RZ/T2M RZ/T2ME Group

Example of separating loader 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

RZ/T2ME 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)
	RZ/T2ME Group (R9A07G075M29GBG)[2]
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 9.60.3 from IAR systems
	● e² studio 2025-04.1 (25.4.1) (R20250508-1327) 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 3.0.0 (RZ/T2 FSP)

Note　1.　When using CPU core1, refer to "5.3 How to Debug CPU1 Program".
Note　2.　When using RZ/T2ME, please replace "RZ/T2M" to RZ/T2ME".
　　　　　And, when checking the operation, also check the docment No. r01an7439ej0110.

1.2 File Structure

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

RZT2M_loader_application

├──r01an6729jj0300-rzt2m.pdf

├──r01an6729ej0300-rzt2m.pdf

├──r01an7439ej0110-rzt2me-procedure-for-converting-from-rzt2m-to-rzt2me.pdf

├──iccarm: for EWARM

│   ├──16bitbusboot: sample program for NOR flash

│   │   └──Loader_application_projects

│   │   ├──RZT2M_bsp_16bitbusboot_app: project for application program

│   │   ├──RZT2M_bsp_16bitbusboot_loader: project for loader program

│   │   ├──RZT2M_bsp_16bitbusboot_app_CPU1: project for CPU1 program

│   │   └──RZT2M_bsp_16bitbusboot_separating_loader.eww: EWARM workspace

│   └──xspi0bootx1: sample program for SPI flash

│   └──Loader_application_projects

│   ├──RZT2M_bsp_xspi0bootx1_app: project for application program

│   ├──RZT2M_bsp_xspi0bootx1_loader: project for loader program

│   ├──RZT2M_bsp_xspi0bootx1_app_CPU1: project for CPU1 program

│   └──RZT2M_bsp_xspi0bootx1_separating_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_bsp_16bitbusboot_app_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_bsp_xspi0bootx1_app_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.

When using RZ/T2ME, please refer the document No. r01an7439ej0110.

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:
MD1	Input	• MD0 = "L", MD1 = "L", MD2 = "L" (xSPI0 boot mode)
MD2	Input	• MD0 = "L", MD1 = "H", MD2 = "L" (16-bit bus boot mode)
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.
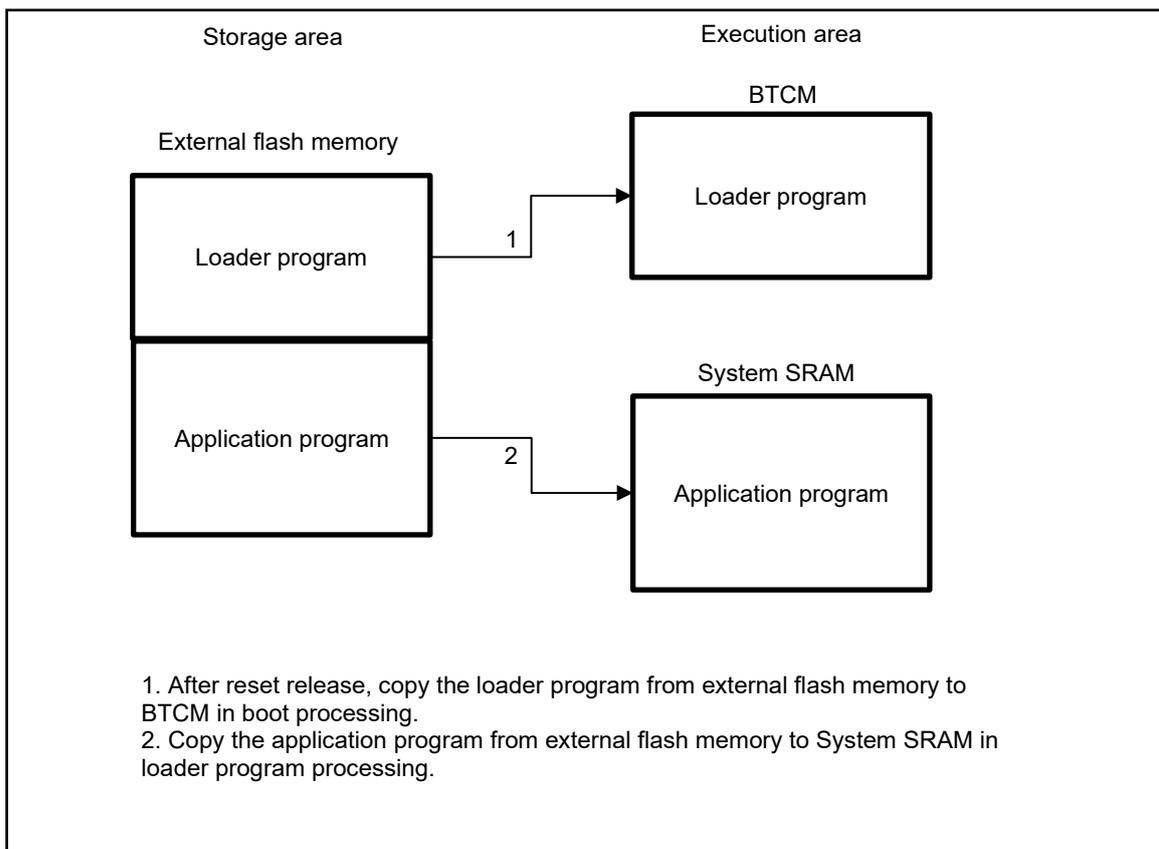

Figure 3.1   Operation overview

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.

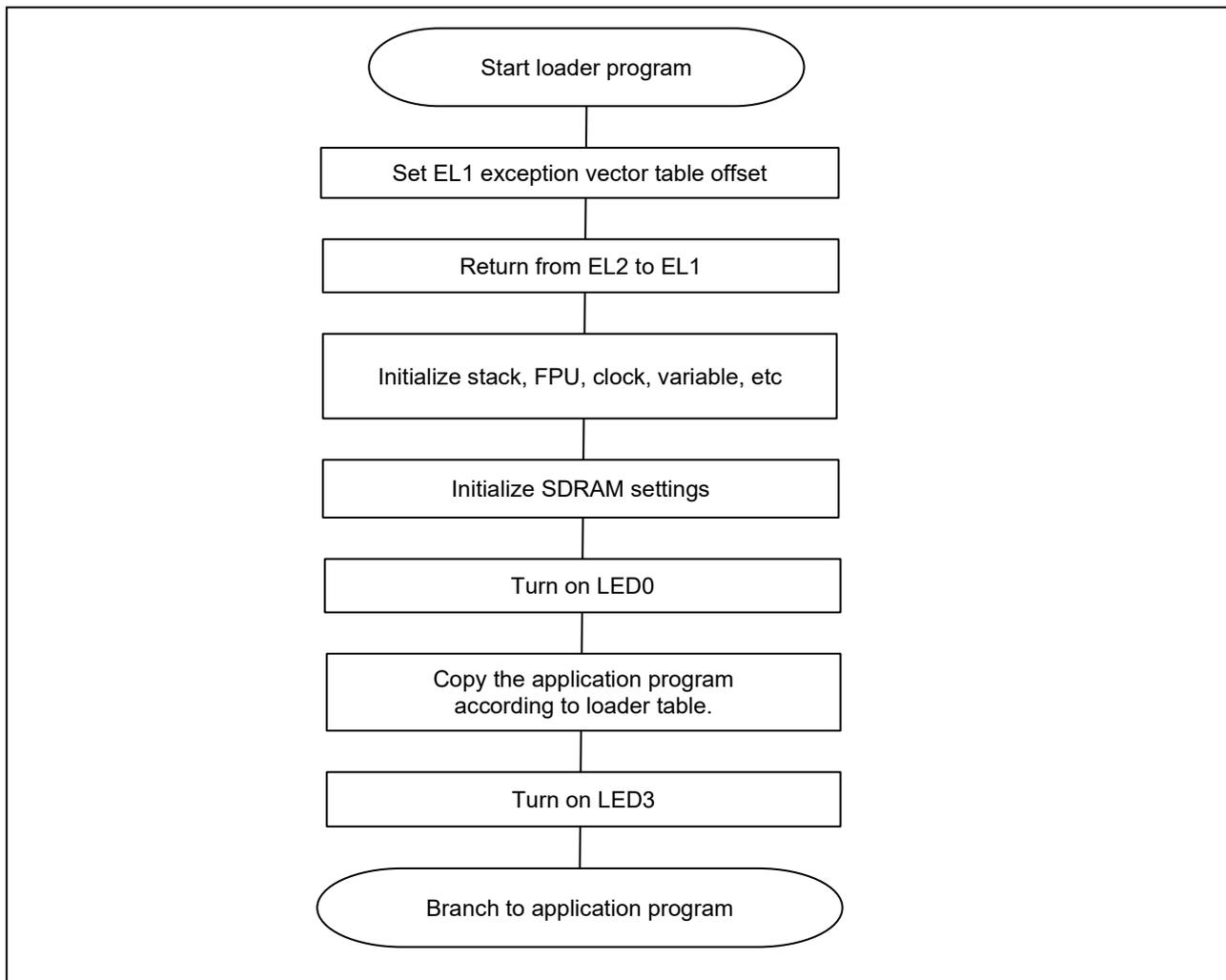Figure 3.2 shows operation overview of loader program.

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.

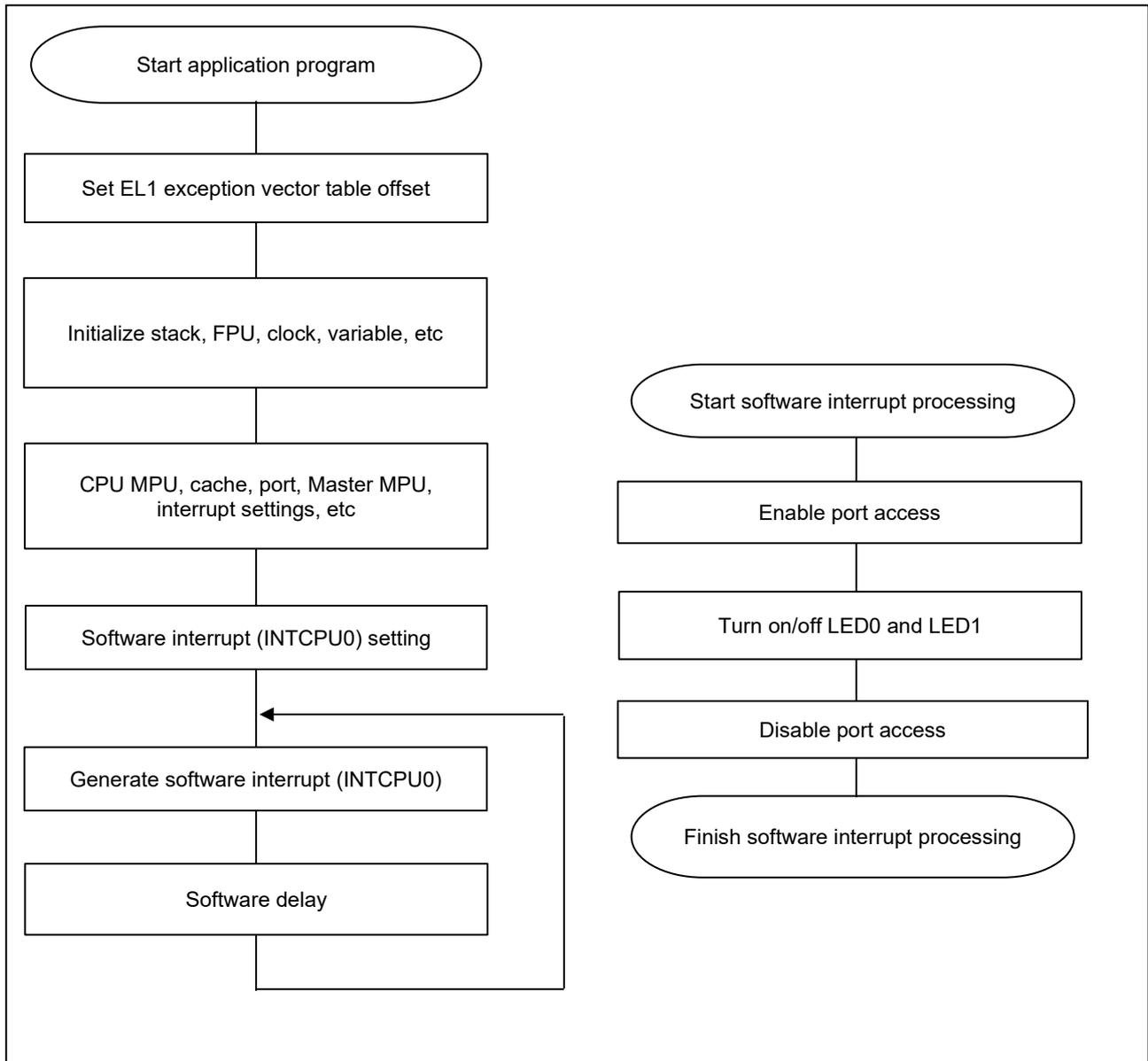Figure 3.3 shows operation overview of application program.

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
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
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
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 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 0x101F_FFFF	Area 2 Normal, cache enabled, non-shared
System SRAM (mirror area)	0x3000_0000 to 0x301F_FFFF	Area 3 Normal, cache disabled, shared
Extended address space (mirror area) xSPI0, xSPI1 CS0, CS2, CS3, CS5	0x4000_0000 to 0x5FFF_FFFF	Area 4, 5, 6, 7, 8, 9 Normal, cache disabled, shared
Extended address space xSPI0, xSPI1 CS0, CS2, CS3, CS5	0x6000_0000 to 0x7FFF_FFFF	Area 10, 11, 12, 13, 14, 15 Normal, cache enabled, non-shared
Non-safety peripheral modules	0x8000_0000 to 0x80FF_FFFF	Area 16 Device (nGnRE) , instruction fetch disabled
Safety peripheral modules	0x8100_0000 to 0x81FF_FFFF	Area 17 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.
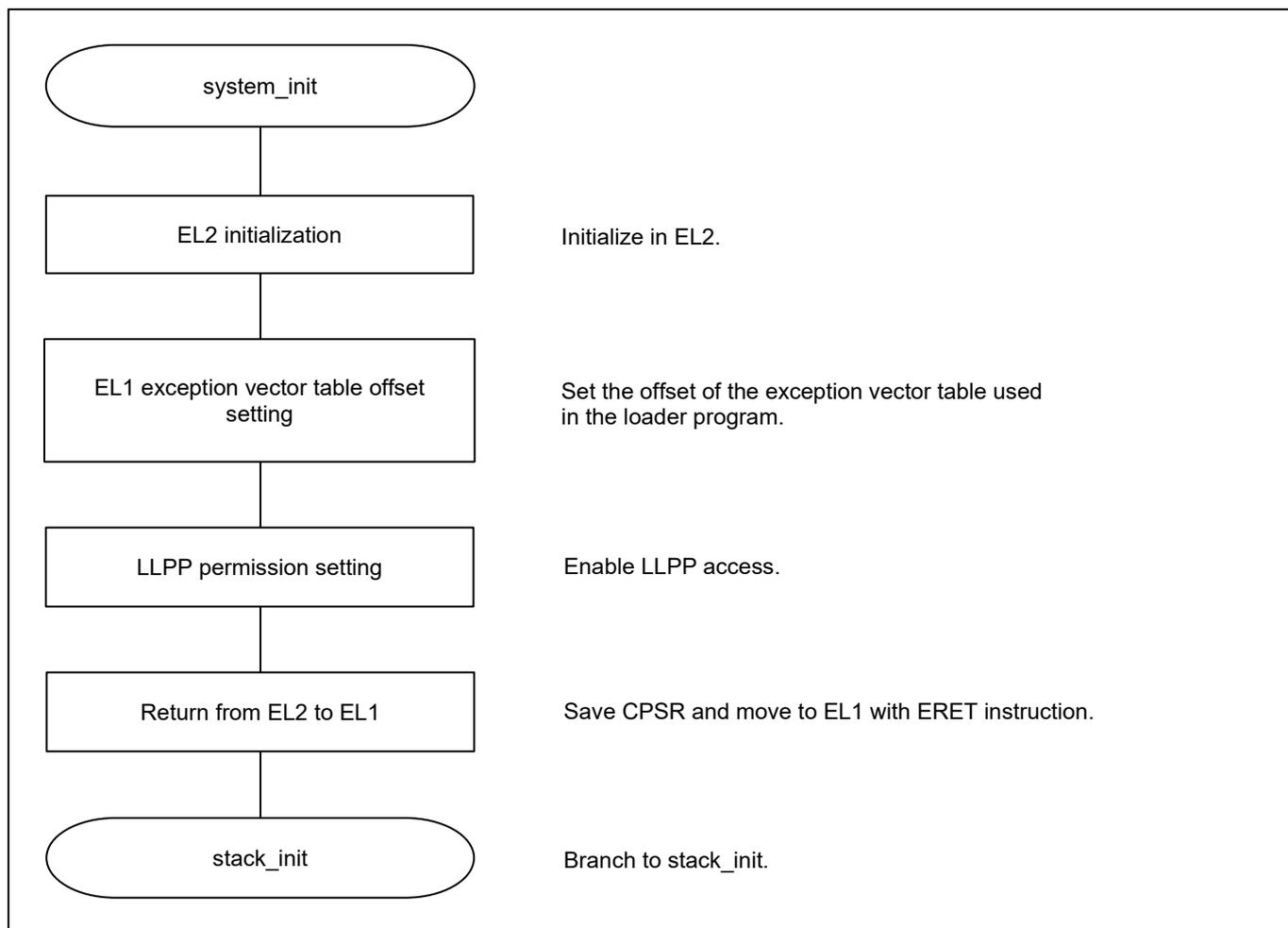
Figure 3.4   system_init processing (loader program)

3.5.1.2 stack_init

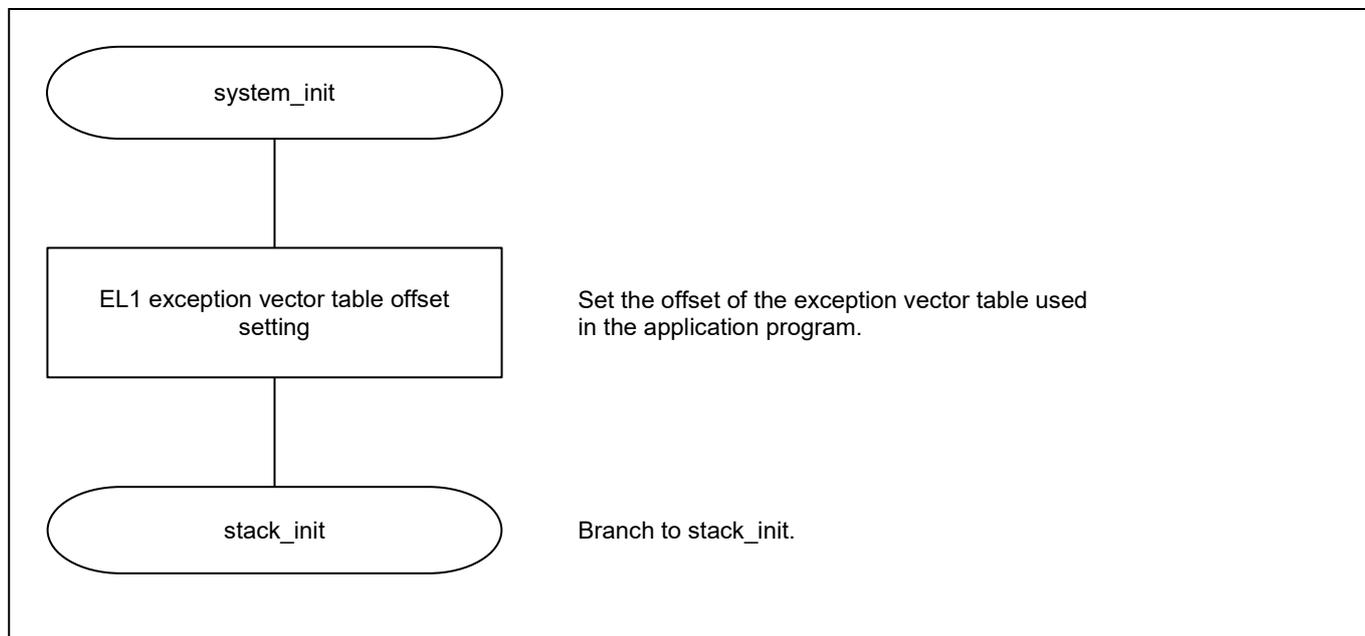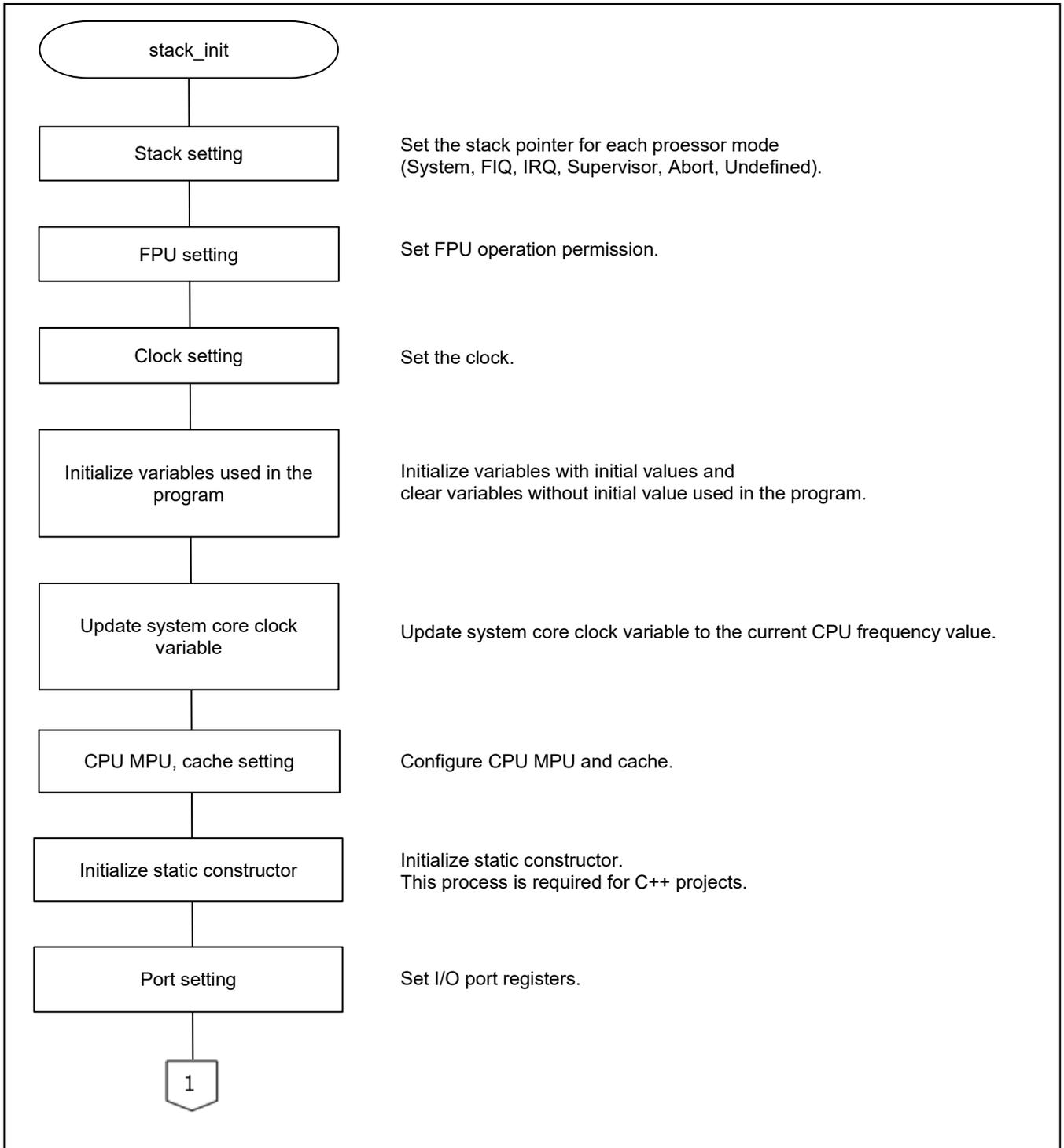Figure 3.5 shows flowchart of stack_init in the loader program.

Stack setting	Set the stack pointer for each processor mode (System, FIQ, IRQ, Supervisor, Abort, Undefined).
FPU setting	Set FPU operation permission.
Clock setting	Set the clock.
Initialize variables used in the program	Initialize variables with initial values and clear variables without initial value used in the program.
Update system core clock variable	Update system core clock variable to the current CPU frequency value.
Initialize static constructor	Initialize static constructor. This process is required for C++ projects.
Port setting	Set I/O port registers.
hal_entry	Branch to hal_entry.

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.

hal_entry	
SDRAM setting	Initialize registers related to SDRAM
Disable port protection	Enable writing to registers related to I/O port.
Turn on LED0	Turns on LED0 to signal the start of copy processing.
Copy application program from external flash memory to RAM	Copy the application program to System SRAM from external flash memory according to the parameters of loader table.
Turn on LED3	Turns on LED3 to signal the end of copy processing.
Enable port protection	Disable writing to registers related to I/O port.
app_prg	Branch to application 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.

stack_init	
Stack setting	Set the stack pointer for each proessor mode (System, FIQ, IRQ, Supervisor, Abort, Undefined).
FPU setting	Set FPU operation permission.
Clock setting	Set the clock.
Initialize variables used in the program	Initialize variables with initial values and clear variables without initial value used in the program.
Update system core clock variable	Update system core clock variable to the current CPU frequency value.
CPU MPU, cache setting	Configure CPU MPU and cache.
Initialize static constructor	Initialize static constructor. This process is required for C++ projects.
Port setting	Set I/O port registers.
1	

Figure 3.8   stack_init processing (1/2)(application program)
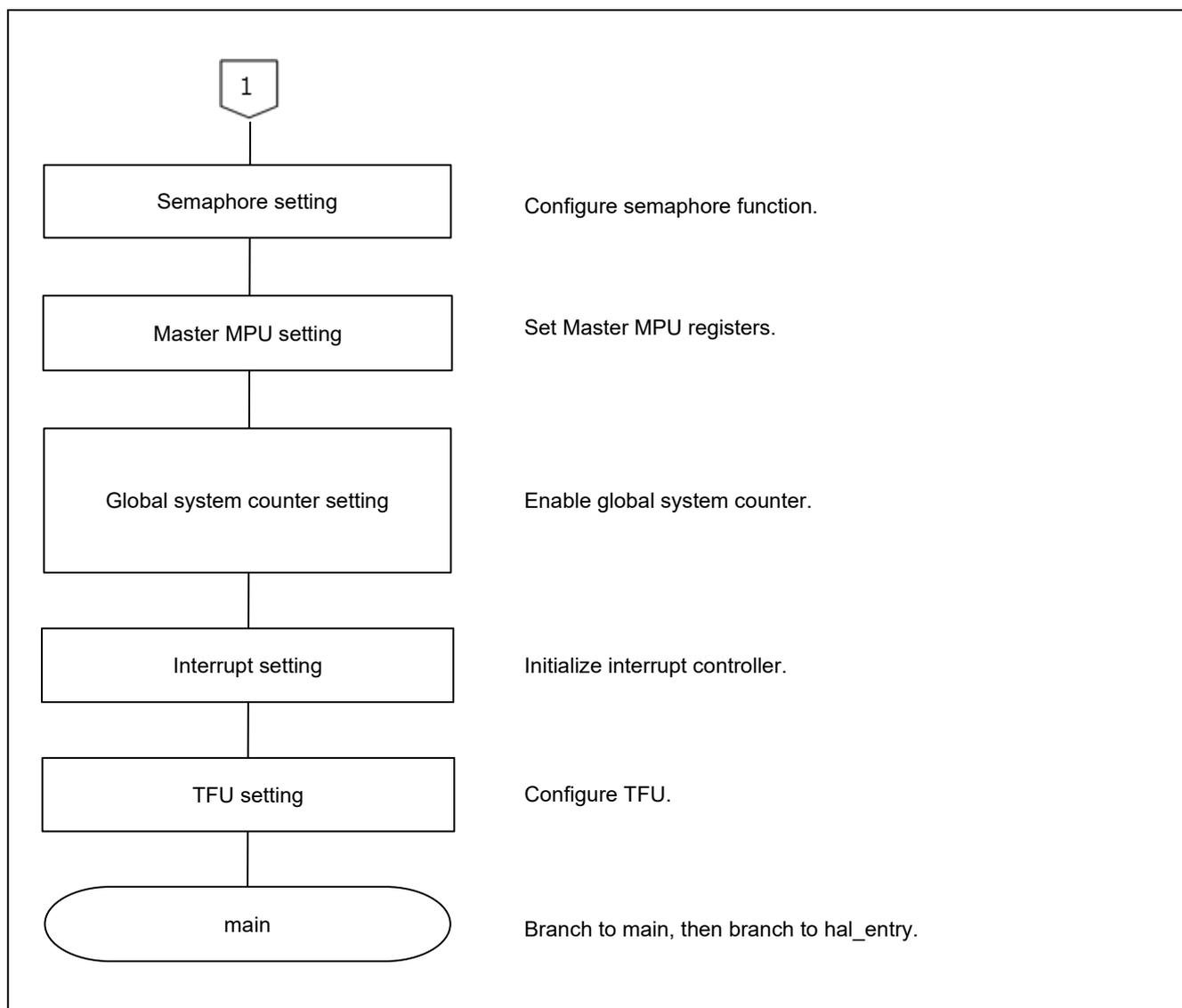
Figure 3.9   stack_init processing (2/2)(application program)

3.5.2.3 hal_entry

Figure 3.11 shows flowchart of hal_entry in the application program.
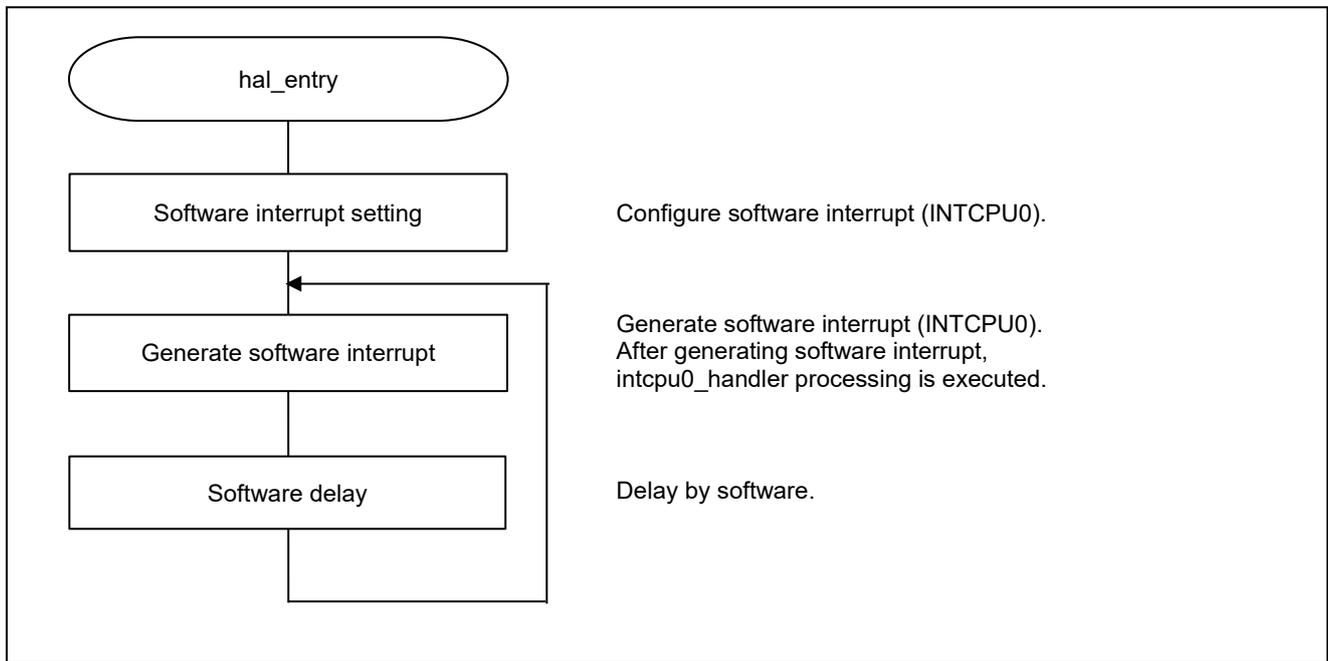
Figure 3.10   hal_entry processing (application program)

Figure 3.12 shows flowchart of interrupt processing in the application program.
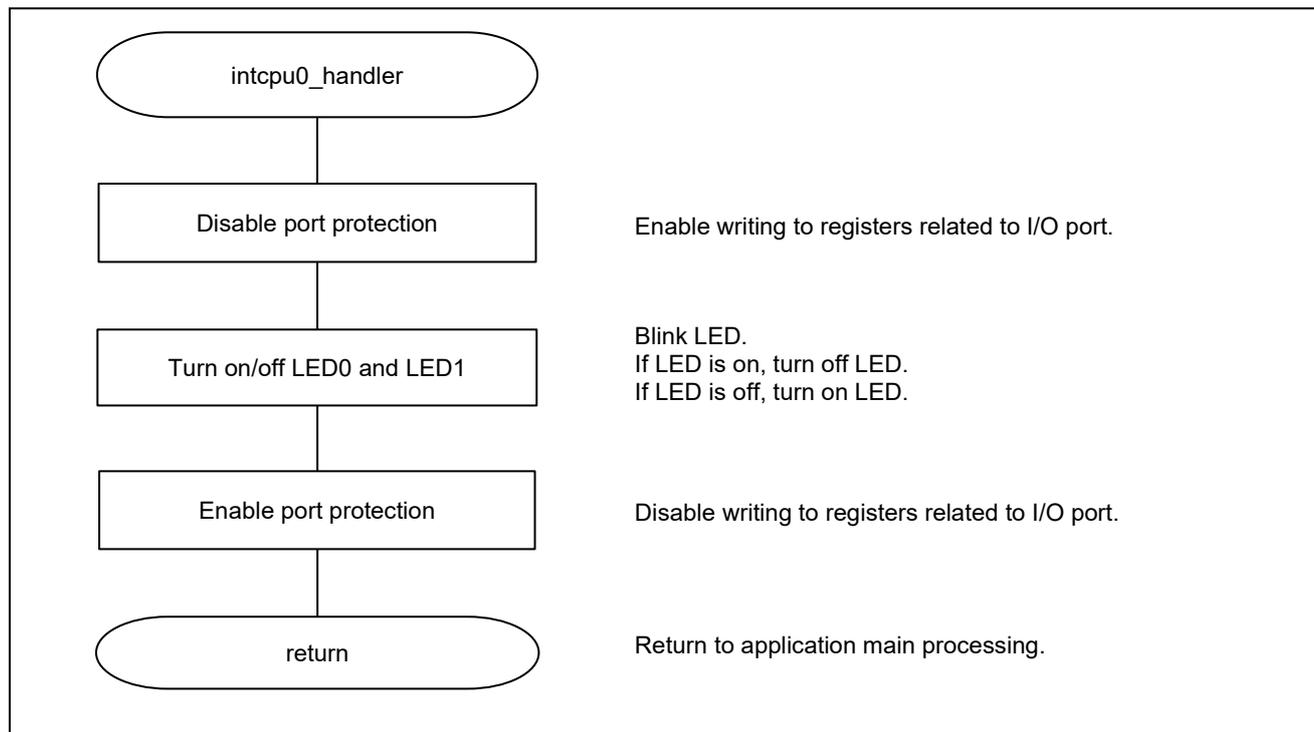
Figure 3.11   interrupt processing (application program)

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_xspi0bootx1_separating_loader.eww" with following procedure.
   "[File] > [Open Workspace] > select Loader_application_projects\RZT2M_bsp_xspi0bootx1_separating_loader. eww"

2. Select "RZT2M_bsp_xspi0boot1_app Debug" project in Workspace box as Figure 5-1.
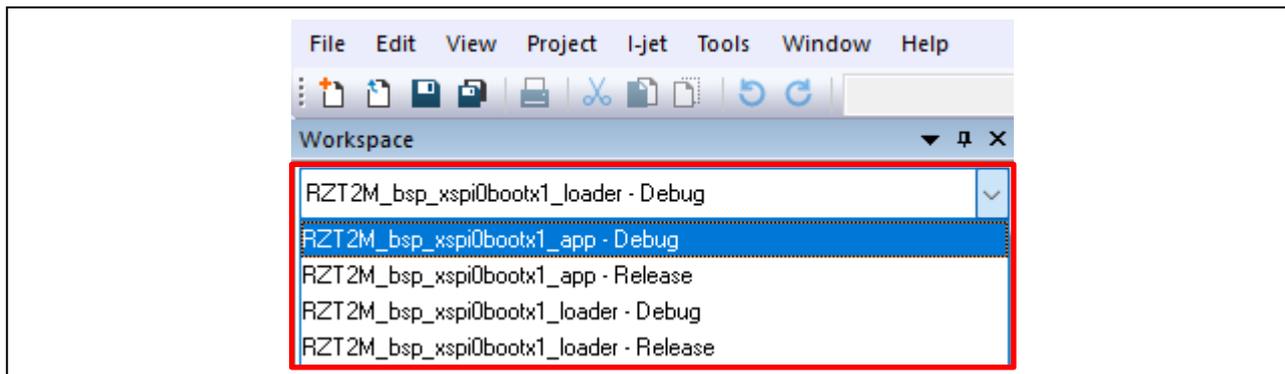
Figure 5-1 Project selection

3. Open FSP Configurator from [Tool], push the [Generate Project Content] button to execute code generation. After that, close the FSP Configurator.
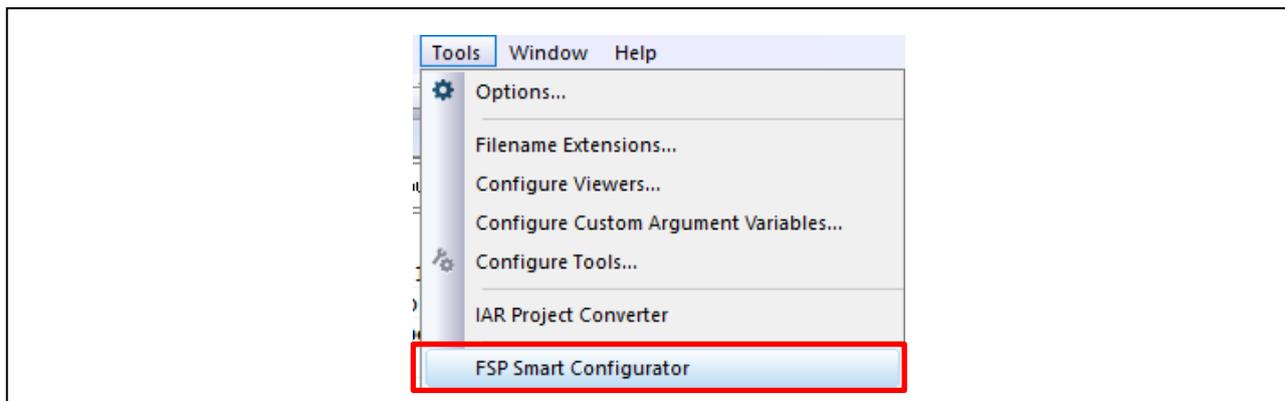
Figure 5-2 Start FSP Configurator

※　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_v2025-4.1_fsp_v3.0.0\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.
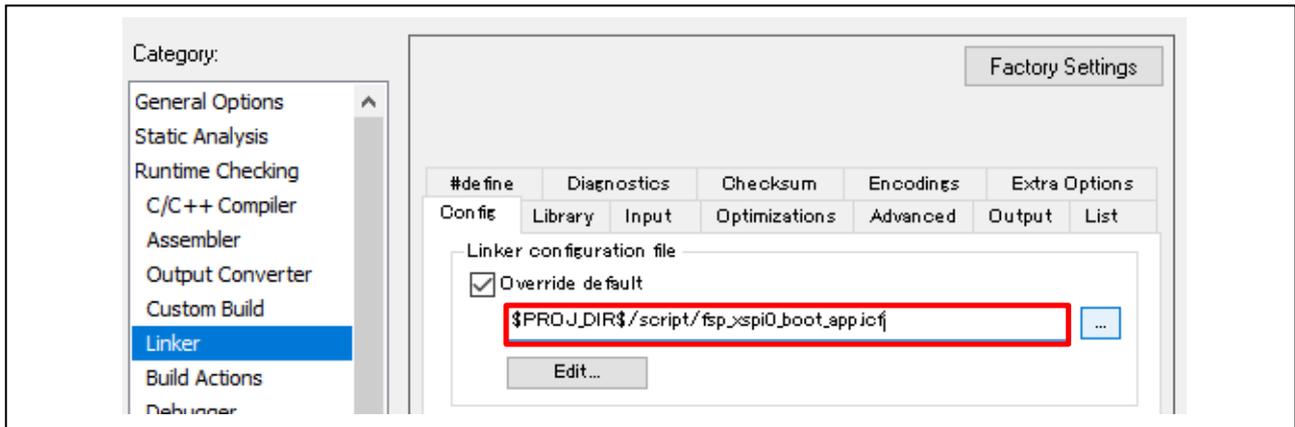
Figure 5-3 Linker script setting (application program)

5. Push "[Project] -> [Rebuild All]" to build the application program.

6. Then, select "RZT2M_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.
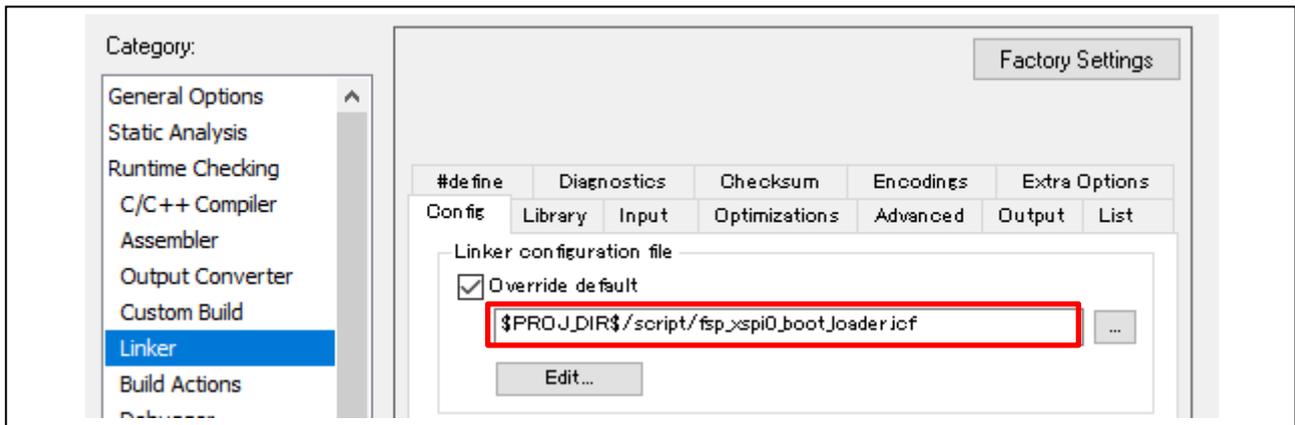
Figure 5-4 Linker script setting (loader program)

9. Push "[Project] -> [Rebuild All]" to build the application program.

10. Make sure that your PC and RZ/T2M RSK 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 .
Application program is already specified as extra image in loader project option.
"Right click loader project -> [Options...] -> [Debugger] -> [Images] -> [Download extra image]"

Path:
　$PROJ_DIR$\..\RZT2M_bsp_xspi0bootx1_app\Debug\Exe\RZT2M_bsp_xspi0bootx1_app.out
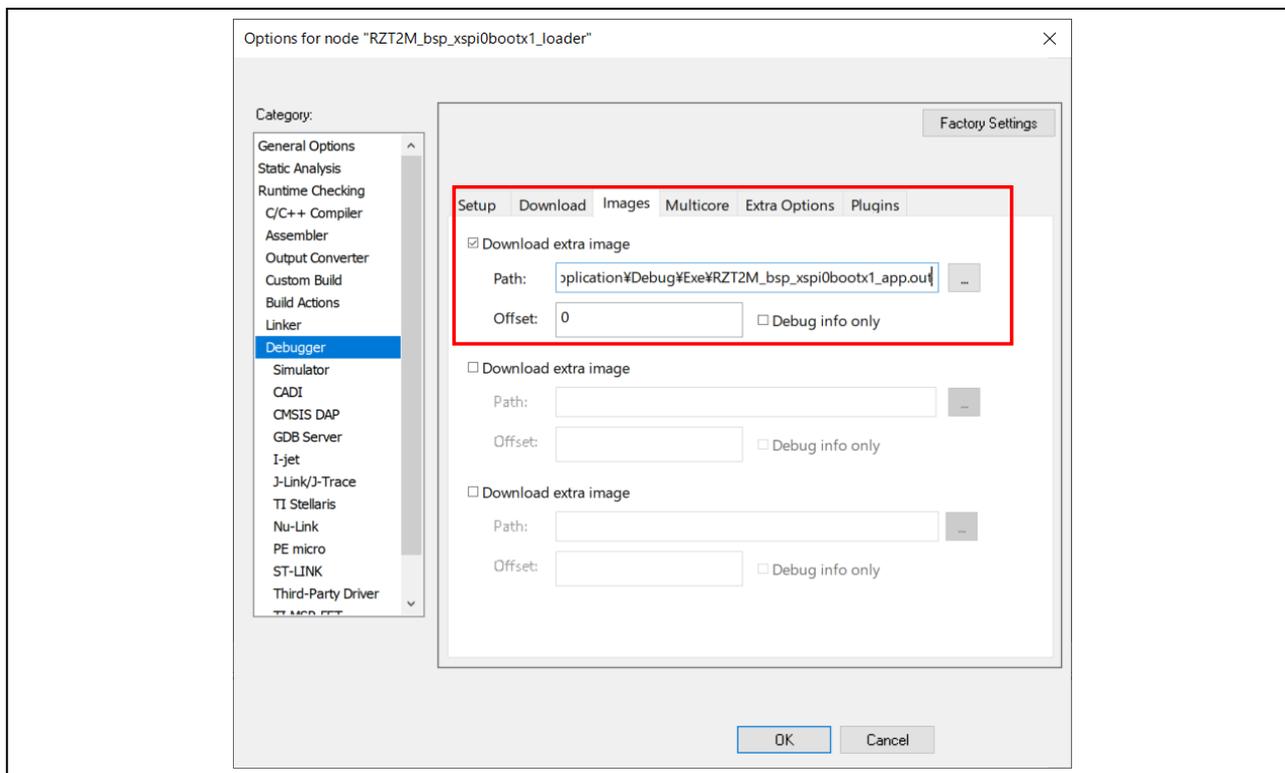
Figure 5-5 EWARM option setting (1)

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:
　$PROJ_DIR$\..\RZT2M_bsp_xspi0bootx1_app\Debug\Exe\RZT2M_bsp_xspi0bootx1_app.bin
Symbol:
　APPLICATION_PRG_SECTION
Section:
　APPLICATION_PRG_SECTION
Align:
　4
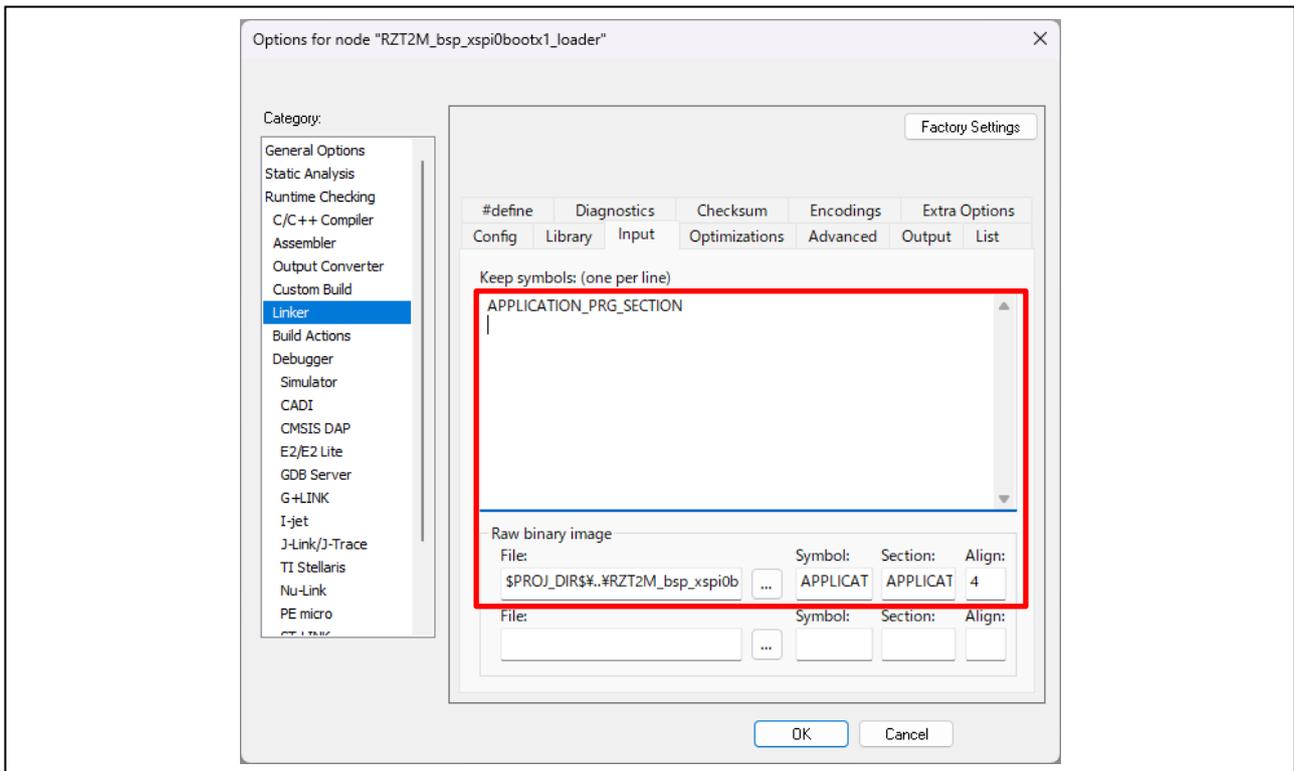
Figure 5-6 EWARM option setting (2)

Supplementary Notes for the EWARM Environment (3)

The following option settings will output raw binary build artifacts when building the Application project.

1. Open [Project] > [Options].
2. Open [Output Converter] and change [Output format] to "Raw binary".
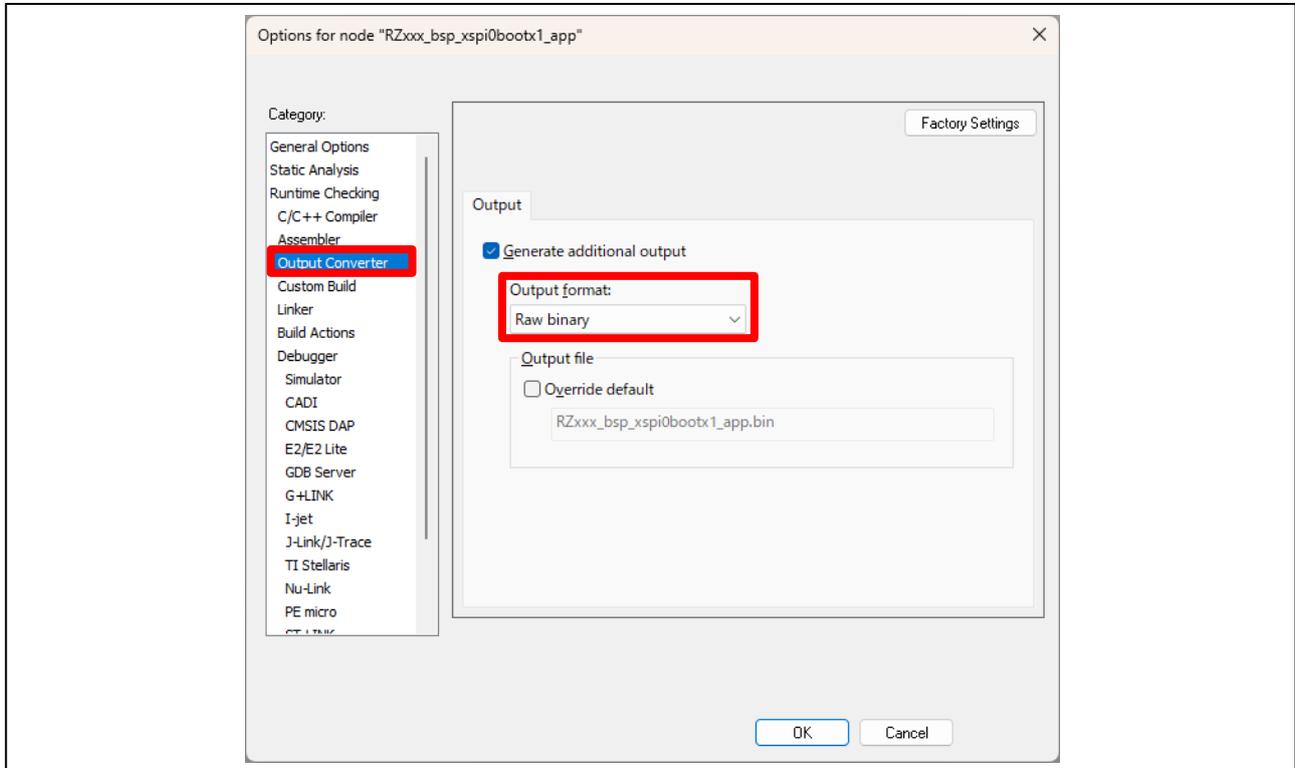
Figure 5-7 Configuring the output format for the Application project

Supplementary Notes for the EWARM Environment (4)

The following option settings remove the code generation actions that are set before compiling the Loader project and Application project.

This action is a function that generates code before compiling, and when code generation is performed, the path of the linker script file is overwritten with the default setting, so this setting prevents this.

1. Open [Project] > [Options].
2. Open [Build Actions], select the one with [Build order] set to "Pre-compile", and click [Remove].
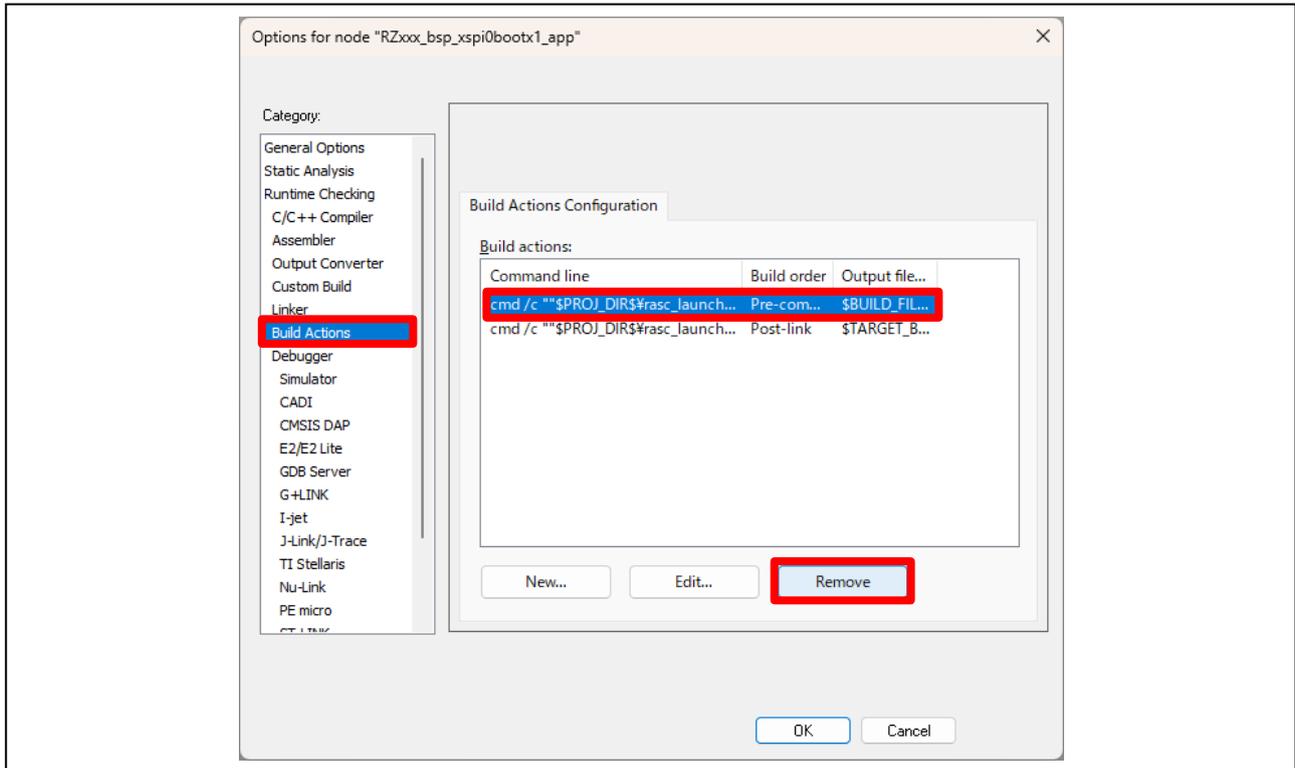
Figure 5-8 Remove Build Action

Supplementary Notes for the EWARM Environment (5)

The debug settings for the Loader project and Application project have been changed as follows:

[Loader Project]
1. Open [Project] > [Options].
2. Open [Debugger] > [Setup] and uncheck "Run to".
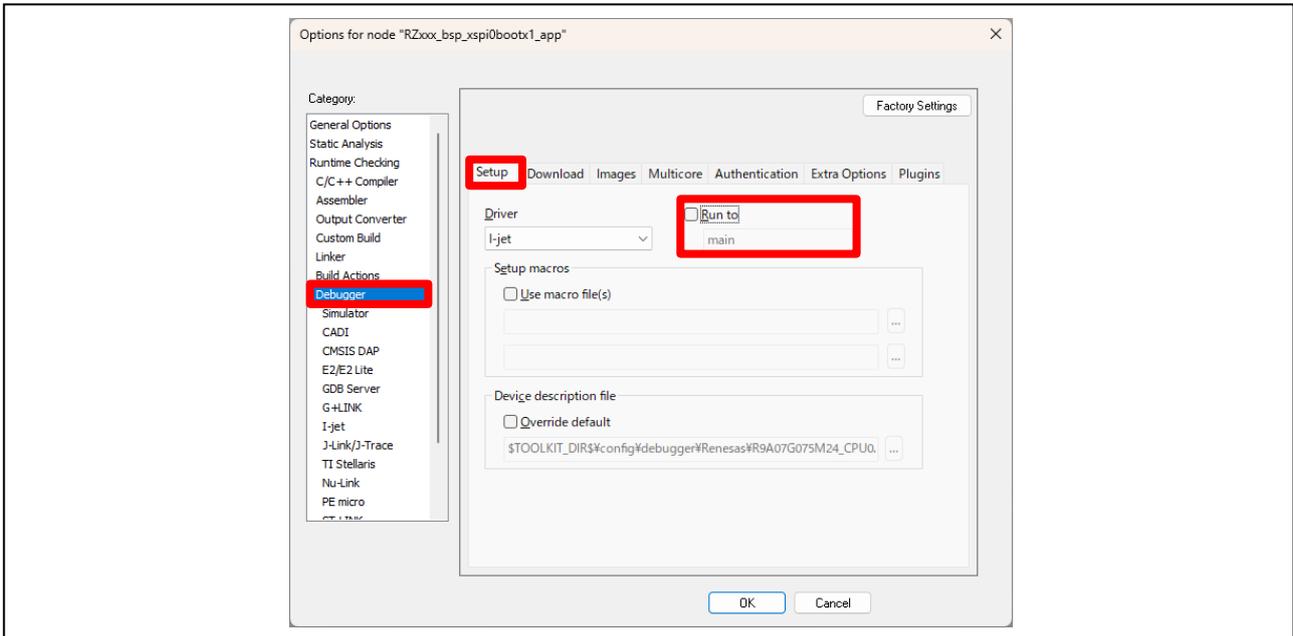3. Open [I-jet] > [Setup] and uncheck "From the probe".
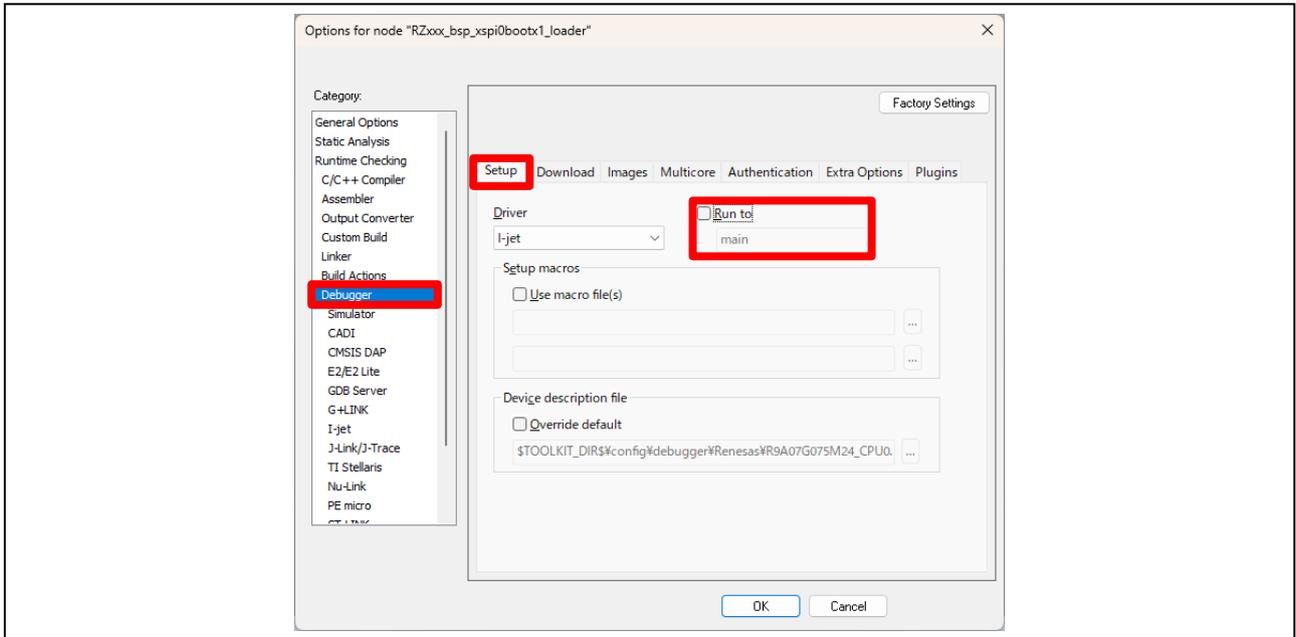
Figure 5-9 Loader project debug settings (1)
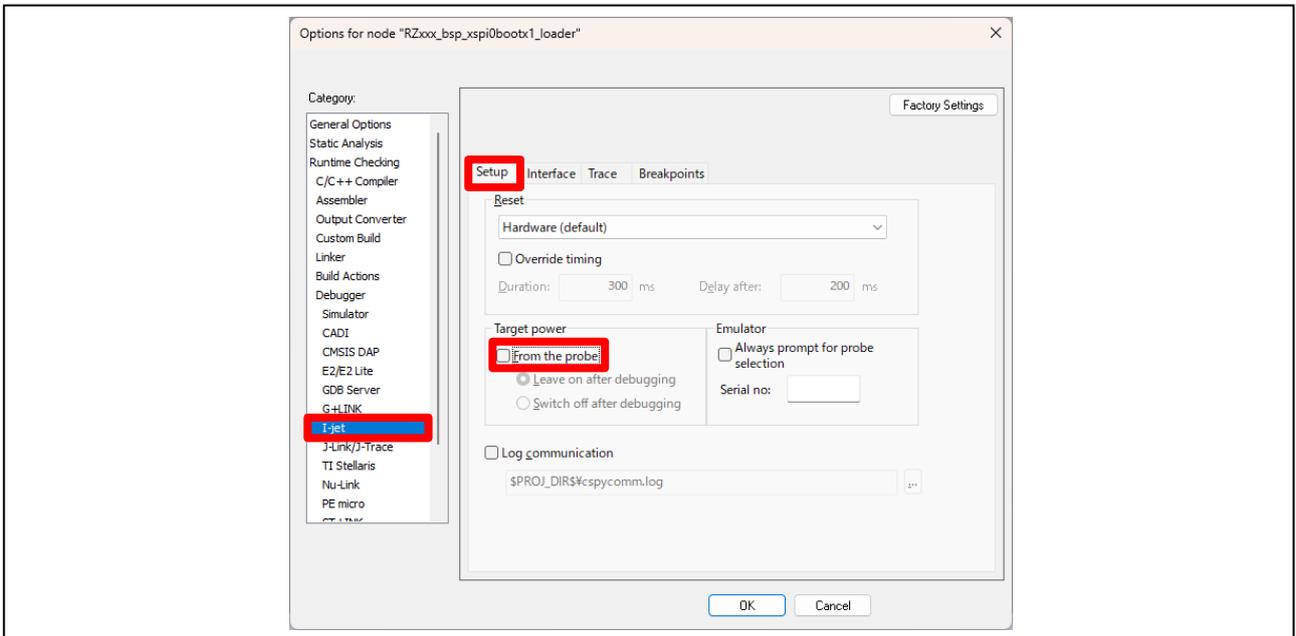
Figure 5-10 Loader project debug settings (2)

[Application Project]
1. Open [Project] > [Options].
2. Open [Debugger] > [Setup] and uncheck "Run to".
3. Open [I-jet] > [Setup], uncheck "From the probe", and change [Reset] to "Software".

Figure 5-11 Application project debug settings (1)
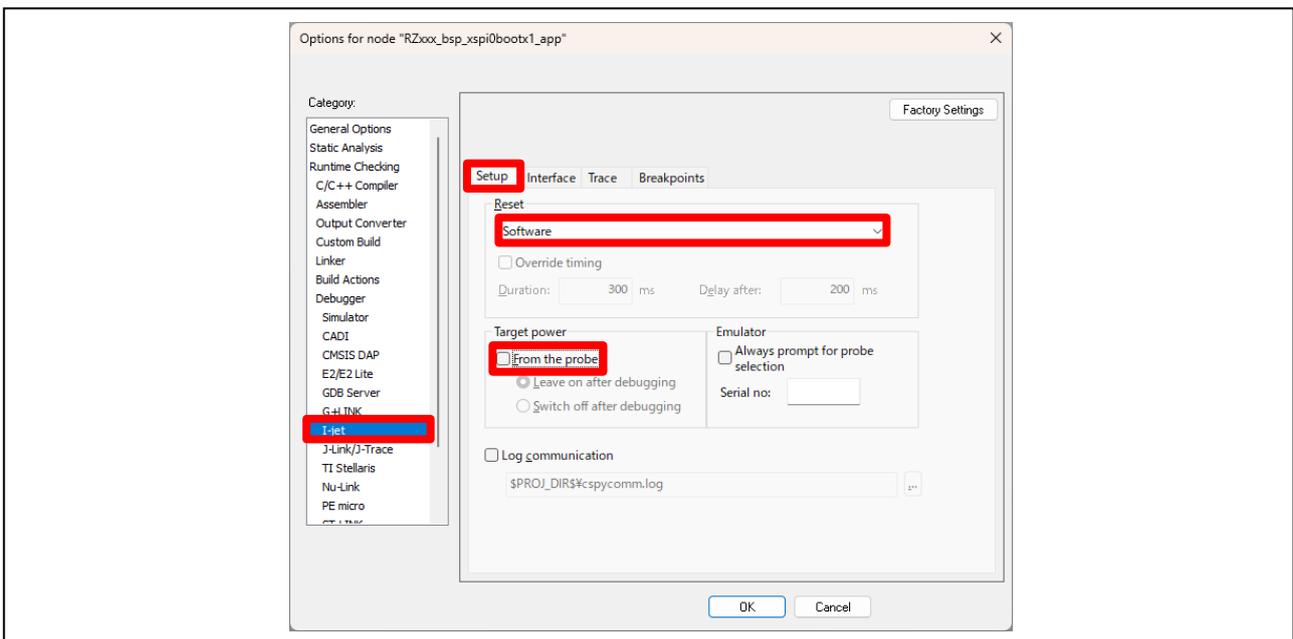
Figure 5-12 Application project debug settings (2)

Supplementary Notes for the EWARM Environment (6)

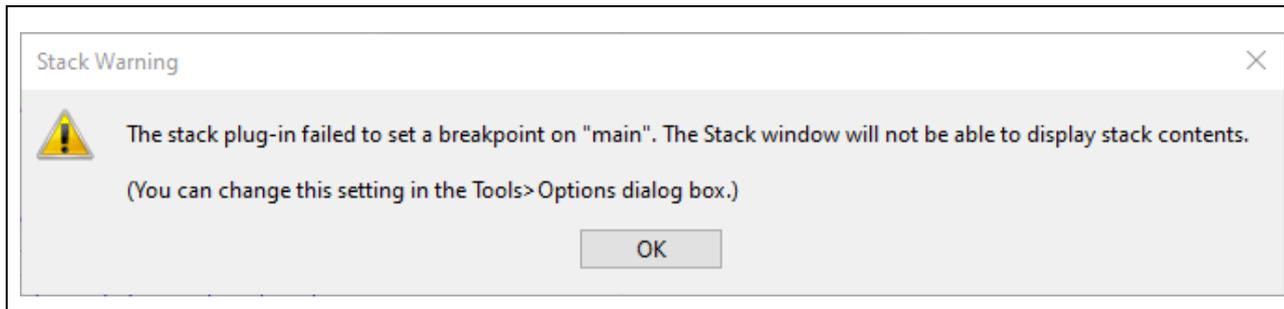When starting debug, the following warning about stack plug-in may be displayed.

Figure 5-13 Warnig about stack plug-in

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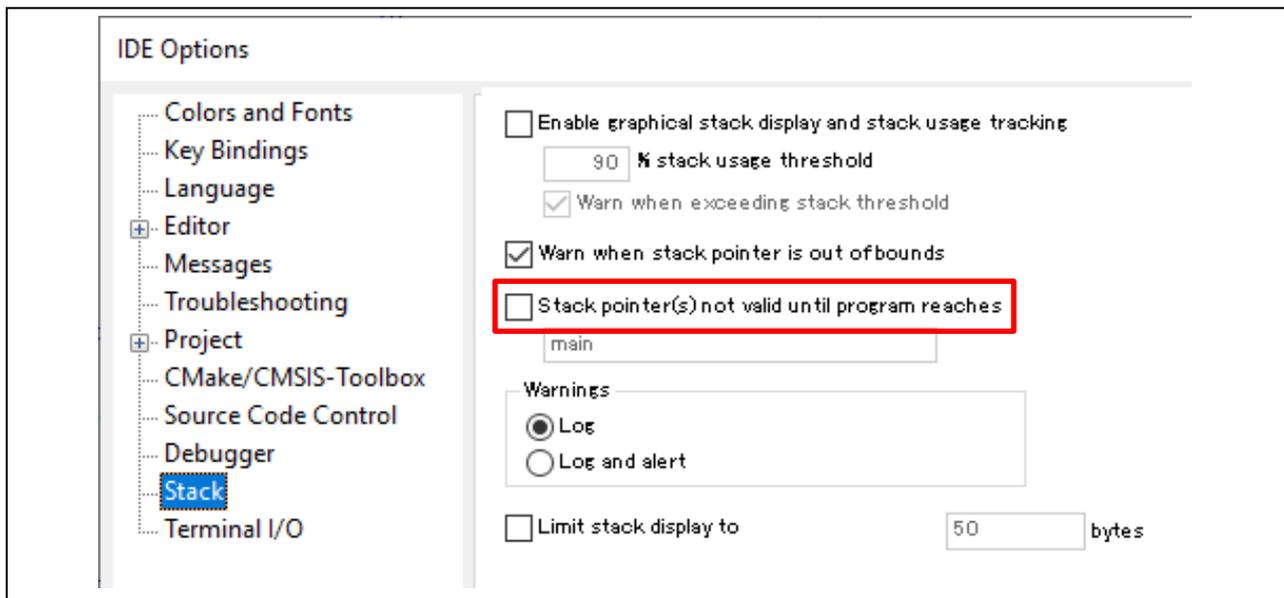Then, that warning window will not be displayed.

**Figure 5-14 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 "RZT2M_bsp_xspi0bootx1_app" project, push the [Generate Project Content] button to execute code generation.

4. After executing the clean of "RZT2M_bsp_xspi0bootx1_app", execute the build.

5. Open configuration.xml on "RZT2M_bsp_xspi0bootx1_loader" project, push the [Generate Project Content] button to execute code generation.

6. After executing the clean of "RZT2M_bsp_xspi0bootx1_loader", execute the build.

7. Make sure that your PC and RZ/T2M RSK board are connected with J-Link. Then, select "RZT2M_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).

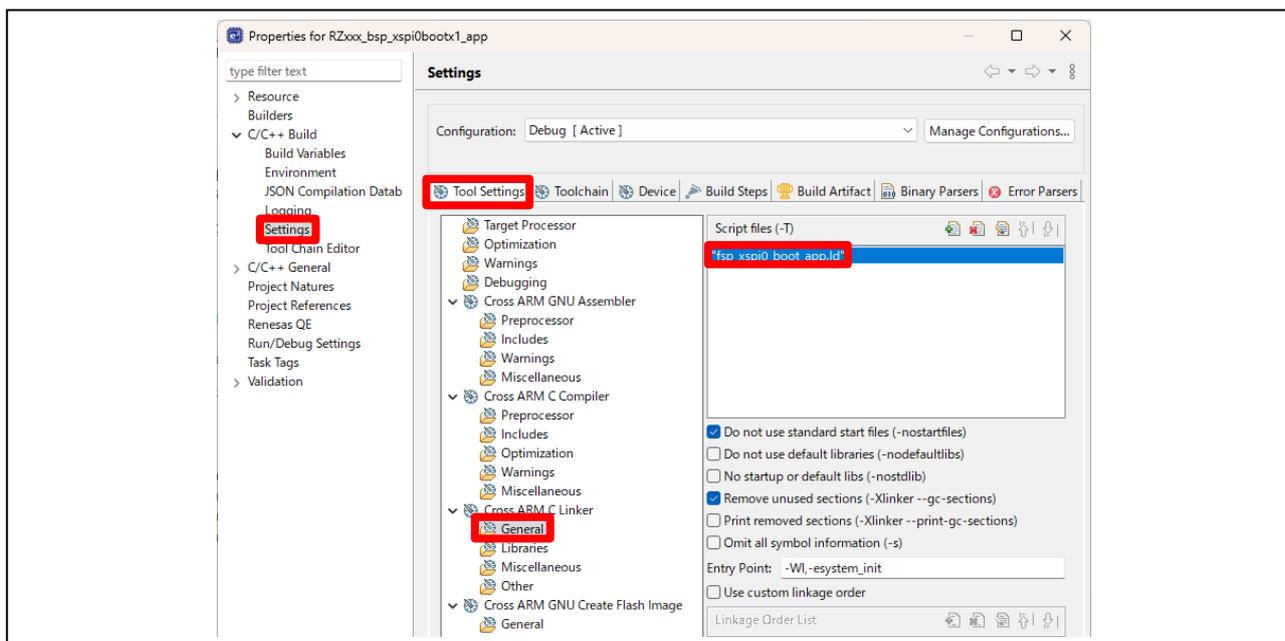**Supplementary Notes for the e² studio Environment (1)**

The following option settings change the paths of the linker script files for the Loader project and Application project.

1. Right-click the project node and open [Properties].
2. Open [C/C++ Build] > [Settings] > [Tool Settings] > [Cross ARM C Linker] > [General] and change the file path.
    [Loader Project]   : "fsp_xspi0_boot_loader.ld"
    [Application Project]   : "fsp_xspi0_boot_app.ld"
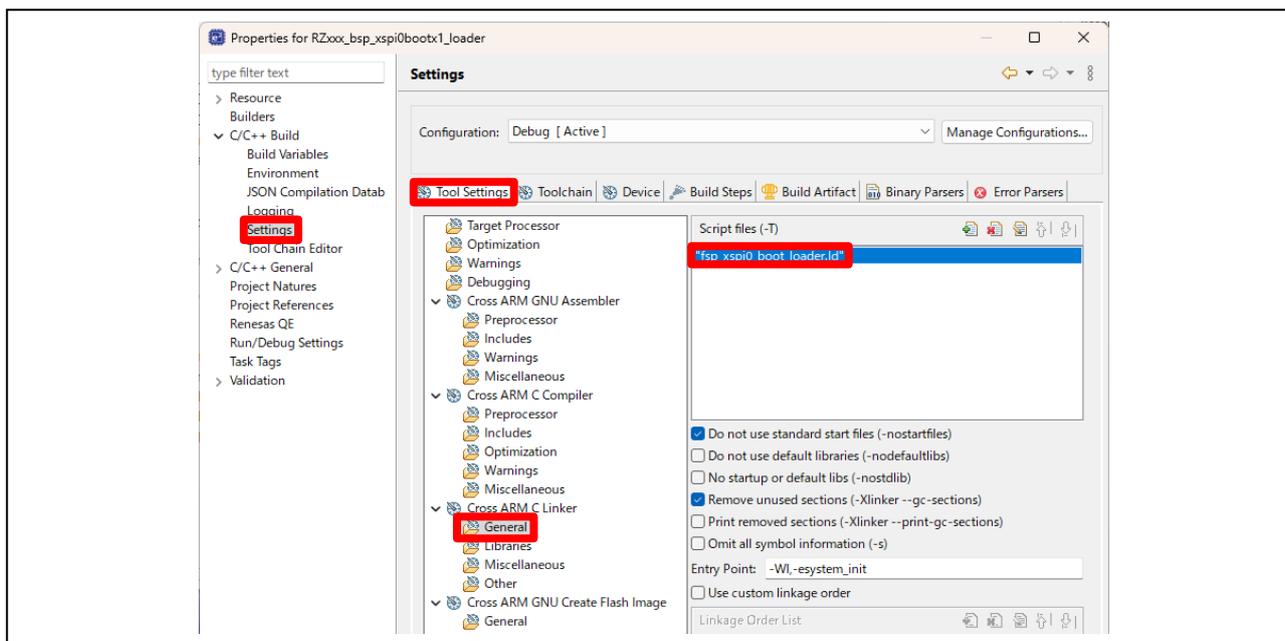
**Figure 5-15 Setting the linker script file path for the Loader project**

Figure 5-16 Setting the linker script file path for the Application project

Supplementary Notes for the e² studio Environment (2)

The following option settings will output raw binary build artifacts when building the Application project.

1. Right-click the Application project node and open [Properties].
2. Open [C/C++ Build] > [Settings] > [Tool Settings] > [Cross ARM GNU Create Flash Image] > [General] and change [Output file format (-O)] to "Raw binary".
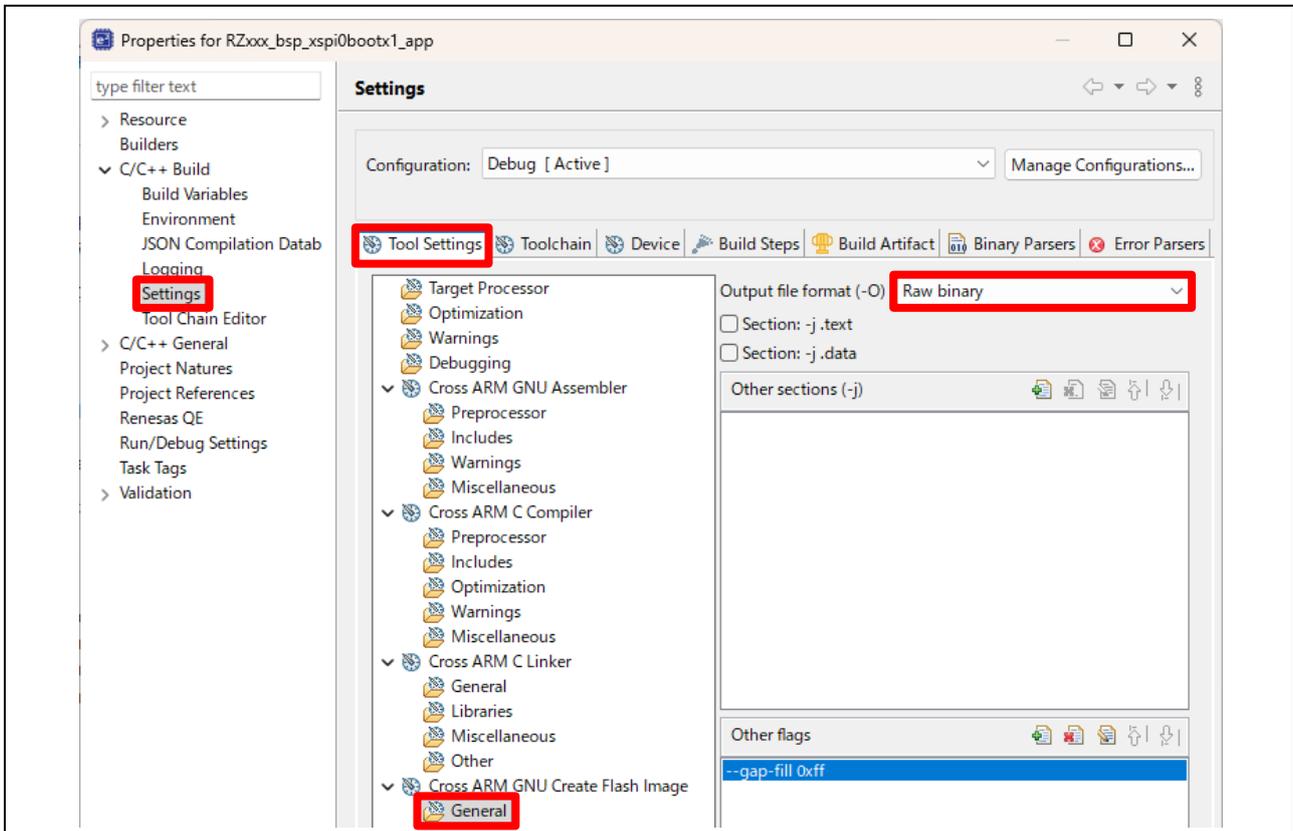
Figure 5-17 Configuring the output format for the Application project

Supplementary Notes for the e^2 studio Environment (3)

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.
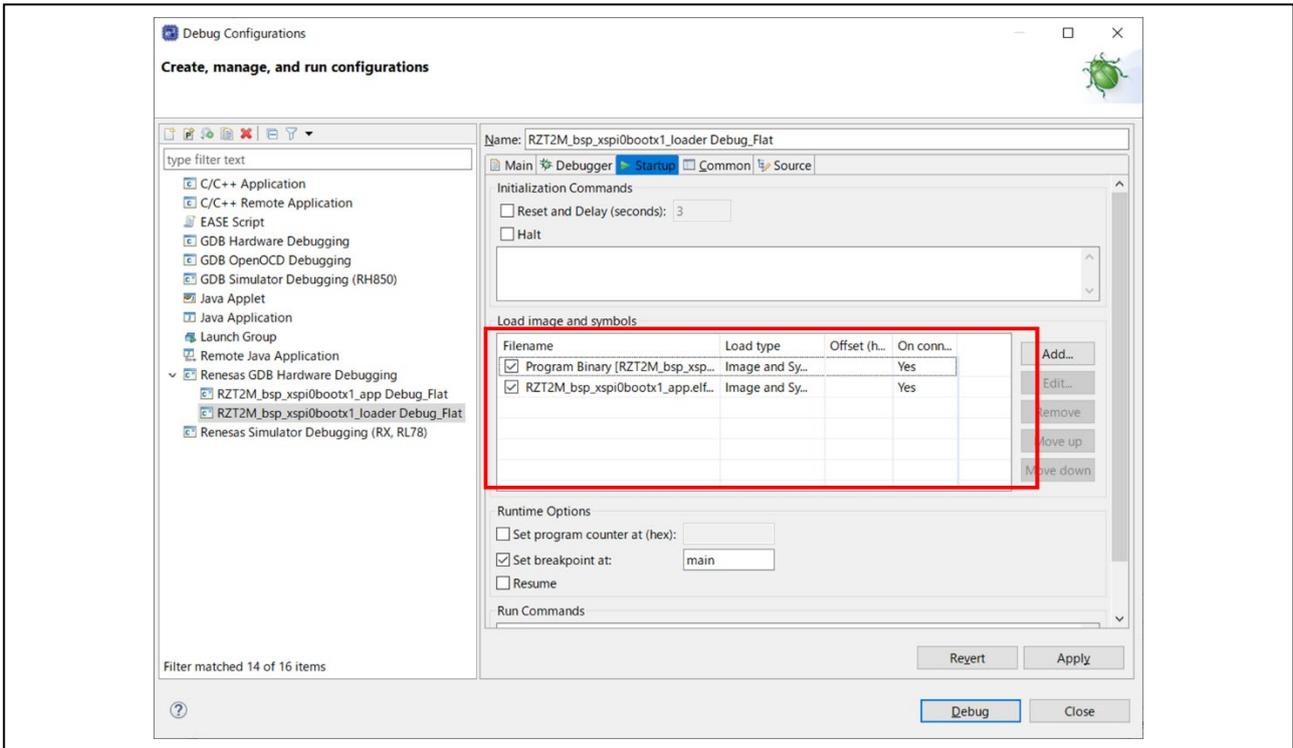
Figure 5-18 Debug configurations in e^2 studio

5.1.3 Supplementary Notes for the application and loader program

Supplementary Notes (1)

The following FSP settings disable the C Runtime in the Application project.
The C Runtime Initialization for the Application project is done in the Loader project.

1. Open the configuration.xml of Application project.
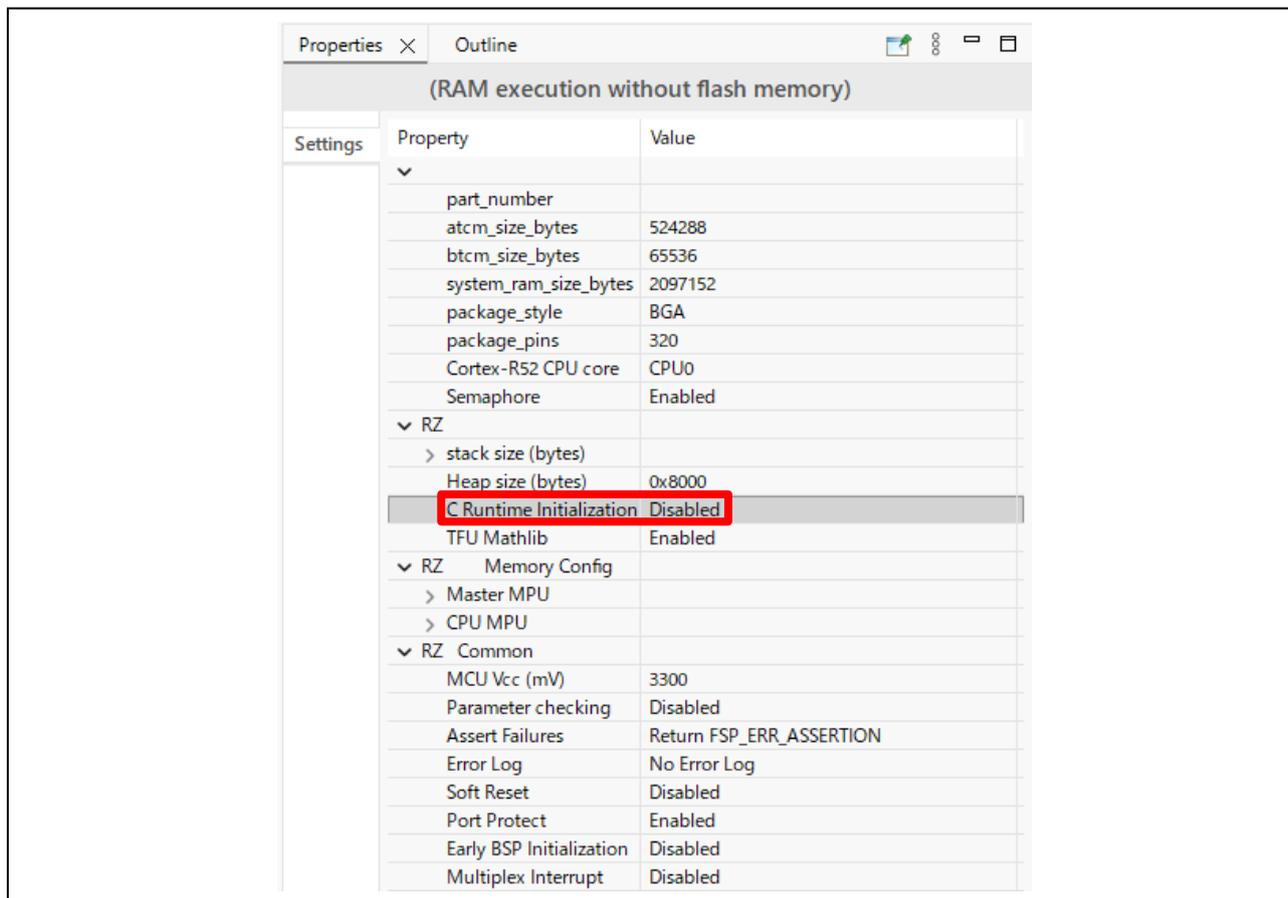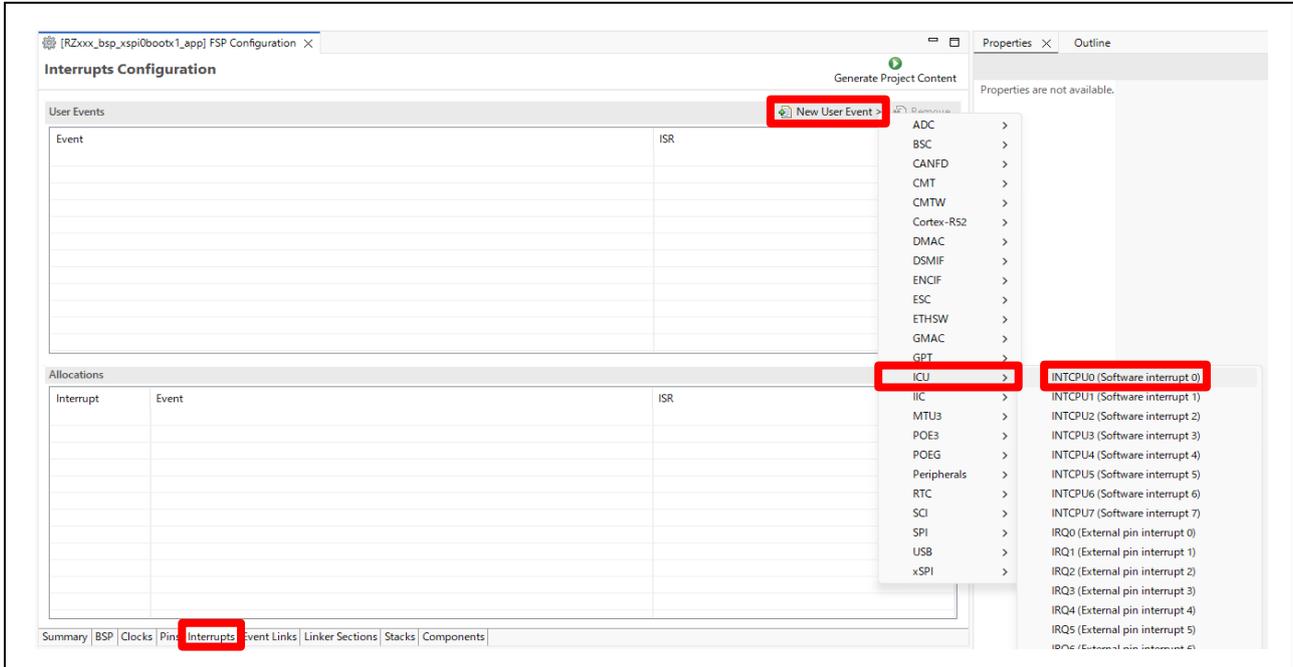2. Change [BSP] > [C Runtime Initialization] to "Disabled".

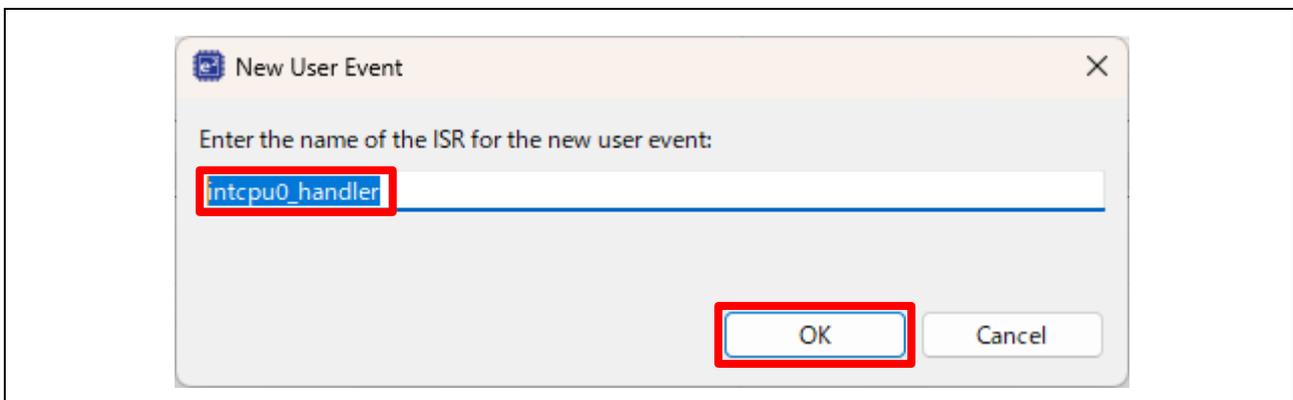**Figure 5-19 BSP settings for Application programs**

Supplementary Notes (2)

The following FSP settings configure the "INTCPU0" interrupt in the Application project:

1. Open the configuration.xml of Application project.
2. Select [Interrupts] > [New User Event] > [ICU] > [INTCPU0 (Software interrupt 0)] and set "intcpu0_handler".

Figure 5-20 Application program Interrupts settings (1)

Figure 5-21 Application program Interrupts settings (2)

Supplementary Notes (3)

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, #0x68bf              \n"
    "   movt  r1, #0x478              \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 5-22 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 5-23 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 0x60080000 size 64K];
define region APPLICATION_ROM_region  = mem:[from 0x60100000 size 1M];
~~~

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 0x60080000 size 64K];
define region APPLICATION_ROM_region  = mem:[from 0x60100000 size 1M];
~~~
```

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))
} > SYSTEM_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 = .;
} > FLASH_CONTENTS
~~~

After changing
~~~
.IMAGE_APP_RAM 0x00000000 : AT (0x00000000) /* Change copy destination address to ATCM */
{
    IMAGE_APP_RAM_start = .;
    KEEP(*(APP_IMAGE_RAM))
} > SYSTEM_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 = .;
} > FLASH_CONTENTS
~~~
```

fsp_xspi0_boot_app.ld (Linker script for application program)

```
Default
~~~
.text 0x10080000 : AT (TEXT_IMAGE)
{
    ~~~
    _text_end = .;
} > SYSTEM_RAM

    ~~~
    __extab_end = .;
} > SYSTEM_RAM

    ~~~
    __exidx_end = .;
} > SYSTEM_RAM

    ~~~
    _data_end = .;
} > SYSTEM_RAM

    ~~~
    _end = .;
} > SYSTEM_RAM

    ~~~
    __HeapLimit = .;
} > SYSTEM_RAM

    ~~~
    __AArch64StackLimit = .;
} > SYSTEM_RAM

    ~~~
    __ThreadStackLimit = .;
} > SYSTEM_RAM
```

```
    ~~~
    __AbtStackLimit = .;
} > SYSTEM_RAM

~~~
```

<u>After Changing</u>
```
~~~
.text 0x00000000 : AT (TEXT_IMAGE) /* Change code area to ATCM */
{
    ~~~
    _text_end = .;
} > ATCM

    ~~~
    __extab_end = .;
} > ATCM

    ~~~
    __exidx_end = .;
} > ATCM

    ~~~
    _data_end = .;
} > ATCM

    ~~~
    _end = .;
} > ATCM

    ~~~
    __HeapLimit = .;
} > ATCM

    ~~~
    __AArch64StackLimit = .;
} > ATCM

    ~~~
    __ThreadStackLimit = .;
} > ATCM

    ~~~
    __AbtStackLimit = .;
} > ATCM

~~~
```

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\RZT2M_bsp_xspi0bootx1_app_CPU1
   \RZT2M_bsp_xspi0bootx1_app_CPU1.eww".

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

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

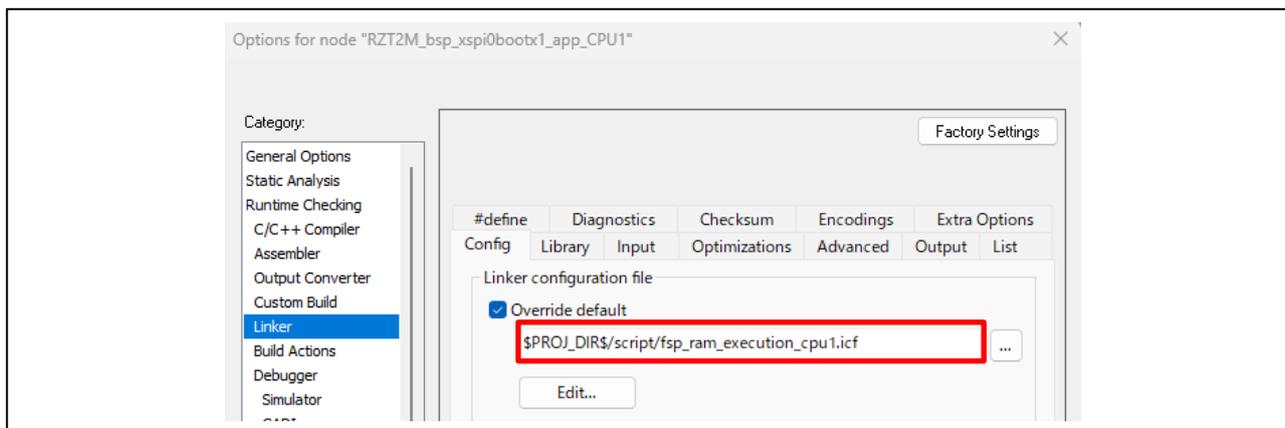   Set " **$PROJ_DIR$/script/ fsp_ram_execution_cpu1.icf** " in Linker configuration file.

Figure 5-24 CPU1 Linker script setting

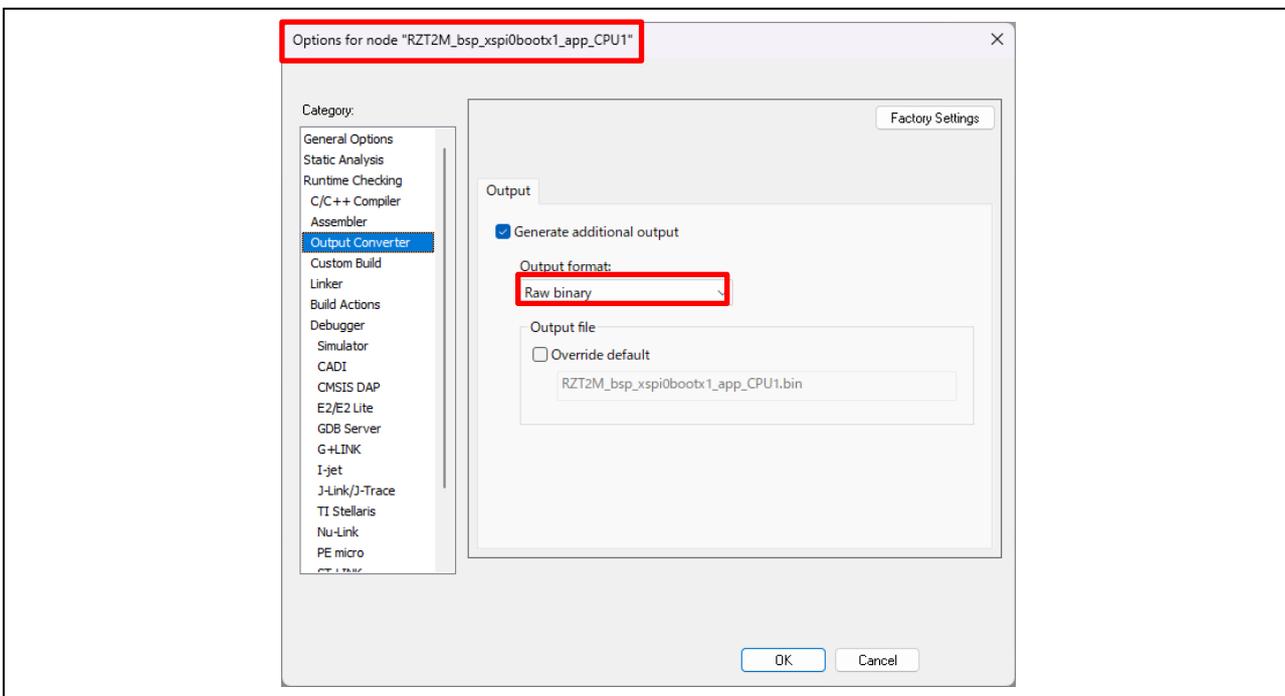Build the CPU1 project. The following project option settings will output build artifacts in raw binary.

Figure 5-25 CPU1 program option settings

After building the CPU1 program, close the RZT2M_bsp_xspi0bootx1_app_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   : $PROJ_DIR$\..\RZT2M_bsp_xspi0bootx1_app_CPU1\
         Debug\Exe\RZT2M_bsp_xspi0bootx1_app_CPU1.bin
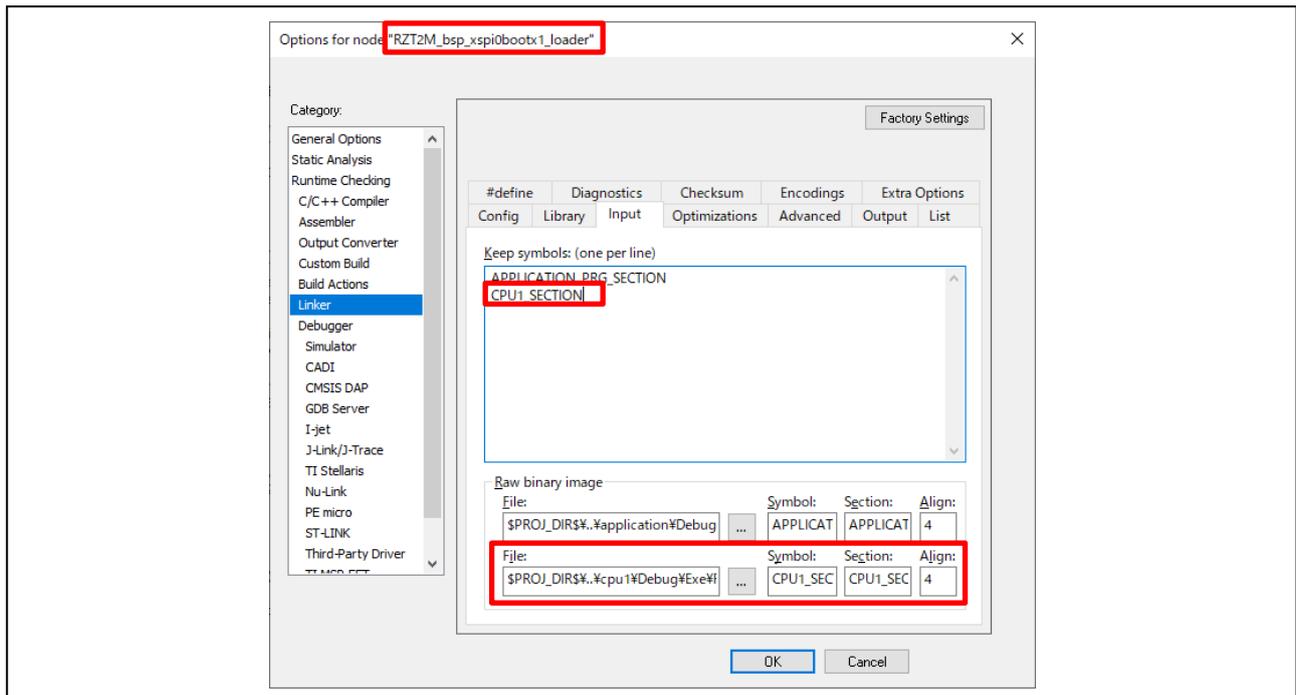 Symbol   : CPU1_SECTION
  Section   : CPU1_SECTION
  Align   : 4

Figure 5-26 CPU0 loader program option settings

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**
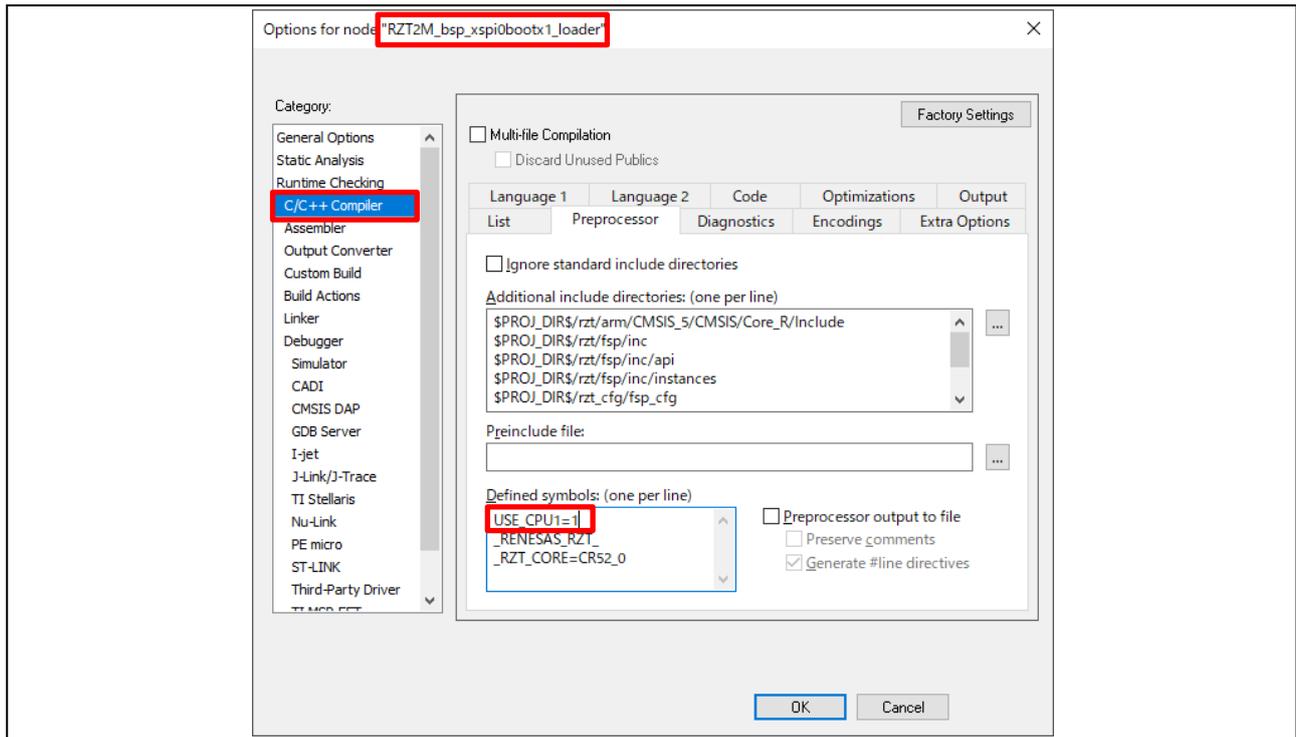　[Options] -> [C/C++ Compiler] -> [Preprocessor]: USE_CPU1=1

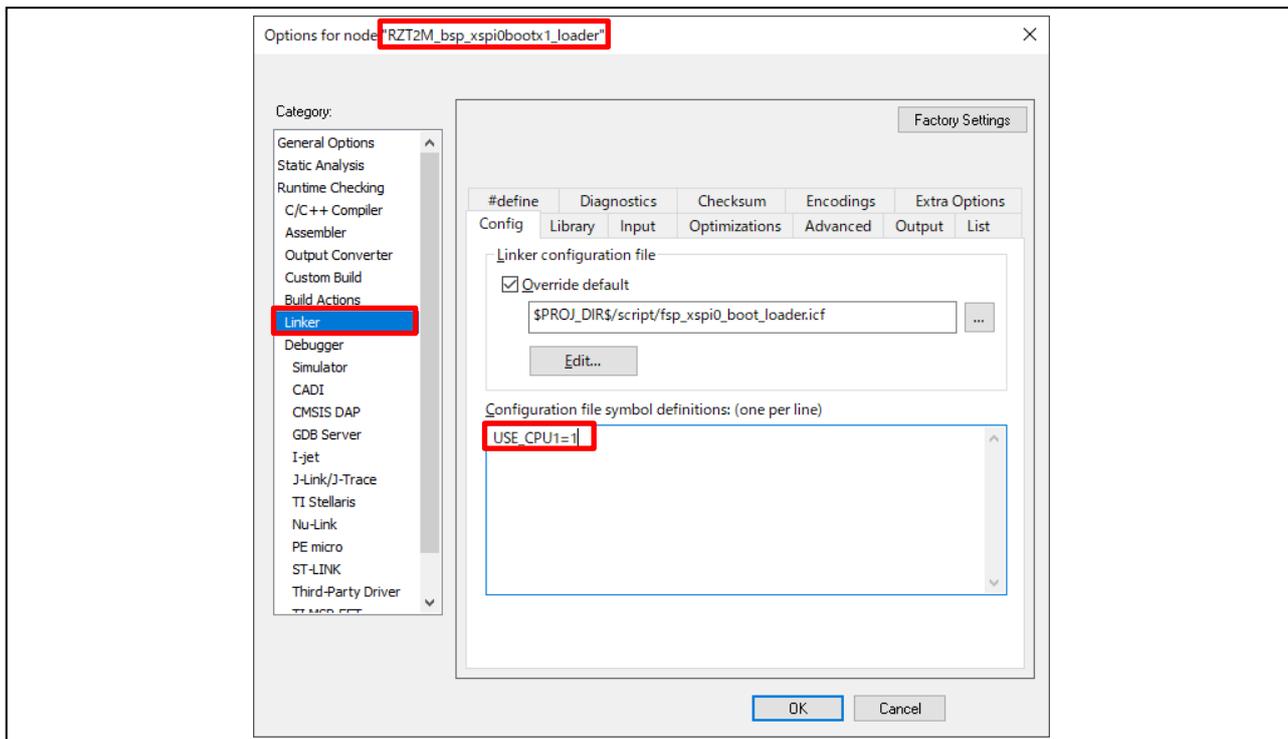**Figure 5-27 Enabling USE_CPU1 definition of CPU0 loader program (1/2)**

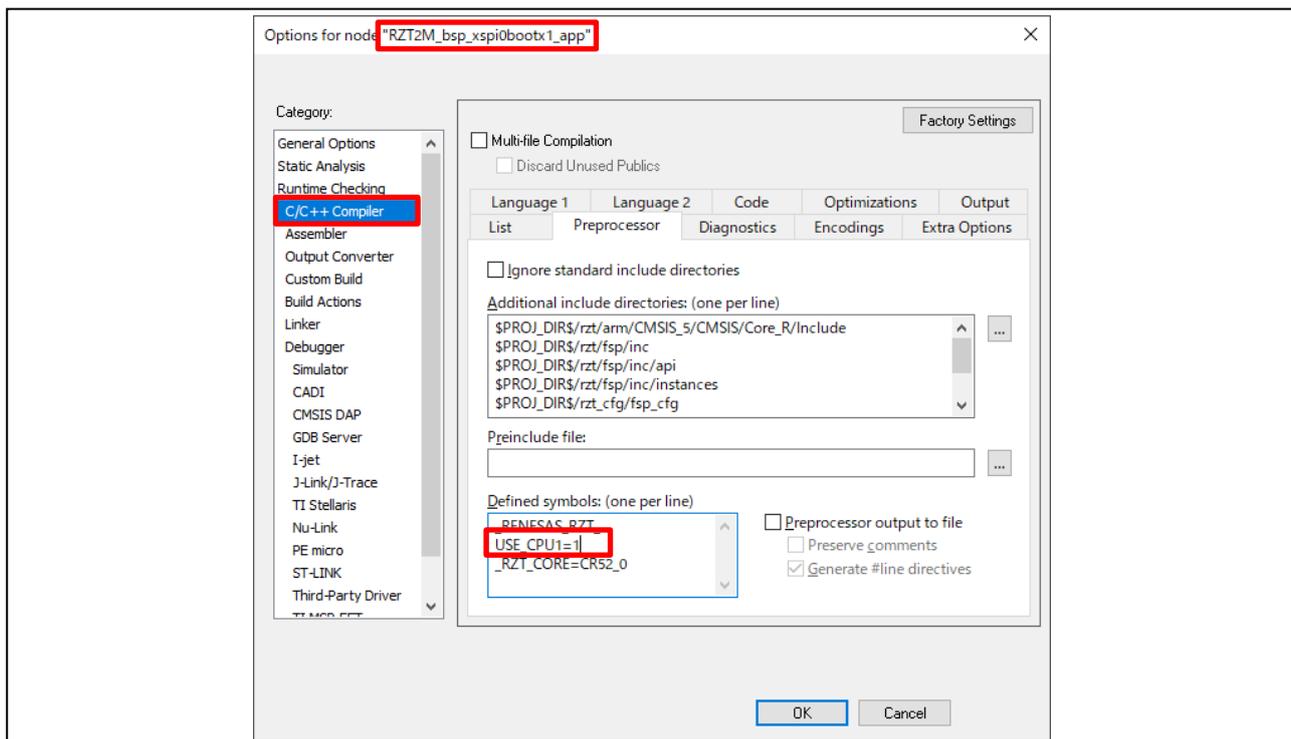**Figure 5-28 Enabling USE_CPU1 definition of CPU0 loader program (2/2)**

Figure 5-29 Enabling USE_CPU1 definition of CPU0 application program

4. Rebuild the CPU0 project

Rebuild the projects in the order of application program -> loader program.

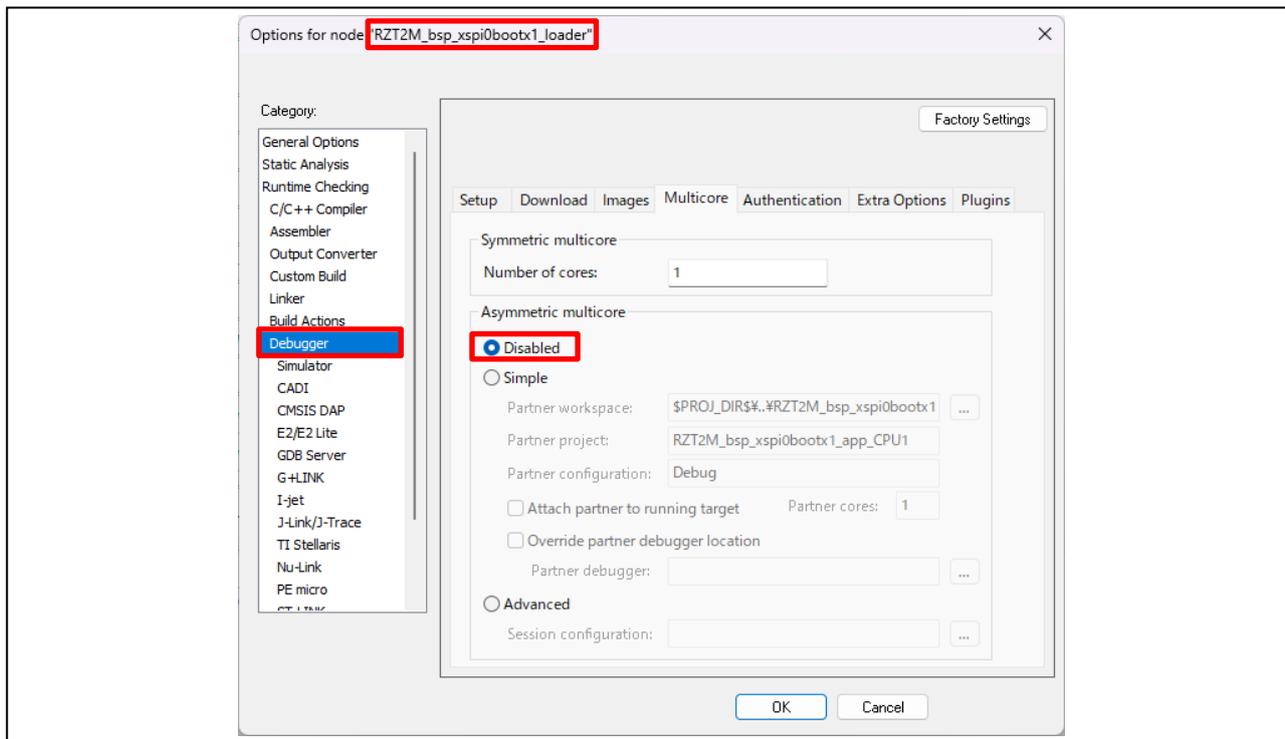Select "Disable" from [Project] > [Options...] > [Debugger] > [Multicore].

Figure 5-30 Multicore settings for CPU0 loader program

Select the "RZT2M_bsp_xspi0bootx1_loader Debug" project from the workspace, and with the RZ/T2M RSK board and I-jet connected, select [Project] -> [Download] -> [Download File] and select "RZT2M_bsp_xspi0bootx1_loader.out".
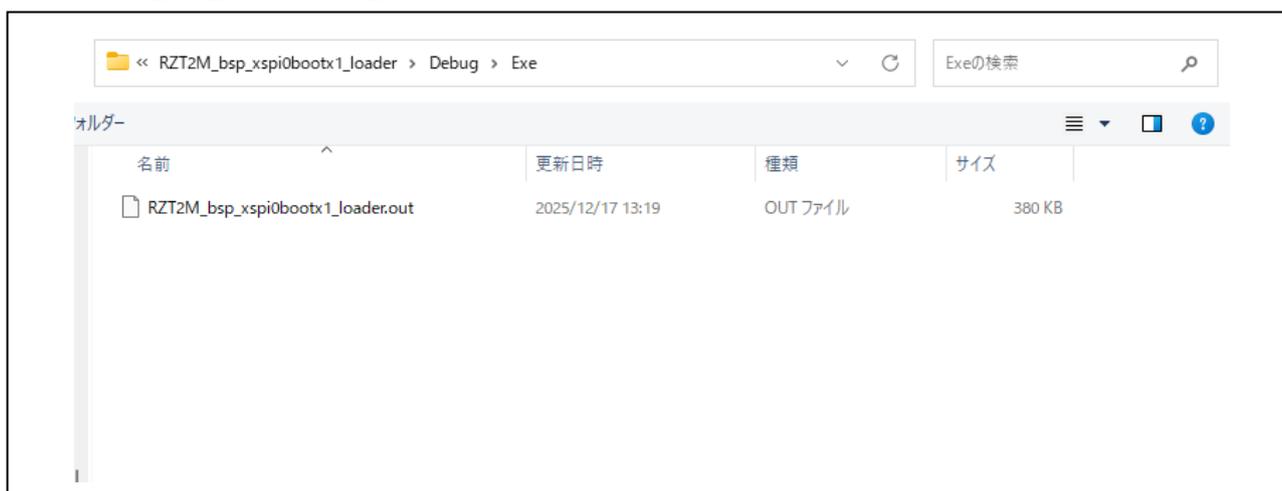

Figure 5-31 Select out file for CPU0 loader program

5. 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   : $PROJ_DIR$\..\RZT2M_bsp_xspi0bootx1_app_CPU1\
                                RZT2M_bsp_xspi0bootx1_app_CPU1.eww
    Partner project   : RZT2M_bsp_xspi0bootx1_app_CPU1
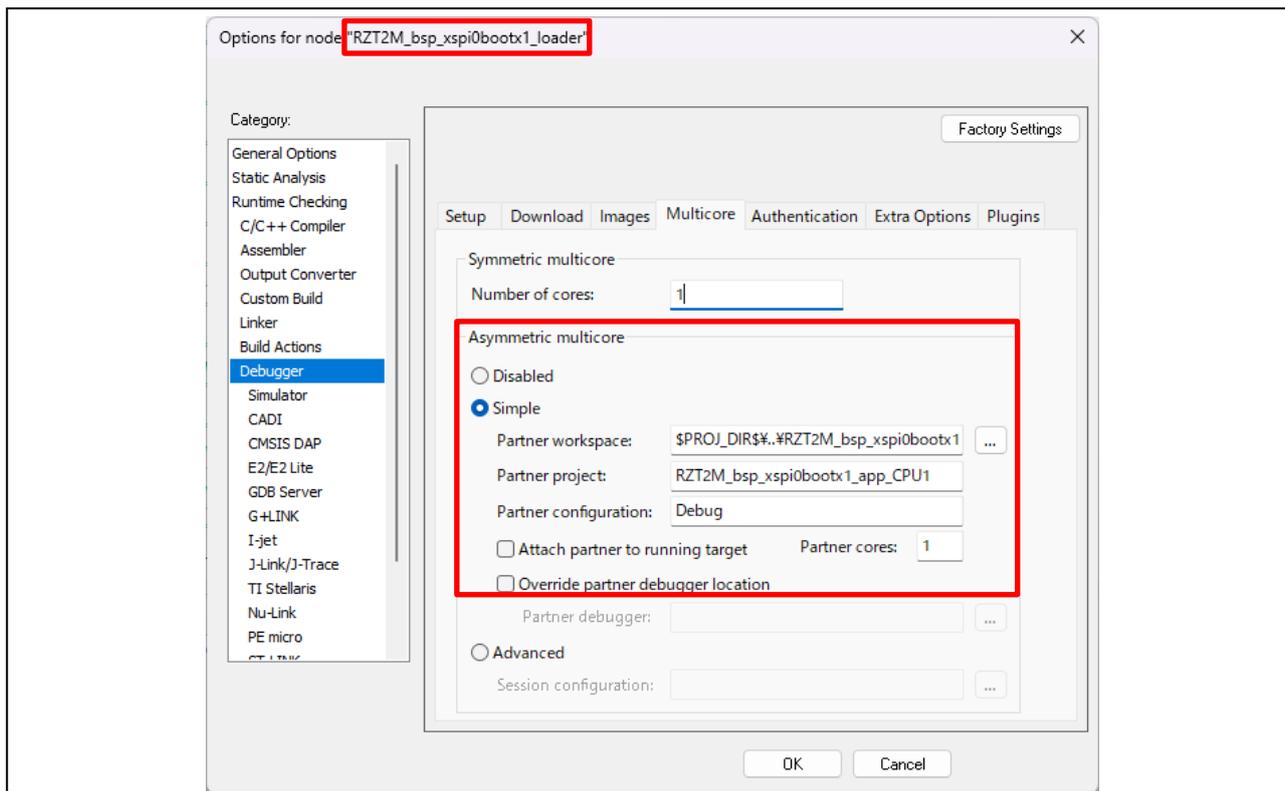   Partner configuration   : Debug

Figure 5-32 Multicore debug settings for CPU0 loader program

6. Run the project

    Select "Debug without Downloading" 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.

Supplementary Notes for the EWARM Environment (1)

The following option settings add a command line to the debugger additional options for the CPU1 project.

1. Open [Project] > [Options].
   Open [Debugger] > [Extra Options] and add "--macro_param cpu1_enable=1" to [Command line options: (one per line)].
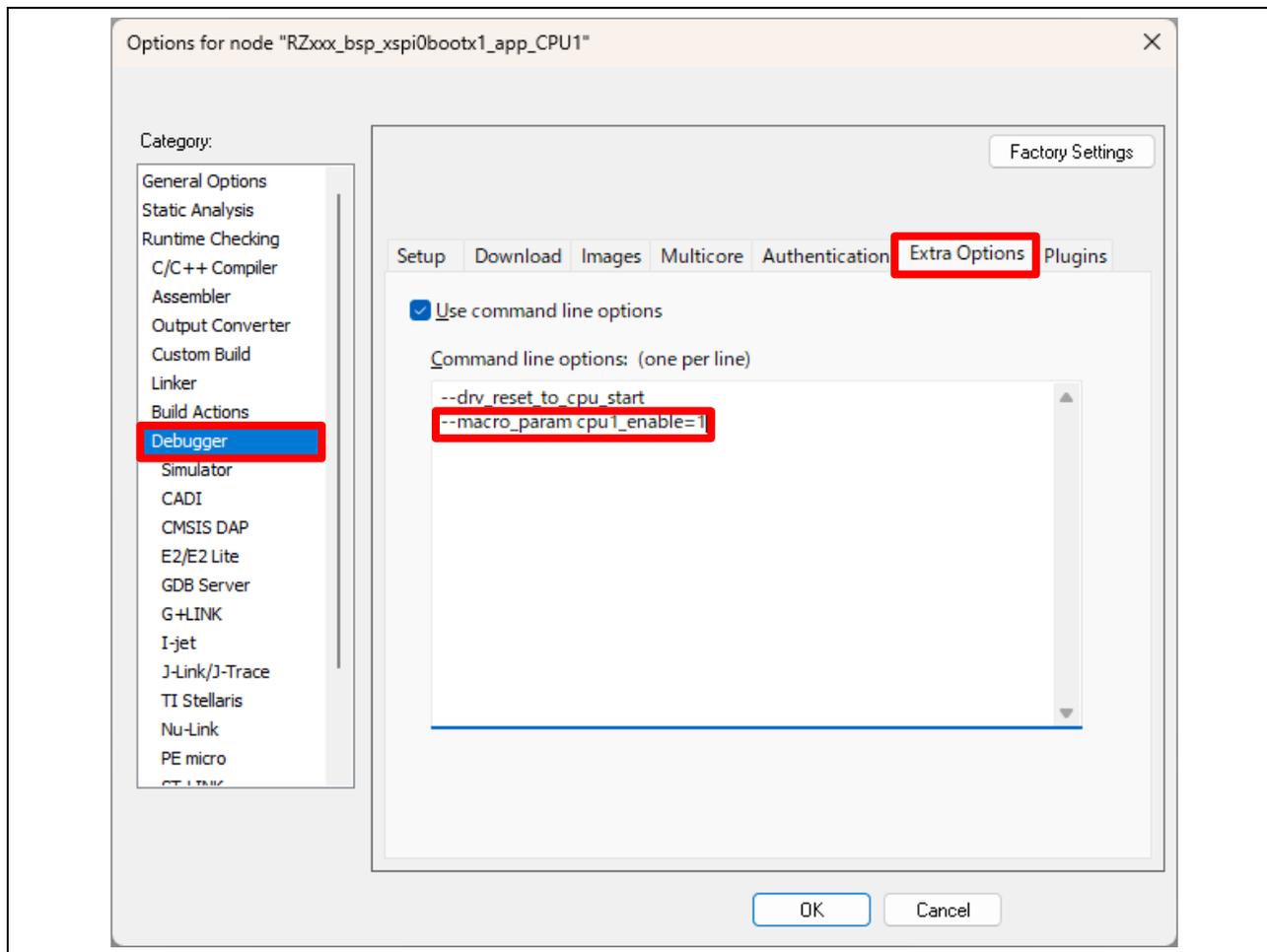

Figure 5-33 Command line settings for CPU1 project

Supplementary Notes for the EWARM Environment (2)

The following option settings remove the code generation actions set before compilation.

This action is a function that generates code before compiling, and when code generation is performed, the path of the linker script file is overwritten with the default setting, so this setting prevents this.

    1.   Open [Project] > [Options].

    2.   Open [Build Actions], select the one with [Build order] set to "Pre Compile", and click [Remove].
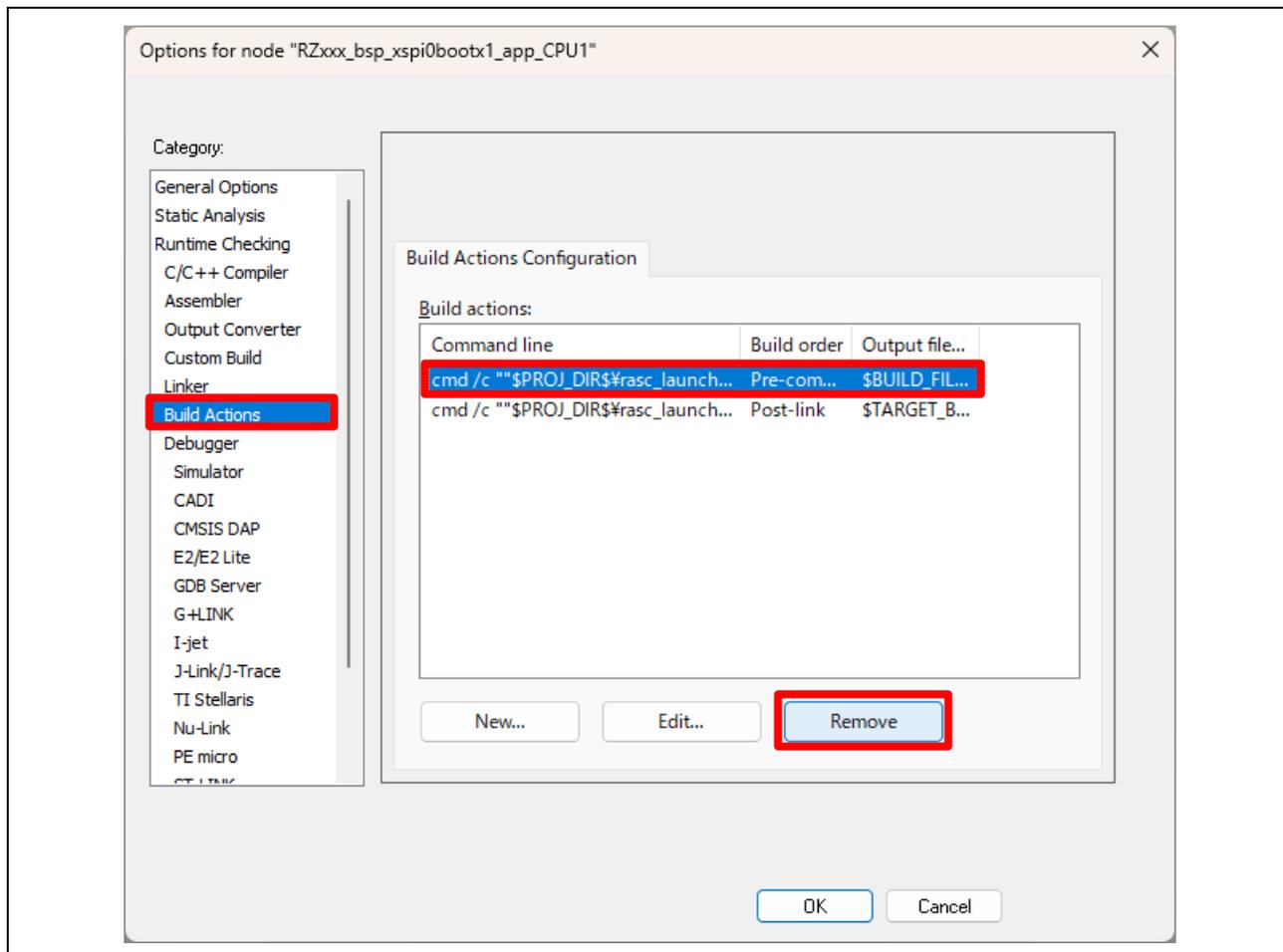
Figure 5-34 Remove Build Actions for CPU1 project

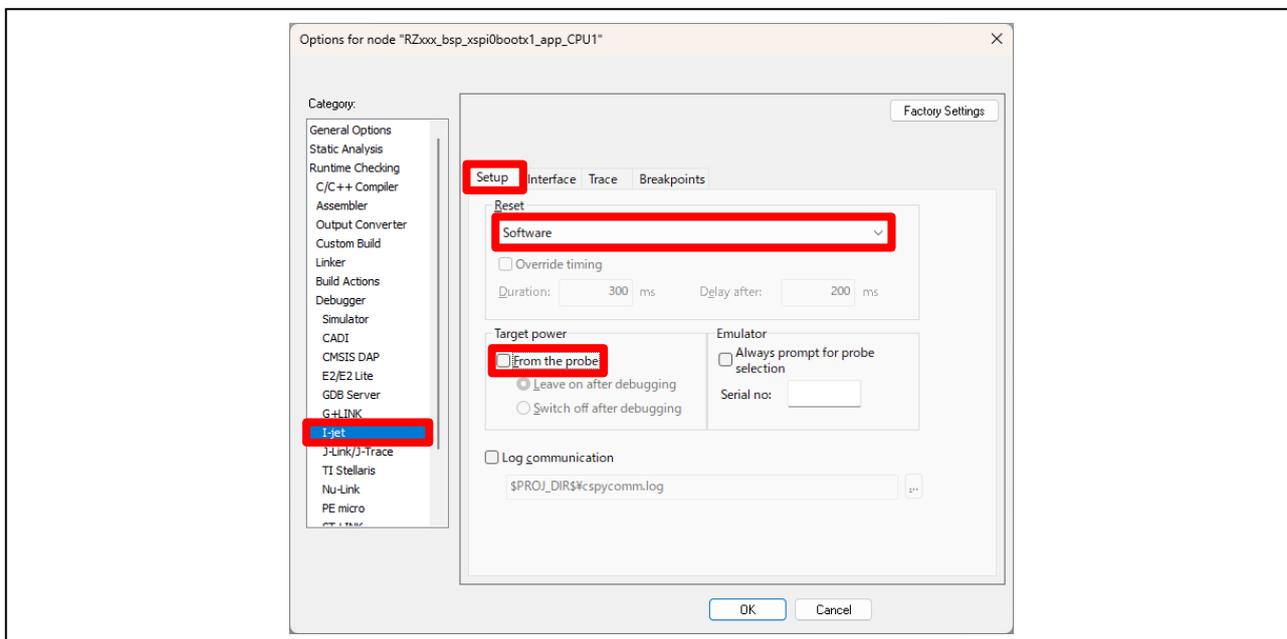**Supplementary Notes for the EWARM Environment (3)**

The debug settings for the CPU1 project have been changed as follows:

1. Open [Project] > [Options].
2. Open [Debugger] > [Setup] and uncheck "Run to".
3. Open [I-jet] > [Setup], uncheck "From the probe", and change [Reset] to "Software".
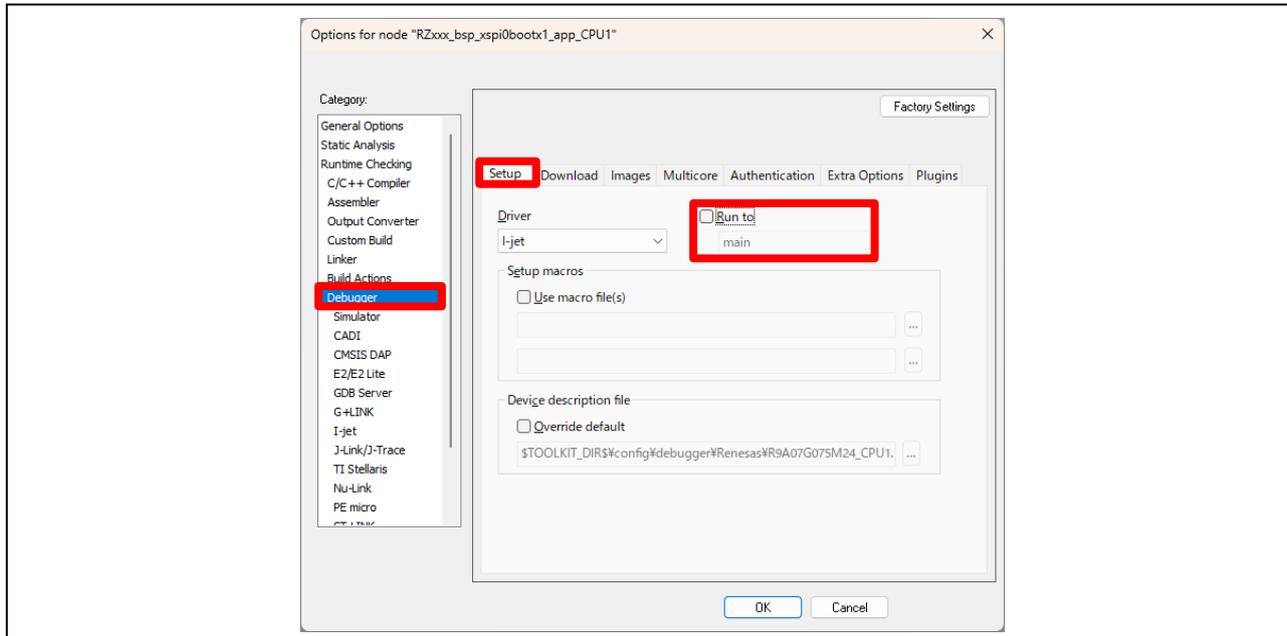
Figure 5-35 Debug settings for CPU1 project (1)

Figure 5-36 Debug settings for CPU1 project (2)

5.3.2　e² studio from Renesas

1.　Build CPU1 program

Open configuration.xml on "RZT2M_bsp_xspi0bootx1_app_CPU1" project, push the [Generate Project Content] button to execute code generation.

Build the "RZT2M_bsp_xspi0bootx1_app_CPU1" program. The following project option settings will output build artifacts in raw binary.
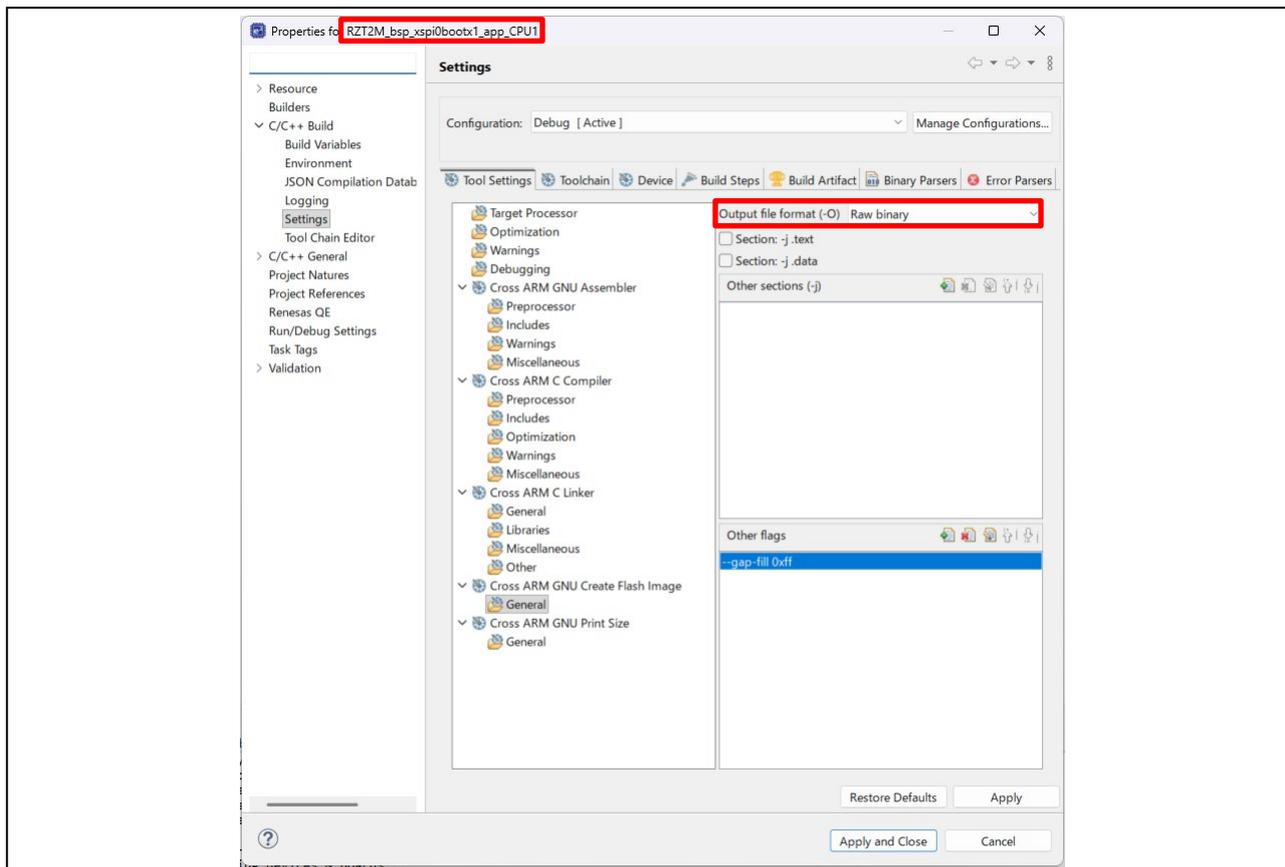

Figure 5-37 CPU1 program option settings

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))
    } > SYSTEM_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 = .;
    } > FLASH_CONTENTS

    .IMAGE_CPU1_RAM 0x10000000 : AT (0x10000000) /* CPU1 program, RAM section for execution. */
    {
        IMAGE_CPU1_RAM_start = .;
        KEEP(*(CPU1_IMAGE_RAM))
    } > SYSTEM_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 = .;
    } > FLASH_CONTENTS


~~~~~~~~
}

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
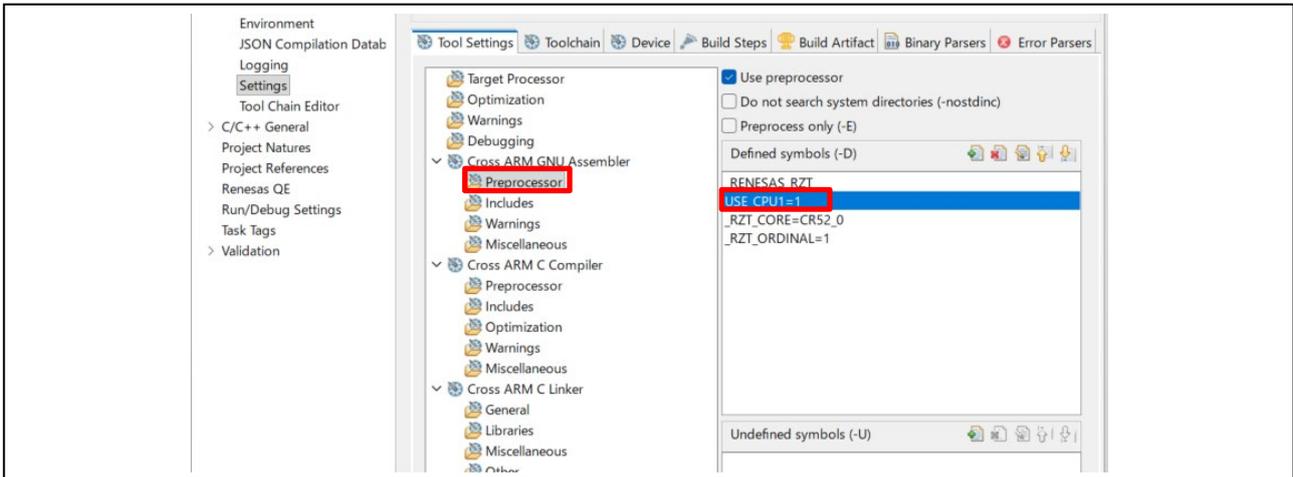   [Cross ARM C Compiler] -> [Preprocessor]: USE_CPU1=1

Figure 5-38 Enabling USE_CPU1 definition of CPU0 program (1/2)
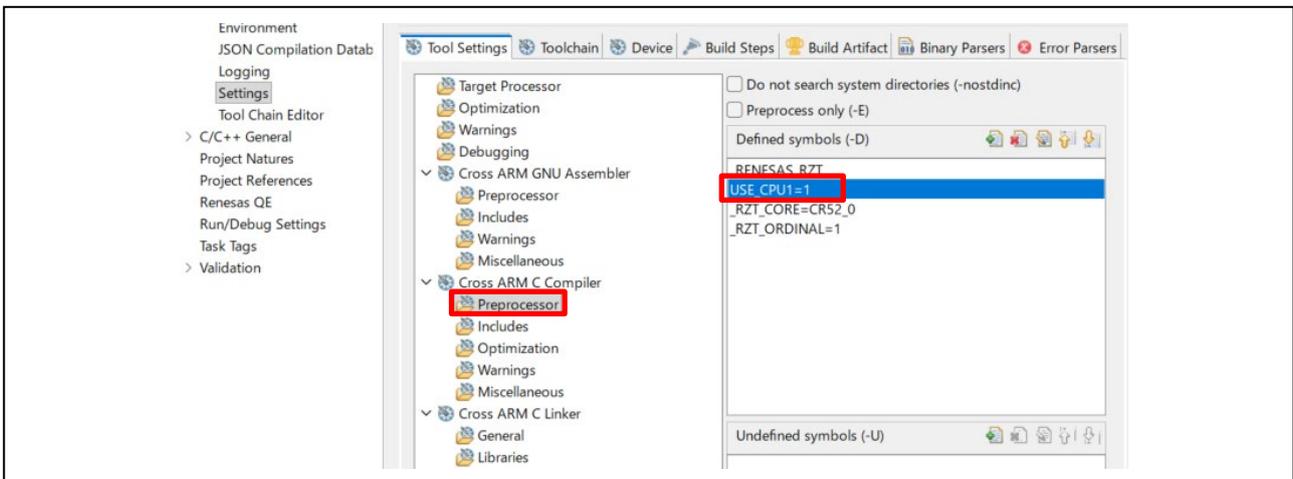
Figure 5-39 Enabling USE_CPU1 definition of CPU0 program (2/2)

4.   Rebuild and run the project

After rebuilding the application program and 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.
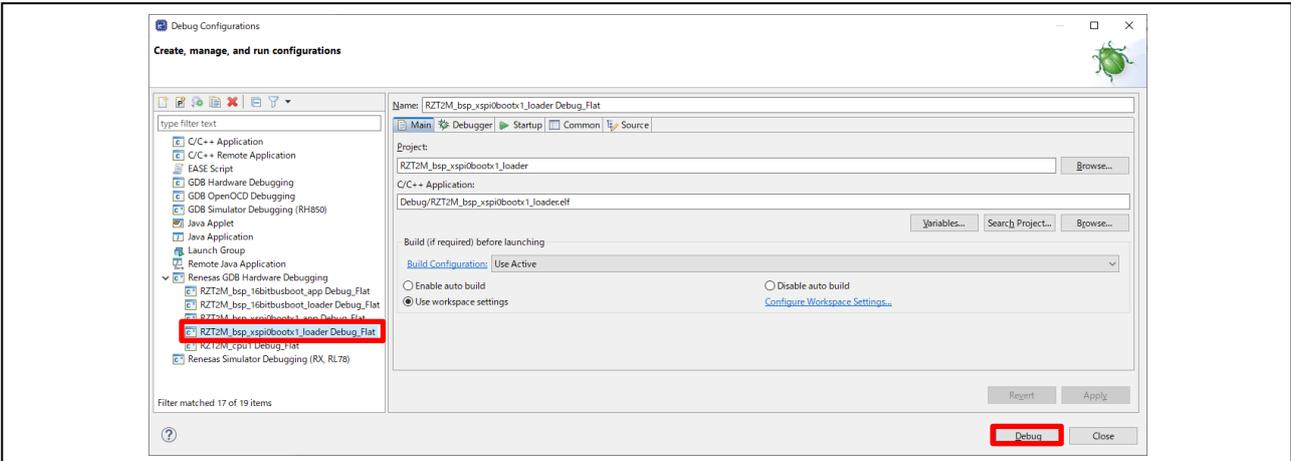
Figure 5-40 Start debugging the CPU0 loader program

When debugging the CPU1 program, start the debugging connection of the CPU1 project after making the debugging connection of the loader program.
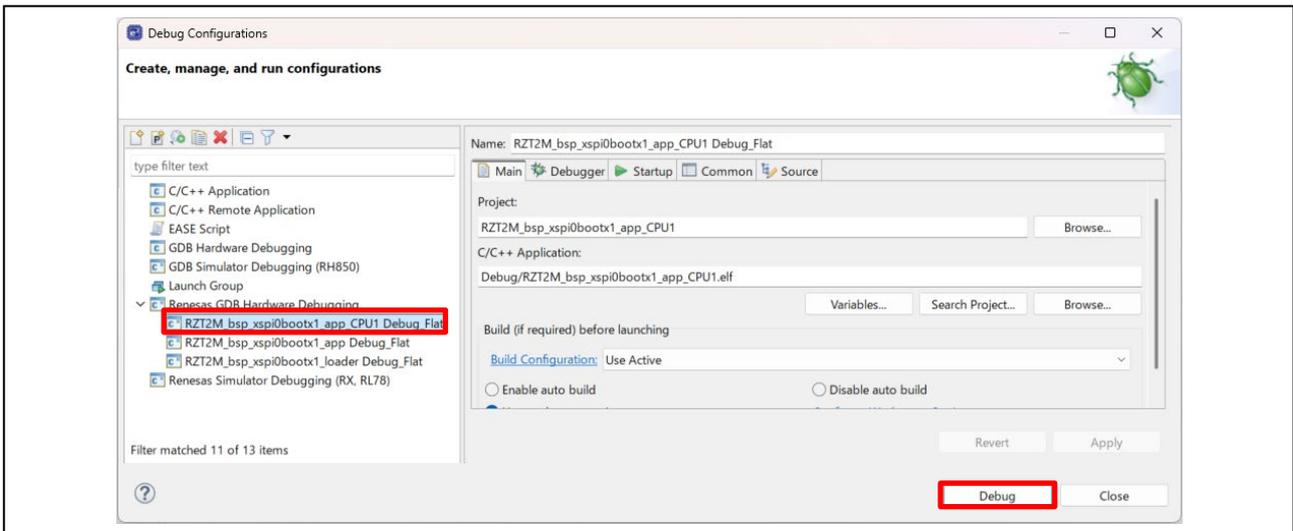
Figure 5-41 Start debugging the CPU1 program

When the following message is displayed, select "No".
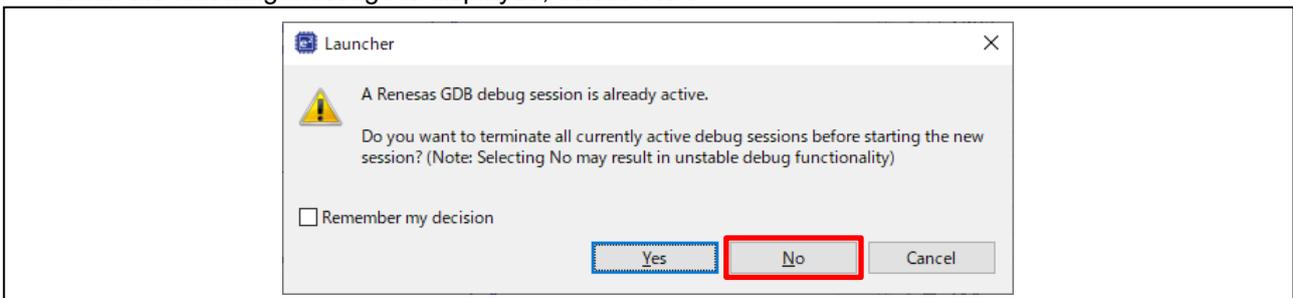
Figure 5-42 CPU0 loader program debug continuation confirmation screen

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.
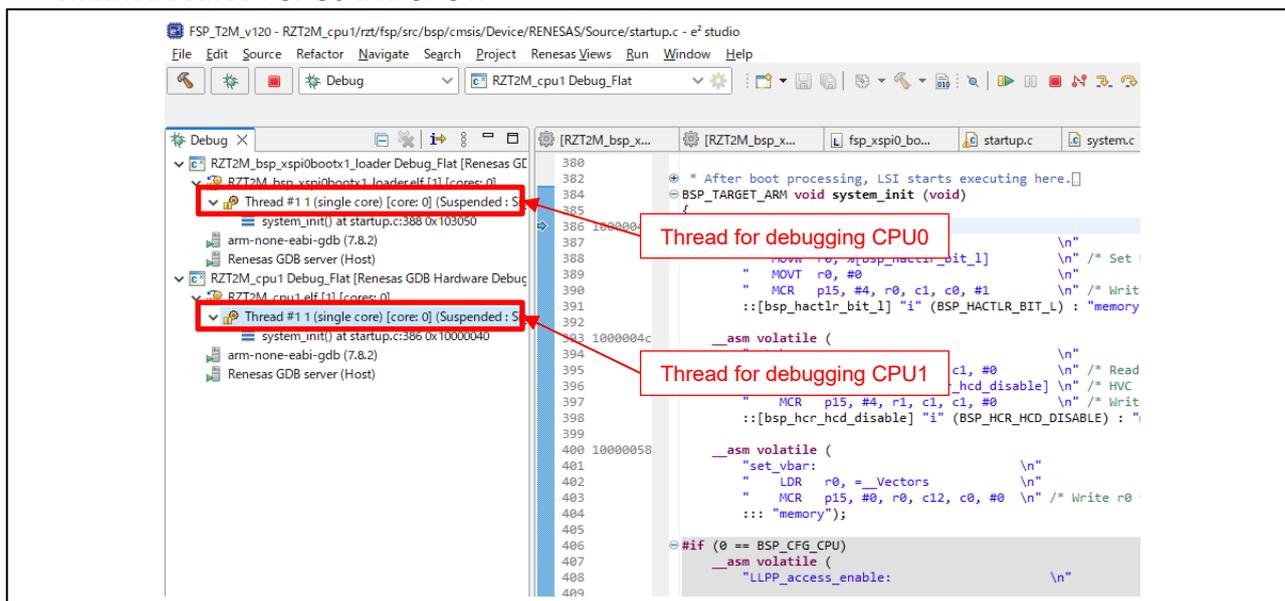
Figure 5-43 Debugging e^2 studio

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.

Supplementary Notes for the e² studio Environment

The following option settings change the path of the linker script file for the CPU1 project.

1. Right-click the CPU1 project node and open [Properties].
2. Open [C/C++ Build] > [Settings] > [Tool Settings] > [Cross ARM C Linker] > [General] and change the file path to "fsp_ram_execution_cpu1.ld".
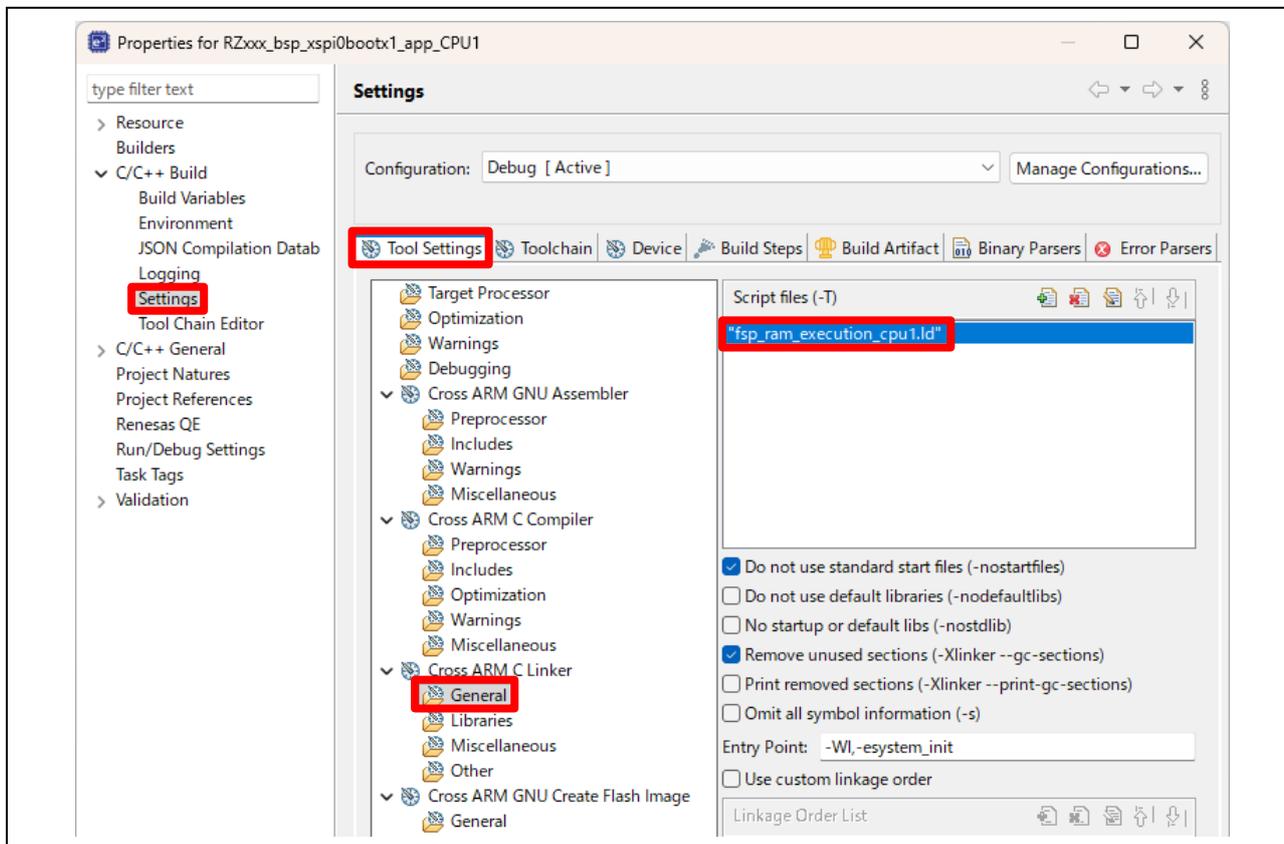
Figure 5-44 Setting the path for the linker script file in the CPU1 project

5.3.3　Supplementary Notes for the CPU1 program

The following FSP settings disable the C Runtime for the CPU1 project.
The C Runtime Initialization for the CPU1 project is done in the Loader project.

1. Open configuration.xml in the CPU1 project.
2. Change [BSP] > [C Runtime Initialization] to "Disabled".
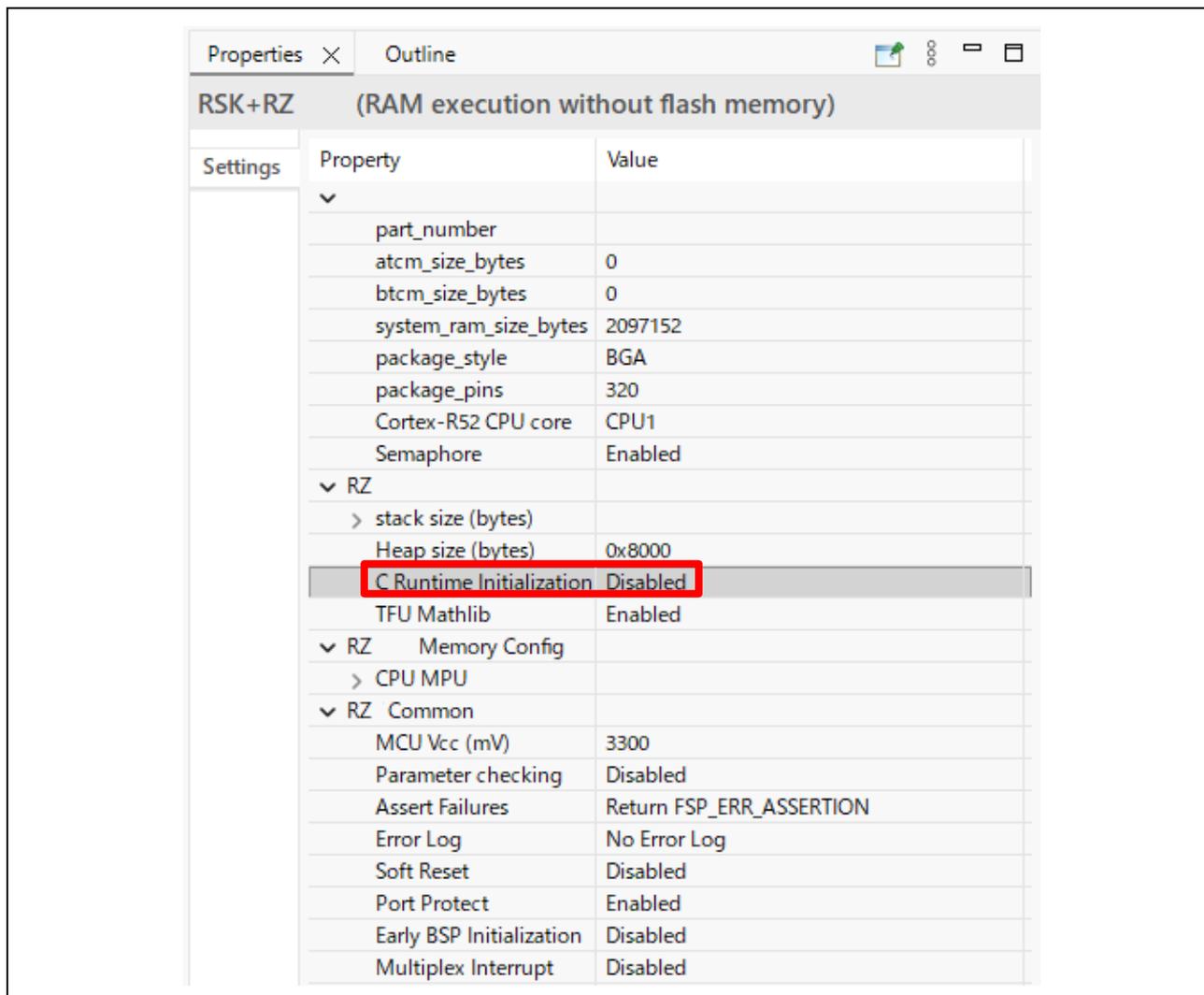
Figure 5-45 BSP settings for CPU1 program

6.　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.
2.10	May. 16, 2025	14, 30	Change DATA_RBLOCK to USER_DATA_RBLOCK Change DATA_WBLOCK to USER_DATA_WBLOCK Change DATA_ZBLOCK to USER_DATA_ZBLOCK
		31, 34	Removed instructions for placing sections in BTCM.
3.00	Dec. 17, 2025	-	FSP v3.00 support
		-	RZ/T2ME support
3.01	Mar. 3, 2026	31 ~ 34	Added supplementary notes for the sample program in the EWARM environment.
		37, 38	Added supplementary notes for the sample program in the e[2] studio environment.
		40, 41	Added supplementary notes for the application program and loader program.
		53 ~ 55	Added supplementary notes for the CPU1 program in the EWARM environment.
		61	Added supplementary notes for the CPU1 program in the e[2] studio environment.
		62	Added supplementary notes for the CPU1 program.

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/.