RZ/T2, RZ/N2

Quick Start Guide: Firmware Update

Introduction

This document describes the procedure for updating user application programs via Ethernet by using the RZ/T2, RZ/N2 firmware update sample programs.

Target Device

RZ/T2M Group

RZ/T2L Group

RZ/N2L Group

RZ/T2H Group

RZ/N2H Group

Contents

1. Overview

1.1 Introduction

This document describes the functions provided by the RZ/T2, RZ/N2 firmware update sample programs and explains how to use the various tools.

This sample program package uses the Flexible Software Package for RZ/T2M, RZ/T2L, RZ/T2H, RZ/N2L and RZ/N2H. For more information about FSP, please refer to RZ/T2, RZ/N2 Getting Started with Flexible Software Package.

The sample program can update user application programs in the external flash memory on the boards, and the outline of the sample program is as follows. For details, please refer to Chapters 2, 3 and 4.

- Upon system reset, a loader program called the Secondary Stage Boot Loader (SSBL) is initiated from external flash.
- Based on the setting of a DIP switch, the SSBL will either load and start the user application, or an update program.
- The update program receives a binary via ethernet and replaces the user application in the external flash.
- Preparation of the binary for the update and transfer to the board is done via Python script on the PC.
- The initial flashing of the firmware (SSBL, update program and user application) is handled by scripts and sample program included in the package from the *RZ/T2, RZ/N2 Device Setup Guide for Flash boot (R01AN6471EJ****)* Application Note.

1.2 Features

The sample program has the following features:

- It is possible to update via Ethernet user application programs written to the QSPI flash, OSPI flash and NOR flash memory on Renesas Starter Kit+ for RZ/T2M, RZ/T2L and RZ/N2L.
- It is possible to update via Ethernet user application programs written to the QSPI flash, OSPI flash memory on Evaluation Board for RZ/T2H and RZ/N2H.
- If the update of a user application program fails, the user can redo the update as many times as necessary until the update is successful.
- RZ/T2M, RZ/T2H Cortex®-R52 and RZ/N2H Cortex®-R52 can update applications that use dual cores.
- RZ/T2H Cortex®-A55 and RZ/N2H Cortex®-A55 can update applications that use multi cores.

1.3 Limitations

The sample program has the following limitations:

- It is not possible to update a user application program while a user application program is running.

1.4 Package Contents

RZ/T2, RZ/N2 firmware update sample program package contains several files with software and tools. The following table lists their contents.

Table 1.1 Firmware Update Sample Program Common Package Contents

No.	File Path	Classification	Remarks
1	fwupdate_utility.py	Tool	Update file generator tool
2	fwupdate.py	Tool	Update files send tool
3	r01an6472ej0310-rzt2-n2-fwupdate.pdf	Document	This document RZ/T2, RZ/N2 Quick Start Guide: Firmware Update
4	r01an6641ej0310-rzt2-n2-releasenote.pdf	Document	Release Note

Table 1.2 Firmware Update Sample Program Common Package Contents for RZ/T2M

No.	File Path	Classification	Remarks
1	RZT2M_RSK_FWUpdate_Rev300.zip	Software	Sample program code for RZ/T2M Note1
2	RZT2M_RSK_FWUpdate.bin	Software	Programs and data for RZ/T2M stored in the “Pre-built parameters and programs” folder. Note2 (The application's bin file was generated by the IAR Compiler project)
3	RZT2M_RSK_SSBL.bin	Software	
4	RZT2M_RSK_SSBL_xspi0.bin	Software & data	
5	parameter_RZT2M_bsp_LED_0.bin	Data	
6	parameter_RZT2M_bsp_LED_1.bin	Data	
7	initial_image_RZT2M_xspi0.bin	Data	
8	RZT2M_bsp_LED_0.bin	Software	
9	RZT2M_bsp_LED_1.bin	Software	
10	RZT2M_bsp_LED_x.zip	Software	
11	RZT2M_bsp_LED_x_CPU1.zip	Software	

Note1 Contains sample program projects for GCC and IAR compilers.
Note2 These files are used in Chapters 3 and 4 of this document as a reference for building the environment.

Table 1.3 Firmware Update Sample Program Common Package Contents for RZ/T2L

No.	File Path	Classification	Remarks
1	RZT2L_RSK_FWUpdate_Rev300.zip	Software	Sample program code for RZ/T2L Note1
2	RZT2L_RSK_FWUpdate.bin	Software	Programs and data for RZ/T2L stored in the “Pre-built parameters and programs” folder. Note2 (The application's bin file was generated by the IAR Compiler project)
3	RZT2L_RSK_SSBL.bin	Software	
4	RZT2L_RSK_SSBL_xspi0.bin	Software & data	
5	parameter_RZT2L_bsp_LED_1.bin	Data	
6	parameter_RZT2L_bsp_LED_3.bin	Data	
7	initial_image_RZT2L_xspi0.bin	Data	
8	RZT2L_bsp_LED_1.bin	Software	
9	RZT2L_bsp_LED_3.bin	Software	
10	RZT2L_bsp_LED_x.zip	Software	

Note1 Contains sample program projects for GCC and IAR compilers.
Note2 These files are used in Chapters 3 and 4 of this document as a reference for building the environment.

Table 1.4 Firmware Update Sample Program Common Package Contents for RZ/N2L

No.	File Path	Classification	Remarks
1	RZN2L_RSK_FWUpdate_Rev310.zip	Software	Sample program code for RZ/N2L Note1
2	RZN2L_RSK_FWUpdate.bin	Software	Programs and data for RZ/N2L stored in the “Pre-built parameters and programs” folder. Note2 (The application's bin file was generated by the IAR Compiler project)
3	RZN2L_RSK_SSBL.bin	Software	
4	RZN2L_RSK_SSBL_xspi0.bin	Software & data	
5	parameter_RZN2L_bsp_LED_0.bin	Data	
6	parameter_RZN2L_bsp_LED_3.bin	Data	
7	initial_image_RZN2L_xspi0.bin	Data	
8	RZN2L_bsp_LED_0.bin	Software	
9	RZN2L_bsp_LED_3.bin	Software	
10	RZN2L_bsp_LED_x.zip	Software	

Note1 Contains sample program projects for GCC and IAR compilers.
Note2 These files are used in Chapters 3 and 4 of this document as a reference for building the environment.

Table 1.5 Firmware Update Sample Program Common Package Contents for RZ/T2H Cortex®-R52

No.	File Path	Classification	Remarks
1	RZT2H_EVB_FWUpdate_Rev300.zip	Software	Sample program code for RZ/T2H Cortex®-R52 Note1
2	RZT2H_EVB_FWUpdate.bin	Software	Programs and data for RZ/T2H Cortex®-R52 stored in the “Pre-built parameters and programs/cr52” folder. Note2 (The application's bin file was generated by the IAR Compiler project)
3	RZT2H_EVB_SSBL.bin	Software	
4	RZT2H_EVB_SSBL_xspi0.bin	Software & data	
5	parameter_RZT2H_bsp_LED_0.bin	Data	
6	parameter_RZT2H_bsp_LED_1.bin	Data	
7	initial_image_RZT2H_xspi0.bin	Data	
8	RZT2H_bsp_LED_0.bin	Software	
9	RZT2H_bsp_LED_1.bin	Software	
10	RZT2H_bsp_LED_x.zip	Software	
11	RZT2H_bsp_LED_x_CPU1.zip	Software	

Note1 Contains sample program projects for GCC and IAR compilers.
Note2 These files are used in Chapters 3 and 4 of this document as a reference for building the environment.

Table 1.6 Firmware Update Sample Program Common Package Contents for RZ/T2H Cortex®-A55

No.	File Path	Classification	Remarks
1	RZT2H_EVB_CA55_FWUpdate_Rev300.zip	Software	Sample program code for RZ/T2H Cortex®-A55 Note1
2	RZT2H_EVB_CA55_FWUpdate.bin	Software	Programs and data for RZ/T2H Cortex®-A55 stored in the “Pre-built parameters and programs/ca55” folder. Note2 (The application's bin file was generated by the IAR Compiler project)
3	RZT2H_EVB_CA55_SSBL.bin	Software	
4	RZT2H_EVB_SSBL_xspi0.bin	Software & data	
5	parameter_RZT2H_bsp_LED_0.bin	Data	
6	parameter_RZT2H_bsp_LED_1.bin	Data	
7	initial_image_RZT2H_xspi0.bin	Data	
8	RZT2H_bsp_LED_0.bin	Software	
9	RZT2H_bsp_LED_1.bin	Software	
10	RZT2H_bsp_LED_x_CA55_Core0.zip	Software	
11	RZT2H_bsp_LED_x_CA55_Core1.zip	Software	
12	RZT2H_bsp_LED_x_CA55_Core2.zip	Software	
13	RZT2H_bsp_LED_x_CA55_Core3.zip	Software	

Note1 Contains sample program projects for GCC and IAR compilers.
Note2 These files are used in Chapters 3 and 4 of this document as a reference for building the environment.

Table 1.7 Firmware Update Sample Program Common Package Contents for RZ/N2H Cortex®-R52

No.	File Path	Classification	Remarks
1	RZN2H_EVB_FWUpdate_Rev310.zip	Software	Sample program code for RZ/N2H Cortex®-R52 Note1
2	RZN2H_EVB_FWUpdate.bin	Software	Programs and data for RZ/N2H Cortex®-R52 stored in the “Pre-built parameters and programs/cr52” folder. Note2 (The application's bin file was generated by the IAR Compiler project)
3	RZN2H_EVB_SSBL.bin	Software	
4	RZN2H_EVB_SSBL_xspi0.bin	Software & data	
5	parameter_RZN2H_bsp_LED_3.bin	Data	
6	parameter_RZN2H_bsp_LED_4.bin	Data	
7	initial_image_RZN2H_xspi0.bin	Data	
8	RZN2H_bsp_LED_3.bin	Software	
9	RZN2H_bsp_LED_4.bin	Software	
10	RZN2H_bsp_LED_x.zip	Software	
11	RZN2H_bsp_LED_x_CPU1.zip	Software	

Note1 Contains sample program projects for GCC and IAR compilers.
Note2 These files are used in Chapters 3 and 4 of this document as a reference for building the environment.

Table 1.8 Firmware Update Sample Program Common Package Contents for RZ/N2H Cortex®-A55

No.	File Path	Classification	Remarks
1	RZN2H_EVB_CA55_FWUpdate_Rev310.zip	Software	Sample program code for RZ/N2H Cortex®-A55 Note1
2	RZN2H_EVB_CA55_FWUpdate.bin	Software	Programs and data for RZ/N2H Cortex®-A55 stored in the “Pre-built parameters and programs/ca55” folder. Note2 (The application's bin file was generated by the IAR Compiler project)
3	RZN2H_EVB_CA55_SSBL.bin	Software	
4	RZN2H_EVB_SSBL_xspi0.bin	Software & data	
5	parameter_RZN2H_bsp_LED_3.bin	Data	
6	parameter_RZN2H_bsp_LED_4.bin	Data	
7	initial_image_RZN2H_xspi0.bin	Data	
8	RZN2H_bsp_LED_3.bin	Software	
9	RZN2H_bsp_LED_4.bin	Software	
10	RZN2H_bsp_LED_x_CA55_Core0.zip	Software	
11	RZN2H_bsp_LED_x_CA55_Core1.zip	Software	
12	RZN2H_bsp_LED_x_CA55_Core2.zip	Software	
13	RZN2H_bsp_LED_x_CA55_Core3.zip	Software	

Note1 Contains sample program projects for GCC and IAR compilers.
Note2 These files are used in Chapters 3 and 4 of this document as a reference for building the environment.

1.5 Related Documents

Table 1.9 lists documents related to this document.

Table 1.9 Related Documents

Title	Document Number
RZ/T2M Group Renesas Starter Kit+ for RZ/T2M User's Manual	R20UT4939EG****
RZ/T2M Group Renesas Starter Kit+ for RZ/T2M Quick Start Guide	R20UT4941EG****
RZ/T2, RZ/N2 Getting Started with Flexible Software Package	R01AN6434EJ****
RZ/T2M Group User's Manual: Hardware	R01UH0916EJ****
RZ/T2, RZ/N2 Device Setup Guide for Flash boot	R01AN6471EJ****
RZ/T2L Group Renesas Starter Kit+ for RZ/T2L User's Manual	R20UT5164EJ****
RZ/T2L Group Renesas Starter Kit+ for RZ/T2L Quick Start Guide	R20UT5235EJ****
RZ/T2L Group User's Manual: Hardware	R01UH0985EJ****
RZ/N2L Group Renesas Starter Kit+ for RZ/N2L User's Manual	R20UT4984EG****
RZ/N2L Group Renesas Starter Kit+ for RZ/N2L Quick Start Guide	R20UT4986EG****
RZ/N2L Group User's Manual: Hardware	R01UH0955EJ****
RZ/T2H and RZ/N2H Groups User's Manual: Hardware	R01UH1039EJ****
RZ/T2H Group RZ/T2H Evaluation Board User's Manual	R20UT5405EJ****
RZ/N2H Group RZ/N2H Evaluation Board User's Manual	R20UT5522EJ****

1.6 Explanation of Terms

The meanings of terms used in this document are indicated below.

Term Used in This Document	Meaning of Term
Update program	The program, contained in the sample program package, used to update user application programs.
User application program	Program that can be updated with this sample program package.
Update file	The program to be updated.
Firmware update system	The file containing the program to be updated.
SSBL	Second stage boot loader, referred to as a loader program in following documents: RZ/T2M Group Renesas Starter Kit+ for RZ/T2M User's Manual RZ/T2L Group Renesas Starter Kit+ for RZ/T2L User's Manual RZ/N2L Group Renesas Starter Kit+ for RZ/N2L User's Manual RZ/T2H Group RZ/T2H Evaluation Board User's Manual RZ/N2H Group RZ/N2H Evaluation Board User's Manual
Loader program	SSBL, Second stage boot loader

2. Firmware Update Mechanism

The sample program can update user application programs written to the external flash memory on the boards. Figure 2.1 illustrates the system structure of the sample program.

The target devices are RZ/T2M, RZ/T2L, RZ/T2H, RZ/N2L or RZ/N2H. The target boards are the Renesas Starter Kit+ (hereinafter referred to as RSK+) for RZ/T2M, RZ/T2L, RZ/N2L or Evaluation Board for RZ/T2H, RZ/N2H (hereinafter referred to as EVB).

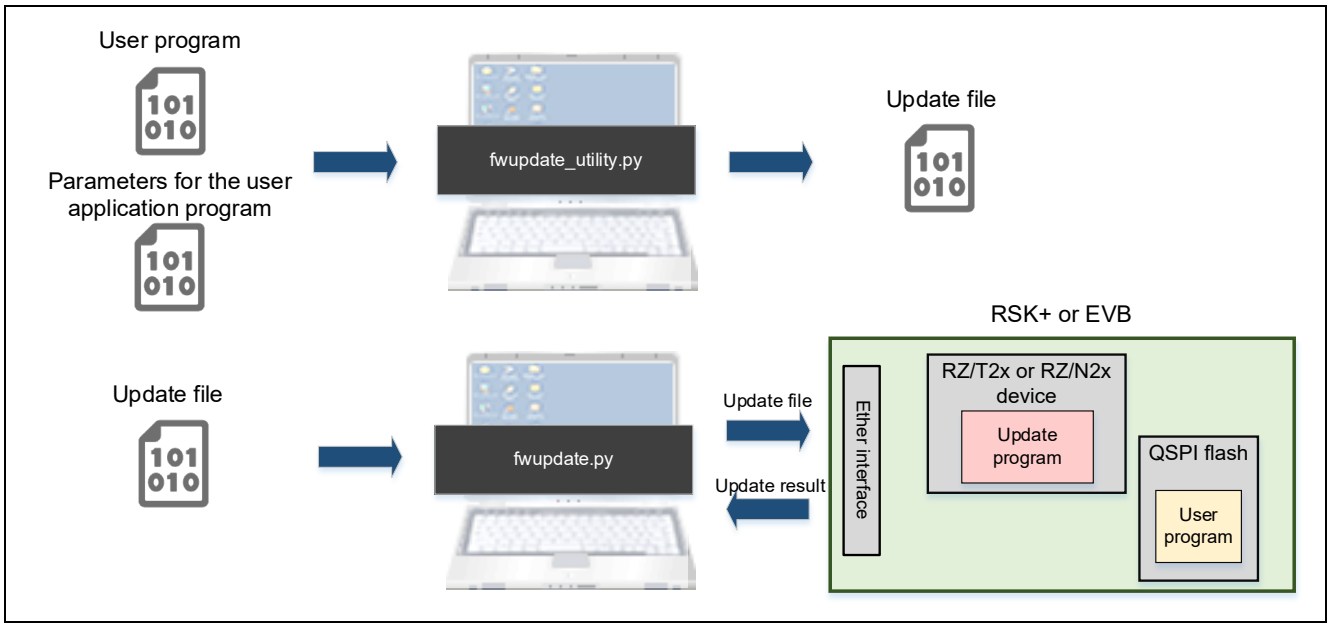


Figure 2.1 System Structure of Firmware Update Sample Program

The sample program updates user application programs through the following sequence of steps:

1. Using fwupdate_utility.py, the user creates an update file containing the user application program to be applied as an update and information on its location in the external flash memory. Refer to 5.4.1 for details of fwupdate_utility.py and to 5.1 for details of the update file.
2. The user starts the device in update mode.
3. Using fwupdate.py, the user sends the update file from the host PC to the device via an Ethernet connection. Refer to 5.2 for the communication protocols used between the host PC and the board and the packet format of the communication protocols.
4. When the update file is received by the device, the update program on the device extracts the user application program and writes it to the external flash memory. Refer to 5.3 for details of the update program.

2.1 Operating Modes

You can select the operating mode of the sample program by means of switch settings on the evaluation board. Refer to Table 2.1 for the operating mode selection method.

Table 2.1 Switches Used for Operating Mode Selection

Board	Switch	MCU Port	MCU Pin	Operating Mode
RZ/T2M RSK+	SW3-1	P11_0	Y18	OFF: Application mode ON: Update mode
RZ/T2L RSK+	SW3-2	P04_1	F1	OFF: Application mode ON: Update mode
RZ/N2L RSK+	SW3-1	P13_6	M13	OFF: Application mode ON: Update mode
RZ/T2H EVB	SW12-1	P35_3	V22	OFF: Application mode ON: Update mode
RZ/N2H EVB	DSW1-1	P27_2	B24	OFF: Application mode ON: Update mode

The update program is launched at startup when update mode is selected as the operating mode, and the user application program is launched at startup when application mode is selected. Therefore, the sample program cannot perform an update while a user application program is running.

2.2 Sample Program Configuration

Operating mode checking and launching of the program corresponding to the operating mode is performed by a loader program. In this document, the loader program is referred to as SSBL.

2.3 Using External Flash Memory

The sample program stores the programs that comprise the system in the external flash memory on the evaluation board, then loads them into the RAM and runs them. Figure 2.2, Figure 2.3, Figure 2.4, Figure 2.5 and Figure 2.6 show how the sample program utilizes the external flash memory.

The sample program updates the target user application program, but it does not update the loader program and update program. Therefore, when updating fails, any number of update attempts may be performed until the update succeeds.

In this sample program, you can configure whether the user application program area of the external flash memory that can be updated by the update program is treated as one plane or divided into two planes. If the user application program area of the external flash memory is treated as one plane, updating the user application program will overwrite a user application program in the external flash memory. If the user application program area is divided into two planes, updating the user application program area will update the user application program area on one side of the two planes in the external flash memory and leave the program before the update in the other user application program area.

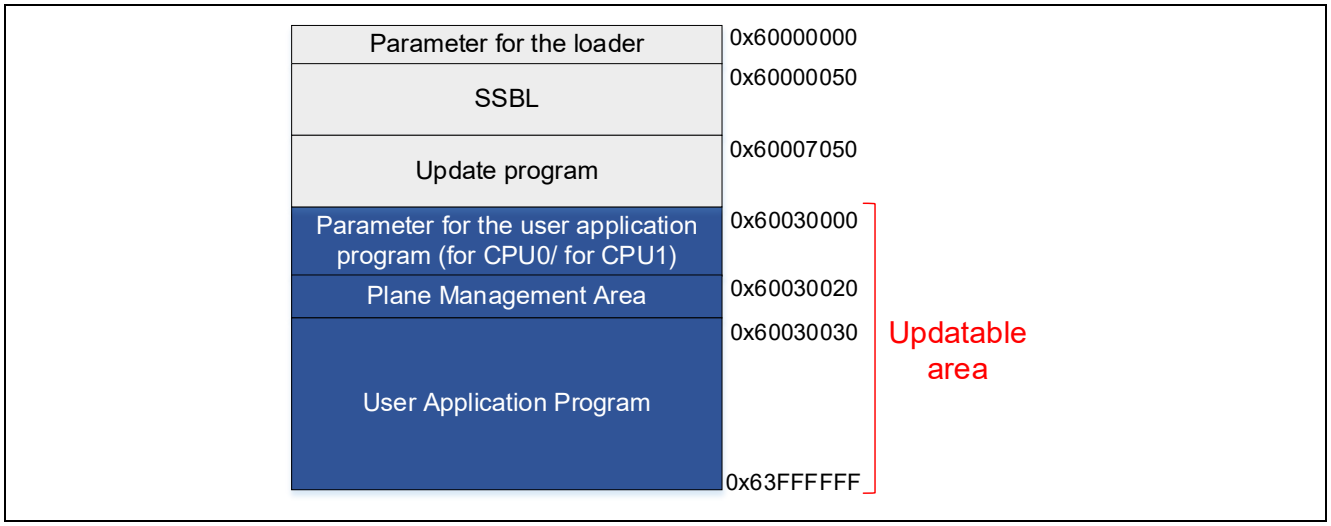


Figure 2.2 xSPI0 Area Flash Memory Usage on RSK+

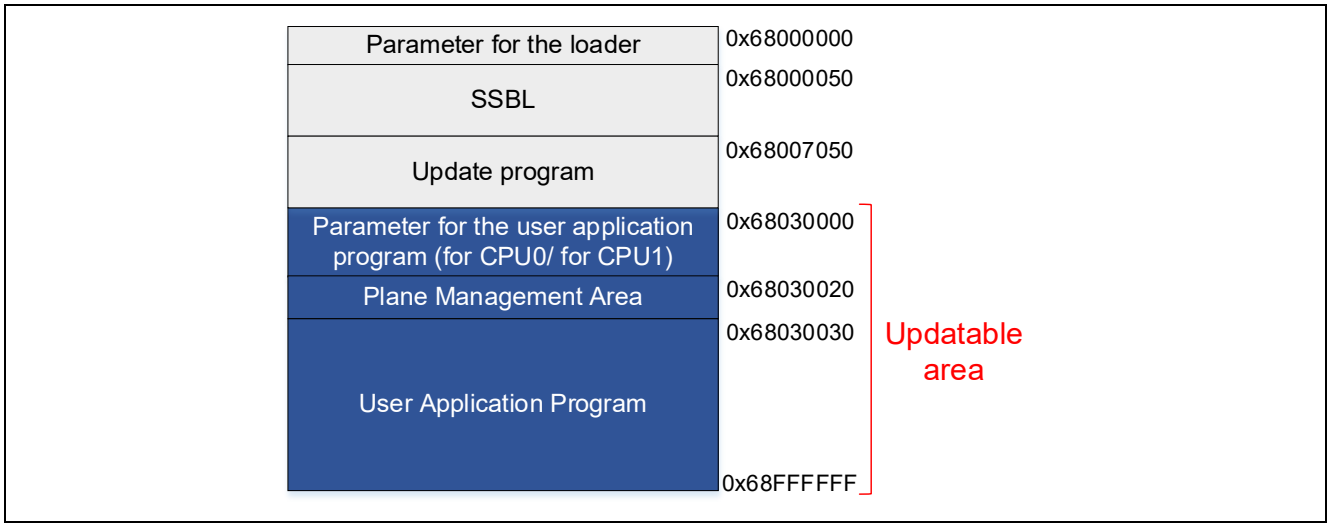


Figure 2.3 xSPI1 Area Flash Memory Usage on RSK+

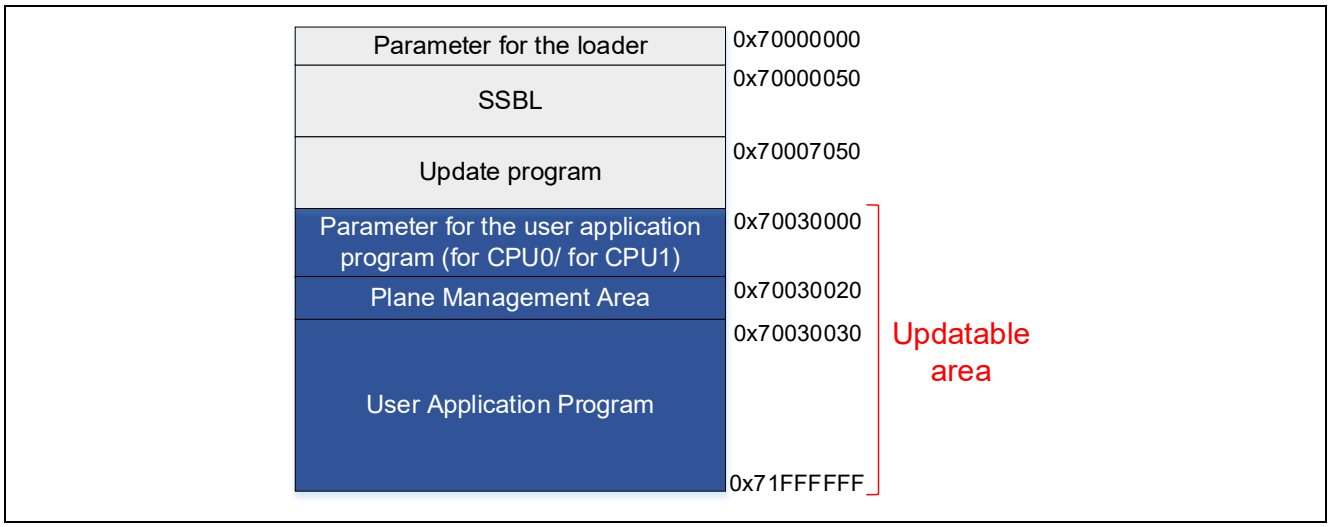


Figure 2.4 External Bus Area Flash Memory Usage on RSK+

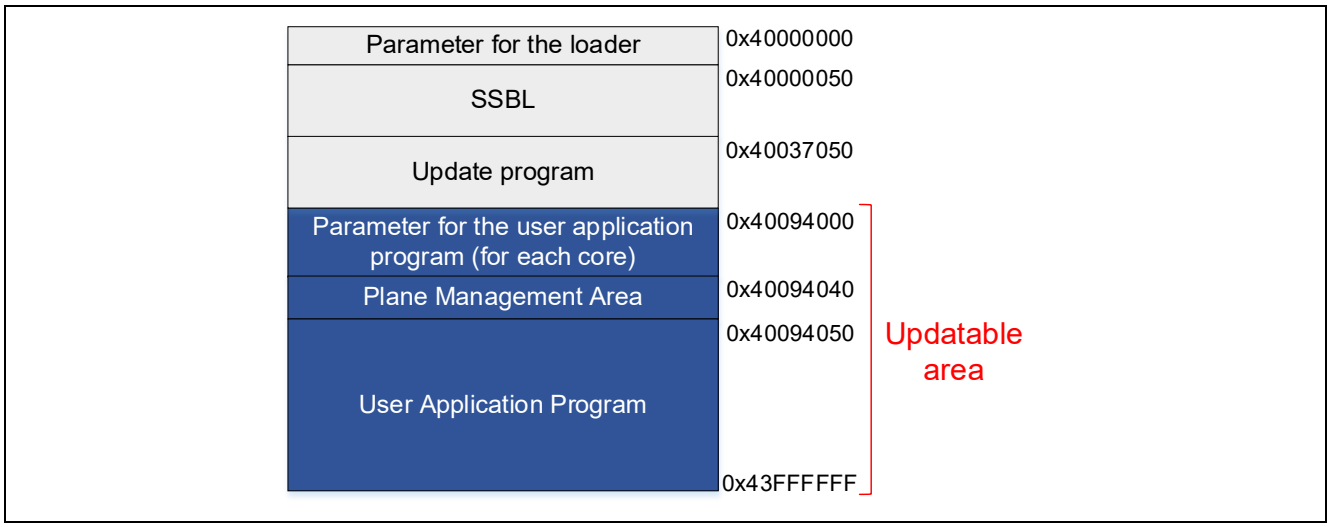


Figure 2.5 xSPI0 Area Flash Memory Usage on EVB

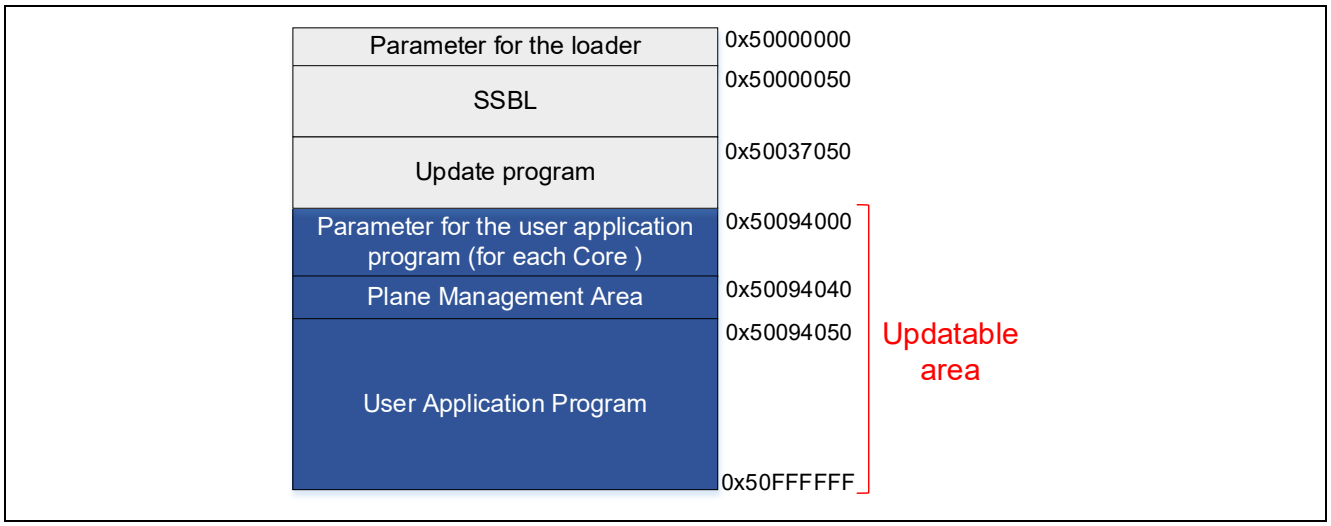


Figure 2.6 xSPI1 Area Flash Memory Usage on EVB

3. Configuring the Firmware Update System

The system configuration procedures for updating user application programs using the pre-built items supplied as part of the sample program package are explained in Sections 3.1 to 3.5 for each corresponding MPU.

If you want to build an update program, SSBL and user application by building the sample project instead of using the pre-built items as they are, please refer to Sections 3.6 to 3.8.

3.1 Configuring the Firmware Update System for RZ/T2M

Table 3.1 shows the environment required for configuring the RZ/T2M RSK+. Table 3.2 lists the user application program written during device setup for RZ/T2M.

Table 3.1 Setup Environment for RZ/T2M RSK+

Name	Remarks
Renesas Starter Kit+ for RZ/T2M	RZ/T2M RSK+
USB cables	1 (Mini-B, type-A)
	1 (Type-C, type-A)
Windows host PC	IAR Embedded Workbench or e^2 studio installed
parameter_generator.py Note	Generation tool for the parameter for the loader and the parameter for the user application program
device_setup.py Note	Command sending tool for device setup
RZT2M_RSK_DeviceSetup.out.srec Note	S-Record format device setup sample program

Note Included in the sample program package for *RZ/T2, RZ/N2 Device Setup Guide for Flash boot (R01AN6471EJ****)*.

Table 3.2 User Application Program Set Up for RZ/T2M

File Name	Description
RZT2M_bsp_LED_0.bin	A user application program based on the Blinky sample application included in the Flexible Software Package RZ/T2M pack. See section 3.8 for detailed creation instructions. The program is assigned from 0x00000000 in the ATCM area. The following LEDs will light up at startup. User LED0: BSP_IO_PORT_19_PIN_6
parameter_RZT2M_bsp_LED_0.bin	Parameter for the user application program (RZT2M_bsp_LED_0.bin). This file is created after building RZT2M_bsp_LED_0.bin. See section 3.8 for detailed creation instructions. The parameter file sets the following xSPI0 address space. External flash address where the program is stored: 0x60030030 RAM address where the program is loaded: 0x00000000

3.1.1 Concatenate Program and Parameter

Use fwupdate_utility.py to concatenate the Update program, SSBL including the parameter for the loader, and the parameter for the user application program. The concatenated data is programmed at the beginning of the external flash. See Table 5.13 for memory maps of the concatenated data.

Open a command prompt on the host PC and run the following command.

[Only for multi-core configuration] You must specify both the user application parameters for CPU0 (--param_cpu_core0) and the user application parameters for CPU1 (--param_cpu_core_1).

The following command will generate initial_image_RZT2M_xspi0.bin:

```
python fwupdate_utility.py setupfile --param_loader RZT2M_RSK_SSBL_xspi0.bin
--param_cpu_core_0 parameter_RZT2M_bsp_LED_0.bin --mpu rzt2m --update_prog
RZT2M_RSK_FWUpdate.bin -o initial_image_RZT2M_xspi0.bin
```

3.1.2 Program to Flash

Programming the update program and user application program are accomplished using device setup tool and sample program. Refer to chapter 2 in *RZ/T2, RZ/N2 Device Setup Guide for Flash boot* for the procedure.

1. Write program files to external flash on RSK+ using device_setup.py. Open a command prompt on the host PC and run the following command.

   The programmable area changes depending on the set boot mode. Please check 5.3.5 for boot modes and corresponding memory maps.
   The initial image file (initial_image_RZ*_xspi0.bin) is programmed at the top of the external. The user application (RZ*_bsp_LED_*.bin) is programmed with the flash address specified during parameter generation.

   RZ/T2M:

```
> python device_setup.py writeflash --port COM4 --addr 60000000 -i
initial_image_RZT2M_xspi0.bin
writeflash : Setup success.
> python device_setup.py writeflash --port COM4 --addr 60030030 -i
RZT2M_bsp_LED_0.bin
writeflash : Setup success.
```

2. Set the SW on the RSK+ board to the following, User DIP Switch SW3-1 to OFF and press the reset button S3.
   After that, to confirm that the user application program starts and User LED0 on the board blinks.

   RZ/T2M:

SW	Setting	Description
SW4.1	ON	xSPI0 boot mode (x1 boot Serial flash)
SW4.2	ON	
SW4.3	ON	
SW4.4	ON	JTAG Authentication by Hash is disabled.
SW4.5	OFF	ATCM 1 wait
SW6.1	ON	Enable LED0

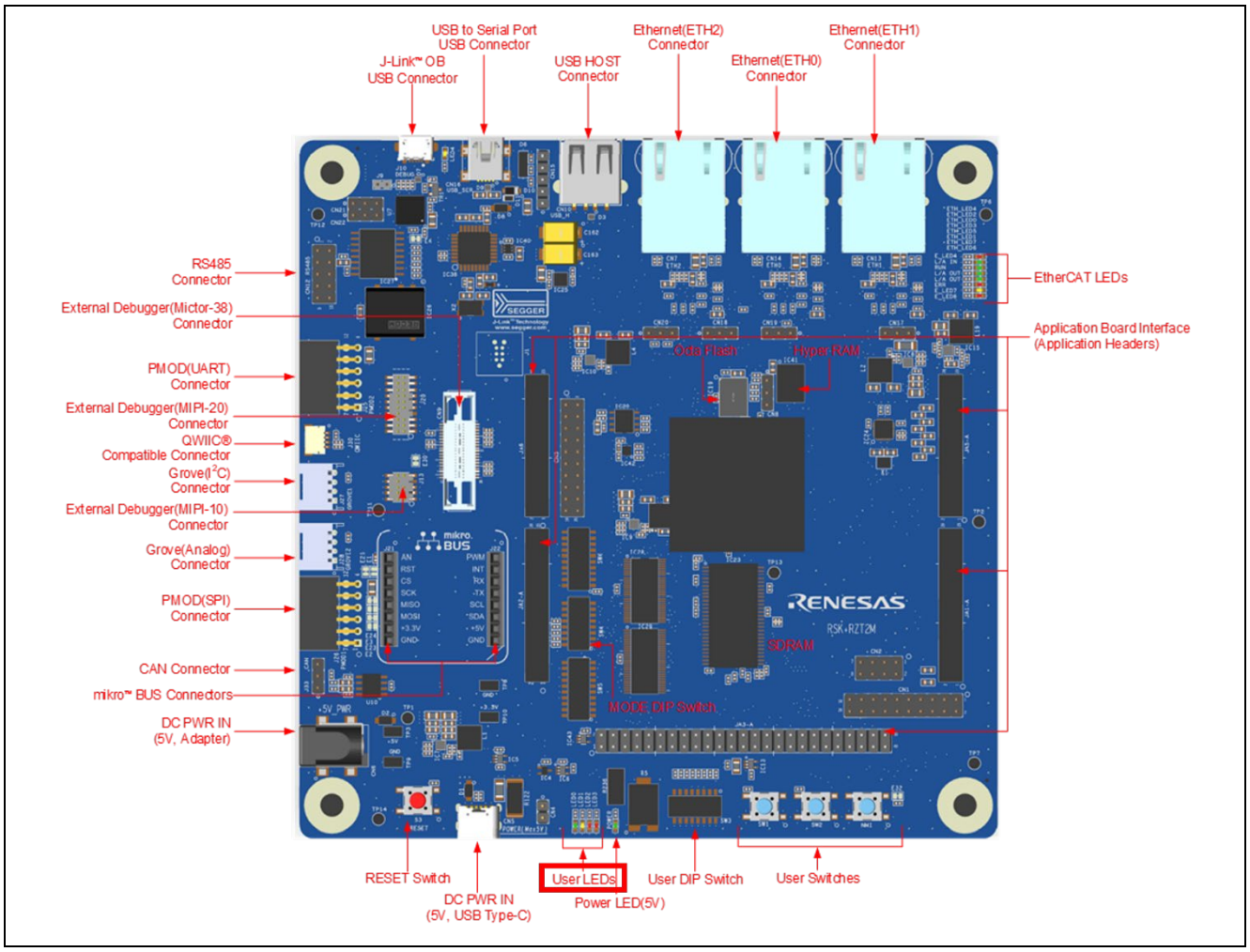


Figure 3.1 Location of User LEDs (LED0) for RZ/T2M

3.2 Configuring the Firmware Update System for RZ/T2L

Table 3.3 shows the environment required for configuring the RZ/T2L RSK+. Table 3.4 lists the user application program written during device setup for RZ/T2L.

Table 3.3 Setup Environment for RZ/T2L RSK+

Name	Remarks
Renesas Starter Kit+ for RZ/T2L	RZ/T2L RSK+
USB cables	1 (Mini-B, type-A)
	1 (Type-C, type-A)
Windows host PC	IAR Embedded Workbench or e^2 studio installed
parameter_generator.py Note	Generation tool for the parameter for the loader and the parameter for the user application program
device_setup.py Note	Command sending tool for device setup
RZT2L_RSK_DeviceSetup_usb.out.srec Note	S-Record format device setup sample program

Note Included in the sample program package for *RZ/T2, RZ/N2 Device Setup Guide for Flash boot (R01AN6471EJ****)*.

Table 3.4 User Application Program Set Up for RZ/T2L

File Name	Description
RZT2L_bsp_LED_1.bin	A user application program based on the Blinky sample application included in the Flexible Software Package RZ/T2L pack. See section 3.8 for detailed creation instructions. The program is assigned from 0x00000000 in the ATCM area. The following LEDs will light up at startup. User LED0: BSP_IO_PORT_17_PIN_6
parameter_RZT2L_bsp_LED_1.bin	Parameter for the user application program (RZT2L_bsp_LED_1.bin). This file is created after building RZT2L_bsp_LED_1.bin. See section 3.8 for detailed creation instructions. The following OSPI flash addresses are set in the parameter file. External flash address where the program is stored: 0x60030030 RAM address where the program is loaded: 0x00000000

3.2.1 Concatenate Program and Parameter

Use fwupdate_utility.py to concatenate the Update program, SSBL including the parameter for the loader, and the parameter for the user application program. The concatenated data is programmed at the beginning of the external flash. See Table 5.13 for memory maps of the concatenated data.

Open a command prompt on the host PC and run the following command.

The following command will generate initial_image_RZT2L_xspi0.bin:

```
python fwupdate_utility.py setupfile --param_loader RZT2L_RSK_SSBL_xspi0.bin
--param_cpu_core_0 parameter_RZT2L_bsp_LED_1.bin --mpu rzt2l --update_prog
RZT2L_RSK_FWUpdate.bin -o initial_image_RZT2L_xspi0.bin
```

3.2.2 Program to Flash

Programming the update program and user application program are accomplished using device setup tool and sample program. Refer to chapter 2 in *RZ/T2, RZ/N2 Device Setup Guide for Flash boot (R01AN6471EJ****)* for the procedure.

1. Write program files to external flash on RSK+ using device_setup.py. Open a command prompt on the host PC and run the following command.

   The programmable area changes depending on the set boot mode. Please check 5.3.5 for boot modes and corresponding memory maps.
   The initial image file (initial_image_RZ*_xspi0.bin) is programmed at the top of the external. The user application (RZ*_bsp_LED_*.bin) is programmed with the flash address specified during parameter generation.

   RZ/T2L:

```
> python device_setup.py writeflash --port COM4 --addr 60000000 -i
initial_image_RZT2L_xspi0.bin
writeflash : Setup success.
> python device_setup.py writeflash --port COM4 --addr 60030030 -i
RZT2L_bsp_LED_1.bin
writeflash : Setup success.
```

2. Set the SW on the RSK+ board to the following, User DIP Switch SW3-2 to OFF and press the reset button S3.
   After that, to confirm that the user application program starts and User LED1 on the board blinks.

   RZ/T2L:

SW	Setting	Description
SW4.1	ON	xSPI0 boot mode (x1 boot Serial flash)
SW4.2	ON	
SW4.3	ON	
SW4.4	OFF	ATCM wait cycle = 1 wait.
SW4.5	ON	JTAG Authentication by Hash is disabled.

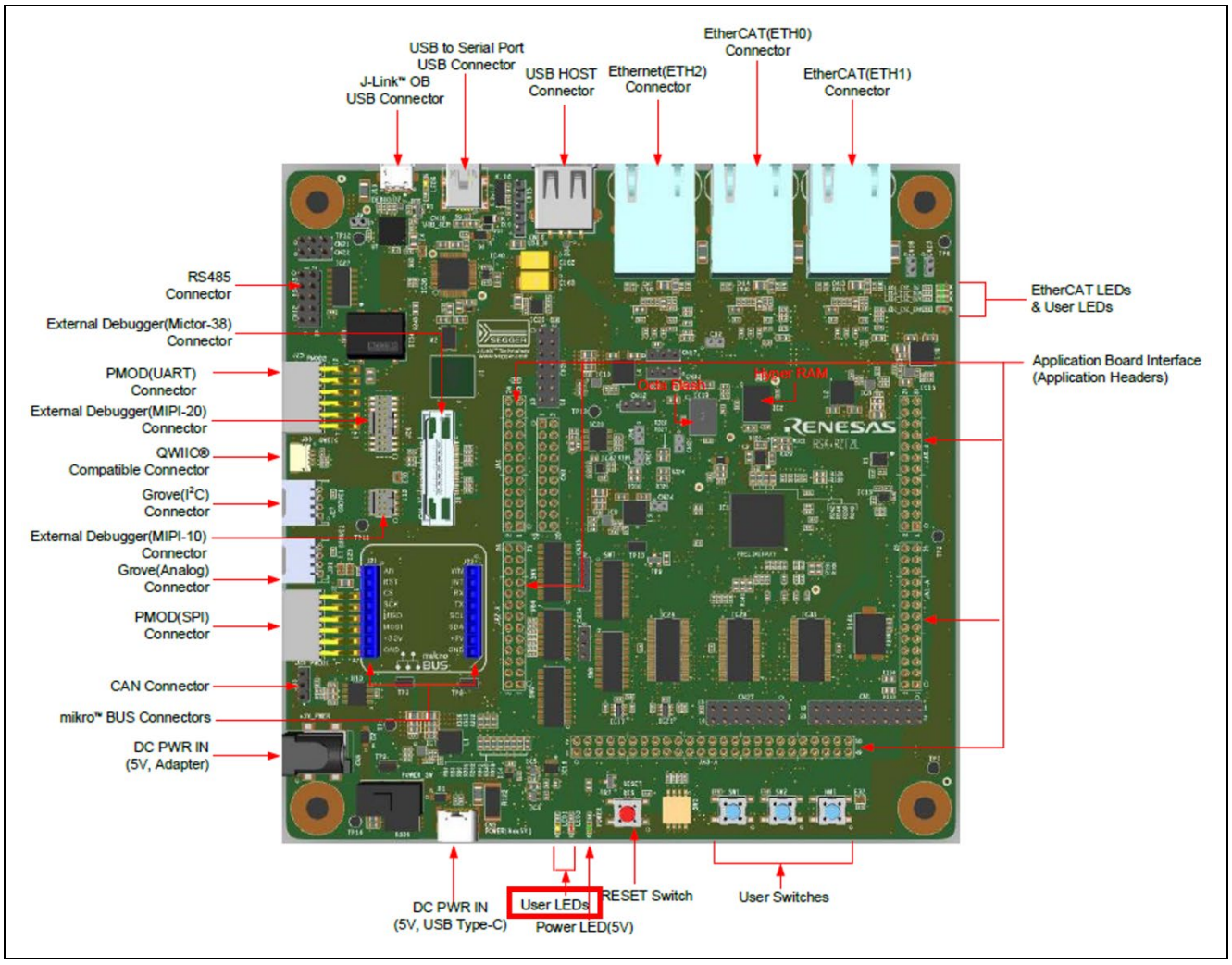


Figure 3.2 Location of User LEDs (LED1) for RZ/T2L

3.3 Configuring the Firmware Update System for RZ/N2L

Table 3.5 shows the environment required for configuring the RZ/N2L RSK+.

Table 3.6 lists the user application program written during device setup for RZ/N2L.

Table 3.5 Setup Environment for RZ/N2L RSK+

Name	Remarks
Renesas Starter Kit+ for RZ/N2L	RZ/N2L RSK+
USB cables	1 (Mini-B, type-A)
	1 (Type-C, type-A)
Windows host PC	IAR Embedded Workbench or e^2 studio installed
parameter_generator.py Note	Generation tool for the parameter for the loader and the parameter for the user application program
device_setup.py Note	Command sending tool for device setup
RZN2L_RSK_DeviceSetup_qspi.out.srec Note	S-Record format device setup sample program

Note Included in the sample program package for *RZ/T2, RZ/N2 Device Setup Guide for Flash boot (R01AN6471EJ****)*.

Table 3.6 User Application Program Set Up for RZ/N2L

File Name	Description
RZN2L_bsp_LED_0.bin	A user application program based on the Blinky sample application included in the Flexible Software Package RZ/N2L pack. See section 3.8 for detailed creation instructions. The program is assigned from 0x00000000 in the ATCM area. The following LEDs will light up at startup. User LED0: BSP_IO_PORT_18_PIN_2
parameter_RZN2L_bsp_LED_0.bin	Parameter for the user application program (RZN2L_bsp_LED_0.bin). This file is created after building RZN2L_bsp_LED_0.bin. See section 3.8 for detailed creation instructions. The parameter file sets the following xSPI0 address space. External flash address where the program is stored: 0x60030030 RAM address where the program is loaded: 0x00000000

3.3.1 Concatenate Program and Parameter

Use fwupdate_utility.py to concatenate the Update program, SSBL including the parameter for the loader, and the parameter for the user application program. The concatenated data is programmed at the beginning of the external flash. See Table 5.13 for memory maps of the concatenated data.

Open a command prompt on the host PC and run the following command.

The following command will generate initial_image_RZN2L_xspi0.bin:

```
python fwupdate_utility.py setupfile --param_loader RZN2L_RSK_SSBL_xspi0.bin
--param_cpu_core_0 parameter_RZN2L_bsp_LED_0.bin --mpu rzn2l --update_prog
RZN2L_RSK_FWUpdate.bin -o initial_image_RZN2L_xspi0.bin
```

3.3.2 Program to Flash

Programming the update program and user application program are accomplished using device setup tool and sample program. Refer to chapter 2 in *RZ/T2, RZ/N2 Device Setup Guide for Flash boot (R01AN6471EJ****)* for the procedure.

1. Write program files to external flash on RSK+ using device_setup.py. Open a command prompt on the host PC and run the following command.

   The programmable area changes depending on the set boot mode. Please check 5.3.5 for boot modes and corresponding memory maps.
   The initial image file (initial_image_RZ*_xspi0.bin) is programmed at the top of the external. The user application (RZ*_bsp_LED_*.bin) is programmed with the flash address specified during parameter generation.

   RZ/N2L:

```
> python device_setup.py writeflash --port COM4 --addr 60000000 -i
initial_image_RZN2L_xspi0.bin
writeflash : Setup success.
> python device_setup.py writeflash --port COM4 --addr 60030030 -i
RZN2L_bsp_LED_0.bin
writeflash : Setup success.
```

2. Set the SW on the RSK+ board to the following, User DIP Switch SW3-1 to OFF and press the reset button S3.
   After that, to confirm that the user application program starts and User LED0 on the board blinks.

   RZ/N2L:

SW	Setting	Description
SW4.1	ON	xSPI0 boot mode (x1 boot Serial flash)
SW4.2	ON	
SW4.3	ON	
SW4.4	ON	JTAG Authentication by Hash is disabled.

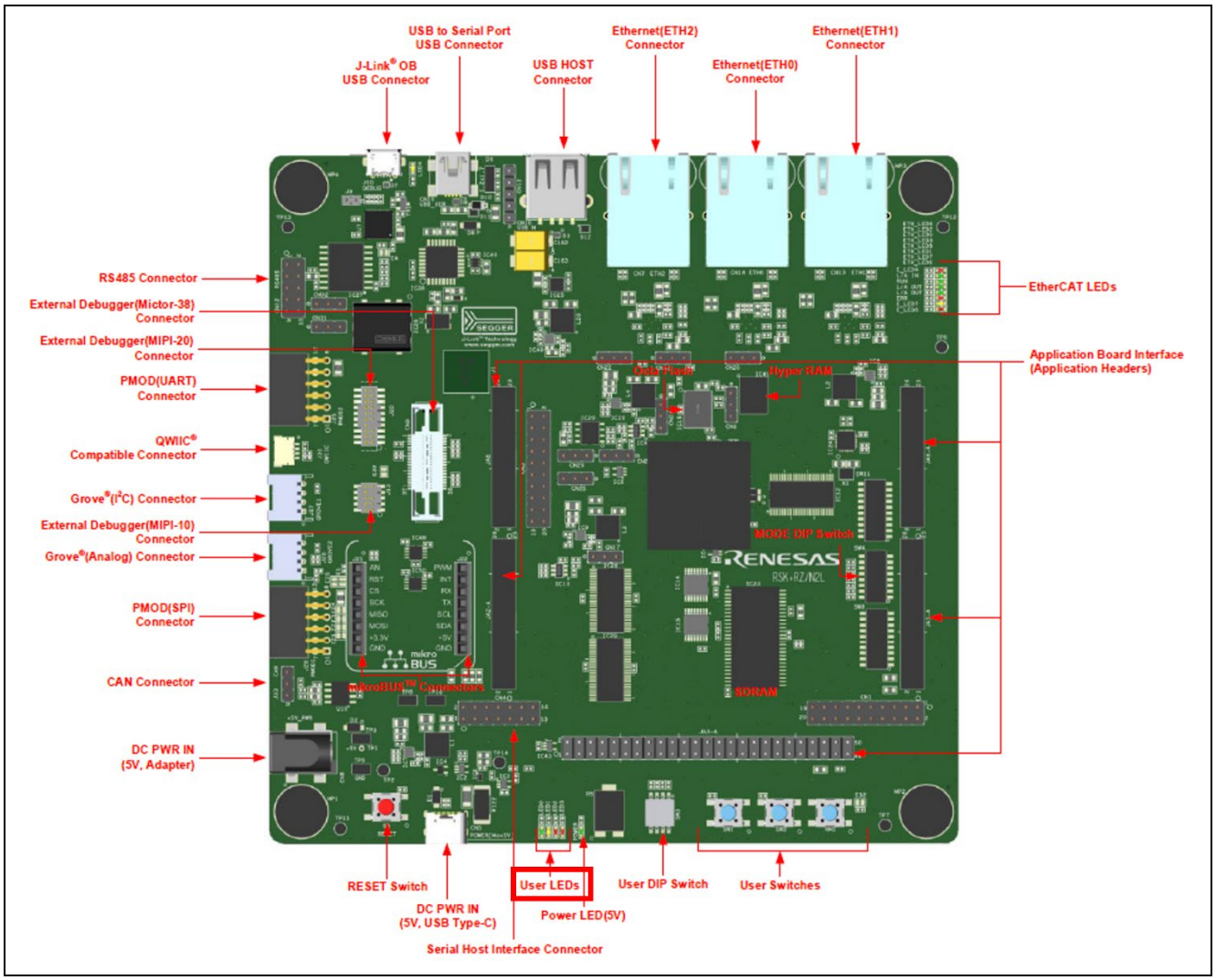


Figure 3.3 Location of User LEDs (LED0) for RZ/N2L

3.4 Configuring the Firmware Update System for RZ/T2H

Table 3.7 shows the environment required for configuring the RZ/T2H EVB. Table 3.8 lists the user application program written during device setup for RZ/T2H Cortex®-R52. Table 3.9 lists the user application program written during device setup for RZ/T2H Cortex®-A55.

Table 3.7 Setup Environment for RZ/T2H EVB

Name	Remarks
RZ/T2H Evaluation Board	RZ/T2H EVB
USB cables	1 (Mini-B, type-A)
	1 (Type-C, type-A)
Windows host PC	IAR Embedded Workbench or e^2 studio installed
parameter_generator.py Note	Generation tool for the parameter for the loader and the parameter for the user application program
device_setup.py Note	Command sending tool for device setup
RZT2H_EVB_DeviceSetup.out.srec Note	S-Record format device setup sample program

Note Included in the sample program package for *RZ/T2, RZ/N2 Device Setup Guide for Flash boot (R01AN6471EJ****)*.

Table 3.8 User Application Program Set Up for RZ/T2H Cortex®-R52

File Name	Description
RZT2H_bsp_LED_0.bin	A user application program based on the Blinky sample application included in the Flexible Software Package RZ/T2H pack. See section 3.8 for detailed creation instructions. The program is assigned from 0x00000000 in the ATCM area. The following LEDs will light up at startup. User LED0: BSP_IO_PORT_23_PIN_1
parameter_RZT2H_bsp_LED_0.bin	Parameter for the user application program (RZT2H_bsp_LED_0.bin). This file is created after building RZT2H_bsp_LED_0.bin. See section 3.8 for detailed creation instructions. The parameter file sets the following xSPI0 address space. External flash address where the program is stored: 0x40094050 RAM address where the program is loaded: 0x00000000

Table 3.9 User Application Program Set Up for RZ/T2H Cortex®-A55

File Name	Description
RZT2H_bsp_LED_0.bin	A user application program based on the Blinky sample application included in the Flexible Software Package RZ/T2H pack. See section 3.8 for detailed creation instructions. The program is assigned from 0x10050000 in the SystemRAM area. The following LEDs will light up at startup. User LED0: BSP_IO_PORT_23_PIN_1
parameter_RZT2H_bsp_LED_0.bin	Parameter for the user application program (RZT2H_bsp_LED_0.bin). This file is created after building RZT2H_bsp_LED_0.bin. See section 3.8 for detailed creation instructions. The parameter file sets the following xSPI0 address space. External flash address where the program is stored: 0x40094050 RAM address where the program is loaded: 0x10050000

3.4.1 Concatenate Program and Parameter

Use fwupdate_utility.py to concatenate the Update program, SSBL including the parameter for the loader, and the parameter for the user application program. The concatenated data is programmed at the beginning of the external flash. See Table 5.18 for memory maps of the concatenated data.

Open a command prompt on the host PC and run the following command.

RZ/T2H Cortex®-R52:

[Only for multi-core configuration] You must specify both the user application parameters for CPU0 (--param_cpu_core_0) and the user application parameters for CPU1 (--param_cpu_core_1).

The following command will generate initial_image_RZT2H_xspi0.bin:

```
python fwupdate_utility.py setupfile --param_loader RZT2H_EVB_SSBL_xspi0.bin
--param_cpu_core_0 parameter_RZT2H_bsp_LED_0.bin --mpu rzt2h --update_prog
RZT2H_EVB_FWUpdate.bin -o initial_image_RZT2H_xspi0.bin
```

RZ/T2H Cortex®-A55:

[Only for multi-core configuration] You must specify either the user application parameters for Core 1 (--param_cpu_core_1), Core 2 (--param_cpu_core_2) or Core 3 (--param_cpu_core_3) along with the user application parameter for Core 0 (--param_cpu_core_0).

The following command will generate initial_image_RZT2H_xspi0.bin:

```
python fwupdate_utility.py setupfile --param_loader
RZT2H_EVB_CA55_SSBL_xspi0.bin --param_cpu_core_0
parameter_RZT2H_bsp_LED_0.bin --mpu rzt2h --update_prog
RZT2H_EVB_CA55_FWUpdate.bin -o initial_image_RZT2H_xspi0.bin
```

3.4.2 Program to Flash

Programming the update program and user application program are accomplished using device setup tool and sample program. Refer to chapter 2 in *RZ/T2, RZ/N2 Device Setup Guide for Flash boot* for the procedure.

1. Write program files to external flash on EVB using device_setup.py. Open a command prompt on the host PC and run the following command.

   The programmable area changes depending on the set boot mode. Please check 5.3.6 for boot modes and corresponding memory maps.
   The initial image file (initial_image_RZ*_xspi0.bin) is programmed at the top of the external. The user application (RZ*_bsp_LED_*.bin) is programmed with the flash address specified during parameter generation.

   RZ/T2H Cortex®-R52 or Cortex®-A55:

```
> python device_setup.py writeflash --port COM4 --addr 40000000 -i
initial_image_RZT2H_xspi0.bin
writeflash : Setup success.
> python device_setup.py writeflash --port COM4 --addr 40094050 -i
RZT2H_bsp_LED_0.bin
writeflash : Setup success.
```

2. Set the SW on the EVB to the following, User DIP Switch SW12-1 to OFF and press the reset button SW13.
   After that, to confirm that the user application program starts and User LED0 on the board blinks.

RZ/T2H:

SW	Setting	Description
SW14.1	ON	xSPI0 boot mode (x1 boot Serial flash)
SW14.2	ON	
SW14.3	ON	
SW14.4	OFF	Cortex®-R52 CPU0 ATCM 1 wait
SW14.5	OFF	Cortex®-R52 CPU1 ATCM 1 wait
SW14.6	OFF	Supply voltage of boot peripheral is 3.3 V
SW14.7	ON	JTAG Authentication by Hash is disabled.

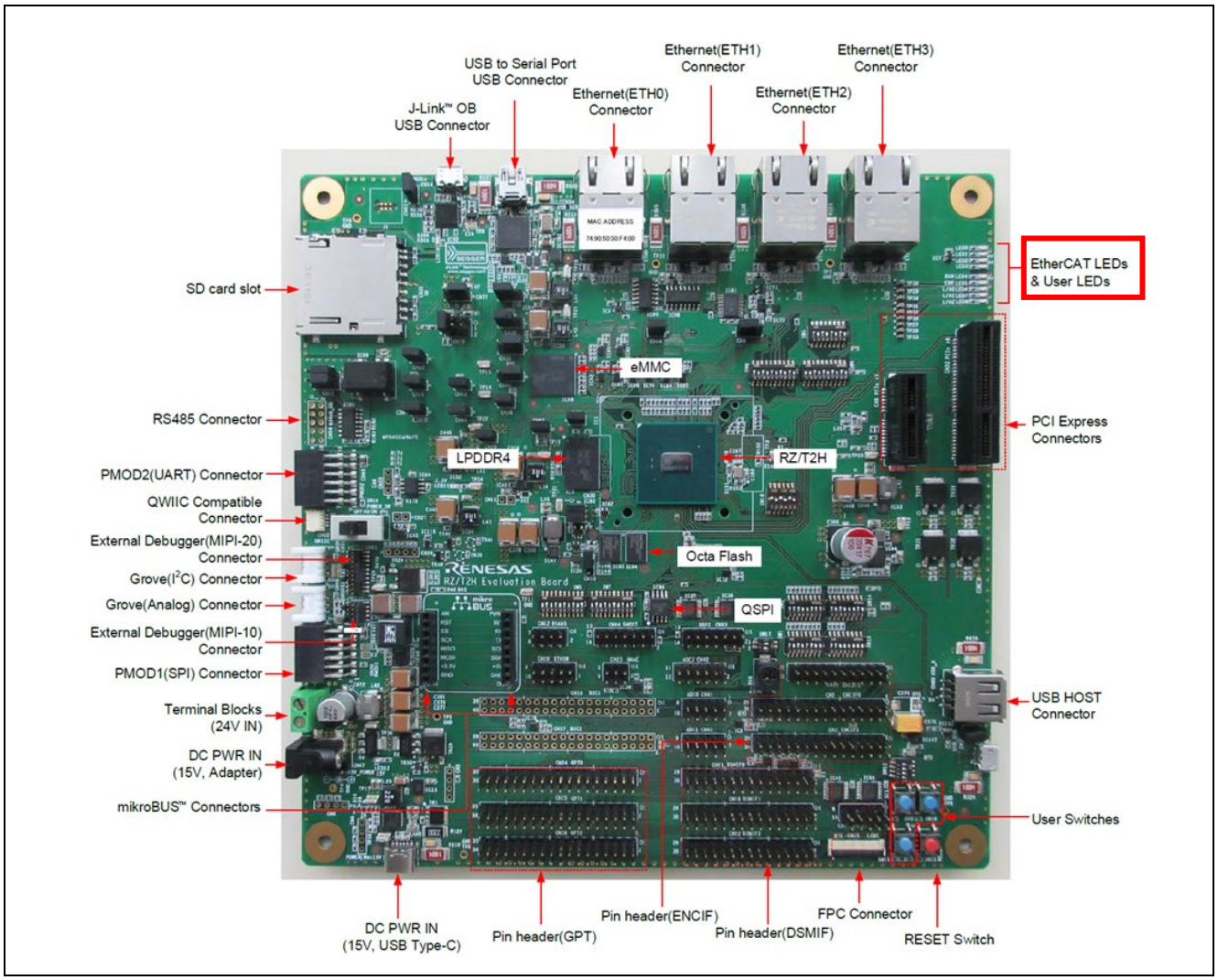


Figure 3.4 Location of User LEDs (LED0) for RZ/T2H

3.5 Configuring the Firmware Update System for RZ/N2H

Table 3.10 shows the environment required for configuring the RZ/N2H EVB. Table 3.11 lists the user application program written during device setup for RZ/N2H Cortex®-R52. Table 3.12 lists the user application program written during device setup for RZ/N2H Cortex®-A55.

Table 3.10 Setup Environment for RZ/N2H EVB

Name	Remarks
RZ/N2H Evaluation Board	RZ/N2H EVB
USB cables	1 (Mini-B, type-A)
	1 (Type-C, type-A)
Windows host PC	IAR Embedded Workbench or e^2 studio installed
parameter_generator.py Note	Generation tool for the parameter for the loader and the parameter for the user application program
device_setup.py Note	Command sending tool for device setup
RZN2H_EVB_DeviceSetup.out.srec Note	S-Record format device setup sample program

Note Included in the sample program package for *RZ/T2, RZ/N2 Device Setup Guide for Flash boot (R01AN6471EJ****)*.

Table 3.11 User Application Program Set Up for RZ/N2H Cortex®-R52

File Name	Description
RZN2H_bsp_LED_3.bin	A user application program based on the Blinky sample application included in the Flexible Software Package RZ/N2H pack. See section 3.8 for detailed creation instructions. The program is assigned from 0x00000000 in the ATCM area. The following LEDs will light up at startup. User LED3: BSP_IO_PORT_31_PIN_6
parameter_RZN2H_bsp_LED_3.bin	Parameter for the user application program (RZN2H_bsp_LED_3.bin). This file is created after building RZN2H_bsp_LED_3.bin. See section 3.8 for detailed creation instructions. The parameter file sets the following xSPI0 address space. External flash address where the program is stored: 0x40094050 RAM address where the program is loaded: 0x00000000

Table 3.12 User Application Program Set Up for RZ/N2H Cortex®-A55

File Name	Description
RZN2H_bsp_LED_3.bin	A user application program based on the Blinky sample application included in the Flexible Software Package RZ/N2H pack. See section 3.8 for detailed creation instructions. The program is assigned from 0x10050000 in the SystemRAM area. The following LEDs will light up at startup. User LED3: BSP_IO_PORT_31_PIN_6
parameter_RZN2H_bsp_LED_3.bin	Parameter for the user application program (RZN2H_bsp_LED_3.bin). This file is created after building RZN2H_bsp_LED_3.bin. See section 3.8 for detailed creation instructions. The parameter file sets the following xSPI0 address space. External flash address where the program is stored: 0x40094050 RAM address where the program is loaded: 0x10050000

3.5.1 Concatenate Program and Parameter

Use fwupdate_utility.py to concatenate the Update program, SSBL including the parameter for the loader, and the parameter for the user application program. The concatenated data is programmed at the beginning of the external flash. See Table 5.18 for memory maps of the concatenated data.

Open a command prompt on the host PC and run the following command.

RZ/N2H Cortex®-R52:

[Only for multi-core configuration] You must specify both the user application parameters for CPU0 (--param_cpu_core0) and the user application parameters for CPU1 (--param_cpu_core_1).

The following command will generate initial_image_RZN2H_xspi0.bin:

```
python fwupdate_utility.py setupfile --param_loader RZN2H_EVB_SSBL_xspi0.bin
--param_cpu_core_0 parameter_RZN2H_bsp_LED_3.bin --mpu rzn2h --update_prog
RZN2H_EVB_FWUpdate.bin -o initial_image_RZN2H_xspi0.bin
```

RZ/N2H Cortex®-A55:

[Only for multi-core configuration] You must specify either the user application parameters for Core 1 (--param_cpu_core_1), Core 2 (--param_cpu_core_2) or Core 3 (--param_cpu_core_3) along with the user application parameter for Core 0 (--param_cpu_core_0).

The following command will generate initial_image_RZN2H_xspi0.bin:

```
python fwupdate_utility.py setupfile --param_loader
RZN2H_EVB_CA55_SSBL_xspi0.bin --param_cpu_core_0
parameter_RZN2H_bsp_LED_3.bin --mpu rzn2h --update_prog
RZN2H_EVB_CA55_FWUpdate.bin -o initial_image_RZN2H_xspi0.bin
```

3.5.2 Program to Flash

Programming the update program and user application program are accomplished using device setup tool and sample program. Refer to chapter 2 in *RZ/T2, RZ/N2 Device Setup Guide for Flash boot* for the procedure.

1. Write program files to external flash on EVB using device_setup.py. Open a command prompt on the host PC and run the following command.

   The programmable area changes depending on the set boot mode. Please check 5.3.6 for boot modes and corresponding memory maps.
   The initial image file (initial_image_RZ*_xspi0.bin) is programmed at the top of the external. The user application (RZ*_bsp_LED_*.bin) is programmed with the flash address specified during parameter generation.

   RZ/N2H Cortex®-R52 or Cortex®-A55:

```
> python device_setup.py writeflash --port COM4 --addr 40000000 -i
initial_image_RZN2H_xspi0.bin
writeflash : Setup success.
> python device_setup.py writeflash --port COM4 --addr 40094050 -i
RZN2H_bsp_LED_3.bin
writeflash : Setup success.
```

2. Set the SW on the EVB to the following, User DIP Switch DSW1-1 to OFF and press the reset button SW5.
   After that, to confirm that the user application program starts and User LED3 on the board blinks.

RZ/N2H:

SW	Setting	Description
DSW3.1	ON	xSPI0 boot mode (x1 boot Serial flash)
DSW3.2	ON	
DSW3.3	ON	
DSW3.4	OFF	Cortex®-R52 CPU0 ATCM 1 wait
DSW3.5	OFF	Cortex®-R52 CPU1 ATCM 1 wait
DSW3.6	OFF	Supply voltage of boot peripheral is 3.3 V
DSW3.7	ON	JTAG Authentication by Hash is disabled.

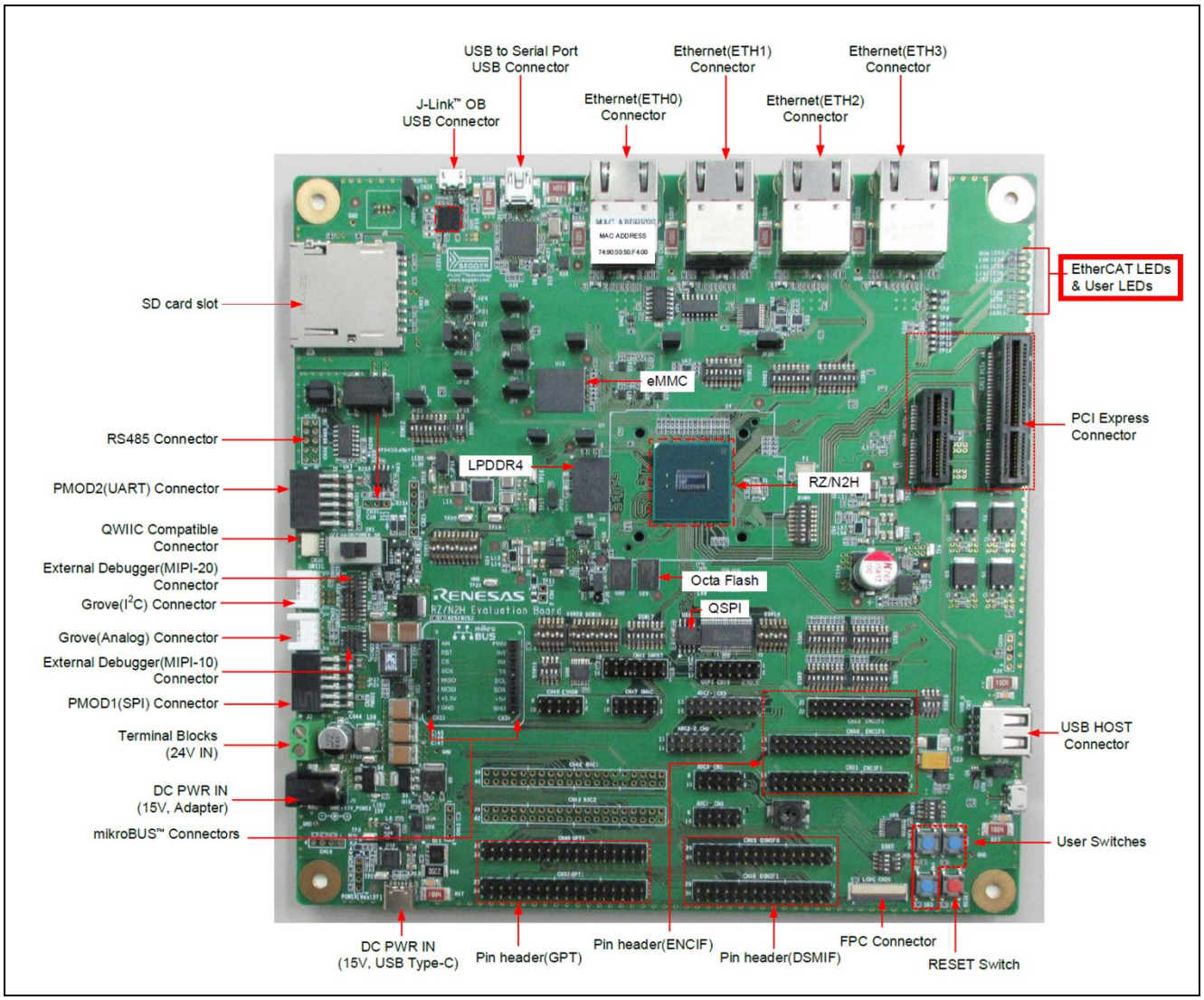


Figure 3.5 Location of User LEDs (LED3) for RZ/N2H

3.6 Update Program Configuration

IAR Embedded Workbench for ARM is used as the development environment of the update program.

Open the following workspace, select the update program project (RZ*_*_FWUpdate), and build the project.

RZ*_*_FWUpdate_Rev*\RZ*_*_FWUpdate.eww

In the default configuration, the update program is for xSPI0 boot mode and flash one plane. The configurations of the update program can be changed with the following files.

RZ*_*_FWUpdate_Rev*\src\fwupdate_cfg.h

For RZ/N2L only, if you use 16-bit bus boot mode, you also need to change the sample program project settings by referring to Section 5.3.7.

The configuration of the update program is shown in Table 3.13.

Table 3.13 Configurations for update program

Configuration items	Configurable values	Description
FWUPDATE_CFG_BOOT_MODE_SELECT	BOOT_MODE_XSPI0	Default settings. Specified when using in xSPI0 boot mode (x1 boot serial flash).
	BOOT_MODE_XSPI1	Specified when using in xSPI1 boot mode (x1 boot serial flash).
	BOOT_MODE_NOR	Specified when using in 16-bit bus boot mode (NOR flash). Not supported on RZ/T2H and RZ/N2H.
FWUPDATE_CFG_FLASH_MNG_AREA	FLASH_MNG_AREA_SINGLE_BANK	Default settings. Specify when using the user application program area on the external flash for one plane management.
	FLASH_MNG_AREA_DUAL_BANK	Specify this when using the user application program area of the external flash for two plane management.

3.7 SSBL Configuration

IAR Embedded Workbench for ARM is used as the development environment of SSBL.

Open the following workspace, select the SSBL project (RZ*_*_SSBL), and build the project.

RZ*_*_SSBL_Rev*\RZ*_*_SSBL.eww

In the default configuration, the update program is for xSPI0 boot mode, single core, and flash one plane. The configurations of the SSBL can be changed with the following files.

RZ*_*_SSBL_Rev*\src\ssbl_cfg.h

For RZ/N2L only, if you use 16-bit bus boot mode, you also need to change the sample program project settings by referring to Section 5.3.7.

[Only for multi-core configuration] If you want to run the user application on Cortex®-R52 CPU1 or Cortex®-A55 Core 1, Core 2, and Core 3, you will need SSBL that has been changed to multi-core settings.

The configuration of SSBL is shown in Table 3.14.

Table 3.14 Configurations for SSBL

Configuration items	Configurable values	Description
SSBL_CFG_BOOT_MODE_SELECT	BOOT_MODE_XSPI0	Default settings. Specified when using in xSPI0 boot mode (x1 boot serial flash).
	BOOT_MODE_XSPI1	Specified when using in xSPI1 boot mode (x1 boot serial flash).
	BOOT_MODE_NOR	Specified when using in 16-bit bus boot mode (NOR flash). Not supported on RZ/T2H and RZ/N2H.
SSBL_CFG_OPERATING_CORE_MODE	SINGLE_CORE	Default settings. Single-core configuration. Specify when using as a single core.
	MULTI_CORE	Multi-core configuration. Specify when using as Cortex®-R52 dual core or Cortex®-A55 multi core.
SSBL_CFG_MULTI_CORE_CA55_CORE_1 Note	DISABLE	Default settings. Disables user application launch on Cortex®-A55 Core 1.
	ENABLE	Enables user application launch on Cortex®-A55 Core 1.
SSBL_CFG_MULTI_CORE_CA55_CORE_2 Note	DISABLE	Default settings. Disables user application launch on Cortex®-A55 Core 2.
	ENABLE	Enables user application launch on Cortex®-A55 Core 2.
SSBL_CFG_MULTI_CORE_CA55_CORE_3 Note	DISABLE	Default settings. Disables user application launch on Cortex®-A55 Core 3.
	ENABLE	Enables user application launch on Cortex®-A55 Core 3.
SSBL_CFG_FLASH_MNG_AREA	FLASH_MNG_AREA_SINGLE_BANK	Default settings. Specify when using the user application program area on the external flash for one plane management.
	FLASH_MNG_AREA_DUAL_BANK	Specify this when using the user application program area of the external flash for two plane management.

Note This config is used when SSBL_CFG_OPERATING_CORE_MODE is MULTI_CORE and the operating core is Cortex®-A55 core.

After building SSBL (RZ*_RSK_SSBL.bin), generate parameter for the loader for SSBL (hereinafter referred to as parameter for SSBL).

The SSBL program size (binary data size) must be a multiple of 512 bytes and less than the maximum size shown in Table 3.15. If the binary data size after the user program build is not a multiple of 512 bytes, add dummy data after the binary data to adjust it to a multiple of 512 bytes. For information on how to adjust the binary data to 512-byte units by adding project settings, refer to Section 3.3.7 in *RZ/T2, RZ/N2 Device Setup Guide for Flash boot (R01AN6471EJ****)*.

Parameter for SSBL is generated using the tool parameter_generator.py, which is included in the sample program package for the *RZ/T2, RZ/N2 Device Setup Guide for Flash boot (R01AN6471EJ****)*.

In section 3.7.1 to 3.7.6, the procedure for generating data that concatenates the parameter for SSBL and SSBL program is explained for each MCU type.

Table 3.15 Maximum user program size

Products	First booting CPU core	Maximum user program size
RZ/T2M	Cortex®-R52	56 KB
RZ/T2L	Cortex®-R52	56 KB
RZ/N2L	Cortex®-R52	120 KB
RZ/T2H and RZ/N2H	Cortex®-R52	52 KB
	Cortex®-A55	2036 KB

3.7.1 Concatenate SSBL and Parameter for RZ/T2M

The following shows an example of tool execution when xSPI0 address space flash is specified:

- External flash address where the program is stored (--src_addr): 0x60000050
- RAM address where the program is loaded (--dest_addr): 0x00102000

The following command generates RZT2M_RSK_SSBL_xspi0.bin in which the parameter for the loader and the SSBL program are concatenated:

```
python parameter_generator.py loader --mpu rzt2m --target_cpu cr52 --mode
xspi0 --src_addr 60000050 --dest_addr 00102000 -i RZT2M_RSK_SSBL.bin -o
RZT2M_RSK_SSBL_xspi0.bin --concat_loader
```

3.7.2 Concatenate SSBL and Parameter for RZ/T2L

The following shows an example of tool execution when xSPI0 address space flash is specified:

- External flash address where the program is stored (--src_addr): 0x60000050
- RAM address where the program is loaded (--dest_addr): 0x00102000

The following command generates RZT2L_RSK_SSBL_xspi0.bin in which the parameter for the loader and the SSBL program are concatenated:

```
python parameter_generator.py loader --mpu rzt2l --target_cpu cr52 --mode
xspi0 --src_addr 60000050 --dest_addr 00102000 -i RZT2L_RSK_SSBL.bin -o
RZT2L_RSK_SSBL_xspi0.bin --concat_loader
```

3.7.3 Concatenate SSBL and Parameter for RZ/N2L

The following shows an example of tool execution when xSPI0 address space flash is specified:

- External flash address where the program is stored (--src_addr): 0x60000050
- RAM address where the program is loaded (--dest_addr): 0x00102000

The following command generates RZN2L_RSK_SSBL_xspi0.bin in which the parameter for the loader and the SSBL program are concatenated:

```
python parameter_generator.py loader --mpu rzn2l --target_cpu cr52 --mode
xspi0 --src_addr 60000050 --dest_addr 00102000 -i RZN2L_RSK_SSBL.bin -o
RZN2L_RSK_SSBL_xspi0.bin --concat_loader
```

3.7.4 Concatenate SSBL and Parameter for RZ/T2H Cortex®-R52

The following shows an example of tool execution when xSPI0 address space flash is specified:

- External flash address where the program is stored (--src_addr): 0x40000050
- RAM address where the program is loaded (--dest_addr): 0x00102000

The following command generates RZT2H_EVB_SSBL_xspi0.bin in which the parameter for the loader and the SSBL program are concatenated:

```
python parameter_generator.py loader --mpu rzt2h --target_cpu cr52 --mode
xspi0 --src_addr 40000050 --dest_addr 00102000 -i RZT2H_EVB_SSBL.bin -o
RZT2H_EVB_SSBL_xspi0.bin --concat_loader
```

3.7.5 Concatenate SSBL and Parameter for RZ/T2H Cortex®-A55

The following shows an example of tool execution when xSPI0 address space flash is specified:

- External flash address where the program is stored (--src_addr): 0x40000050
- RAM address where the program is loaded (--dest_addr): 0x10000000

The following command generates RZT2H_EVB_SSBL_xspi0.bin in which the parameter for the loader and the SSBL program are concatenated:

```
python parameter_generator.py loader --mpu rzt2h --target_cpu ca55 --mode
xspi0 --src_addr 40000050 --dest_addr 10000000 -i RZT2H_EVB_CA55_SSBL.bin -o
RZT2H_EVB_SSBL_xspi0.bin --concat_loader
```

3.7.6 Concatenate SSBL and Parameter for RZ/N2H Cortex®-R52

The following shows an example of tool execution when xSPI0 address space flash is specified:

- External flash address where the program is stored (--src_addr): 0x40000050
- RAM address where the program is loaded (--dest_addr): 0x00102000

The following command generates RZN2H_EVB_SSBL_xspi0.bin in which the parameter for the loader and the SSBL program are concatenated:

```
python parameter_generator.py loader --mpu rzn2h --target_cpu cr52 --mode
xspi0 --src_addr 40000050 --dest_addr 00102000 -i RZN2H_EVB_SSBL.bin -o
RZN2H_EVB_SSBL_xspi0.bin --concat_loader
```

3.7.7 Concatenate SSBL and Parameter for RZ/N2H Cortex®-A55

The following shows an example of tool execution when xSPI0 address space flash is specified:

- External flash address where the program is stored (--src_addr): 0x40000050
- RAM address where the program is loaded (--dest_addr): 0x10000000

The following command generates RZN2H_EVB_SSBL_xspi0.bin in which the parameter for the loader and the SSBL program are concatenated:

```
python parameter_generator.py loader --mpu rzn2h --target_cpu ca55 --mode
xspi0 --src_addr 40000050 --dest_addr 10000000 -i RZN2H_EVB_CA55_SSBL.bin -o
RZN2H_EVB_SSBL_xspi0.bin --concat_loader
```

3.8 User Application Program Configuration

Create a user application program based on the Blinky sample application included in the Flexible Software Package RZ/T2M, RZ/T2L, RZ/T2H, RZ/N2L or RZ/N2H pack. For more information about FSP, please refer to RZ/T2, RZ/N2 Getting Started with Flexible Software Package.

Also refer to the sample project for the user application program included in this package.

Sample project for CPU0: RZ*_bsp_LED_x.zip

[Only for multi-core configuration]

Sample project for Cortex®-R52 CPU1: RZ*_bsp_LED_x_CPU1.zip

Sample project for Cortex®-A55 Core*: RZ*_bsp_LED_x_CA55_Core*.zip

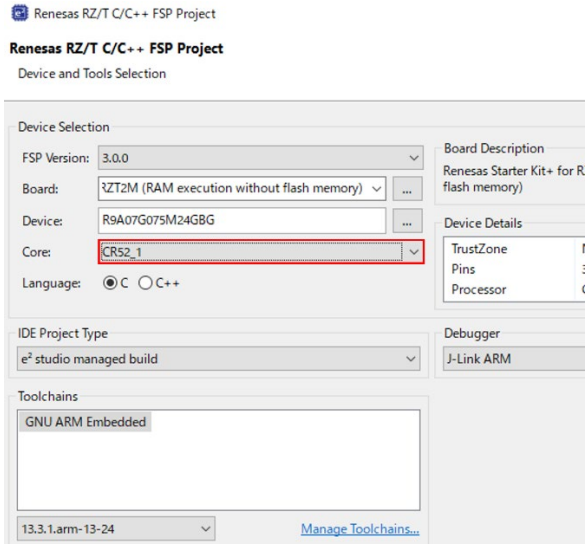
When using a sample project for Cortex®-R52 CPU1, be sure to unzip it to the same workspace or folder as the sample project for Cortex®-R52 CPU0.

When using a sample project for Cortex®-A55 Core 1, Core 2, or Core 3, be sure to unzip it to the same workspace or folder as the sample project for Cortex®-A55 Core 0.

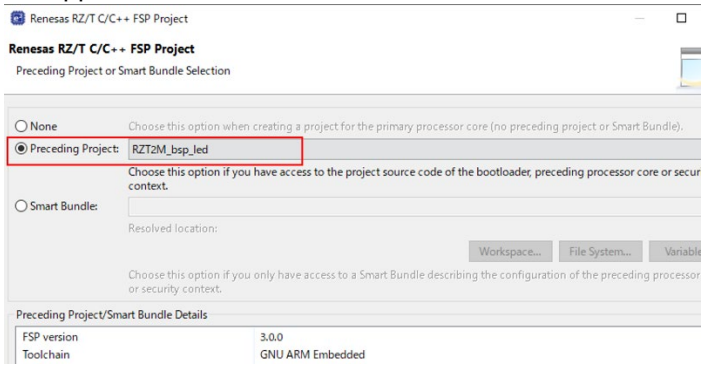
1. Create Blinky sample application.
   See section 5.2 “Tutorial Blinky” in RZ/T2, RZ/N2 Getting Started with Flexible Software Package for the creation instructions.
   Project name example: RZT2M_bsp_LED_0
   The project name is an example for RZ/T2M. For RZ/T2L, RZ/T2H, RZ/N2L and RZ/N2H, read its MPU name instead.
   Replace the LED number with the one you actually use. In the sample, LED0 is used for RZ/T2M, RZ/N2L and RZ/T2H, LED1 is used for RZ/T2L, and LED3 is used for RZ/N2H.

   [Only for multi-core configuration] For user application programs for Cortex®-R52 CPU1 or Cortex®-A55 Core 1, Core 2, or Core 3, please note the following settings.

- Select Cortex®-R52 CPU1 or Cortex®-A55 Core 1, Core 2, or Core 3 by specifying "Core" when creating the project.



- An application for Cortex®-R52 CPU0 or Cortex®-A55 Core 0 is required. In the same workspace, select the application created earlier for Cortex®-R52 CPU0 or Cortex®-A55 Core 0.



2. Open the memory allocation configuration file and change the memory assignment.
   The area that can be used by application programs varies depending on the device. Table 3.16 shows the available areas.

Table 3.16 Operating area of user application program

Device	Core	Memory (address range)
RZ/T2M	CPU0	ATCM (0x00000000 – 0x0007FFFF)
	CPU1	SystemRAM (0x10000000 – 0x101FFFFF)
RZ/T2L	CPU0	ATCM (0x00000000 – 0x0007FFFF)
RZ/N2L	CPU0	ATCM (0x00000000 – 0x0001FFFF)
RZ/T2H Cortex®-R52, RZ/N2H Cortex®-R52	CPU0	ATCM (0x00000000 – 0x0007FFFF)
	CPU1	SystemRAM (0x10000000 – 0x101FFFFF)
RZ/T2H Cortex®-A55, RZ/N2H Cortex®-A55	Core 0, Core 1, Core 2, Core 3	SystemRAM (0x10050000 – 0x101FFFFF) DDR mirror (0xC0000000 – 0xFFFFFFFF)

A modified example for EWARM and e^2 studio of placing the start address of the user application program at 0x00000000 in the ATCM is shown below.

Change the following files in the user application project.
EWARM: RZT2M_bsp_LED_0\script\fsp_ram_execution.icf
e^2 studio: RZT2M_bsp_LED_0\script\fsp_ram_execution.ld
This is the file path when the project name is for RZT2M. For RZ/T2L, RZ/N2L, RZ/T2H and RZ/N2H, read its MPU name instead.

[Only for multi-core configuration] For the user application program for Cortex®-R52 CPU1 or Cortex®-A55 Core 1, Core 2, or Core 3, it is located from SystemRAM or DDR mirror. Please replace the address ranges below with values that correspond to SystemRAM or DDR mirror and check the settings.

2-1. Change the program entry address to 0x00000000.Please note that there is a difference in the number of lines to be modified between RZ/T2M, RZ/T2L and RZ/T2H. The only difference is the number of lines, the addresses and area specifications that need to be changed are the same.

fsp_ram_execution.icf lines 40 before modifying:

```
define symbol INTVEC_ADDRESS                = ATCM_PRV_START;
```

fsp_ram_execution.icf lines 40 after modifying:

```
define symbol INTVEC_ADDRESS                = 0x2E00;
```

fsp_ram_execution.icf lines 552-569 before modifying:

```
place at start of D_LOADER_STACK_region      { block LDR_PRG_RBLOCK };
place in D_LOADER_STACK_region               { block LDR_DATA_WBLOCK };
place in D_LOADER_STACK_region               { block LDR_DATA_ZBLOCK,
                                             block SOFTWARE_RESET_ZBLOCK,
                                             block LOADER_STACK };
place in D_LOADER_STACK_region               { last block BTCM_END_block };

place at start of RAM_region                 { block USER_PRG_RBLOCK };
place in RAM_region                          { readonly,
                                             readwrite };
place in RAM_region                          { block USER_DATA_WBLOCK };
place in RAM_region                          { block USER_DATA_ZBLOCK,
                                             block NOINIT_BLOCK,
                                             block HEAP_BLOCK,
                                             block AARCH64_STACK,
                                             block THREAD_STACK };
place in RAM_region                          { last block ATCM_END_block };
place in RAM_region                          { last block SYSTEM_RAM_END_block };
```

fsp_ram_execution.icf lines 552-569 after modifying:

```
place at start of ATCM_region                { block LDR_PRG_RBLOCK };
place in ATCM_region                         { block LDR_DATA_WBLOCK };
place in ATCM_region                         { block LDR_DATA_ZBLOCK,
                                             block SOFTWARE_RESET_ZBLOCK,
                                             block LOADER_STACK };
place in D_LOADER_STACK_region               { last block BTCM_END_block };

place in ATCM_region                         { block USER_PRG_RBLOCK };
place in ATCM_region                         { readonly,
                                             readwrite };
place in ATCM_region                         { block USER_DATA_WBLOCK };
place in ATCM_region                         { block USER_DATA_ZBLOCK,
                                             block NOINIT_BLOCK,
                                             block HEAP_BLOCK,
                                             block AARCH64_STACK,
                                             block THREAD_STACK };
place in ATCM_region                         { last block ATCM_END_block };
place in RAM_region                          { last block SYSTEM_RAM_END_block };
```

fsp_ram_execution.ld lines 124-138 before modifying:

```
        /* Place the functions to be used in the loader program (SSBL). */
    .loader_text LOADER_TEXT_ADDRESS :
    {
      _loader_text_start = .;
      *(.loader_text)
      */startup_core.o(.text*)
      */system_core.o(.text*)
      */startup.o(.text*)
      */system.o(.text*)
      */all*/bsp_*.o(.text*)
      */r_ioport.o(.text*)
      KEEP(*(.eh_frame*))
      KEEP(*(.warm_start))
      _loader_text_end = .;
    } > LOADER_TEXT_STACK
```

fsp_ram_execution.ld lines 124-138 After modifying:

```
        /* Place the functions to be used in the loader program (SSBL). */
    .loader_text 0x00000000 :
    {
      _loader_text_start = .;
      *(.loader_text)
      */startup_core.o(.text*)
      */system_core.o(.text*)
      */startup.o(.text*)
      */system.o(.text*)
      */all*/bsp_*.o(.text*)
      */r_ioport.o(.text*)
      KEEP(*(.eh_frame*))
      KEEP(*(.warm_start))
      _loader_text_end = .;
    } > ATCM
```

If you place your program in the ATCM, you must make the following changes to avoid build errors.

fsp_ram_execution.ld lines 481 before modifying:

```
    __ddsc_BTCM_CR52_0_END      = DEFINED(BTCM_CR52_0_START) ? __AbtStackLimit : __ddsc_BTCM_CR52_0_START;
```

fsp_ram_execution.ld lines 481 after modifying:

```
    __ddsc_BTCM_CR52_0_END      = DEFINED(BTCM_CR52_0_START) ? __ddsc_BTCM_CR52_0_START :
    __ddsc_BTCM_CR52_0_START;
```

2-2. Place program code and static variables with initial values in ATCM. Place uninitialized variables and heap areas in ATCM or other space.
Please refer to the following files for setting examples.
Modifications may be required depending on the processing and size of the user application.
EWARM: RZ*_bsp_LED_x\script\fsp_ram_execution.icf
e^2 studio: RZ*_bsp_LED_x\script\fsp_ram_execution.ld

[Only for multi-core configuration]
For the user application program for Cortex®-R52 CPU1, please refer to the following file as a setting example.
EWARM: RZ*_bsp_LED_x_CPU1\script\fsp_ram_execution.icf
e^2 studio: RZ*_bsp_LED_x_CPU1\script\fsp_ram_execution.ld

For the user application program for Cortex®-A55 Core 1, Core 2, or Core 3, please refer to the following file as a setting example.
EWARM: RZ*_bsp_LED_x_CA55_Core*\script\fsp_ram_execution.icf
e^2 studio: RZ*_bsp_LED_x_CA55_Core*\script\fsp_ram_execution.ld

3. Open the following file and modify the initialization process.
RZT2M_bsp_LED_0\rzt\fsp\src\bsp\cmsis\Device\RENESAS\Source\startup_core.c
This is the file path for RZ/T2M. For RZ/T2L, RZ/N2L, RZ/T2H and RZ/N2H read its MPU name instead.

[Only for multi-core configuration] For user application programs for Cortex®-R52 CPU1 or Cortex®-A55 Core 1, Core 2, or Core 3, the changes described here are not required.

3-1. For Cortex®-R52 projects:
Delete the lines that set the background to gray in the following code.
Please note that there is a difference in the number of lines to be deleted between RZ/T2M, RZ/T2L, RZ/T2H, RZ/N2L and RZ/N2H. The only difference is the number of lines, the processing that needs to be deleted is the same.

Delete all lines system_init() in startup_core.c, lines 157-210

```
/*****************************************************************************************************************
*************//**
 * After boot processing, LSI starts executing here.

*****************************************************************************************************************
***************/
BSP_TARGET_ARM BSP_ATTRIBUTE_STACKLESS void system_init (void)
{
    __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");

    __asm volatile (
        "set_hcr:                                       \n"
        "    MRC   p15, #4, r1, c1, c1, #0              \n" /* Read Hyp Configuration Register */
        "    ORR   r1, r1, %[bsp_hcr_hcd_disable]       \n" /* HVC instruction disable */
        "    MCR   p15, #4, r1, c1, c1, #0              \n" /* Write Hyp Configuration Register */
        ::[bsp_hcr_hcd_disable] "i" (BSP_HCR_HCD_DISABLE) : "memory");

    __asm volatile (
        "set_vbar:                                      \n"
        "    LDR   r0, =__Vectors                       \n"
        "    MCR   p15, #0, r0, c12, c0, #0             \n" /* Write r0 to VBAR */
        ::: "memory");

#if (0 == BSP_CFG_CORE_CR52) || (1 == BSP_FEATURE_BSP_HAS_CR52_CPU1_LLPP)
    __asm volatile (
        "LLPP_access_enable:                            \n"

        /* Enable PERIPHPREGIONR (LLPP) */
        "    MRC   p15, #0, r1, c15, c0,#0              \n" /* PERIPHPREGIONR */
        "    ORR   r1, r1, #(0x1 << 1)                  \n" /* Enable PERIPHPREGIONR EL2 */
        "    ORR   r1, r1, #(0x1)                       \n" /* Enable PERIPHPREGIONR EL1 and EL0 */
        "    DSB                                        \n" /* Ensuring memory access complete */
        "    MCR   p15, #0, r1, c15, c0,#0              \n" /* PERIPHREGIONR */
        "    ISB                                        \n" /* Ensuring Context-changing */
        ::: "memory");
#endif

    __asm volatile (
        "cpsr_save:                                     \n"
        "    MRS   r0, CPSR                             \n" /* Original PSR value */
        "    BIC   r0, r0, %[bsp_mode_mask]             \n" /* Clear the mode bits */
        "    ORR   r0, r0, %[bsp_svc_mode]              \n" /* Set SVC mode bits */
        "    MSR   SPSR_hyp, r0                         \n"
        ::[bsp_mode_mask] "i" (BSP_MODE_MASK), [bsp_svc_mode] "i" (BSP_SVC_MODE) : "memory");

    __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");
}
```

Delete lines 275-279 in fpu_slavetcm_init() in startup_core.c

```
#if __FPU_USED

    /* Initialize FPU and Advanced SIMD setting */
    bsp_fpu_advancedsimd_init();
#endif
```

Change the name of the system_init() function in startup_core.c to system_init() and add some processing. The code before and after the modify is shown below. The red-marked parts are modified. Please note that there is a difference in the number of lines to be modified between RZ/T2M, RZ/T2L, RZ/T2H, RZ/N2L and RZ/N2H. The only difference is the number of lines, the processing that needs to be changed is the same.

startup_core.c before modifying:
Lines 242-259

```
BSP_TARGET_ARM BSP_ATTRIBUTE_STACKLESS void stack_init (void)
{
    __asm volatile (
        "stack_initialization:                              \n"

        /* Stack setting for EL1 */
        "    CPS  #17                                       \n" /* FIQ mode */
        "    MOV  sp, %[bsp_fiq_stack_end_address]          \n"
        "    CPS  #18                                       \n" /* IRQ mode */
        "    MOV  sp, %[bsp_irq_stack_end_address]          \n"
        "    CPS  #23                                       \n" /* Abort mode */
        "    MOV  sp, %[bsp_abort_stack_end_address]        \n"
        "    CPS  #27                                       \n" /* Undefined mode */
        "    MOV  sp, %[bsp_undefined_stack_end_address]    \n"
        "    CPS  #31                                       \n" /* System mode */
        "    MOV  sp, %[bsp_system_stack_end_address]       \n"
        "    CPS  #19                                       \n" /* SVC mode */
        "    MOV  sp, %[bsp_svc_stack_end_address]          \n"
```

startup_core.c after modifying:
Lines 187-210

```
BSP_TARGET_ARM BSP_ATTRIBUTE_STACKLESS void system_init (void)
{
    __asm volatile (
        "set_vbar:                              \n"
        "    LDR   r0, =__Vectors               \n"
        "    MCR   p15, #0, r0, c12, c0, #0  \n" /* Write r0 to VBAR */
        ::: "memory");

    __asm volatile (
        "stack_initialization:                              \n"

        /* Stack setting for EL1 */
        "    CPS  #17                                       \n" /* FIQ mode */
        "    MOV  sp, %[bsp_fiq_stack_end_address]          \n"
        "    CPS  #18                                       \n" /* IRQ mode */
        "    MOV  sp, %[bsp_irq_stack_end_address]          \n"
        "    CPS  #23                                       \n" /* Abort mode */
        "    MOV  sp, %[bsp_abort_stack_end_address]        \n"
        "    CPS  #27                                       \n" /* Undefined mode */
        "    MOV  sp, %[bsp_undefined_stack_end_address]    \n"
        "    CPS  #31                                       \n" /* System mode */
        "    MOV  sp, %[bsp_system_stack_end_address]       \n"
        "    CPS  #19                                       \n" /* SVC mode */
        "    MOV  sp, %[bsp_svc_stack_end_address]          \n"
```

3-2. For Cortex®-A55 projects:

Delete the lines that set the background to gray in the following code.

bsp_register_initialization() in startup_core.c, lines 300-324

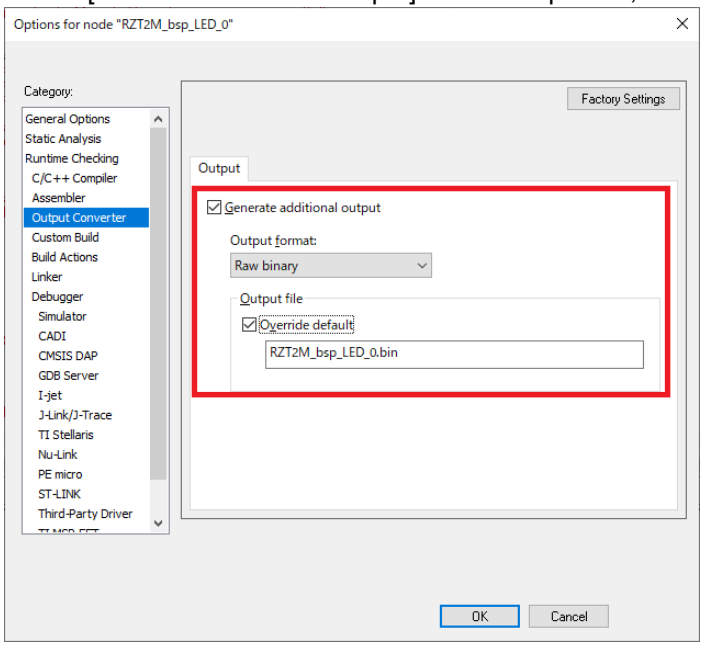
```
BSP_ATTRIBUTE_STACKLESS void bsp_register_initialization (void)
{
    /* Set Vector Base Address Register (VBAR) to point to initializer routine */
    __asm volatile (
        "LDR  x0, =__Vectors                    \n"
        "MSR  VBAR_EL3, x0                      \n"
        "MSR  VBAR_EL2, x0                      \n"
        "MSR  VBAR_EL1, x0                      \n"
        ::: "memory");

    /* Top of level system control init at each EL level
     *  Stop the following systems
     *  MMU, cache, and some memory control system, Tag check fault, pointer authentication */
    __asm volatile (
        "MOV  x0, #0                            \n"
        "MSR  SCTLR_EL3, x0                     \n"
        "MSR  SCTLR_EL2, x0                     \n"
        "MSR  SCTLR_EL1, x0                     \n"
        ::: "memory");

    /* Setting up Stack Area (EL3) */
    __asm volatile (
        "MOV  x1, %0 \n"
        "MOV  sp, x1 \n"
        ::"r" (BSP_STACK_LIMIT) : "memory");
```

4. Change the project settings to output the user application program in binary format. Select the project option and select the output converter in the category list.

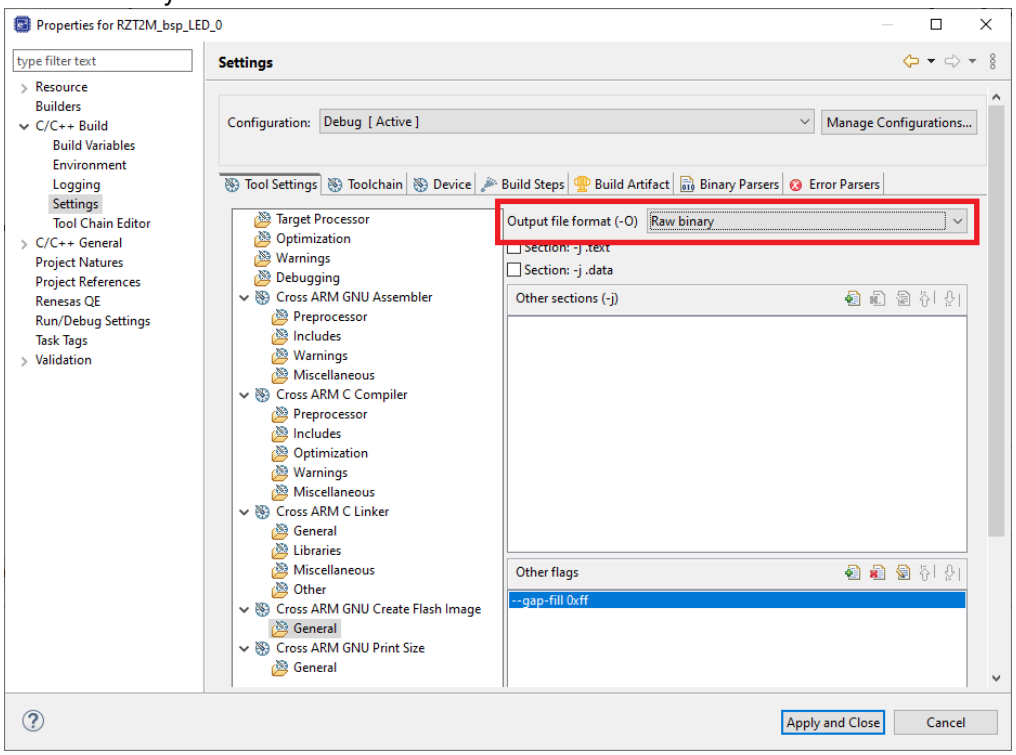
EWARM:

Select the project option and select the output converter in the category list.

Check [Generate additional output] on the Output tab, select [Raw binary] and enter the output file name.



e² studio:
Select the project properties and select Settings under C/C++ Build.
Select "Cross ARM GUN Create Flash Image" on the "Tool Setting" tab and change the Output file format to "Raw binary".



5. Open the following file and change the sample processing.
   RZT2M_bsp_LED_0\src\hal_entry.c
   This is the file path for RZ/T2M. For RZ/T2L, RZ/N2L, RZ/T2H and RZ/N2H read its MPU name instead.

   Add processing as a user application program to the opened file.
   The generated code contains the processing to blink the LED.
   The code for RZT2M is below, but RZ/T2L, RZ/N2L, RZ/T2H and RZ/N2H also include the process of blinking the LED.
   hal_entry.c lines 56-73:

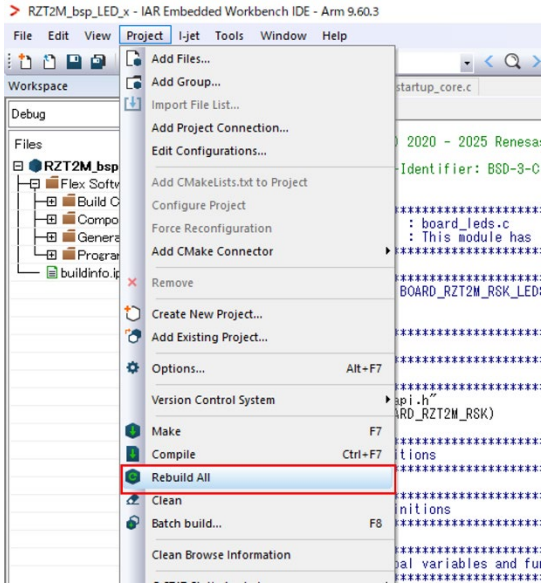
```
    /* This code uses BSP IO functions to show how it is used.*/
    /* Turn off LEDs */
    for (uint32_t i = 0; i < leds.led_count; i++)
    {
        R_BSP_PinClear((bsp_io_region_t) leds.p_leds[i][1], (bsp_io_port_pin_t) leds.p_leds[i][0]);
    }

    while (1)
    {
        /* Toggle board LEDs */
        for (uint32_t i = 0; i < leds.led_count; i++)
        {
            R_BSP_PinToggle((bsp_io_region_t) leds.p_leds[i][1], (bsp_io_port_pin_t) leds.p_leds[i][0]);
        }

        /* Delay */
        R_BSP_SoftwareDelay(delay, bsp_delay_units);
    }
```

   The LED is defined in the following file. Blinks the LED defined here.
   RZT2M_bsp_LED_0\rzt\board\rzt2m_rsk\board_leds.c
   This is the file path for RZ/T2M. For RZ/T2L, RZ/N2L, RZ/T2H and RZ/N2H read its MPU name instead.
   board_leds.c lines 37-48:

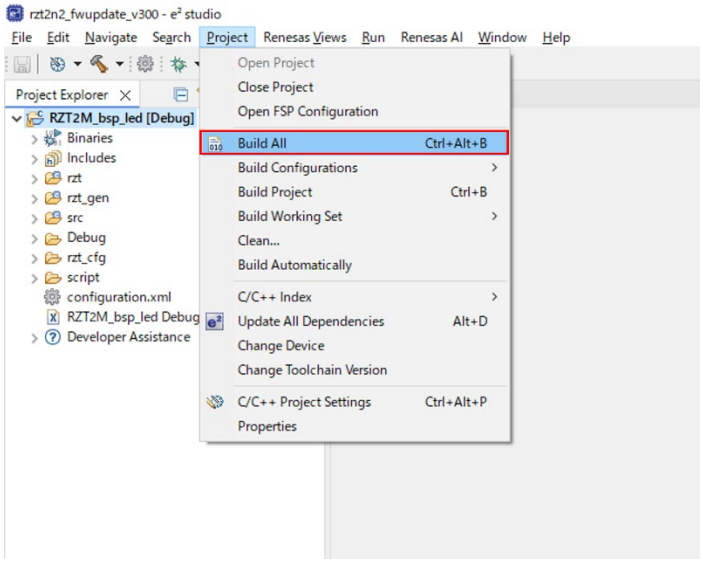
```
static const uint32_t g_bsp_prv_leds[][2] =
{
 #if defined(BSP_CFG_CORE_CR52)
  #if (0 == BSP_CFG_CORE_CR52)
    {(uint32_t) BSP_IO_PORT_19_PIN_6, (uint32_t) BSP_IO_REGION_SAFE}, ///< LED0(Green)
    {(uint32_t) BSP_IO_PORT_19_PIN_4, (uint32_t) BSP_IO_REGION_SAFE}  ///< LED1(Yellow)
  #elif (1 == BSP_CFG_CORE_CR52)
    {(uint32_t) BSP_IO_PORT_20_PIN_0, (uint32_t) BSP_IO_REGION_SAFE}, ///< LED2(Red)
    {(uint32_t) BSP_IO_PORT_23_PIN_4, (uint32_t) BSP_IO_REGION_SAFE}  ///< LED3(Red)
  #endif
 #endif
};
```

6. Build the project.

EWARM:
Select [Project]-[Rebuild All] from the EWARM menu.



e^2 studio:
Select [Project]-[Build All] from the e^2 studio menu.



7. After the build is completed, the extension “bin" file is generated.

After building the application program (RZ*_bsp_LED_*.bin), generate parameter for the user application program (hereinafter referred to as parameter for application).

The parameter for application is generated using the tool parameter_generator.py, which is included in the sample program package for the *RZ/T2, RZ/N2 Device Setup Guide for Flash boot (R01AN6471EJ****)*.

In section 3.8.1 to 3.8.6, the procedure for generating the parameter for application is explained for each MCU type.

3.8.1 Generate Parameter for Application for RZ/T2M

Generates parameter for the user application program (RZT2M_bsp_LED_0.bin).

The following shows an example of tool execution when xSPI0 address space flash is specified:

External flash address where the program is stored (--src_addr): 0x60030030

RAM address where the program is loaded (--app_start_addr): 0x00000000

[Only for multi-core configuration] For the user application program for CPU1, set the load destination address (--dest_addr) to 0x10000000 of the SystemRAM.

The following command generates parameter_RZT2M_bsp_LED_0.bin:

```
python parameter_generator.py userapp --src_addr 60030030 --app_start_addr
00000000 -i RZT2M_bsp_LED_0.bin -o parameter_RZT2M_bsp_LED_0.bin
```

3.8.2 Generate Parameter for Application for RZ/T2L

Generates parameter for the user application program (RZT2L_bsp_LED_1.bin).

The following shows an example of tool execution when xSPI0 address space flash is specified:

External flash address where the program is stored (--src_addr): 0x60030030

RAM address where the program is loaded (--app_start_addr): 0x00000000

The following command generates parameter_RZT2L_bsp_LED_1.bin:

```
python parameter_generator.py userapp --src_addr 60030030 --app_start_addr
00000000 -i RZT2L_bsp_LED_1.bin -o parameter_RZT2L_bsp_LED_1.bin
```

3.8.3 Generate Parameter for Application for RZ/N2L

Generates parameter for the user application program (RZN2L_bsp_LED_0.bin).

The following shows an example of tool execution when xSPI0 address space flash is specified:

External flash address where the program is stored (--src_addr): 0x60030030

RAM address where the program is loaded (--app_start_addr): 0x00000000

The following command generates parameter_RZN2L_bsp_LED_0.bin:

```
python parameter_generator.py userapp --src_addr 60030030 --app_start_addr
00000000 -i RZN2L_bsp_LED_0.bin -o parameter_RZN2L_bsp_LED_0.bin
```

3.8.4 Generate Parameter for Application for RZ/T2H Cortex®-R52

Generates parameter for the user application program (RZT2H_bsp_LED_0.bin).

The following shows an example of tool execution when xSPI0 address space flash is specified:

External flash address where the program is stored (--src_addr): 0x40094050

RAM address where the program is loaded (--app_start_addr): 0x00000000

[Only for multi-core configuration] For the user application program for CPU1, set the load destination address (--dest_addr) to 0x10000000 of the SystemRAM.

The following command generates parameter_RZT2H_bsp_LED_0.bin:

```
python parameter_generator.py userapp --src_addr 40094050 --app_start_addr
00000000 -i RZT2H_bsp_LED_0.bin -o parameter_RZT2H_bsp_LED_0.bin
```

3.8.5 Generate Parameter for Application for RZ/T2H Cortex®-A55

Generates parameter for the user application program (RZT2H_bsp_LED_0.bin).

The following shows an example of tool execution when xSPI0 address space flash is specified:

External flash address where the program is stored (--src_addr): 0x40094050

RAM address where the program is loaded (--app_start_addr): 0x10050000

The following command generates parameter_RZT2H_bsp_LED_0.bin:

```
python parameter_generator.py userapp --src_addr 40094050 --app_start_addr
10050000 -i RZT2H_bsp_LED_0.bin -o parameter_RZT2H_bsp_LED_0.bin
```

3.8.6 Generate Parameter for Application for RZ/N2H Cortex®-R52

Generates parameter for the user application program (RZN2H_bsp_LED_3.bin).

The following shows an example of tool execution when xSPI0 address space flash is specified:

External flash address where the program is stored (--src_addr): 0x40094050

RAM address where the program is loaded (--app_start_addr): 0x00000000

[Only for multi-core configuration] For the user application program for CPU1, set the load destination address (--dest_addr) to 0x10000000 of the SystemRAM.

The following command generates parameter_RZN2H_bsp_LED_3.bin:

```
python parameter_generator.py userapp --src_addr 40094050 --app_start_addr
00000000 -i RZN2H_bsp_LED_3.bin -o parameter_RZN2H_bsp_LED_3.bin
```

3.8.7 Generate Parameter for Application for RZ/N2H Cortex®-A55

Generates parameter for the user application program (RZN2H_bsp_LED_3.bin).

The following shows an example of tool execution when xSPI0 address space flash is specified:

External flash address where the program is stored (--src_addr): 0x40094050

RAM address where the program is loaded (--app_start_addr): 0x10050000

The following command generates parameter_RZN2H_bsp_LED_3.bin:

```
python parameter_generator.py userapp --src_addr 40094050 --app_start_addr
10050000 -i RZN2H_bsp_LED_3.bin -o parameter_RZN2H_bsp_LED_3.bin
```

4. Applying Firmware Updates

The procedure for updating user application programs using the firmware update system configured as described in section 3 is as follows.

First, set up the host PC as described in 4.1, and then update the program or programs as described in 4.2.

4.1 Host PC Setup

4.1.1 Tool Setup

Copy fwupdate_utility.py and fwupdate.py to a location of your choice on the host PC.

Next, install Python 3.8 on the host PC.

To install Python 3.8, download the Python 3.8 installer from the URL below and run it.

https://www.python.org/downloads/windows/

The tools fwupdate_utility.py and fwupdate.py are intended to be run on a Windows system with Python 3.8 installed. Their operation has been confirmed on systems running Windows 10.

Run the following command to view help on using fwupdate_utility.py:

```
fwupdate_utility.py -h
```

Run the following command to view help on using fwupdate.py:

```
fwupdate.py -h
```

4.1.2 Network Adapter Settings

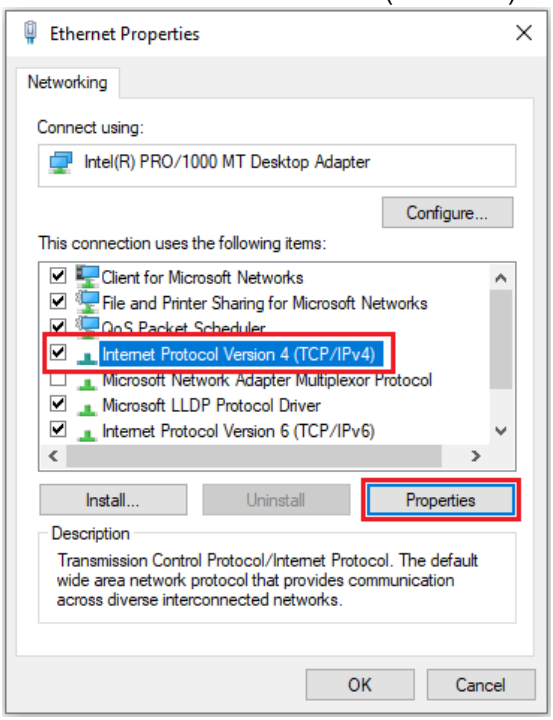
In order to use fwupdate.py to send update files to a RZ/T2M, RZ/T2L, RZ/T2H, RZ/N2L or RZ/N2H device, the host PC and the device must be connected to the same network. Table 4.1 lists the address settings for the device and the host PC.

Table 4.1 Update Environment Address Settings

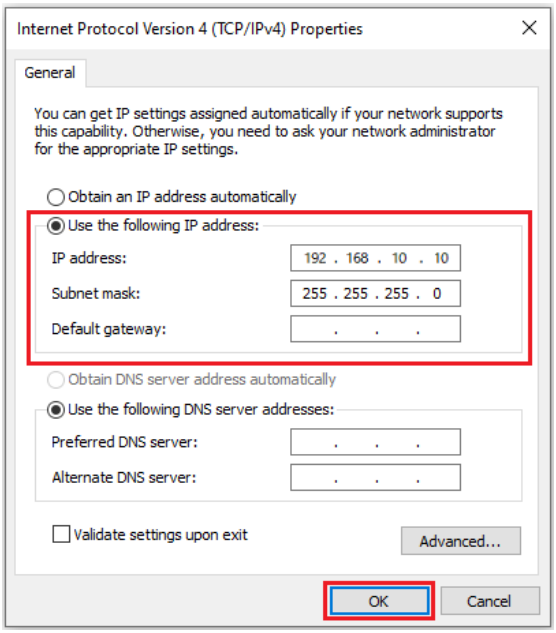
Device	IP Address	Net Mask
RZ/T2M, RZ/T2L, RZ/T2H, RZ/N2L or RZ/N2H device	192.168.10.100	255.255.255.0
Host PC	192.168.10.10	255.255.255.0

Example host PC network adapter settings are shown below (example of settings on Windows 10).

1. Open the network adapter properties window on the host PC.
2. Select Internet Protocol Version (TCP/IPv4) and open the properties window.



3. In the Use following IP address section, enter settings for the IP address and subnet mask, then click the OK button.



4.2 Update Procedure

The procedure for updating the user application program using the firmware update system configured on the RSK+ is described below. If your system has not yet been configured as described in section 3, first complete the system configuration before proceeding.

Table 4.2 shows the environment required to update programs on the RZ/T2M, RZ/T2L, RZ/N2L, RZ/T2H or RZ/N2H.

Table 4.2 Update Environment

Name	Remarks
Evaluation board	RZ/T2M RSK+, RZ/T2L RSK+, RZ/N2L RSK+, RZ/T2H EVB or RZ/N2H EVB
USB cable	1 (Type-C, type-A)
Ether cable	
Host PC	Operation confirmed on Windows 10.
fwupdate_utility.py	Update file generator tool
fwupdate.py	Update files send tool

4.2.1 Update Procedure for RZ/T2M

Table 4.3 lists the user application programs to be updated for RZ/T2M.

Table 4.3 User Application Programs to be Updated for RZ/T2M

File Name	Description
RZT2M_bsp_LED_1.bin	A user application program based on the Blinky sample application included in the Flexible Software Package RZ/T2M pack. See section 3.8 for detailed creation instructions. Note The program is assigned from 0x00000000 in the ATCM area. The following LEDs will light up at startup. User LED1: BSP_IO_PORT_19_PIN_4
parameter_RZT2M_bsp_LED_1.bin	Parameter for the user application program (RZT2M_bsp_LED_1.bin). This file is created after building RZT2M_bsp_LED_1.bin. See section 3.8 for detailed creation instructions. Note The parameter file sets the following xSPI0 address space. External flash address where the program is stored: 0x60040000 RAM address where the program is loaded: 0x00000000

Note The explanation of 3.8 is about RZT2M_bsp_LED_0, but please replace the file name and address with the user application program to be updated.

4.2.1.1 Creating Update File

Use fwupdate_utility.py to create the update file (RZT2M_bsp_LED_1.bin.fwup). Open a command prompt on the host PC and run the following command.

[Only for multi-core configuration] For user application programs for CPU1, specify "1" for the CPU specification option (--cpu_core).

The following command will generate RZT2M_bsp_LED_1.bin.fwup:

```
python fwupdate_utility.py updatefile --cpu_core 0 --param
parameter_RZT2M_bsp_LED_1.bin --write_addr 60040000 -i RZT2M_bsp_LED_1.bin -
o RZT2M_bsp_LED_1.bin.fwup
```

4.2.1.2 Applying Update

Connect the host PC to the RSK+ with an Ethernet cable. Ethernet uses ETH0 for RZ/T2M. The host PC must be set up as described in 4.1 beforehand.

Set SW3-1 to ON. Reset the device to boot with the update.

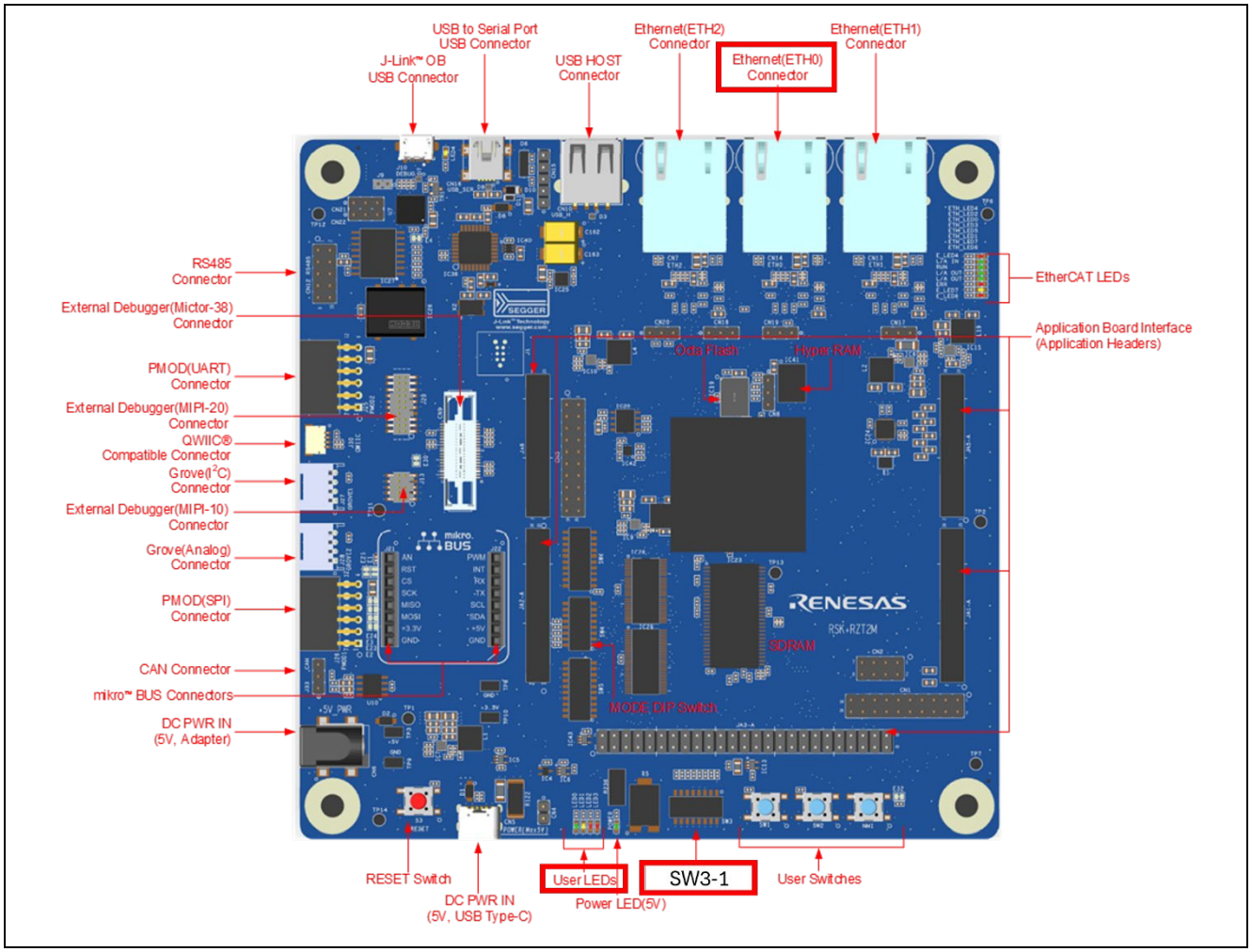


Figure 4.1 Location of User DIP Switch (SW3-1), ETH0 and User LEDs (LED1) for RZ/T2M

1. Set SW3-1 to ON. After setting, press the reset button S3 on RSK+.
2. Use fwupdate.py to transfer the update file to the device. Open a command prompt on the host PC and run the following command.

   The following command will transfer RZT2M_bsp_LED_1.bin.fwup:

```
python fwupdate.py --ip_address 192.168.10.100 --udp_port 10001 --tcp_port
10000 -i RZT2M_bsp_LED_1.bin.fwup
```

3. When the device receives the update file and successfully updates the user application program, the following result is displayed at the command prompt:

```
192.168.10.100 Update success.
```

4. Set SW3-1 to OFF. After setting, press reset button S3 on RSK+ to launch the updated user application program. For RZT2M, User LED3 on the board blinks.

4.2.2 Update Procedure for RZ/T2L

Table 4.4 lists the user application programs to be updated for RZ/T2L.

Table 4.4 User Application Programs to be Updated for RZ/T2L

File Name	Description
RZT2L_bsp_LED_3.bin	A user application program based on the Blinky sample application included in the Flexible Software Package RZ/T2L pack. See section 3.8 for detailed creation instructions. Note The program is assigned from 0x00000000 in the ATCM area. The following LEDs will light up at startup. User LED3: BSP_IO_PORT_18_PIN_1
parameter_RZT2L_bsp_LED_3.bin	Parameter for the user application program (RZT2L_bsp_LED_3.bin). This file is created after building RZT2L_bsp_LED_3.bin. See section 3.8 for detailed creation instructions. Note The following OSPI flash addresses are set in the parameter file. External flash address where the program is stored: 0x60040000 RAM address where the program is loaded: 0x00000000

Note The explanation of 3.8 is about RZT2M_bsp_LED_0, but please replace the file name and address with the user application program to be updated.

4.2.2.1 Creating Update File

Use fwupdate_utility.py to create the update file (RZT2L_bsp_LED_3.bin.fwup). Open a command prompt on the host PC and run the following command.

The following command will generate RZT2L_bsp_LED_3.bin.fwup:

```
python fwupdate_utility.py updatefile --cpu_core 0 --param
parameter_RZT2L_bsp_LED_3.bin --write_addr 60040000 -i RZT2L_bsp_LED_3.bin -
o RZT2L_bsp_LED_3.bin.fwup
```

4.2.2.2 Applying Update

Connect the host PC to the RSK+ with an Ethernet cable. Ethernet uses ETH2 for RZ/T2L. The host PC must be set up as described in 4.1 beforehand.

Set SW3-2 to ON. Reset the device to boot with the update.

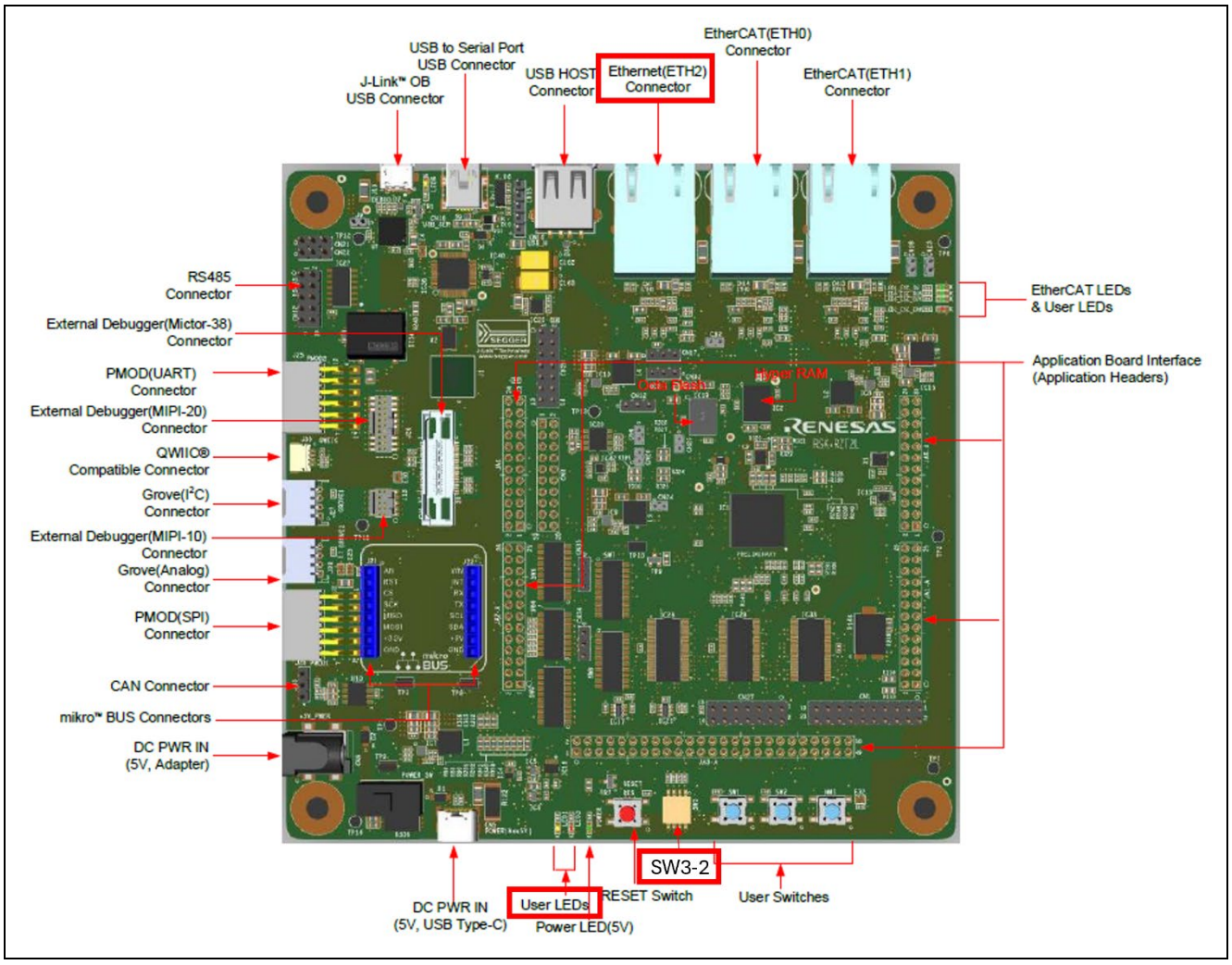


Figure 4.2 Location of User DIP Switch (SW3-2), ETH2 and User LEDs (LED3) for RZ/T2L

1. Set SW3-2 to ON. After setting, press the reset button S3 on RSK+.
2. Use fwupdate.py to transfer the update file to the device. Open a command prompt on the host PC and run the following command.

   The following command will transfer RZT2L_bsp_LED_3.bin.fwup:

```
python fwupdate.py --ip_address 192.168.10.100 --udp_port 10001 --tcp_port
10000 -i RZT2L_bsp_LED_3.bin.fwup
```

3. When the device receives the update file and successfully updates the user application program, the following result is displayed at the command prompt:

```
192.168.10.100 Update success.
```

4. Set SW3-2 to OFF. After setting, press reset button S3 on RSK+ to launch the updated user application program. For RZ/T2L, User LED3 on the board blinks.

4.2.3 Update Procedure for RZ/N2L

Table 4.5 lists the user application programs to be updated for RZ/N2L.

Table 4.5 User Application Programs to be Updated for RZ/N2L

File Name	Description
RZN2L_bsp_LED_3.bin	A user application program based on the Blinky sample application included in the Flexible Software Package RZ/N2L pack. See section 3.8 for detailed creation instructions. Note The program is assigned from 0x00000000 in the ATCM area. The following LEDs will light up at startup. User LED3: BSP_IO_PORT_17_PIN_3
parameter_RZN2L_bsp_LED_3.bin	Parameter for the user application program (RZN2L_bsp_LED_3.bin). This file is created after building RZN2L_bsp_LED_3.bin. See section 3.8 for detailed creation instructions. Note The parameter file sets the following xSPI0 address space. External flash address where the program is stored: 0x60040000 RAM address where the program is loaded: 0x00000000

Note The explanation of 3.8 is about RZT2M_bsp_LED_0, but please replace the file name and address with the user application program to be updated.

4.2.3.1 Creating Update File

Use fwupdate_utility.py to create the update file (RZN2L_bsp_LED_3.bin.fwup). Open a command prompt on the host PC and run the following command.

The following command will generate RZN2L_bsp_LED_3.bin.fwup:

```
python fwupdate_utility.py updatefile --cpu_core 0 --param
parameter_RZN2L_bsp_LED_3.bin --write_addr 60040000 -i RZN2L_bsp_LED_3.bin -
o RZN2L_bsp_LED_3.bin.fwup
```

4.2.3.2 Applying Update

Connect the host PC to the RSK+ with an Ethernet cable. Ethernet uses ETH0 for RZ/N2L. The host PC must be set up as described in 4.1 beforehand.

Set SW3-1 to ON. Reset the device to boot with the update.

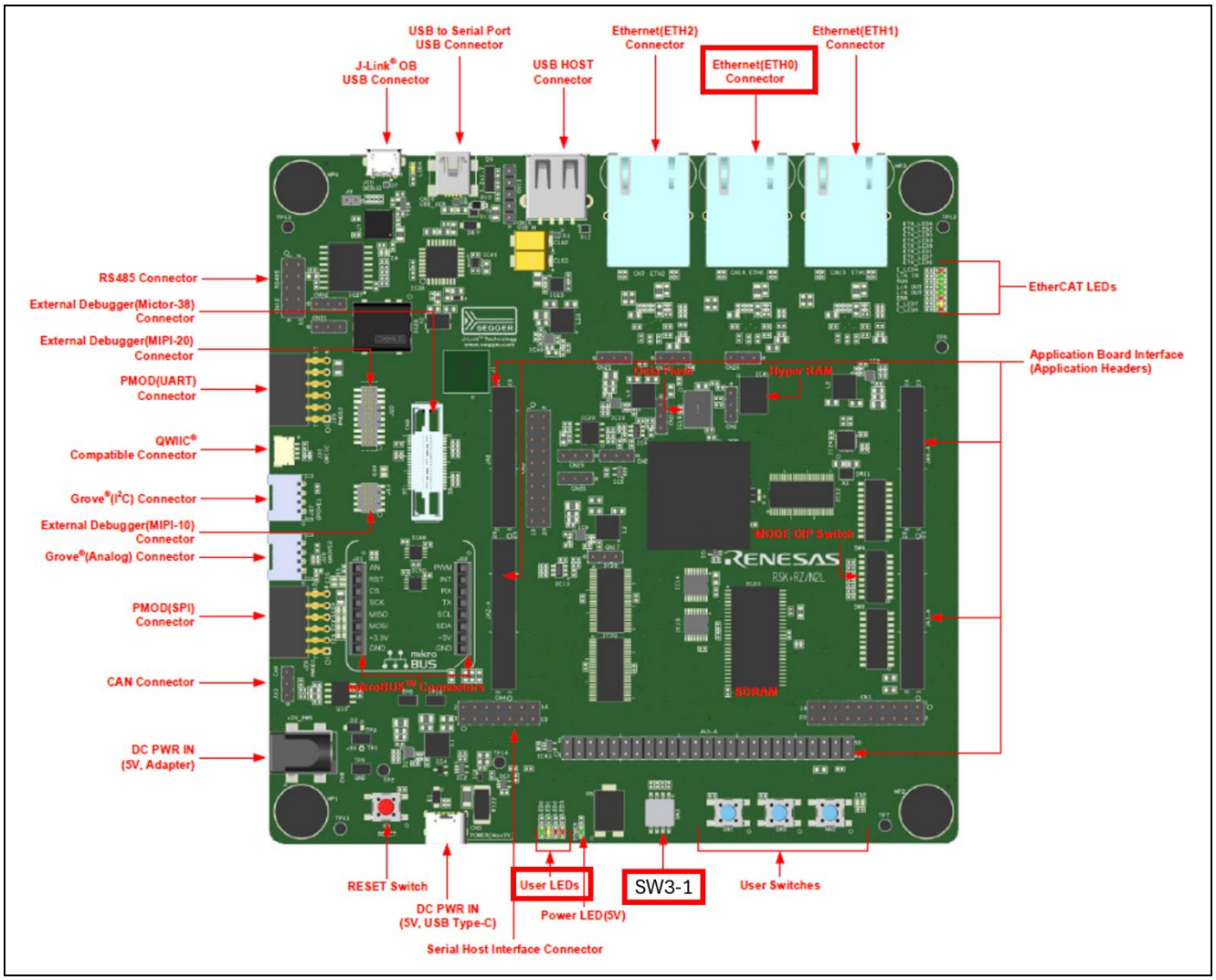


Figure 4.3 Location of User DIP Switch (SW3-1), ETH0 and User LEDs (LED3) for RZ/N2L

3. Set the User DIP Switch to ON. Use SW3-1 for RZ/N2L. After setting, press the reset button S3 on RSK+.

4. Use fwupdate.py to transfer the update file to the device. Open a command prompt on the host PC and run the following command.

   The following command will transfer RZN2L_bsp_LED_3.bin.fwup:

```
python fwupdate.py --ip_address 192.168.10.100 --udp_port 10001 --tcp_port
10000 -i RZN2L_bsp_LED_3.bin.fwup
```

3. When the device receives the update file and successfully updates the user application program, the following result is displayed at the command prompt:

```
192.168.10.100 Update success.
```

4. Set SW3-1 to OFF. After setting, press reset button S3 on RSK to launch the updated user application program. For RZ/N2L, User LED3 on the board blinks.

4.2.4 Update Procedure for RZ/T2H

Table 4.6 lists the user application programs to be updated for RZ/T2H Cortex®-R52. Table 4.7 lists the user application programs to be updated for RZ/T2H Cortex®-A55.

Table 4.6 User Application Programs to be Updated for RZ/T2H Cortex®-R52

File Name	Description
RZT2H_bsp_LED_1.bin	A user application program based on the Blinky sample application included in the Flexible Software Package RZ/T2H pack. See section 3.8 for detailed creation instructions. Note The program is assigned from 0x00000000 in the ATCM area. The following LEDs will light up at startup. User LED1: BSP_IO_PORT_32_PIN_2
parameter_RZT2H_bsp_LED_1.bin	Parameter for the user application program (RZT2H_bsp_LED_1.bin). This file is created after building RZT2H_bsp_LED_1.bin. See section 3.8 for detailed creation instructions. Note The parameter file sets the following xSPI0 address space. External flash address where the program is stored: 0x400A0000 RAM address where the program is loaded: 0x00000000

Note The explanation of 3.8 is about RZT2H_bsp_LED_0, but please replace the file name and address with the user application program to be updated.

Table 4.7 User Application Programs to be Updated for RZ/T2H Cortex®-A55

File Name	Description
RZT2H_bsp_LED_1.bin	A user application program based on the Blinky sample application included in the Flexible Software Package RZ/T2H pack. See section 3.8 for detailed creation instructions. Note The program is assigned from 0x10050000 in the SystemRAM area. The following LEDs will light up at startup. User LED1: BSP_IO_PORT_32_PIN_2
parameter_RZT2H_bsp_LED_1.bin	Parameter for the user application program (RZT2H_bsp_LED_1.bin). This file is created after building RZT2H_bsp_LED_1.bin. See section 3.8 for detailed creation instructions. Note The parameter file sets the following xSPI0 address space. External flash address where the program is stored: 0x400A0000 RAM address where the program is loaded: 0x10050000

Note The explanation of 3.8 is about RZT2H_bsp_LED_0, but please replace the file name and address with the user application program to be updated.

4.2.4.1 Creating Update File

Use fwupdate_utility.py to create the update file (RZT2H_bsp_LED_1.bin.fwup). Open a command prompt on the host PC and run the following command.

RZ/T2H Cortex®-R52:

[Only for multi-core configuration] For user application programs for CPU1, specify "1" for the CPU specification option (--cpu_core).

The following command will generate RZT2H_bsp_LED_1.bin.fwup:

```
python fwupdate_utility.py updatefile --cpu_core 0 --param
parameter_RZT2H_bsp_LED_1.bin --write_addr 400A0000 -i RZT2H_bsp_LED_1.bin -
o RZT2H_bsp_LED_1.bin.fwup
```

RZ/T2H Cortex®-A55:

[Only for multi-core configuration] For user application programs for Core 1, Core 2, or Core 3, specify one of "1" to "3" for the CPU specification option (--cpu_core).

The following command will generate RZT2H_bsp_LED_1.bin.fwup:

```
python fwupdate_utility.py updatefile --cpu_core 0 --param
parameter_RZT2H_bsp_LED_1.bin --write_addr 400A0000 -i RZT2H_bsp_LED_1.bin -
o RZT2H_bsp_LED_1.bin.fwup
```

4.2.4.2 Applying Update

Connect the host PC to the EVB with an Ethernet cable. Ethernet uses ETH0 for RZ/T2H. The host PC must be set up as described in 4.1 beforehand.

Set SW12-1 to ON. Reset the device to boot with the update.

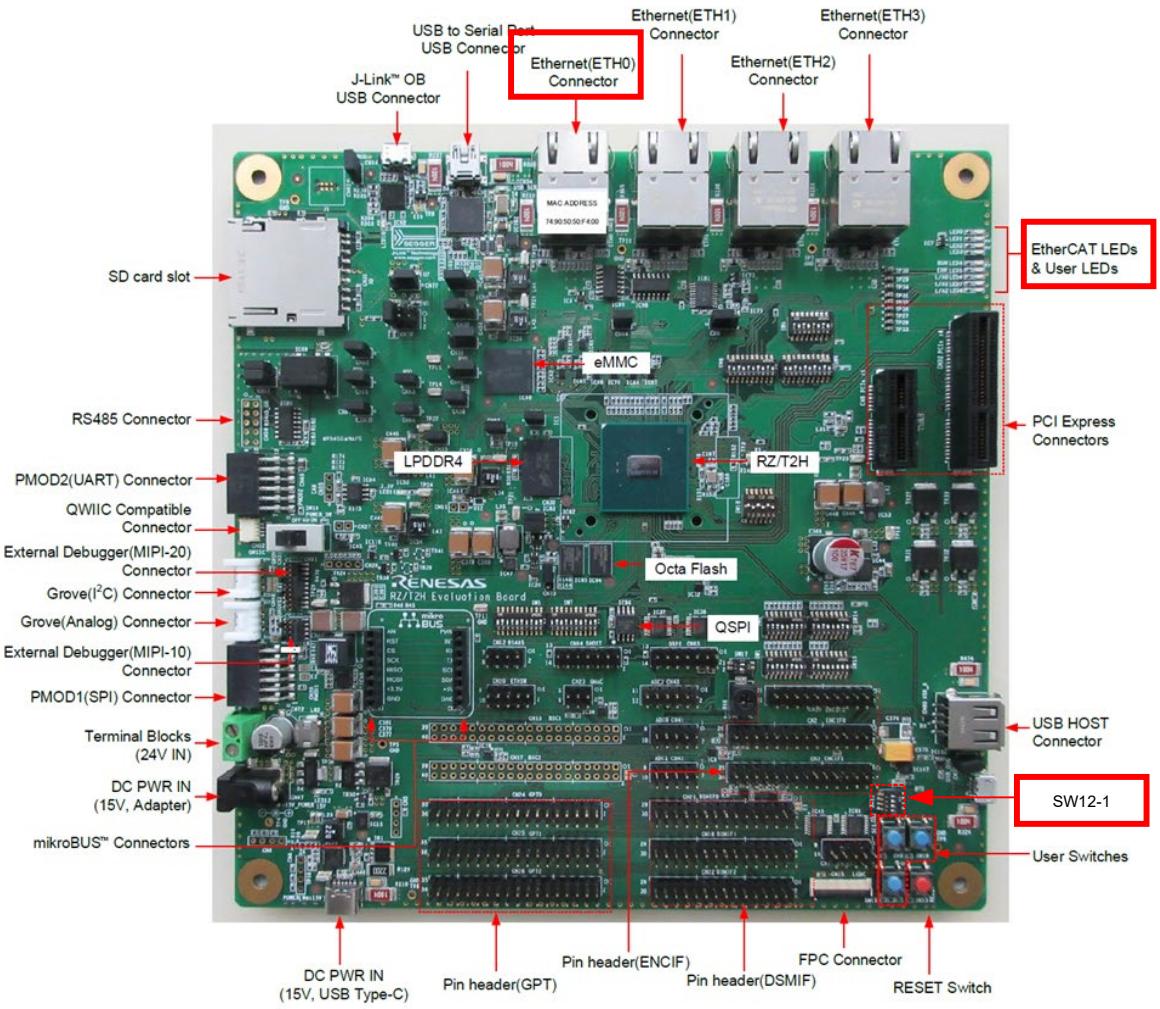


Figure 4.4 Location of User DIP Switch (SW12-1), ETH0 and User LEDs (LED1) for RZ/T2H

1. Set SW12-1 to ON. After setting, press the reset button SW13 on EVB.
2. Use fwupdate.py to transfer the update file to the device. Open a command prompt on the host PC and run the following command.

   The following command will transfer RZT2H_bsp_LED_1.bin.fwup:

```
python fwupdate.py --ip_address 192.168.10.100 --udp_port 10001 --tcp_port
10000 -i RZT2H_bsp_LED_1.bin.fwup
```

3. When the device receives the update file and successfully updates the user application program, the following result is displayed at the command prompt:

```
192.168.10.100 Update success.
```

4. Set SW12-1 to OFF. After setting, press reset button SW13 on EVB to launch the updated user application program. For RZT2H, User LED1 on the board blinks.

4.2.5 Update Procedure for RZ/N2H

Table 4.8 lists the user application programs to be updated for RZ/N2H Cortex®-R52. Table 4.9 lists the user application programs to be updated for RZ/N2H Cortex®-A55.

Table 4.8 User Application Programs to be Updated for RZ/N2H Cortex®-R52

File Name	Description
RZN2H_bsp_LED_4.bin	A user application program based on the Blinky sample application included in the Flexible Software Package RZ/N2H pack. See section 3.8 for detailed creation instructions. Note The program is assigned from 0x00000000 in the ATCM area. The following LEDs will light up at startup. User LED4: BSP_IO_PORT_18_PIN_1
parameter_RZN2H_bsp_LED_4.bin	Parameter for the user application program (RZN2H_bsp_LED_4.bin). This file is created after building RZN2H_bsp_LED_4.bin. See section 3.8 for detailed creation instructions. Note The parameter file sets the following xSPI0 address space. External flash address where the program is stored: 0x400A0000 RAM address where the program is loaded: 0x00000000

Note The explanation of 3.8 is about RZN2H_bsp_LED_3, but please replace the file name and address with the user application program to be updated.

Table 4.9 User Application Programs to be Updated for RZ/N2H Cortex®-A55

File Name	Description
RZN2H_bsp_LED_4.bin	A user application program based on the Blinky sample application included in the Flexible Software Package RZ/N2H pack. See section 3.8 for detailed creation instructions. Note The program is assigned from 0x10050000 in the SystemRAM area. The following LEDs will light up at startup. User LED4: BSP_IO_PORT_18_PIN_1
parameter_RZN2H_bsp_LED_4.bin	Parameter for the user application program (RZN2H_bsp_LED_4.bin). This file is created after building RZN2H_bsp_LED_4.bin. See section 3.8 for detailed creation instructions. Note The parameter file sets the following xSPI0 address space. External flash address where the program is stored: 0x400A0000 RAM address where the program is loaded: 0x10050000

Note The explanation of 3.8 is about RZN2H_bsp_LED_3, but please replace the file name and address with the user application program to be updated.

4.2.5.1 Creating Update File

Use fwupdate_utility.py to create the update file (RZN2H_bsp_LED_4.bin.fwup). Open a command prompt on the host PC and run the following command.

RZ/N2H Cortex®-R52:

[Only for multi-core configuration] For user application programs for CPU1, specify "1" for the CPU specification option (--cpu_core).

The following command will generate RZN2H_bsp_LED_4.bin.fwup:

```
python fwupdate_utility.py updatefile --cpu_core 0 --param
parameter_RZN2H_bsp_LED_4.bin --write_addr 400A0000 -i RZN2H_bsp_LED_4.bin -
o RZN2H_bsp_LED_4.bin.fwup
```

RZ/N2H Cortex®-A55:

[Only for multi-core configuration] For user application programs for Core 1, Core 2, or Core 3, specify one of "1" to "3" for the CPU specification option (--cpu_core).

The following command will generate RZN2H_bsp_LED_4.bin.fwup:

```
python fwupdate_utility.py updatefile --cpu_core 0 --param
parameter_RZN2H_bsp_LED_4.bin --write_addr 400A0000 -i RZN2H_bsp_LED_4.bin -
o RZN2H_bsp_LED_4.bin.fwup
```

4.2.5.2 Applying Update

Connect the host PC to the EVB with an Ethernet cable. Ethernet uses ETH0 for RZ/N2H. The host PC must be set up as described in 4.1 beforehand.

Set DSW1-1 to ON. Reset the device to boot with the update.

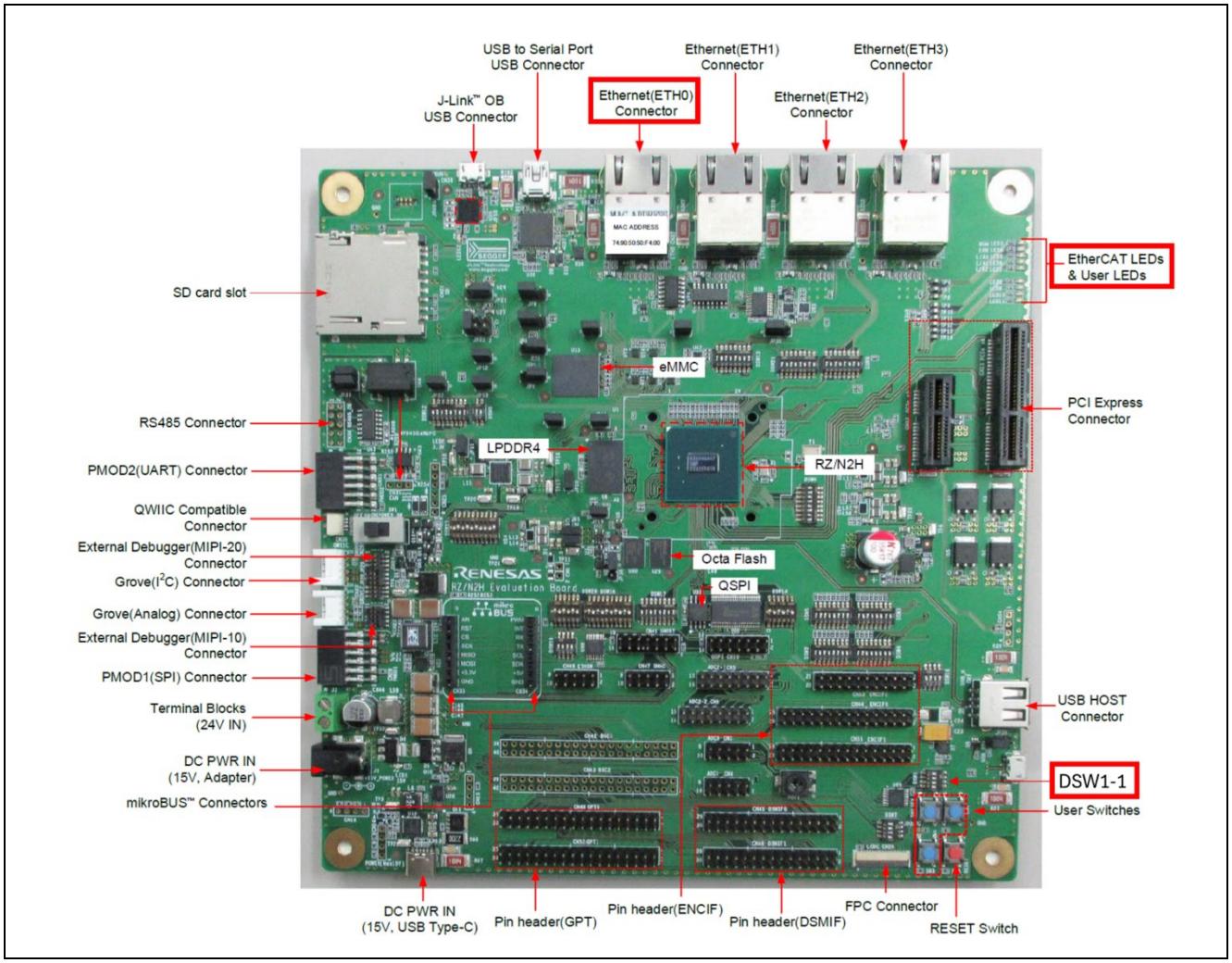


Figure 4.5 Location of User DIP Switch (DSW1-1), ETH0 and User LEDs (LED4) for RZ/N2H

1. Set DSW1-1 to ON. After setting, press the reset button SW5 on EVB.
2. Use fwupdate.py to transfer the update file to the device. Open a command prompt on the host PC and run the following command.

   The following command will transfer RZN2H_bsp_LED_4.bin.fwup:

```
python fwupdate.py --ip_address 192.168.10.100 --udp_port 10001 --tcp_port
10000 -i RZN2H_bsp_LED_4.bin.fwup
```

3. When the device receives the update file and successfully updates the user application program, the following result is displayed at the command prompt:

```
192.168.10.100 Update success.
```

4. Set DSW1-1 to OFF. After setting, press reset button SW5 on EVB to launch the updated user application program. User LED4 on the board blinks.

5. Sample Program

This package is provided as a set of sample program projects including source codes and tool body files in the execution format. This sample program projects and tools can be modified for each user environment.

In this section, the external specifications of the update program included in the sample program package are described in 5.1 and 5.2, and the implementation specifications of the update program are described in 5.3. In addition, the specifications of the tools used with the sample program are described in 5.4.

5.1 Update File Format

Figure 5.1 shows the update file format that can be handled by the update program.

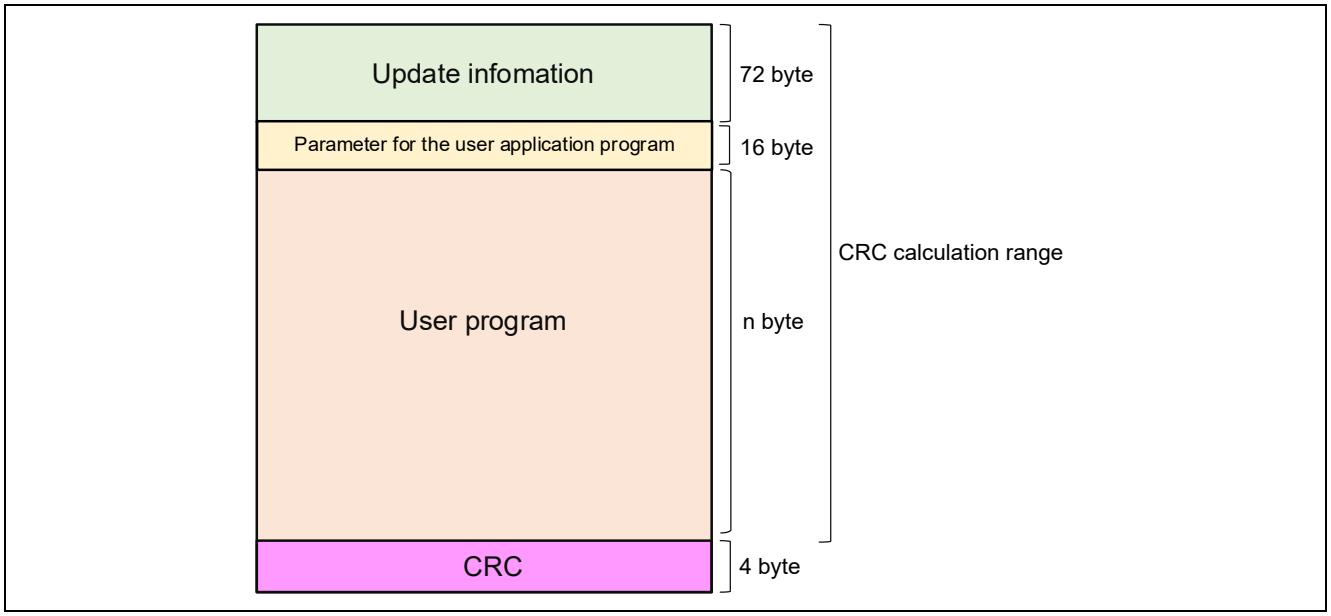


Figure 5.1 Update File Format

At the start of the update file is information such as the size of the user application program and the write destination address in the external flash memory, stored in the form of a 72-byte update information block. Table 5.1 shows the format of the file information. The unit of the Offset and Size values listed in Table 5.1 is bytes.

The byte order of each field in the Update information shall be little-endian.

Table 5.1 Update Information Format

<table>
<tr><th>Offset</th><th>Field</th><th>Size</th><th>Description</th></tr>
<tr><td>0</td><td>Magic Number</td><td>4</td><td>Magic number
Set ASCII code (0x75706469) for "updi"</td></tr>
<tr><td>4</td><td>Reserved</td><td>8</td><td>Fixed 0</td></tr>
<tr><td>12</td><td>Write Address</td><td>4</td><td>When fwupdate_utility.py is executed, set the external flash address to which the firmware specified by the "--write_addr" option is written.</td></tr>
<tr><td>16</td><td>Reserved</td><td>4</td><td>Fixed 0</td></tr>
<tr><td>20</td><td>Image Size</td><td>4</td><td>Total size of parameters for the user application program and the User application program.</td></tr>
<tr><td>24</td><td>Reserved</td><td>4</td><td>Fixed 0</td></tr>
<tr><td>28</td><td>Update Target</td><td>4</td><td>Information required at Update is set in a bit field.
Each bit has the following meaning.
<table>
<tr><td>bit</td><td>Description</td></tr>
<tr><td>0-3</td><td>CPU cores to be updated.
0 (b0000) : Cortex®-R52 CPU0 or Cortex®-A55 Core 0
1 (b0001) : Cortex®-R52 CPU1 or Cortex®-A55 Core 1
2 (b0010) : Cortex®-A55 Core 2
3 (b0011) : Cortex®-A55 Core 3</td></tr>
<tr><td>4</td><td>With or without parameter file input.
0 : Without parameter file
1 : With parameter file</td></tr>
</table></td></tr>
<tr><td>32</td><td>TLV Length</td><td>4</td><td>Total byte size of TLV field.
Fixed 0x00000024</td></tr>
<tr><td>36</td><td>TLV field</td><td>36</td><td>Field consisting of Type&Length and Value
<table>
<tr><td>Offset</td><td>Field</td><td>Size</td><td>Description</td></tr>
<tr><td>0</td><td>Type&Length</td><td>4</td><td>Fixed 0x60000008</td></tr>
<tr><td>4</td><td>Value</td><td>32</td><td>Product name</td></tr>
</table></td></tr>
</table>

5.2 Communication Protocols of Update Program

Figure 5.2 illustrates the communication protocols of the update program running on the device when receiving an update file. The control packets included in the communication protocols are sent and received via UDP and TCP communication.

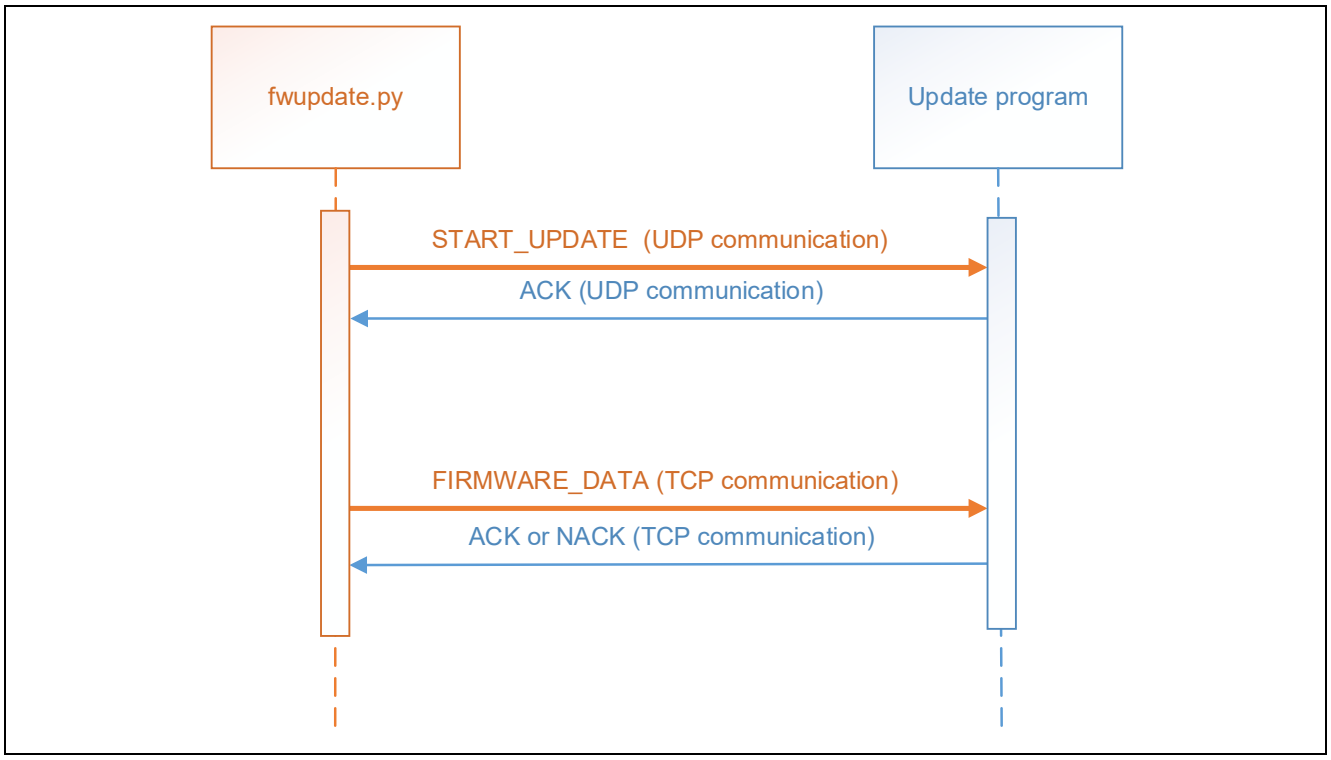


Figure 5.2 Communication Protocols of Update Program

Table 5.2 shows the format of the control packets sent and received by the update program. Note that the unit of the Offset and Size values shown below is bytes.

Table 5.2 Control Packet Format

Offset	Field	Size	Value
0	Command Code	1	Command code
1	Dummy	3	Dummy (For alignment adjustment)
4	Payload size	4	Size of payload: n (little-endian)
8	Payload	n	Data of various types is stored here.

Table 5.3 lists the command codes of the control packets sent and received by the update program. The contents of the packets corresponding to each command code are described in 5.2.1 to 5.2.4.

Table 5.3 Command Codes

Command Code	Value	Description
START_UPDATE	0x11	Reports start of firmware update.
FIRMWARE_DATA	0x12	Sends update file.
ACK	0x81	Firmware update acknowledgement response
NACK	0x82	Firmware update negative acknowledgement response

5.2.1 START_UPDATE

This firmware update start notification is received by the update program. The update program can receive this packet as a UDP broadcast or unicast.

Table 5.4 Contents of START_UPDATE Packet

Offset	Field	Size	Value
0	Command Code	1	0x11
1	Dummy	3	Dummy
4	Payload size	4	0x00000000

5.2.2 FIRMWARE_DATA

This is the form in which the update program receives the update file. The update program receives this packet via TCP communication.

Table 5.5 Contents of FIRMWARE_DATA

Offset	Field	Size	Value
0	Command Code	1	0x12
1	Dummy	3	Dummy
4	Payload size	4	n
8	Payload	n	Update file data

5.2.3 ACK

This acknowledge response is sent by the update program when a command is received successfully. It is sent via UDP in response to a START_UPDATE packet and via TCP in response to a FIRMWARE_DATA packet.

Table 5.6 Contents of ACK

Offset	Field	Size	Value
0	Command Code	1	0x81
1	Dummy	3	Dummy
4	Payload size	4	0x00000000

5.2.4 NACK

This negative acknowledge response is sent by the update program when an error occurs when receiving a command. The update program sends this packet via TCP communication.

Table 5.7 Contents of NACK

Offset	Field	Size	Value
0	Command Code	1	0x82
1	Dummy	3	Dummy
4	Payload size	4	0x00000001
8	Error code	1	Error code • Exceeding the maximum update file size: 0x01 • Update file verification failure: 0x02 • User application write failure: 0x03 • Parameter information writing failure: 0x04 • Key installation failure: 0x05 • Booting surface switching failure: 0x06 • Version update failure: 0x07

5.3 Implementation Specifications of Update Program

5.3.1 Development Environment

Refer to RZ/T2, RZ/N2 Getting Started with Flexible Software Package.

5.3.2 File Structure

Table 5.8 and Table 5.9 list the main files contained in the firmware update sample program project.

Table 5.8 File Structure of Update Program

Folder Name	File Name	Description
RZ*_*_FWUpdate_Rev*\		
├	*.jlink, *.launch, *project, *.eww, *.ewd, *.ewp	Project files
├	*.pincfg, *.xml, *.ipcf	Flexible Software Package Files
├	rz*_cfg.txt	
├ rz*\		
├ rz*_cfg\		
├ rz*_gen\		
├ script\	*.ld, *.icf	Memory allocation
└ src\	*.c, *.h	Update program source code folder

Table 5.9 File Structure of SSBL

Folder Name	File Name	Description
RZ*_*_SSBL_Rev*\		
├	*.jlink, *.launch, *project, *.eww, *.ewd, *.ewp	Project files
├	*.pincfg, *.xml, *.ipcf	Flexible Software Package Files
├	rz*_cfg.txt	
├ rz*\		
├ rz*_cfg\		
├ rz*_gen\		
├ script\	*.ld, *.icf	Memory allocation
└ src\	*.c, *.h	SSBL Source code folder

5.3.3 Functions

Table 5.10 lists the main functions defined in the update program, and Table 5.11 lists the main functions defined in the SSBL.

Table 5.10 Functions of Update Program

File Name	Function Name	Description
fwupdate.c	firmware_update	Main routine of firmware update processing
	check_updatefile	Update file confirmation processing
	write_user_application	Write user applications to flash
	write_param_info	Write parameter for the user application program to flash
	change_flash_mng_area	Update plane management area
	packet_handler	Packet analysis and firmware update control processing
crc32.c	calc_crc32	CRC32 calculation
fwupdate_thread_entry.c	fwupdate_thread_entry	Firmware update thread processing
net_thread_entry.c	net_thread_entry	FreeRTOS TCP
tcp_svr_thread_entry.c	tcp_svr_thread_entry	
udp_svr_thread_entry.c	udp_svr_thread_entry	
flash.c	write_to_qspi_area	QSPI flash memory driver
	read_to_qspi_area	
	write_to_ospi_area	OSPI flash memory driver
	read_to_ospi_area	
	write_to_nor_area	NOR flash memory driver
	read_to_nor_area	

Table 5.11 Functions of SSBL

File Name	Function Name	Description
ssbl.c	second_application_boot_loader	Main routine of loader program
	load_user_application	Load the user application program
	load_user_application_for_multi_core	Load the user application program for multi core
	load_update_program	Load the update program

5.3.4 Flowchart of Update Program Processing

Figure 5.3 is a flowchart showing the processing of the update program.

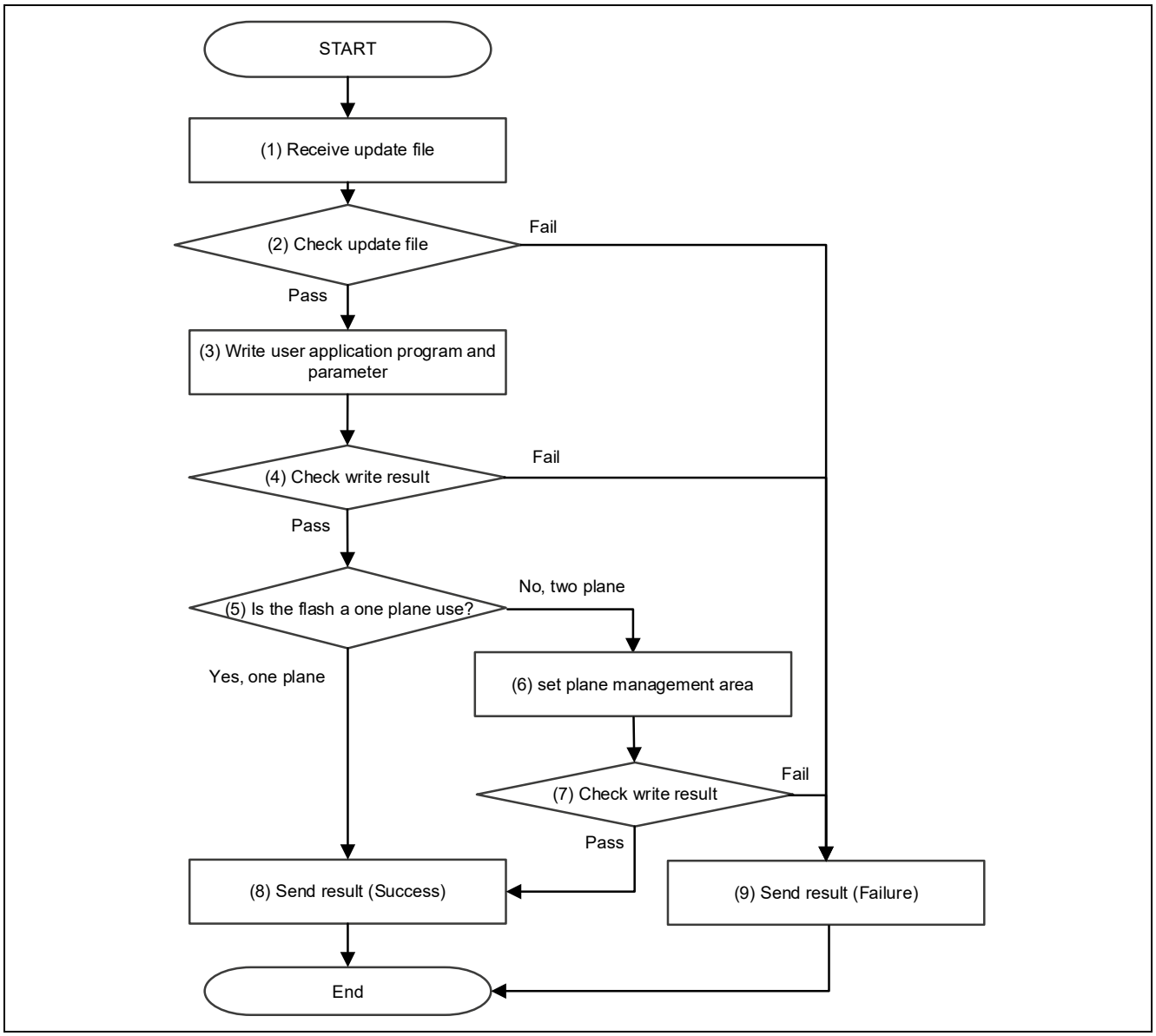


Figure 5.3 Flowchart of Update Program Processing

Details of the update program processing flowchart are described below.

(1) Receive Update File

Related function: packet_handler (fwupdate.c)

UDP and TCP communication are used to receive the update file. Refer to 5.2 for the communication protocols used during update file reception.

(2) Check Update File

Related function: check_updatefile (fwupdate.c)

The CRC of the update file is used to confirm that the file information and user application program data in the update file are not corrupt. The CRC of the file information area and user application program area is calculated using CRC32, and the result is compared to the CRC of the update file to confirm that there are no defects in the update file. If the comparison result is a match, processing jumps to (3) Write User Application Program and Parameter, and if the result is a mismatch, processing jumps to (9) Send Result (Failure).

(3) Write User Application Program and Parameter
Related function: write_user_application, write_param_info (fwupdate.c)
The user application program is written to the external flash memory. The *Write Address* contained in the update information of the update file is used as the write address. The *Image Size* contained in the file information of the update file minus the fixed length parameter size is used as the size of the user application program to be written.

(4) Check Write Result
Related function: write_to_qspi_area, write_to_ospi_area, write_to_nor_area (flash.c)
The data written to the external flash memory is read from the flash memory and checked against the original write data in the RAM to confirm that they match. The write result is success if they match and failure if they do not match. If the write result is success, processing jumps to (5) Is the Flash a One Plain Use?, and if the write result is failure, processing jumps to (9) Send Result (Failure).

(5) Is the Flash a One Plain Use?
Related function: firmware_update (fwupdate.c)
Whether the flash is one-plane use or not is set in the firmware update configuration. If the flash is one-plane use, processing jumps to (8) Send Result (Success), and if the flash is two-plane use, processing jumps to (6) Set Plane Management Area.

(6) Set Plane Management Area
Related function: change_flash_mng_area (fwupdate.c)
Update the settings in the Plane Management Area on the flash to switch the startup plane at the next startup.

(7) Check Write Result
Related function: write_to_qspi_area, write_to_ospi_area, write_to_nor_area (flash.c)
The data written to the external flash memory is read from the flash memory and checked against the original write data in the RAM to confirm that they match. The write result is success if they match and failure if they do not match. If the write result is success, processing jumps to (8) Send Result (Success), and if the write result is failure, processing jumps to (9) Send Result (Failure).

(8) Send Result (Success)
Related function: packet_handler (fwupdate.c)
An ACK packet is transmitted.

(9) Send Result (Failure)
Related function: packet_handler (fwupdate.c)
A NACK packet is transmitted. In addition, an error code is appended indicating a file error, if an error occurred in (2) Check Update File, or indicating a write error, if an error occurred in (4) and (7) Check Write Result.

5.3.5 Memory Maps for RSK+

Table 5.12, Table 5.13, , Table 5.14, Table 5.15 and show memory maps for the sample program using RSK+.

In the table the Update Target column indicates memory areas that can be updated using the update program. Areas with a check mark (✓) in the Update Target column can be updated using the update program. Areas with a check mark (✓) in the Initial Image column can be concatenated using fwupdate_utility.py.

Table 5.12 Memory Map

Memory Type	Address	Size	Description
ATCM Note	0x00000000 -	-	User application program area, Update program area
BTCM Note	0x00100000 - 0x00101FFF	8 KB	Boot loader area
	0x00102000 -	-	SSBL area
System RAM Note	0x10000000 -	-	User application program area, Update program RAM area
External Memory (xSPI0)	0x60000000 - 0x63FFFFFF	64 MB	User application program area (RZ/T2M RSK+ and RZ/N2L RSK+ have QSPI flash, RZ/T2L RSK+ has Octa flash)
External Memory (xSPI1)	0x68000000 - 0x68FFFFFF	16 MB	User application program area (RZ/T2L RSK+ has QSPI flash)
External Memory (CS0)	0x70000000 - 0x71FFFFFF	32 MB	User application program area (RZ/T2M RSK+ and RZ/N2L RSK+ have NOR Flash)

Note The size of the RAM area varies depending on the product.

Table 5.13 Memory Map for the xSPI0 Flash in xSPI0 Boot Mode

Memory Type	Address	Size	Description	Initial Image	Update Target
xSPI0 (64MB)	0x60000000 - 0x6000004F	80 Byte	Area for the parameter for the loader	✓	–
	0x60000050 - 0x6000704F	28 KB	SSBL area	✓	–
	0x60007050 - 0x6002FFFF	163 KB	Update program area	✓	–
	0x60030000 - 0x6003000F	16 Byte	Area for the parameter for the user application program (CPU0)	✓	✓
	0x60030010 - 0x6003001F	16 Byte	Area for the parameter for the user application program (CPU1)	✓	✓
	0x60030020 - 0x6003002F	16 Byte	Plane management area	✓	✓
	0x60030030 - 0x63FFFFFF	63 MB	Area for User application program	–	✓

Table 5.14 Memory Map for the xSPI1 Flash in xSPI1 Boot Mode

Memory Type	Address	Size	Description	Initial Image	Update Target
xSPI1 (16MB)	0x68000000 - 0x6800004F	80 Byte	Area for the parameter for the loader	✓	–
	0x68000050 - 0x6800704F	28 KB	SSBL area	✓	–
	0x68007050 - 0x6802FFFF	163 KB	Update program area	✓	–
	0x68030000 - 0x6803000F	16 Byte	Area for the parameter for the user application program (CPU0)	✓	✓
	0x68030010 - 0x6803001F	16 Byte	Area for the parameter for the user application program (CPU1)	✓	✓
	0x68030020 - 0x6803002F	16 Byte	Plane management area	✓	✓
	0x68030030 - 0x68FFFFFF	15 MB	Area for User application program	–	✓

Table 5.15 Memory Map for the NOR CS0 Flash in 16-bit Bus Boot Mode

Memory Type	Address	Size	Description	Initial Image	Update Target
CS0 (32MB)	0x70000000 - 0x7000004F	80 Byte	Area for the parameter for the loader	✓	–
	0x70000050 - 0x7000704F	28 KB	SSBL area	✓	–
	0x70007050 - 0x7002FFFF	163 KB	Update program area	✓	–
	0x70030000 - 0x7003000F	16 Byte	Area for the parameter for the user application program (CPU0)	✓	✓
	0x70030010 - 0x7003001F	16 Byte	Area for the parameter for the user application program (CPU1)	✓	✓
	0x70030020 - 0x7003002F	16 Byte	Plane management area	✓	✓
	0x70030030 - 0x71FFFFFF	31 MB	Area for User application program	–	✓

5.3.6 Memory Maps for EVB

Table 5.16, Table 5.17, Table 5.18 and Table 5.19 show memory maps for the sample program using EVB.

In table the Update Target column indicates memory areas that can be updated using the update program. Areas with a check mark (✓) in the Update Target column can be updated using the update program. Areas with a check mark (✓) in the Initial Image column can be concatenated using fwupdate_utility.py.

Table 5.16 Memory Map for Cortex®-R52

Memory Type	Address	Size	Description
ATCM	0x00000000 - 0x0007FFFF	512 KB	User application program area, Update program area
BTCM	0x00100000 - 0x0010FFFF	64 KB	Boot loader area, SSBL area
System RAM	0x10000000 - 0x101FFFFF	2.0 MB	User application program area, Update program RAM area
External Memory (xSPI0)	0x40000000 - 0x43FFFFFF	64 MB	User application program area (EVB has Octa flash 64 M Byte)
External Memory (xSPI1)	0x50000000 - 0x50FFFFFF	16 MB	User application program area (EVB has QSPI flash 16 M Byte)

Table 5.17 Memory Map for Cortex®-A55

Memory Type	Address	Size	Description
System RAM	0x10000000 - 0x101FFFFF	2.0 MB	SSBL area, User application program area, Update program area
External Memory (xSPI0)	0x40000000 - 0x43FFFFFF	64 MB	User application program area (EVB has Octa flash 64 M Byte)
External Memory (xSPI1)	0x50000000 - 0x50FFFFFF	16 MB	User application program area (EVB has QSPI flash 16 M Byte)
DDR mirror	0xC0000000 - 0xFFFFFFFF	1.0 GB	User application program area, Update program RAM area

Table 5.18 Memory Map for the xSPI0 Flash in xSPI0 Boot Mode

Memory Type	Address	Size	Description	Initial Image	Update Target
xSPI0 (64MB)	0x40000000 - 0x4000004F	80 Byte	Area for the parameter for the loader	✓	–
	0x40000050 - 0x4003704F	220 KB	SSBL area	✓	–
	0x40037050 - 0x40093FFF	371 KB	Update program area	✓	–
	0x40094000 - 0x4009400F	16 Byte	Area for the parameter for the user application program (Cortex®-R52 CPU0 or Cortex®-A55 Core0)	✓	✓
	0x40094010 - 0x4009401F	16 Byte	Area for the parameter for the user application program (Cortex®-R52 CPU1 or Cortex®-A55 Core1)	✓	✓
	0x40094020 - 0x4009402F	16 Byte	Area for the parameter for the user application program (Cortex®-A55 Core2)	✓	✓
	0x40094030 - 0x4009403F	16 Byte	Area for the parameter for the user application program (Cortex®-A55 Core3)	✓	✓
	0x40094040 - 0x4009404F	16 Byte	Plane management area	✓	✓
	0x40094050 - 0x43FFFFFF	63 MB	Area for User application program	–	✓

Table 5.19 Memory Map for the xSPI1 Flash in xSPI1 Boot Mode

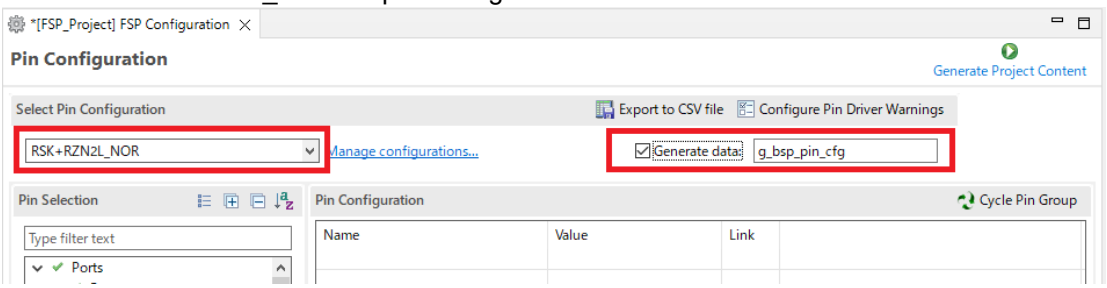
Memory Type	Address	Size	Description	Initial Image	Update Target
xSPI1 (16MB)	0x50000000 - 0x5000004F	80 Byte	Area for the parameter for the loader	✓	–
	0x50000050 - 0x5003704F	220 KB	SSBL area	✓	–
	0x50037050 - 0x50093FFF	371 KB	Update program area	✓	–
	0x50094000 - 0x5009400F	16 Byte	Area for the parameter for the user application program (Cortex®-R52 CPU0 or Cortex®-A55 Core0)	✓	✓
	0x50094010 - 0x5009401F	16 Byte	Area for the parameter for the user application program (Cortex®-R52 CPU1 or Cortex®-A55 Core1)	✓	✓
	0x50094020 - 0x5009402F	16 Byte	Area for the parameter for the user application program (Cortex®-A55 Core2)	✓	✓
	0x50094030 - 0x5009403F	16 Byte	Area for the parameter for the user application program (Cortex®-A55 Core3)	✓	✓
	0x50094040 - 0x5009404F	16 Byte	Plane management area	✓	✓
	0x50094050 - 0x50FFFFFF	16 MB	Area for User application program	–	✓

5.3.7 How to Use NOR Flash in the RZ/N2L Project

The firmware update sample program project for RZ/N2L must be configured to use the external flash.

By default, the setting to use QSPI flash is enabled; to use NOR flash, the following settings are required.

1. Start FSP Configuration.
   For GCC version, use e^2 studio.
   For IAR version, use FSP Smart Configurator.
   For details, Refer to RZ/T2, RZ/N2 Getting Started with Flexible Software Package.
2. Select "RSK + RZN2L_NOR" in pin settings and enable Generate data check.



3. Click Generate Project Content (green play icon).

Sample program code is generated that can use the NOR flash.

5.4 Specifications of Tools Used with Sample Program

5.4.1 fwupdate_utility.py

The tool fwupdate_utility.py is used to create update files or initial image file.

Using fwupdate_utility.py, you can create an update file by specifying the user application program to be updated and the write destination address on the device. The specified address is stored in the file information of the update file.

The initial image file specifies each file that needs to be written during initial setup, and outputs them all in one file. Creating an initial configuration file makes it easier to write to flash during initial setup.

The command format of fwupdate_utility.py is as follows:

```
python fwupdate_utility.py < command > < options >
```

Table 5.20 lists the commands, and Table 5.21 and Table 5.22 lists the options corresponding to each command.

Table 5.20 Commands of fwupdate_utility.py

Commands	Description
updatefile	Create the update file.
setupfile	Create the initial image file to concatenate the Update program, the SSBL including the parameter for the loader, and parameter for the user application program.

Table 5.21 Options for updatefile command

Option	Required/Optional	Description
-i < file name >	Required	Specify the file name of the user application program to be updated as the < file name > string.
-o < file name >	Required	Specify the file name of the update file to be output as the < file name > string.
--write_addr < address >	Required	Specify the address in the RZ/T2M, RZ/T2L, RZ/N2L, RZ/T2H or RZ/N2H external flash memory to write the user application program to be updated to as the < address > string. Specify the address as eight digits of hexadecimal notation. Example: 00100000
--param <file name>	Required	Specify the parameter file name of the user application program to be updated as the < file name > string.
--cpu_core <0 - 3>	Optional	Specifies the CPU number or core number on which the user application program to be updated runs. If this option is omitted, it is assumed that "0" is specified.
-h	Optional	Specify this option to display help on using this tool.

Table 5.22 Options for setupfile command

Option	Required/Optional	Description
--param_loader < file name >	Required	Specify the file name of the loader parameter information to be set on the device as the < file name > string. This option is an option to specify the loader parameter file (loader program combined) that is generated by enabling the "--concat_loader" option of the parameter information generation tool (parameter_generator.py).
--param_cpu_core_0 < file name >	Required	Specify the file name of the user application parameters for Cortex®-R52 CPU0 or Cortex®-A55 Core 0 to be set on the device as the < file name > string.
--update_prog < address >	Required	Specify the file name of the update program to be set on the device as the < file name > string.
--mpu [rzt2h / rzn2h / rzt2m / rzt2l / rzn2l]	Required	Specifies the MPU to be set up. The MPUs that correspond to this option's argument are as follows: rzt2h: RZ/T2H rzn2h: RZ/N2H rzt2m: RZ/T2M rzt2l: RZ/T2L rzn2l: RZ/N2L
-o <file name>	Required	Specify the file name of the initial image file to be output as the < file name > string.
--ssbl <file name>	Optional	Specify the file name of the SSBL to be set on the device as the < file name > string. This option is only used if you specify the following file with the "--param_loader" option: - Loader parameter file generated using the parameter information generation tool (parameter_generator.py) without specifying the "--concat_lodaer" option
--param_cpu_core_1 <file name>	Optional	Specify the file name of the user application parameters for Cortex®-R52 CPU1 or Cortex®-A55 Core 1 to be set on the device as the < file name > string.
--param_cpu_core_2 <file name>	Optional	Specify the file name of the user application parameters for Cortex®-A55 Core 2 to be set on the device as the < file name > string.
--param_cpu_core_3 <file name>	Optional	Specify the file name of the user application parameters for Cortex®-A55 Core 3 to be set on the device as the < file name > string.
-h	Optional	Specify this option to display help on using this tool.

5.4.2 fwupdate.py

The tool fwupdate.py is used to send an update file to the device.

Using fwupdate.py, you can send an update file to the device via Ethernet by specifying the update file, the port number to be used for communication, and the IP address of the transfer destination device. Afterward, the tool receives the update result from the device and outputs it to the console.

The command format of fwupdate.py is as follows:

```
python fwupdate.py < options >
```

Some of the options of fwupdate.py are required and some may be omitted. Table 5.23 lists the required options and Table 5.24 lists the optional options.

Table 5.23 Required Options of fwupdate.py

Option	Description
--udp_port < port number >	Specify the port number to be used by fwupdate.py for UDP transmission and reception as the < port number > string.
--tcp_port < port number >	Specify the port number to be used by fwupdate.py for TCP transmission and reception as the < port number > string.
-i < file name >	Specify the update file to be sent to the device as the < file name > string.

Table 5.24 Optional Options of fwupdate.py

Option	Description
--ip_address < IP address >	Specify the IP address of the device with the user application program to be updated as the < IP address > string. When attempting to update the user application program, START_UPDATE is unicast to < IP address > if this option is specified, and it is broadcast if the option is not specified.
-h	Specify this option to display help on using this tool.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jul 8, 2022	-	First edition issued
1.10	Oct 21, 2022	-	RZ/N2L is supported
1.20	Apr 28, 2023	-	RZ/T2L is supported
2.00	Apr 10, 2024	p3, p11-p35	Support for updating applications that use dual cores on RZ/T2M.
		p9-p27, p42-p53	Updated FSP for RZ/T2M and RZ/T2L sample projects to RZ/T2 FSP v2.0.0. Changed command specification from fwupdate_utility.py V2.00 and added a command to create an initial image.
2.10	Jun 14, 2024	-	Updated FSP for RZ/N2L sample projects to RZ/N2 FSP v2.0.0.
2.20	Nov 15, 2024	-	RZ/T2H Cortex®-R52 core is supported. Updated FSP for RZ/T2M and RZ/T2L sample projects to RZ/T2 FSP v2.2.0.
		p7, p8, p50, p51	Changed the memory layout of the external flash in the sample program for RZ/T2M and RZ/T2L.
2.30	Dec 13, 2024	-	RZ/N2H Cortex®-R52 core is supported. Updated FSP for RZ/N2L sample projects to RZ/N2 FSP v2.1.0.
		p11-p38, p41-p50	The procedures in Chapter 3, “Configuring the Firmware Update System” and Chapter 4, “Applying Firmware Updates” have been changed to explain it for each target device.
2.40	Apr 21, 2025	-	RZ/T2H Cortex®-A55 and RZ/N2H Cortex®-A55 cores are supported.
3.00	Sep 10, 2025	-	Updated FSP for RZ/T2M, RZ/T2L and RZ/T2H sample projects to RZ/T FSP v3.0.0.
		p10-p14, p16, p17, p22, p23, p50-p52, p56, p57 p71, p72, p74, p75	Changed the memory layout of the external flash in the sample program for RZ/T2M, RZ/T2L, and RZ/T2H.
		p14, p16, p23, p31, p32, p78	Change command options for fwupdate_utility.py.
3.10	Oct 10, 2025	-	Updated FSP for RZ/N2L and RZ/N2H sample projects to RZ/N FSP v3.0.0.

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/.