) /* CPU1 program, RAM section for execution. */
    {
        IMAGE_CPU1_RAM_start = .;
        KEEP(*(CPU1_IMAGE_RAM))
    }
    .IMAGE_CPU1_FLASH_section 0x60200000 : AT (0x60200000) /* CPU1 program, ROM section. */
    {
        IMAGE_CPU1_FLASH_section_start = .;
        KEEP(./src/Flash_section.o(.IMAGE_CPU1_FLASH_section))
        IMAGE_CPU1_FLASH_section_end = .;
    }
    .loader_param 0x60000000 : AT (0x60000000)
    {
        KEEP(*(.loader_param))
    } > xSPI0_CS0_SPACE
    .flash_contents 0x6000004C : AT (0x6000004C)
    {
        _mtext = .;
        . = . + (_text_end - _text_start);
        _mdata = .;
        . = . + (_data_end - _data_start);
        flash_contents_end = .;
    } > xSPI0_CS0_SPACE
~~~~~~~~
}

IMAGE_APP_FLASH_section_size = SIZEOF(.IMAGE_APP_FLASH_section);
IMAGE_CPU1_FLASH_section_size = SIZEOF(.IMAGE_CPU1_FLASH_section);
```

3. Enable USE_CPU1 definition

Enable the definition to run the CPU1 program. Change the value of "USE_CPU1" defined in the project options of the **loader program** and **application program** from 0 to 1.

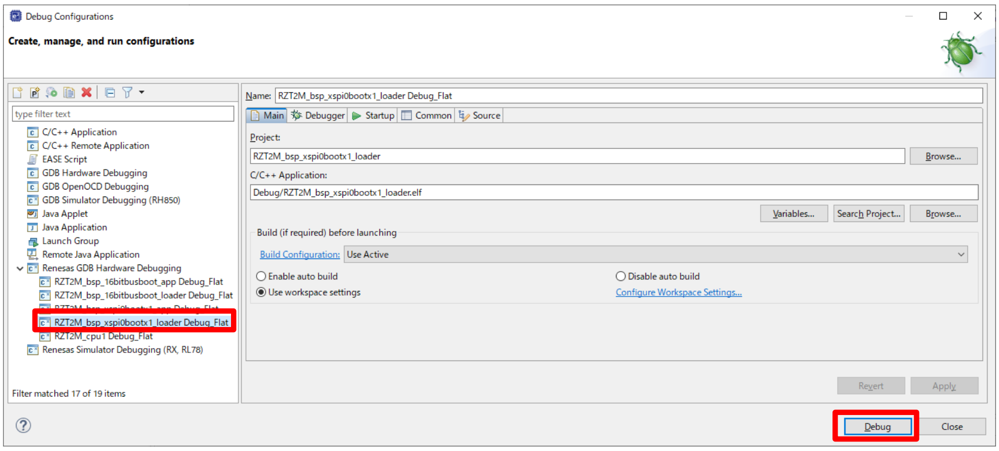
[Properties] -> [C/C++ Build] -> [Settings] -> [Tool Settings]
[Cross ARM GNU Assembler] -> [Preprocessor]: USE_CPU1=1
[Cross ARM C Compiler] -> [Preprocessor]: USE_CPU1=1



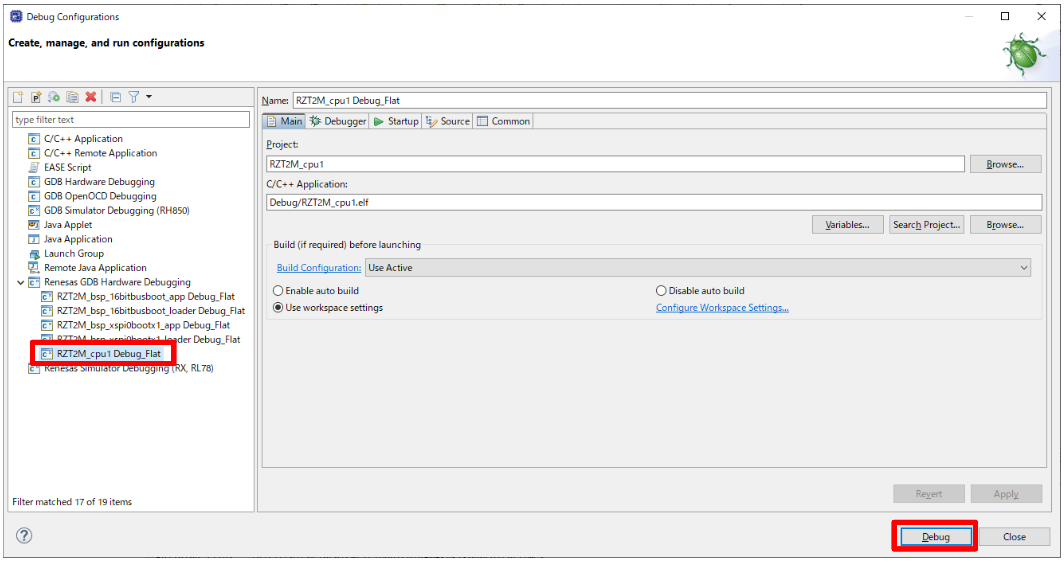


4. Rebuild and run the project

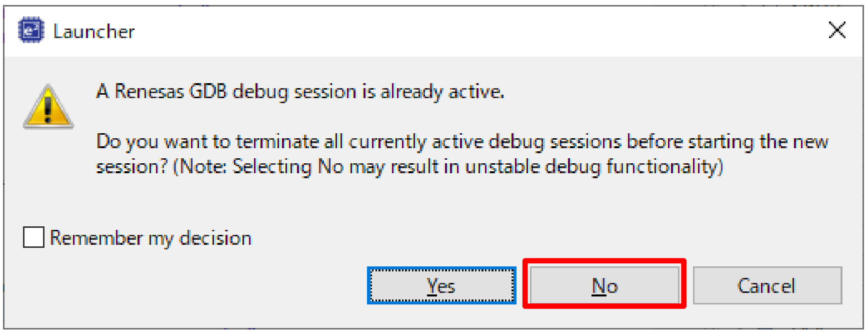
After rebuilding the loader program project, start debugging the loader program. The CPU1 program is also written to flash memory at the same time during the debug connection.



When debugging the CPU1 program, start the debugging connection of the CPU1 project after making the debugging connection of the loader program.

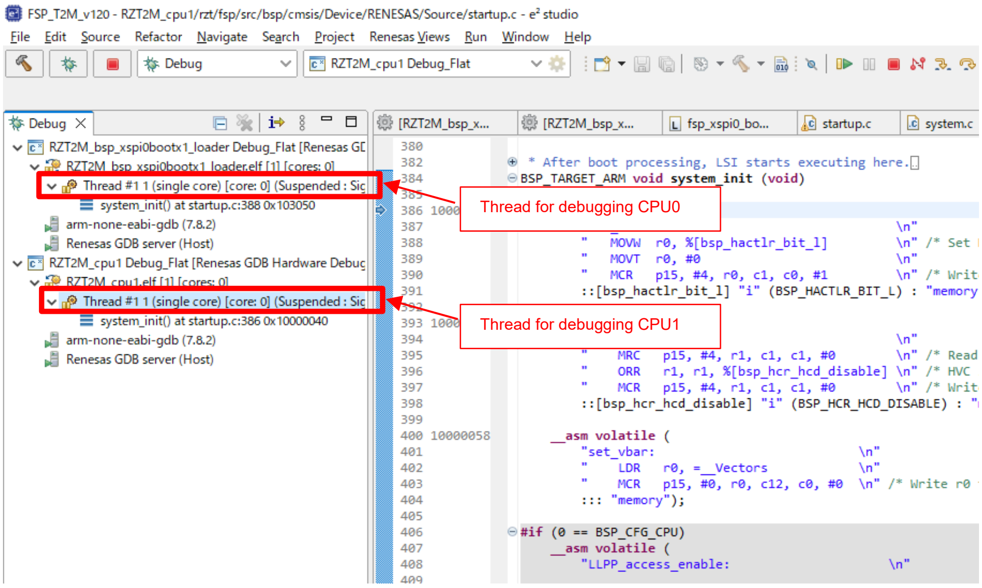


When the following message is displayed, select "No".



If the debugging connection of the CPU1 project succeeds, CPU0 and CPU1 are connected to the debugger. Thereafter, CPU0 and CPU1 projects can be debugged.

By selecting Thread in the "Debug" view on the left side of the screen, the debug target core can be switched between CPU0 and CPU1.



When the thread for debugging CPU0 project is selected and the program is executed, the loader program and the application program runs and LED0 and LED1 blink.
When the thread for debugging CPU1 project is selected and the program is executed, LED2 and LED3 blink.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Nov.29, 2022	-	First edition issued.
1.10	Jun. 15, 2023	-	Modify behavior of loader program and application program.
		-	Support CPU1 operation.
		4	Chang RZ/T2 FSP support version v1.0.0 -> v1.2.0
		31	Add “Example of changing RAM placement in application program”
1.20	Jul. 26, 2024	4	Change RZ/T2 FSP support version v1.2.0 -> v2.0.0 Change the version of the integrated development environment.
		-	Deleted the mpu_cache_init function.
		12、13	Change to Table 3.2, Table 3.3, and Table 3.5
		17	Changed bsp_copy_4byte to bsp_copy_multibyte.

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

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

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

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

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

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

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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