# RZ/T2L 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).
- 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/T2L Group

## 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/T2L Group (R9A07G074M04GBG) |
| Operating Frequency | CPU core0: 800MHz (Arm® Cortex®-R52) |
| Operating Voltage | 3.3V / 1.8V / 1.1V |
| Integrated Development Environment | • Embedded Workbench® for Arm Version 9.60.3 from IAR systems<br>• $e^2$ studio 2025-04.1 (25.4.1) (R20250508-1327) from Renesas |
| Operating mode | • xSPI0 boot mode (x1 serial flash)<br>• xSPI1 boot mode (x1 serial flash) |
| Board | Renesas Starter Kit+ for RZ/T2L |
| Flexible Software Package (FSP) | Version 3.0.0 (RZ/T2 FSP) |

## 1.2 File Structure

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

```
RZT2L_loader_application
├─r01an7465jj0301-rzt2l-separating-loader-and-application.pdf
├─r01an7465ej0301-rzt2l-separating-loader-and-application.pdf
├─iccarm: for EWARM
│   ├─xspi0bootx1: sample program for SPI0 flash
│   │   └─Loader_application_projects.zip
│   │       ├─RZT2L_bsp_xspi0bootx1_app: project for application program
│   │       ├─RZT2L_bsp_xspi0bootx1_loader: project for loader program
│   │       └─RZT2L_bsp_xspi0bootx1_separating_loader.eww: EWARM workspace
│   └─xspi1bootx1: sample program for SPI1 flash
│       └─Loader_application_projects.zip
│           ├─RZT2L_bsp_xspi1bootx1_app: project for application program
│           ├─RZT2L_bsp_xspi1bootx1_loader: project for loader program
│           └─RZT2L_bsp_xspi1bootx1_separating_loader.eww: EWARM workspace
└─gcc : for e2 studio
    ├─xspi0bootx1: sample program for SPI0 flash
    │   └─Loader_application_projects.zip
    │       ├─RZT2L_bsp_xspi0bootx1_app: project for application program
    │       └─RZT2L_bsp_xspi0bootx1_loader: project for loader program
    └─xspi1bootx1: sample program for SPI1 flash
        └─Loader_application_projects.zip
            ├─RZT2L_bsp_xspi1bootx1_app: project for application program
            └─RZT2L_bsp_xspi1bootx1_loader: project for loader 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 and one project for the application program.

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

## 1.3 Switch and Jumper Settings

The switch and jumper settings required to run the sample program are shown below. For details on each setting, see the Renesas Starter Kit+ for RZ/T2L User's Manual.

Table 1-2 Switch settings

| Project | SW4-1 | SW4-2 | SW4-3 | SW4-4 | SW4-5 | SW4-6 | SW4-7 |
|---|---|---|---|---|---|---|---|
| xSPI0 boot mode | ON | ON | ON | OFF | ON | ON | ON |
| xSPI1 boot mode | ON | ON | OFF | OFF | ON | ON | - |

| Project | SW7-4 | SW7-5 | SW7-6 | SW7-7 | SW7-8 | SW7-9 | SW7-10 |
|---|---|---|---|---|---|---|---|
| xSPI0 boot mode | OFF | OFF | ON | OFF | OFF | OFF | ON |
| xSPI1 boot mode | OFF | OFF | ON | OFF | OFF | OFF | ON |

| Project | SW8-9 | SW8-10 |
|---|---|---|
| xSPI0 boot mode | - | OFF |
| xSPI1 boot mode | ON | OFF |

Table 1-3 Jumper settings

| Project | CN18 | CN23 | CN32 | CN33 |
|---|---|---|---|---|
| xSPI0 boot mode | Short | Short | Short 2-3 | - |
| xSPI1 boot mode | Short | Short | - | Short 1-2 |

# 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/T2L 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_CKP | Output | Clock output |
| XSPI0_CS0# | Output | Device selection signal output to Octa flash memory attached to external address space xSPI0 |
| XSPI0_RESET0# | Output | Master reset status output |
| XSPI0_DS | Output | Read Data Strobe / Write Data Mask |
| XSPI0_ECS0# | Output | Error Correction Status for slave0 |
| XSPI0_IO0 ~ XSPI0_IO7 | Input/Output | Data input / output |
| XSPI1_CKP | Output | Clock output |
| XSPI1_CS0# | Output | Device selection signal output to Quad flash memory attached to external address space xSPI1 |
| XSPI1_IO0 ~ XSPI1_IO3 | Input/Output | Data input / output |
| MD0 | Input | Operating mode selection:<br>• MD0 = “L”, MD1 = “L”, MD2 = “L” (xSPI0 boot mode)<br>• MD0 = “L”, MD1 = “L”, MD2 = “H” (xSPI1 boot mode) |
| MD1 | Input | |
| MD2 | Input | |
| P21_3 | Output | Lighting LED0 on and off |
| P17_6 | Output | Lighting LED1 on and off |
| P20_3 | Output | Lighting LED2 on and off |
| P18_1 | Output | Lighting LED3 on and off |
| P21_6 | Output | Lighting LED4 on and off |
| P20_4 | Output | Lighting LED5 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.

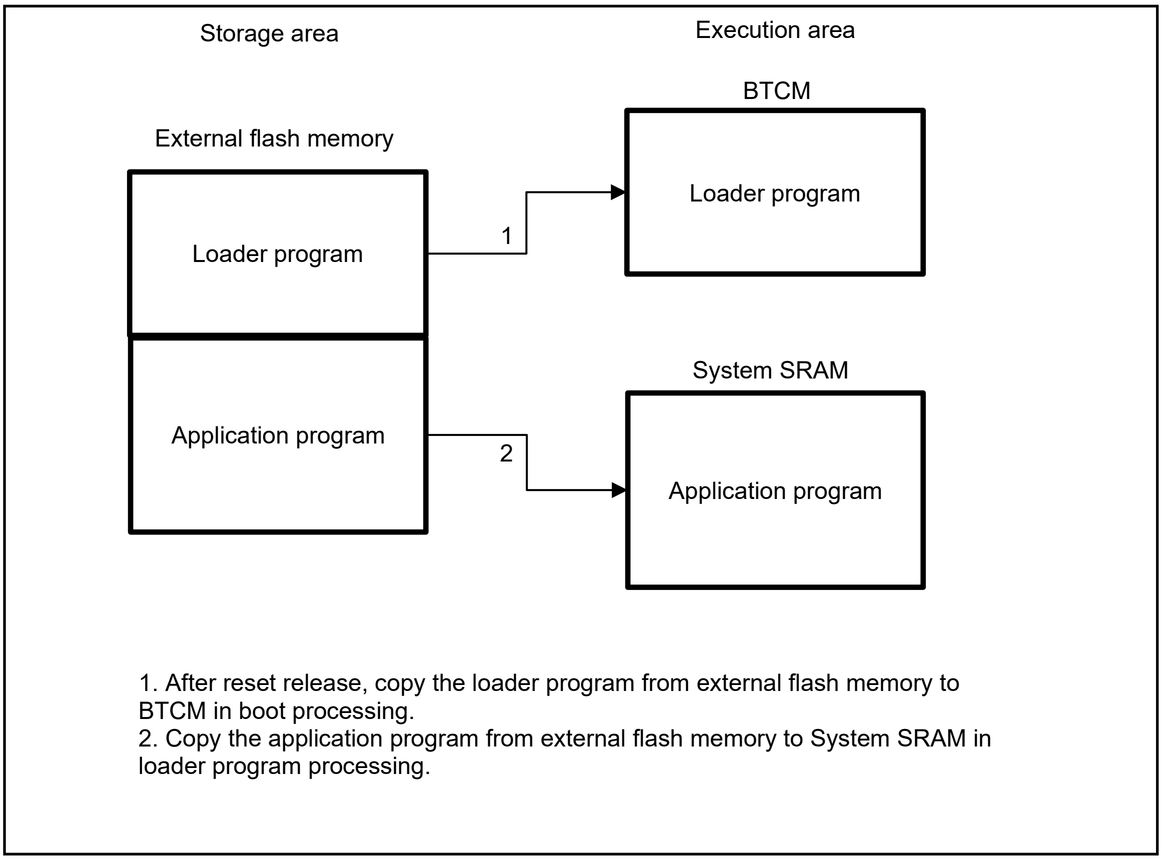


Figure 3-1 Operation overview

### 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 LED5 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.

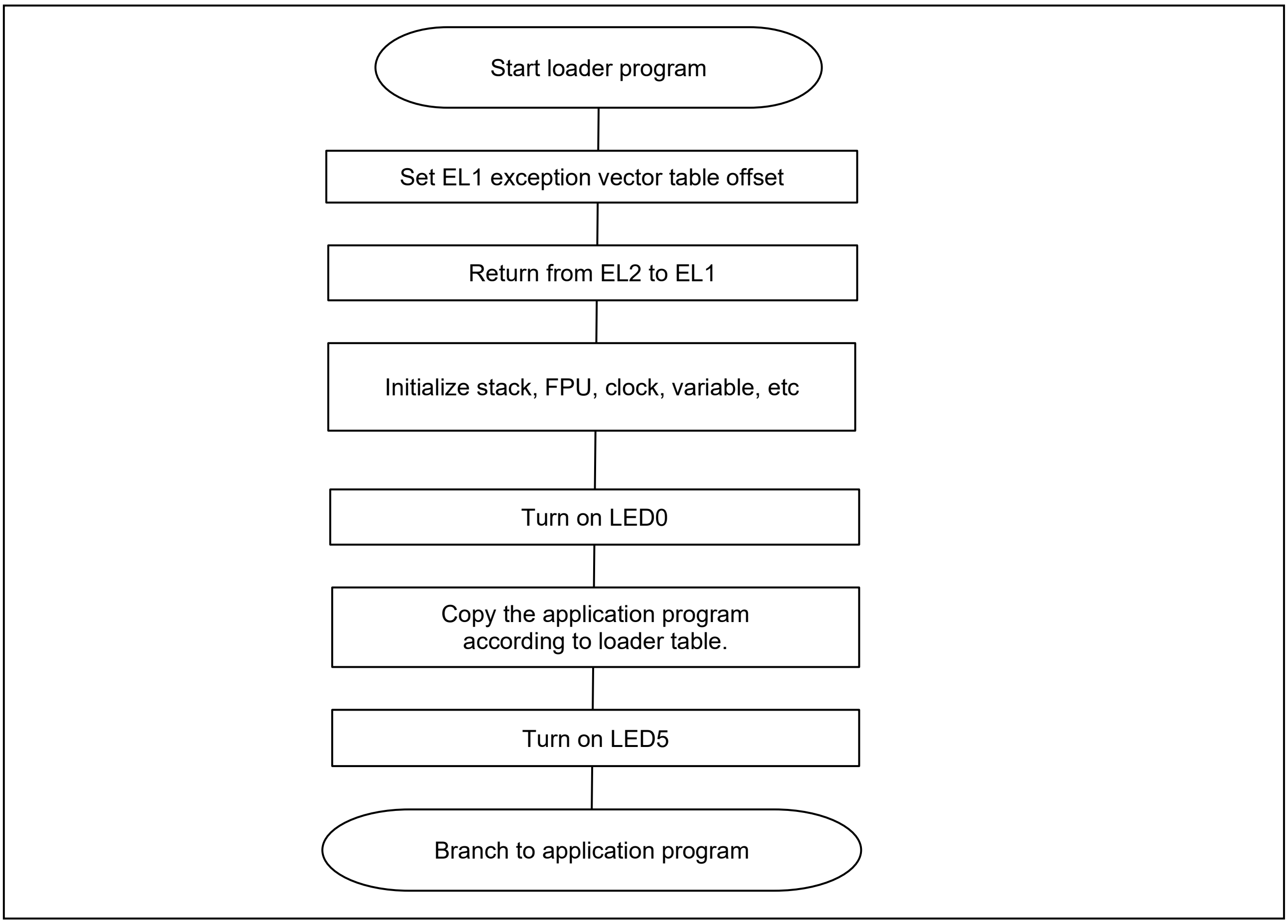


Figure 3-2 Operation overview of loader program

### 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 to LED5 blink.

Figure 3.3 shows operation overview of application program.

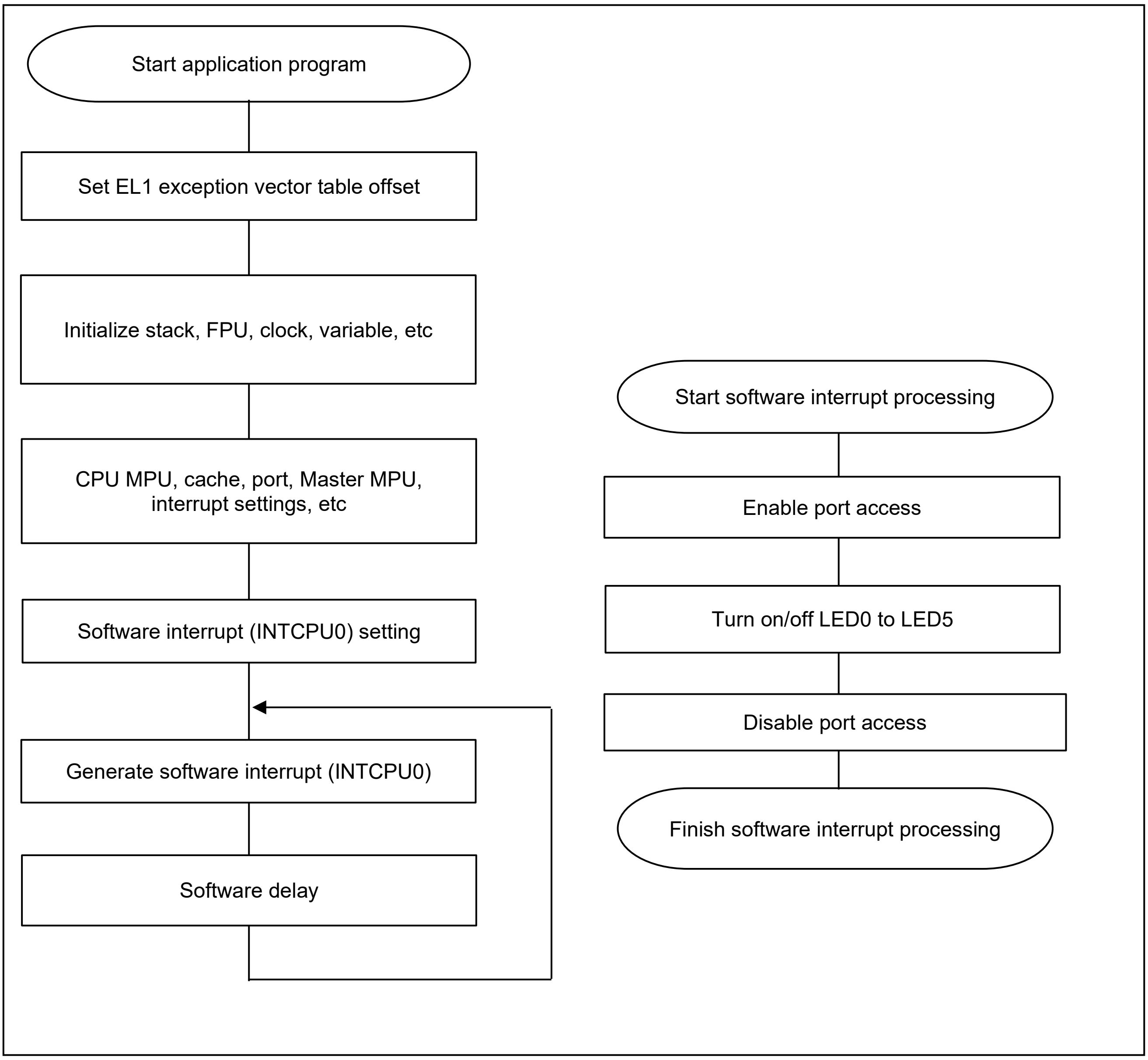


Figure 3-3 Operation overview of application program

## 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.<br>If this flag is disabled, copy processing will not be performed even if other parameters are set.<br>0: Disable<br>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_2044 | 0x1 |
| 1[*1] | 0xFFFF_FFFF | 0xFFFF_FFFF | 0xFFFF_FFFF | 0x0 |
| 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.

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

| Table | Src | Dst | Size | Enable flag |
|---|---|---|---|---|
| 0 | 0x6810_0000 | 0x1008_0000 | 0x0000_1FE4 | 0x1 |
| 1 | 0xFFFF_FFFF | 0xFFFF_FFFF | 0xFFFF_FFFF | 0x0 |
| 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.

## 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) |

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

| Flash memory address | Contents | Load destination address |
|---|---|---|
| 0x6800_0000 | Parameters for the loader | - |
| 0x6800_004C | Loader program | 0x0010_2000 (BTCM) |
| 0x6808_0000 | Loader table | - |
| 0x6810_0000 | Application program | 0x1008_0000 (System SRAM) |

### 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 |

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

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 and xSPI1 boot, serial flash memory is storing area.

#### 3.3.2.2 e[2] 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(e[2] 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<br>.intvec<br>.reset_handler<br>.loader_text | Code area (for execution) | BTCM |
| .data<br>.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 |

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

Table 3-9 Sections used by application program(e[2] studio)

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

Note 1. In xSPI0 and xSPI1 boot, serial 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 3-10 CPU MPU Settings

| Contents | Address | Memory type |
|---|---|---|
| System SRAM | 0x1000_0000<br>to<br>0x100F_FFFF | Area 2<br>Normal, cache enabled, non-shared |
| System SRAM (mirror area) | 0x3000_0000<br>to<br>0x300F_FFFF | Area 3<br>Normal, cache disabled, shared |
| Extended address space (mirror area)<br>xSPI0, xSPI1<br>CS0, CS2, CS3, CS5 | 0x4000_0000<br>to<br>0x5FFF_FFFF | Area 4, 5, 6, 7, 8, 9<br>Normal, cache disabled, shared |
| Extended address space<br>xSPI0, xSPI1<br>CS0, CS2, CS3, CS5 | 0x6000_0000<br>to<br>0x7FFF_FFFF | Area 10, 11, 12, 13, 14, 15<br>Normal, cache enabled, non-shared |
| Non-safety peripheral modules | 0x8000_0000<br>to<br>0x80FF_FFFF | Area 16<br>Device (nGnRE) , instruction fetch disabled |
| Safety peripheral modules | 0x8100_0000<br>to<br>0x81FF_FFFF | Area 17<br>Device (nGnRE) , instruction fetch disabled |

### 3.3.4 Exception Processing Vector Table

Exception level 1 of RZ/T2L 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 3-11 Exception Processing Vector Table

| Exception | Handler Address*1 | Remark*2 |
|---|---|---|
| RESET | Offset | Branches to startup program |
| Undefined instruction | Offset + 0x0000_00004 | Branches Default_Handler |
| SVC | Offset + 0x0000_00008 | Branches Default_Handler |
| Prefetch abort | Offset + 0x0000_0000C | Branches Default_Handler |
| Data abort | Offset + 0x0000_00010 | Branches Default_Handler |
| Reserved | Offset + 0x0000_00014 | Branches Default_Handler |
| IRQ | Offset + 0x0000_00018 | Branches IRQ_Handler (Used for interrupt) |
| FIQ | Offset + 0x0000_0001C | Branches Default_Handler |

Note 1. The offset is defined as following.
Loader program : 0x0010_2000
Application program : 0x1008_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 LED5 after copy processing is complete.<br>• Application program: Then Blinks LED0 to LED5 with software interrupt (INTCPU0). |
| Arguments | None |
| Return value | None |
| Remarks | None |

### 3.4.4 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.<br>• uintptr_t *dst: Copy destination address.<br>• uintptr_t bytesize: Copy data size. |
| Return value | None |
| Remarks | None |

## 3.5 Flowchart

### 3.5.1 Loader Program

#### 3.5.1.1 system_init

Figure 3.4 shows flowchart of system_init in the loader program.

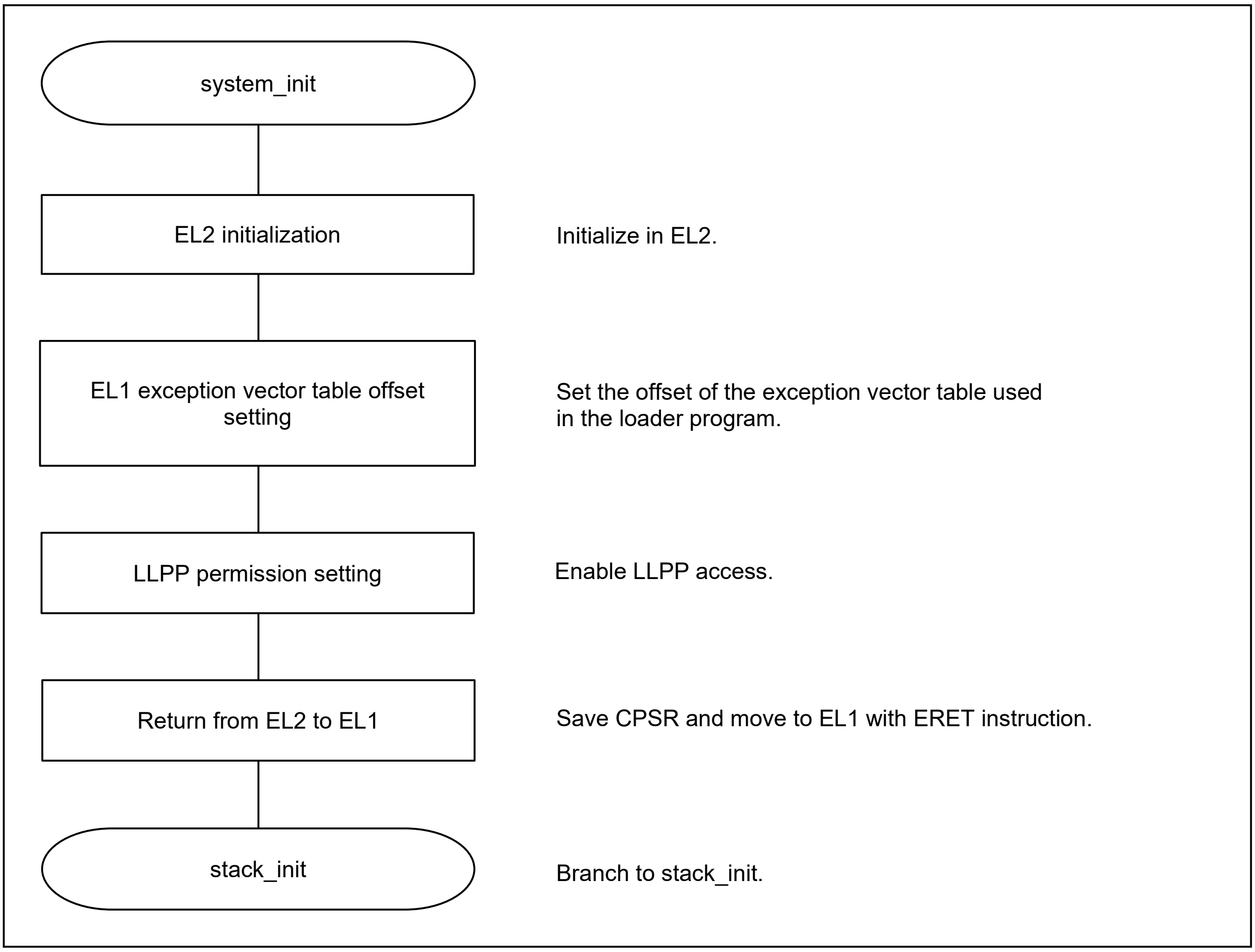


Figure 3-4 system_init processing (loader program)

#### 3.5.1.2 stack_init

Figure 3.5 shows flowchart of stack_init in the loader program.

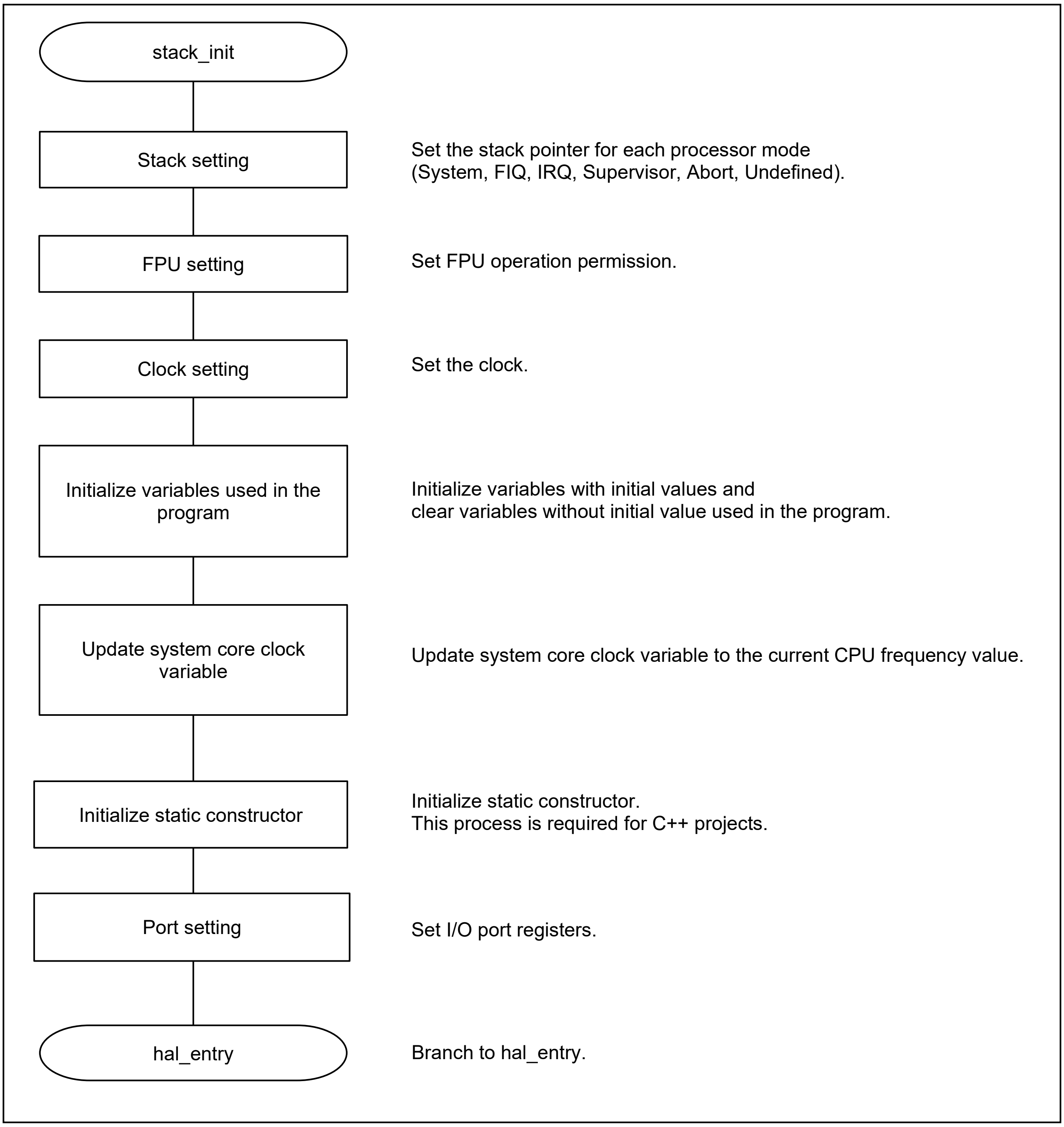


Figure 3-5 stack_init processing (loader program)

#### 3.5.1.3 hal_entry

Figure 3.6 shows flowchart of hal_entry in the loader program.

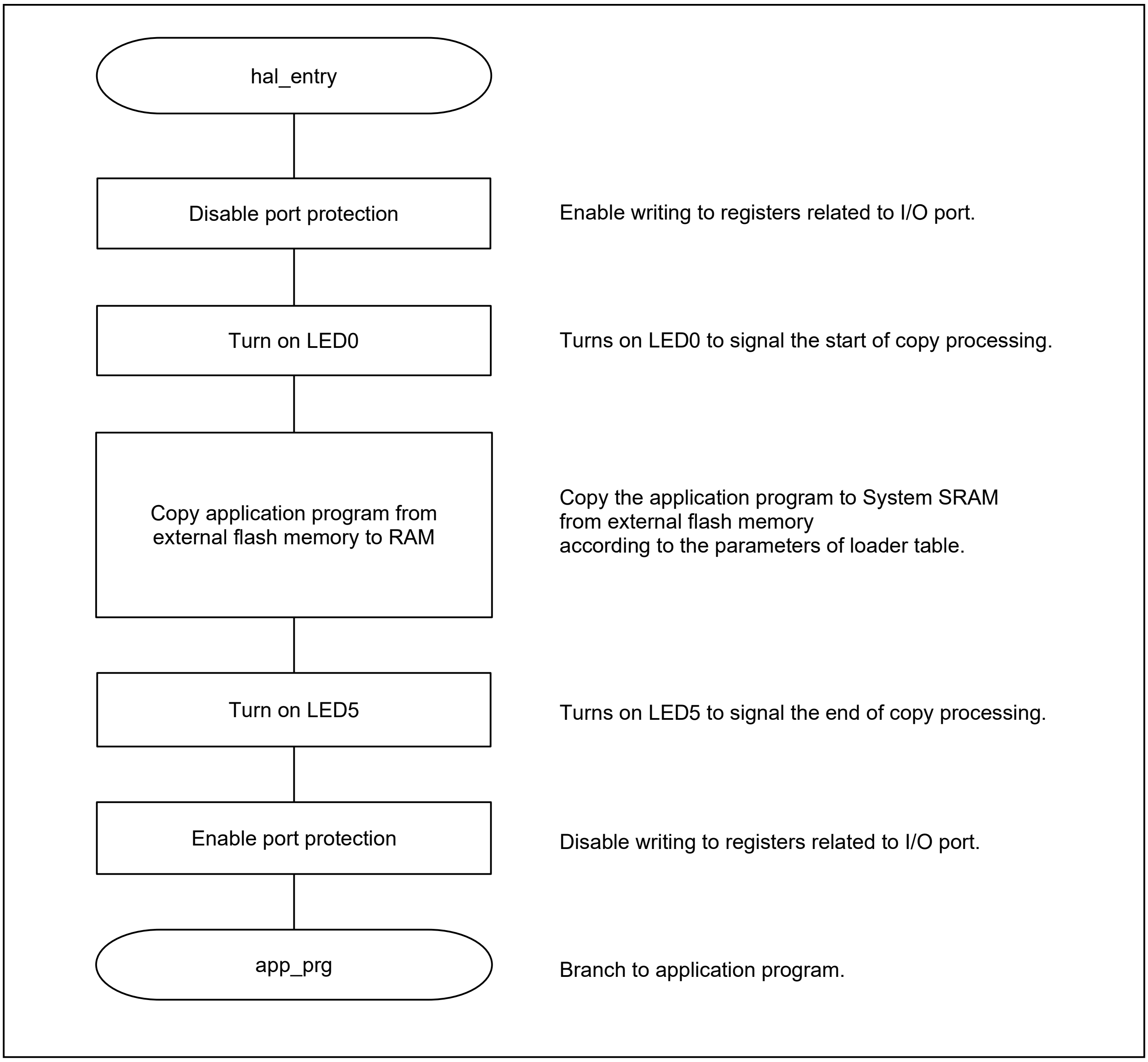


Figure 3-6 hal_entry processing (loader program)

### 3.5.2 Application Program

#### 3.5.2.1 system_init

Figure 3.7 shows flowchart of system_init in the application program.

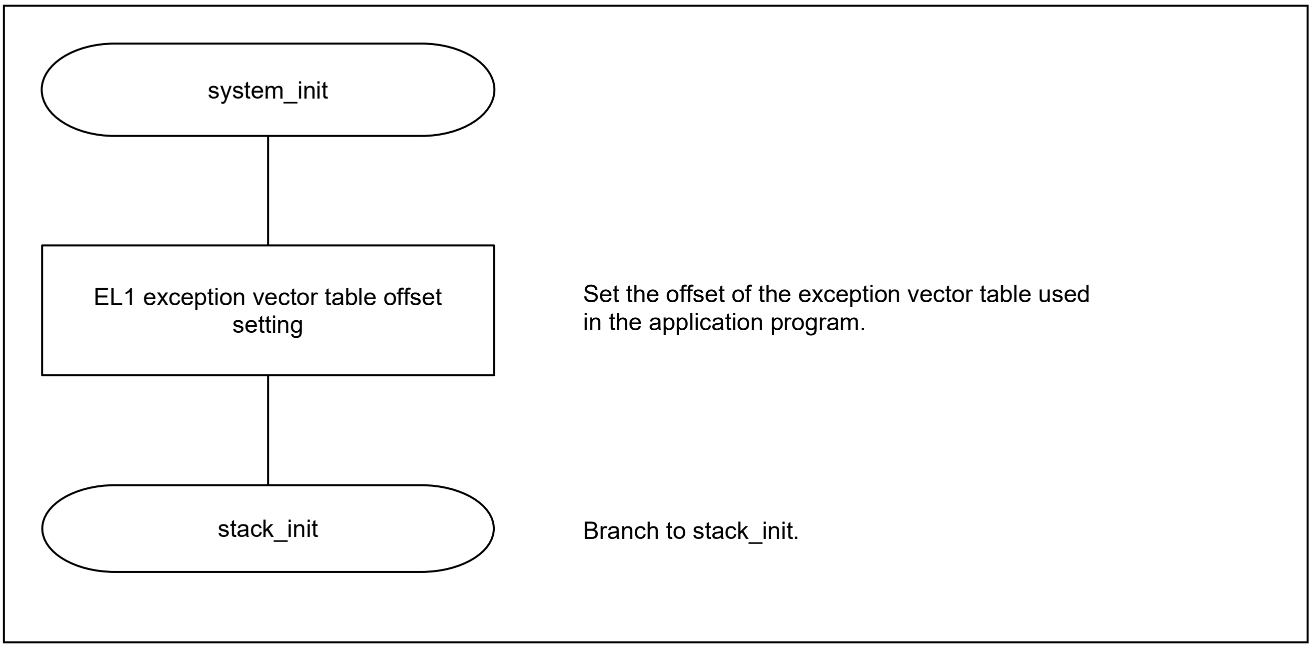


Figure 3-7 system_init processing (application program)

#### 3.5.2.2 stack_init

Figure 3.8, Figure 3.9 shows flowchart of stack_init in the application program.

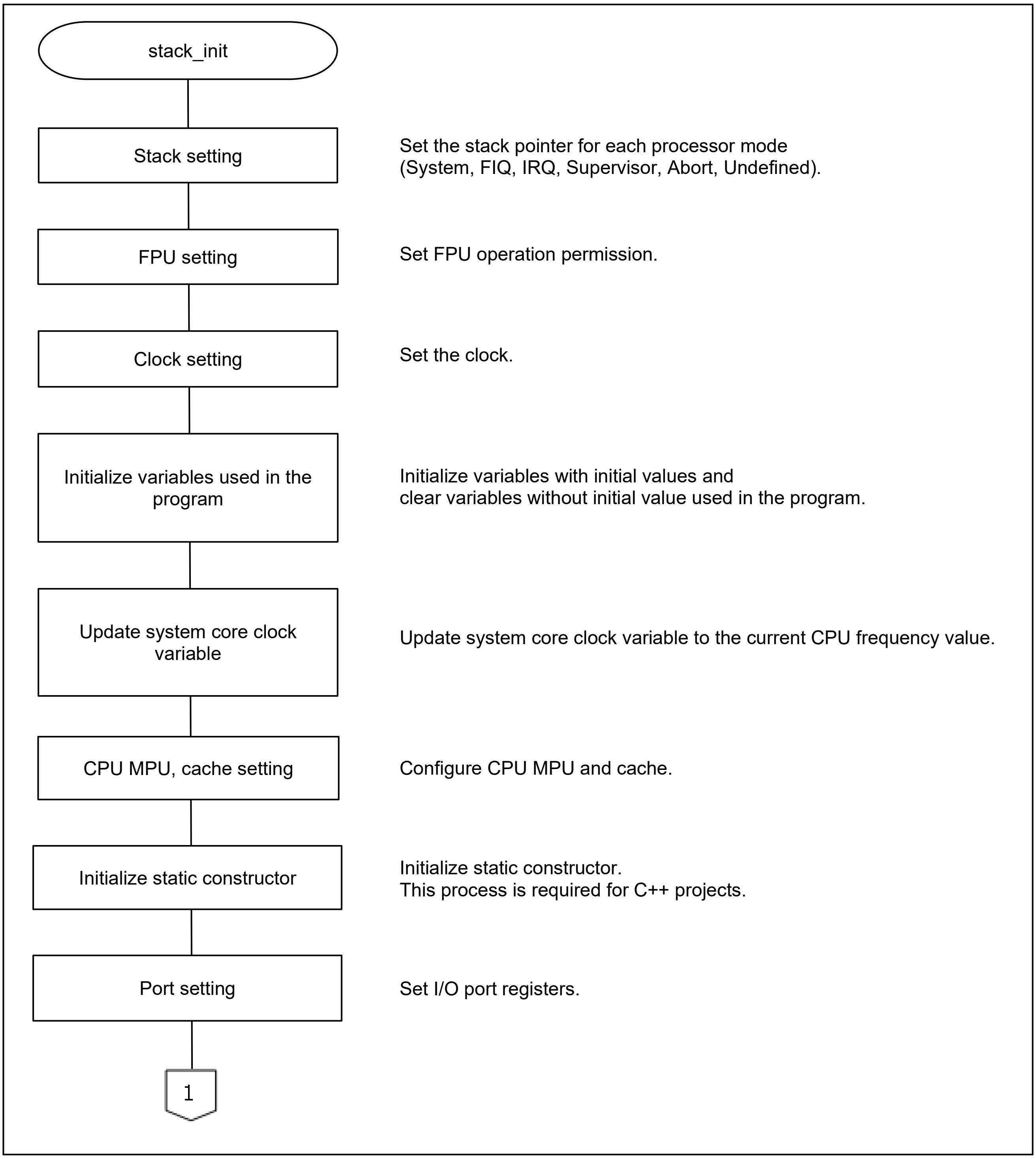


Figure 3-8 stack_init processing (1/2)(application program)

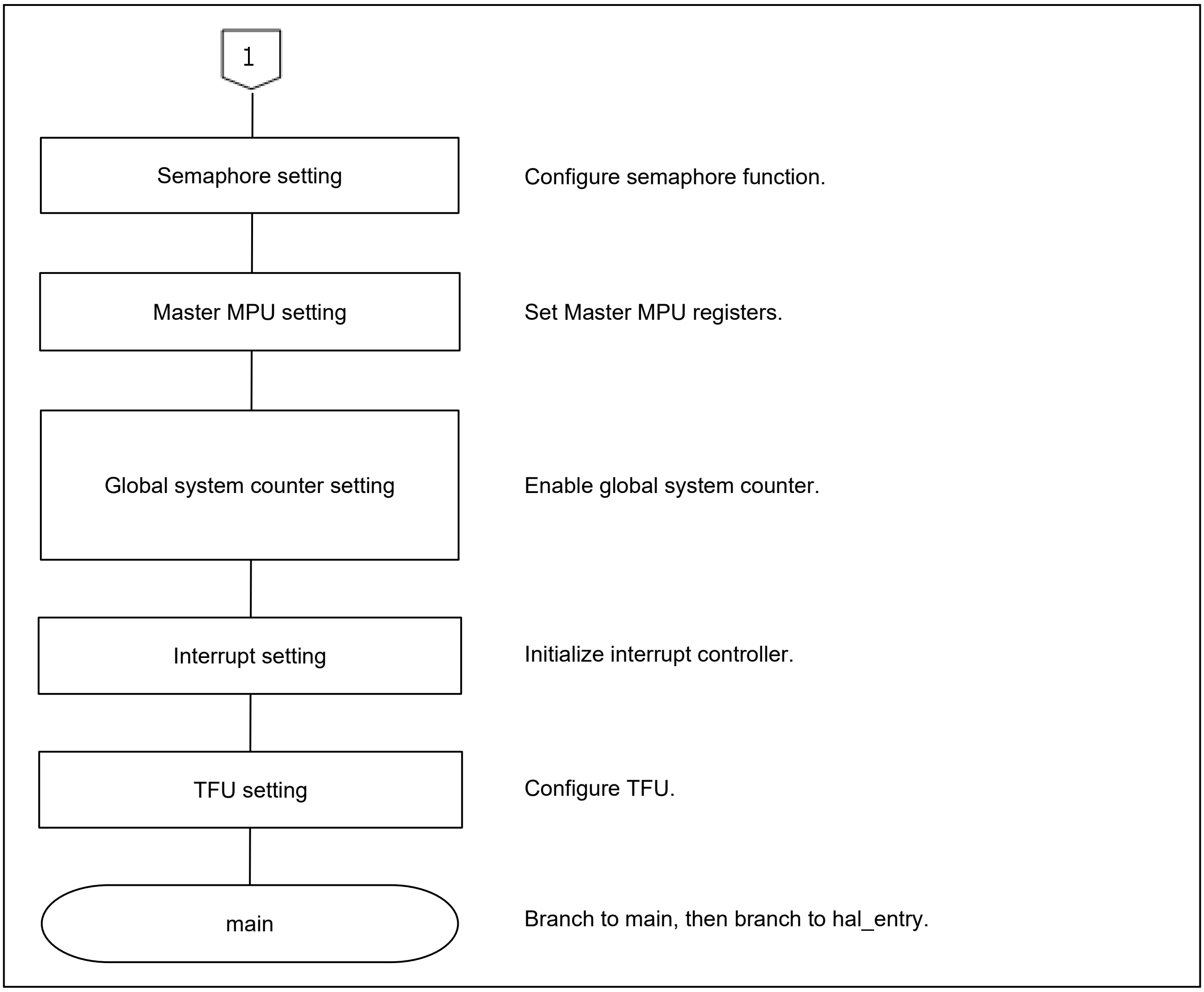


Figure 3-9 stack_init processing (2/2)(application program)

### 3.5.2.3 hal_entry

Figure 3.10 shows flowchart of hal_entry in the application program.

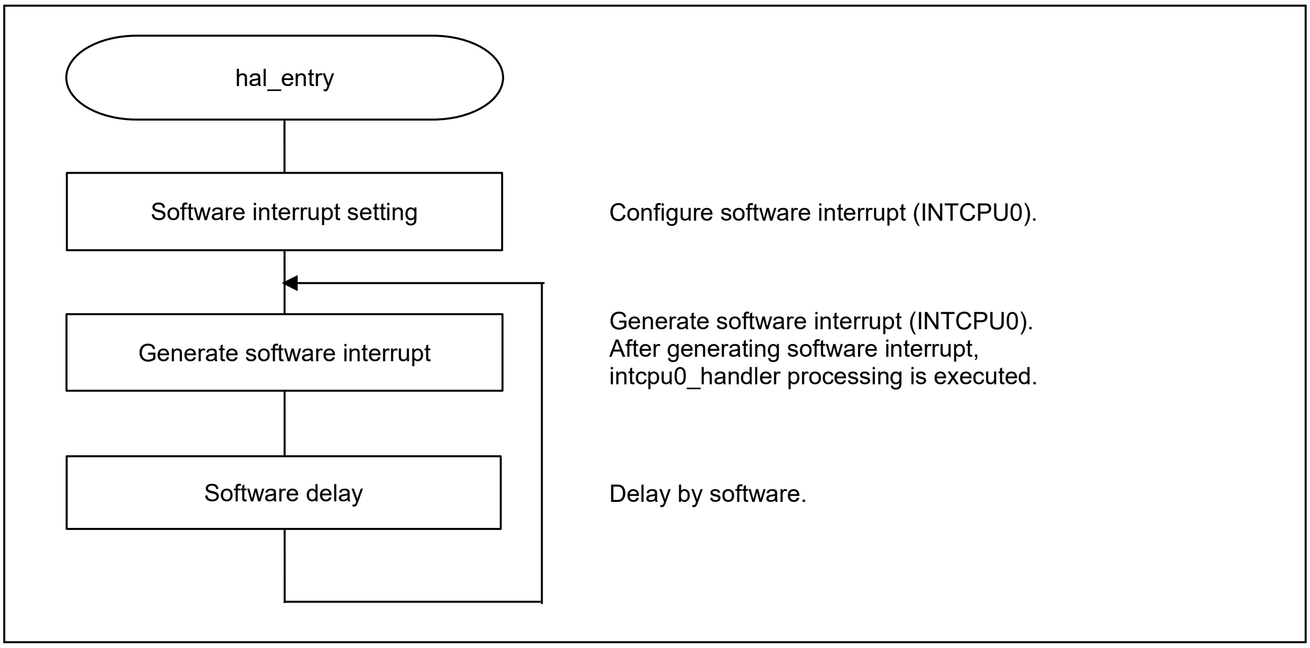


Figure 3-10 hal_entry processing (application program)

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

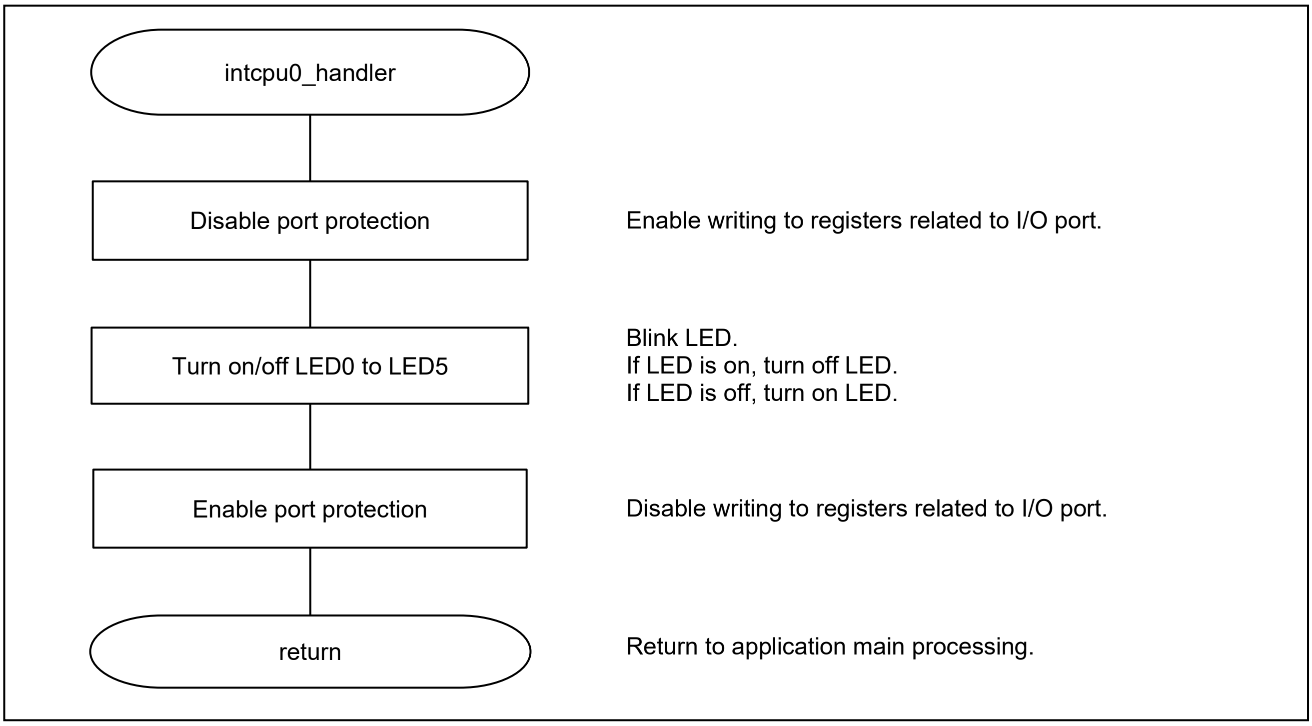


Figure 3-11 interrupt processing (application program)

## 4. Related Documents

- User's Manual: Hardware
  RZ/T2L Group User's Manual: Hardware
  Download the latest version from the Renesas Electronics website.

  Renesas Starter Kit+ for RZ/T2L
  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 "RZT2L_bsp_xspi0bootx1_separating_loader.eww" with following procedure.
   "[File] -> [Open Workspace] -> select Loader_application_projects\RZT2L_bsp_xspi0bootx1_separating_loader.eww"
2. Select "RZT2L_bsp_xspi0boot1_app" project in Workspace box as Figure 5-1.

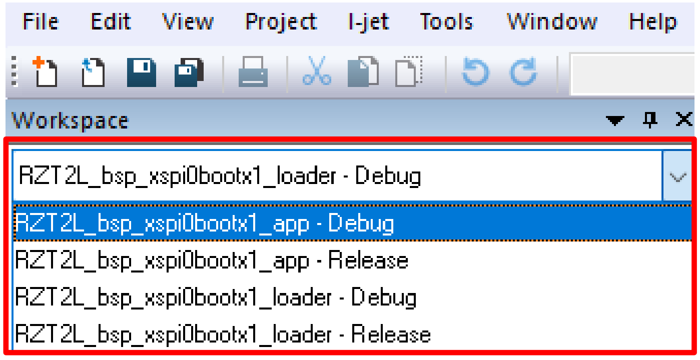


**Figure 5-1 Project selection**

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

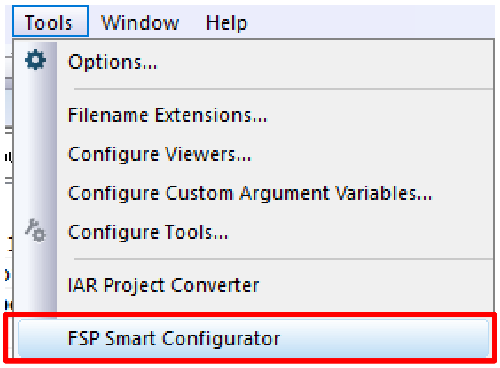


**Figure 5-2 Start FSP Configurator**

※ 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.

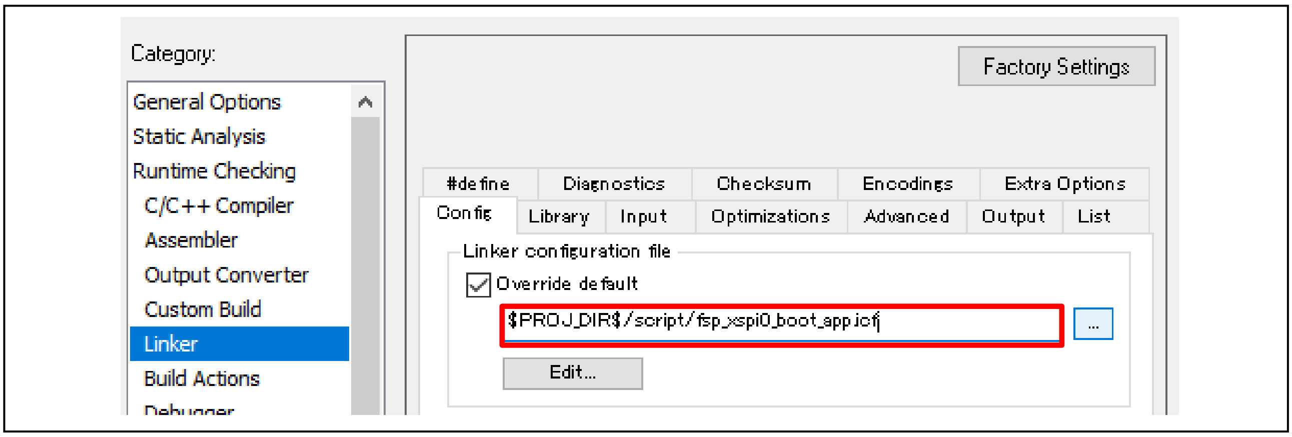


**Figure 5-3 Linker script setting (application program)**

5. Push "[Project] -> [Rebuild All]" to build the application program.
6. Then, select "RZT2L_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.

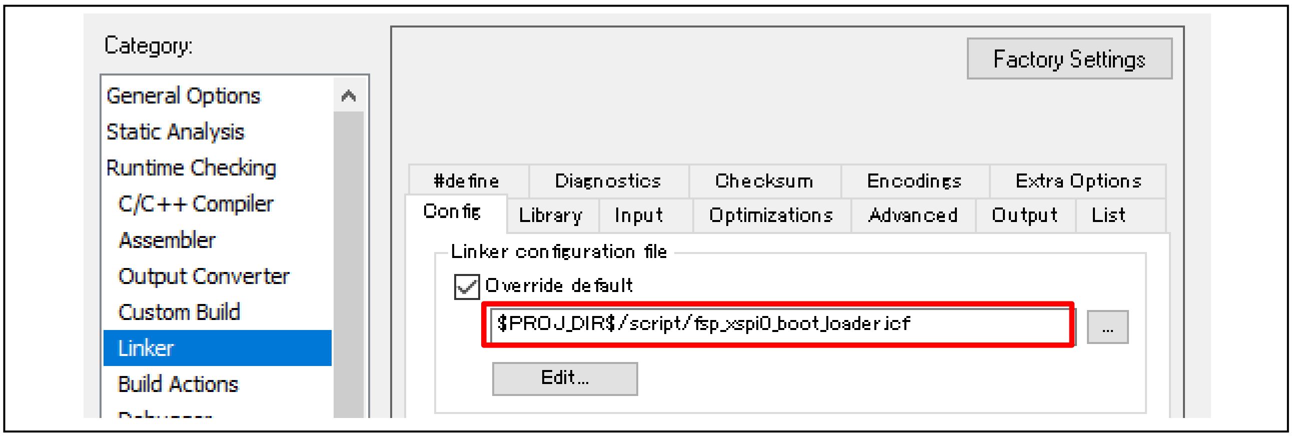


**Figure 5-4 Linker script setting (loader program)**

9. Push "[Project] -> [Rebuild All]" to build the loader program.
10. Make sure that your PC and RZ/T2L 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 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:
$PROJ_DIR$\..\RZT2L_bsp_xspi0bootx1_app\Debug\Exe\RZT2L_bsp_xspi0bootx1_app.out

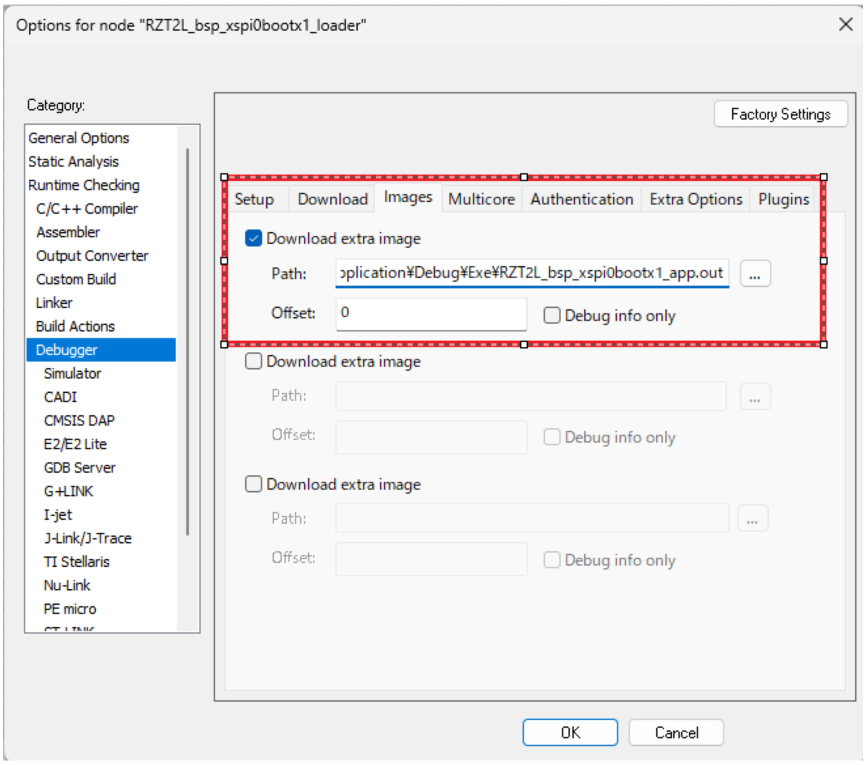


**Figure 5-5 EWARM option setting (1)**

**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:
$PROJ_DIR$\..\RZT2L_bsp_xspi0bootx1_app\Debug\Exe\RZT2L_bsp_xspi0bootx1_app.bin
Symbol:
APPLICATION_PRG_SECTION
Section:
APPLICATION_PRG_SECTION
Align:
4

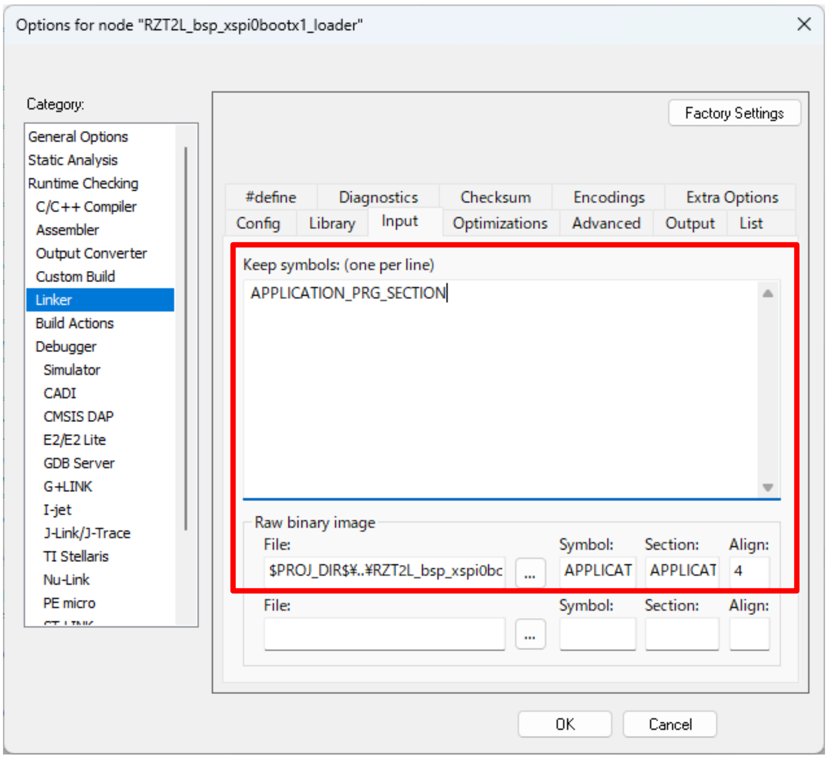


**Figure 5-6 EWARM option setting (2)**

**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".

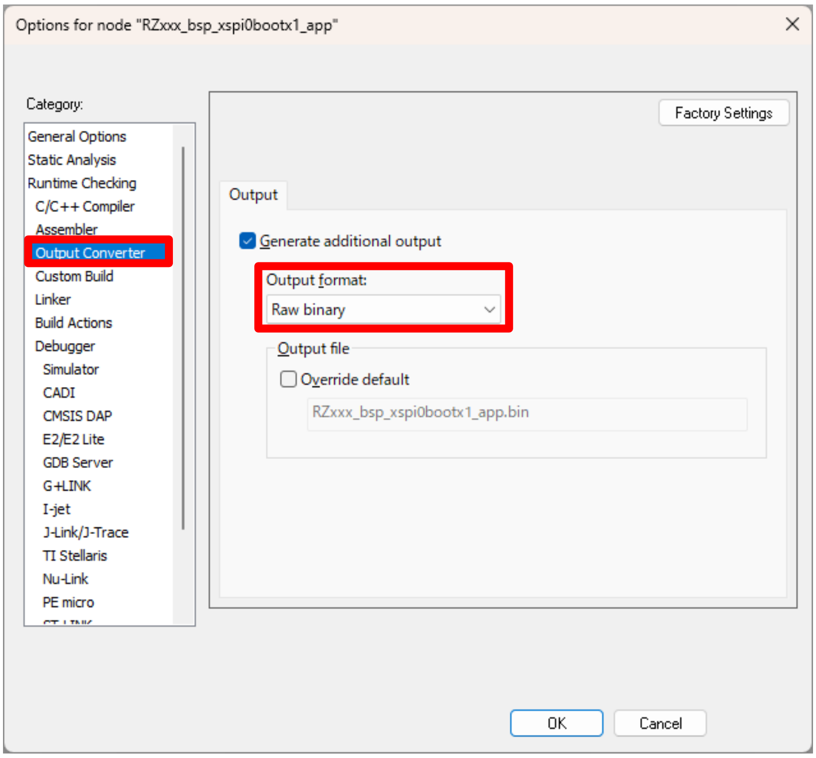


**Figure 5-7 Configuring the output format for the Application project**

**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].

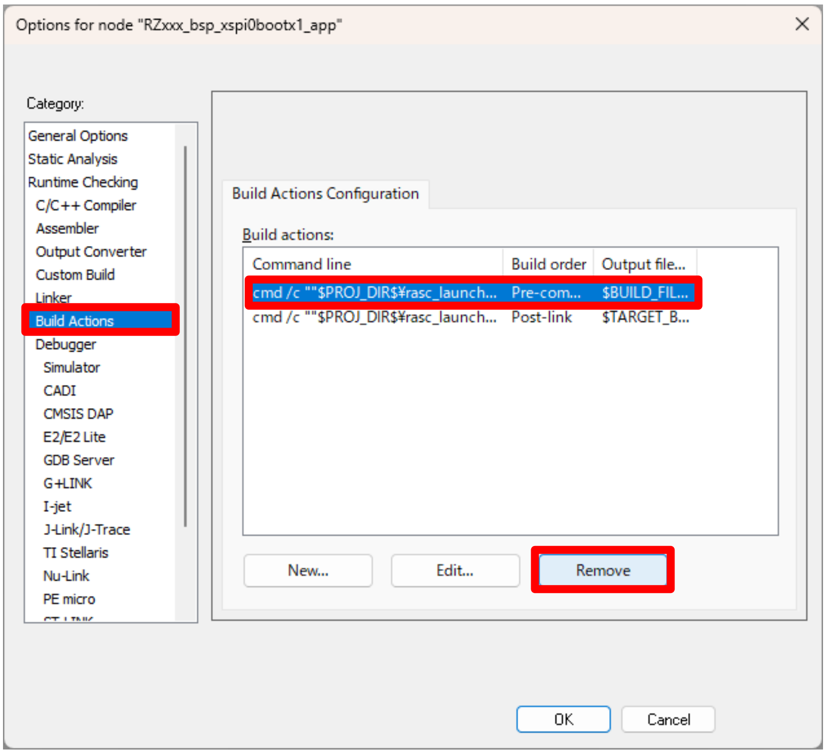


**Figure 5-8 Remove Build Action**

**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".

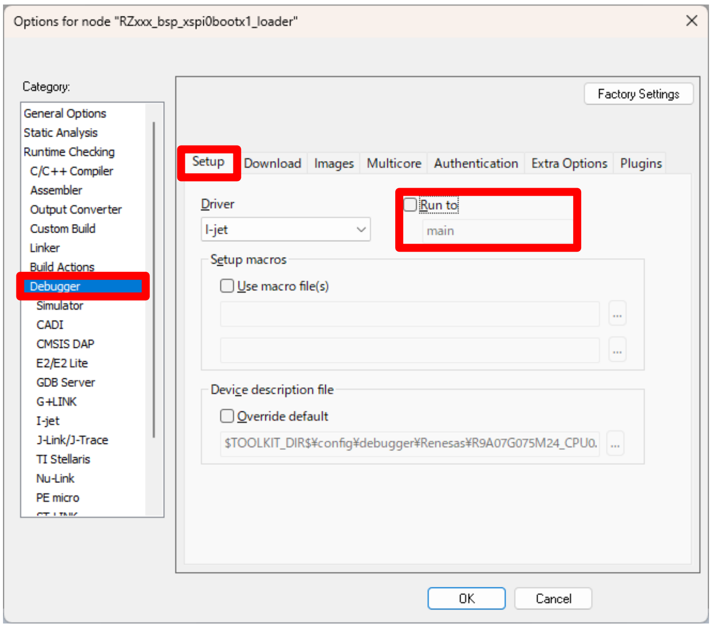


**Figure 5-9 Loader project debug settings (1)**

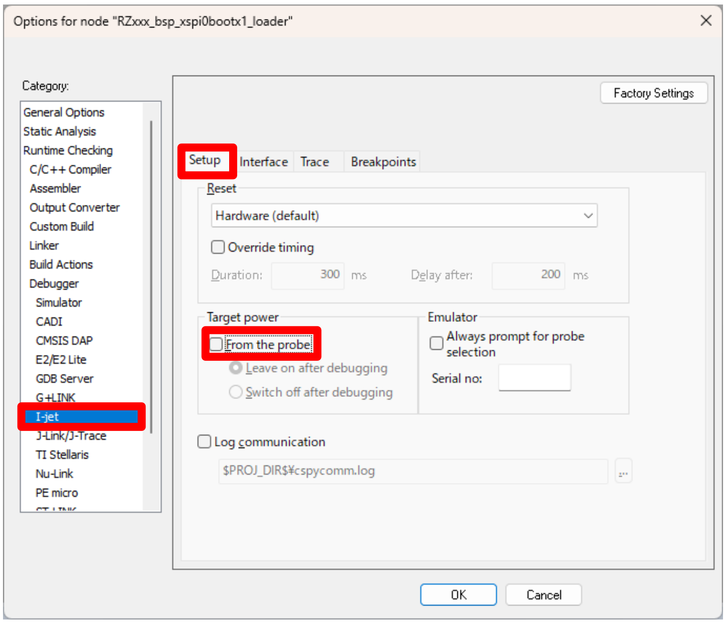


**Figure 5-10 Loader project debug settings (2)**

[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".

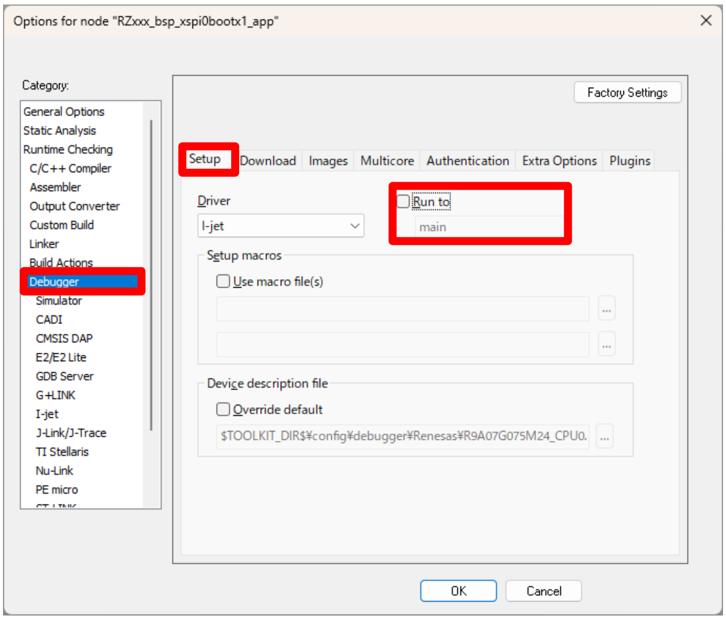


**Figure 5-11 Application project debug settings (1)**

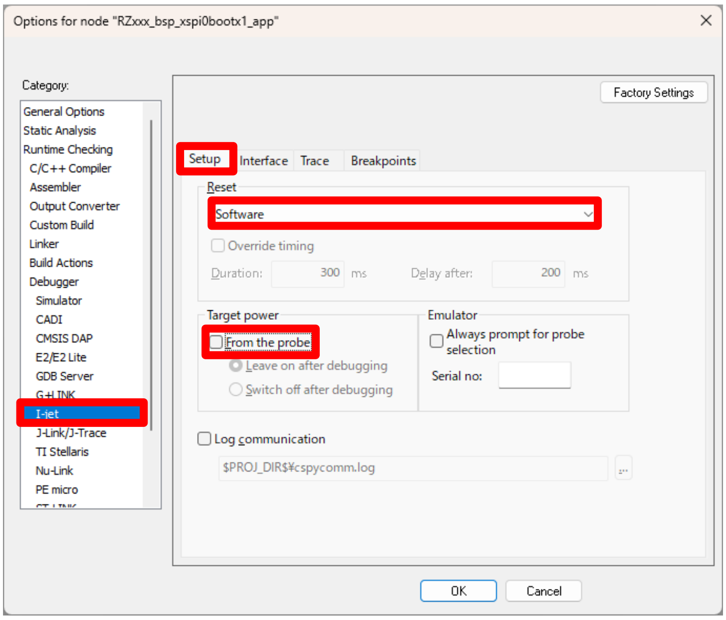


**Figure 5-12 Application project debug settings (2)**

**Supplementary Notes for the EWARM Environment (6)**

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

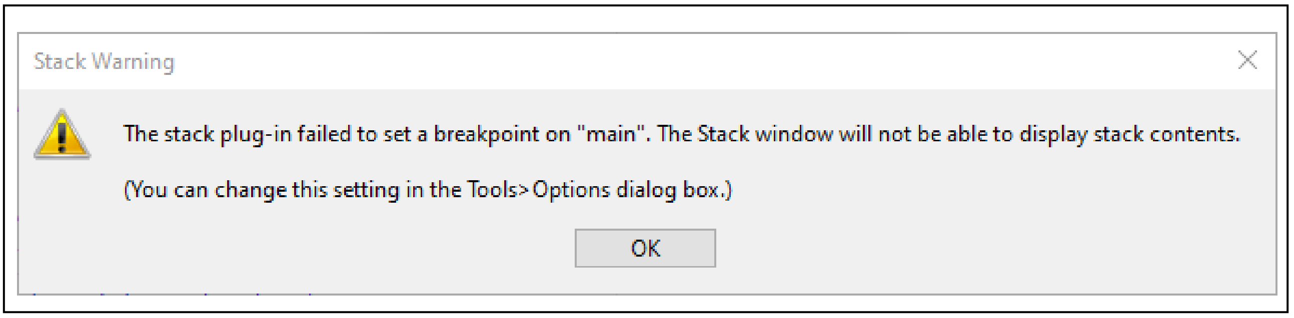


**Figure 5-13 Warning about stack plug-in**

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

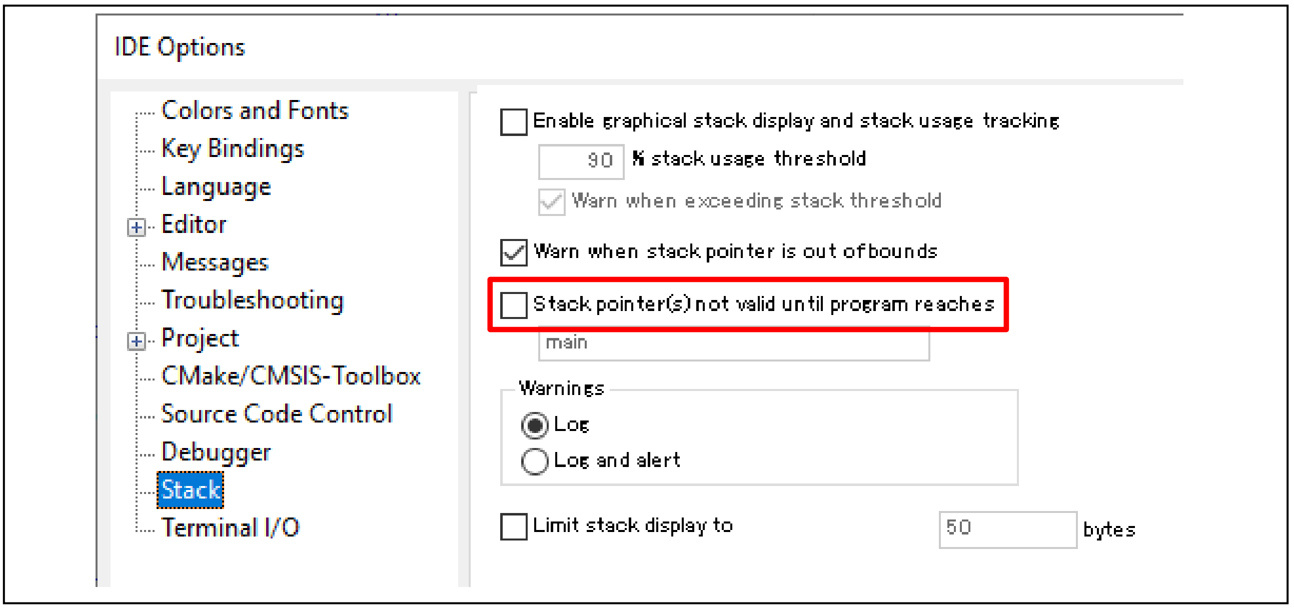


**Figure 5-14 IDE Options setting**

### 5.1.2 e² studio from Renesas

1. Launch e² studio using your workspace. Then, click the 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 “RZT2L_bsp_xspi0bootx1_app” project, push the [Generate Project Content] button to execute code generation.

4. After executing the clean of “RZT2L_bsp_xspi0bootx1_app” project, execute the build.

5. Open configuration.xml on “RZT2L_bsp_xspi0bootx1_loader” project, push the [Generate Project Content] button to execute code generation.

6. After executing the clean of “RZT2L_bsp_xspi0bootx1_loader” project, execute the build.

7. Make sure that your PC and RZ/T2L RSK board are connected via J-Link. Then, select “RZT2L_bsp_xspi0bootx1_loader” in the connection settings, and click [Debug] to start debugging.

8. After the emulator is connected, both the loader program and the application program are downloaded to the external serial flash memory by Flash Downloader. After the download is complete, debugging starts and the program begins 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”

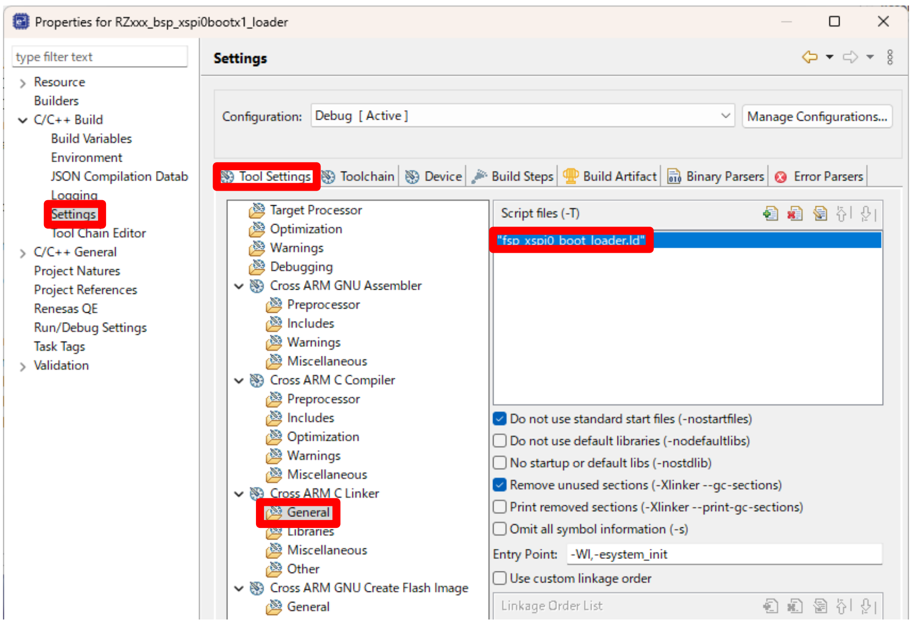


**Figure 5-15 Setting the linker script file path for the Loader project**

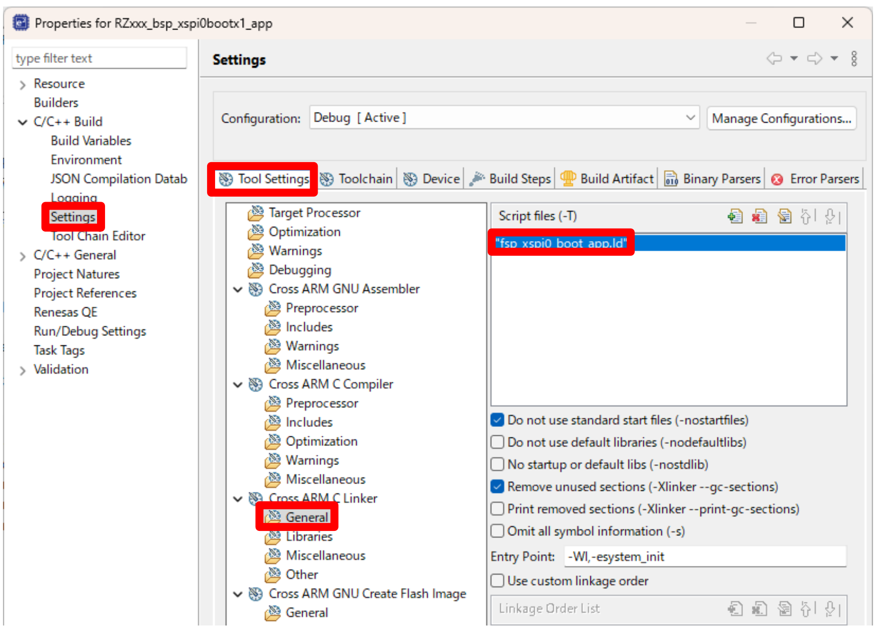


**Figure 5-16 Setting the linker script file path for the Application project**

**Supplementary Notes for the e[2] 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".

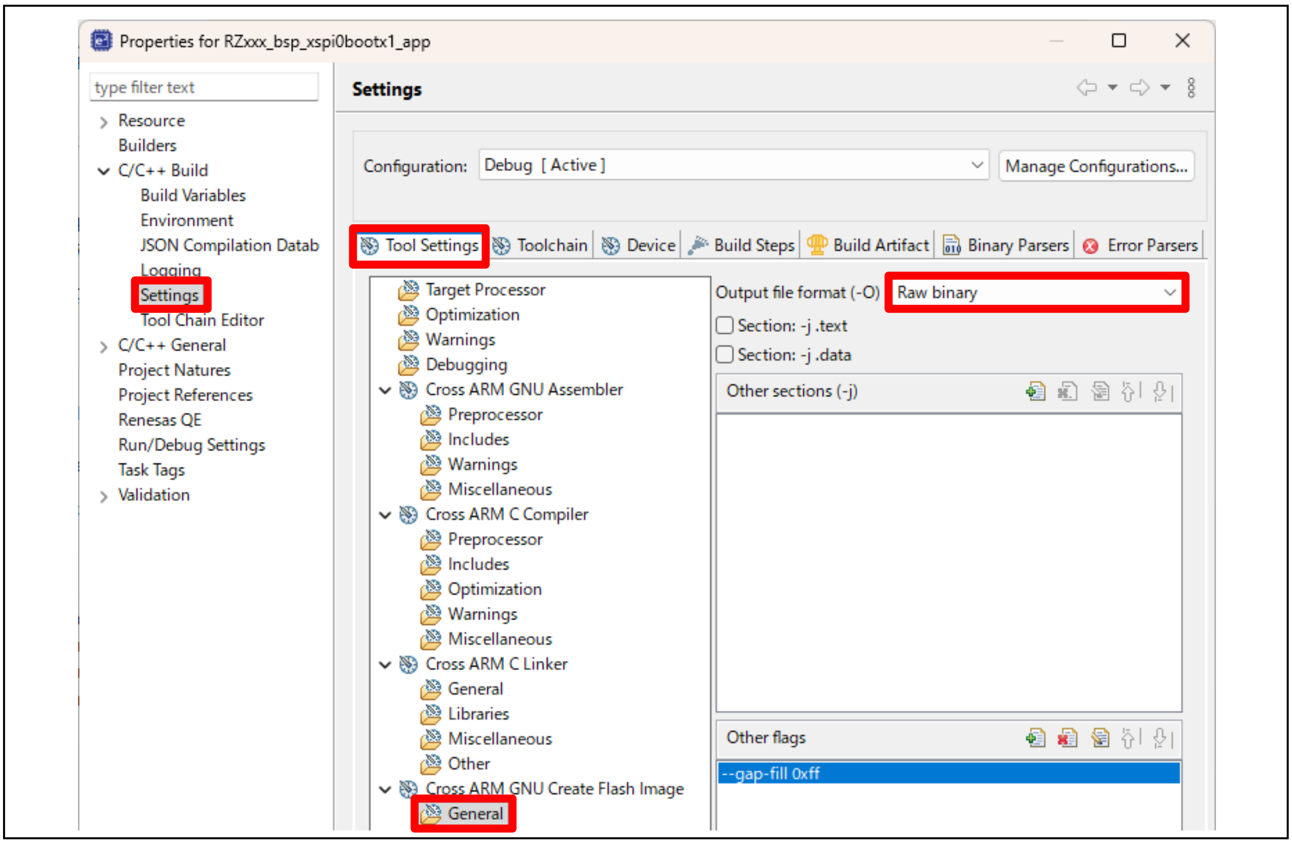


**Figure 5-17 Configuring the output format for the Application project**

**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.

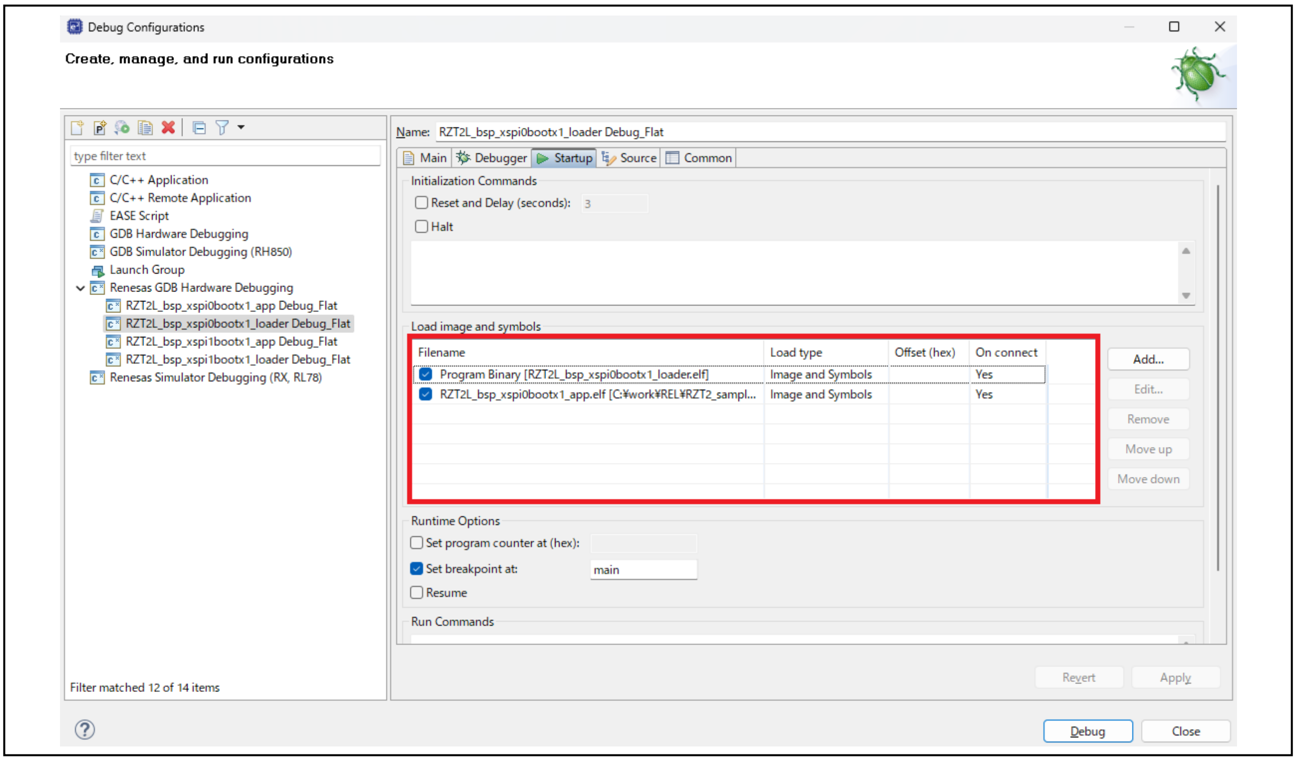


**Figure 5-18 Debug configurations in e[2] studio**

### 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".

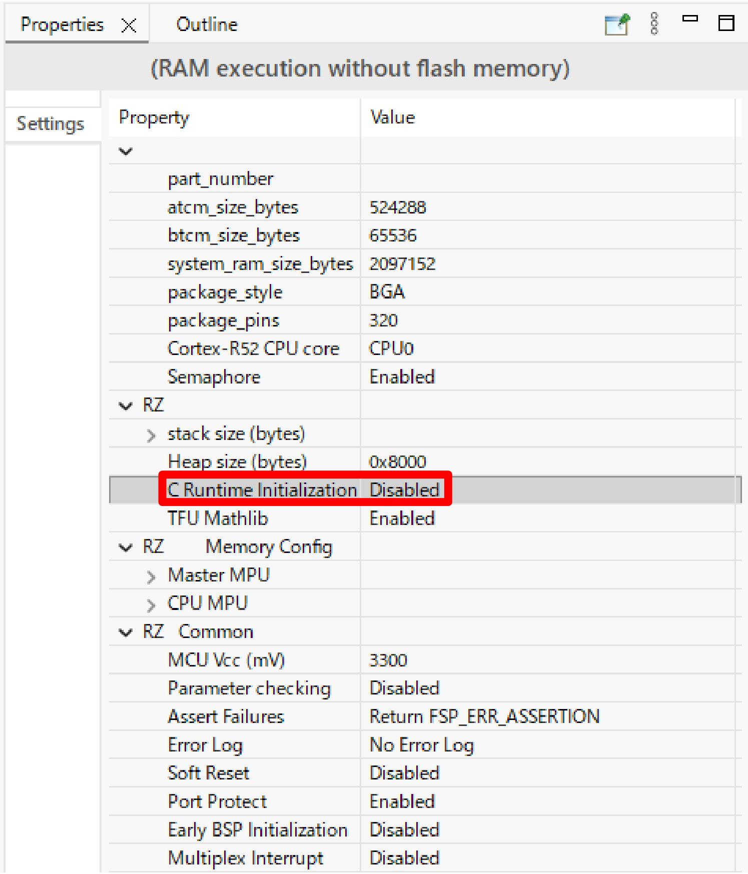


**Figure 5-19 BSP settings for Application programs**

**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".

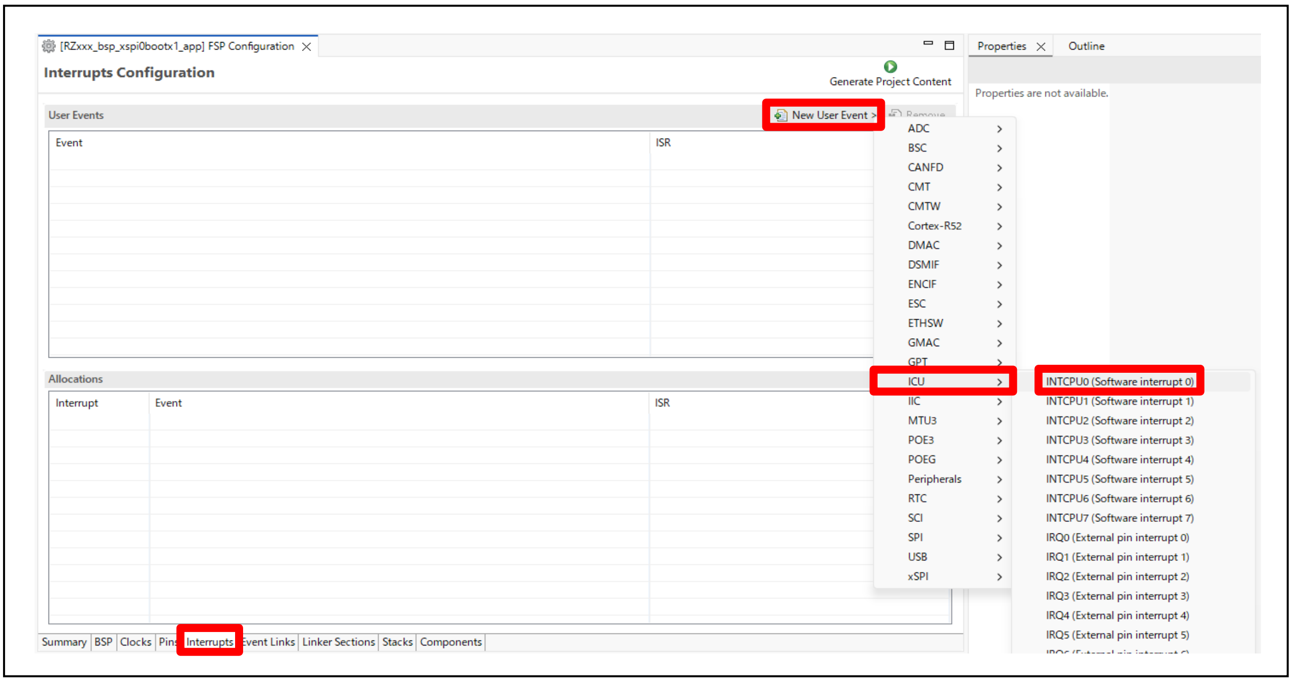


**Figure 5-20 Application program Interrupts settings (1)**

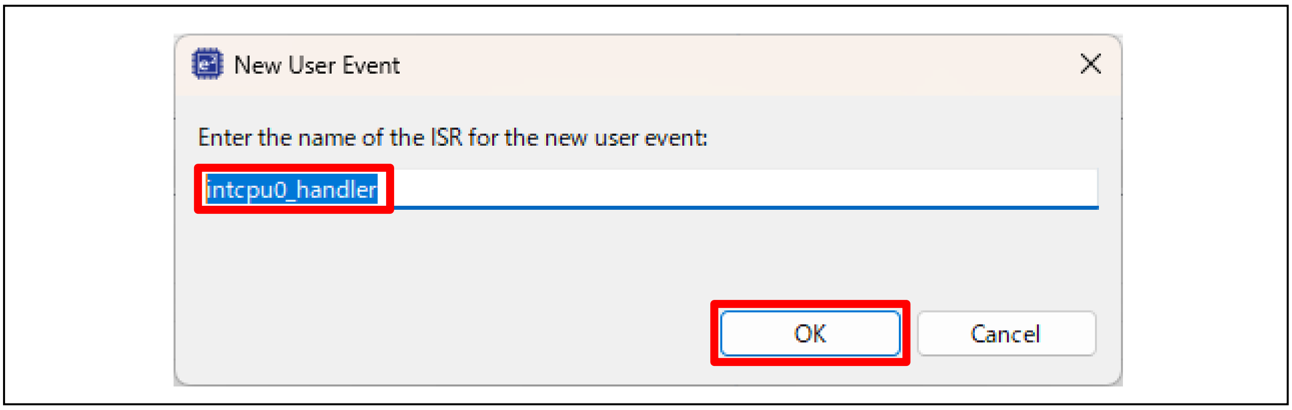


**Figure 5-21 Application program Interrupts settings (2)**

**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 (アプリケーションプログラムのリンカスクリプト)

```
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 stack 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

~~~

After Changing
~~~
.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

~~~
```

## Revision History

| Rev. | Date | Description | |
|---|---|---|---|
| | | Page | Summary |
| 2.00 | Sep.13, 2024 | - | First edition issued. |
| 2.10 | May.16, 2025 | 14, 31 | Change DATA_RBLOCK to USER_DATA_RBLOCK<br>Change DATA_WBLOCK to USER_DATA_WBLOCK<br>Change DATA_ZBLOCK to USER_DATA_ZBLOCK |
| | | 31, 34 | Removed instructions for placing sections in BTCM. |
| 3.00 | Dec. 19, 2025 | - | FSP v3.00 support |
| 3.01 | Mar. 31, 2026 | 32 ~ 35 | Added supplementary notes for the sample program in the EWARM environment. |
| | | 37 | Changes to the debugging procedure for the $e^2$ studio environment |
| | | 38, 39 | Added supplementary notes for the sample program in the $e^2$ studio environment. |
| | | 41, 42 | Added supplementary notes for the application program and loader 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/.