RZ/T2H 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 xSPI1 boot mode (x1 boot serial 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/T2H 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/T2H Group (R9A09G077M44GBG)
Operating Frequency	CPU core0: 1000MHz (Arm® Cortex®-R52) CPU core0: 1200MHz (Arm® Cortex®-A55)*1
Operating Voltage	3.3V / 1.8V / 1.1V
Integrated Development Environment	• Embedded Workbench® for Arm Version 9.60.2 + patch from IAR systems • e² studio 2024-10 (24.10.0) (R20241003-1714) from Renesas
Operating mode	• xSPI0 boot mode (x1 serial flash) • xSPI1 boot mode (x1 serial flash)
Board	RZ/T2H Evaluation Board
Flexible Software Package (FSP)	Version 2.2.0 (RZ/T2 FSP)

Note 1. When using Cortex-A55 CPU core0, refer to "5.3 How to Debug Cortex-A55 CPU0 Program".

1.2 File Structure

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

```
RZT2H_loader_application
├──r01an7758jj0200-rzt2h.pdf
├──r01an7758ej0200-rzt2h.pdf
├──iccarm: for EWARM
│   ├──xspi0bootx1: sample program for SPI0 flash
│   │   └──Loader_application_projects.zip
│   │       ├──RZT2H_bsp_xspi0bootx1_app: project for application program
│   │       ├──RZT2H_bsp_xspi0bootx1_loader: project for loader program
│   │       ├──RZT2H_bsp_xspi0bootx1_app_CA55_0: project for Cortex-A55 CPU0 program
│   │       ├──setting : workspace settings file
│   │       ├──multicore_setup.xml : multicore debugging configuration file
│   │       └──RZT2H_bsp_xspi0bootx1_separating_loader.eww: EWARM workspace
│   └──xspi1bootx1: sample program for SPI1 flash
│       └──Loader_application_projects.zip
│           ├──RZT2H_bsp_xspi1bootx1_app: project for application program
│           ├──RZT2H_bsp_xspi1bootx1_loader: project for loader program
│           ├──RZT2H_bsp_xspi1bootx1_app_CA55_0: project for Cortex-A55 CPU0 program
│           ├──setting : workspace settings file
│           ├──multicore_setup.xml : multicore debugging configuration file
│           └──RZT2H_bsp_xspi1bootx1_separating_loader.eww: EWARM workspace
└──gcc : for e² studio
    ├──xspi0bootx1: sample program for SPI0 flash
    │   └──Loader_application_projects.zip
    │       ├──RZT2H_bsp_xspi0bootx1_app: project for application program
    │       ├──RZT2H_bsp_xspi0bootx1_loader: project for loader program
    │       └──RZT2H_bsp_xspi0bootx1_app_CA55_0: project for Cortex-A55 CPU0 program
    └──xspi1bootx1: sample program for SPI1 flash
        └──Loader_application_projects.zip
            ├──RZT2H_bsp_xspi1bootx1_app: project for application program
            ├──RZT2H_bsp_xspi1bootx1_loader: project for loader program
            └──RZT2H_bsp_xspi0bootx1_app_CA55_0: project for Cortex-A55 CPU0 program
```

The files of the package are separated to EWARM and e2 studio environment at first level, and to SPI0 flash and SPI1 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 Cortex-A55 CPU0 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 RZ/T2H Evaluation Board User's Manual.

Table 1-2 Switch settings (1/2)

Project	SW14-1	SW14-2	SW14-3	SW14-4	SW14-6
xSPI0 boot mode	ON	ON	ON	OFF	OFF
xSPI1 boot mode	ON	OFF	ON	OFF	OFF

Table 1-3 Switch settings (2/2)

Project	SW1-6	SW5-5	SW5-6	SW8-9	SW8-10
xSPI0 boot mode	-	OFF	ON	ON	OFF
xSPI1 boot mode	ON	-	-	ON	OFF

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)
Expanded serial peripheral interface (xSPI)	Used to attach Serial flash memory to external address space xSPI0 and xSPI1
General purpose I/O ports	Used to control pins to light LEDs on and off

See the RZ/T2H, RZ/N2H 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
XSPI0_RESET0#	Output	Master reset status output for slave 0
XSPI0_CS0#	Output	Device selection signal output to OSPI flash memory attached to CS0 space
XSPI0_DS	Input/Output	Read Data Strobe / Write Data Mask
XSPI0_ECS#	Input	Slave 0 error correction status input
XSPI0_CKP	Output	Clock Positive Output
XSPI0_CKN	Output	Clock Negative Output
XSPI0_IO0 ~ XSPI0_IO7	Input/Output	Data input / output
XSPI1_CKP#	Output	Clock output
XSPI1_CS0#	Output	Device selection signal output to QSPI flash memory attached to CS0 space
XSPI1_IO0 ~ XSPI1_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” (xSPI1 boot mode)
MD1	Input	
MD2	Input	
P23_1	Output	Lighting LED0 on and off
P06_7	Output	Lighting LED2 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 (xSPI0 boot /xSPI1 boot) stored on the external flash memory (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 (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.

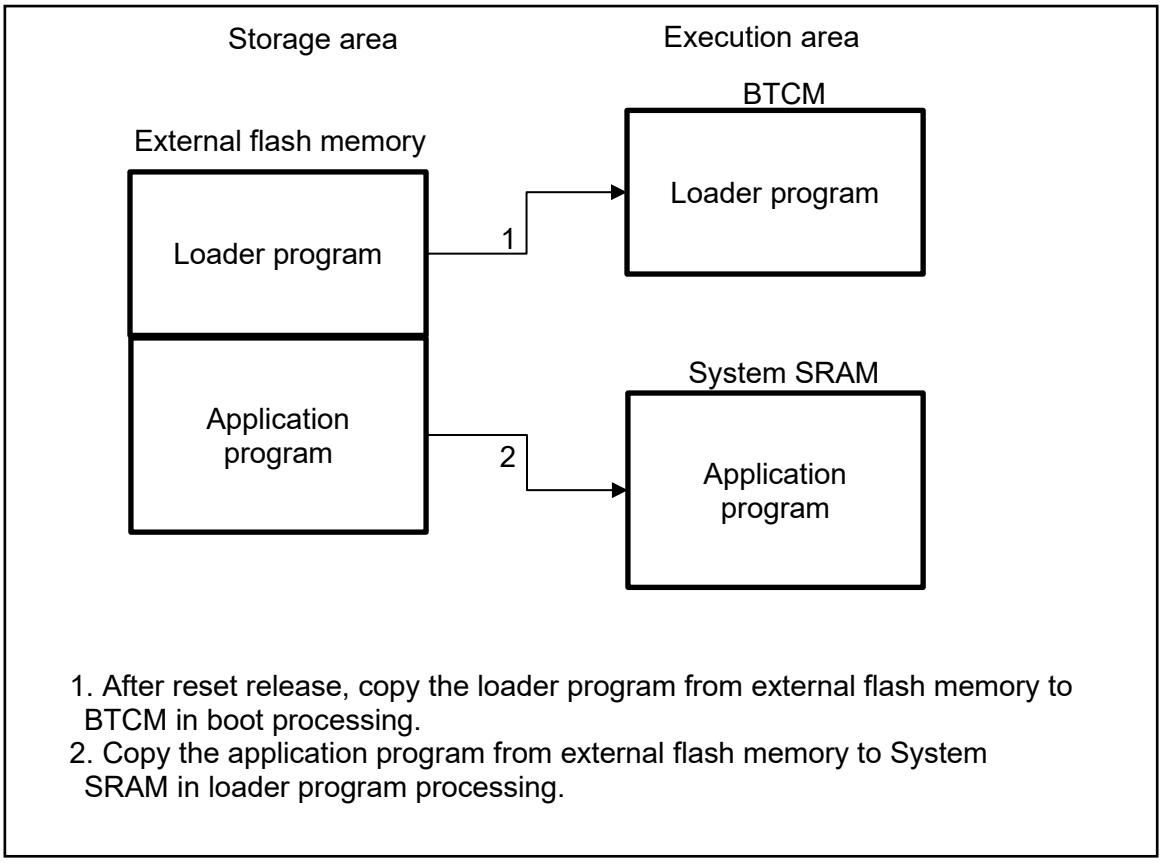

Figure 3-1 Operation overview

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 (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 LED2 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.

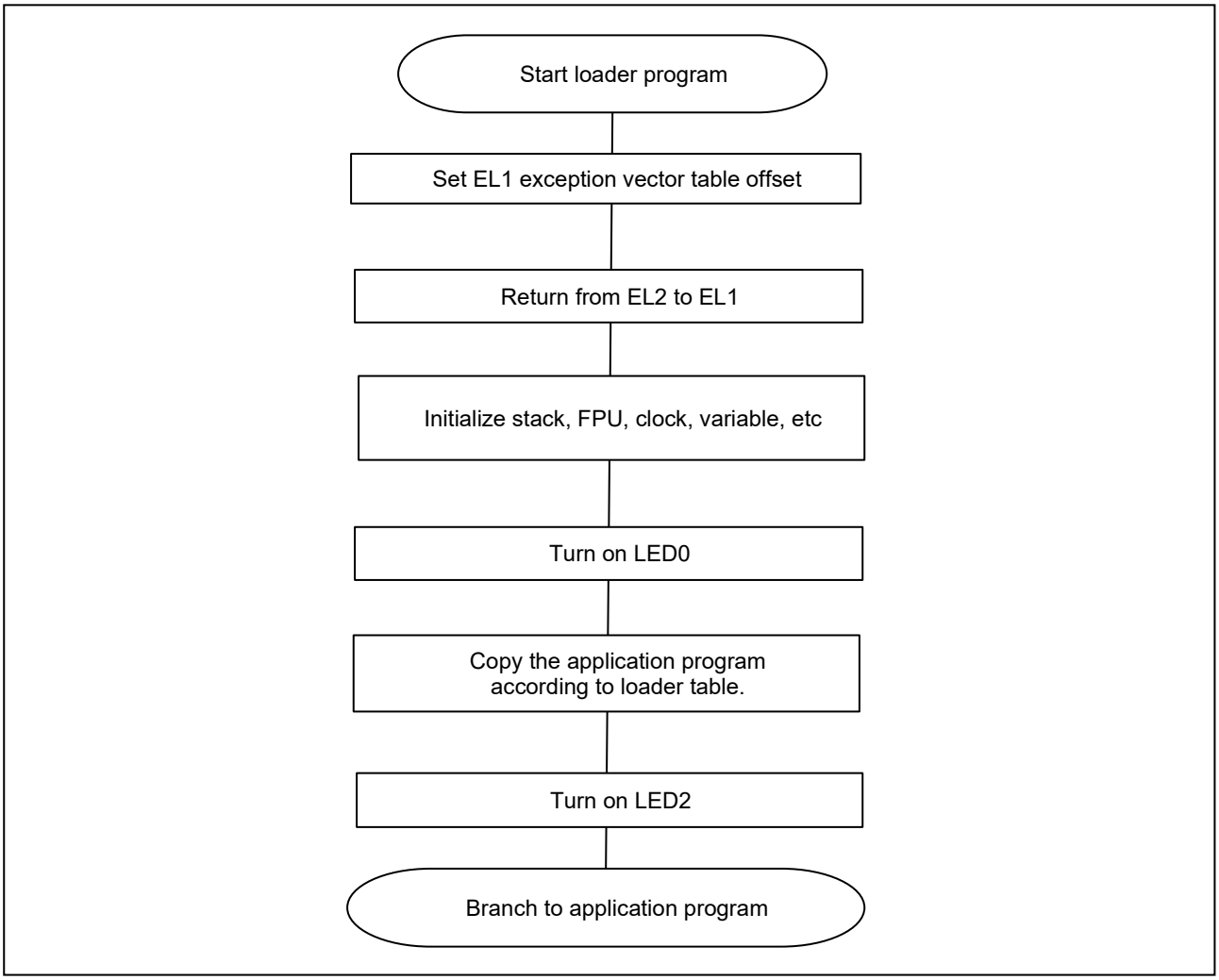

Figure 3-2 Operation overview of loader program

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 LED2, 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 blink.

Figure 3-3 shows operation overview of application program.

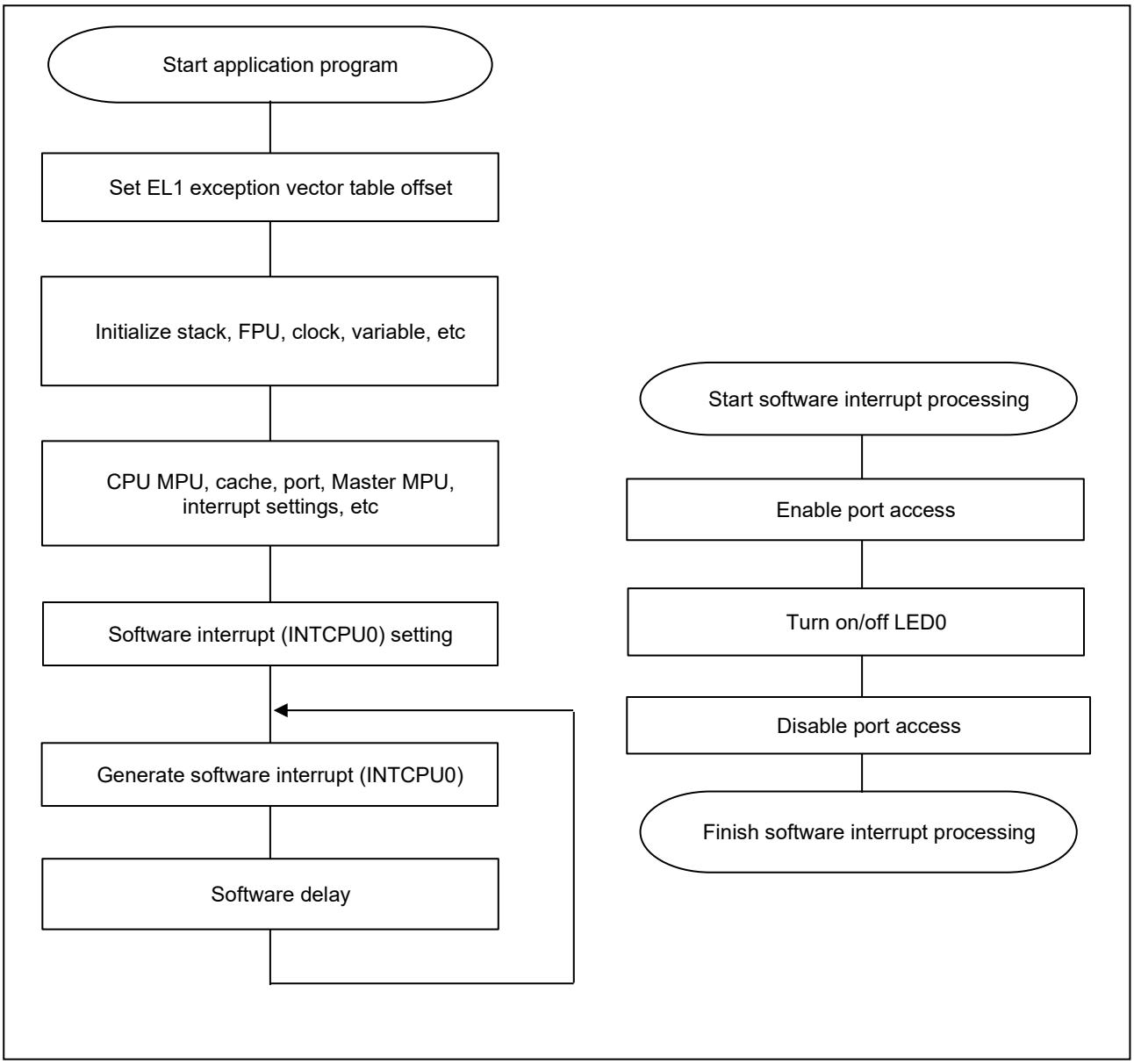

Figure 3-3 Operation overview of application program

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. Table 3-2 and Table 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	0x4001_0000	0x1008_0000	0x0000_3670	0x1
1[*1*2]	0xFFFF_FFFF or 0x4004_0000	0xFFFF_FFFF or 0x1000_0000	0xFFFF_FFFF or 0x0000_A0A8	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 Cortex-A55 CPU0 program is enabled, Table1 hold the parameters for Cortex-A55 CPU0 program. For details, see "5.3 How to Debug Cortex-A55 CPU0 Program".

Table 3-3 Loader table parameters in this sample program (xSPI1 boot mode)

Table	Src	Dst	Size	Enable flag
0	0x5001_0000	0x1008_0000	0x0000_3670	0x1
1	0xFFFF_FFFF or 0x5004_0000	0xFFFF_FFFF or 0x1000_0000	0xFFFF_FFFF or 0x0000_A0A8	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 Cortex-A55 CPU0 program is enabled, Table1 hold the parameters for Cortex-A55 CPU0 program. For details, see "5.3 How to Debug Cortex-A55 CPU0 Program".

3.3 Memory Map

3.3.1 Program Placement in Flash Memory

Table 3-4 and Table 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 copied 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
0x4000_0000	Parameters for the loader	-
0x4000_0050	Loader program	0x0010_2000 (BTCM)
0x4008_0000	Loader table	-
0x4001_0000	Application program	0x1008_0000 (System SRAM)
0x4004_0000*1	Cortex-A55 CPU0 program	0x1000_0000 (System SRAM)

Note 1. Cortex-A55 CPU0 program is disabled by default. To enable it, see "5.3 How to Debug Cortex-A55 CPU0 Program".

Table 3-5 Program placement in flash memory and load destination address (xSPI1 boot mode)

Flash memory address	Contents	Load destination address
0x5000_0000	Parameters for the loader	-
0x5000_0050	Loader program	0x0010_2000 (BTCM)
0x5008_0000	Loader table	-
0x5001_0000	Application program	0x1008_0000 (System SRAM)
0x5004_0000	Cortex-A55 CPU0 program	0x1000_0000 (System SRAM)

Note 1. Cortex-A55 CPU0 program is disabled by default. To enable it, see "5.3 How to Debug Cortex-A55 CPU0 Program".

3.3.2 Section Assignment in Sample Program

3.3.2.1 EWARM

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 (EWARM)

Area Name	Description	Storing/Execution Area*1
LOADER_PARAM_BLOCK	Parameters for the loader	Flash
PRG_RBLOCK	Code area (for storing)	Flash
USER_DATA_RBLOCK	Variable area (for storing)	Flash
PRG_WBLOCK	Code area (for execution)	BTCM
USER_DATA_WBLOCK	Variable with initial value area (for execution)	BTCM
USER_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
CA55_CPU0_RBLOCK*2	Cortex-A55 CPU0 program area (for storing)	Flash
CA55_CPU0_PRG_WBLOCK*2	Cortex-A55 CPU0 program area (for execution)	System SRAM

Note 1. In xSPI0, xSPI1 bus boot, serial flash memory is storing area.
2. Cortex-A55 CPU0 program is disabled by default. To enable it, see "5.3 How to Debug Cortex-A55 CPU0 Program".

Table 3-7 Sections used by application program (EWARM)

Area name	Description	Storing/Execution Area*1
PRG_RBLOCK	Code area (for storing)	Flash
USER_DATA_RBLOCK	Variable area (for storing)	Flash
PRG_WBLOCK	Code area (for execution)	System SRAM
USER_DATA_WBLOCK	Variable with initial value area (for execution)	System SRAM
USER_DATA_ZBLOCK	Variable without initial value area (for execution)	System SRAM

Note 1. In xSPI0, xSPI1 bus boot, serial flash memory is storing area.

3.3.2.2 e² studio

Table 3-8 shows the memory sections used by the loader program, and Table 3-9 shows the sections used by the application program. These sections are defined in the linker script.

Table 3-8 Sections used by loader program(e² studio)

Area Name	Description	Storing/Execution Area*1
.loader_param	Parameters for the loader	Flash
.flash_contents	Code area (for storing)	Flash
.flash_contents	Variable area (for storing)	Flash
.text .intvec .reset_handler .loader_text .warm_start	Code area (for execution)	BTCM
.data .rodata	Variable with initial value area (for execution)	BTCM
.bss	Variable without initial value area (for execution)	BTCM
.IMAGE_APP_FLASH_section	Application program area (for storing)	Flash
.IMAGE_APP_RAM	Application program area (for executing)	System SRAM
.IMAGE_CPU1_FLASH_section *2	CPU1 program area (for storing)	Flash
.IMAGE_CPU1_RAM *2	CPU1 program area (for execution)	System SRAM
.cpu0_loader_table	Loader table area	Flash

Note 1. In xSPI0, xSPI1 bus boot, serial flash memory is storing area.
2. Cortex-A55 CPU0 program is disabled by default. To enable it, see "5.3 How to Debug Cortex-A55 CPU0 Program".
3. The stack and heap areas are omitted.

Table 3-9 Sections used by application program(e² studio)

Area name	Description	Storing/Execution Area*1
.flash_contents	Code area (for storing)	Flash
.flash_contents	Variable area (for storing)	Flash
.text .intvec .reset_handler .loader_text .warm_start	Code area (for execution)	System SRAM
.data .rodata	Variable with initial value area (for execution)	System SRAM
.bss	Variable without initial value area (for execution)	System SRAM

Note 1. In xSPI0, xSPI1 bus boot, serial flash memory is storing area.
2. The stack and heap areas are omitted.

3.3.3 CPU MPU Settings

Table 3-10 shows the CPU MPU settings for areas accessed by CPU in this sample program. These settings are applied during startup processing of the application program.

Table 3-10 CPU MPU Settings

Contents	Address	Memory type
System SRAM	0x1000_0000 to 0x100F_FFFF	Area 2 Normal, cache enabled, non-shared
System SRAM	0x1010_0000 to 0x101F_FFFF	Area 3 Normal, cache disabled, shared
Extended address space xSPI0, xSPI1 CS0, CS2, CS3, CS5	0x4000_0000 to 0x5FFF_FFFF	Area 6 Normal, cache disabled, shared
R-Bus Non-safety / Safety peripheral modules0	0x8000_0000 to 0x81FF_FFFF	Area 8 Device (nGnRE) , instruction fetch disabled
R-Bus Safety peripheral modules1	0x8800_0000 to 0x89FF_FFFF	Area 9 Device (nGnRE) , instruction fetch disabled

3.3.4 Exception Processing Vector Table

Exception level 1 of RZ/T2H 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 3-11 lists the contents of exceptional processing vector table for this sample program. Modify the setting to suit your needs.

Table 3-11 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
Cortex-A55 CPU0 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 stack initialization. After that, branches to fpu_slavetcm_init process.
Arguments	None
Return value	None
Remarks	None

3.4.3 fpu_slavetcm_init

fpu_slavetcm_init	
Overview	System initialization 3.
Declaration	void fpu_slavetcm_init (void)
Description	Execute FPU setting and so on. After that, branches to SystemInit process.
Arguments	なし
Return value	なし
Remarks	なし

3.4.4 SystemInit

SystemInit	
Overview	System initialization 4.
Declaration	void SystemInit (void)
Description	Executes system initialization such as initializing variables for startup process, CPU MPU, cache, ports, and so on. After that, branches to the main process.
Arguments	なし
Return value	なし
Remarks	なし

3.4.5 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 LED2 after copy processig is complete. • Application program: Blinks LED0 with software interruption (INTCPU0).
Arguments	None
Return value	None
Remarks	None

3.4.6 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 the flowchart of system_init in the loader program.

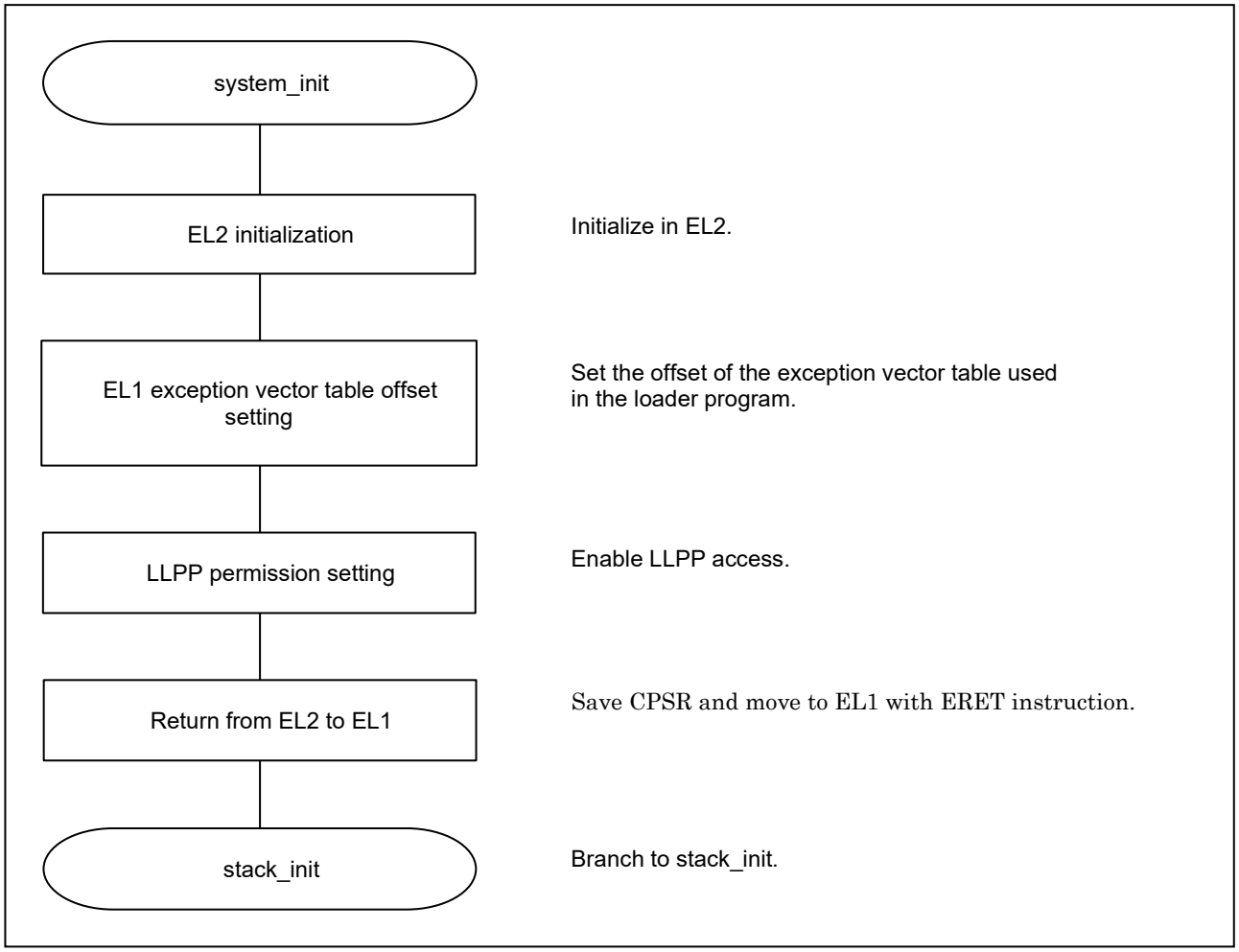

Figure 3-4 system_init processing (loader program)

3.5.1.2 stack_init

Figure 3-5 shows the flowchart of stack_init in the loader program.

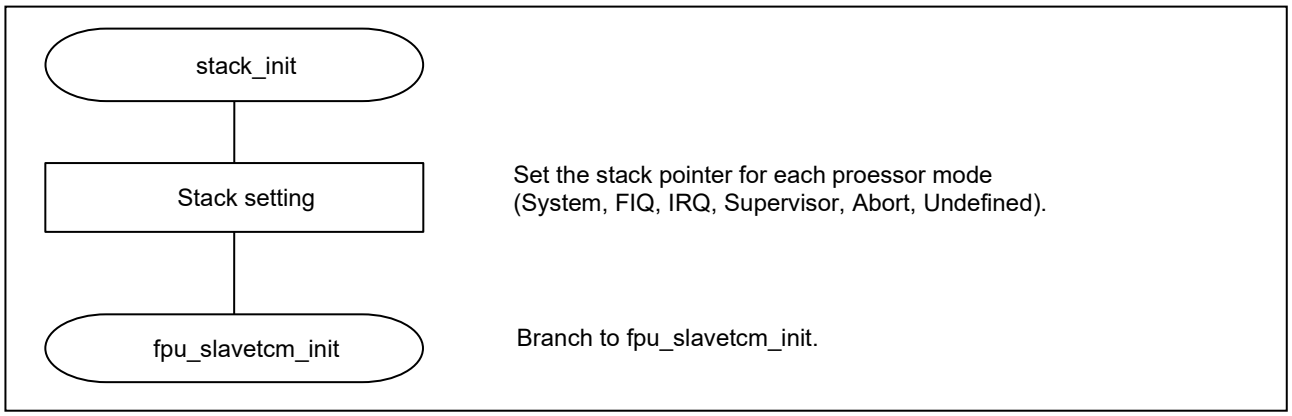

Figure 3-5 stack_init processing (loader program)

3.5.1.3 fpu_slavetcm_init

Figure 3-6 shows the flowchart of fpu_slavetcm_init in the loader program.

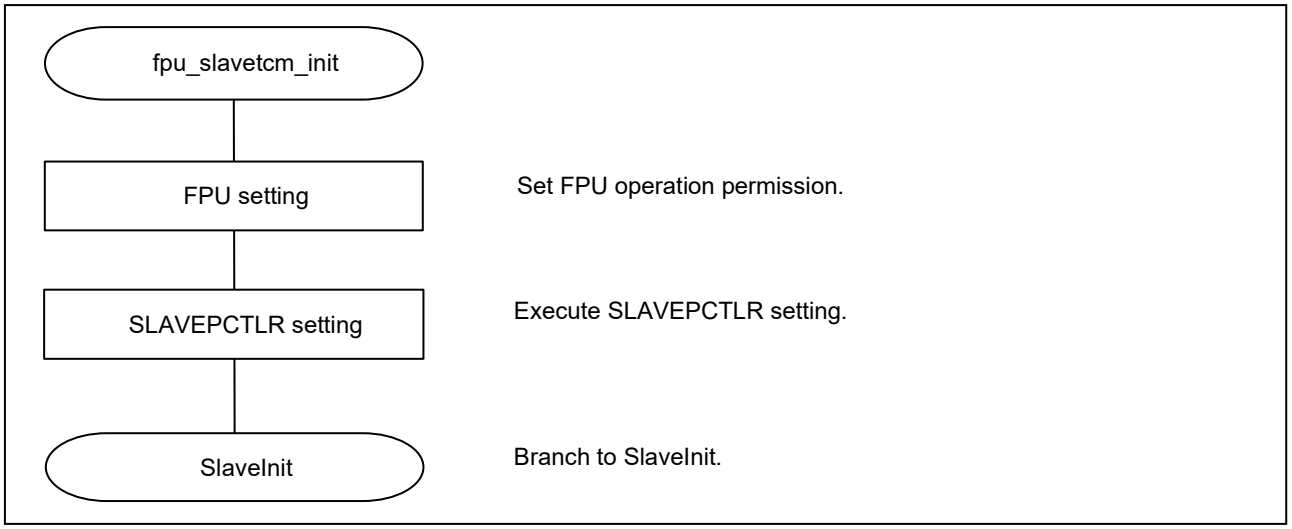

Figure 3-6 fpu_slave_init processing (loader program)

3.5.1.4 SystemInit

Figure 3-7 shows the flowchart of SystemInit in the loader program.

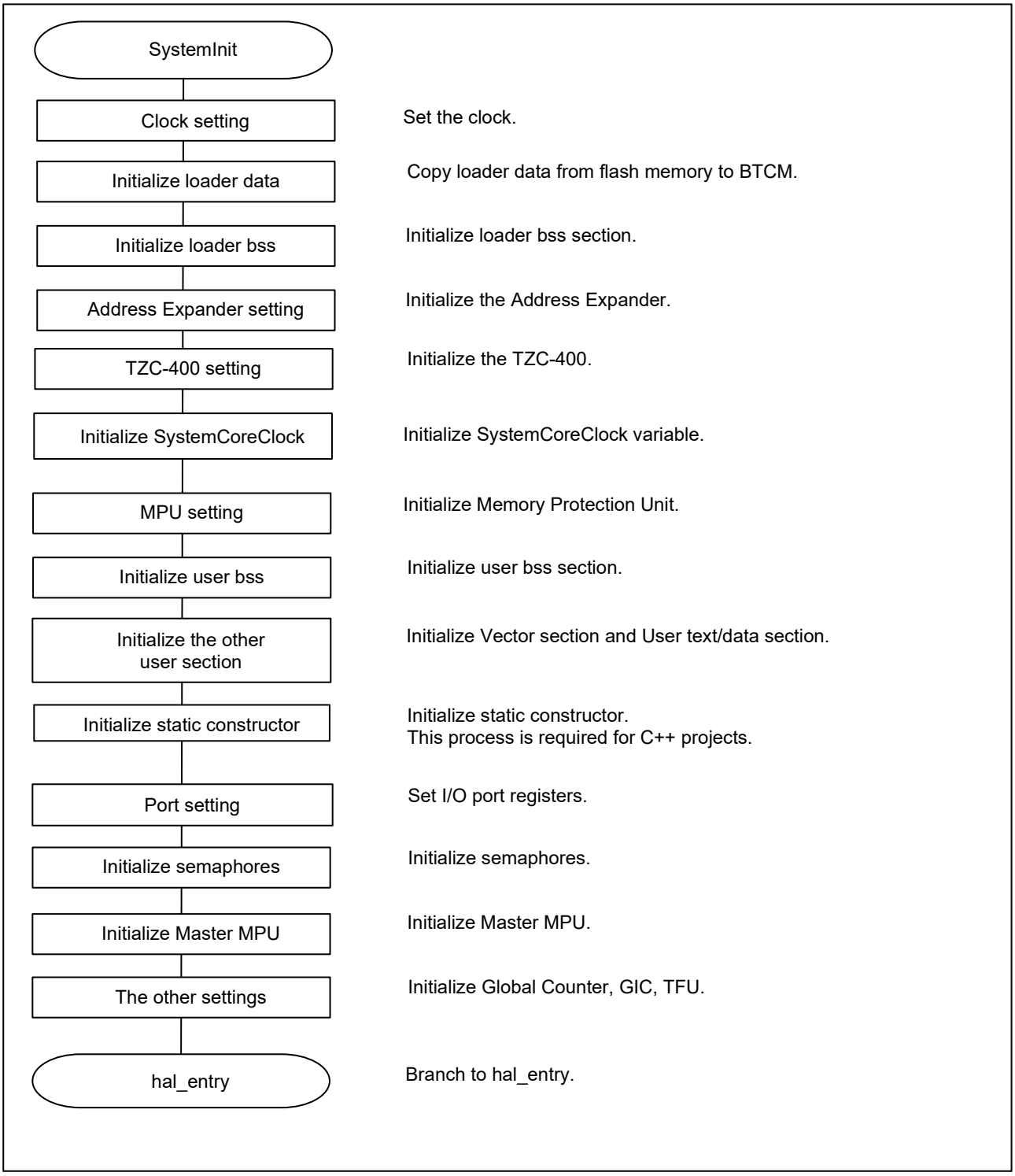

Figure 3-7 SystemInit processing (loader program)

3.5.1.5 hal_entry

Figure 3-8 shows the flowchart of hal_entry in the loader program.

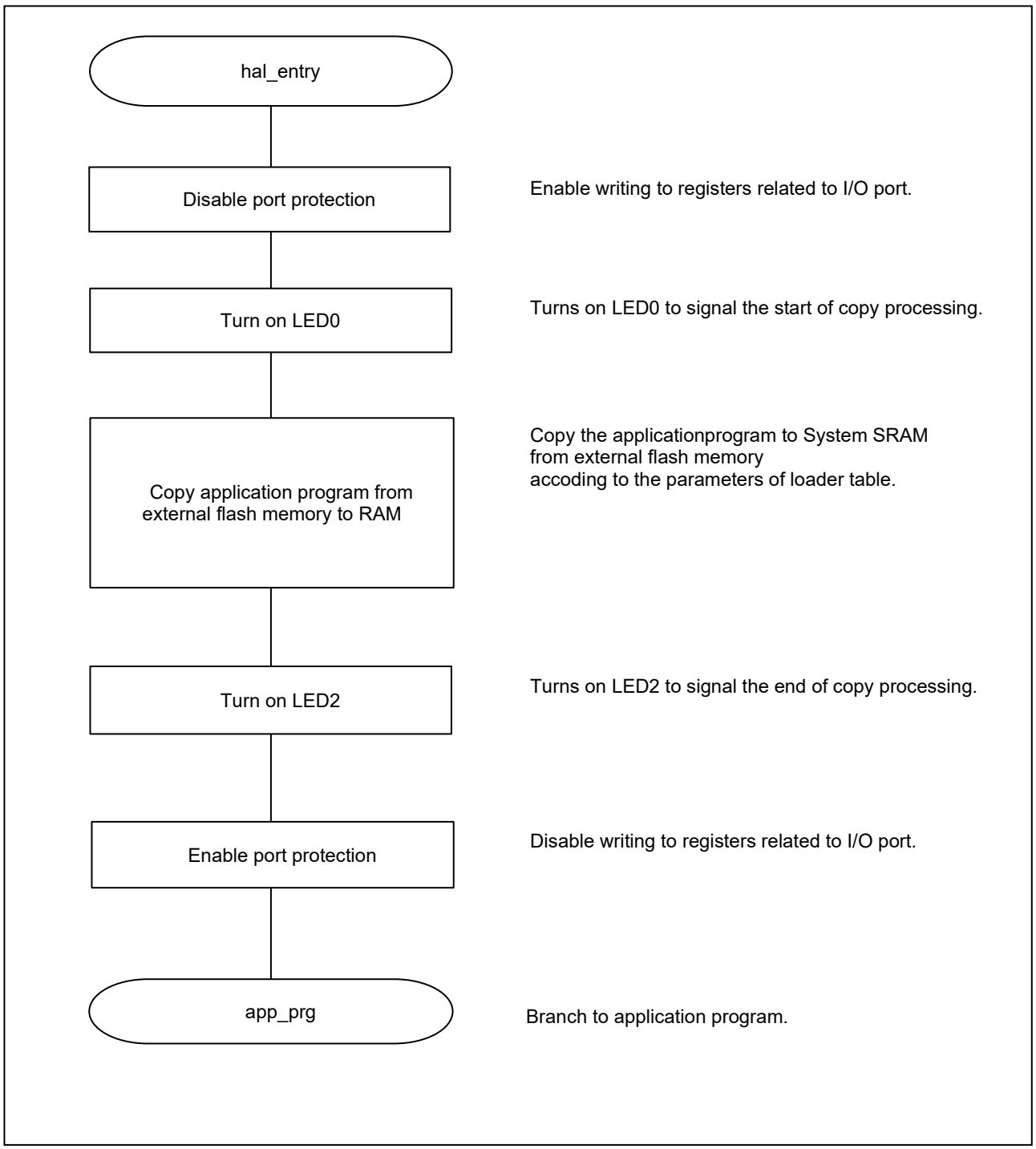

Figure 3-8 hal_entry processing (loader program)

3.5.2 Application Program

3.5.2.1 system_init

Figure 3-9 shows the flowchart of system_init in the application program.

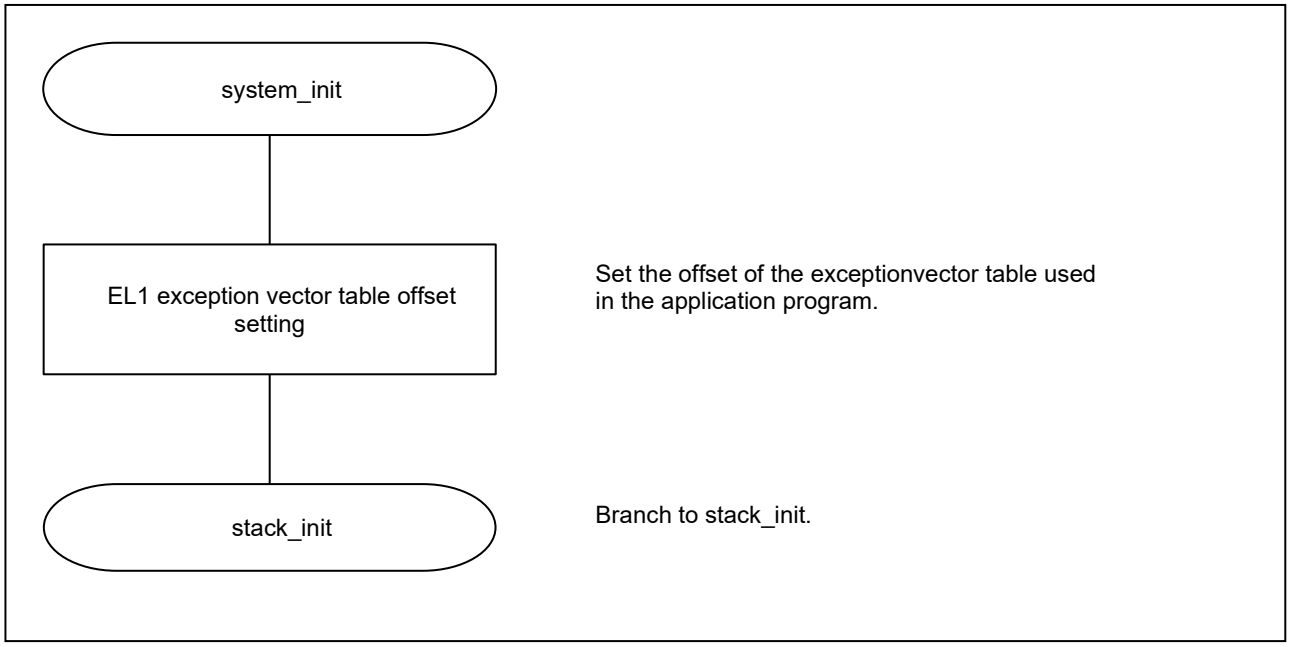

Figure 3-9 system_init processing (application program)

3.5.2.2 stack_init

The flowchart about stack_init on application program is same as loader program (Figure 3-5).

3.5.2.3 fpu_slavetcm_init

The flowchart about spu_slavetcm_init on application program is same as loader program (Figure 3-6).

3.5.2.4 SystemInit

Figure 3-10 shows the flowchart of SystemInit in the loader program.

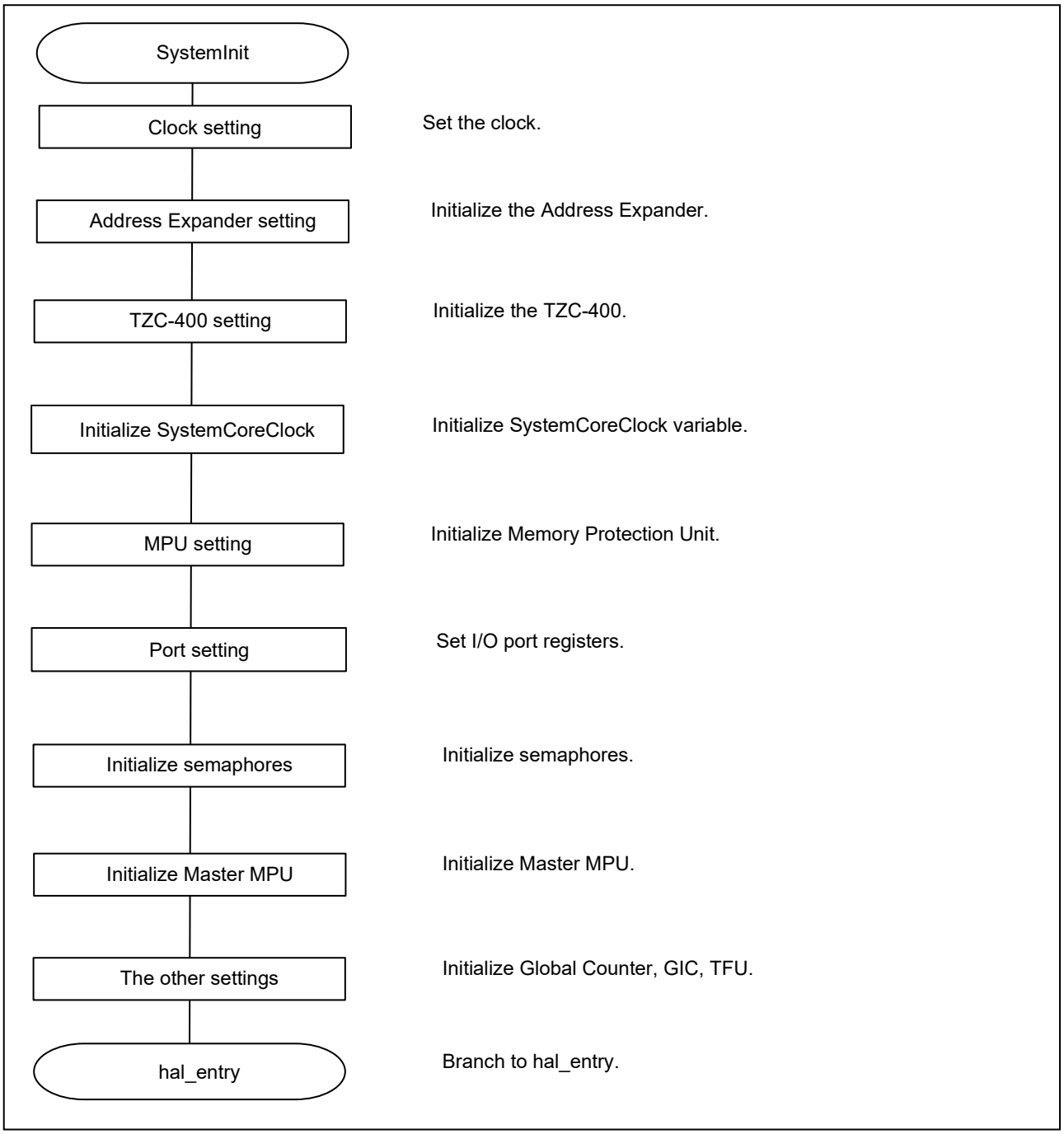

Figure 3-10 SystemInit processing (application program)

3.5.2.5 hal_entry

Figure 3-11 shows the flowchart of hal_entry in the application program.

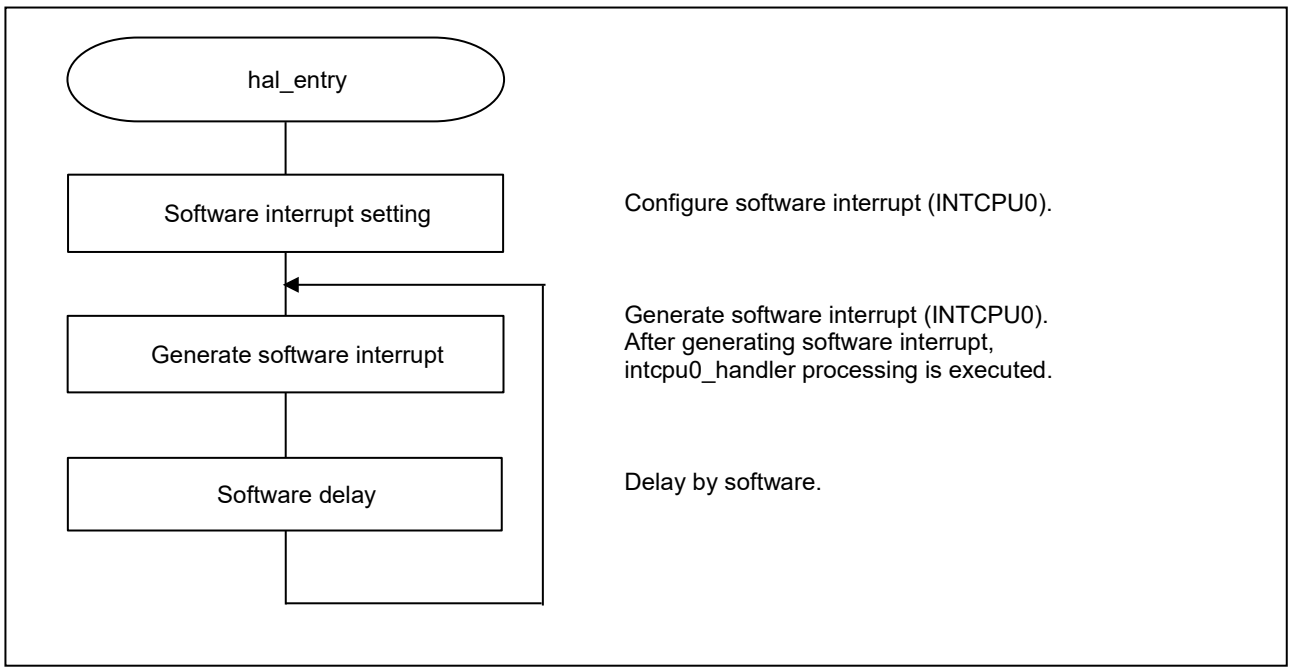

Figure 3-11 hal_entry processing (application program)

Figure 3-12 shows flowchart of interrupt processing in the application program.

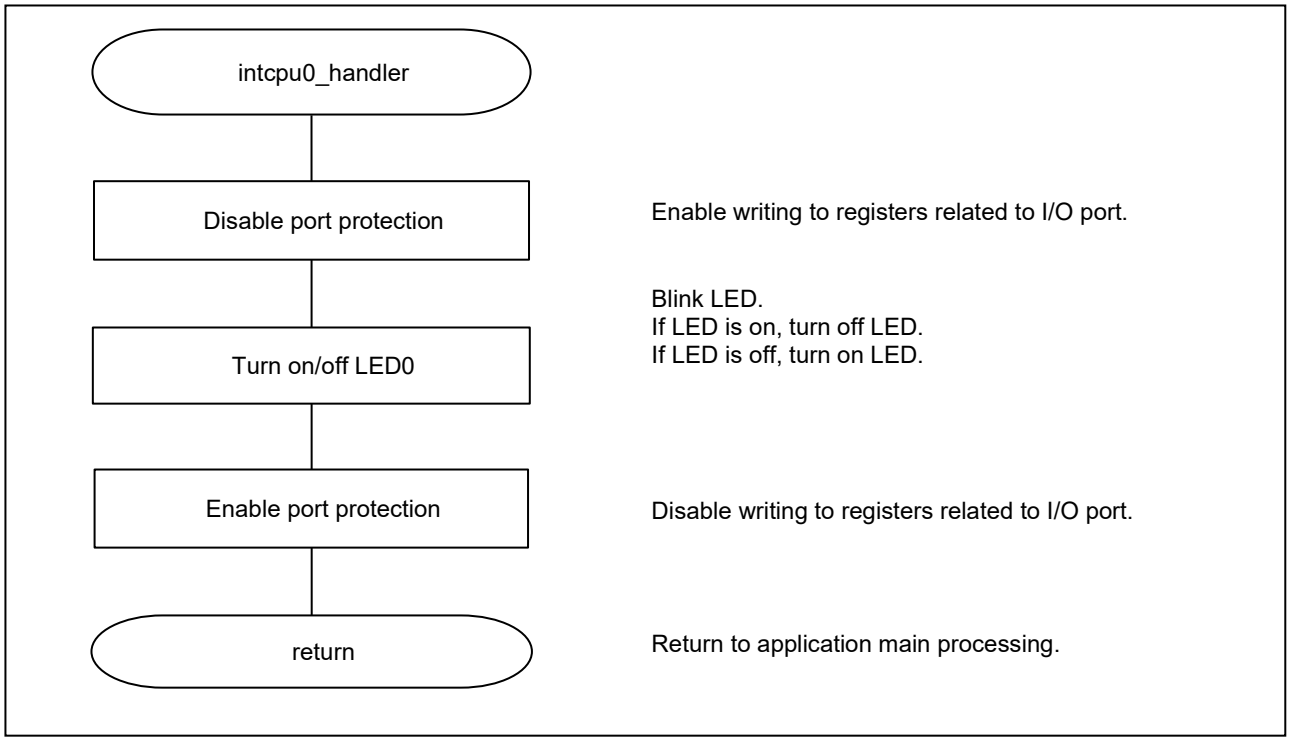

Figure 3-12 interrupt processing (application program)

4. Related Documents

- User’s Manual: Hardware
  RZ/T2H, RZ/N2H Group User's Manual: Hardware
  Download the latest version from the Renesas Electronics website.

  RZ/T2H Evaluation Board User's Manual
  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. The following is an example using the xSPI0 boot mode version.

5.1 Debug procedure for this sample program.

5.1.1 EWARM from IAR systems

1. Launch EWARM and open "RZT2H_bsp_xpi0bootx1_separating_loader.eww" with following procedure.
   "[File] > [Open Workspace] > select Loader_application_projects\RZT2H_bsp_xpi0bootx1_separating_loader.eww"
2. Select "RZ/T2H_bsp_xspi0boot1_app Debug" project in Workspace box as Figure 5-1.

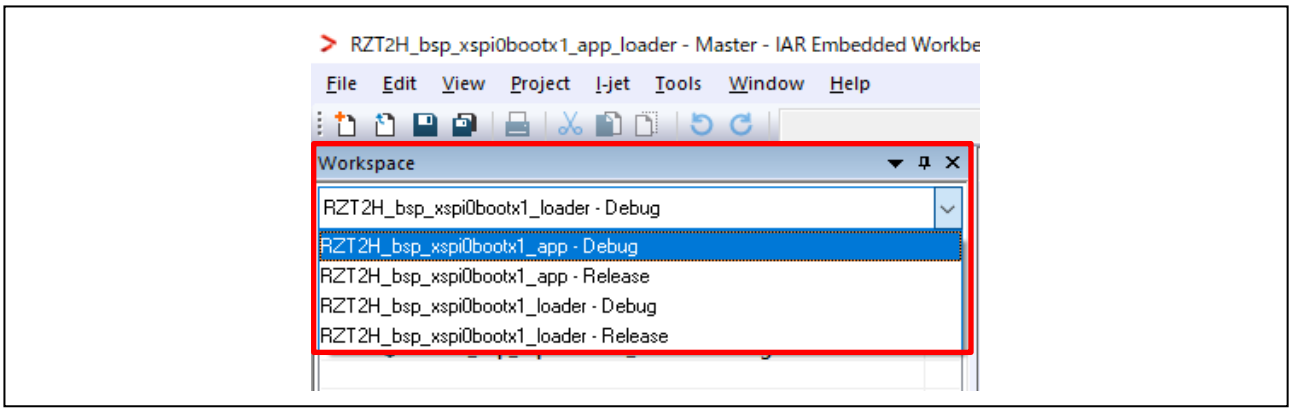

Figure 5-1 Project selection

3. Open FSP Configurator from [Tool], push the [Generate Project Content] button to execute code generation. After that, close the FSP Configurator.

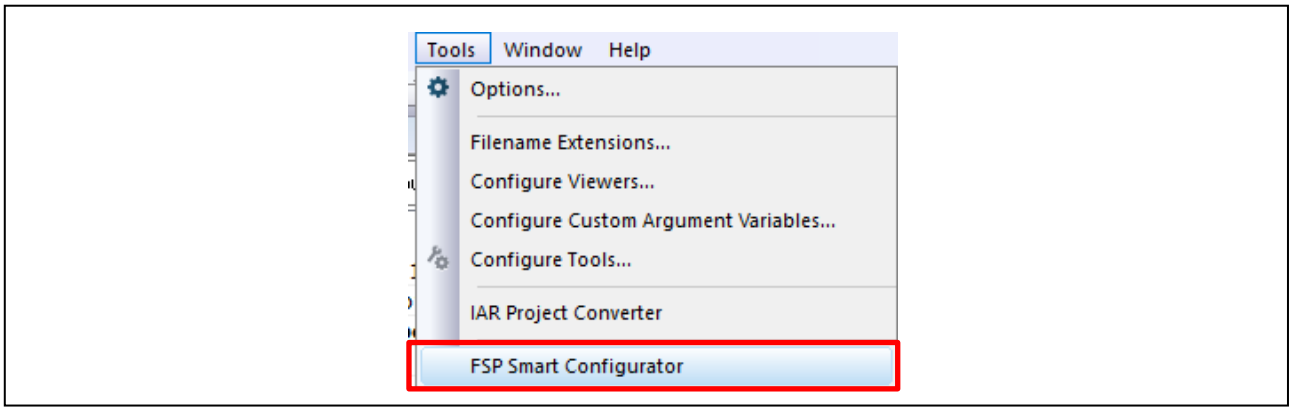

Figure 5-2 Start FSP Configurator

※ Open [Tools] > [Configure Tools…], the FSP to be used must be registered in advance.

Table 5-1 Example of registration in the FSP Smart Configurator

Setting item	Setting example
Menu Text	FSP Smart Configurator
Command	C:\Renesas\rzt\sc_v2024-10_fsp_v2.2.0\eclipse\rasc.exe
Argument	--compiler IAR configuration.xml
Initial Directory	$PROJ_DIR$

4. Open [project] > [Option], and open the [linker] category.

   Set " **$PROJ_DIR$/script/fsp_xspi0_boot_app.icf** " in Linker configuration file.

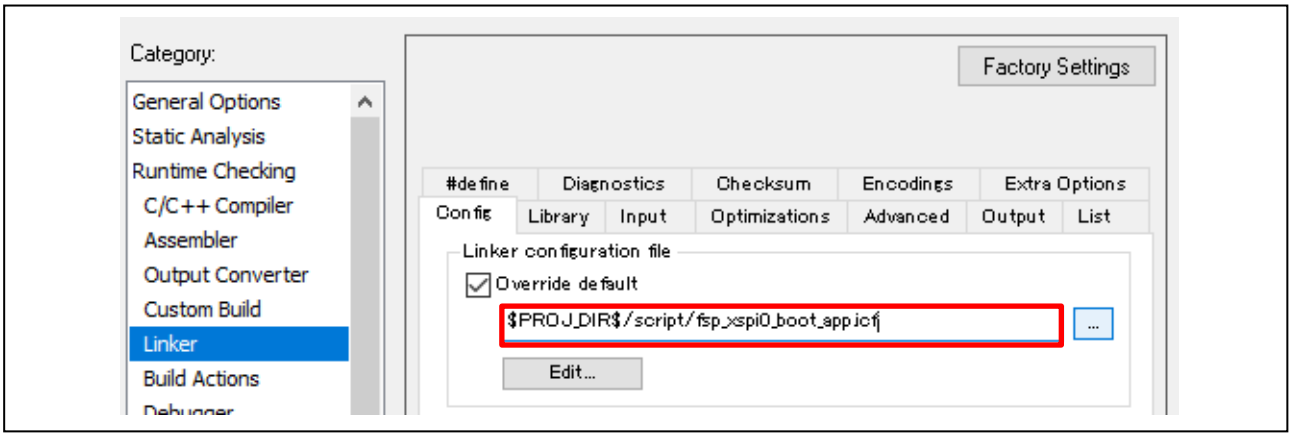

Figure 5-3 Linker script setting (application program)

5. Push "[Project] -> [Rebuild All]" to build the application program.
6. Then, select "RZ/T2H_bsp_xspi0boot1_loader Debug" project in Workspace box.
7. Open FSP Configurator from [Tool], push the [Generate Project Content] button to execute code generation. After that, close the FSP Configurator.
8. Open [project] > [Option], and open the [linker] category.

   Set " **$PROJ_DIR$/script/fsp_xspi0_boot_loader.icf** " in Linker configuration file.

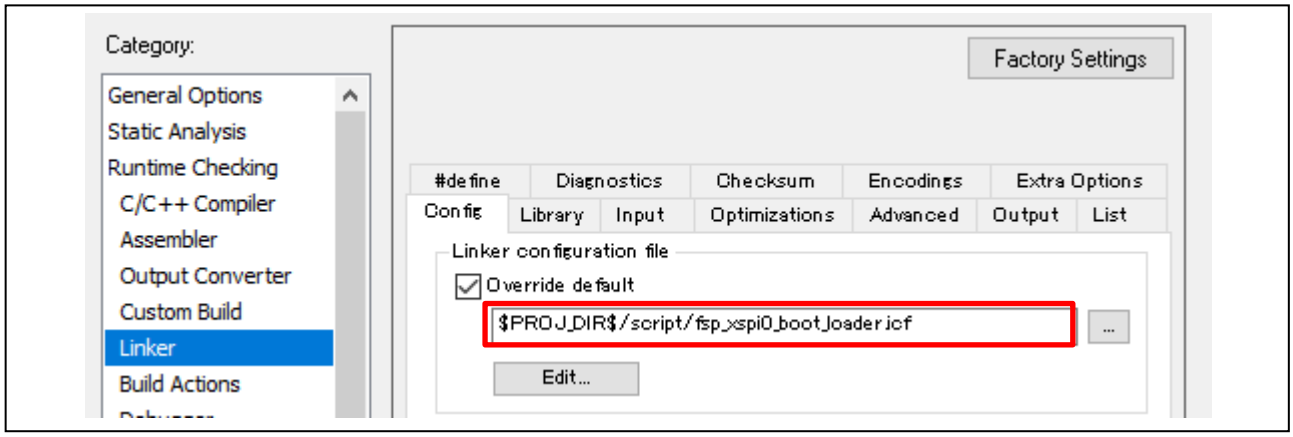

Figure 5-4 Linker script setting (loader program)

9. Push "[Project] -> [Rebuild All]" to build the application program.
10. Make sure that your PC and RZ/T2H evaluation board are connected with I-jet.
    Then, start debugging with "[Project] -> [Download and debug]"
11. 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).

Supplementary Notes for the EWARM Environment (1)

Select loader project when you start debugging as Figure 5-5.
Application program is already specified as extra image in loader project option.
"Right click loader project -> [Options...] -> [Debugger] -> [Images] -> [Download extra image]"

Path:
$PROJ_DIR$\..\RZT2H_bsp_xspi0bootx1_app\Debug\Exe\RZT2H_bsp_xspi0bootx1_app.out

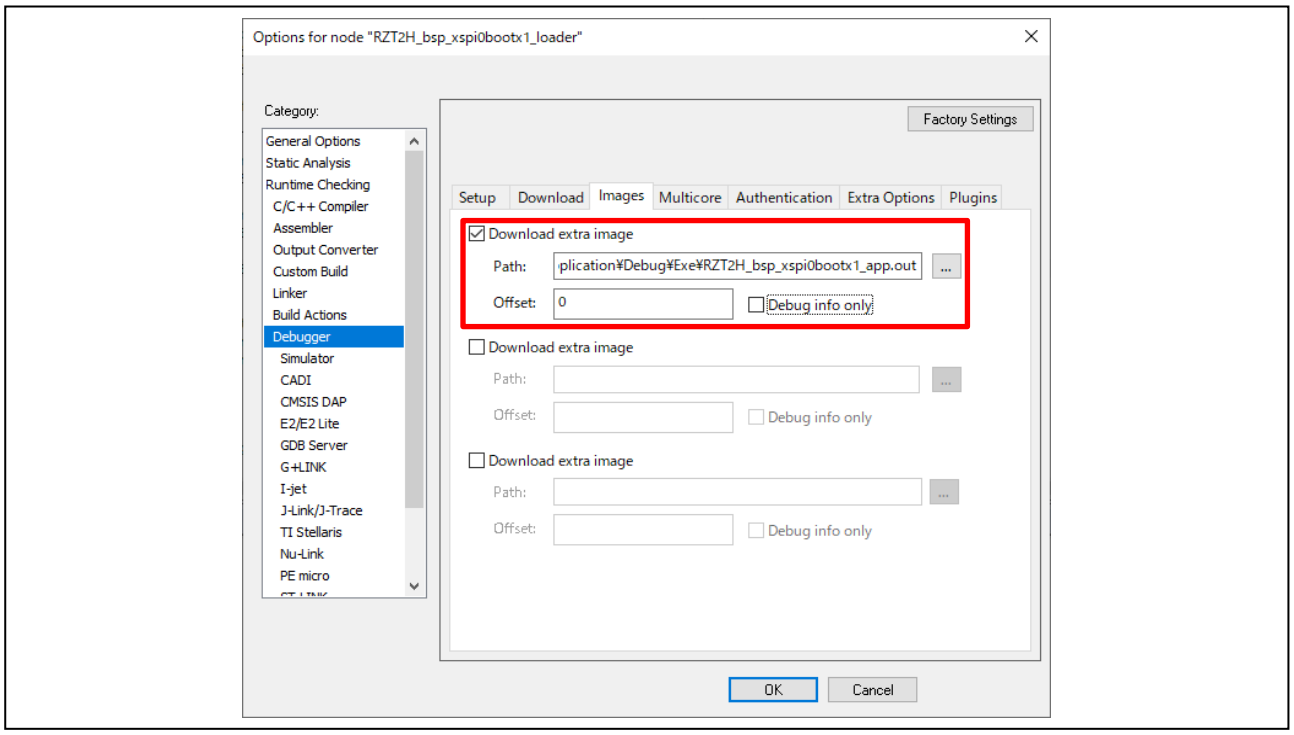

Figure 5-5 EWARM option setting (1)

Supplementary Notes for the EWARM Environment (2)

The following option settings ensure that the application program is built along with the loader program.

Keep symbols:
APPLICATION_PRG_SECTION
File:
$PROJ_DIR$\..\RZT2H_bsp_xspi0bootx1_app\Debug\Exe\RZT2H_bsp_xspi0bootx1_app.bin
Symbol:
APPLICATION_PRG_SECTION
Section:
APPLICATION_PRG_SECTION
Align:
4

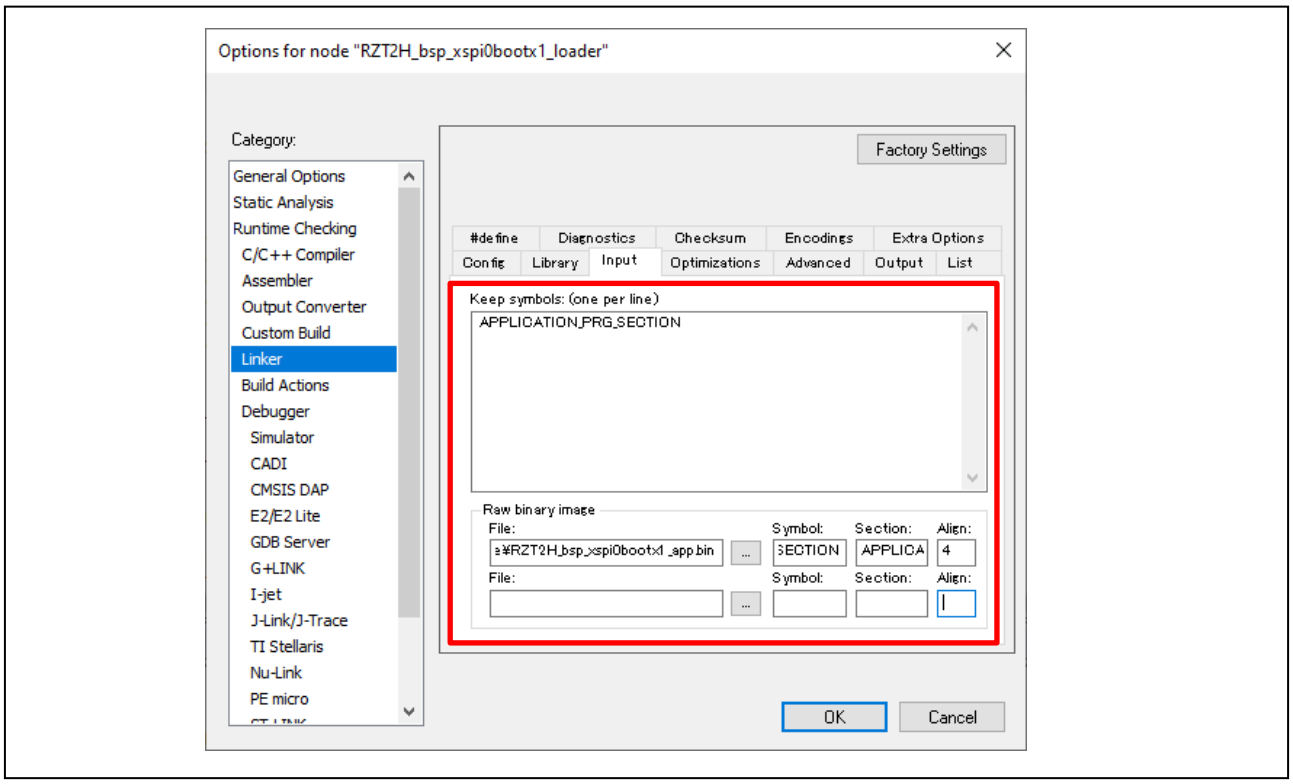

Figure 5-6 EWARM option setting (2)

Supplementary Notes for the EWARM Environment (3)

When starting debug, the following warning about stack plug-in may be displayed.

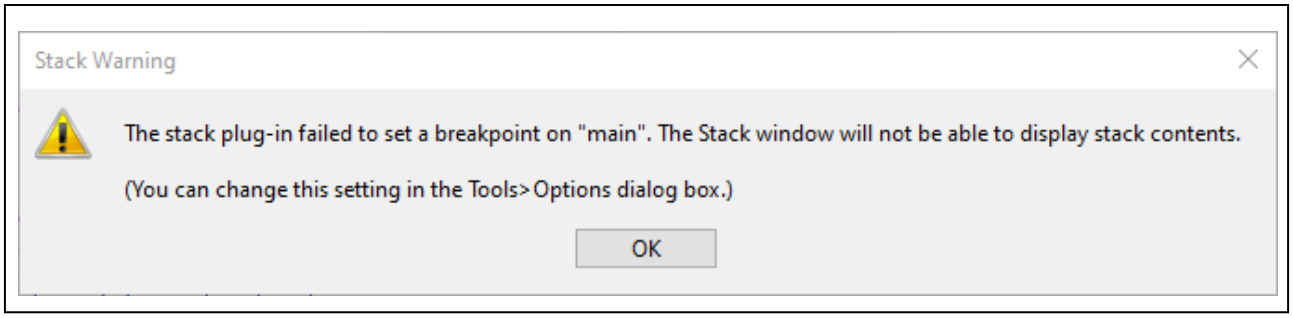

Figure 5-7 Warnig about stack plug-in

Select [Tool] > [Option] > [Stack]、uncheck on [Stack pointer(s) not valid until program reaches]. Then, that warning window will not be displayed.

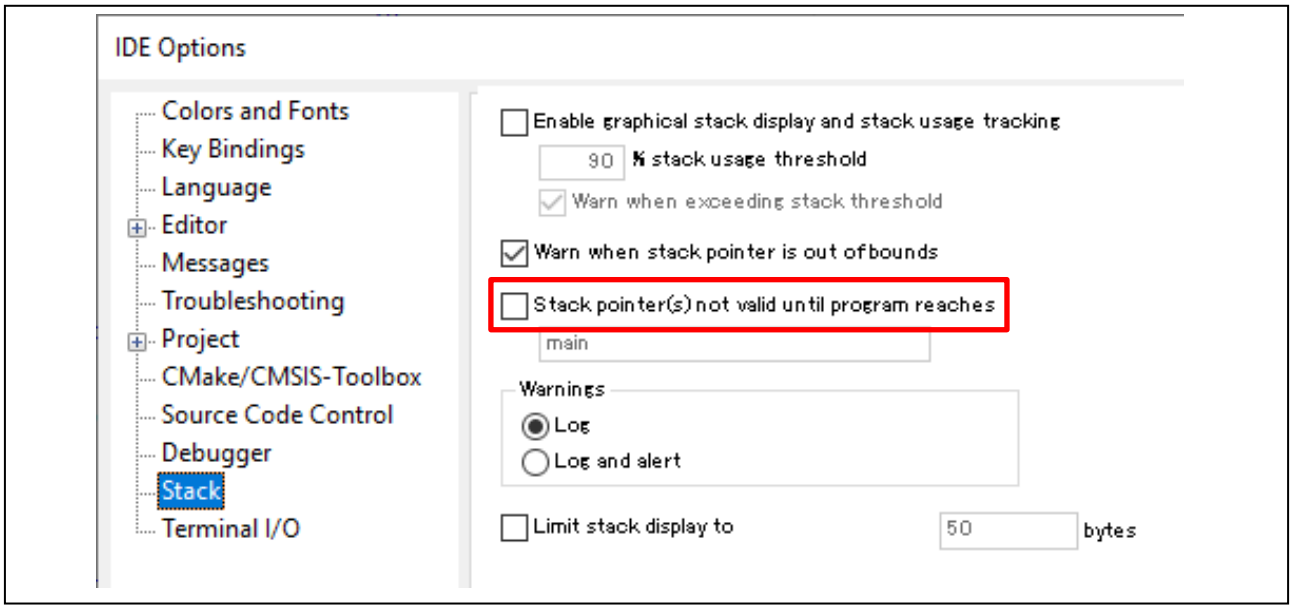

Figure 5-8 IDE Options setting

5.1.2 e² studio from Renesas

1. Launch e² studio 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.zip".
   Then, click "[Finish]"

3. Open configuration.xml on “RZ/T2H_bsp_xspi0bootx1_app” project, push the [Generate Project Content] button to execute code generation.

4. Build RZ/T2H_bsp_xspi0bootx1_app.

5. Open configuration.xml on “RZ/T2H_bsp_xspi0bootx1_loader” project, push the [Generate Project Content] button to execute code generation.

6. Build RZ/T2H_bsp_xspi0bootx1_loader.

7. Make sure that your PC and RZ/T2H evaluation board are connected with J-Link. Then, select “RZ/T2H_bsp_xspi0bootx1_loader” in connection setting and start debugging with "[Debug]"

8. 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).

Supplementary Notes for the e²studio Environment (1)

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.

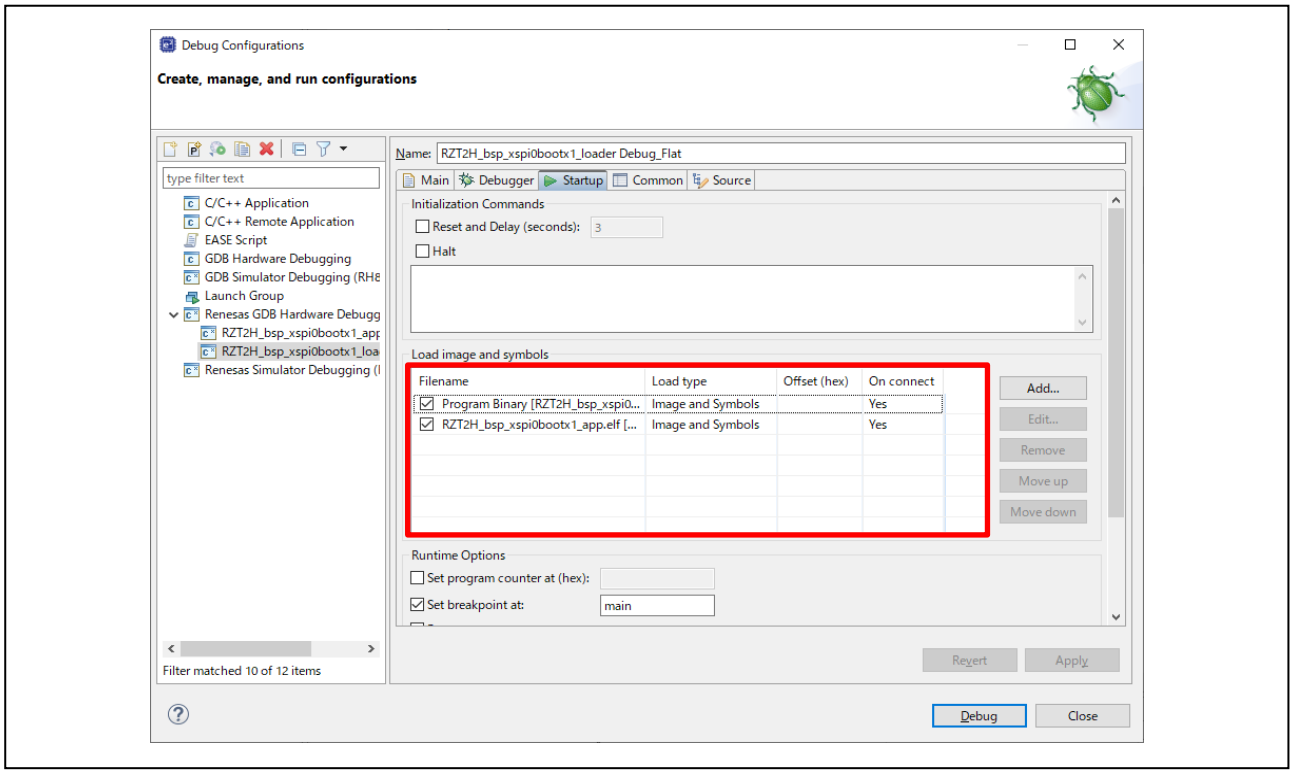

Figure 5-9 Debug configurations in e2 studio

5.1.3 Supplementary Notes for the application and loader program

The startup_core.c files for both the application program and the loader program have been modified from the versions generated by the FSP Configurator.

The modified startup_core.c files are included in this sample package even before code generation.

If you change the FSP version, the files may be overwritten by newly generated ones, so please make sure to apply the same modifications again.

```
BSP_TARGET_ARM BSP_ATTRIBUTE_STACKLESS void system_init (void)
{
#if 1 // Software loops are only needed when debugging.
    __asm volatile (
        "   mov   r0, #0                        \n"
        "   movw  r1, #0xf07f                    \n"
        "   movt  r1, #0x2fa                     \n"
        "software_loop:                          \n"
        "   adds  r0, #1                        \n"
        "   cmp   r0, r1                        \n"
        "   bne   software_loop                  \n"
        ::: "memory");
#endif

    __asm volatile (
        "set_hactlr:                             \n"
        "    MOVW  r0, %[bsp_hactlr_bit_l]       \n" /* Set HACTLR bits(L) */
        "    MOVT  r0, #0                        \n"
        "    MCR   p15, #4, r0, c1, c0, #1       \n" /* Write r0 to HACTLR */
        ::[bsp_hactlr_bit_l] "i" (BSP_HACTLR_BIT_L) : "memory");
```

Figure 5-10 Modification to startup_core.c (loader program)

```
BSP_TARGET_ARM BSP_ATTRIBUTE_STACKLESS void system_init (void)
{
/* These settings are invalid for application project.
 * The necessary processing has been performed in the loader program. */
#if 0 // Original program
    __asm volatile (
        "set_hactlr:                             \n"
        "    MOVW  r0, %[bsp_hactlr_bit_l]       \n" /* Set HACTLR bits(L) */
        "    MOVT  r0, #0                        \n"
        "    MCR   p15, #4, r0, c1, c0, #1       \n" /* Write r0 to HACTLR */
        ::[bsp_hactlr_bit_l] "i" (BSP_HACTLR_BIT_L) : "memory");

~~~ omission ~~~

    __asm volatile (
        "exception_return:                       \n"
        "    LDR   r1, =stack_init               \n"
        "    MSR   ELR_hyp, r1                   \n"
        "    ERET                               \n" /* Branch to stack_init and enter EL1 */
        ::: "memory");

#else
    /* Set exception vector offset for application program. */
    __asm volatile (
        "set_vbar:                               \n"
        "    LDR   r0, =__Vectors                \n"
        "    MCR   p15, #0, r0, c12, c0, #0       \n" /* Write r0 to VBAR */
        ::: "memory");

    __asm volatile (
        "jump_stack_init:                        \n"
        "    LDR   r0, =stack_init               \n"
        "    BLX   r0                           \n"
        ::: "memory");
#endif
}
```

Figure 5-11 Modification to startup_core.c (application program)

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 BTCM_LDR_region  =   mem:[from 0x00102000 size  56K];
define region APPLICATION_RAM_region  =  mem:[from 0x10080000 size 128K];

/* Flash memory */
define region LOADER_TABLE_region  =  mem:[from 0x40080000 size 64K];
define region APPLICATION_ROM_region  =  mem:[from 0x40010000 size 64K];
~~~

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

/* Flash memory */
define region LOADER_TABLE_region  =  mem:[from 0x40080000 size 64K];
define region APPLICATION_ROM_region  =  mem:[from 0x40010000 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 USER_DATA_WBLOCK };
place in SYSTEM_RAM_PRG_region          { block USER_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_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 USER_DATA_WBLOCK };    /* Change data area to ATCM */
place in ATCM_region           { block USER_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 */
~~~
```

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 0x40010000 : AT (0x40010000)
{
      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 0x40010000 : AT (0x40010000)
{
      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 (TEXT_IMAGE)
{
    ~~~
} > SYSTEM_RAM
.rvectors :
{
    ~~~
} > SYSTEM_RAM
.ARM.extab :
{
    ~~~
} > SYSTEM_RAM
.ARM.exidx :
{
    ~~~
} > SYSTEM_RAM
.data _text_end :  AT (DATA_IMAGE)
{
    ~~~
} > SYSTEM_RAM
.got :
{
    ~~~
} > SYSTEM_RAM
.bss :
{
    ~~~
} > SYSTEM_RAM
```

```
.heap (NOLOAD) :
{
    ~~~
} > SYSTEM_RAM
.aarch64_stack (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 (TEXT_IMAGE) /* Change code area to ATCM */
{
    ~~~
} > ATCM
.rvectors :
{
    ~~~
} > ATCM
.ARM.extab :
{
    ~~~
} > ATCM
.ARM.exidx :
{
    ~~~
} > ATCM
.data _text_end :  AT (DATA_IMAGE) /* Change data area to ATCM */
{
    ~~~
} > ATCM
.got :
{
    ~~~
} > ATCM
.bss : /* Change bss area to ATCM */
{
    ~~~
} > ATCM
.heap (NOLOAD) : /* Change heap area to ATCM */
{
    ~~~
} > ATCM
.aarch64_stack (NOLOAD) :
{
    ~~~
```

```
  } > 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
  ~~~
```

5.3 How to Debug Cortex-A55 CPU0 Program

The sample program is Coertex-R52 CPU0 single core operation with default configuration. The definition "USE_CA55_CPU0" is added to the project options, and changing its value enables the program required to run Cortex-A55 CPU0.

When Cortex-A55 CPU0 configuration is enabled, the Table 1 parameters in the loader table are replaced with the information for copying the Cortex-A55 CPU0 program. The loader program refers to the parameters and copies the Cortex-A55 CPU0 program in addition to the application program.

In addition, Cortex-A55 CPU0 reset release process is added to the application program. After the reset release process is executed, Cortex-A55 CPU0 program runs from the beginning of System SRAM (0x1000_0000).

Detailed procedures for debugging Cortex-A55 CPU0 programs in each development environment are shown on the following pages.

5.3.1 EWARM from IAR systems

1. Build Cortex-A55 CPU0 program

   Open the file
   “Loader_application_projects\ RZT2H_bsp_xspi0bootx1_app_CA55_0\RZT2H_bsp_xspi0bootx1_app_CA55_0.eww ” and build the Cortex-A55 CPU0 program.

   Open FSP Configurator from [Tool], push the [Generate Project Content] button to execute code generation. After that, close the FSP Configurator.

   Build the Cortex-A55 project. The following project option settings will output build artifacts in raw binary.

   After building the Cortex-A55 CPU0 program, close the RZ/T2H_ca55_cpu0.eww workspace.

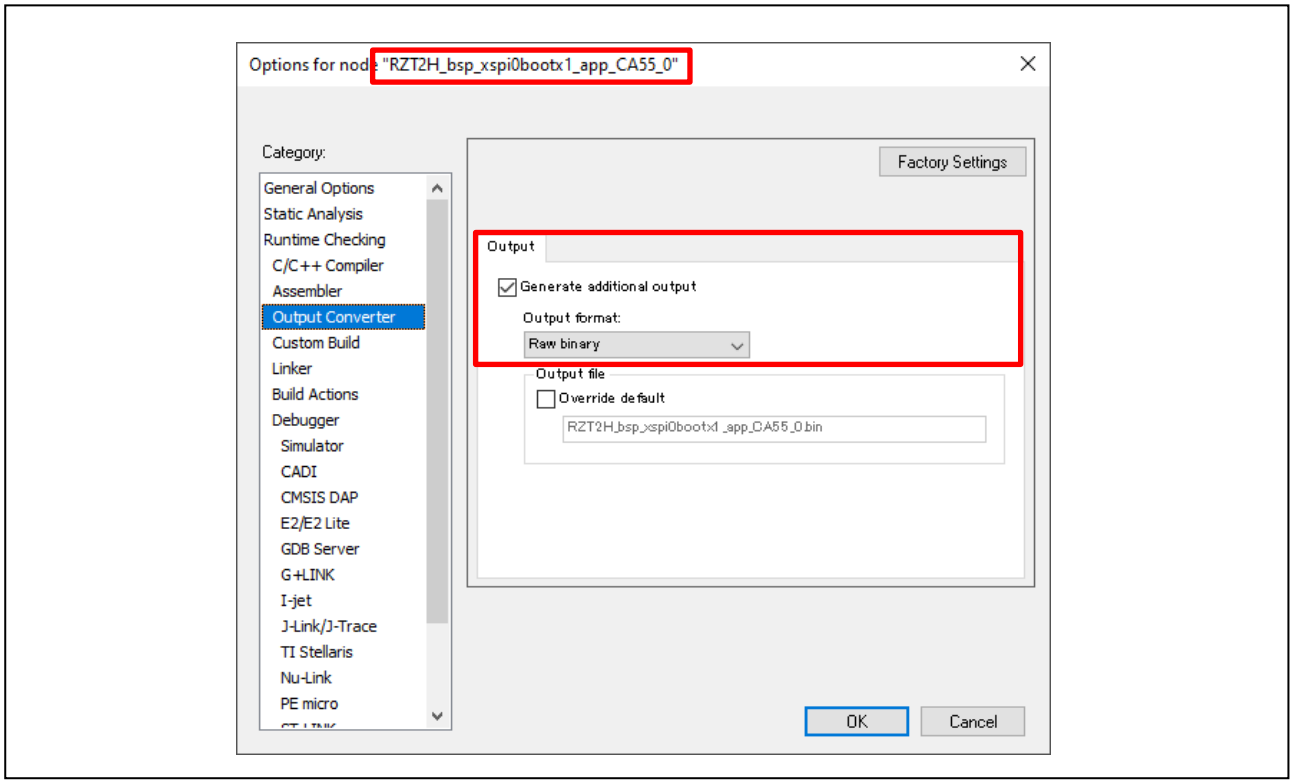

Figure 5-12 Cortex-A55 CPU0 program option settings

2. Link Cortex-A55 CPU0 program to Cortex-R52 CPU0 loader program

Add the following project option settings to link the Cortex-A55 CPU0 binary to the Cortex-R52 CPU0 loader program.

Project for the loader program : [Options] -> [Linker] -> [Input]

Keep symbols : CA55_CPU0_SECTION
File :
$PROJ_DIR$\..\RZT2H_bsp_xspi0bootx1_app_CA55_0\Debug\Exe\RZT2H_bsp_xspi0bootx1_app_CA55_0.bin
Symbol : CA55_CPU0_SECTION
Section : CA55_CPU0_SECTION
Align : 4

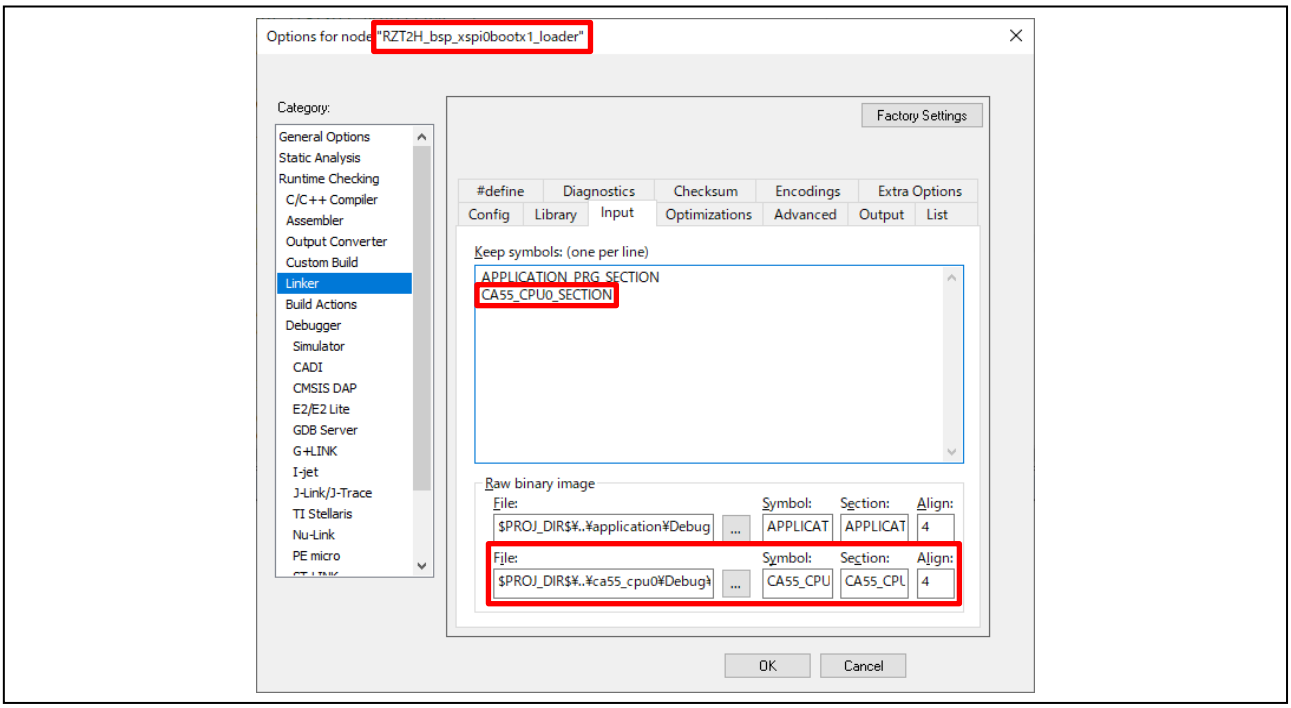

Figure 5-13 Cortex-R52 CPU0 loader program option settings

3. Enable USE_CA55_CPU0 definition

Activate the definition to run the Cortex-A55 CPU0 program. Change the value of "USE_CA55_CPU0" 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_CA55_CPU0=1
  [Options] -> [Linker] -> [Config]: USE_CA55_CPU0=1

- **Project for the Application program**
  [Options] -> [C/C++ Compiler] -> [Preprocessor]: USE_CA55_CPU0=1

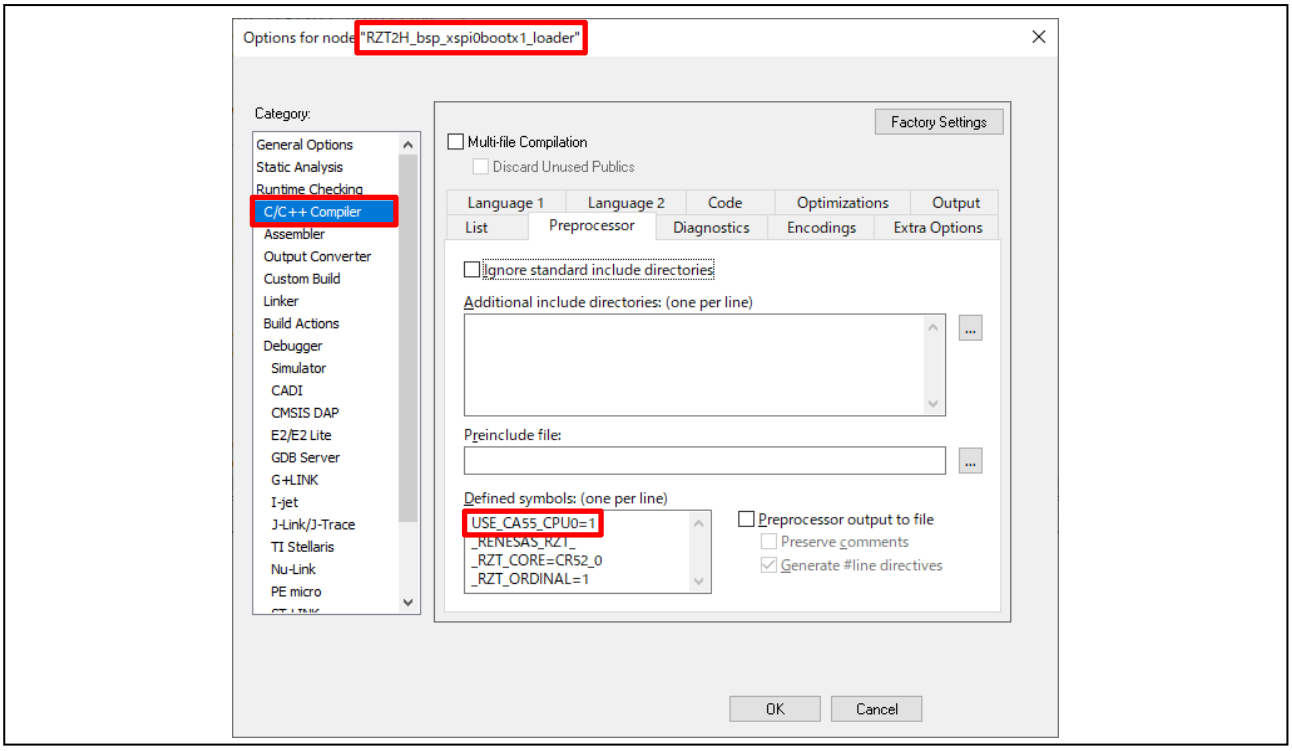

Figure 5-14 Enabling USE_CA55_CPU0 definition of Cortex-R52 CPU0 loader program (1/2)

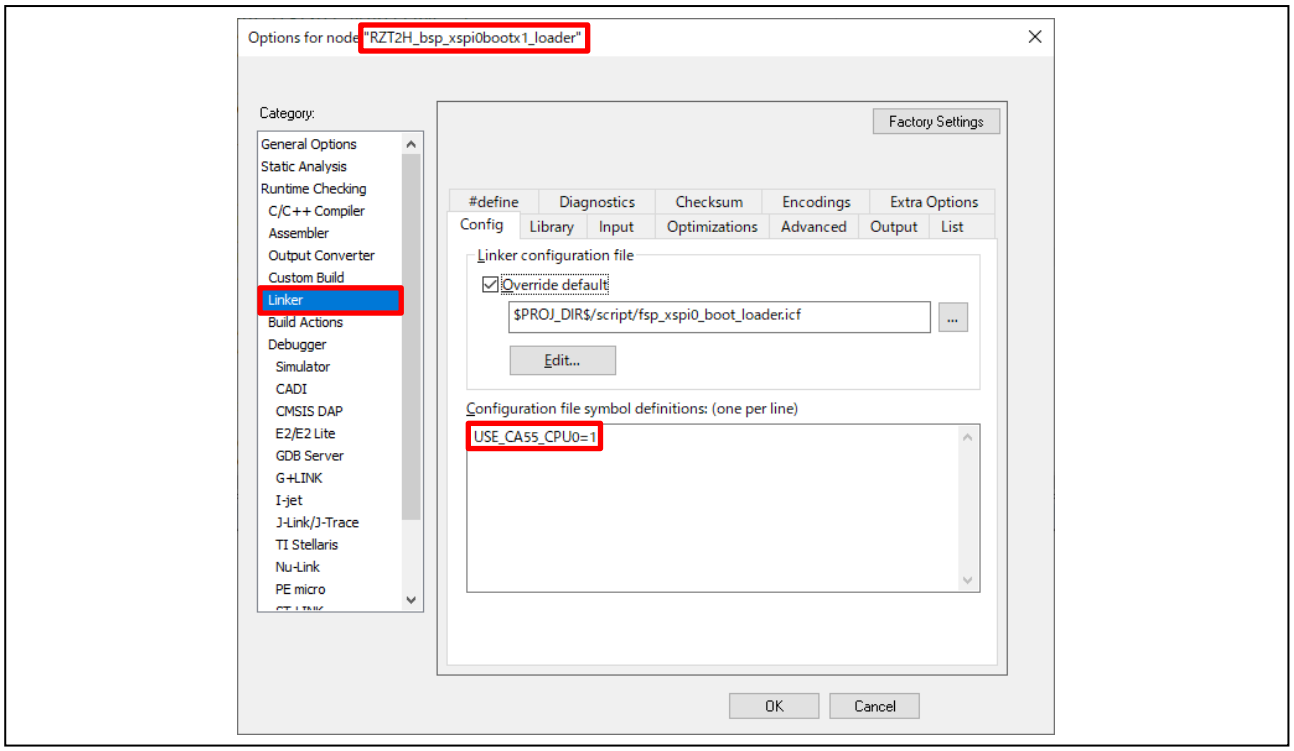

Figure 5-15 Enabling USE_CA55_CPU0 definition of Cortex-R52 CPU0 loader program (2/2)

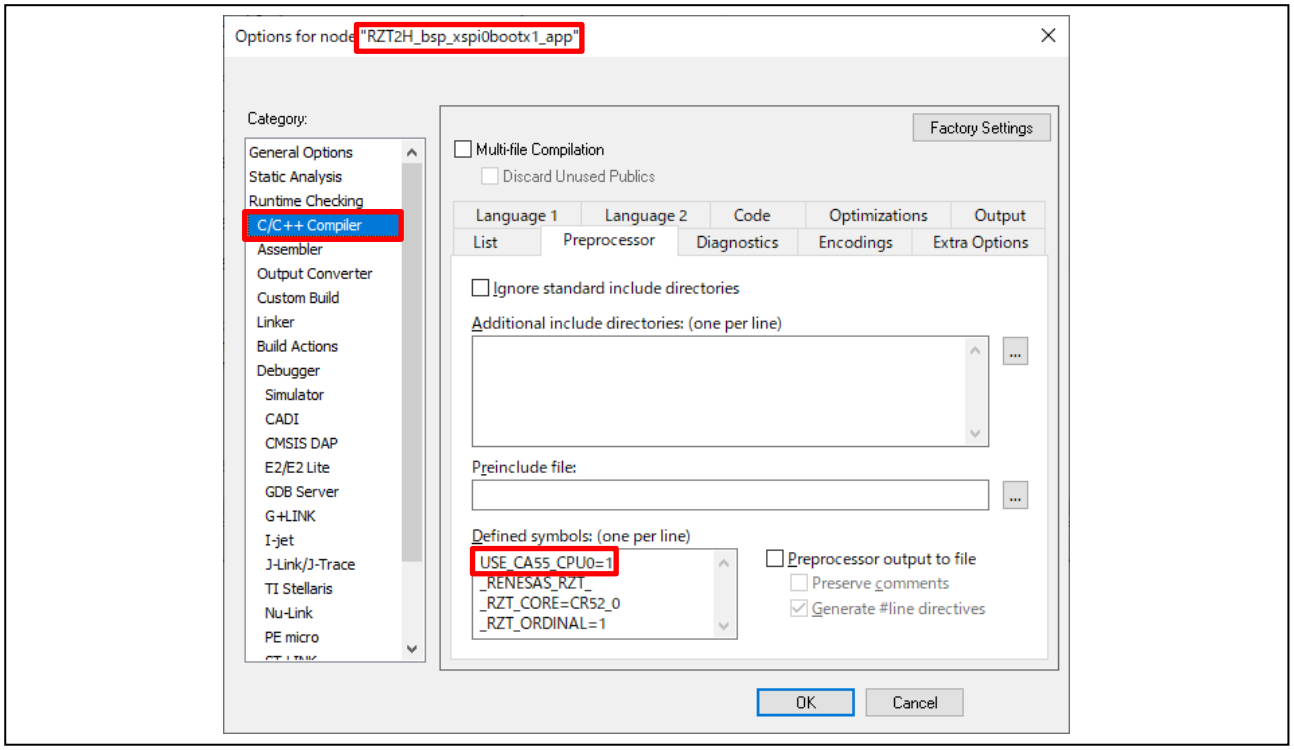

Figure 5-16 Enabling USE_CA55_CPU0 definition of Cortex-R52 CPU0 application program

4. Settings for debugging Cortex-A55 CPU0 project

To debug the Cortex-A55 CPU0 program, use EWARM's multicore debugging function. The following additional project option settings enable debugging of Cortex-A55 CPU0 projects.

Project for the loader program: [Options] -> [Debugger] -> [Multicore]
Asymmetric multicore : Enable “Simple”.
Partner workspace :
$PROJ_DIR$\..\RZT2H_bsp_xspi0bootx1_app_CA55_0\RZT2H_bsp_xspi0bootx1_app_CA55_0.eww
Partner project : RZT2H_bsp_xspi0bootx1_app_CA55_0
Partner configuration : Debug

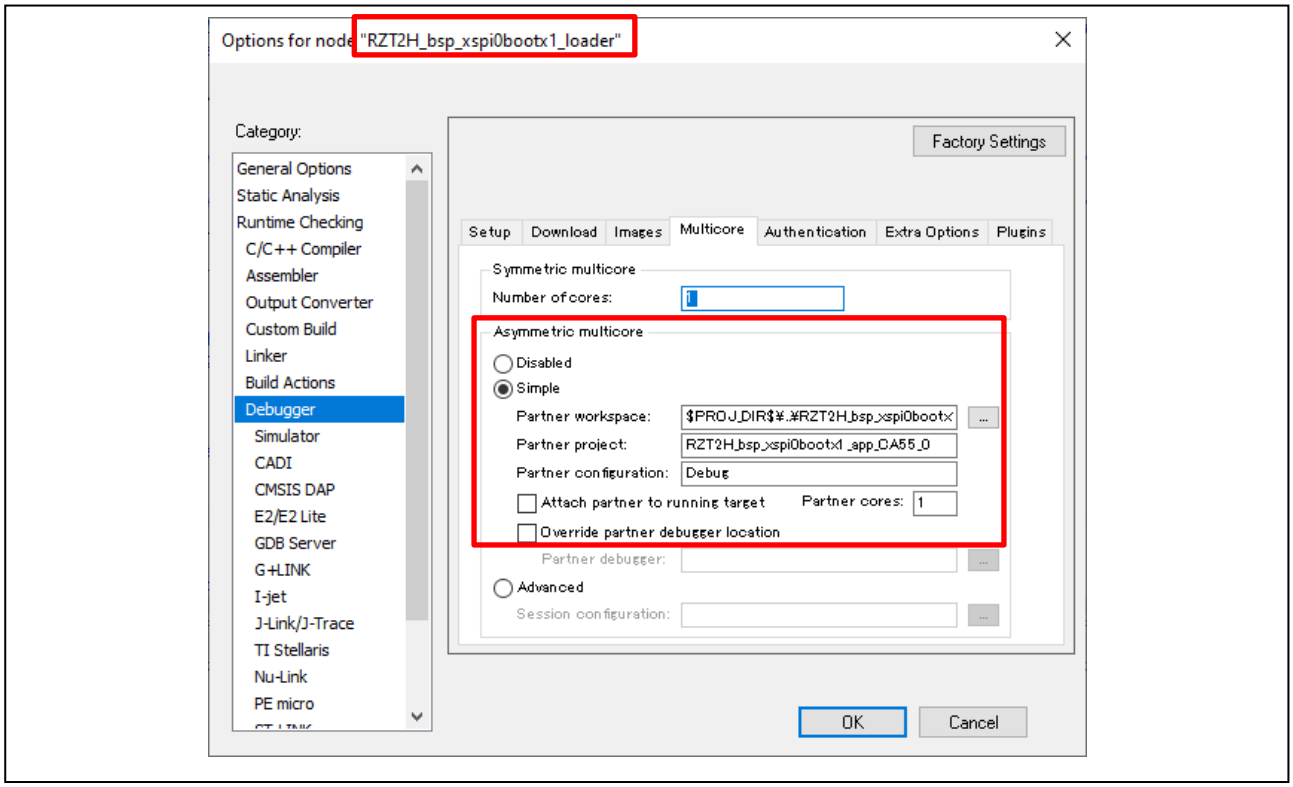

Figure 5-17 Multicore debug settings for Cortex-R52 CPU0 loader program

5. Rebuild and run the project

Follow the 5.1.1 process, start the download and debug.

During the debugging connection, workspace for the Cortex-A55 CPU0 project automatically opens as well. Thereafter, Cortex-R52 CPU0 and Cortex-A55 CPU0 projects can be debugged.
When the Cortex-A55 CPU0 program is executed, LED2 blink.

5.3.2 e2studio from Renesas

1. Build Cortex-A55 CPU0 program

   Open configuration.xml on “RZ/T2H_bsp_xspi0bootx1_app_CA55_0” project, push the [Generate Project Content] button to execute code generation.

   Build the “RZ/T2H_bsp_xspi0bootx1_app_CA55_0” program. The following project option settings will output build artifacts in raw binary.

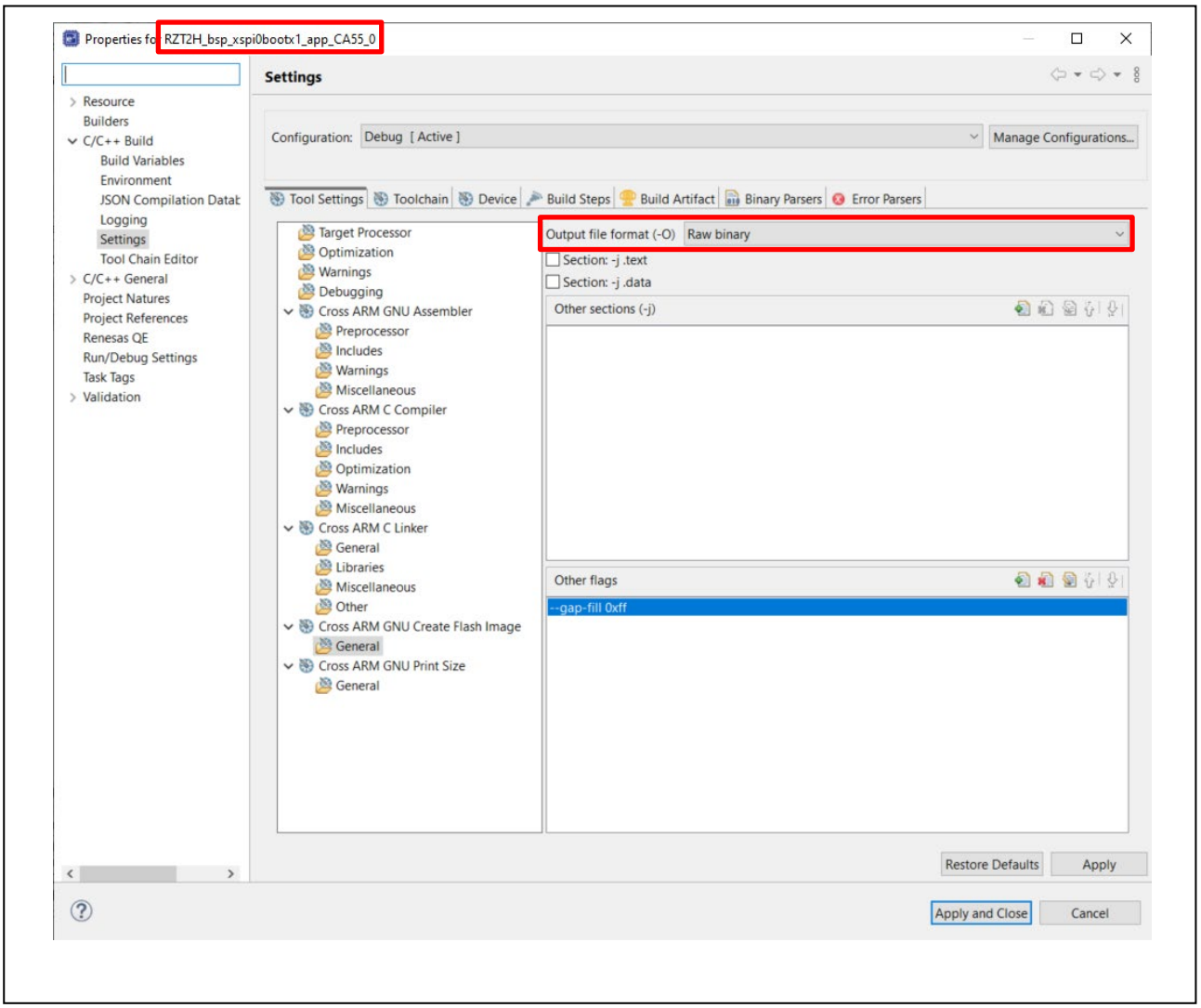

Figure 5-18 Cortex-A55 CPU0 program option settings

2. Link Cortex-A55 CPU0 program to Cortex-R52 CPU0 loader program

A section definition is added to the linker script file to link the Cortex-A55 CPU0 binary to the Cortex-R52 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))
    } > SYSTEM_RAM
    .IMAGE_APP_FLASH_section 0x40010000 : AT (0x40010000)
    {
        IMAGE_APP_FLASH_section_start = .;
        KEEP(./src/Flash_section.o(.IMAGE_APP_FLASH_section))
        IMAGE_APP_FLASH_section_end = .;
    } > FLASH_CONTENTS
    .IMAGE_CA55_CPU0_RAM 0x10000000 : AT (0x10000000)/* CA55_CPU0 program RAM section for execution */
    {
        IMAGE_CA55_CPU0_RAM_start = .;
        KEEP(*(CA55_CPU0_IMAGE_RAM))
    } > SYSTEM_RAM
    .IMAGE_CA55_CPU0_FLASH_section 0x40040000 : AT (0x40040000) /* CA55_CPU0 program, ROM section. */
    {
        IMAGE_CA55_CPU0_FLASH_section_start = .;
        KEEP(./src/Flash_section.o(.IMAGE_CA55_CPU0_FLASH_section))
        IMAGE_CA55_CPU0_FLASH_section_end = .;
    } > FLASH_CONTENTS
    .loader_param LOADER_PARAM_ADDRESS : AT (LOADER_PARAM_ADDRESS)
    {
        KEEP(*(.loader_param))
    } > FLASH_CONTENTS
    .flash_contents FLASH_CONTENTS_ADDRESS : AT (FLASH_CONTENTS_ADDRESS)
    {
        _mtext = .;
        . = (1 == _RZT_ORDINAL) ? . + (_text_end - _text_start) : .;
        . = ALIGN(8);
        _mdata = .;
        _loader_data_start = .;
        . = (1 == _RZT_ORDINAL) ? . + (_data_end - _data_start) : .;
        _loader_data_end = .;
        flash_contents_end = .;
    } > FLASH_CONTENTS
~~~~~~~~
}

IMAGE_APP_FLASH_section_size = SIZEOF(.IMAGE_APP_FLASH_section);
IMAGE_CA55_CPU0_FLASH_section_size = SIZEOF(.IMAGE_CA55_CPU0_FLASH_section);
```

3. Enable USE_CA55_CPU0 definition

Enable the definition to run the Cortex-A55 CPU0 program. Change the value of "USE_CA55_CPU0" 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_CA55_CPU0=1
[Cross ARM C Compiler] -> [Preprocessor]: USE_CA55_CPU0=1

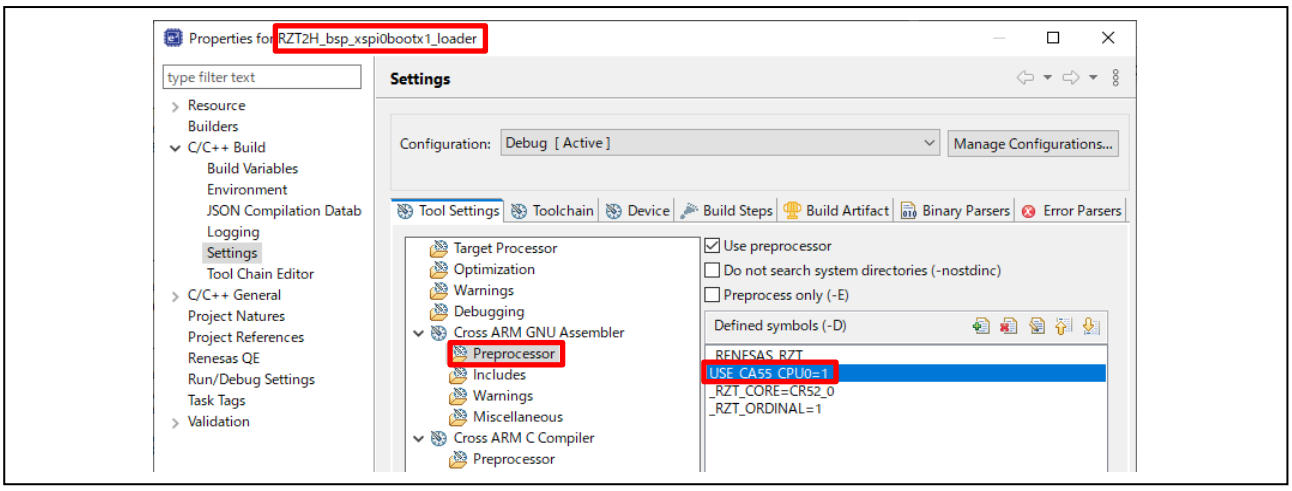

Figure 5-19 Enabling USE_CA55_CPU0 definition of Cortex-R52 CPU0 loader program (1/2)

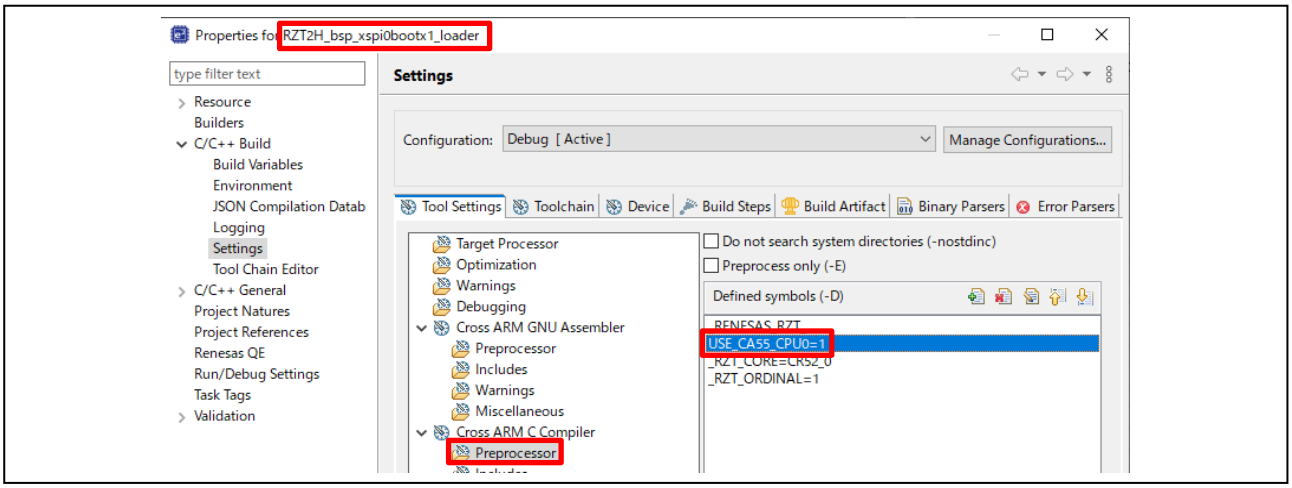

Figure 5-20 Enabling USE_CA55_CPU0 definition of Cortex-R52 CPU0 loader program (2/2)

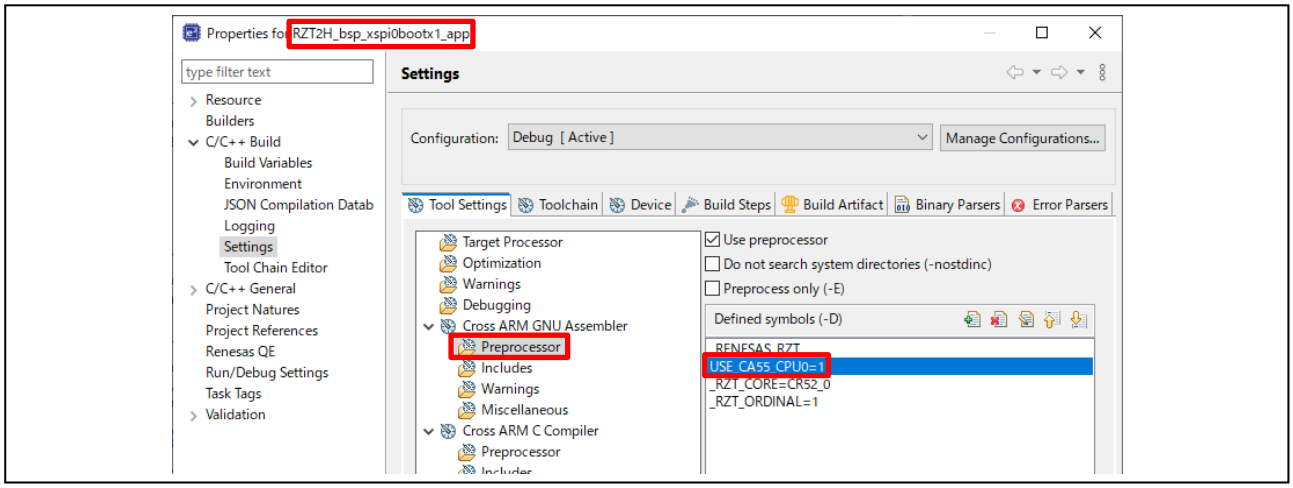

Figure 5-21 Enabling USE_CA55_CPU0 definition of Cortex-R52 CPU0 application program (1/2)

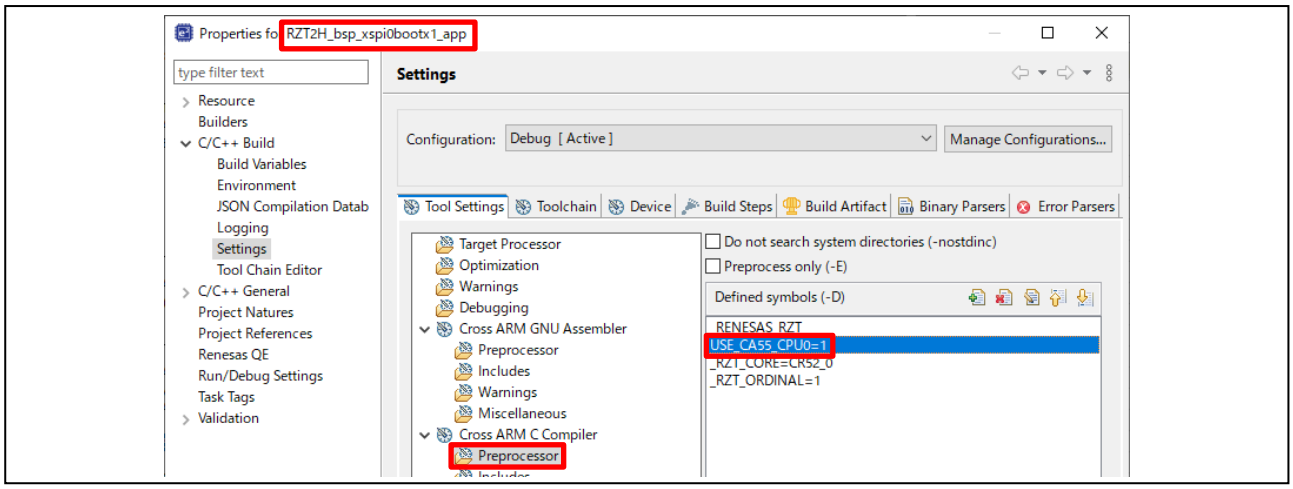

Figure 5-22 Enabling USE_CA55_CPU0 definition of Cortex-R52 CPU0 application program (2/2)

4. Rebuild and run the project

Follow the 5.1.2 process, start the download and debug.

The Cortex-A55 CPU0 program is also written to flash memory at the same time during the debug connection.

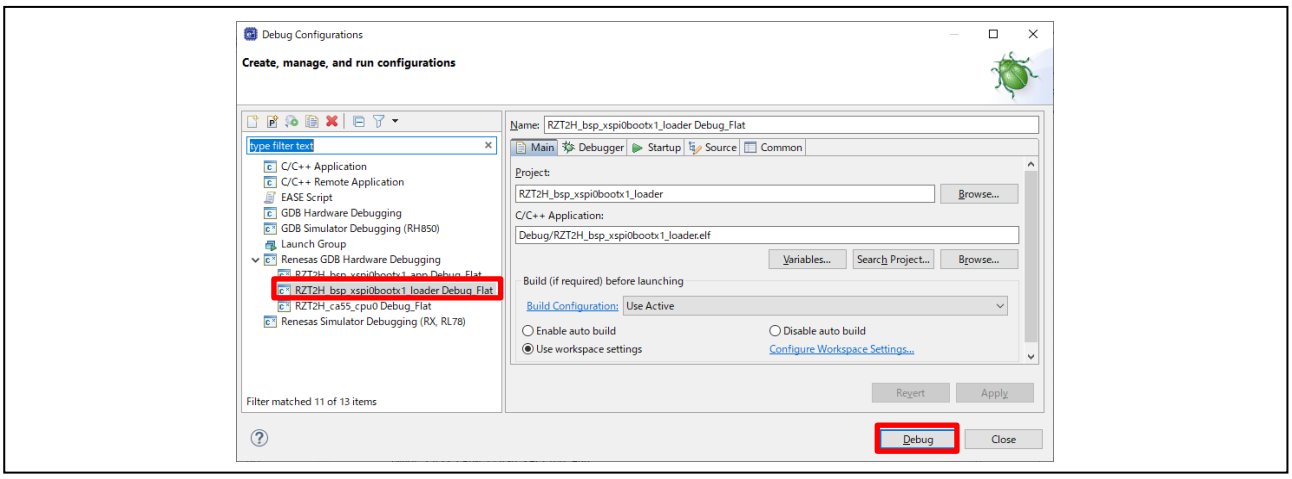

Figure 5-23 Start debugging the Cortex-R52 CPU0 loader program

When debugging the Cortex-A55 CPU0 program, start the debugging connection of the Cortex-A55 CPU0 project after making the debugging connection of the loader program.

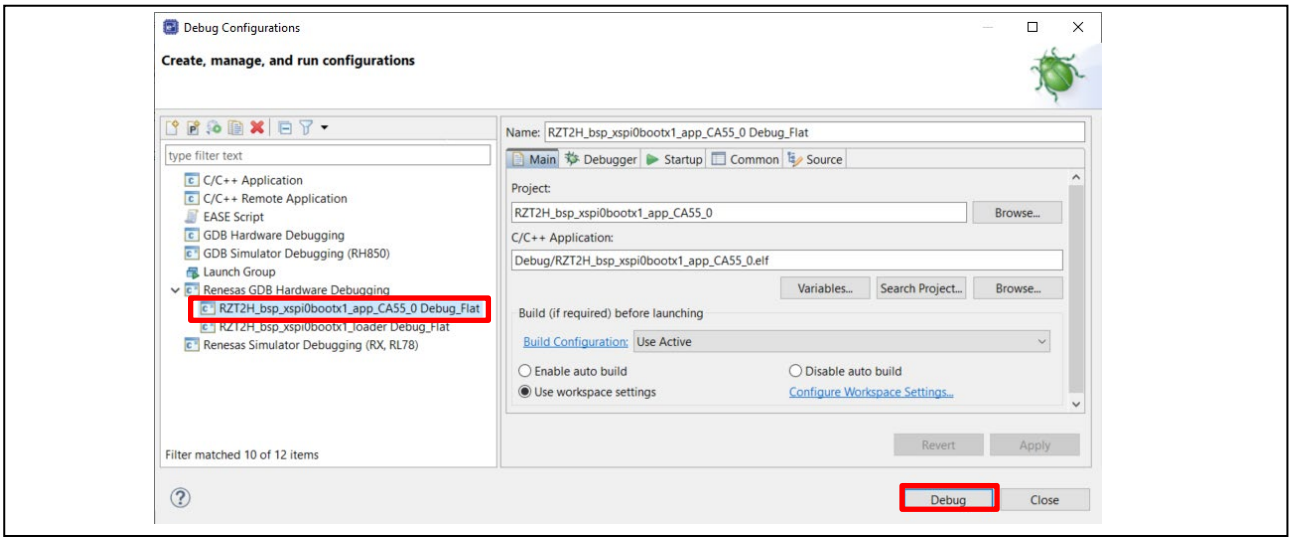

Figure 5-24 Start debugging the Cortex-A55 CPU0 program

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

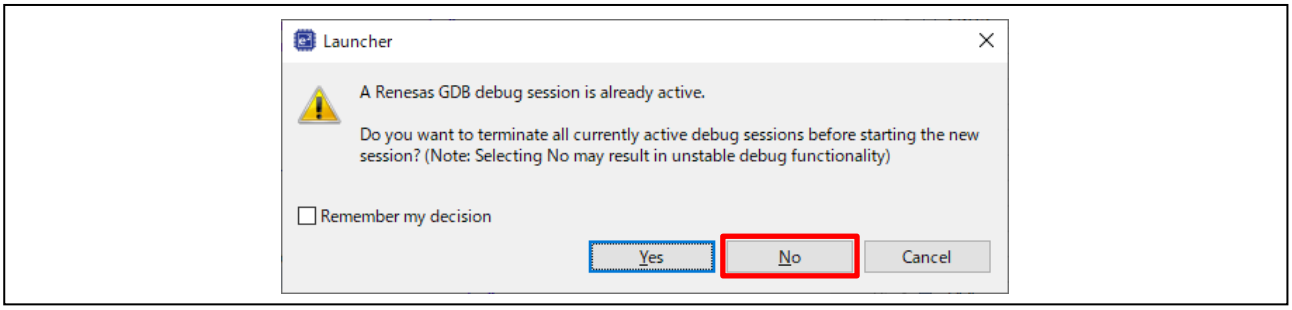

Figure 5-25 Cortex-R52 CPU0 loader program debug continuation confirmation screen

If the debugging connection of the Cortex-A55 CPU0 project succeeds, Cortex-R52 CPU0 and Cortex-A55 CPU0 are connected to the debugger. Thereafter, Cortex-R52 CPU0 and Cortex-A55 CPU0 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 Cortex-R52 CPU0 and Cortex-A55 CPU0.

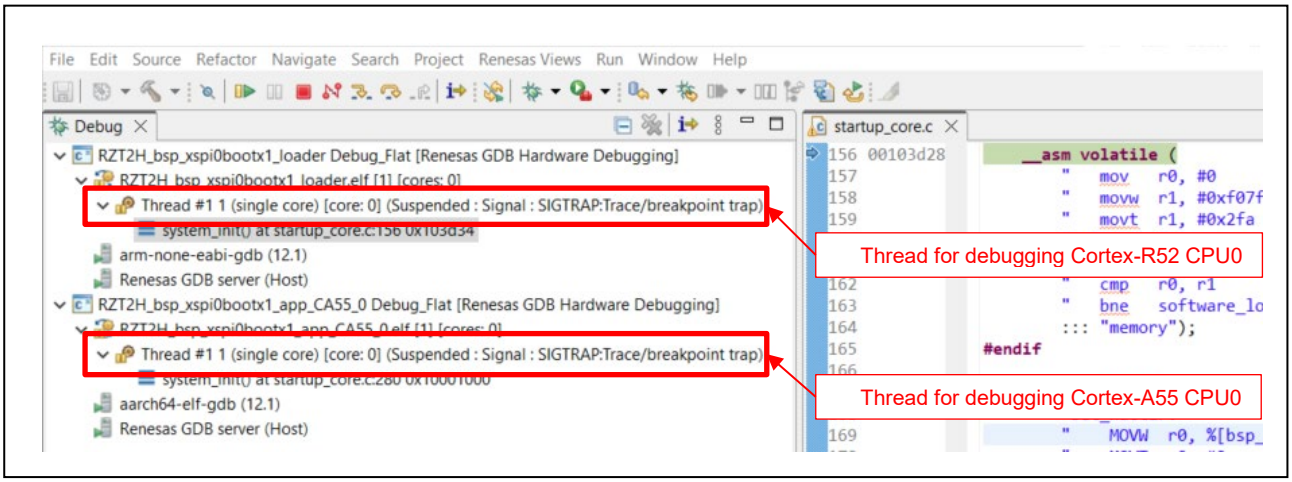

Figure 5-26 Debugging e²studio

When the thread for debugging Cortex-R52 CPU0 project is selected and the program is executed, the loader program and the application program runs and LED0 blink.
When the thread for debugging Cortex-A55 CPU0 project is selected and the program is executed, LED2 blink.

Revision History

Rev.	Date	Description	
		Page	Summary
2.00	Jun.16, 2025	-	First edition issued.

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

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

1. 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/.